Lab 5 - Ingest and load data into the Data Warehouse

This lab teaches you how to ingest data into the data warehouse through T-SQL scripts and Synapse Analytics integration pipelines. You will learn how to load data into Synapse dedicated SQL pools with PolyBase and COPY using T-SQL. You will also learn how to use workload management along with a Copy activity in a Azure Synapse pipeline for petabyte-scale data ingestion.

After completing this lab, you will be able to:

- Perform petabyte-scale ingestion with Azure Synapse Pipelines
- Import data with PolyBase and COPY using T-SQL
- Use data loading best practices in Azure Synapse Analytics

Lab setup and pre-requisites

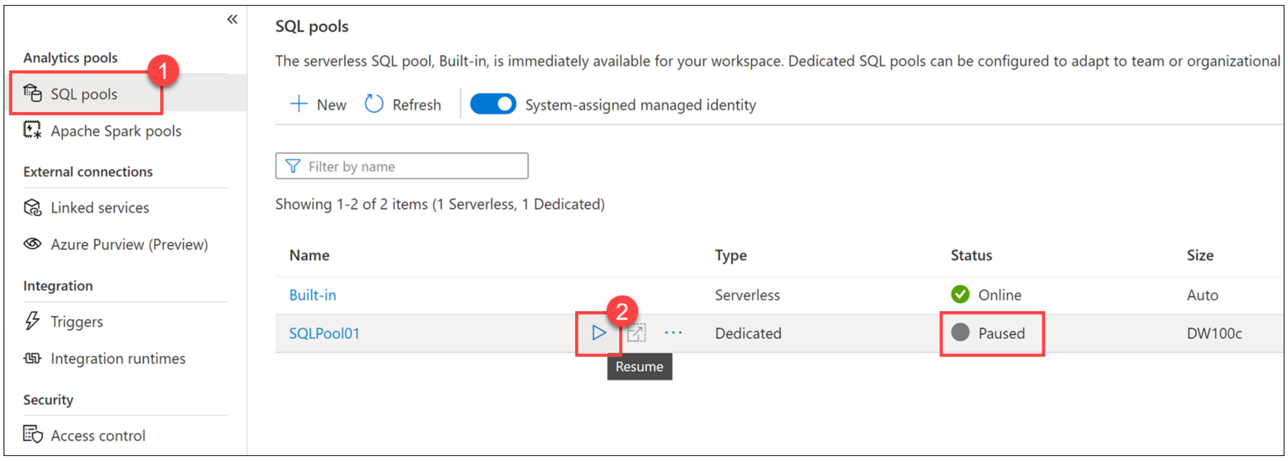
Before starting this lab, you must complete **Lab 4: *Explore, transform, and load data into the Data Warehouse using Apache Spark***.

This lab uses the dedicated SQL pool you created in the previous lab. You should have paused the SQL pool at the end of the previous lab, so resume it by following these instructions:

1. Open Synapse Studio (https://web.azuresynapse.net/).

2. Select the **Manage** hub.

3. Select **SQL pools** in the left-hand menu. If the **SQLPool01** dedicated SQL pool is paused, hover over its name and select ▷.


SQL pools

The serverless SQL pool, Built-in, is immediately available for your workspace. Dedicated SQL pools can be configured to adapt to team or organizational

Showing 1-2 of 2 items (1 Serverless, 1 Dedicated)

Name	Type	Status	Size
Built-in	Serverless	Online	Auto
SQLPool01	Dedicated	Paused	DW100c

4. When prompted, select **Resume**. It will take a minute or two to resume the pool.

> **Important:** Once started, a dedicated SQL pool consumes credits in your Azure subscription until it is paused. If you take a break from this lab, or decide not to complete it; follow the instructions at the end of the lab to pause your SQL pool!

Exercise 1 - Import data with PolyBase and COPY using T-SQL

There are different options for loading large amounts and varying types of data into Azure Synapse Analytics, such as through T-SQL commands using a Synapse SQL Pool, and with Azure Synapse pipelines. In our scenario, Wide World Importers stores most of their raw data in a data lake and in different formats. Among the data loading options available to them, WWI's data engineers are most comfortable using T-SQL.

However, even with their familiarity with SQL, there are some things to consider when loading large or disparate file types and formats. Since the files are stored in ADLS Gen2, WWI can use either PolyBase external tables or the new COPY statement. Both options enable fast and scalable data load operations, but there are some differences between the two:

PolyBase	COPY
Needs CONTROL permission	Relaxed permission
Has row width limits	No row width limit
No delimiters within text	Supports delimiters in text
Fixed line delimiter	Supports custom column and row delimiters
Complex to set up in code	Reduces amount of code

WWI has heard that PolyBase is generally faster than COPY, especially when working with large data sets.

In this exercise, you will help WWI compare ease of setup, flexibility, and speed between these loading strategies.

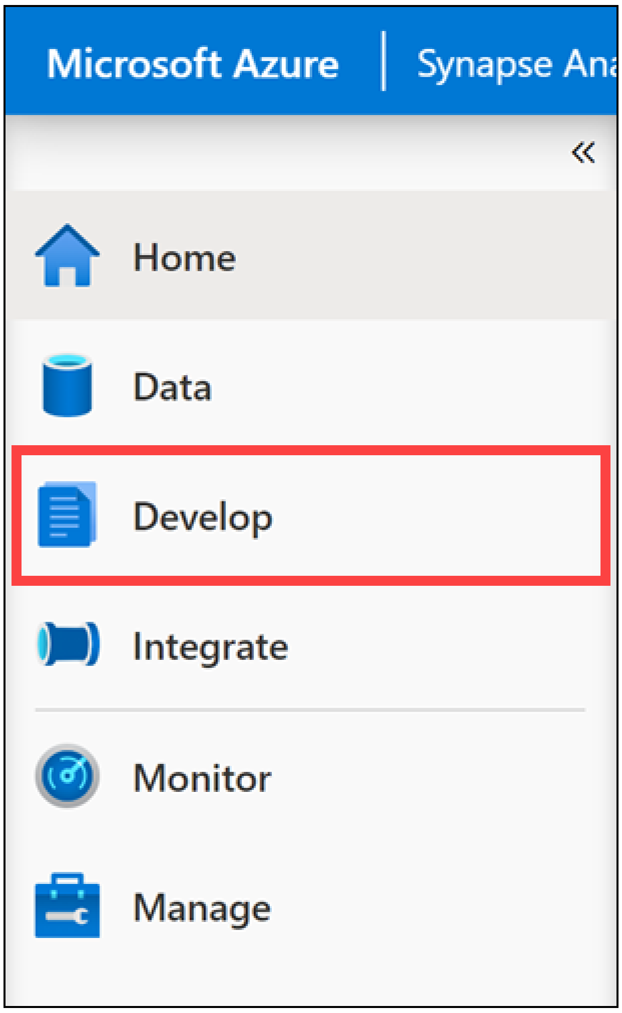
Task 1: Create staging tables

The **Sale** table has a columnstore index to optimize for read-heavy workloads. It is also used heavily for reporting and ad-hoc queries. To achieve the fastest loading speed and minimize the impact of heavy data inserts on the **Sale** table, WWI has decided to create a staging table for loads.

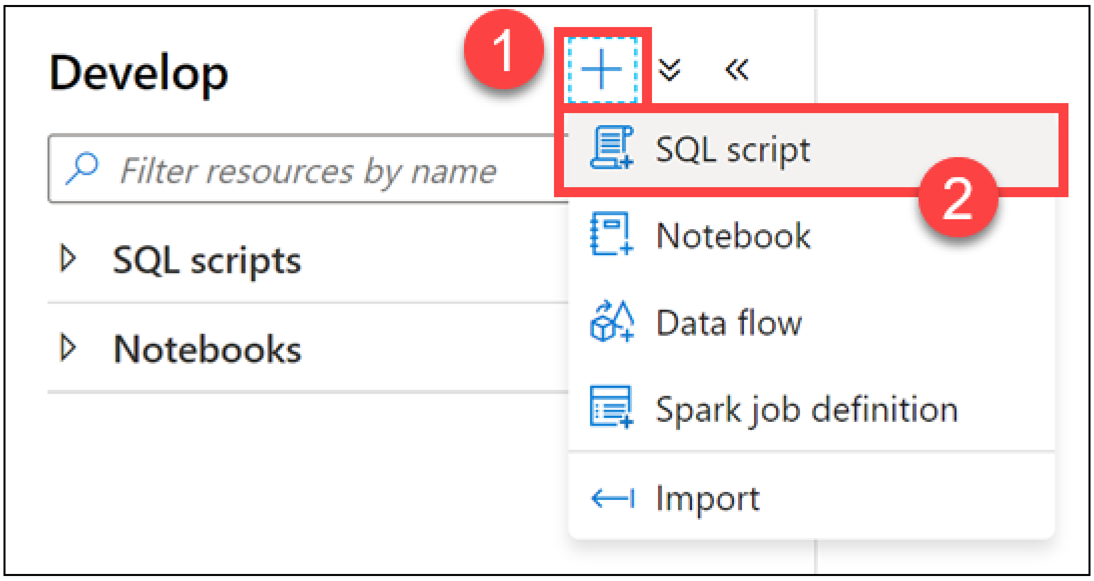
In this task, you will create a new staging table named **SaleHeap** in a new schema named **wwi_staging**. You will define it as a heap and use round-robin distribution. When WWI finalizes their data loading pipeline, they will load the data into **SaleHeap**, then insert from the heap table into **Sale**. Although this is a two-step process, the second step of inserting the rows to the production table does not incur data movement across the distributions.

You will also create a new **Sale** clustered columnstore table within the **wwi_staging** schema to compare data load speeds.

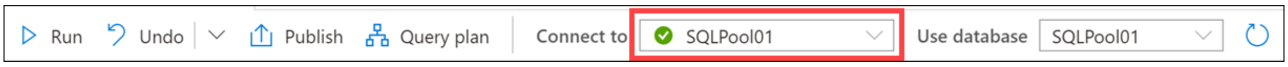
1. In Synapse Analytics Studio, navigate to the **Develop** hub.



2. In the **+** menu, select **SQL script**.



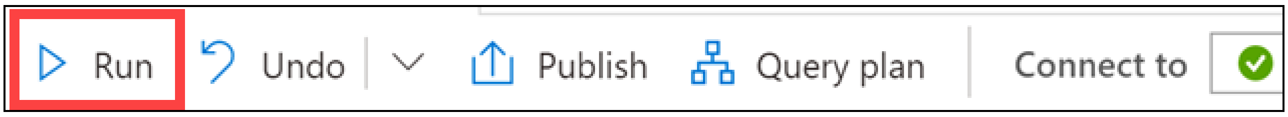
3. In the toolbar menu, connect to the **SQLPool01** database.



4. In the query window, enter the following code to verify that the **wwi_staging** schema exists.

```
SELECT * FROM sys.schemas WHERE name = 'wwi_staging'
```

5. Select **Run** from the toolbar menu to run the script.



The results should include a single row for the **wwi_staging** schema, which was created when setting up the previous lab.

6. In the query window, replace the script with the following to create the heap table:

```sql
CREATE TABLE [wwi_staging].[SaleHeap]
(
    [TransactionId] [uniqueidentifier]  NOT NULL,
    [CustomerId] [int]  NOT NULL,
    [ProductId] [smallint]  NOT NULL,
    [Quantity] [smallint]  NOT NULL,
    [Price] [decimal](9,2)  NOT NULL,
    [TotalAmount] [decimal](9,2)  NOT NULL,
    [TransactionDate] [int]  NOT NULL,
    [ProfitAmount] [decimal](9,2)  NOT NULL,
    [Hour] [tinyint]  NOT NULL,
    [Minute] [tinyint]  NOT NULL,
    [StoreId] [smallint]  NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    HEAP
)
```

7. Run the SQL script to create the table.

8. In the query window, replace the script with the following to create the **Sale** table in the **wwi_staging** schema for load comparisons:

```sql
CREATE TABLE [wwi_staging].[Sale]
(
    [TransactionId] [uniqueidentifier]  NOT NULL,
    [CustomerId] [int]  NOT NULL,
    [ProductId] [smallint]  NOT NULL,
    [Quantity] [smallint]  NOT NULL,
    [Price] [decimal](9,2)  NOT NULL,
    [TotalAmount] [decimal](9,2)  NOT NULL,
    [TransactionDate] [int]  NOT NULL,
    [ProfitAmount] [decimal](9,2)  NOT NULL,
    [Hour] [tinyint]  NOT NULL,
    [Minute] [tinyint]  NOT NULL,
    [StoreId] [smallint]  NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [CustomerId] ),
    CLUSTERED COLUMNSTORE INDEX,
    PARTITION
    (
        [TransactionDate] RANGE RIGHT FOR VALUES (20100101, 20100201,
20100301, 20100401, 20100501, 20100601, 20100701, 20100801, 20100901,
20101001, 20101101, 20101201, 20110101, 20110201, 20110301, 20110401,
20110501, 20110601, 20110701, 20110801, 20110901, 20111001, 20111101,
20111201, 20120101, 20120201, 20120301, 20120401, 20120501, 20120601,
20120701, 20120801, 20120901, 20121001, 20121101, 20121201, 20130101,
20130201, 20130301, 20130401, 20130501, 20130601, 20130701, 20130801,
20130901, 20131001, 20131101, 20131201, 20140101, 20140201, 20140301,
20140401, 20140501, 20140601, 20140701, 20140801, 20140901, 20141001,
20141101, 20141201, 20150101, 20150201, 20150301, 20150401, 20150501,
20150601, 20150701, 20150801, 20150901, 20151001, 20151101, 20151201,
20160101, 20160201, 20160301, 20160401, 20160501, 20160601, 20160701,
20160801, 20160901, 20161001, 20161101, 20161201, 20170101, 20170201,
20170301, 20170401, 20170501, 20170601, 20170701, 20170801, 20170901,
20171001, 20171101, 20171201, 20180101, 20180201, 20180301, 20180401,
20180501, 20180601, 20180701, 20180801, 20180901, 20181001, 20181101,
20181201, 20190101, 20190201, 20190301, 20190401, 20190501, 20190601,
20190701, 20190801, 20190901, 20191001, 20191101, 20191201)
    )
)
```

9. Run the script to ceate the table.

Task 2: Configure and run PolyBase load operation

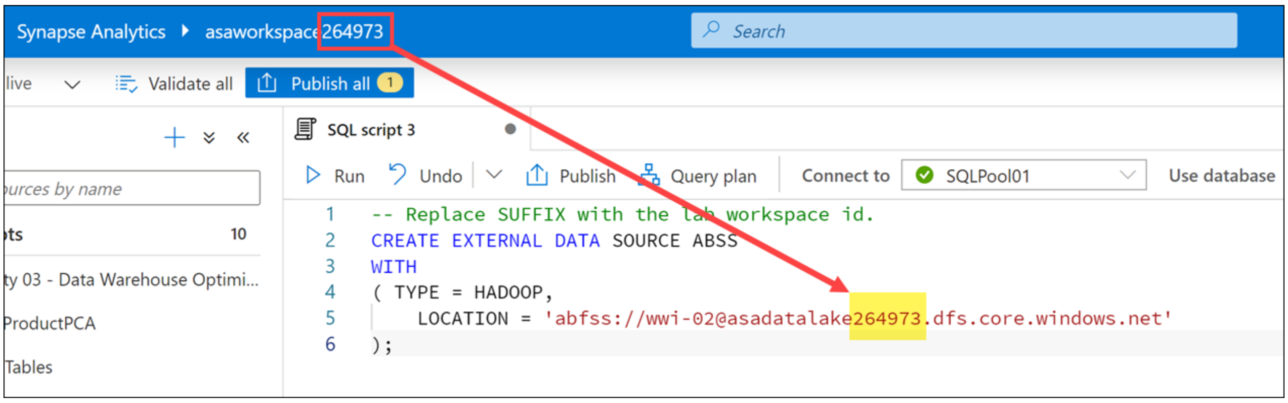
PolyBase requires the following elements:

- An external data source that points to the **abfss** path in ADLS Gen2 where the Parquet files are located
- An external file format for Parquet files
- An external table that defines the schema for the files, as well as the location, data source, and file format

1. In the query window, replace the script with the following code to create the external data source. Be sure to replace ***SUFFIX*** with the unique suffix for your Azure resources in this lab:

```
-- Replace SUFFIX with the lab workspace id.
CREATE EXTERNAL DATA SOURCE ABSS
WITH
( TYPE = HADOOP,
    LOCATION = 'abfss://wwi-02@asadatalakeSUFFIX.dfs.core.windows.net'
);
```

You can find your suffix at the end of the Synapse Analytics workspace name:



```
-- Replace SUFFIX with the lab workspace id.
CREATE EXTERNAL DATA SOURCE ABSS
WITH
( TYPE = HADOOP,
    LOCATION = 'abfss://wwi-02@asadatalake264973.dfs.core.windows.net'
);
```

2. Run the script to create the external data source.

3. In the query window, replace the script with the following code to create the external file format and external data table. Notice that we defined **TransactionId** as an **nvarchar(36)** field instead of **uniqueidentifier**. This is because external tables do not currently support **uniqueidentifier** columns:

```
CREATE EXTERNAL FILE FORMAT [ParquetFormat]
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
GO

CREATE SCHEMA [wwi_external];
GO

CREATE EXTERNAL TABLE [wwi_external].Sales
    (
        [TransactionId] [nvarchar](36)  NOT NULL,
        [CustomerId] [int]  NOT NULL,
        [ProductId] [smallint]  NOT NULL,
        [Quantity] [smallint]  NOT NULL,
        [Price] [decimal](9,2)  NOT NULL,
        [TotalAmount] [decimal](9,2)  NOT NULL,
        [TransactionDate] [int]  NOT NULL,
        [ProfitAmount] [decimal](9,2)  NOT NULL,
        [Hour] [tinyint]  NOT NULL,
        [Minute] [tinyint]  NOT NULL,
        [StoreId] [smallint]  NOT NULL
    )
WITH
    (
        LOCATION = '/sale-small/Year=2019',
        DATA_SOURCE = ABSS,
        FILE_FORMAT = [ParquetFormat]
    )
GO
```

> **Note:** The */sale-small/Year=2019/* folder's Parquet files contain **4,124,857 rows**.

4. Run the script.

5. In the query window, replace the script with the following code to load the data into the **wwi_staging.SalesHeap** table:

```
INSERT INTO [wwi_staging].[SaleHeap]
SELECT *
FROM [wwi_external].[Sales]
```

6. Run the script. It may take some time to complete.

7. In the query window, replace the script with the following to see how many rows were imported:

```
SELECT COUNT(1) FROM wwi_staging.SaleHeap(nolock)
```

8. Run the script. You should see a result of 4124857.

Task 3: Configure and run the COPY statement

Now let's see how to perform the same load operation with the COPY statement.

1. In the query window, replace the script with the following to truncate the heap table and load data using the COPY statement. As you did before, be sure to replace ***SUFFIX*** with your unique suffix:

```
TRUNCATE TABLE wwi_staging.SaleHeap;
GO

-- Replace SUFFIX with the unique suffix for your resources
COPY INTO wwi_staging.SaleHeap
FROM 'https://asadatalakeSUFFIX.dfs.core.windows.net/wwi-02/sale-
small/Year=2019'
WITH (
    FILE_TYPE = 'PARQUET',
    COMPRESSION = 'SNAPPY'
)
GO
```

2. Run the script. Notice how little scripting is required to perform a similar load operation.

3. In the query window, replace the script with the following to see how many rows were imported:

```
SELECT COUNT(1) FROM wwi_staging.SaleHeap(nolock)
```

4. Run the script. Once again, 4124857 rows should have been imported. Note that both load operations copied the same amount of data in roughly the same amount of time.

Task 4: Use COPY to load text file with non-standard row delimiters

One of the advantages COPY has over PolyBase is that it supports custom column and row delimiters.

WWI has a nightly process that ingests regional sales data from a partner analytics system and saves the files in the data lake. The text files use non-standard column and row delimiters where columns are delimited by a period and rows by a comma:

```
20200421.114892.130282.159488.172105.196533,20200420.109934.108377.122039.101946.1
00712,20200419.253714.357583.452690.553447.653921
```

The data has the following fields: **Date**, **NorthAmerica**, **SouthAmerica**, **Europe**, **Africa**, and **Asia**. They must process this data and store it in Synapse Analytics.

1. In the query window, replace the script with the following to create the **DailySalesCounts** table and load data using the COPY statement. As before, be sure to replace ***SUFFIX*** with your unique suffix:

   ```
   CREATE TABLE [wwi_staging].DailySalesCounts
       (
           [Date] [int]  NOT NULL,
           [NorthAmerica] [int]  NOT NULL,
           [SouthAmerica] [int]  NOT NULL,
           [Europe] [int]  NOT NULL,
           [Africa] [int]  NOT NULL,
           [Asia] [int]  NOT NULL
       )
   GO

   -- Replace SUFFIX with the unique suffix for your resources
   COPY INTO wwi_staging.DailySalesCounts
   FROM 'https://asadatalakeSUFFIX.dfs.core.windows.net/wwi-02/campaign-
   analytics/dailycounts.txt'
   WITH (
       FILE_TYPE = 'CSV',
       FIELDTERMINATOR='.',
       ROWTERMINATOR=','
   )
   GO
   ```

   Notice the FIELDTERMINATOR` and ROWTERMINATOR properties that enable the code to correctly parse the file.

2. Run the script.

3. In the query window, replace the script with the following to view the imported data:

   ```
   SELECT * FROM [wwi_staging].DailySalesCounts
   ORDER BY [Date] DESC
   ```

4. Run the script and view the results.

5. Try viewing the results in a Chart and set the **Category column** to **Date**:

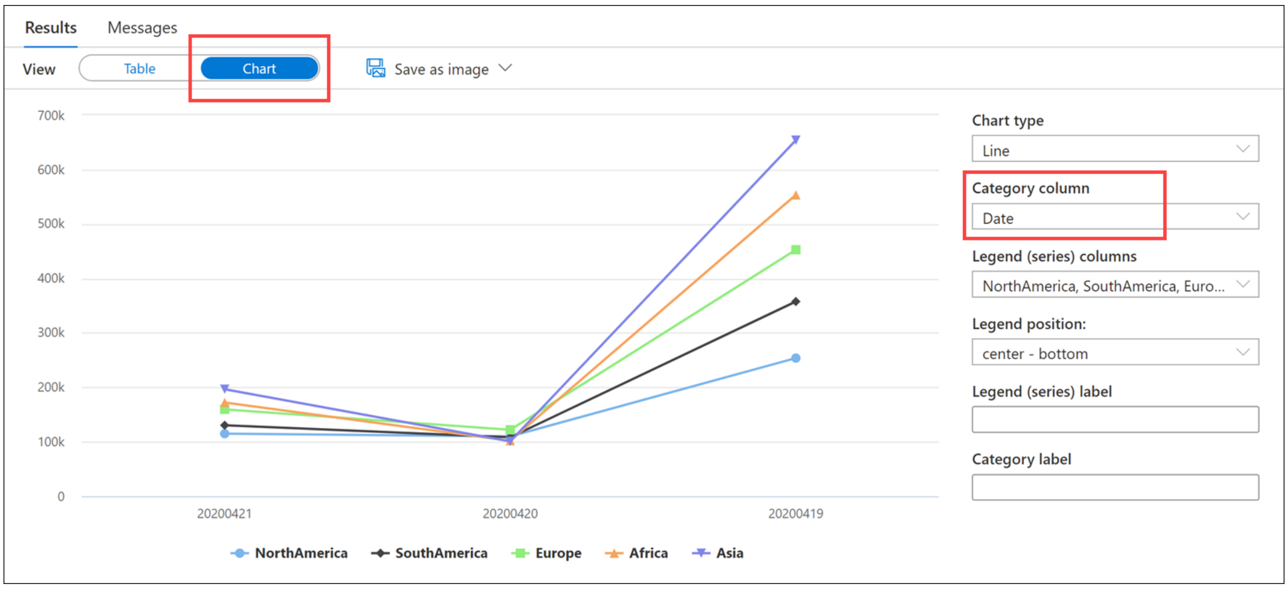


Chart type: Line
Category column: Date
Legend (series) columns: NorthAmerica, SouthAmerica, Euro...
Legend position: center - bottom
Legend (series) label
Category label

Task 5: Use PolyBase to load text file with non-standard row delimiters

Let's try this same operation using PolyBase.

1. In the query window, replace the script with the following to create a new external file format, external table, and load data using PolyBase:

```
CREATE EXTERNAL FILE FORMAT csv_dailysales
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
        FIELD_TERMINATOR = '.',
        DATE_FORMAT = '',
        USE_TYPE_DEFAULT = False
    )
);
GO

CREATE EXTERNAL TABLE [wwi_external].DailySalesCounts
    (
        [Date] [int]  NOT NULL,
        [NorthAmerica] [int]  NOT NULL,
        [SouthAmerica] [int]  NOT NULL,
        [Europe] [int]  NOT NULL,
        [Africa] [int]  NOT NULL,
        [Asia] [int]  NOT NULL
    )
WITH
    (
        LOCATION = '/campaign-analytics/dailycounts.txt',
        DATA_SOURCE = ABSS,
        FILE_FORMAT = csv_dailysales
    )
GO
INSERT INTO [wwi_staging].[DailySalesCounts]
SELECT *
FROM [wwi_external].[DailySalesCounts]
```

2. Run the script. You should see an error similar to:

   Failed to execute query. Error: HdfsBridge::recordReaderFillBuffer - Unexpected error encountered filling record reader buffer: HadoopExecutionException: Too many columns in the line..

   Why is this? According to PolyBase documentation:

   > The row delimiter in delimited-text files must be supported by Hadoop's LineRecordReader. That is, it must be either **\r**, **\n**, or **\r\n**. These delimiters are not user-configurable.

   This is an example of where COPY's flexibility gives it an advantage over PolyBase.

3. Keep the script open for the next exercise.

Exercise 2 - Petabyte-scale ingestion with Azure Synapse Pipelines

Tailwind Traders needs to ingest large volumes of sales data into the data warehouse. They want a repeatable process that can efficiently load the data. When the data loads, they want to prioritize the data movement jobs so they take priority.

You have decided to create a proof of concept data pipeline to import a large Parquet file, following best practices to improve the load performance.

There is often a level of orchestration involved when moving data into a data warehouse, coordinating movement from one or more data sources and sometimes some level of transformation. The transformation step can occur during (extract-transform-load - ETL) or after (extract-load-transform - ELT) data movement. Any modern data platform must provide a seamless experience for all the typical data wrangling actions like extractions, parsing, joining, standardizing, augmenting, cleansing, consolidating, and filtering. Azure Synapse Analytics provides two significant categories of features - data flows and data orchestrations (implemented as pipelines).

> In this exercise, we will focus on the orchestration aspect. A later lab will focus more on the transformation (data flow) pipelines.

Task 1: Configure workload management classification

When loading a large amount of data, it is best to run only one load job at a time for fastest performance. If this isn't possible, run a minimal number of loads concurrently. If you expect a large loading job, consider scaling up your dedicated SQL pool before the load.

Be sure that you allocate enough memory to the pipeline session. To do this, increase the resource class of a user which has permissions to rebuild the index on this table to the recommended minimum.

To run loads with appropriate compute resources, create loading users designated for running loads. Assign each loading user to a specific resource class or workload group. To run a load, sign in as one of the loading users, and then run the load. The load runs with the user's resource class.

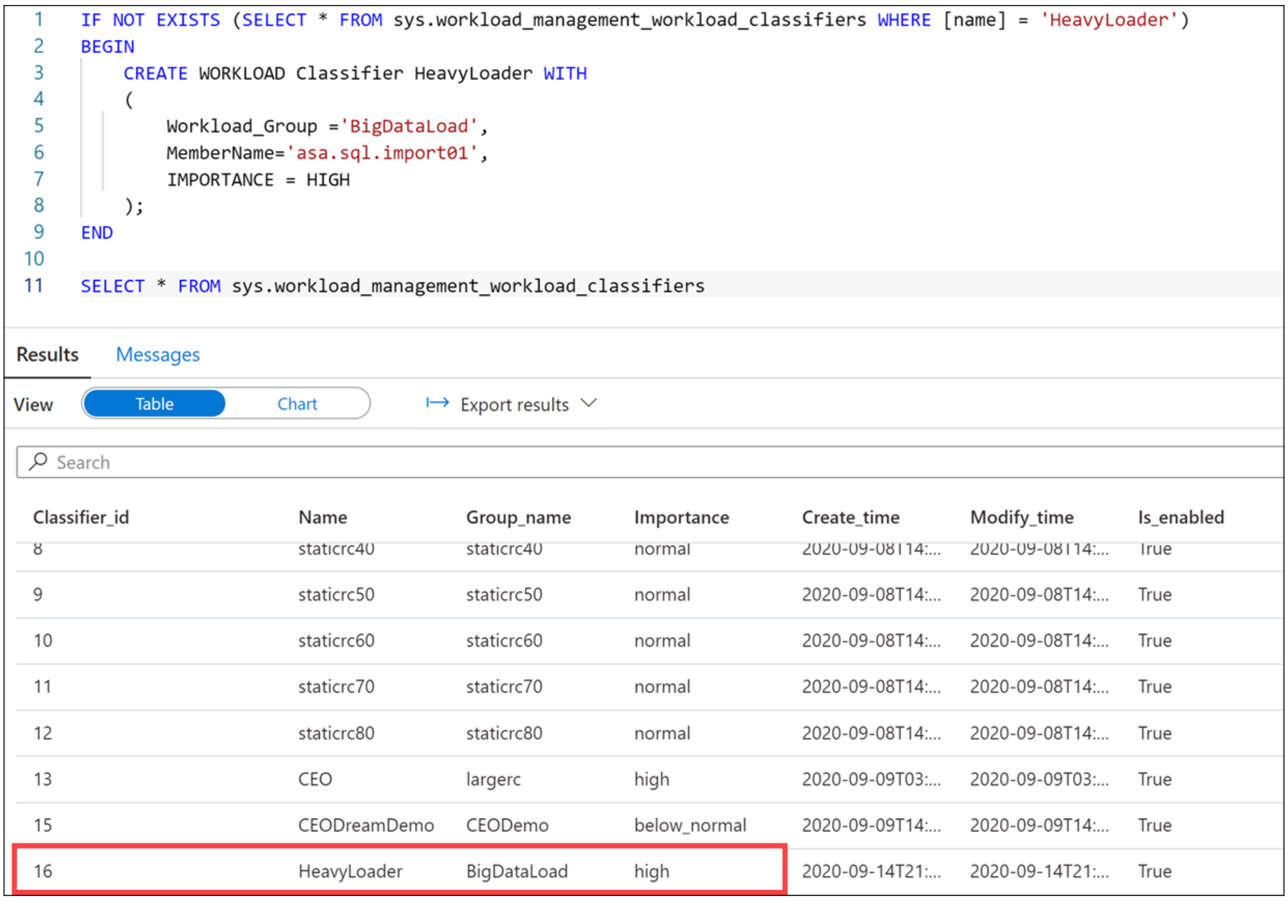
1. In the SQL script query window you worked with in the previous exercise, replace the script with the following to create:

   - A workload group, **BigDataLoad**, that uses workload isolation by reserving a minimum of 50% resources with a cap of 100%
   - A new workload classifier, **HeavyLoader** that assigns the **asa.sql.import01** user to the **BigDataLoad** workload group.

   At the end, we select from **sys.workload_management_workload_classifiers** to view all classifiers, including the one we just created:

   ```
   -- Drop objects if they exist
   IF EXISTS (SELECT * FROM sys.workload_management_workload_classifiers WHERE
   [name] = 'HeavyLoader')
   BEGIN
       DROP WORKLOAD CLASSIFIER HeavyLoader
   END;

   IF EXISTS (SELECT * FROM sys.workload_management_workload_groups WHERE name
   = 'BigDataLoad')
   BEGIN
       DROP WORKLOAD GROUP BigDataLoad
   END;

   --Create workload group
   CREATE WORKLOAD GROUP BigDataLoad WITH
     (
         MIN_PERCENTAGE_RESOURCE = 50, -- integer value
         REQUEST_MIN_RESOURCE_GRANT_PERCENT = 25, --  (guaranteed min 4
   concurrency)
         CAP_PERCENTAGE_RESOURCE = 100
     );

   -- Create workload classifier
   CREATE WORKLOAD Classifier HeavyLoader WITH
   (
       Workload_Group ='BigDataLoad',
       MemberName='asa.sql.import01',
       IMPORTANCE = HIGH
   );

   -- View classifiers
   SELECT * FROM sys.workload_management_workload_classifiers
   ```

2. Run the script, and if necessary, switch the results to **Table** view. You should see the new **HeavyLoader** classifier in the query results:



```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_classifiers WHERE [name] = 'HeavyLoader')
BEGIN
    CREATE WORKLOAD Classifier HeavyLoader WITH
    (
        Workload_Group ='BigDataLoad',
        MemberName='asa.sql.import01',
        IMPORTANCE = HIGH
    );
END

SELECT * FROM sys.workload_management_workload_classifiers
```

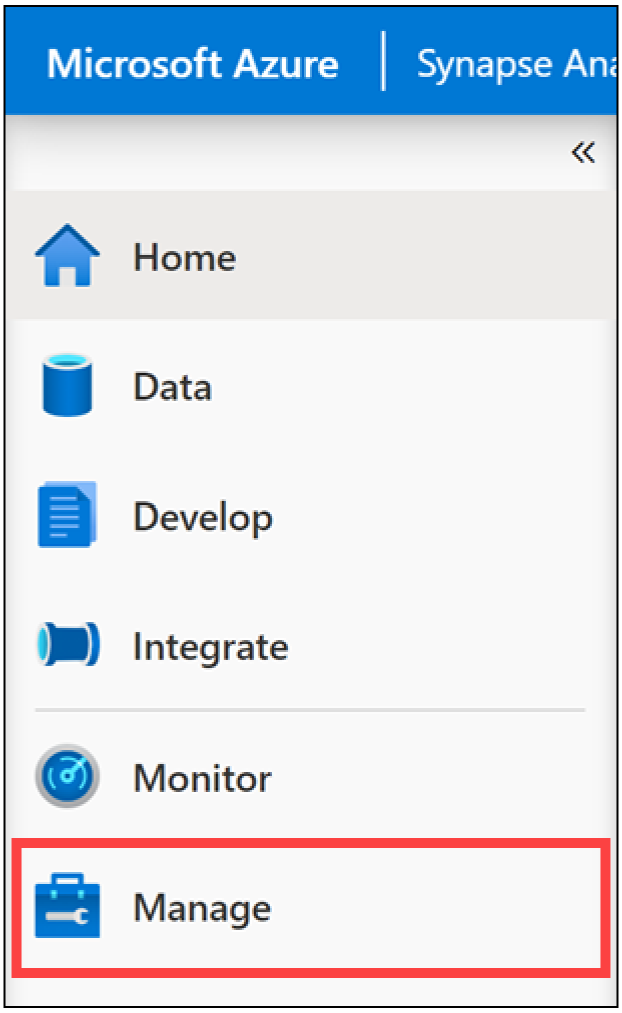
Results Messages

View Table Chart Export results

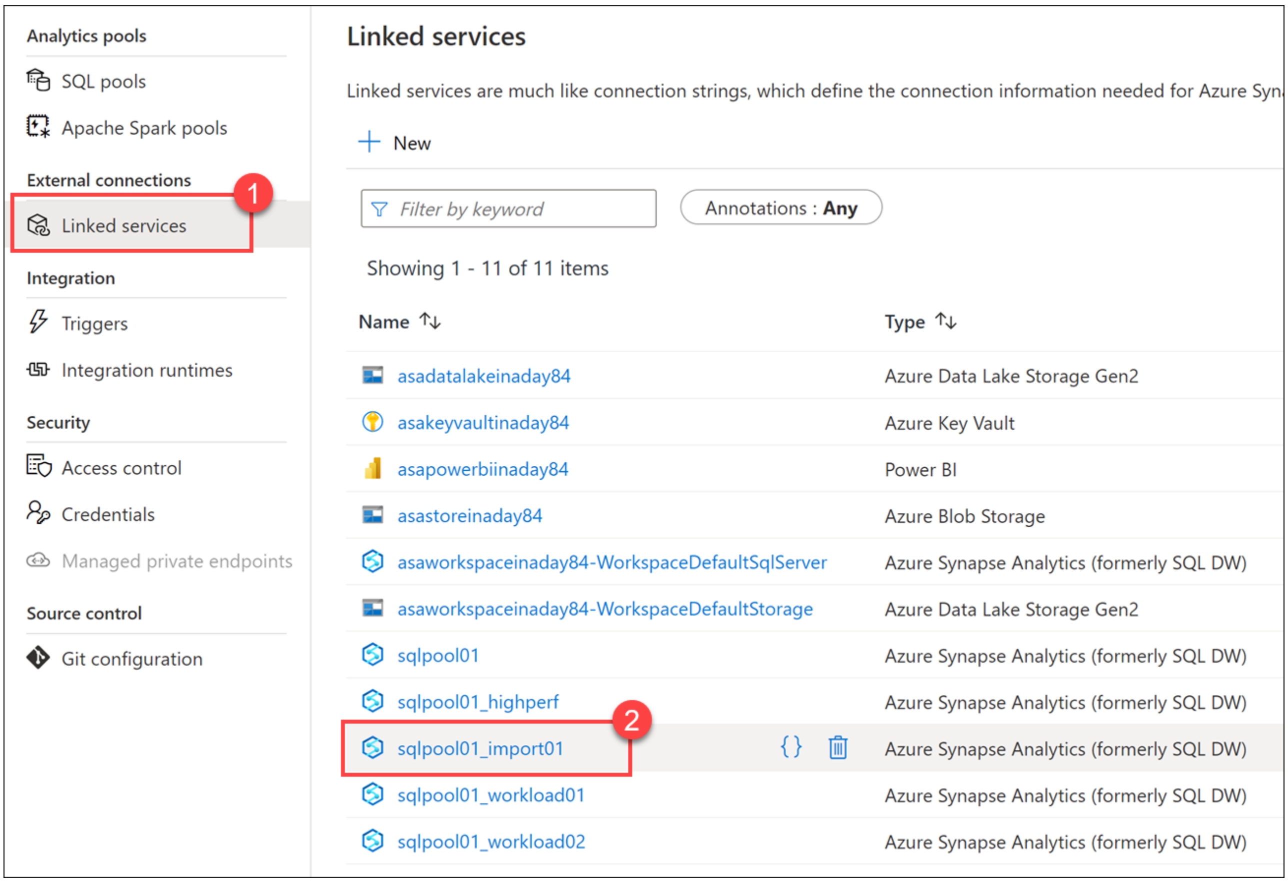
Search

Classifier_id	Name	Group_name	Importance	Create_time	Modify_time	Is_enabled
8	staticrc40	staticrc40	normal	2020-09-08T14:...	2020-09-08T14:...	True
9	staticrc50	staticrc50	normal	2020-09-08T14:...	2020-09-08T14:...	True
10	staticrc60	staticrc60	normal	2020-09-08T14:...	2020-09-08T14:...	True
11	staticrc70	staticrc70	normal	2020-09-08T14:...	2020-09-08T14:...	True
12	staticrc80	staticrc80	normal	2020-09-08T14:...	2020-09-08T14:...	True
13	CEO	largerc	high	2020-09-09T03:...	2020-09-09T03:...	True
15	CEODreamDemo	CEODemo	below_normal	2020-09-09T14:...	2020-09-09T14:...	True
16	HeavyLoader	BigDataLoad	high	2020-09-14T21:...	2020-09-14T21:...	True

3. Navigate to the **Manage** hub.



4. Select **Linked services** in the left-hand menu, then select the **sqlpool01_import01** linked service (if this is not listed, use the ↻ button at the top right to refresh the view).



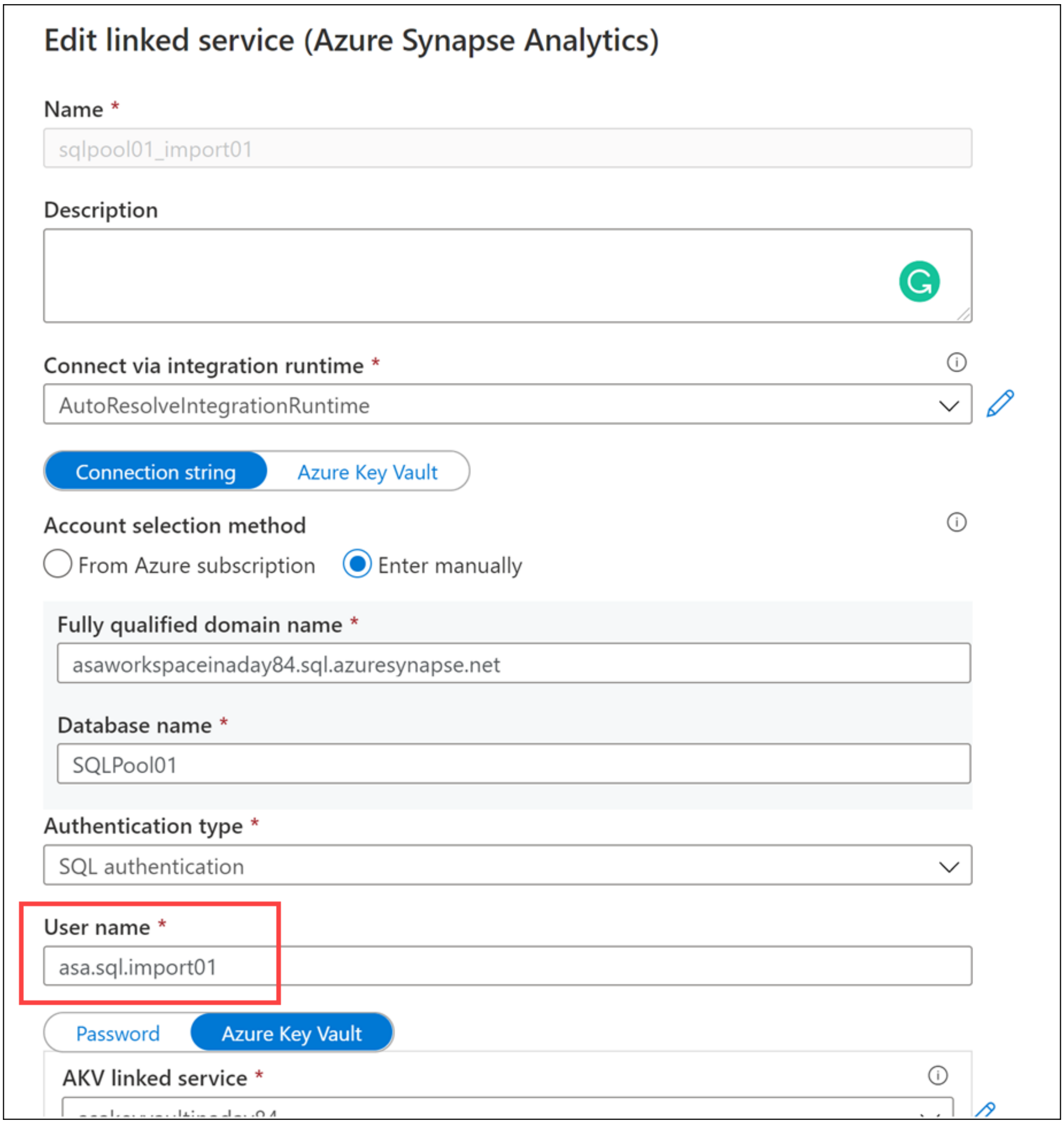
Linked services

Linked services are much like connection strings, which define the connection information needed for Azure Syn

Showing 1 - 11 of 11 items

Name	Type
asadatalakeinaday84	Azure Data Lake Storage Gen2
asakeyvaultinaday84	Azure Key Vault
asapowerbiinaday84	Power BI
asastoreinaday84	Azure Blob Storage
asaworkspaceinaday84-WorkspaceDefaultSqlServer	Azure Synapse Analytics (formerly SQL DW)
asaworkspaceinaday84-WorkspaceDefaultStorage	Azure Data Lake Storage Gen2
sqlpool01	Azure Synapse Analytics (formerly SQL DW)
sqlpool01_highperf	Azure Synapse Analytics (formerly SQL DW)
sqlpool01_import01	Azure Synapse Analytics (formerly SQL DW)
sqlpool01_workload01	Azure Synapse Analytics (formerly SQL DW)
sqlpool01_workload02	Azure Synapse Analytics (formerly SQL DW)

5. Notice that the user name for the dedicated SQL pool connection is the **asa.sql.import01** user you added to the **HeavyLoader** classifier. We will use this linked service in our new pipeline to reserve resources for the data load activity.



Edit linked service (Azure Synapse Analytics)

Name *

sqlpool01_import01

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

[✓ Connection string] Azure Key Vault

Account selection method

From Azure subscription [✓ Enter manually]

Fully qualified domain name *

asaworkspaceinaday84.sql.azuresynapse.net

Database name *

SQLPool01

Authentication type *

SQL authentication

User name *

asa.sql.import01

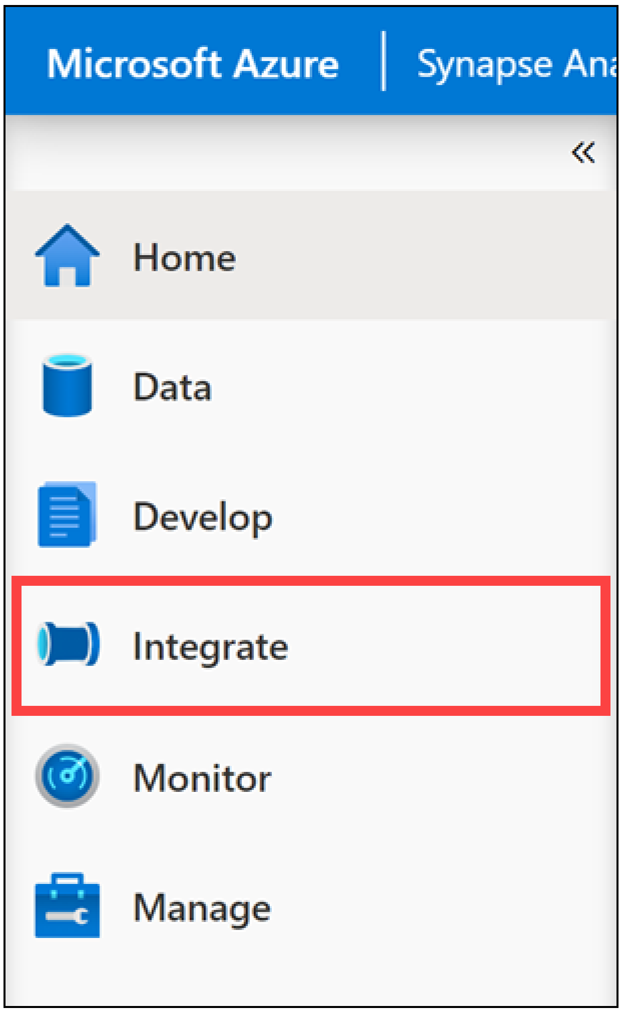
Password [✓ Azure Key Vault]

AKV linked service *

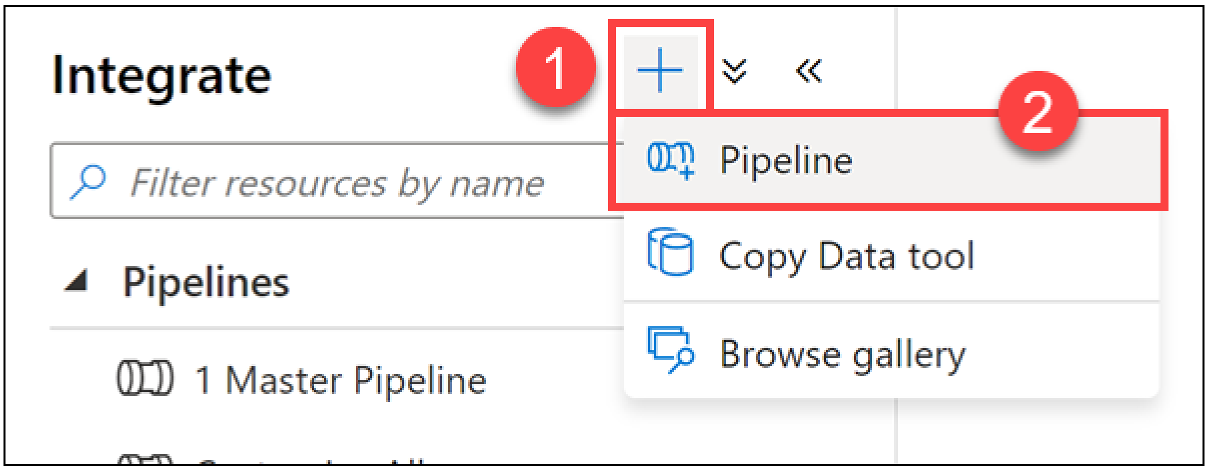
6. Select **Cancel** to close the dialog, and select **Discard changes** if prompted.

Task 2: Create pipeline with copy activity

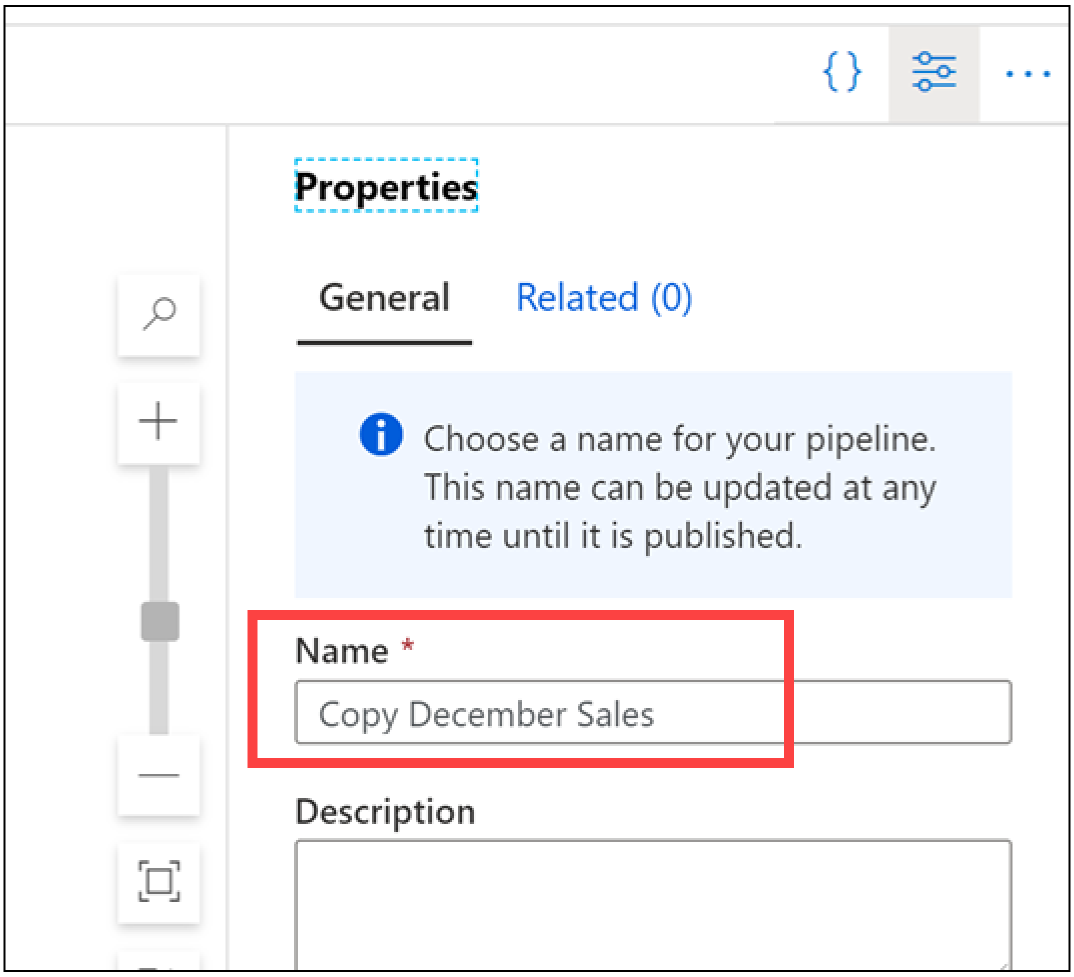
1. Navigate to the **Integrate** hub.



2. In the **+** menu, select **Pipeline** to create a new pipeline.

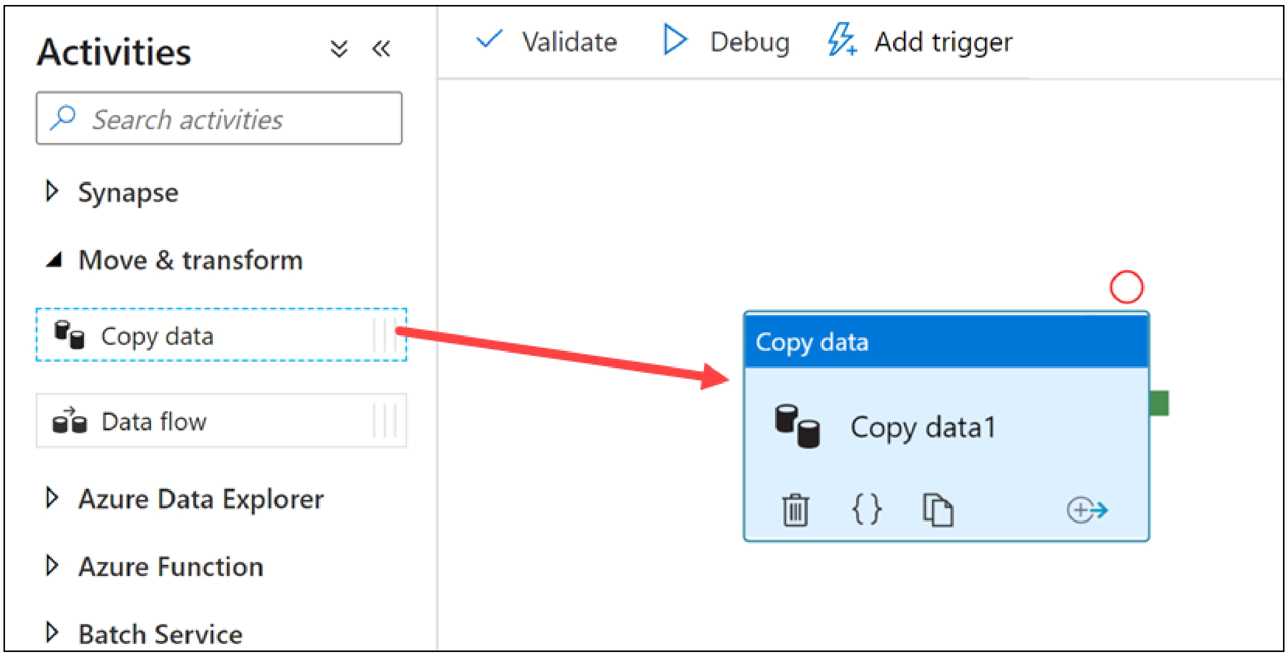


3. In the **Properties** pane for the new pipeline, set the **Name** of the pipeline to `Copy December Sales`.

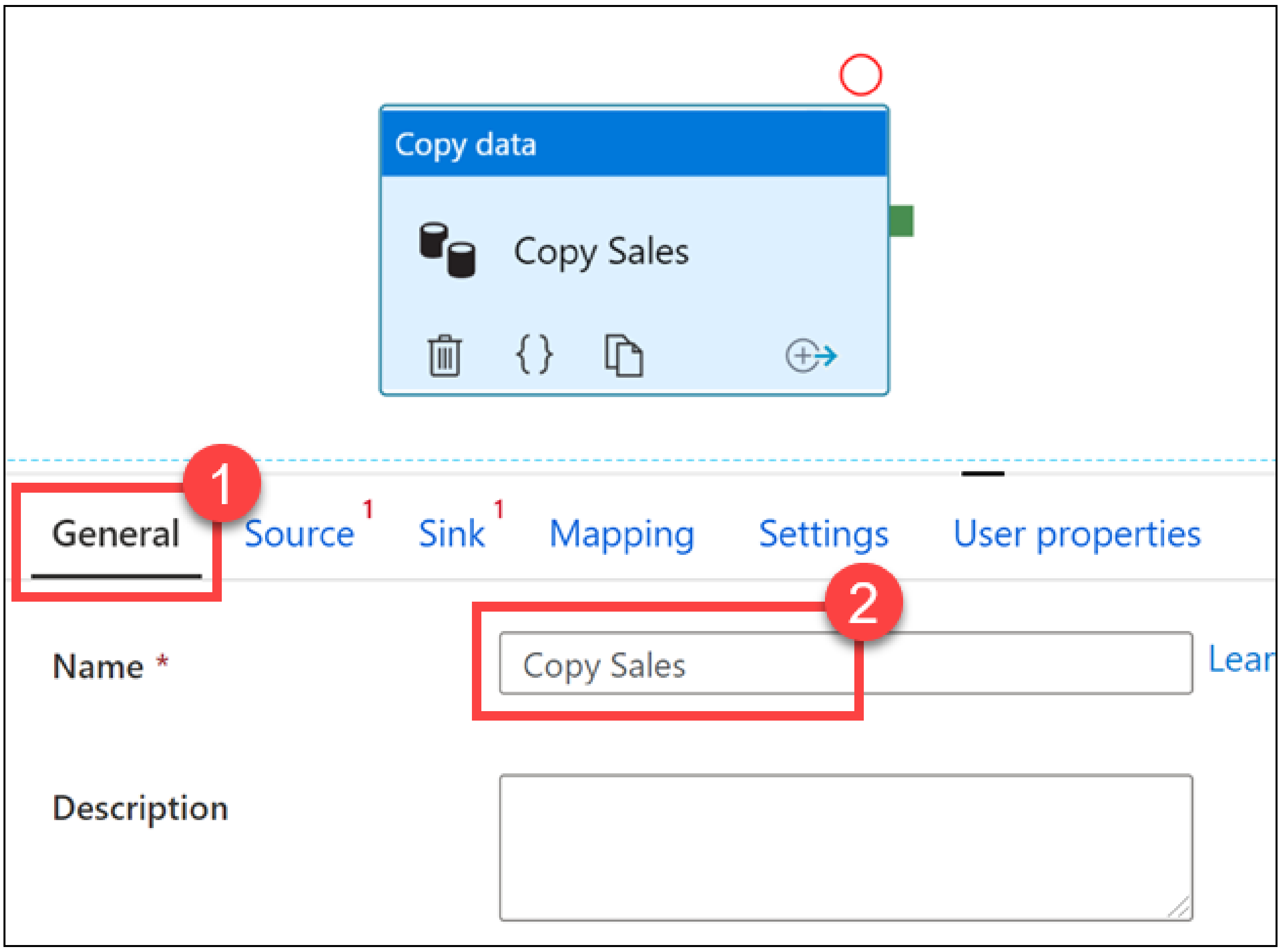

Properties
General
Related (0)
Choose a name for your pipeline. This name can be updated at any time until it is published.
Name *
Copy December Sales
Description

> **Tip**: After setting the name, hide the **Properties** pane.

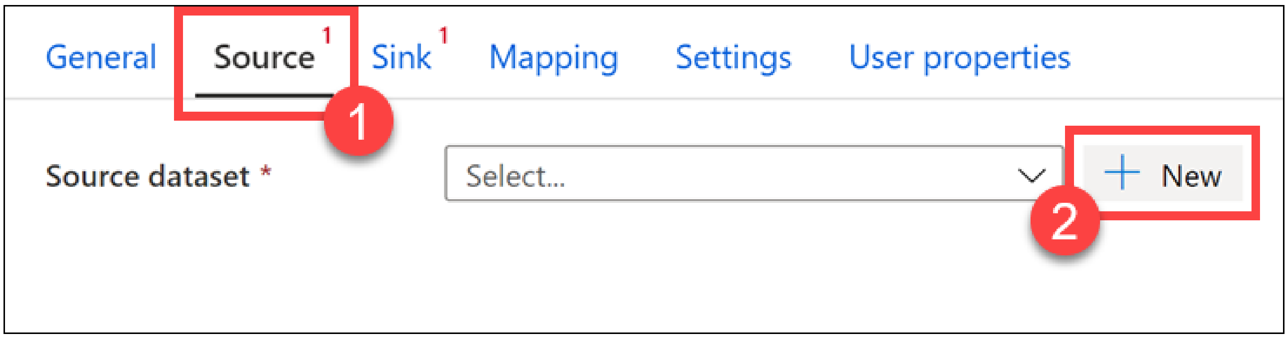
4. Expand **Move & transform** within the Activities list, then drag the **Copy data** activity onto the pipeline canvas.



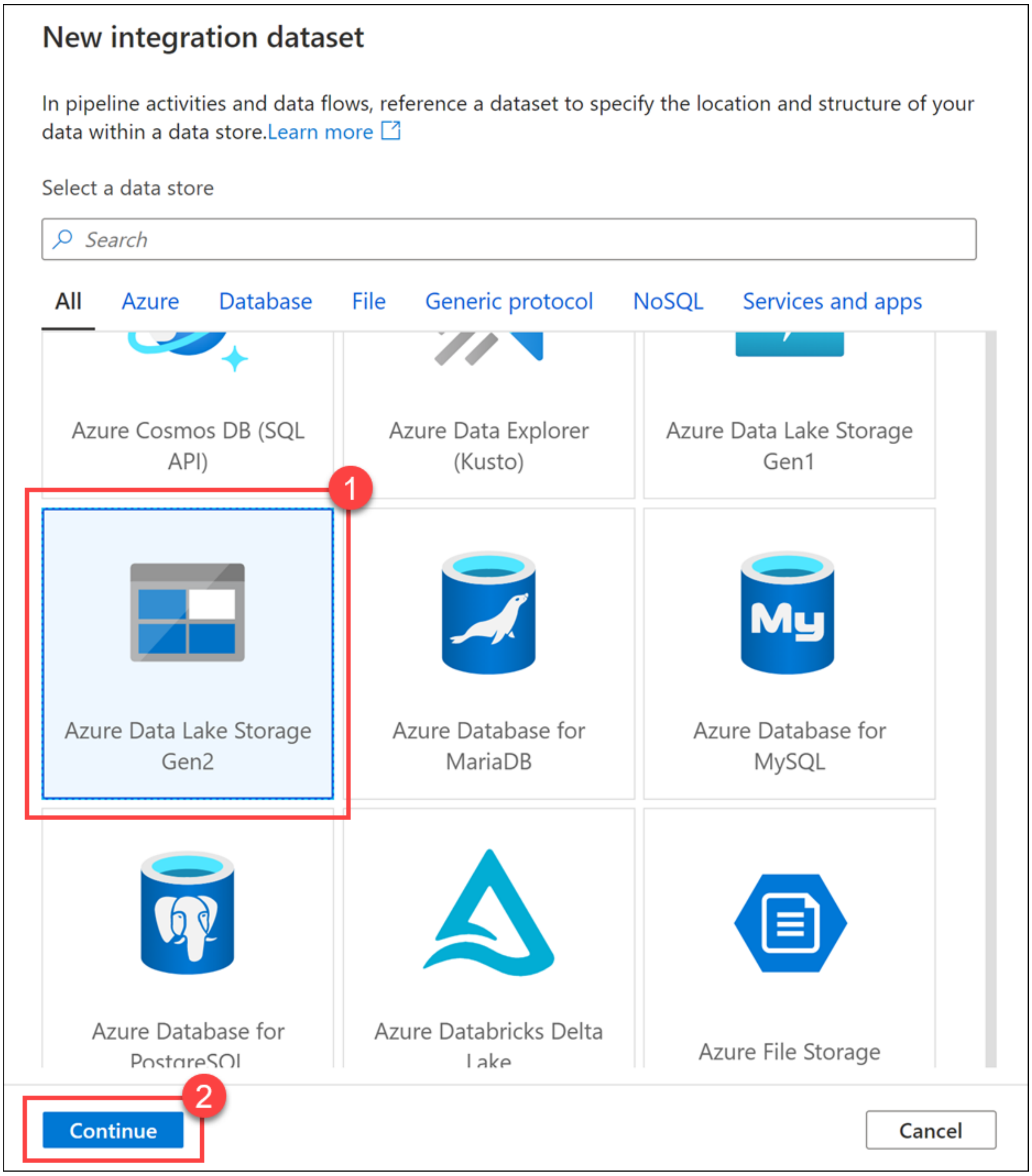
5. Select the **Copy data** activity on the canvas. Then, beneath the canvas, on the **General** tab, set the **Name** of the activity to `Copy Sales`.



6. Select the **Source** tab, then select **+ New** to create a new source dataset.



7. Select the **Azure Data Lake Storage Gen2** data store, then select **Continue**.



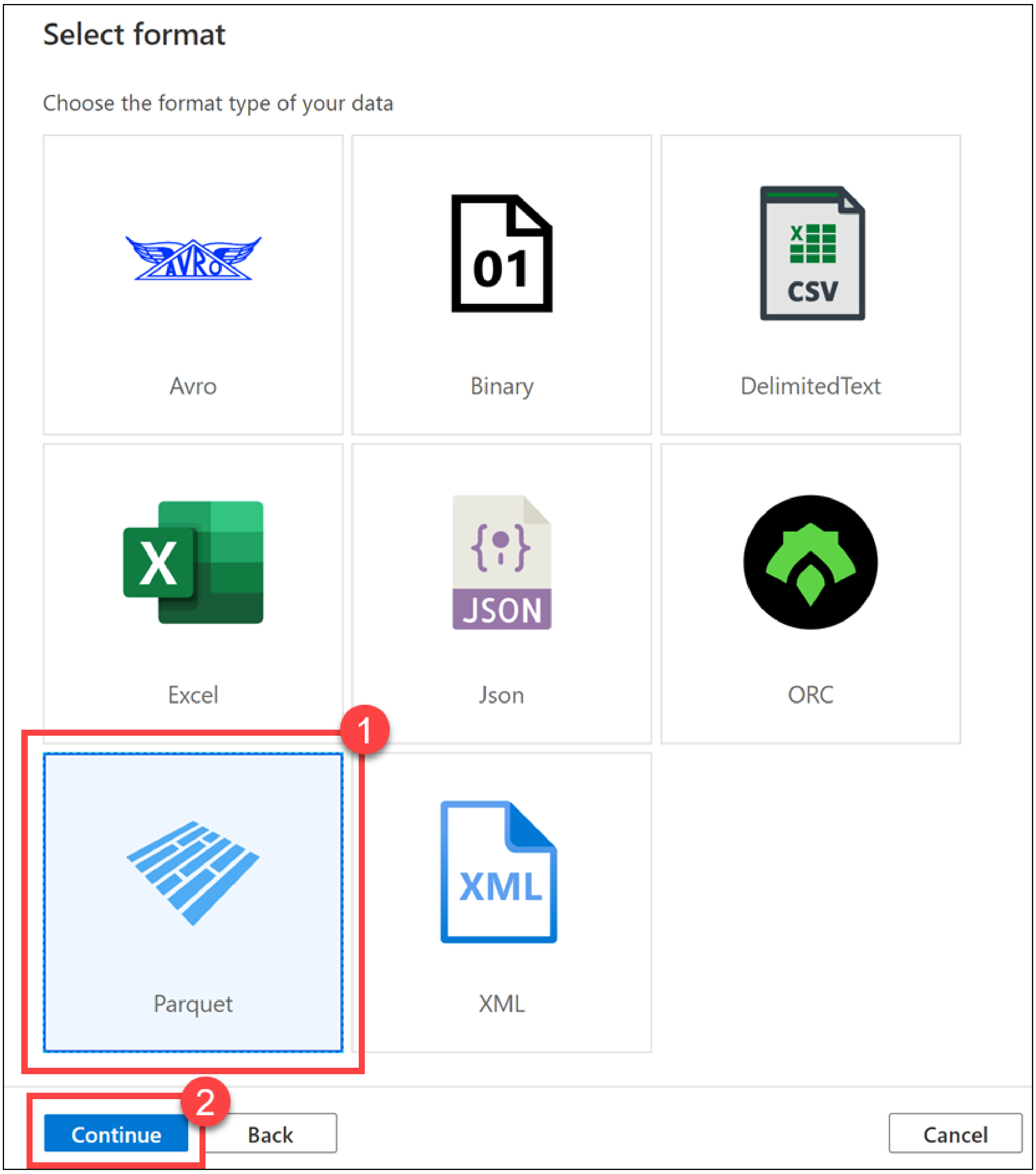
New integration dataset

In pipeline activities and data flows, reference a dataset to specify the location and structure of your data within a data store.Learn more

Select a data store

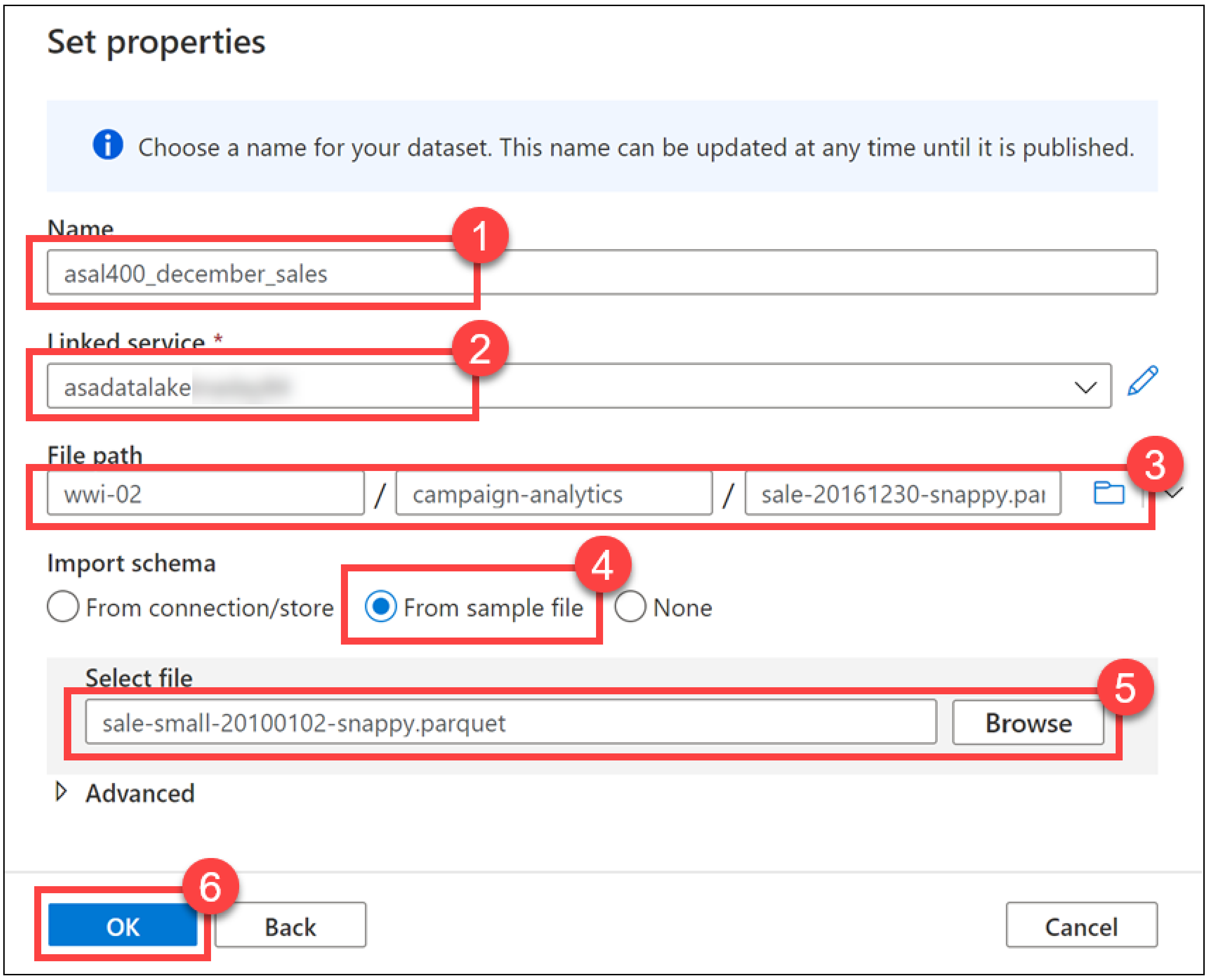
Search
All
Azure
Database
File
Generic protocol
NoSQL
Services and apps
Azure Cosmos DB (SQL API)
Azure Data Explorer (Kusto)
Azure Data Lake Storage Gen1
1
Azure Data Lake Storage Gen2
Azure Database for MariaDB
Azure Database for MySQL
Azure Database for PostgreSQL
Azure Databricks Delta Lake
Azure File Storage
2
Continue
Cancel

8. Choose the **Parquet** format, then select **Continue**.



9. In the **Set properties** pane:

   1. Set the name to `asal400_december_sales`.
   2. Select the **asadatalake*xxxxxxx*** linked service.
   3. Browse to the **wwi-02/campaign-analytics/sale-20161230-snappy.parquet** file
   4. Select **From sample file** for schema import.
   5. Browse to **C:\dp-203\data-engineering-ilt-deployment\Allfiles\samplefiles\sale-small-20100102-snappy.parquet** in the **Select file** field.
   6. Select **OK**.



Set properties

Choose a name for your dataset. This name can be updated at any time until it is published.

Name: asal400_december_sales

Linked service *: asadatalake[redacted]

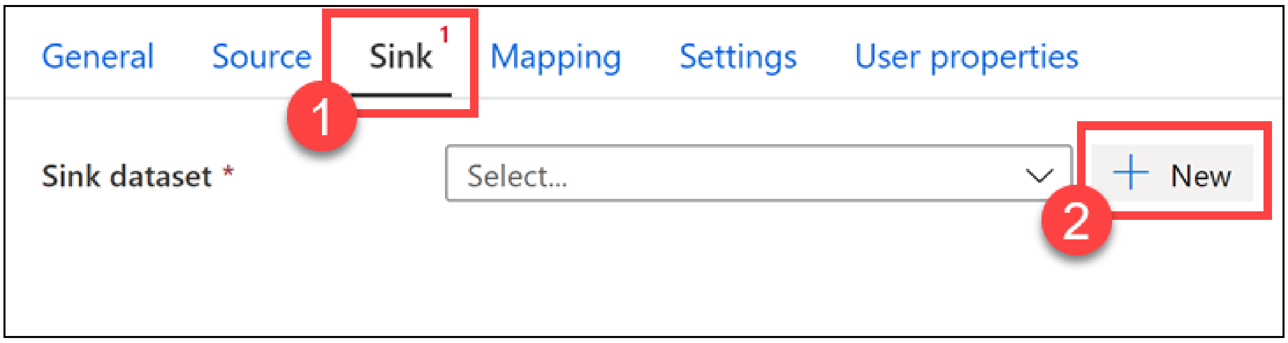
File path: wwi-02 / campaign-analytics / sale-20161230-snappy.pai

Import schema: From connection/store [✓ From sample file] None

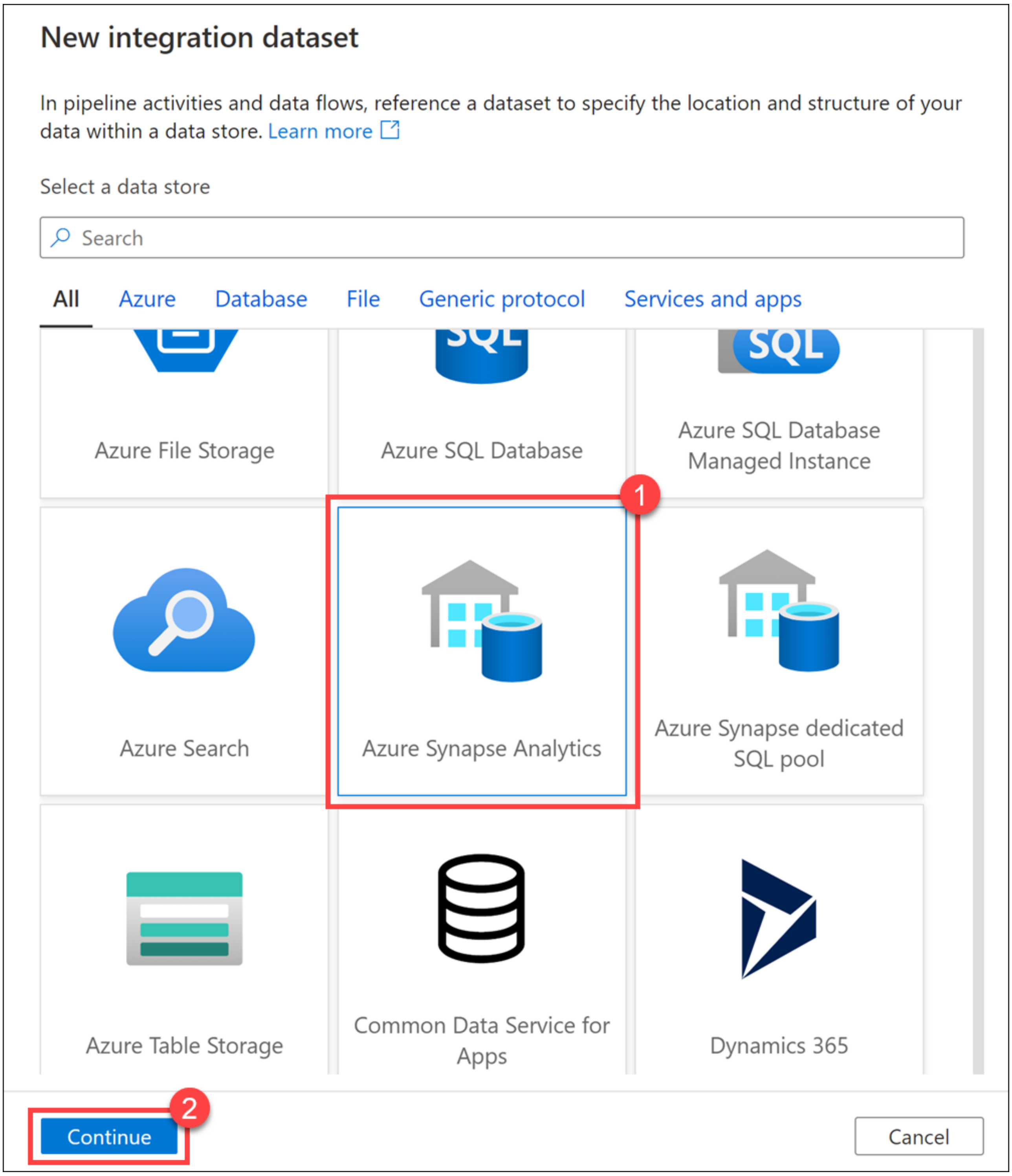
Select file: sale-small-20100102-snappy.parquet

We downloaded a sample Parquet file that has the exact same schema, but is much smaller. This is because the file we are copying is too large to automatically infer the schema in the copy activity source settings.

10. Select the **Sink** tab, then select **+ New** to create a new sink dataset.



11. Select the **Azure Synapse Analytics** data store, then select **Continue**.

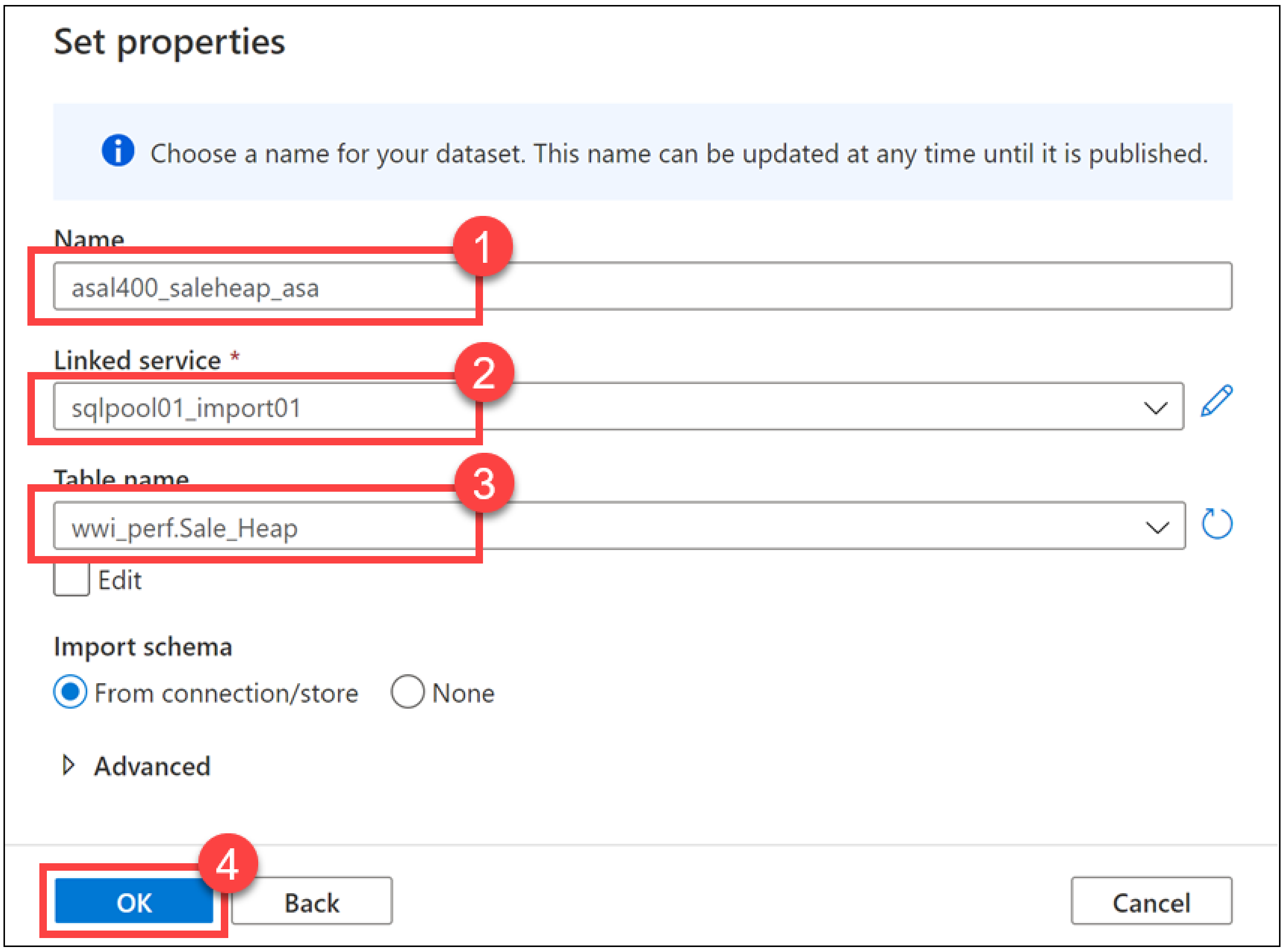

New integration dataset

In pipeline activities and data flows, reference a dataset to specify the location and structure of your data within a data store. Learn more

Select a data store

12. In the **Set properties** pane:

    1. Set the **Name** to `asal400_saleheap_asa`
    2. Select the **sqlpool01_import01** linked service.
    3. Select the **wwi_perf.Sale_Heap** table
    4. Select **OK**.


Set properties

Choose a name for your dataset. This name can be updated at any time until it is published.

Name: asal400_saleheap_asa

Linked service *: sqlpool01_import01

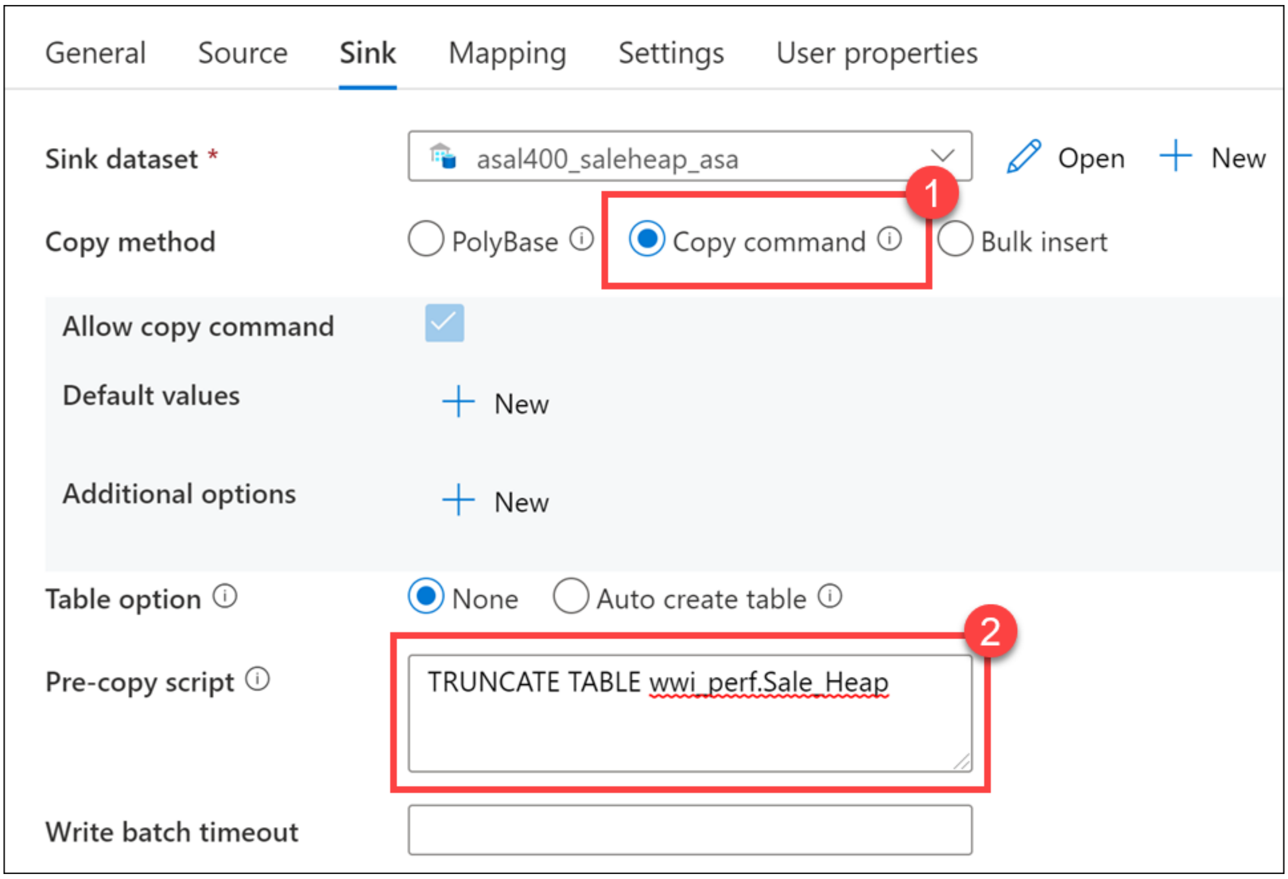
Table name: wwi_perf.Sale_Heap

Edit

Import schema: [✓ From connection/store] None

13. In the **Sink** tab, select the **Copy command** copy method and enter the following code in the pre-copy script to clear the table before import:

```
TRUNCATE TABLE wwi_perf.Sale_Heap
```



Sink dataset *: asal400_saleheap_asa

Copy method: PolyBase [✓ Copy command] Bulk insert

Allow copy command: ✓

Default values: New

Additional options: New

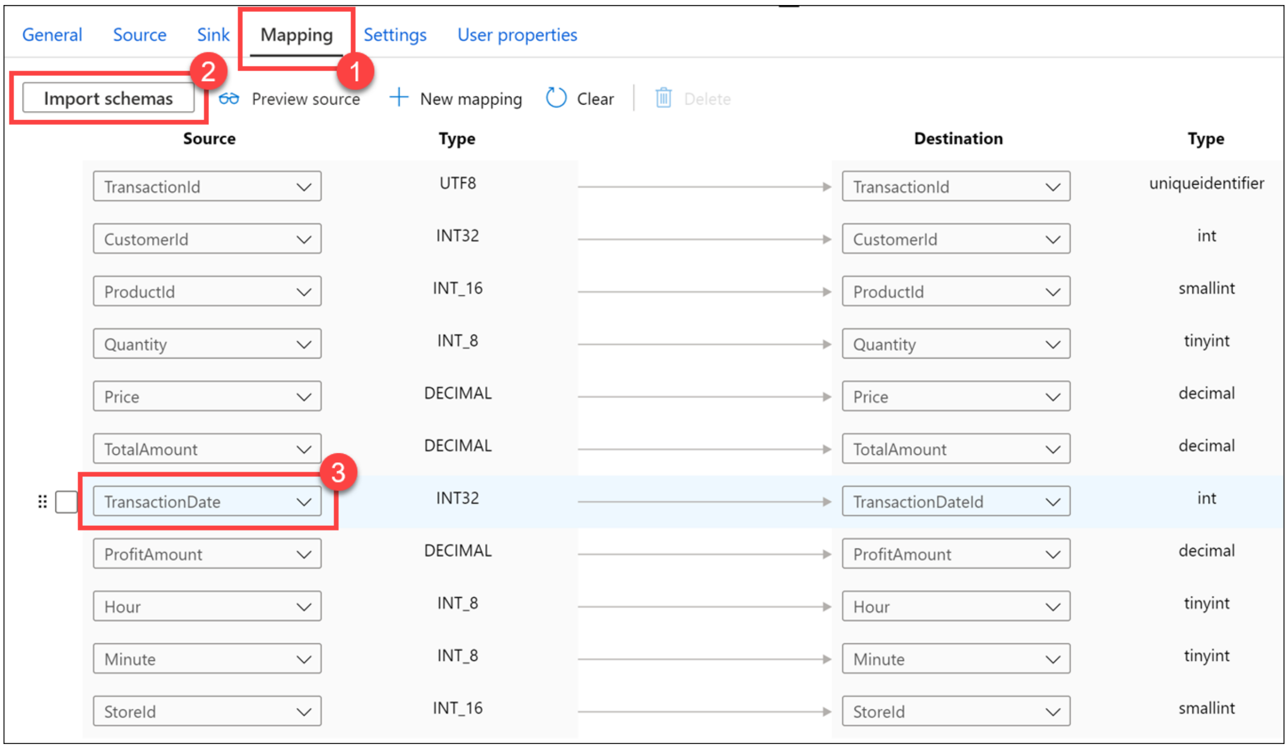
Table option: [✓ None] Auto create table

Pre-copy script: TRUNCATE TABLE wwi_perf.Sale_Heap

Write batch timeout:

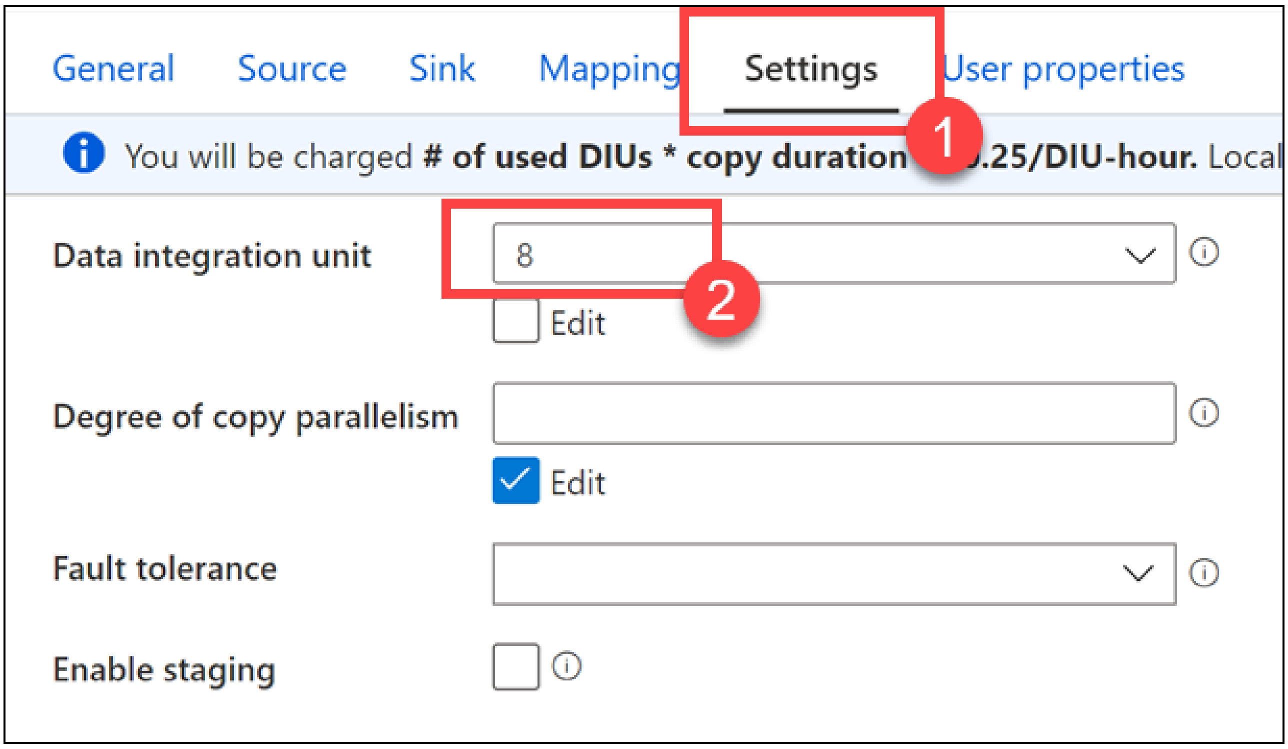
The fastest and most scalable way to load data is through PolyBase or the COPY statement, and the COPY statement provides the most flexibility for high-throughput data ingestion into the SQL pool.

14. Select the **Mapping** tab and select **Import schemas** to create mappings for each source and destination field. Select **TransactionDate** in the source column to map it to the **TransactionDateId** destination column.



Source	Type	Destination	Type
TransactionId	UTF8	TransactionId	uniqueidentifier
CustomerId	INT32	CustomerId	int
ProductId	INT_16	ProductId	smallint
Quantity	INT_8	Quantity	tinyint
Price	DECIMAL	Price	decimal
TotalAmount	DECIMAL	TotalAmount	decimal
TransactionDate	INT32	TransactionDateId	int
ProfitAmount	DECIMAL	ProfitAmount	decimal
Hour	INT_8	Hour	tinyint
Minute	INT_8	Minute	tinyint
StoreId	INT_16	StoreId	smallint

15. Select the **Settings** tab and set the **Data integration unit** to **8**. This is required due to the large size of the source Parquet file.



You will be charged **# of used DIUs * copy duration** **.25/DIU-hour.** Local

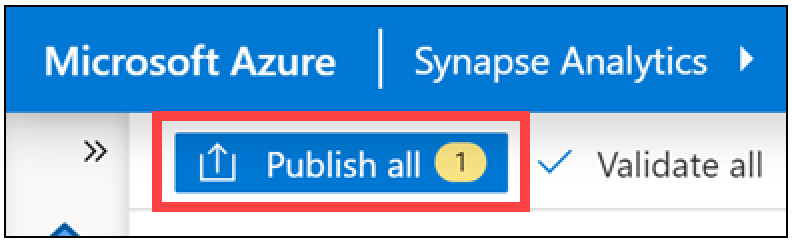
Data integration unit: 8
Edit

Degree of copy parallelism:
Edit ✓

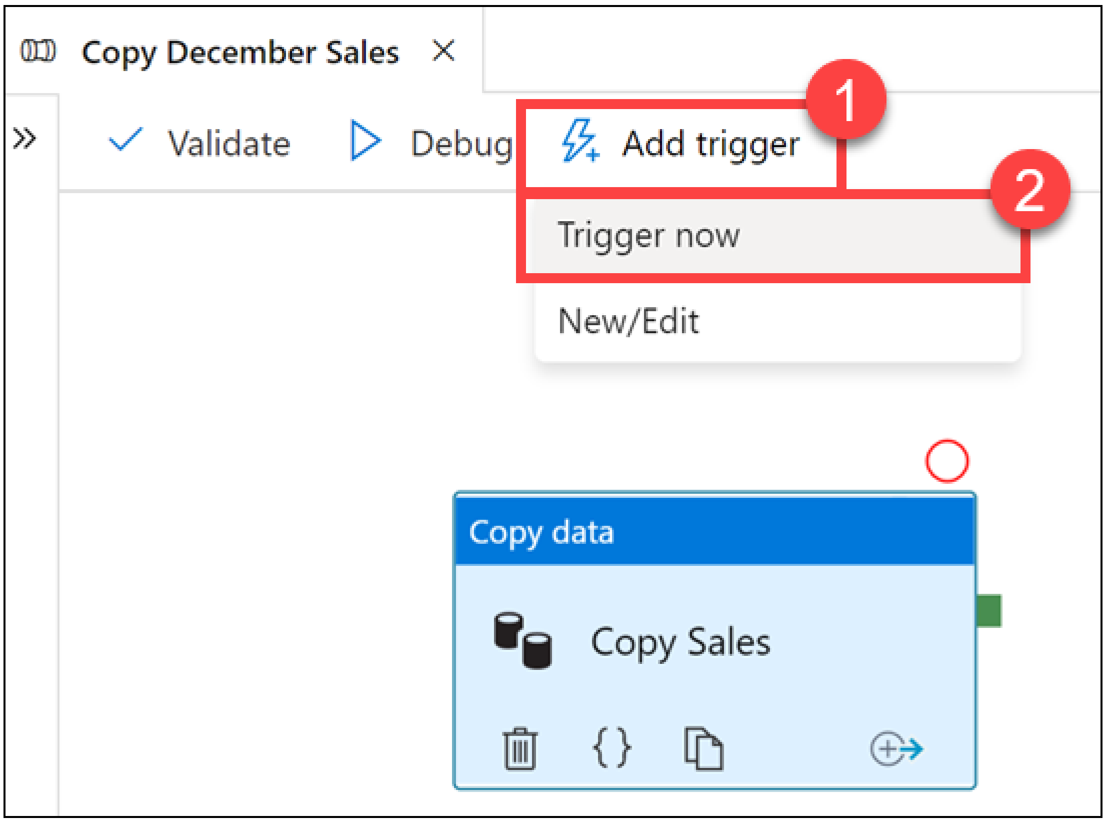
Fault tolerance:

Enable staging

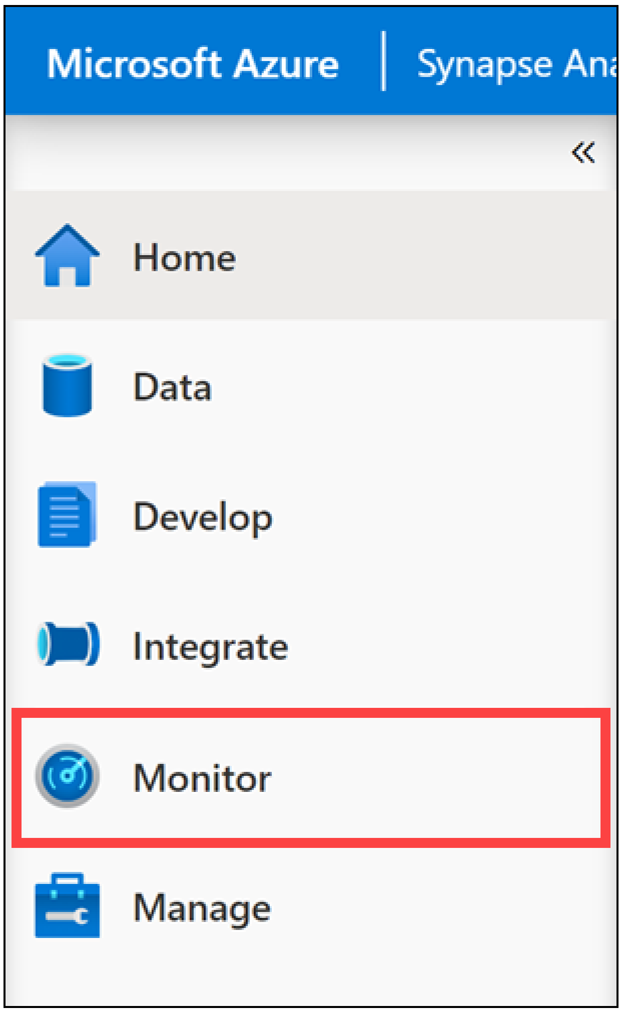
16. Select **Publish all**, then **Publish** to save your new resources.



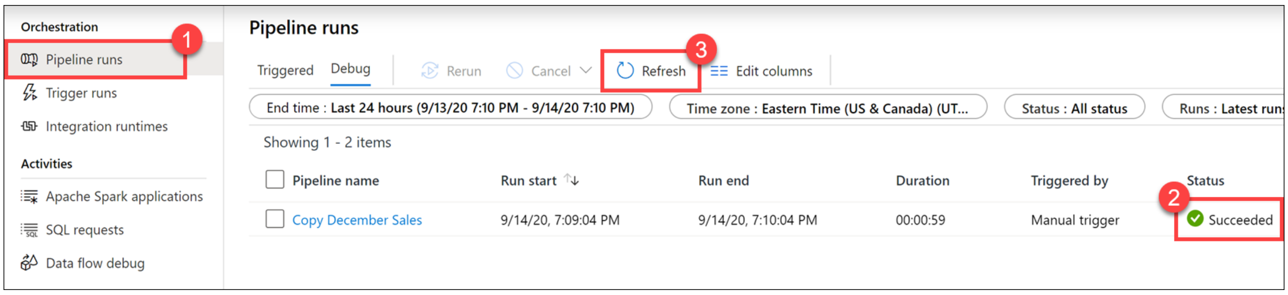
17. Select **Add trigger**, then **Trigger now**. Then select **OK** in the **Pipeline run** pane to start the pipeline.



18. Navigate to the **Monitor** hub.



19. Select **Pipeline Runs**. You can see the status of your pipeline run here. Note that you may need to refresh the view. Once the pipeline run is complete, you can query the **wwi_perf.Sale_Heap** table to view the imported data.



End time : Last 24 hours (9/13/20 7:10 PM - 9/14/20 7:10 PM) | Time zone : Eastern Time (US & Canada) (UT... | Status : All status | Runs : Latest run

Showing 1 - 2 items

Pipeline name	Run start	Run end	Duration	Triggered by	Status
Copy December Sales	9/14/20, 7:09:04 PM	9/14/20, 7:10:04 PM	00:00:59	Manual trigger	✓ Succeeded

Task 3: Execute PowerShell script to Prepare CosmosDB

1. In the hosted VM environment provided for this course, open Powershell in administrator mode, and execute the following to set the execution policy to Unrestricted so you can run the local PowerShell script file:

```
Set-ExecutionPolicy Unrestricted
```

> **Note**: If you receive a prompt that you are installing the module from an untrusted repository, select **Yes to All** to proceed with the setup.

2. Change directories to the root of this repo within your local file system.

```
cd C:\dp-203\data-engineering-ilt-
deployment\Allfiles\00\artifacts\environment-setup\automation\
```

3. Enter the following command to run a PowerShell script that Creates the Cosmos DB objects and runs the pipelines to load the data from Synapse to Cosmos DB:

```
.\dp-203-setup-Part03.ps1
```

> **NOTE**: This script should run in roughly 10-15 minutes and loads data into CosmosDB.

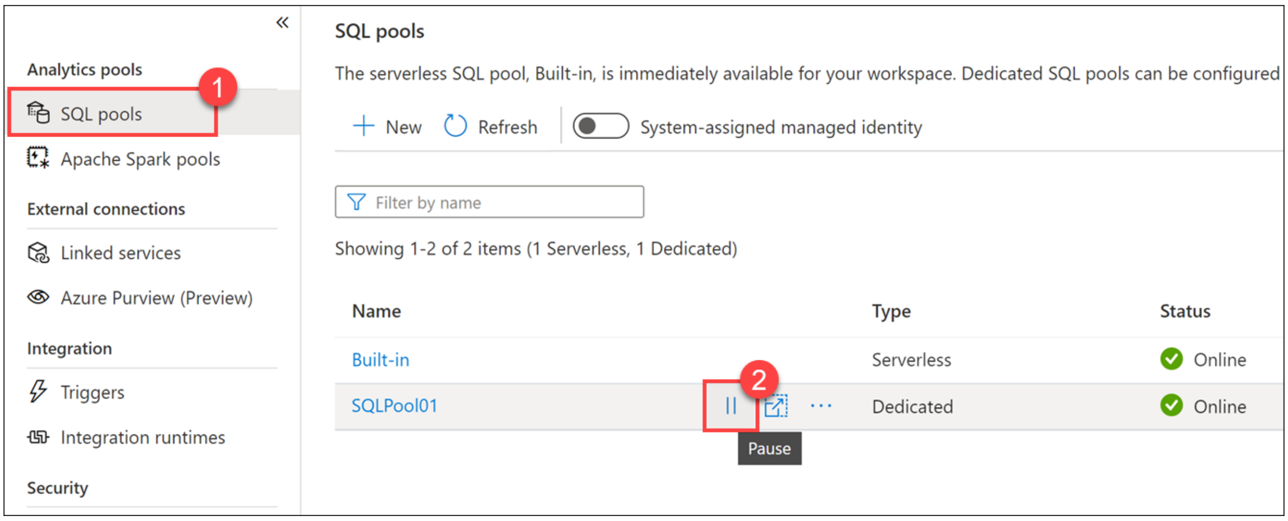
4. When prompted to sign into Azure, and your browser opens; sign in using your credentials. After signing in, you can close the browser and return to Windows PowerShell.

5. When prompted, sign into your Azure account again (this is required so that the script can manage resources in your Azure subscription - be sure you use the same credentials as before).

6. If you have more than one Azure subscription, when prompted, select the one you want to use in the labs by entering its number in the list of subscriptions.

7. In Synapse Studio, select the **Manage** hub.

8. Select **SQL pools** in the left-hand menu. Hover over the **SQLPool01** dedicated SQL pool and select **||**.



SQL pools

The serverless SQL pool, Built-in, is immediately available for your workspace. Dedicated SQL pools can be configured

Showing 1-2 of 2 items (1 Serverless, 1 Dedicated)

Name	Type	Status
Built-in	Serverless	Online
SQLPool01	Dedicated	Online

9. When prompted, select **Pause**.

Important: Pause your SQL pool when the PowerShell script has completed running

Complete these steps to free up resources you no longer need.