Microsoft Cloud Workshop

Azure Synapse Analytics and AI
Hands-on lab step-by-step

Contents

- Exercise 1: Accessing the Azure Synapse Analytics workspace
  - Task 1: Launching Synapse Studio
- Exercise 2: Create and populate the supporting tables in the SQL Pool
  - Task 1: Create the sale table
  - Task 2: Populate the sale table
  - Task 3: Create the customer information table
  - Task 4: Populate the customer information table
  - Task 5: Create the campaign analytics table
  - Task 6: Populate the campaign analytics table
  - Task 7: Populate the product table
- Exercise 3: Exploring raw parquet
  - Task 1: Query sales Parquet data with Synapse SQL Serverless
  - Task 2: Query sales Parquet data with Azure Synapse Spark
- Exercise 4: Exploring raw text based data with Azure Synapse SQL Serverless
  - Task 1: Query CSV data
  - Task 2: Query JSON data
- Exercise 5: Synapse Pipelines and Cognitive Search
  - Task 1: Create the invoice storage container
  - Task 2: Create and train an Azure Forms Recognizer model & setup Cognitive Search
  - Task 3: Configure a skillset with Form Recognizer
  - Task 4: Create the Synapse Pipeline
- Exercise 6: Security
  - Task 1: Column level security
  - Task 2: Row level security
  - Task 3: Dynamic data masking
- Exercise 7: Machine Learning
  - Task 1: Analyze data with Apache Spark
  - Task 2: Training, consuming, and deploying models
- Exercise 8: Monitoring
  - Task 1: Workload importance
  - Task 2: Workload isolation
  - Task 3: Monitoring with Dynamic Management Views
  - Task 4: Orchestration Monitoring with the Monitor Hub
  - Task 5: Monitoring SQL Requests with the Monitor Hub
- After the hands-on lab
  - Task 1: Delete the resource group

Azure Synapse Analytics and AI hands-on lab step-by-step

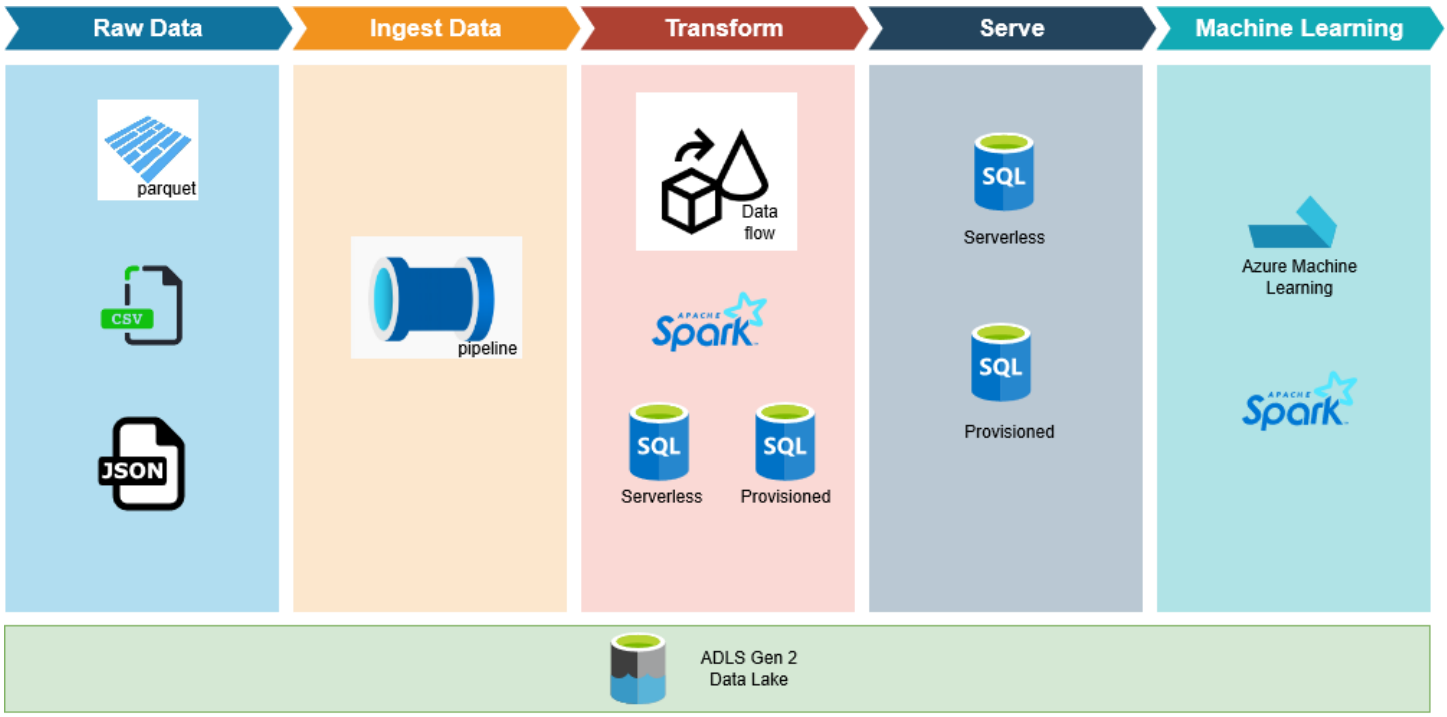
Abstract and learning objectives

In this hands-on-lab, you will build an end-to-end data analytics with machine learning solution using Azure Synapse Analytics. The information will be presented in the context of a retail scenario. We will be heavily leveraging Azure Synapse Studio, a tool that conveniently unifies the most common data operations from ingestion, transformation, querying, and visualization.

Overview

In this lab various features of Azure Synapse Analytics will be explored. Azure Synapse Analytics Studio is a single tool that every team member can use collaboratively. Synapse Studio will be the only tool used throughout this lab through data ingestion, cleaning, and transforming raw files to using Notebooks to train, register, and consume a Machine learning model. The lab will also provide hands-on-experience monitoring and prioritizing data related workloads.

Solution architecture



This lab explores the cold data scenario of ingesting various types of raw data files. These files can exist anywhere. The file types used in this lab are CSV, parquet, and JSON. This data will be ingested into Synapse Analytics via Pipelines. From there, the data can be transformed and enriched using various tools such as data flows, Synapse Spark, and Synapse SQL (both provisioned and serverless). Once processed,

data can be queried using Synapse SQL tooling. Azure Synapse Studio also provides the ability to author notebooks to further process data, create datasets, train, and create machine learning models. These models can then be stored in a storage account or even in a SQL table. These models can then be consumed via various methods, including T-SQL. The foundational component supporting all aspects of Azure Synapse Analytics is the ADLS Gen 2 Data Lake.

Requirements

1. Microsoft Azure subscription
2. Azure Synapse Workspace / Studio

Before the hands-on lab

Refer to the Before the hands-on lab setup guide manual before continuing to the lab exercises.

Resource naming throughout this lab

For the remainder of this lab, the following terms will be used for various ASA (Azure Synapse Analytics) related resources (make sure you replace them with actual names and values from your environment):

Azure Synapse Analytics Resource	To be referred to
Azure Subscription	`WorkspaceSubscription`
Azure Region	`WorkspaceRegion`
Workspace resource group	`WorkspaceResourceGroup`
Workspace / workspace name	`asaworkspace{suffix}`
Primary Storage Account	`asadatalake{suffix}`
Default file system container	`DefaultFileSystem`
SQL Pool	`SqlPool01`
SQL Serverless Endpoint	`SqlServerless01`
Azure Key Vault	`asakeyvault{suffix}`

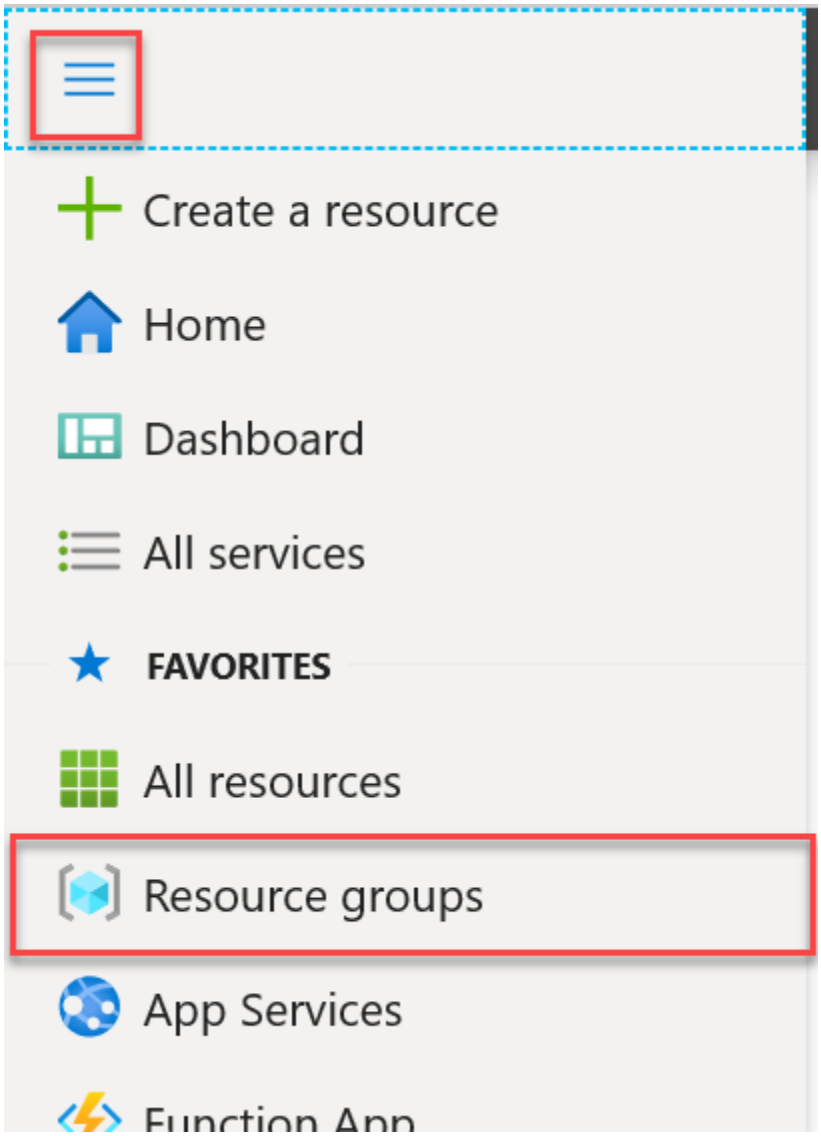
Exercise 1: Accessing the Azure Synapse Analytics workspace

Duration: 5 minutes

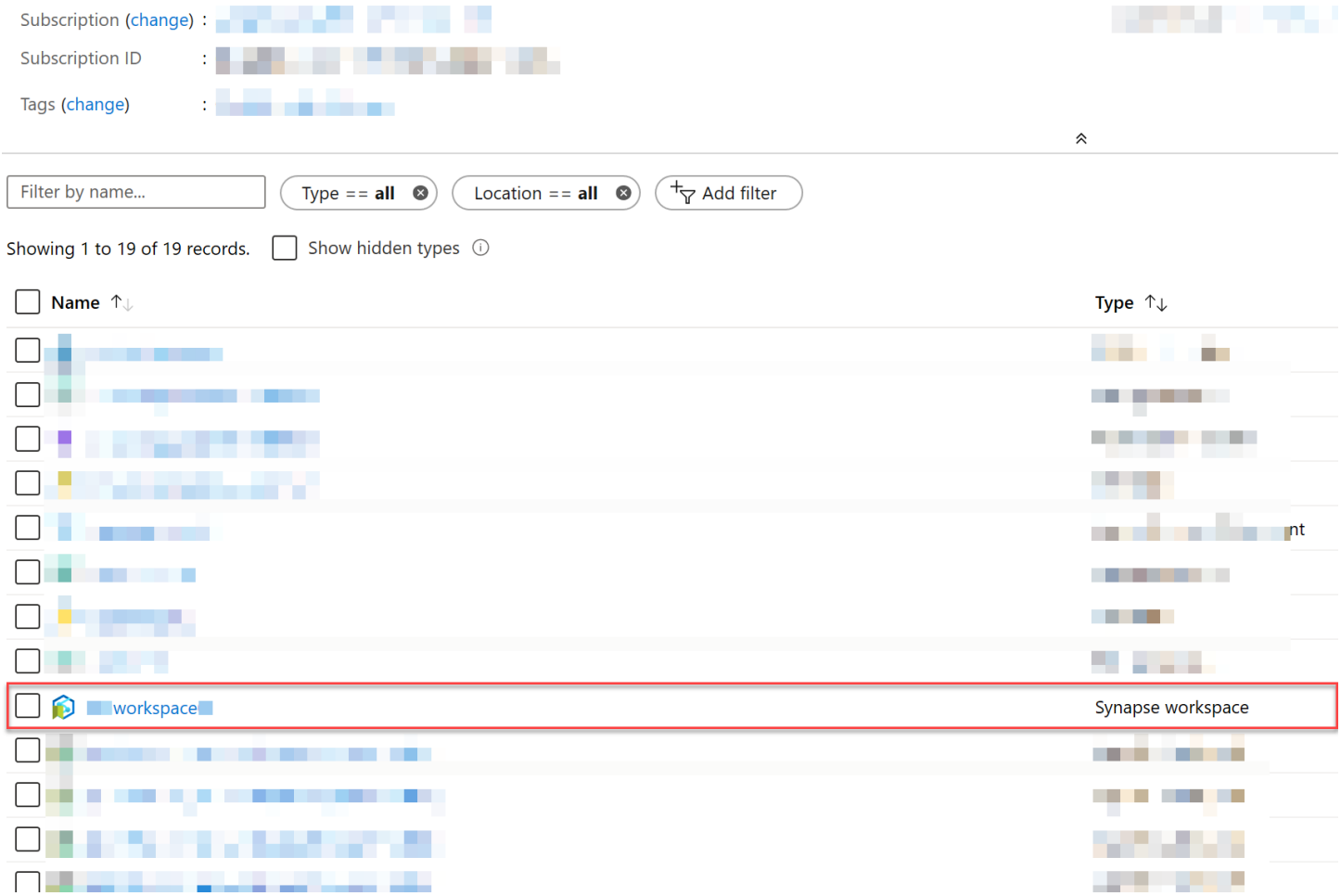
All exercises in this lab utilize the workspace Synapse Studio user interface. This exercise will outline the steps to launch Synapse Studio. Unless otherwise specified, all instruction including menu navigation will occur in Synapse Studio.

Task 1: Launching Synapse Studio

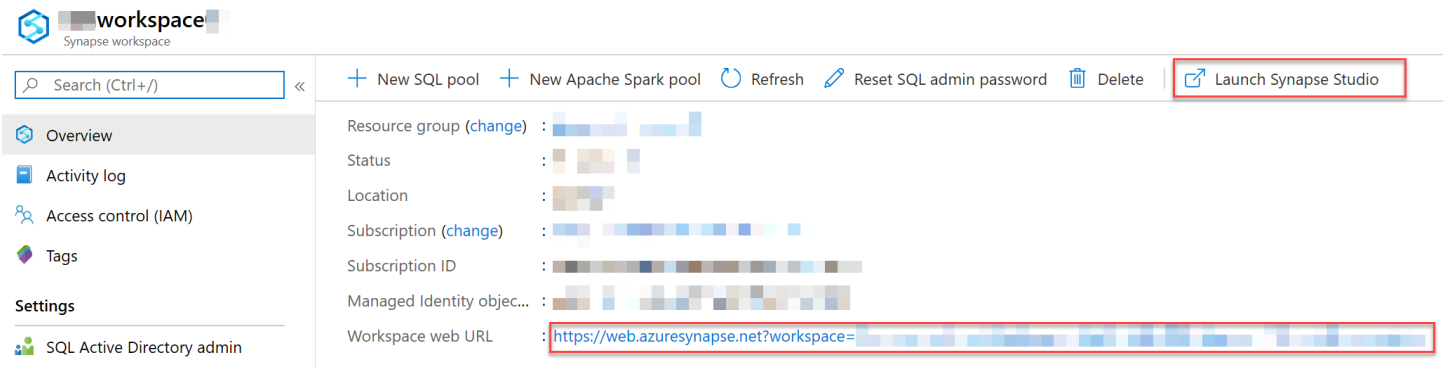
1. Log into the Azure Portal.
2. Expand the left menu, and select the **Resource groups** item.



3. From the list of resource groups, select `WorkspaceResourceGroup`.
4. From the list of resources, select the **Synapse Workspace** resource, `asaworkspace{suffix}`.



5. On the **Overview** tab of the Synapse Workspace page, select the **Launch Synapse Studio** item from the top toolbar. Alternatively you can select the Workspace web URL link.



Exercise 2: Create and populate the supporting tables in the SQL Pool

Duration: 120 minutes

The first step in querying meaningful data is to create tables to house the data. In this case, we will create four different tables: SaleSmall, CustomerInfo, CampaignAnalytics, and Sales. When designing tables in Azure Synapse Analytics, we need to take into account the expected amount of data in each table, as well as how each table will be used. Utilize the following guidance when designing your tables to ensure the best experience and performance.

Table design performance considerations

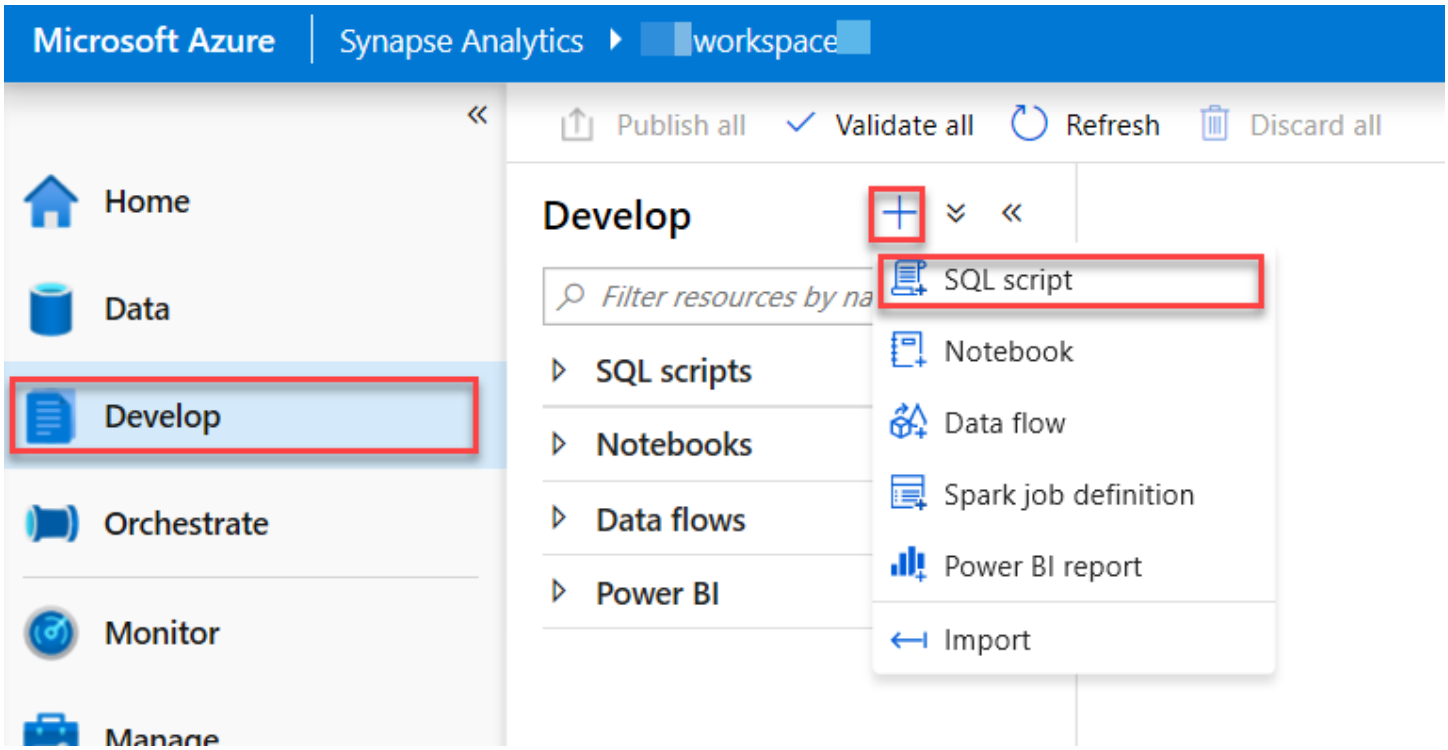
Table Indexing	Recommended use
Clustered Columnstore	Recommended for tables with greater than 100 million rows, offers the highest data compression with best overall query performance.
Heap Tables	Smaller tables with less than 100 million rows, commonly used as a staging table prior to transformation.
Clustered Index	Large lookup tables (> 100 million rows) where querying will only result in a single row returned.
Clustered Index + non-clustered secondary index	Large tables (> 100 million rows) when single (or very few) records are being returned in queries.

Table Distribution/Partition Type	Recommended use
Hash distribution	Tables that are larger than 2 GBs with infrequent insert/update/delete operations, works well for large fact tables in a star schema.
Round robin distribution	Default distribution, when little is known about the data or how it will be used. Use this distribution for staging tables.
Replicated tables	Smaller lookup tables, less than 1.5 GB in size.

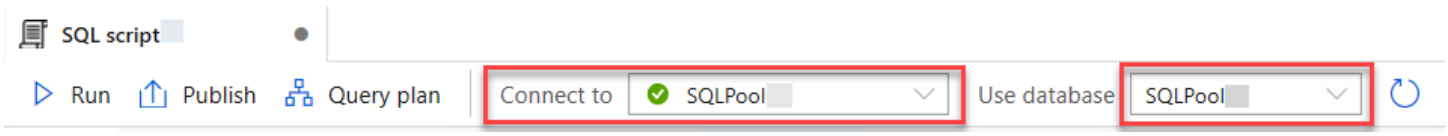
Task 1: Create the sale table

Over the past 5 years, Wide World Importers has amassed over 3 billion rows of sales data. With this quantity of data, the storage consumed would be greater than 2 GB. While we will be using only a subset of this data for the lab, we will design the table for the production environment. Using the guidance outlined in the current Exercise description, we can ascertain that we will need a **Clustered Columnstore** table with a **Hash** table distribution based on the **CustomerId** field which will be used in most queries. For further performance gains, the table will be partitioned by transaction date to ensure queries that include dates or date arithmetic are returned in a favorable amount of time.

1. Expand the left menu and select the **Develop** item. From the **Develop** blade, expand the **+** button and select the **SQL script** item.



2. In the query tab toolbar menu, ensure you connect to your SQL Pool, `SQLPool01`.

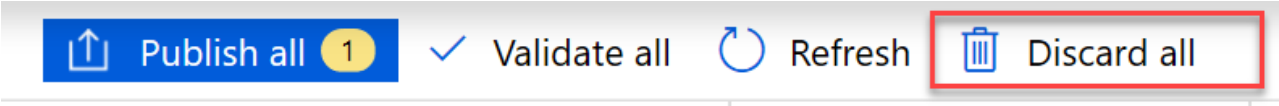


3. In the query window, copy and paste the following query to create the customer information table. Then select the **Run** button in the query tab toolbar.

```
CREATE TABLE [wwi_mcw].[SaleSmall]
(
  [TransactionId] [uniqueidentifier]  NOT NULL,
  [CustomerId] [int]  NOT NULL,
  [ProductId] [smallint]  NOT NULL,
  [Quantity] [tinyint]  NOT NULL,
  [Price] [decimal](9,2)  NOT NULL,
  [TotalAmount] [decimal](9,2)  NOT NULL,
  [TransactionDateId] [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
  (
    [TransactionDateId] RANGE RIGHT FOR VALUES (
```

```
        20180101, 20180201, 20180301, 20180401, 20180501, 20180601, 20180701, 20180801, 2018090
1, 20181001, 20181101, 20181201,
        20190101, 20190201, 20190301, 20190401, 20190501, 20190601, 20190701, 20190801, 2019090
1, 20191001, 20191101, 20191201)
    )
  );
```

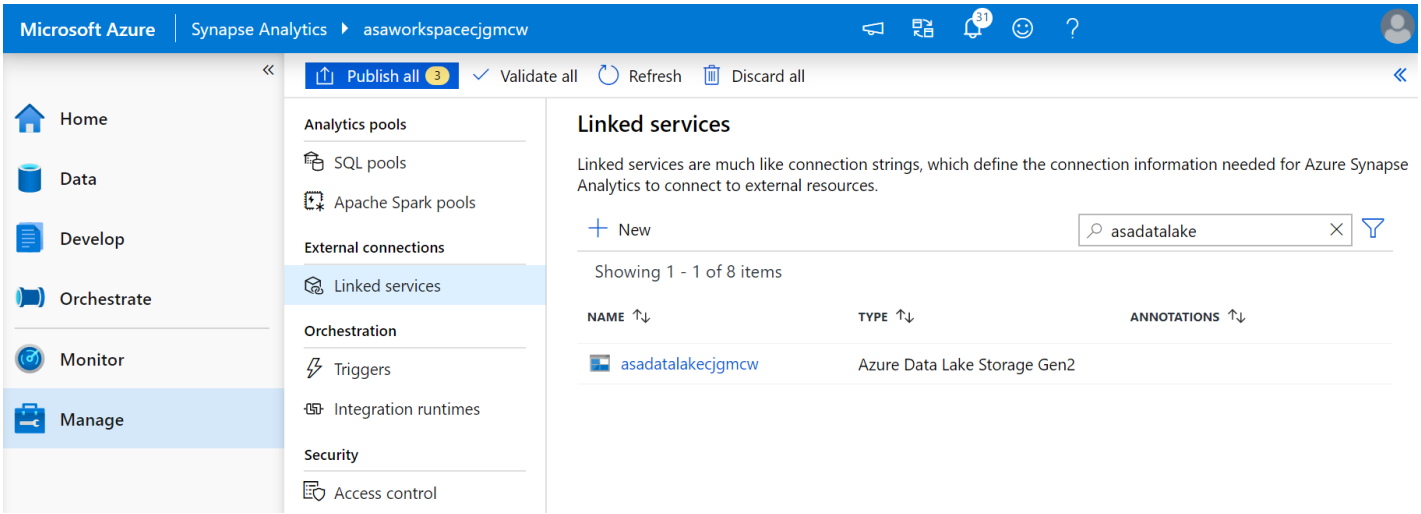
4. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Task 2: Populate the sale table

The data that we will be retrieving to populate the sale table is currently stored as a series of parquet files in the **asadatalake{SUFFIX}** data lake (Azure Data Lake Storage Gen 2). This storage account has already been added as a linked service in Azure Synapse Analytics when the environment was provisioned. Linked Services are synonymous with connection strings in Azure Synapse Analytics. Azure Synapse Analytics linked services provides the ability to connect to nearly 100 different types of external services ranging from Azure Storage Accounts to Amazon S3 and more.

1. Review the presence of the **asadatalake{SUFFIX}** linked service, by selecting **Manage** from the left menu, and selecting **Linked services** from the blade menu. Filter the linked services by the term **asadatalake** to find the **asadatalake{SUFFIX}** item. Further investigating this item will unveil that it makes a connection to the storage account using a storage account key.



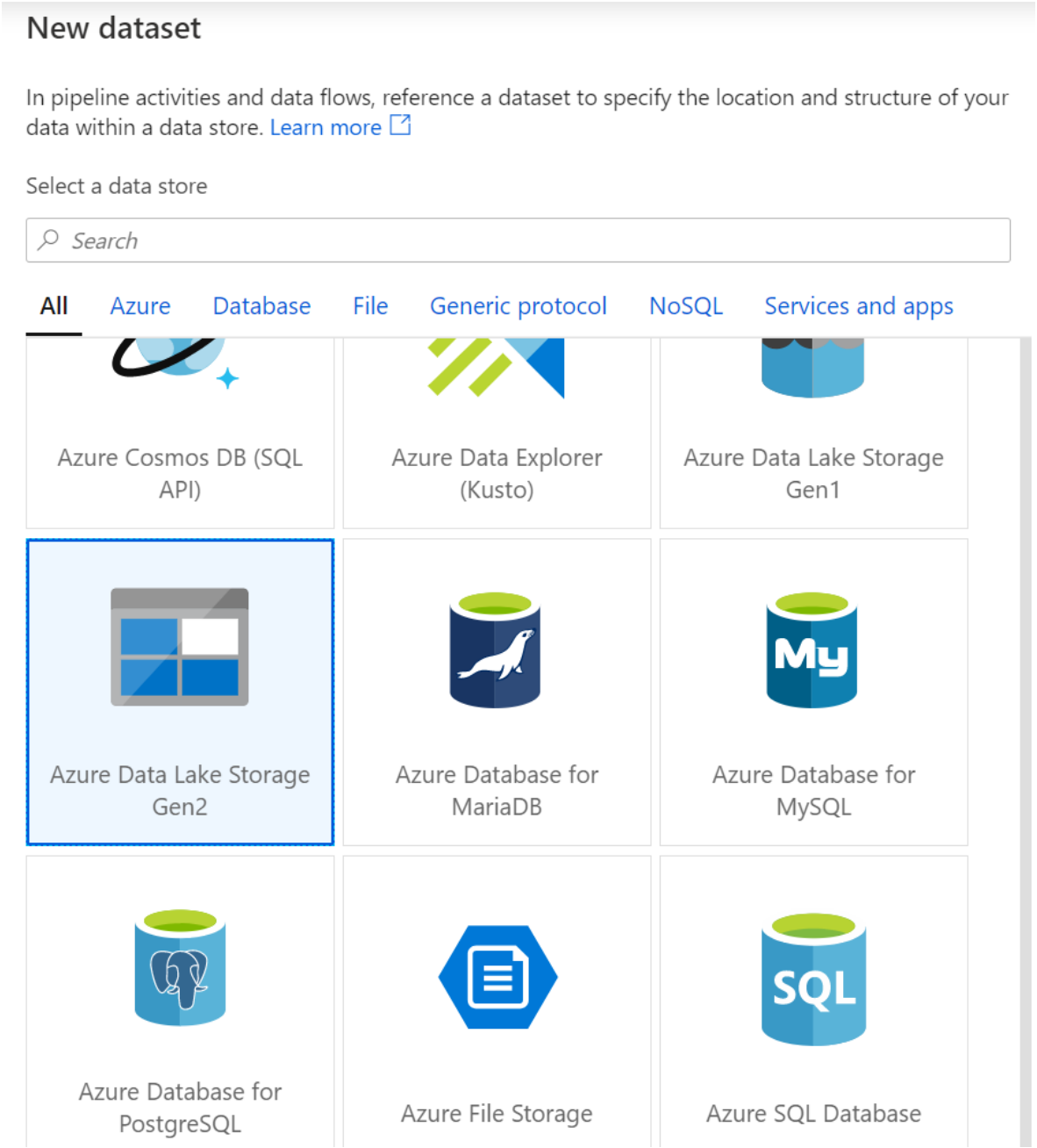
2. The sale data for each day is stored in a separate parquet file which is placed in storage following a known convention. In this lab, we are interested in populating the Sale table with only 2018 and 2019 data. Investigate the structure of the data by selecting the **Data** tab, and in the **Data** pane, select the **Linked** tab, and expanding the `asadatalake{SUFFIX}` Storage account.

Note: The current folder structure for daily sales data is as follows: /wwi-02/sale-small/Year=YYYY/Quarter=Q#/Month=M/Day=YYYYMMDD, where `YYYY` is the 4 digit year (eg. 2019), `Q#` represents the quarter (eg. Q1), `M` represents the numerical month (eg. 1 for January) and finally `YYYYMMDD` represents a numeric date format representation (eg. `20190516` for May 16, 2019). A single parquet file is stored each day folder with the name **sale-small-YYYYMMDD-snappy.parquet** (replacing `YYYYMMDD` with the numeric date representation).

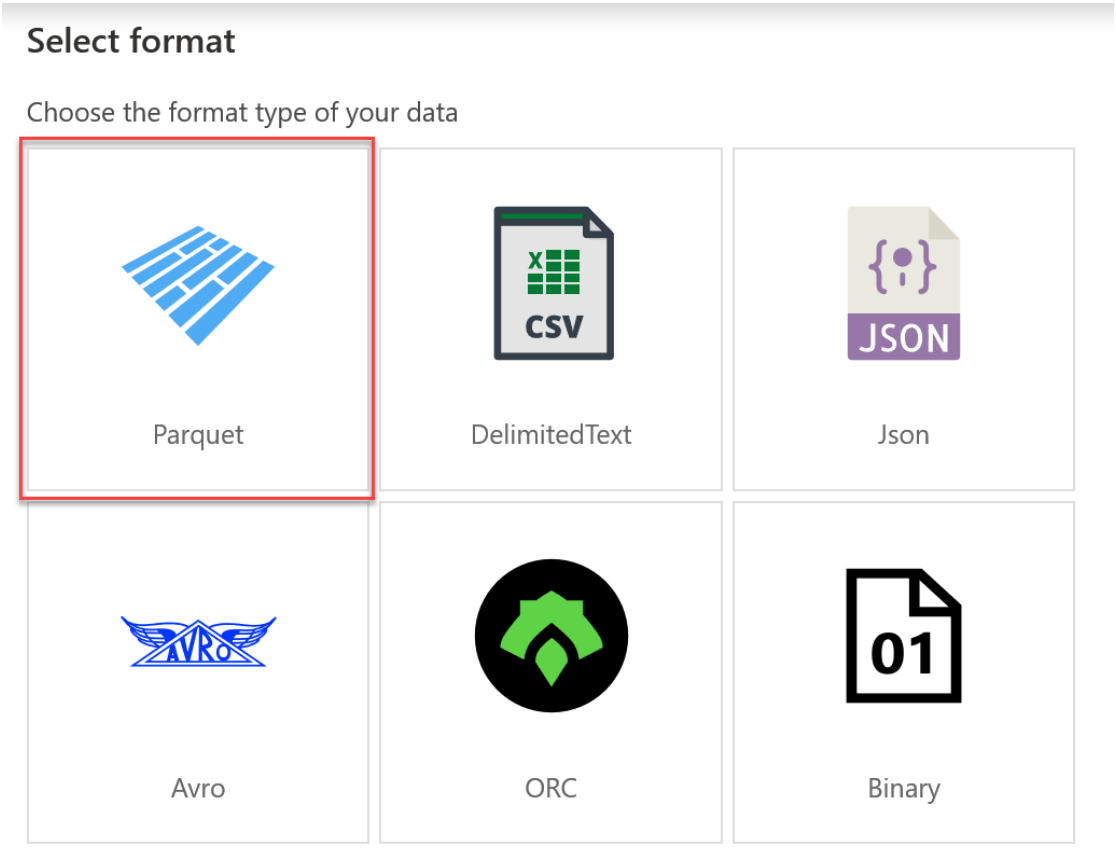
```
Sample path to the parquet folder for January 1, 2019:
/wwi-02/sale-small/Year=2019/Quarter=Q1/Month=1/Day=20190101/sale-small-20190101-snappy.parquet
```

3. Create a new Dataset by selecting **Data** from the left menu, expanding the **+** button on the Data blade and selecting **Dataset**. We will be creating a dataset that will point to the root folder of the sales data in the data lake.

4. In the **New dataset** blade, with the **All** tab selected, choose the **Azure Data Lake Storage Gen2** item. Select **Continue**.

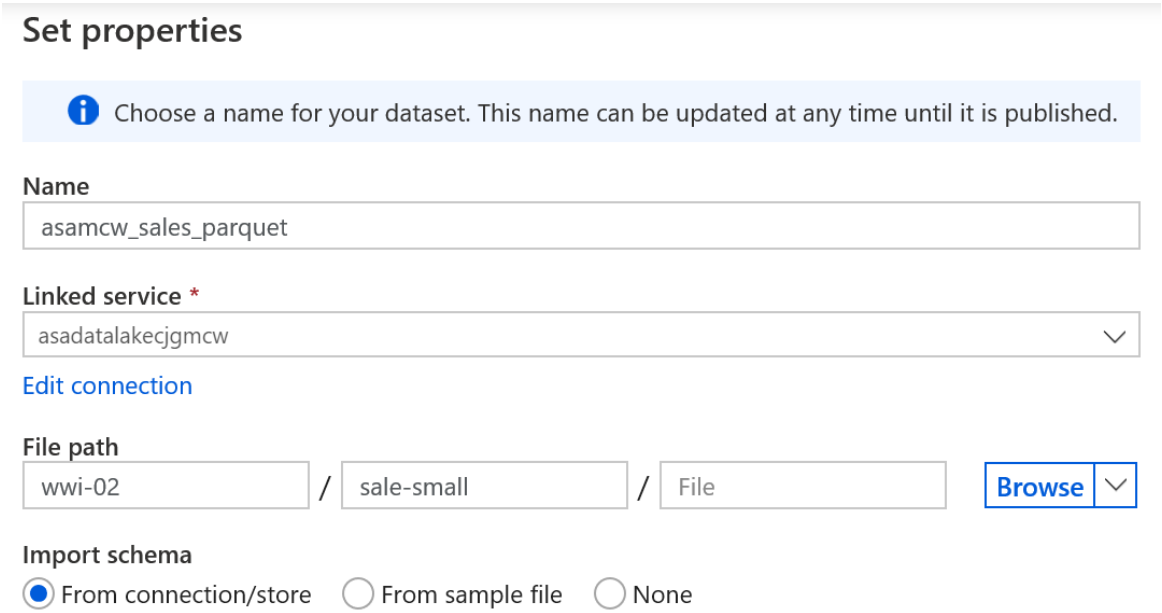


5. In the **Select format** screen, choose the **Parquet** item. Select **Continue**.



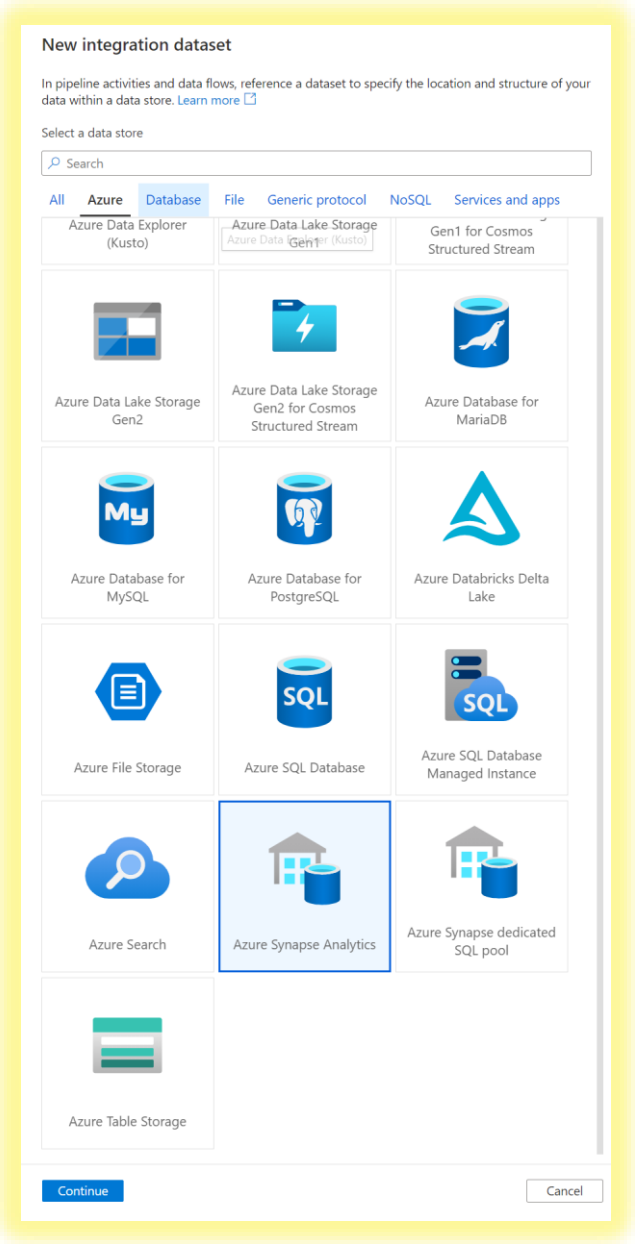
6. In the **Set properties** blade, populate the form as follows then select **OK**.

Field	Value
Name	Enter **asamcw_sales_parquet**.
Linked service	**asadatalake{SUFFIX}**
File path - Container	Enter **wwi-02**.
File path - Folder	Enter **sale-small**.
Import schema	**From connection/store**



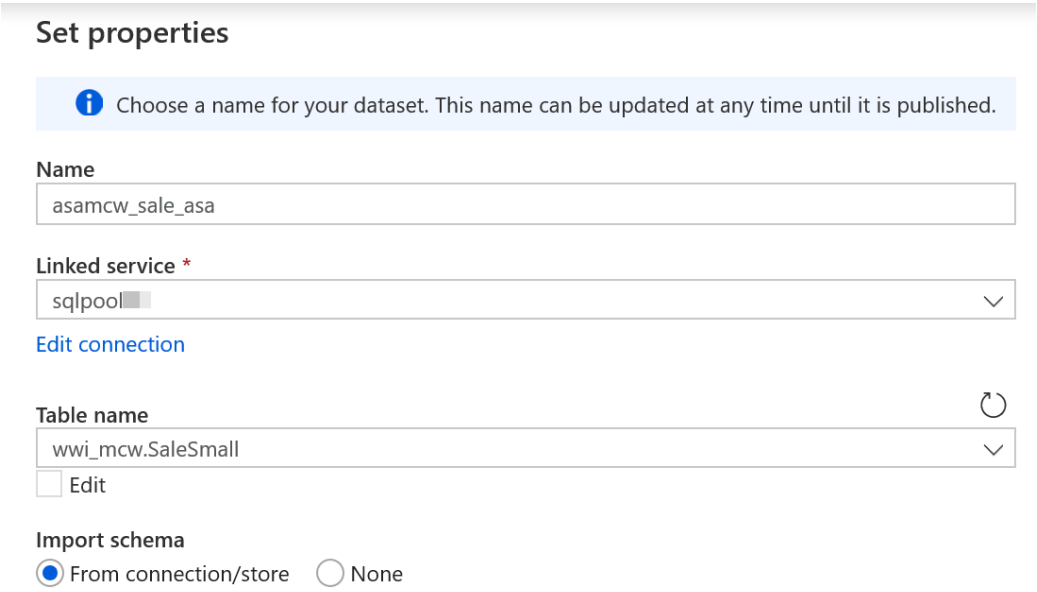
7. Now we will need to define the destination dataset for our data. In this case we will be storing sale data in our SQL Pool. Create a new dataset by expanding the **+** button on the **Data** blade and selecting **Dataset**.

8. On the **New dataset** blade, with the **Azure** tab selected, enter **synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue**.



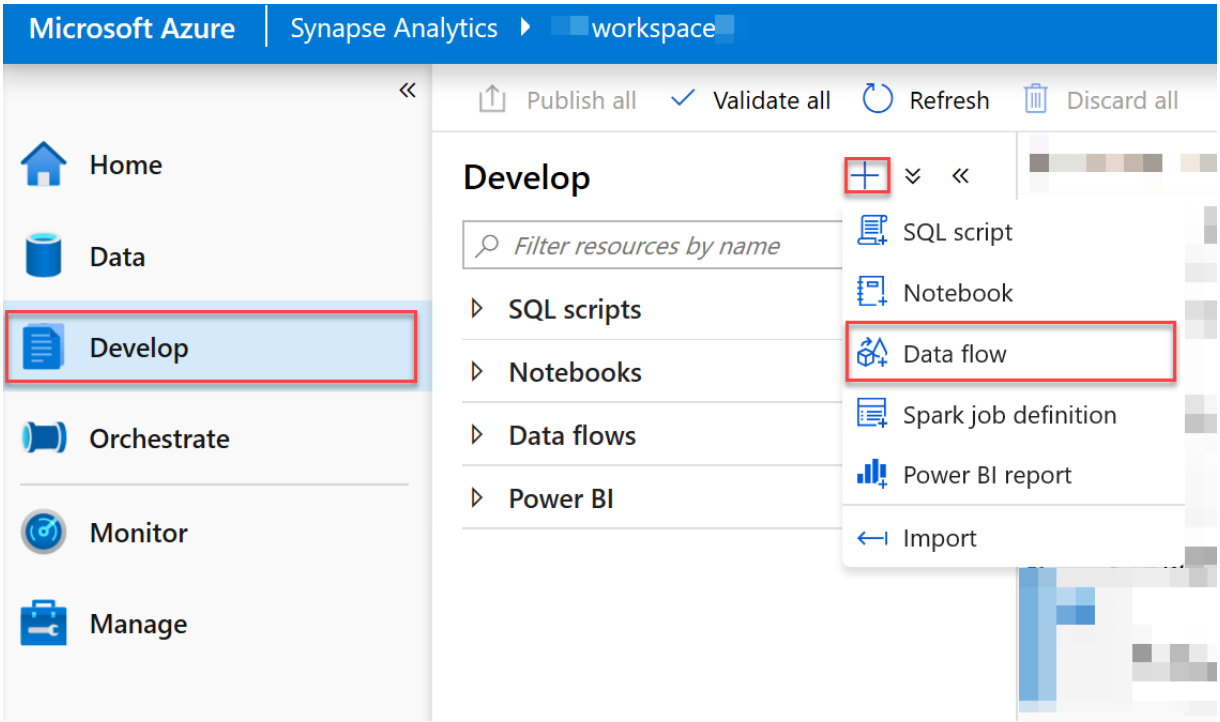
9. On the **Set properties** blade, set the field values to the following, then select **OK**.

Field	Value
Name	Enter **asamcw_sale_asa**.
Linked service	**SQLPool01**
Table name	**wwi_mcw.SaleSmall**
Import schema	**From connection/store**

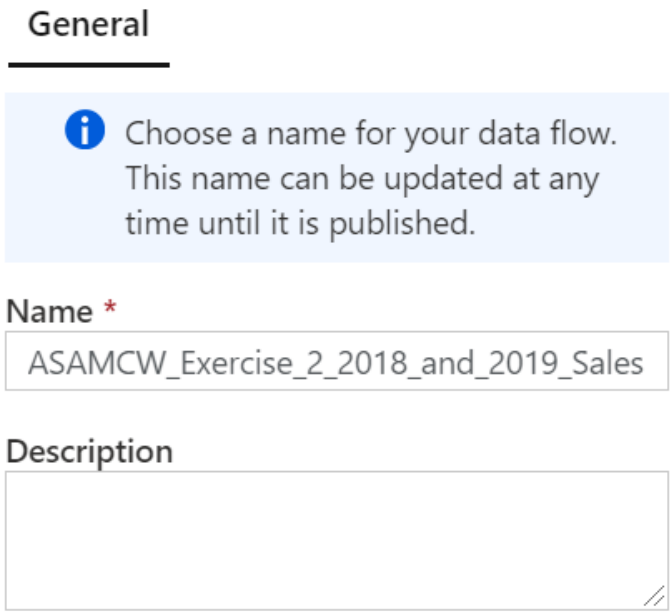


10. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to deploy the changes to the workspace.

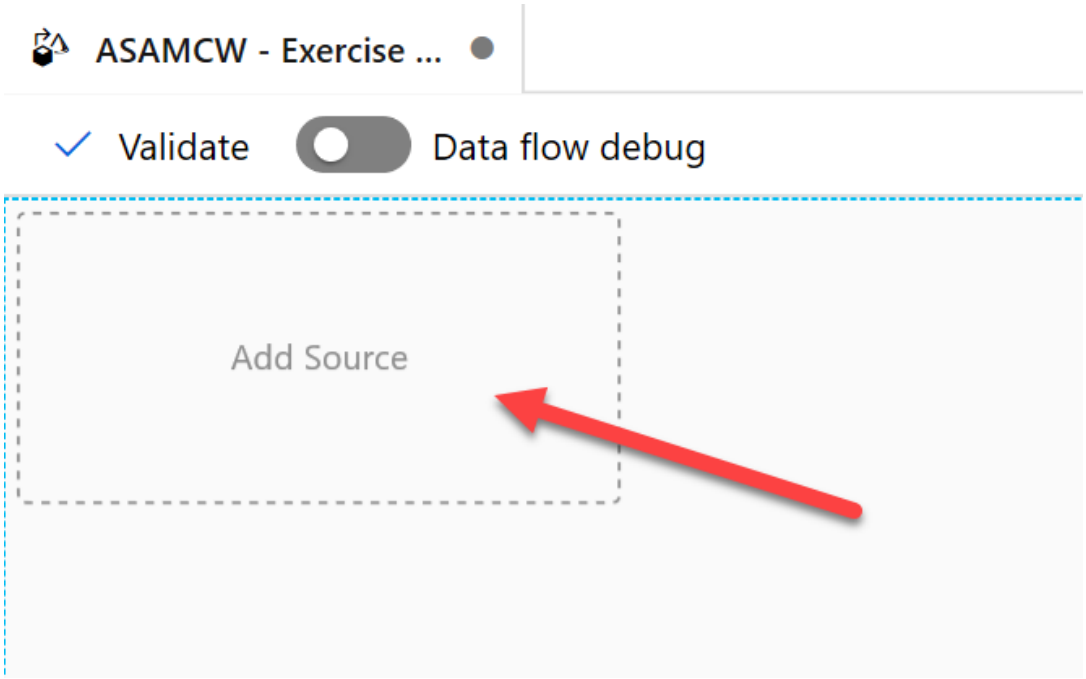
11. Since we want to filter on multiple sale year folders (Year=2018 and Year=2019) and copy only the 2018 and 2019 sales data, we will need to create a data flow to define the specific data that we wish to retrieve from our source dataset. To create a new data flow, start by selecting **Develop** from the left menu, and in the **Develop** blade, expand the **+** button and select **Data flow**. ~exclude 2010 data that is in data lake~



12. In the side pane on the **General** tab, name the data flow by entering **ASAMCW_Exercise_2_2018_and_2019_Sales** in the **Name** field.



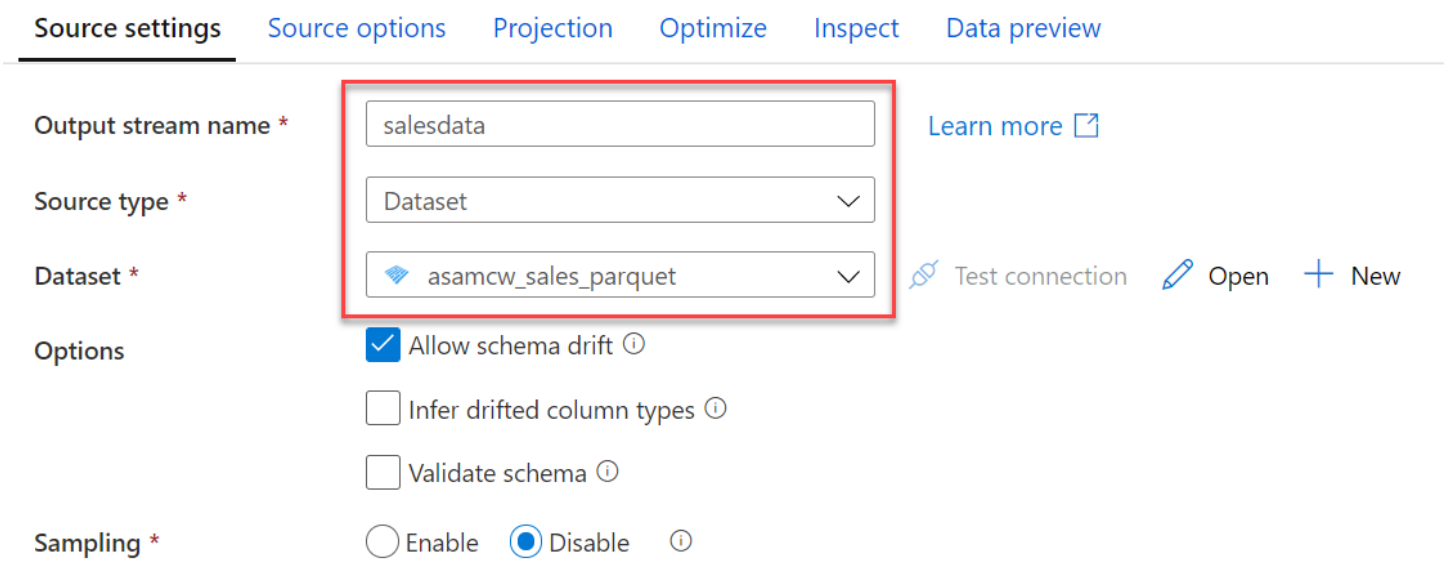
13. In the data flow designer window, select the **Add Source** box.



14. With the added source selected in the designer, in the lower pane with the **Source settings** tab selected, set the following field values:

Field	Value
Output stream name	Enter **salesdata**.

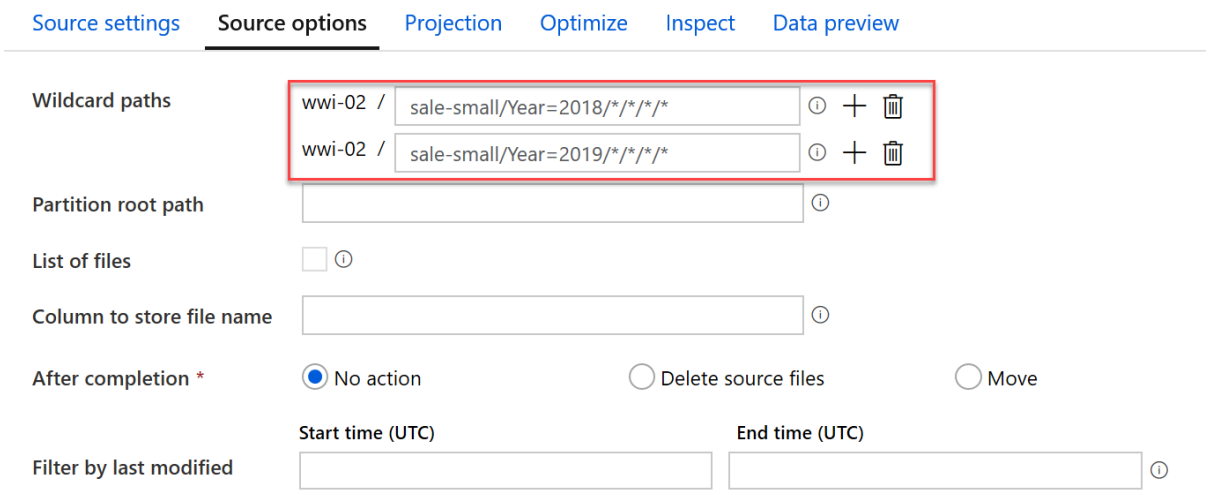
Field	Value
Source type	**Dataset**
Dataset	**asamcw_sales_parquet**



15. Select the **Source options** tab, and add the following as **Wildcard paths**, this will ensure that we only pull data from the parquet files for the sales years of 2018 and 2019: ~exclude 2010 data that is in data lake~

    i. sale-small/Year=2018/*/*/*/*

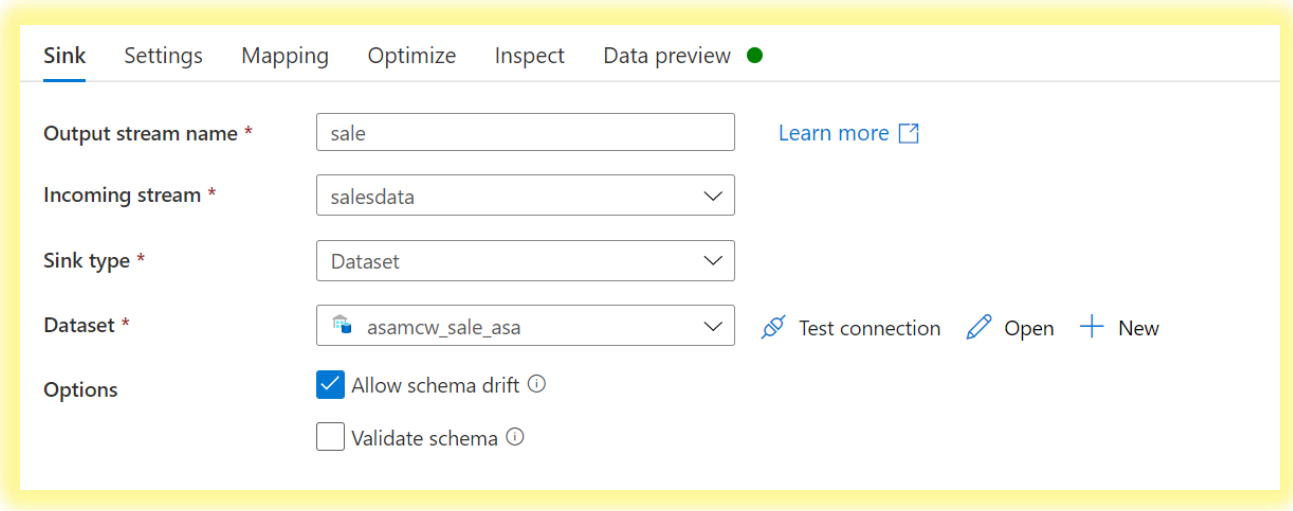
    ii. sale-small/Year=2019/*/*/*/*



16. At the bottom right of the **salesdata** source, expand the **+** button and select the **Sink** item located in the **Destination** section of the menu.

17. In the designer, select the newly added **Sink** element and in the bottom pane with the **Sink** tab selected, fill the form as follows:

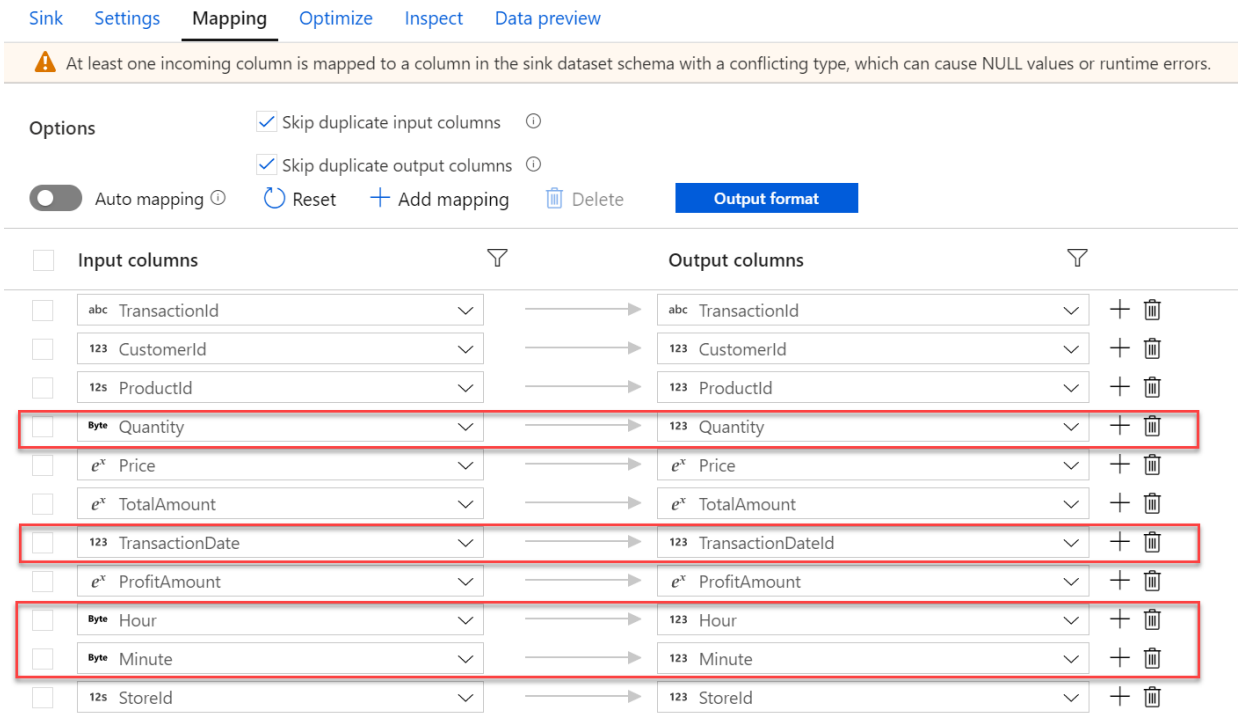
Field	Value
Output stream name	Enter **sale**.
Incoming stream	**salesdata**
Sink type	**Dataset**
Dataset	**asamcw_sale_asa**



18. Select the **Mapping** tab and toggle the **Auto mapping** setting to the off position. You will need to select Input columns for the following:

Input column	Output column
Quantity	Quantity

Input column	Output column
TransactionDate	TransactionDateId
Hour	Hour
Minute	Minute

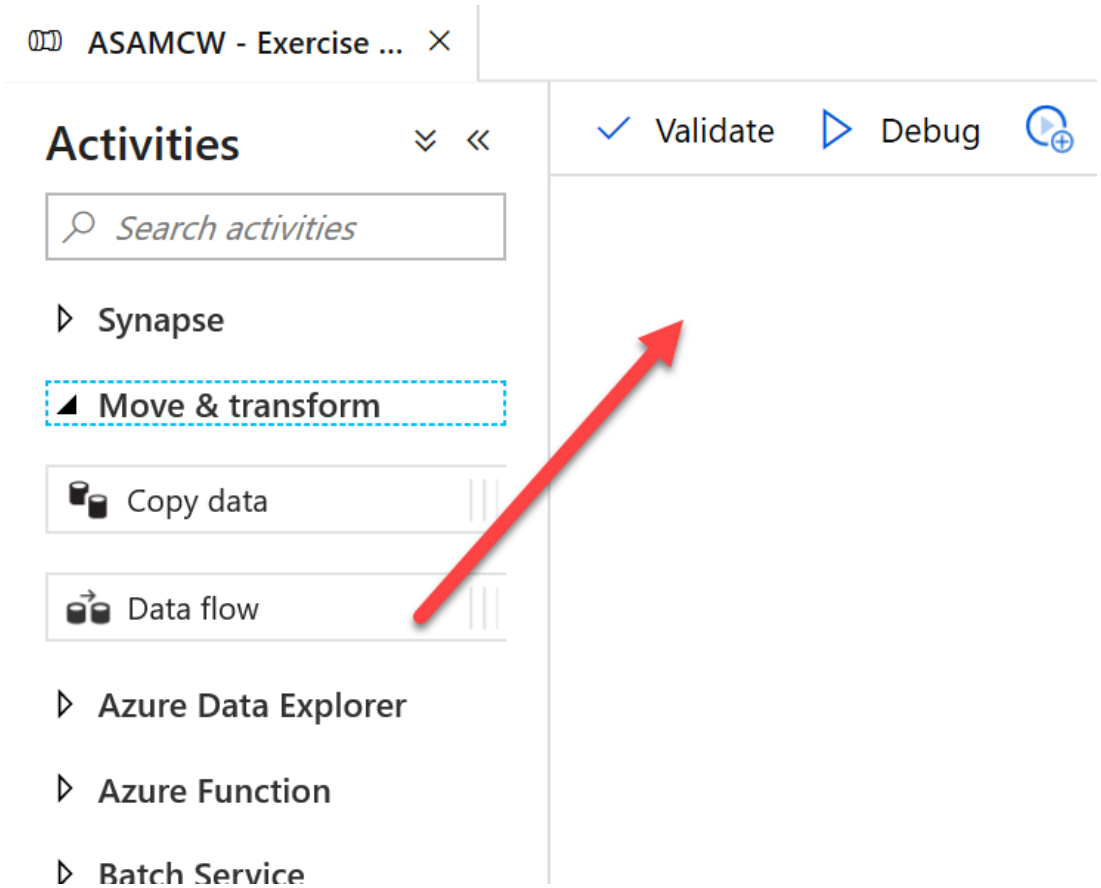


19. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to deploy the new data flow to the workspace.

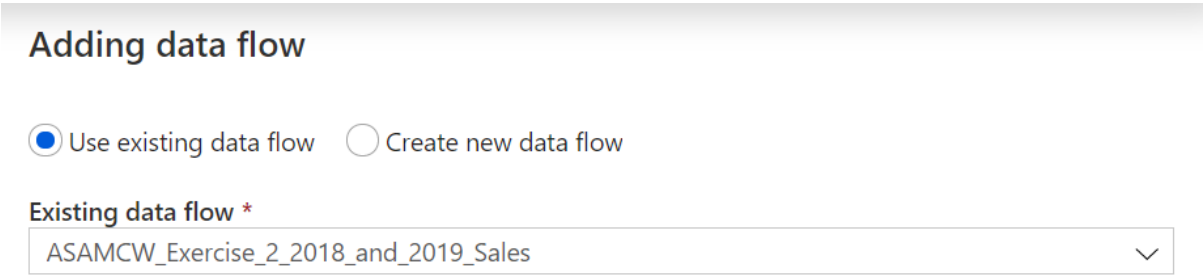
20. We can now use this data flow as an activity in a pipeline. Create a new pipeline by selecting **Integrate** from the left menu, and in the **Orchestrate** blade, expand the **+** button and select **Pipeline**.

21. On the **Properties** blade, Enter **ASAMCW - Exercise 2 - Copy Sale Data** as the Name of the pipeline.

22. From the **Activities** menu, expand the **Move & transform** section and drag an instance of **Data flow** to the design surface of the pipeline.

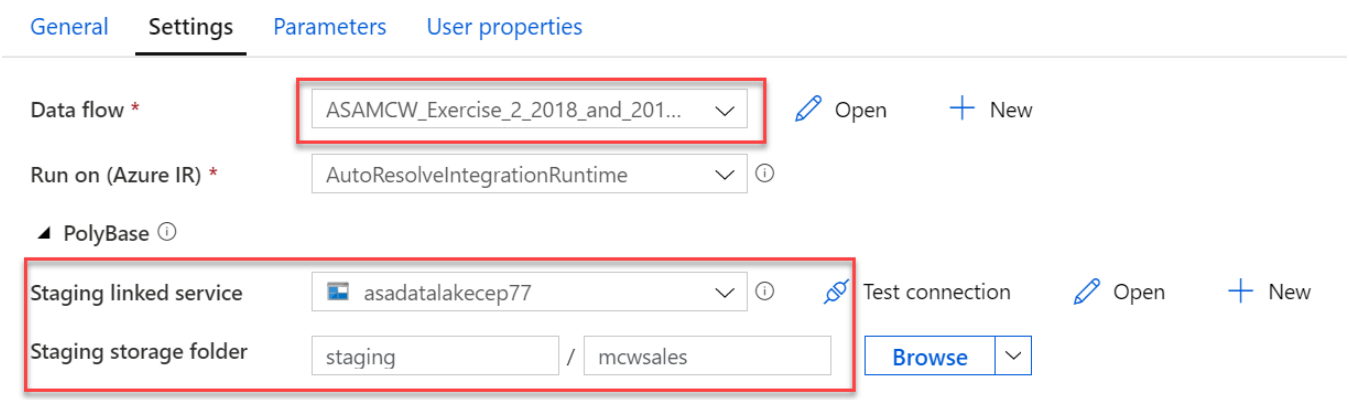


23. In the **Adding data flow** blade, ensure **Use existing data flow** is selected, and choose **ASAMCW_Exercise_2_2018_and_2019_Sales** from the select list and select **Finish**.



24. Select the **Settings** tab and set the form fields to the following values:

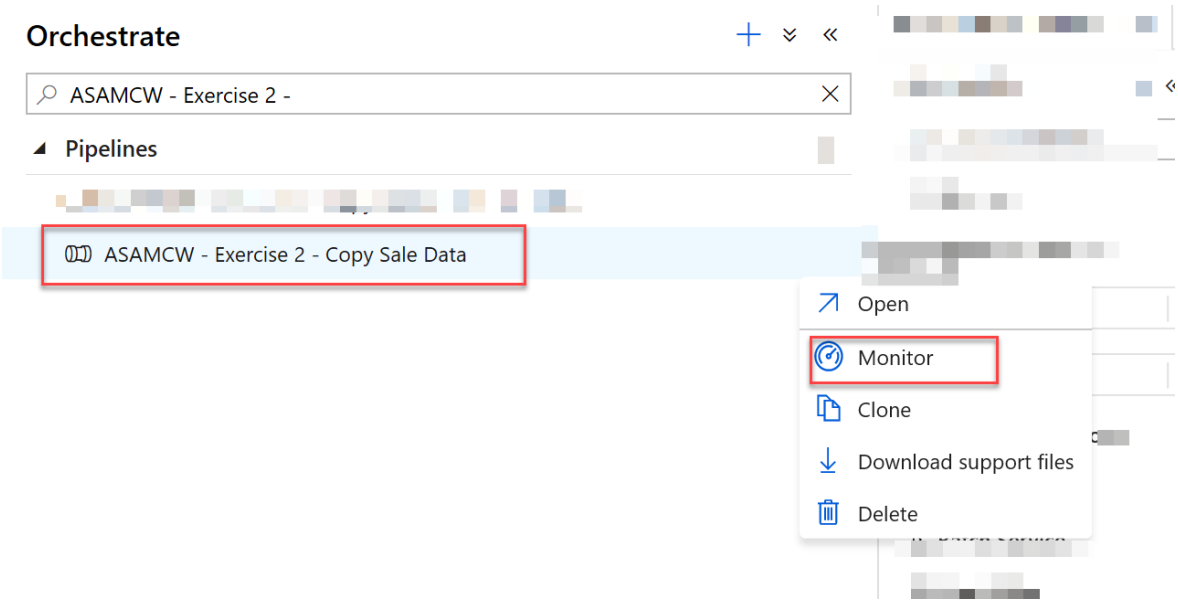
Field	Value
Data flow	**ASAMCW_Exercise_2_2018_and_2019_Sales**
Staging linked service	`asadatalake{SUFFIX}`
Staging storage folder - Container	Enter **staging**.
Staging storage folder - Folder	Enter **mcwsales**.



25. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to commit the changes.

26. Once published, expand the **Add trigger** item on the pipeline designer toolbar, and select **Trigger now**. In the **Pipeline run** blade, select **OK** to proceed with the latest published configuration. You will see notification toast windows indicating the pipeline is running and when it has completed.

27. View the status of the pipeline run by locating the **ASAMCW - Exercise 2 - Copy Sale Data** pipeline in the Orchestrate blade. Expand the actions menu, and select the **Monitor** item.



28. You should see a run of the pipeline we created in the **Pipeline runs** table showing as in progress. It will take approximately 45 minutes for this pipeline operation to complete. You will need to refresh this table from time to time to see updated progress. Once it has completed. You should see the pipeline run displayed with a Status of **Succeeded**. *Feel free to proceed to the following tasks in this exercise while this pipeline runs*.

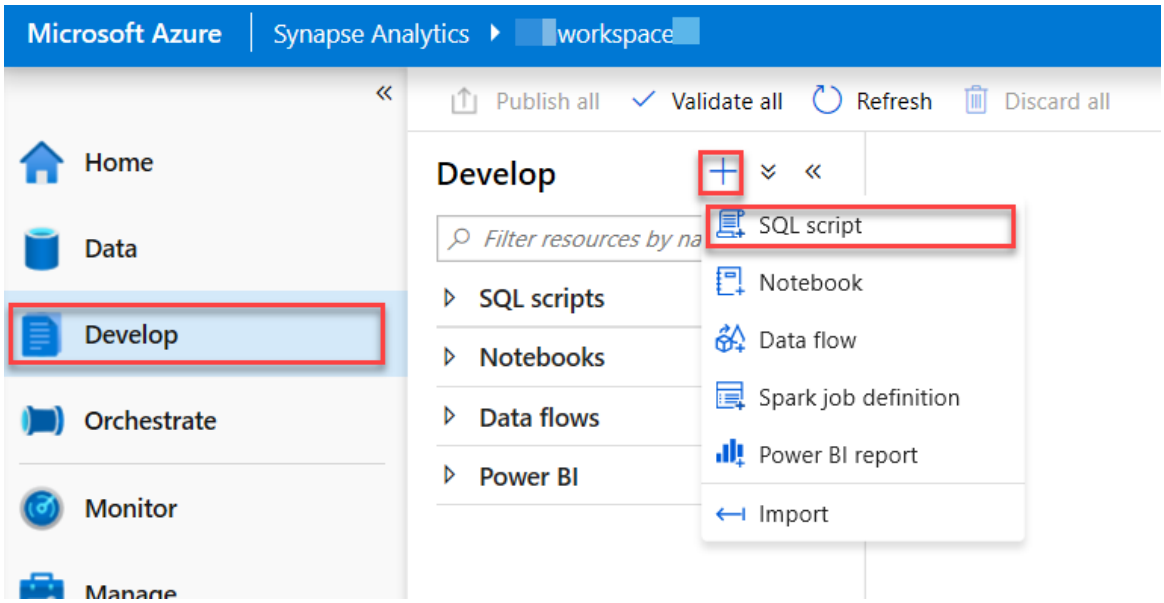
29. Verify the table has populated by creating a new query. Select the **Develop** item from the left menu, and in the **Develop** blade, expand the **+** button, and select **SQL script**. In the query window, be sure to connect to the SQL Pool database (`SQLPool01`), then paste and run the following query. When complete, select the **Discard all** button from the top toolbar.

```
select count(TransactionId) from wwi_mcw.SaleSmall;
```

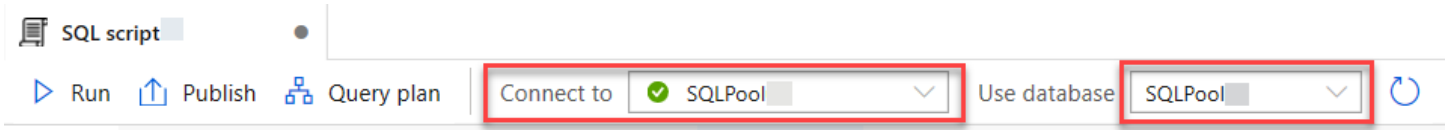
Task 3: Create the customer information table

Over the past 5 years, Wide World Importers has amassed over 3 billion rows of sales data. With this quantity of data, the customer information lookup table is estimated to have over 100 million rows but will consume less than 1.5 GB of storage. While we will be using only a subset of this data for the lab, we will design the table for the production environment. Using the guidance outlined in the Exercising description, we can ascertain that we will need a **Clustered Columnstore** table with a **Replicated** table distribution to hold customer data.

1. Expand the left menu and select the **Develop** item. From the **Develop** blade, expand the **+** button and select the **SQL script** item.



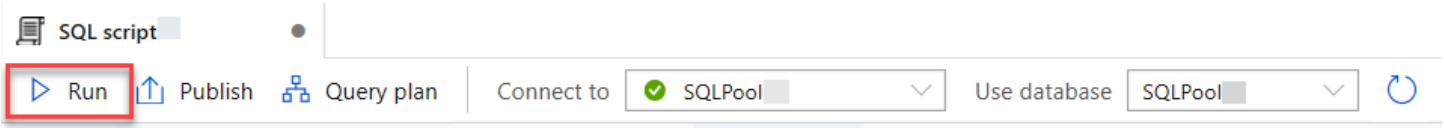
2. In the query tab toolbar menu, ensure you connect to your SQL Pool, `SQLPool01`.



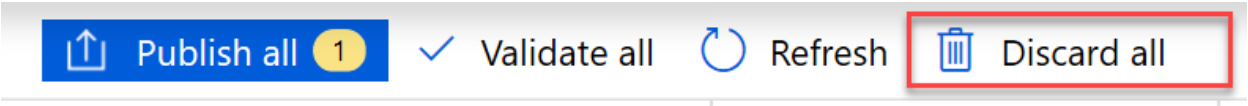
3. In the query window, copy and paste the following query to create the customer information table. Then select the **Run** button in the query tab toolbar.

```
CREATE TABLE [wwi_mcw].[CustomerInfo]
 (
   [UserName] [nvarchar](100)  NULL,
   [Gender] [nvarchar](10)  NULL,
   [Phone] [nvarchar](50)  NULL,
```

```
  [Email] [nvarchar](150)  NULL,
  [CreditCard] [nvarchar](21)  NULL
)
WITH
(
  DISTRIBUTION = REPLICATE,
  CLUSTERED COLUMNSTORE INDEX
)
GO
```



4. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

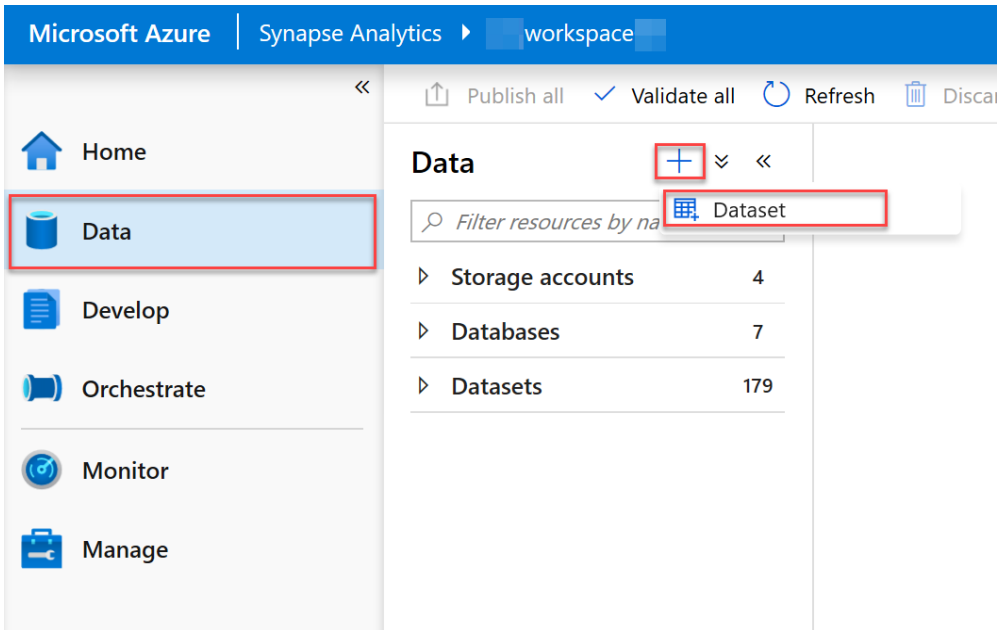


Task 4: Populate the customer information table

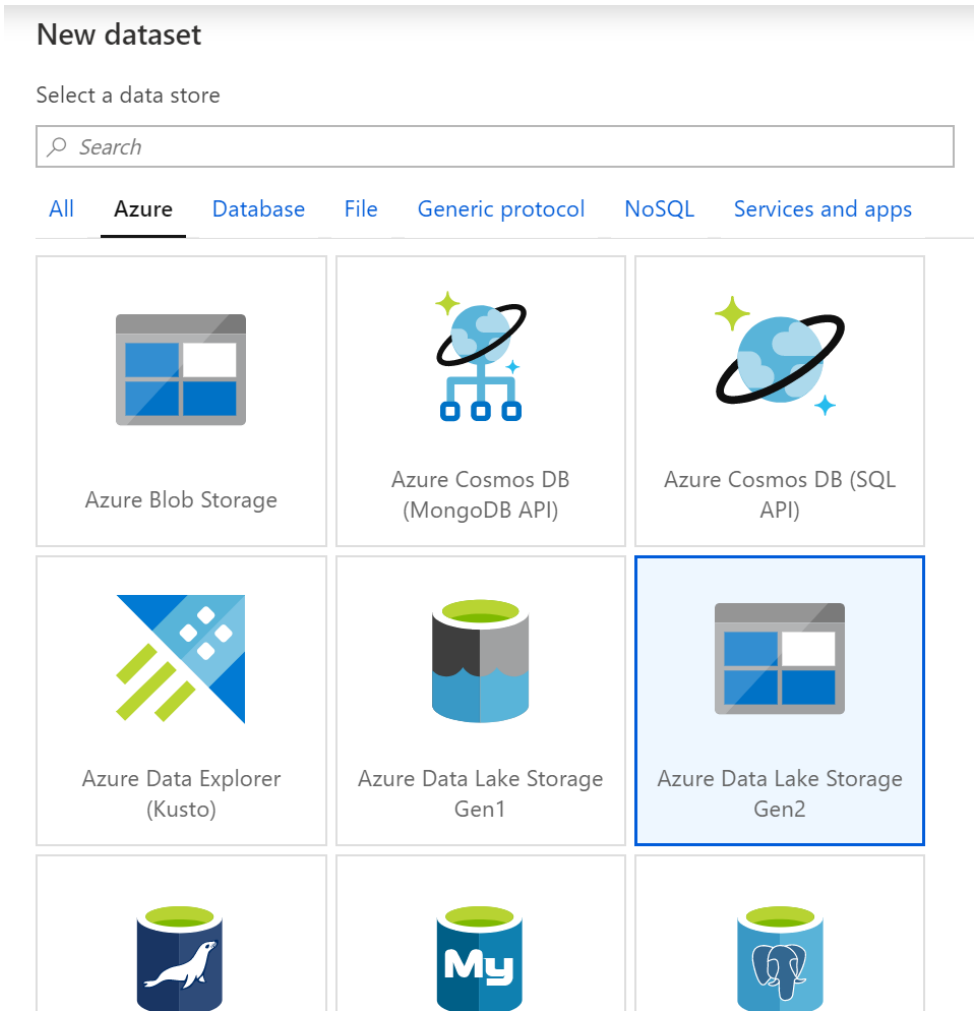
1. The data that we will be retrieving to populate the customer information table is currently stored in CSV format in the data lake (Azure Data Lake Storage Gen2 account). The storage account that possesses this data has already been added as a linked service in Azure Synapse Analytics when the environment was provisioned.

2. Similar to the previous step, the destination for our data has also been added as a linked service. In this case, the destination for our data is our SQL Pool, `SQLPool01`. Repeat the previous step, this time filtering with the term **sqlpool** to verify the existence of the linked service.

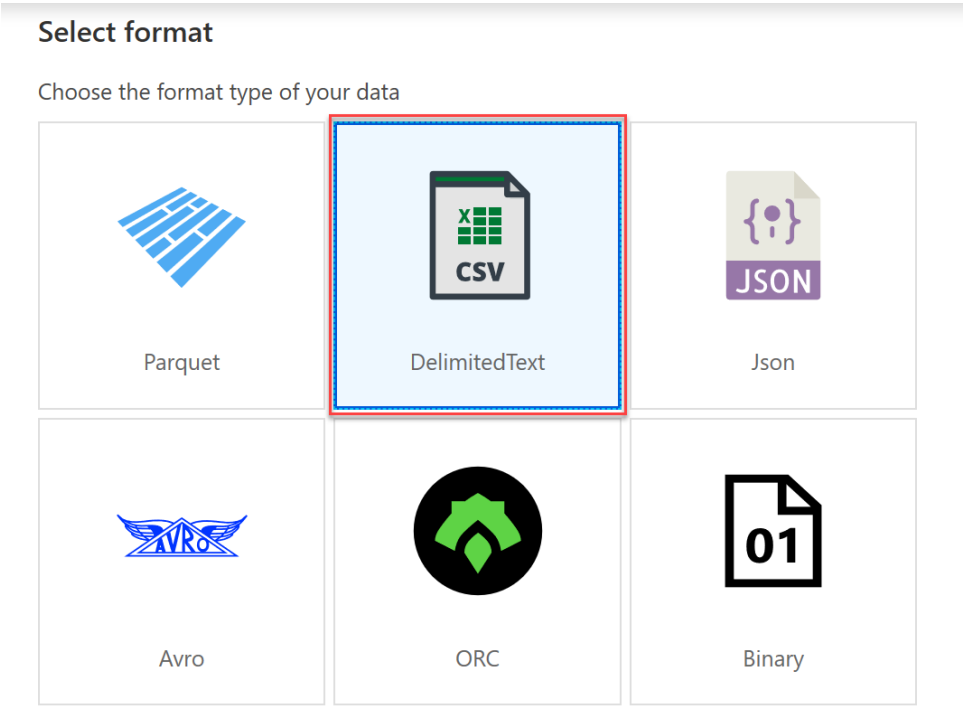
3. The next thing that we will need to do is define a source dataset that will represent the information that we are copying over. This dataset will reference the CSV file containing customer information. From the left menu, select **Data**. From the **Data** blade, expand the **+** button and select **Dataset**.



4. On the **New dataset** blade, with the **Azure** tab selected, choose the **Azure Data Lake Gen2** item. Select **Continue**.

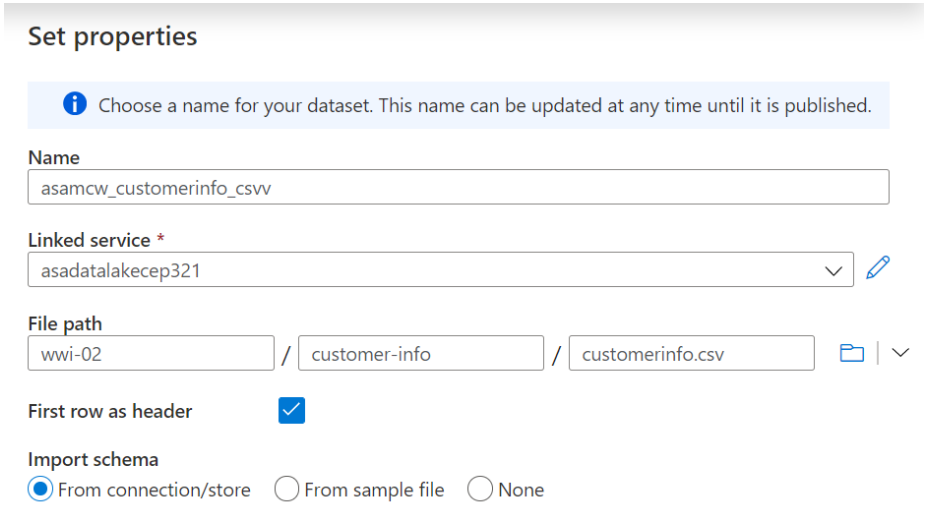


5. On the **Select format** blade, select **CSV Delimited Text**. Select **Continue**.



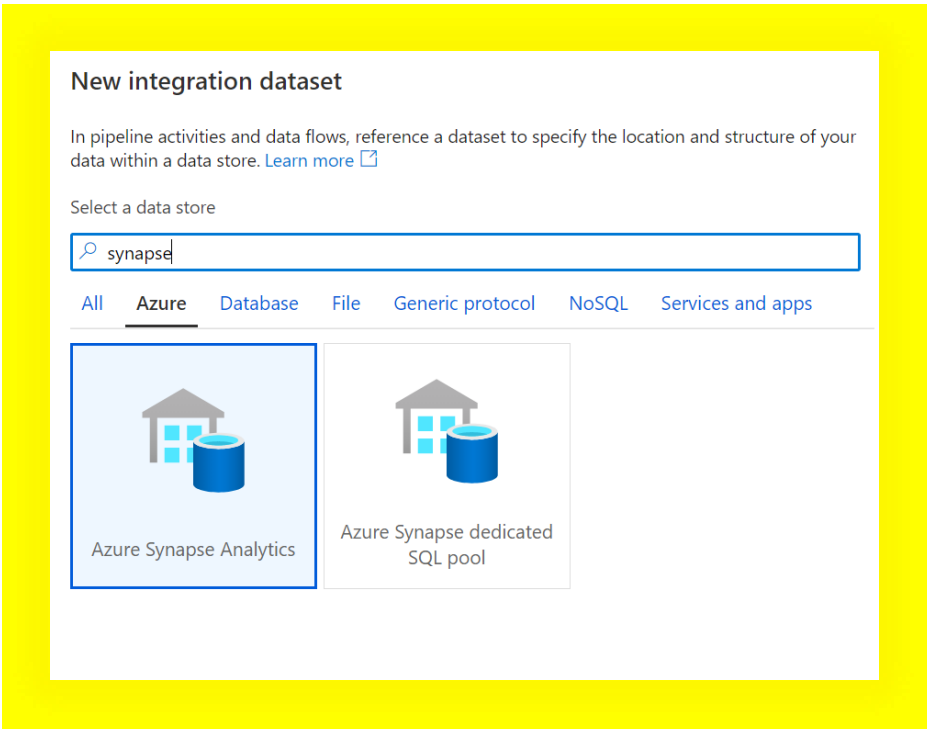
6. On the **Set properties** blade, set the fields to the following values, then select **OK**.

Field	Value
Name	Enter **asamcw_customerinfo_csv**.
Linked service	**asadatalake{SUFFIX}**
File Path - Container	Enter **wwi-02**.
File Path - Directory	Enter **customer-info**.
File Path - File	Enter **customerinfo.csv**.
First row as header	Checked
Import schema	Select **From connection/store**.



7. Now we will need to define the destination dataset for our data. In this case we will be storing customer information data in our SQL Pool. On the **Data** blade, expand the **+** button just as you did in **Step 3**.

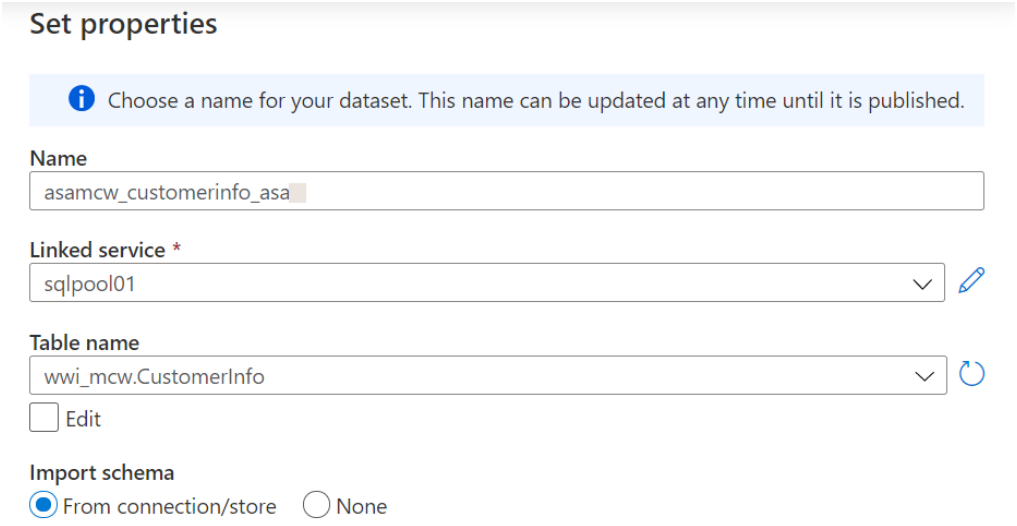
8. On the **New dataset** blade, with the **Azure** tab selected, enter **synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue**.



9. On the **Set properties** blade, set the field values to the following, then select **OK**.

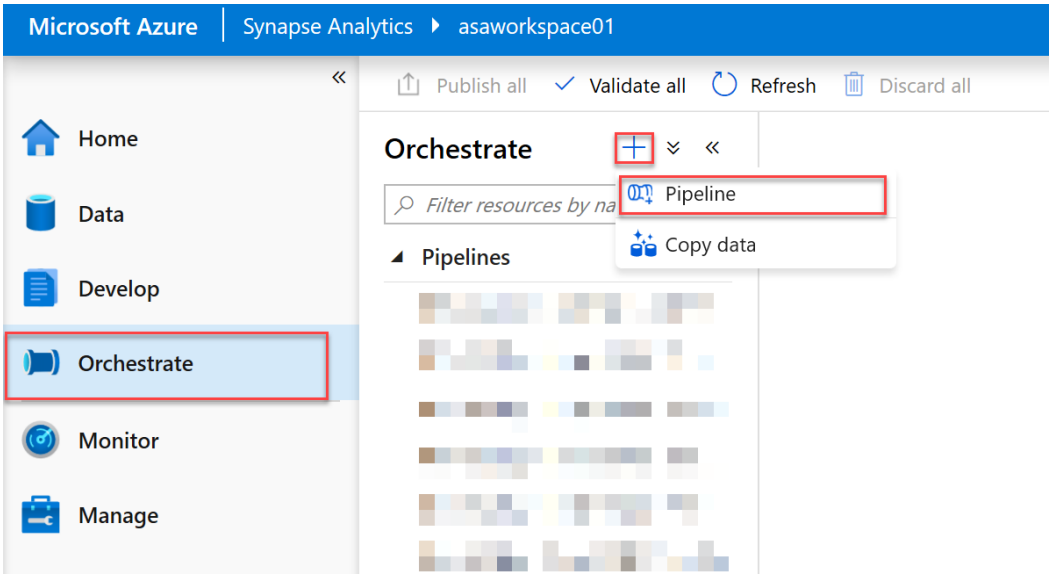
Field	Value
Name	Enter **asamcw_customerinfo_asa**.

Field	Value
Linked service	**SQLPool01**
Table name	**wwi_mcw.CustomerInfo**
Import schema	**From connection/store**

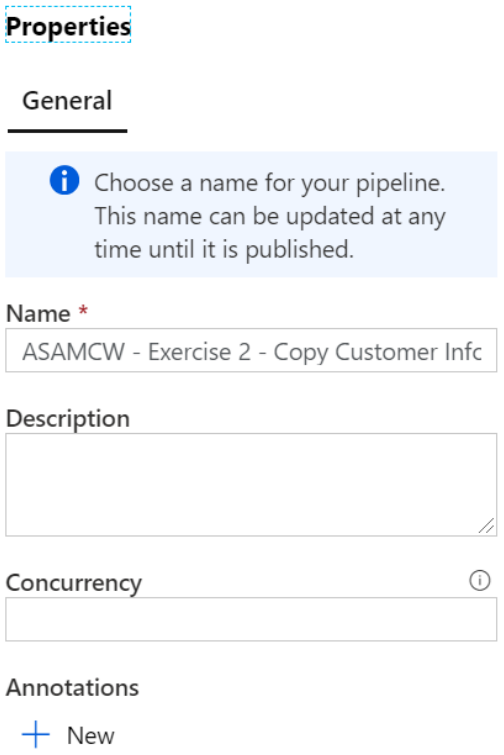


10. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to commit the changes.

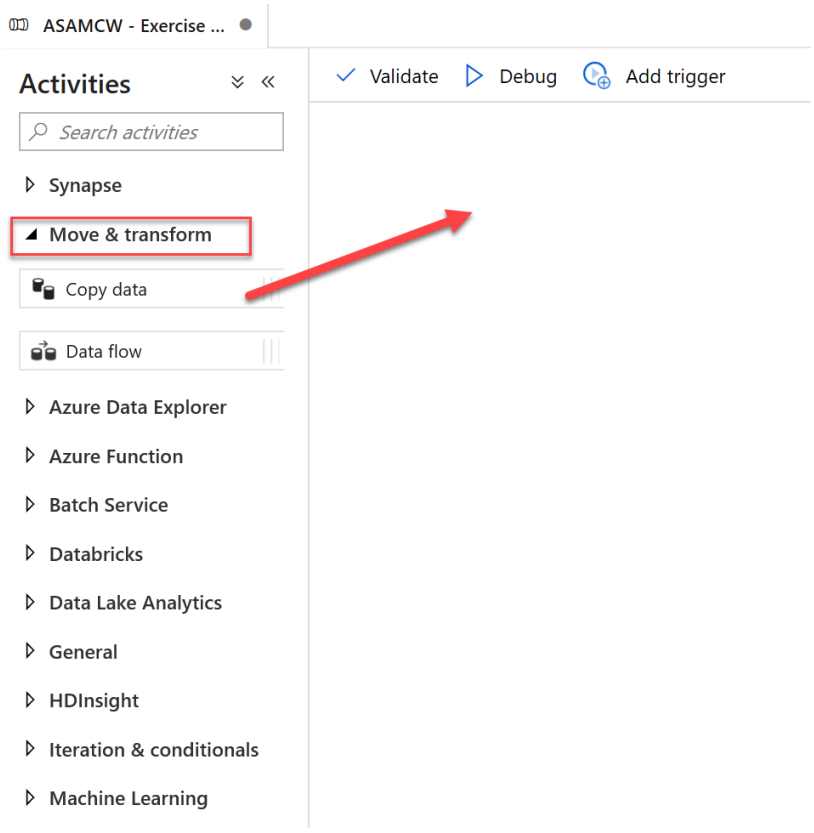
11. Next, we will define a pipeline to populate data into the CustomerInfo table. From the left menu, select **Orchestrate**. From the Orchestrate blade, select the **+** button and select the **Pipeline** item.



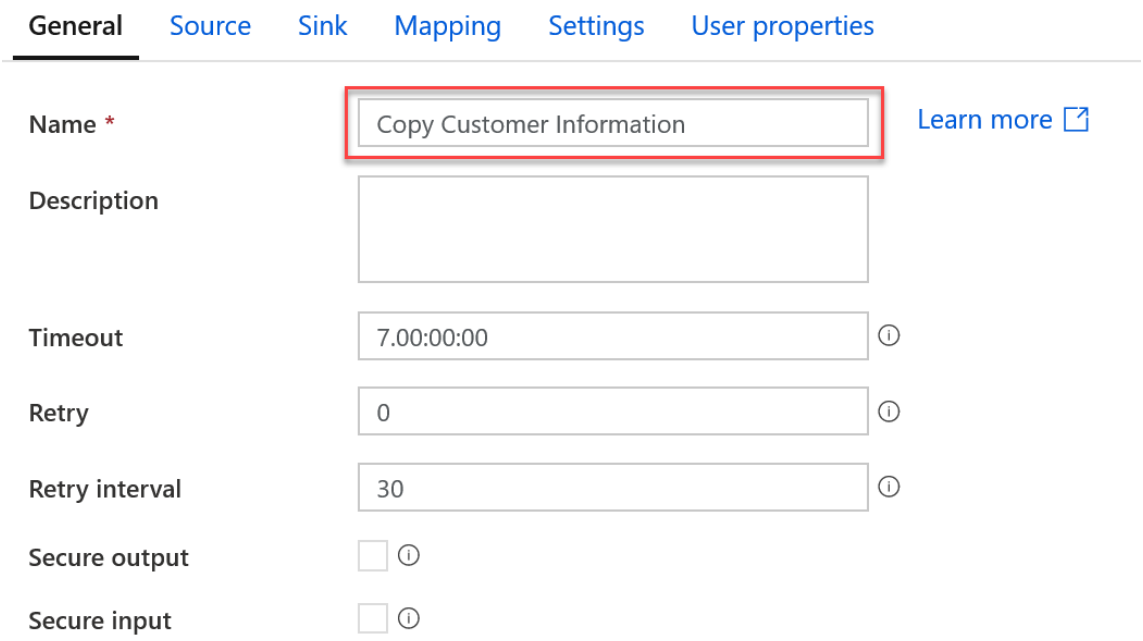
12. In the **Properties** blade, enter **ASAMCW - Exercise 2 - Copy Customer Information** in the **Name** field.



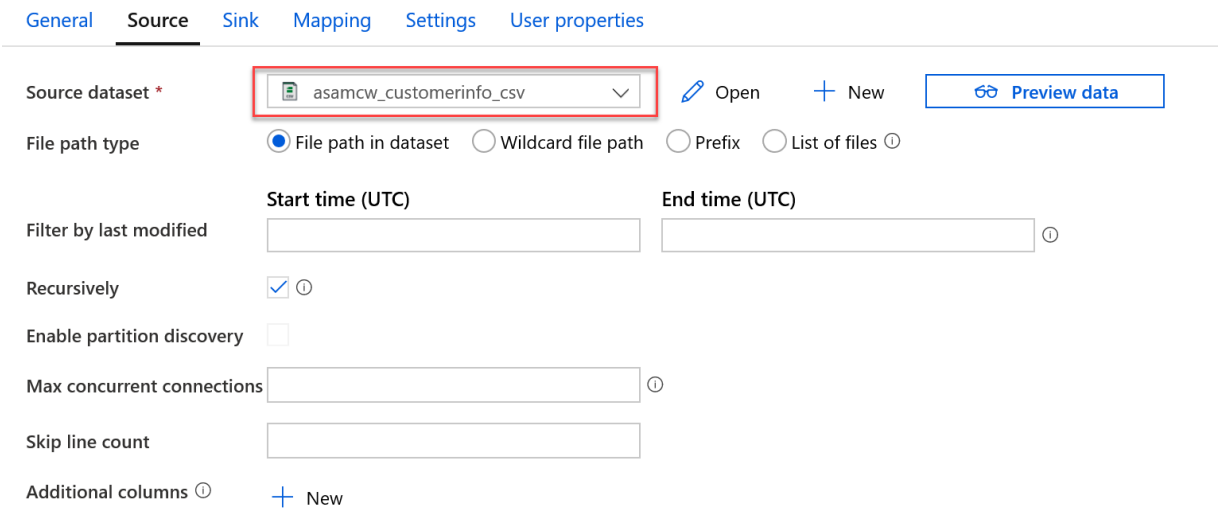
13. In the **Activities** menu, expand the **Move & transform** item. Drag an instance of the **Copy data** activity to the design surface of the pipeline.



14. Select the **Copy data** activity on the pipeline design surface. In the bottom pane, on the **General** tab, enter **Copy Customer Information Data** in the **Name** field.

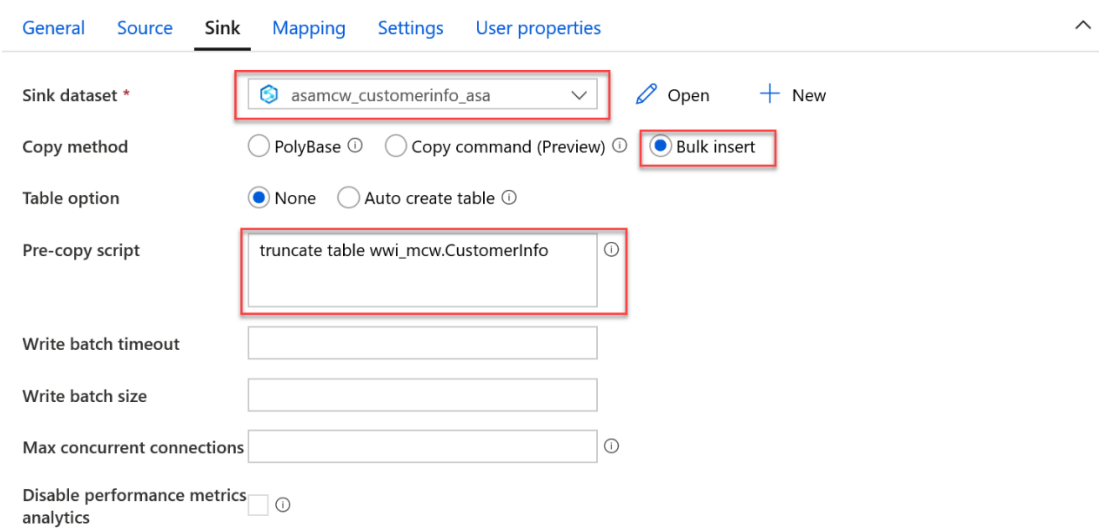


15. Select the **Source** tab in the bottom pane. In the **Source dataset** field, select **asamcw_customerinfo_csv**.

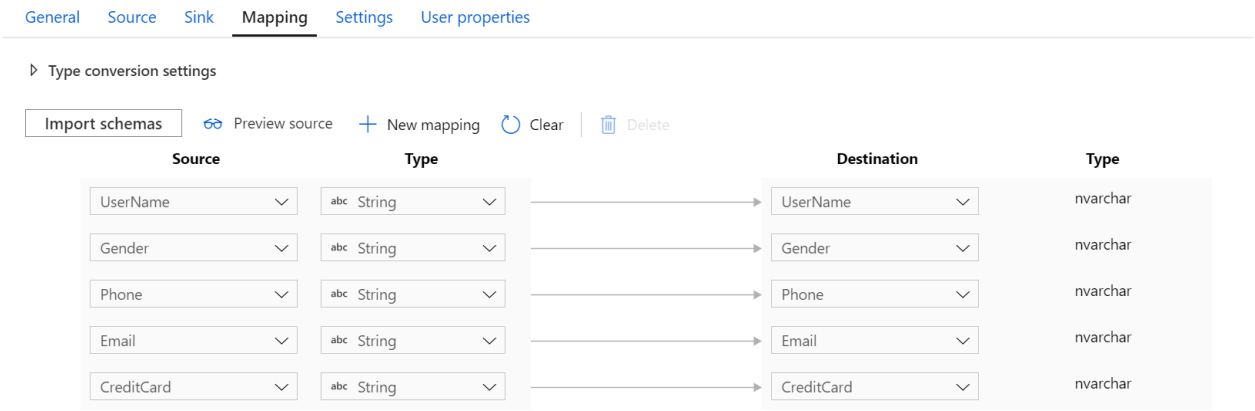


16. Select the **Sink** tab in the bottom pane. In the **Sink dataset** field, select **asamcw_customerinfo_asa**, for the **Copy method** field, select **Bulk insert**, and for **Pre-copy script** enter:

```
truncate table wwi_mcw.CustomerInfo
```



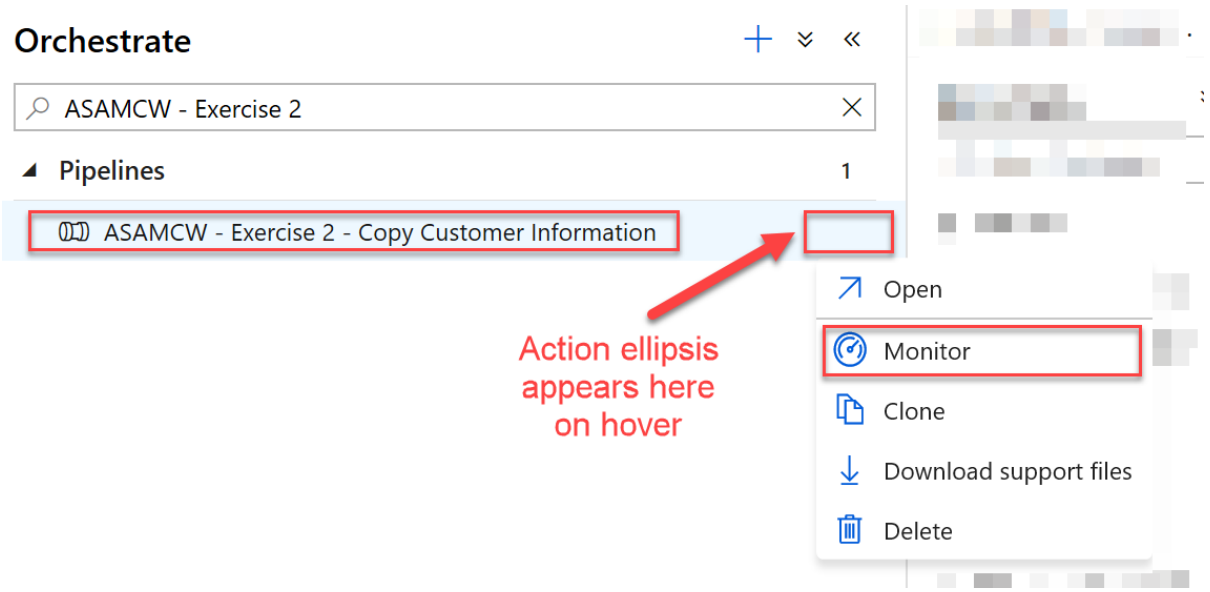
17. Select the **Mapping** tab in the bottom pane. Select the **Import schemas** button. You will notice that Azure Synapse Analytics automated the mapping for us since the field names and types match.



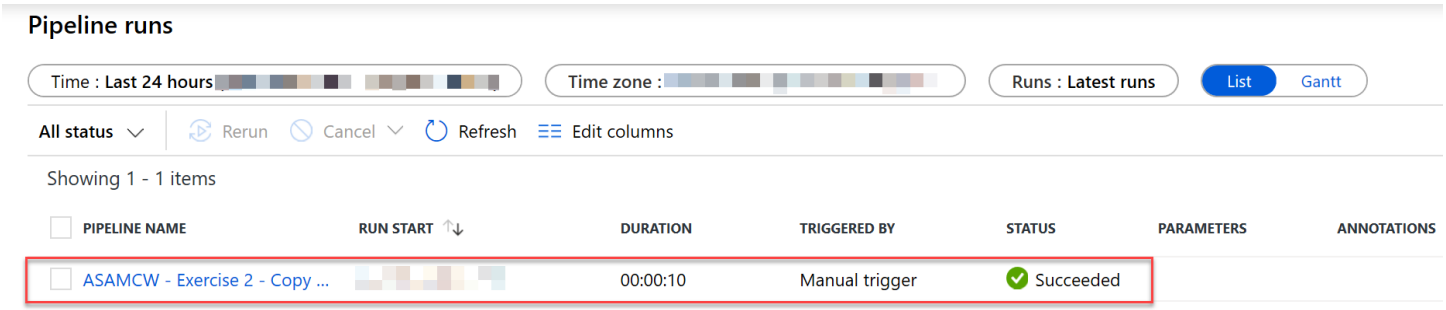
18. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to commit the changes.

19. Once published, expand the **Add trigger** item on the pipeline designer toolbar, and select **Trigger now**. In the **Pipeline run** blade, select **OK** to proceed with the latest published configuration. You will see notification toast windows indicating the pipeline is running and when it has completed.

20. View the status of the completed run by locating the **ASAMCW - Exercise 2 - Copy Customer Information** pipeline in the Orchestrate blade. Expand the actions menu, and select the **Monitor** item.



21. You should see a successful run of the pipeline we created in the **Pipeline runs** table.



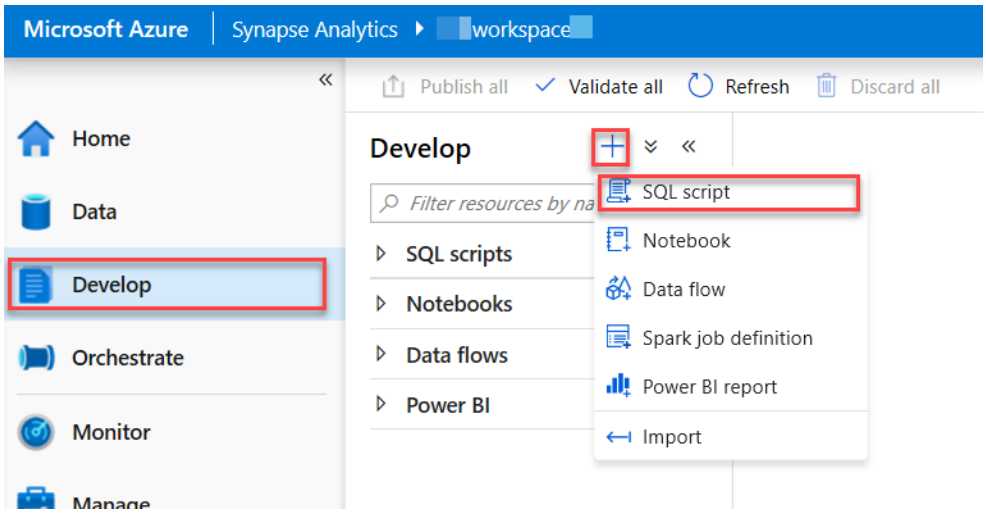
22. Verify the table has populated by creating a new query. Remember from **Task 1**, select the **Develop** item from the left menu, and in the **Develop** blade, expand the **+** button, and select **SQL script**. In the query window, be sure to connect to the SQL Pool database (`SQLPool01`), then paste and run the following query. When complete, select the **Discard all** button from the top toolbar.

```
select * from wwi_mcw.CustomerInfo;
```

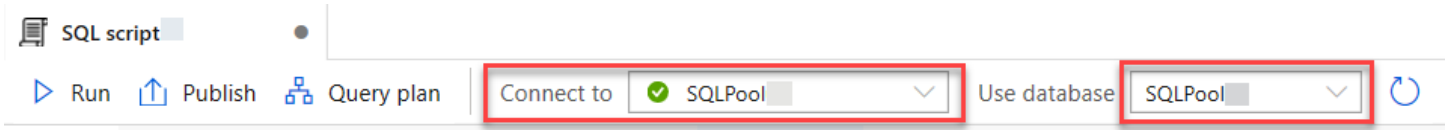
Task 5: Create the campaign analytics table

The campaign analytics table will be queried primarily for dashboard and KPI purposes. Performance is a large factor in the design of this table, and as such we can ascertain that we will need a **Clustered Columnstore** table with a **Hash** table distribution based on the **Region** field which will fairly evenly distribute the data.

1. Expand the left menu and select the **Develop** item. From the **Develop** blade, expand the **+** button and select the **SQL script** item.

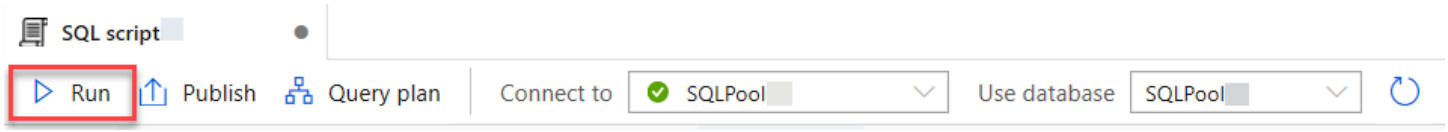


2. In the query tab toolbar menu, ensure you connect to your SQL Pool, `SQLPool01`.

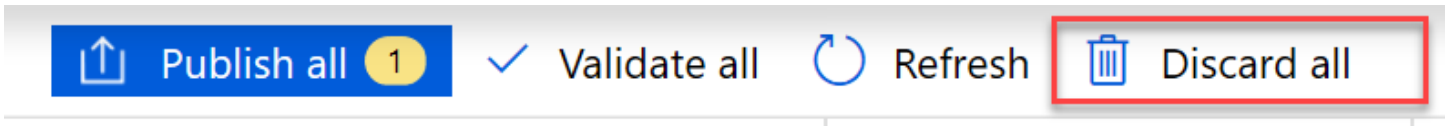


3. In the query window, copy and paste the following query to create the campaign analytics table. Then select the **Run** button in the query tab toolbar.

```
CREATE TABLE [wwi_mcw].[CampaignAnalytics]
(
    [Region] [nvarchar](50)  NULL,
    [Country] [nvarchar](30)  NOT NULL,
    [ProductCategory] [nvarchar](50)  NOT NULL,
    [CampaignName] [nvarchar](500)  NOT NULL,
    [Analyst] [nvarchar](25) NULL,
    [Revenue] [decimal](10,2)  NULL,
    [RevenueTarget] [decimal](10,2)  NULL,
    [City] [nvarchar](50)  NULL,
    [State] [nvarchar](25)  NULL
)
WITH
(
    DISTRIBUTION = HASH ( [Region] ),
    CLUSTERED COLUMNSTORE INDEX
);
```



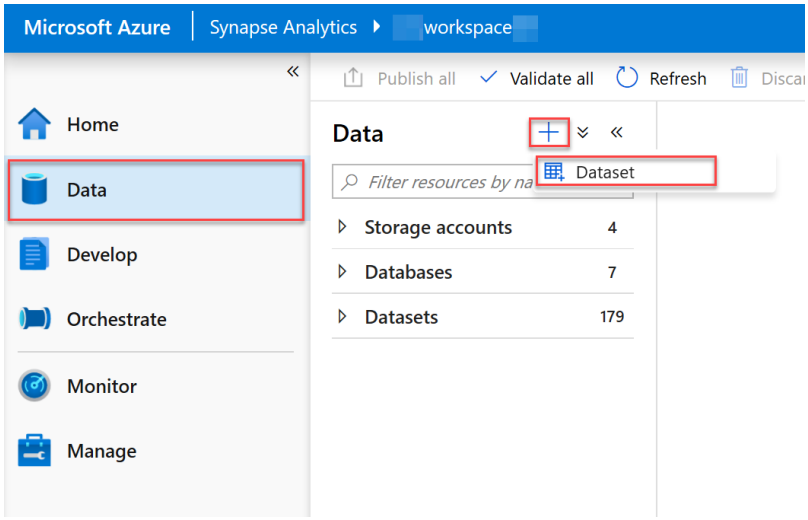
4. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



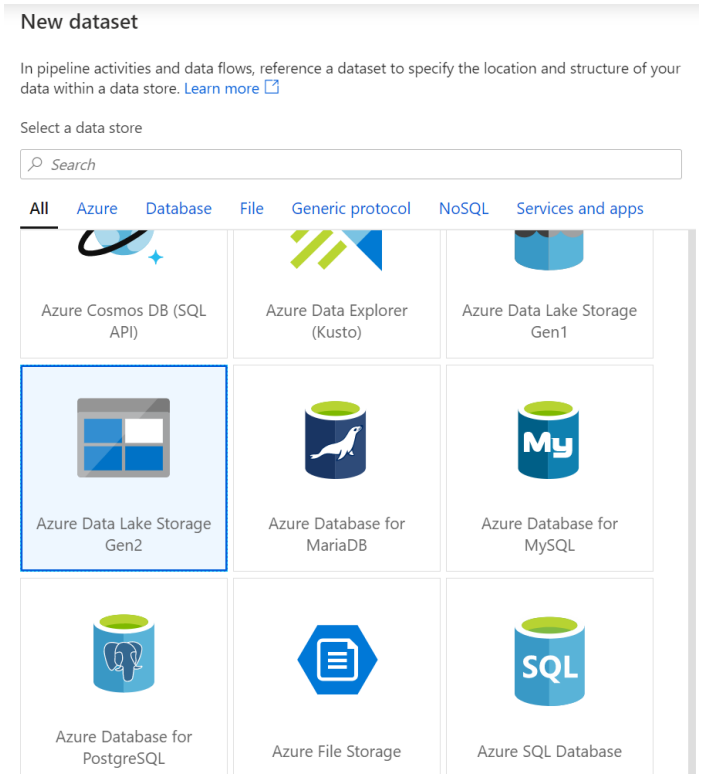
Task 6: Populate the campaign analytics table

Similar to the customer information table, we will also be populating the campaign analytics table via a CSV file located in the data lake. This will require source and sink datasets to point to the CSV file in storage and the campaign analytics table that you just created in the SQL Pool. The source CSV file that was received is poorly formatted - we will need to add data transformations to make adjustments to this data before it is imported into the data warehouse.

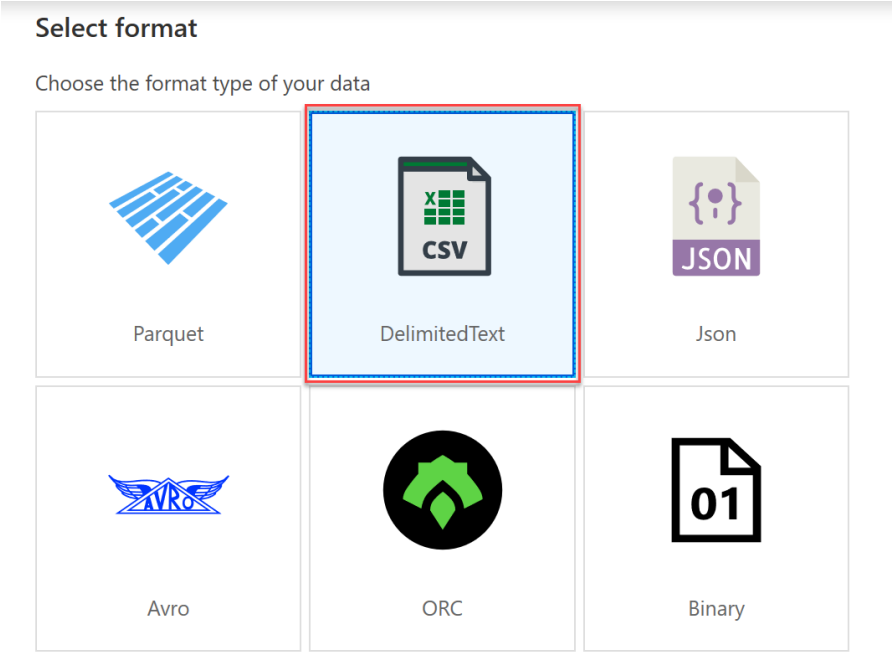
1. The source dataset will reference the CSV file containing campaign analytics information. From the left menu, select **Data**. From the **Data** blade, expand the **+** button and select **Dataset**.



2. On the **New dataset** blade, with the **All** tab selected, choose the **Azure Data Lake Storage Gen2** item. Select **Continue**.

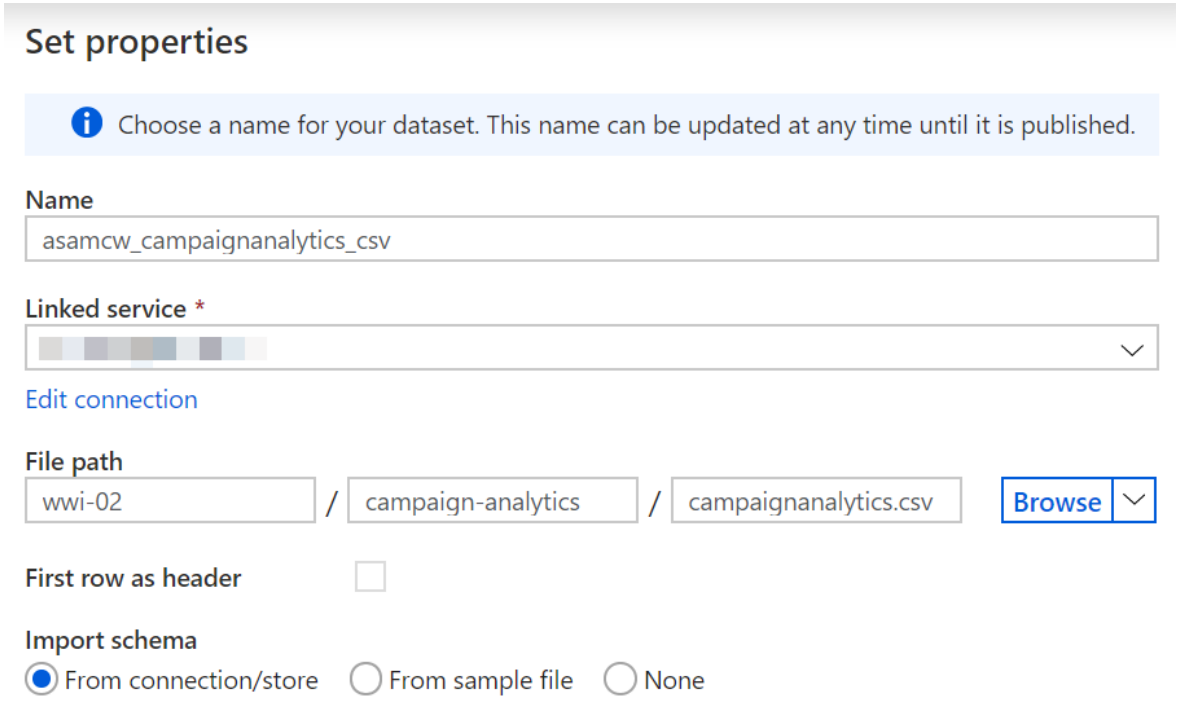


3. On the **Select format** blade, select **CSV Delimited Text**. Select **Continue**.



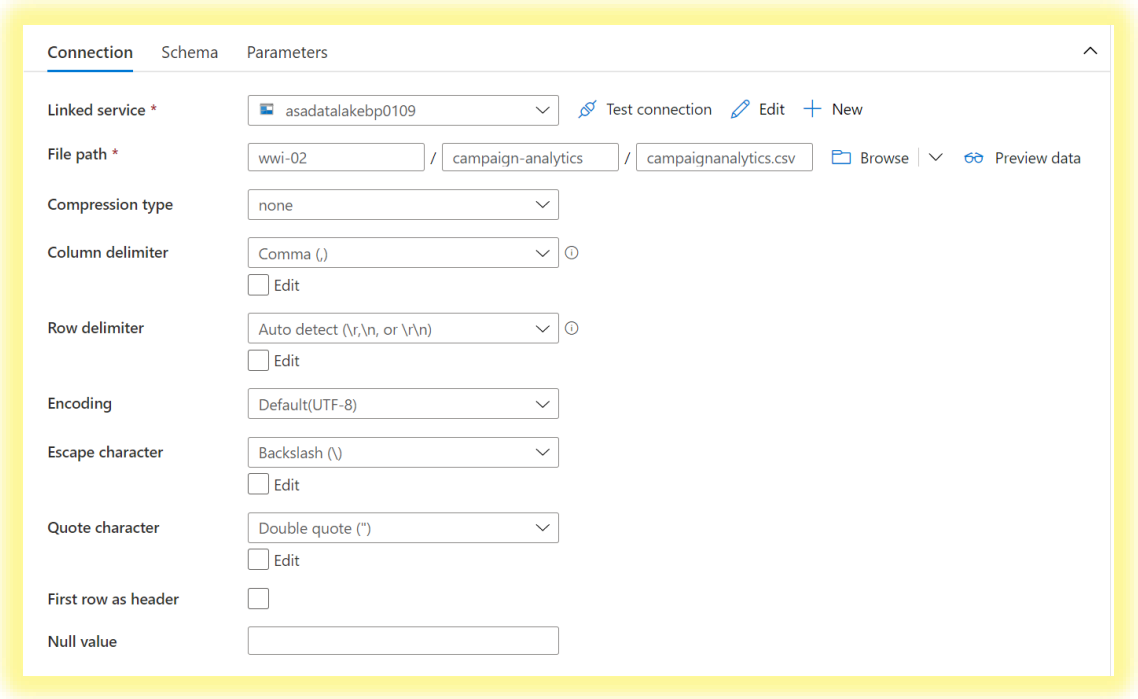
4. On the **Set properties** blade, set the fields to the following values, then select **OK**. You may choose to preview the data which will show a sample of the CSV file. Notice that since we are not setting the first row as the header, the header columns appear as the first row. Also, notice that the city and state values do not appear. This is because of the mismatch in the number of columns in the header row compared to the rest of the file. Soon, we will exclude the first row as we transform the data.

Field	Value
Name	Enter **asamcw_campaignanalytics_csv**
Linked service	Select **asadatalake{SUFFIX}**.
File Path - Container	Enter **wwi-02**
File Path - Directory	Enter **campaign-analytics**
File Path - File	Enter **campaignanalytics.csv**
First row as header	Unchecked
Import schema	Select **From connection/store**



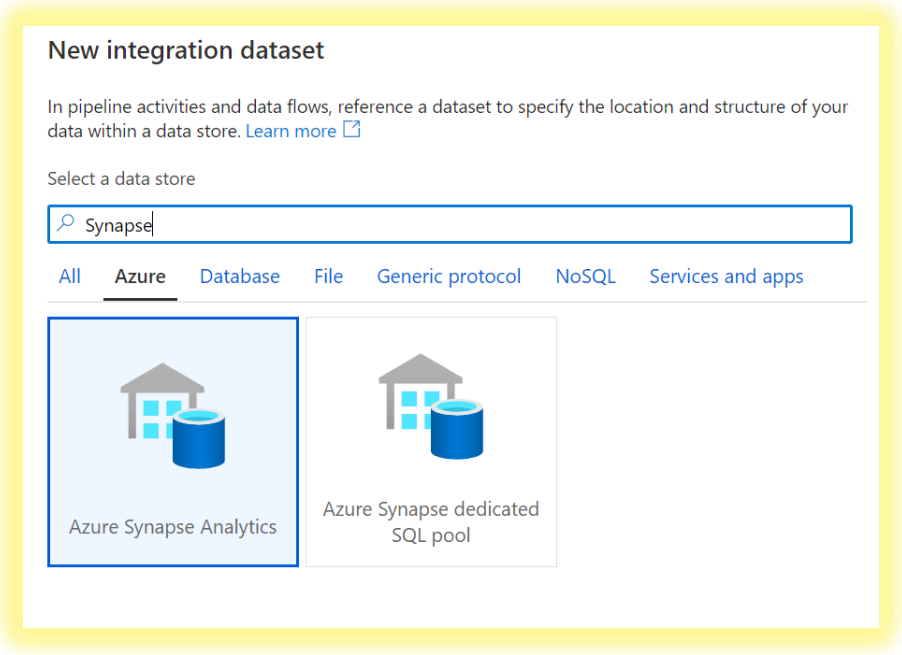
5. In the center pane, on the **Connection** tab of **asamcw_campainganalytics_csv** dataset, ensure the following field values are set:

Field	Value
Escape Character	**Backslash (\)**
Quote Character	**Double quote (")**



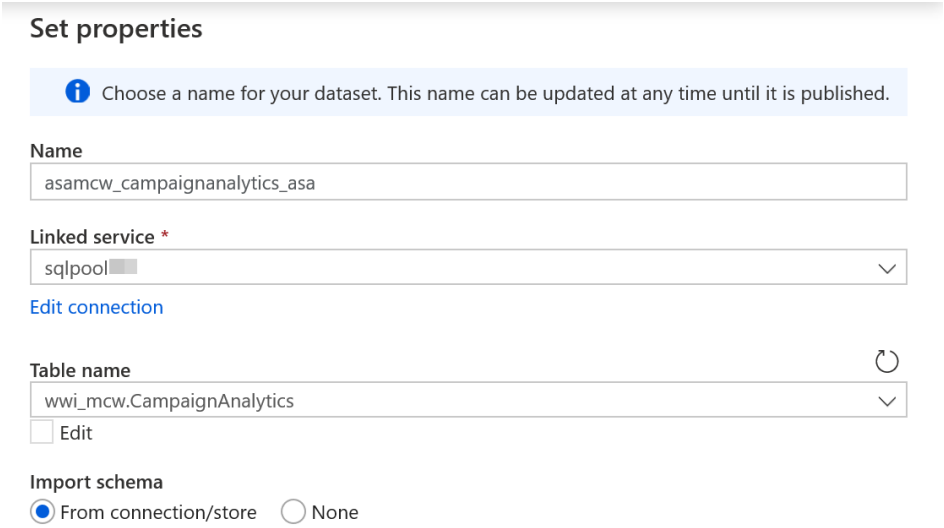
6. Now we will need to define the destination dataset for our data. In this case we will be storing campaign analytics data in our SQL Pool. On the **Data** blade, expand the **+** button and select **Dataset**.

7. On the **New dataset** blade, with the **Azure** tab selected, enter **synapse** as a search term and select the **Azure Synapse Analytics (formerly SQL DW)** item. Select **Continue**.



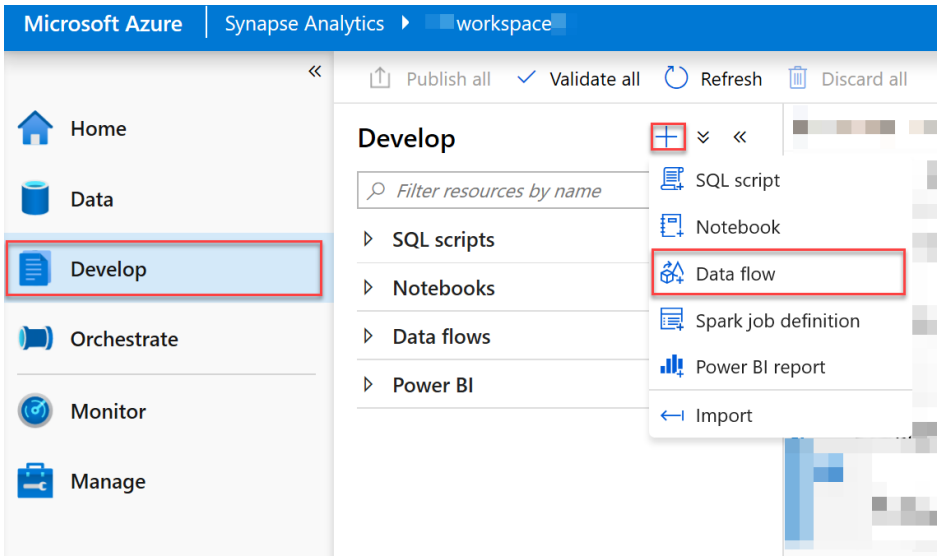
8. On the **Set properties** blade, set the field values to the following, then select **OK**.

Field	Value
Name	Enter **asamcw_campaignanalytics_asa**.
Linked service	**SQLPool01**
Table name	**wwi_mcw.CampaignAnalytics**
Import schema	Select **From connection/store**.

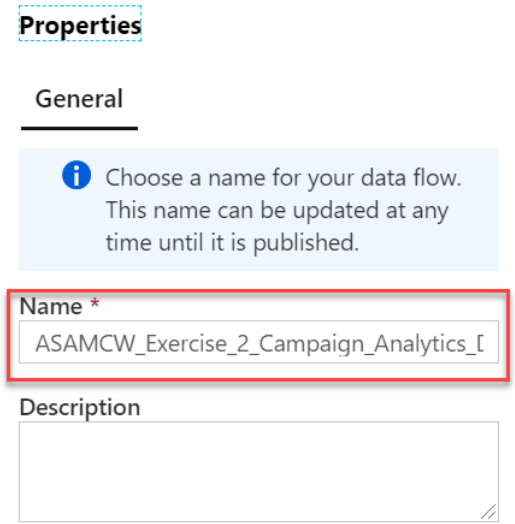


9. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to commit the changes.

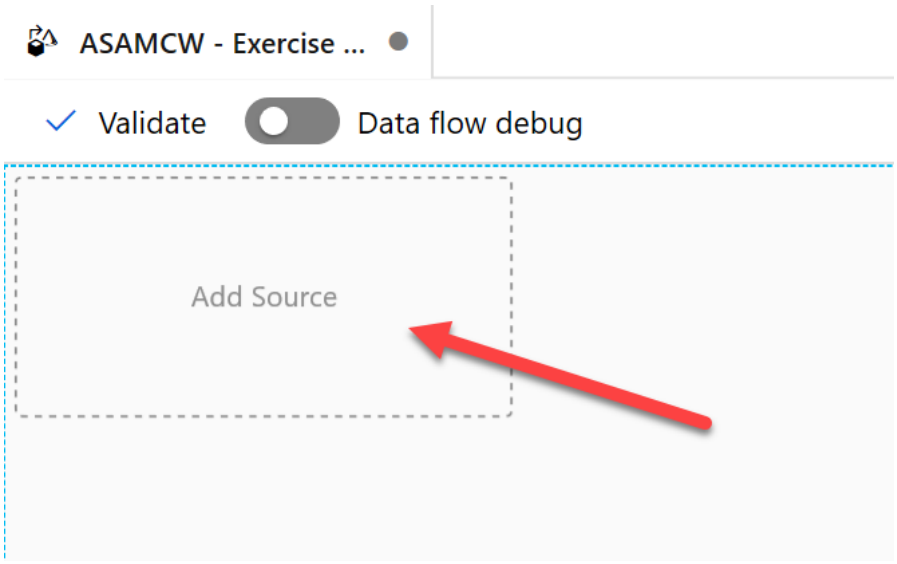
10. Since our source data is malformed and does not contain an Analyst column, we will need to create a data flow to transform the source data. A data flow allows you to graphically define dataset filters and transformations without writing code. These data flows can be leveraged as an activity in an orchestration pipeline. Create a new data flow, start by selecting **Develop** from the left menu, and in the **Develop** blade, expand the **+** button and select **Data flow**.



11. In the **Properties** blade name the data flow by entering **ASAMCW_Exercise_2_Campaign_Analytics_Data** in the **Name** field.

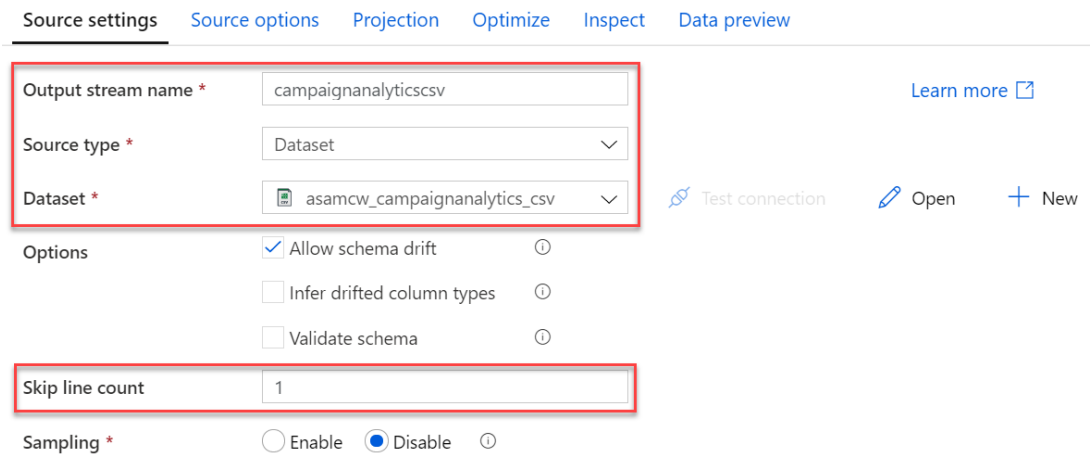


12. In the data flow designer window, select the **Add Source** box.



13. Under **Source settings**, configure the following:

Field	Value
Output stream name	Enter **campaignanalyticscsv**.
Source type	**Dataset**
Dataset	**asamcw_campaignanalytics_csv**
Skip line count	Enter **1**.



14. When you create data flows, certain features are enabled by turning on debug, such as previewing data and importing a schema (projection). Due to the amount of time it takes to enable this option, as well as environmental constraints of the lab environment, we will bypass these features. The data source has a schema we need to set. To do this, select **Script** from the right side of the dataflow designer toolbar menu.

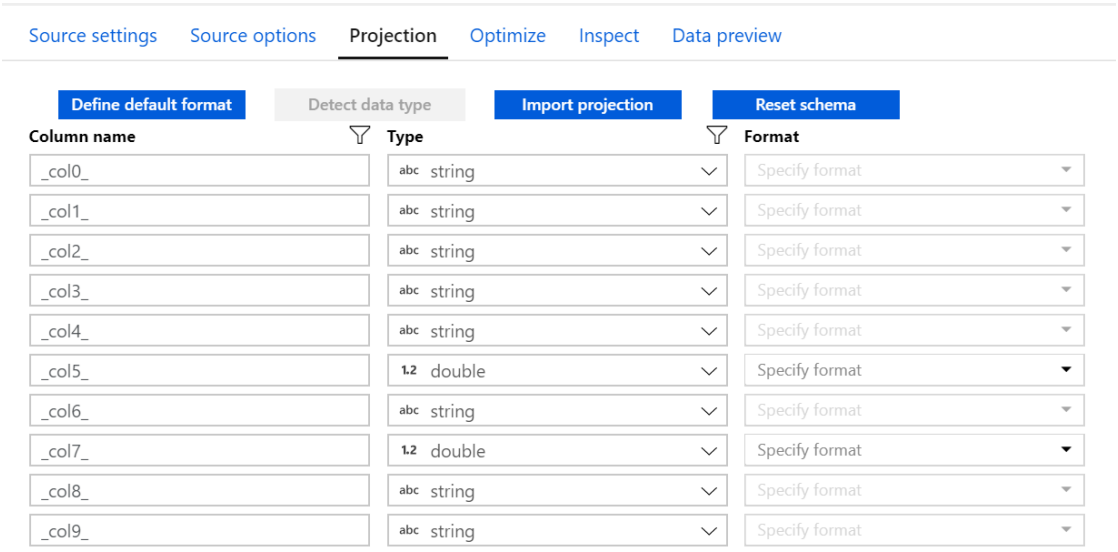
15. Replace the script with the following to provide the column mappings (`output`), then select **OK**:

```
source(output(
        {_col0_} as string,
        {_col1_} as string,
        {_col2_} as string,
        {_col3_} as string,
        {_col4_} as string,
        {_col5_} as double,
        {_col6_} as string,
        {_col7_} as double,
        {_col8_} as string,
        {_col9_} as string
```

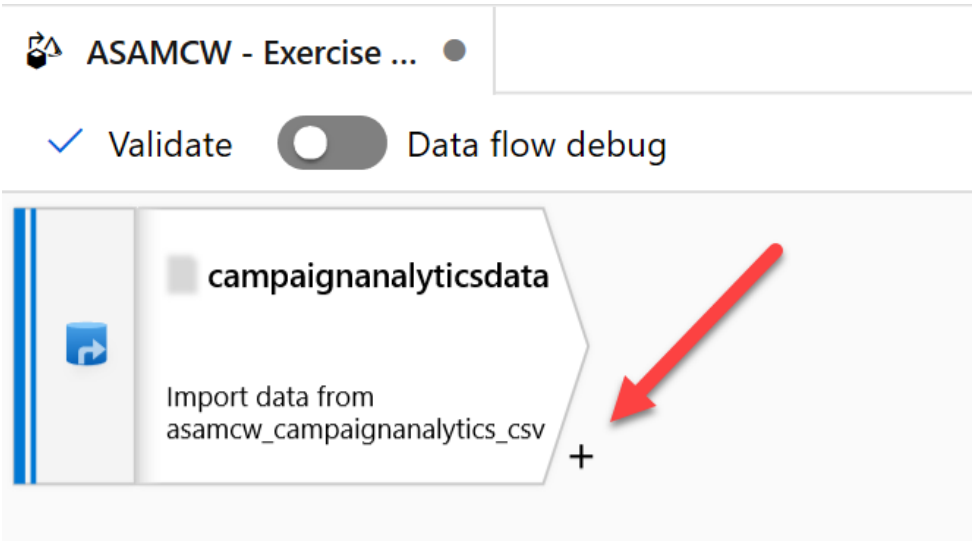
```
),
allowSchemaDrift: true,
validateSchema: false,
skipLines: 1) ~> campaignanalyticscsv
```

Note: We are changing the mappings as the source file was corrupted with the wrong headers.

16. Select the **campaignanalyticscsv** data source, then select **Projection**. The projection should display the following schema:



17. Select the **+** to the bottom right of the **campaignanalyticscsv** source, then select the **Select** schema modifier from the context menu.

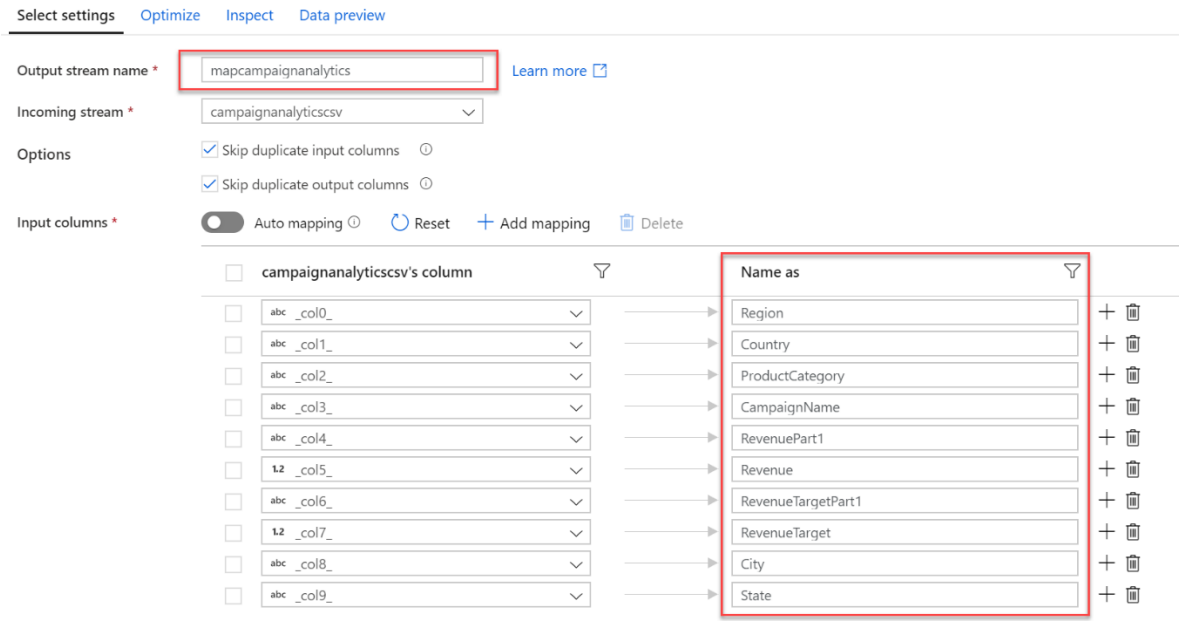


18. In the bottom pane, under **Select settings**, configure the following:

Field	Value
Output stream name	Enter **mapcampaignanalytics**.

19. For **Input Columns**, under the **Name as** column, enter the following list values in order:

    - Region
    - Country
    - ProductCategory
    - CampaignName
    - RevenuePart1
    - Revenue
    - RevenueTargetPart1
    - RevenueTarget
    - City
    - State



20. Select the **+** to the right of the **mapCampaignAnalytics** source, then select the **Derived Column** schema modifier from the context menu.

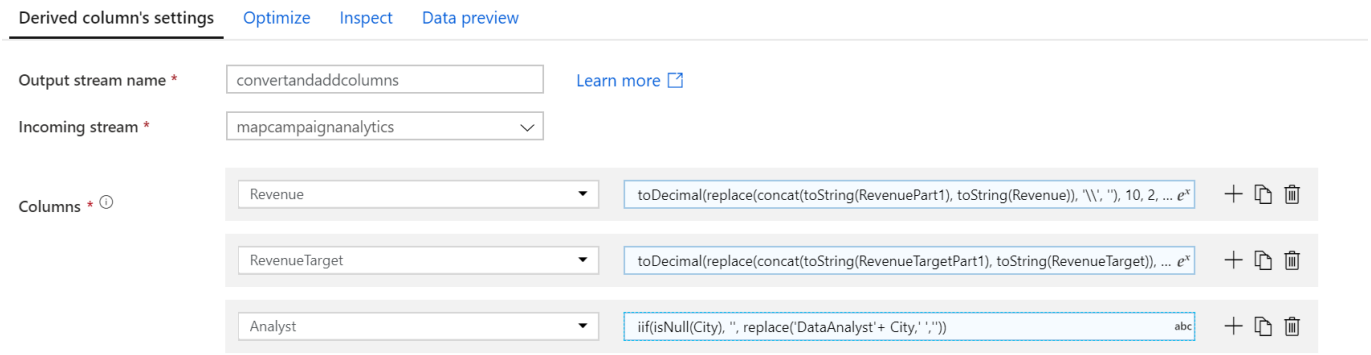
21. Under **Derived column's settings**, configure the following:

Field	Value
Output stream name	Enter **convertandaddcolumns**.

22. For **Columns**, add the following (Note you will need to type in the **Analyst** column):

Column	Expression	Description
Revenue	**toDecimal(replace(concat(toString(RevenuePart1), toString(Revenue)), '\\', ''), 10, 2, '$###,###.##')**	Concatenate the **RevenuePart1** and **Revenue** fields, replace the invalid \ character, then convert and format the data to a decimal type.

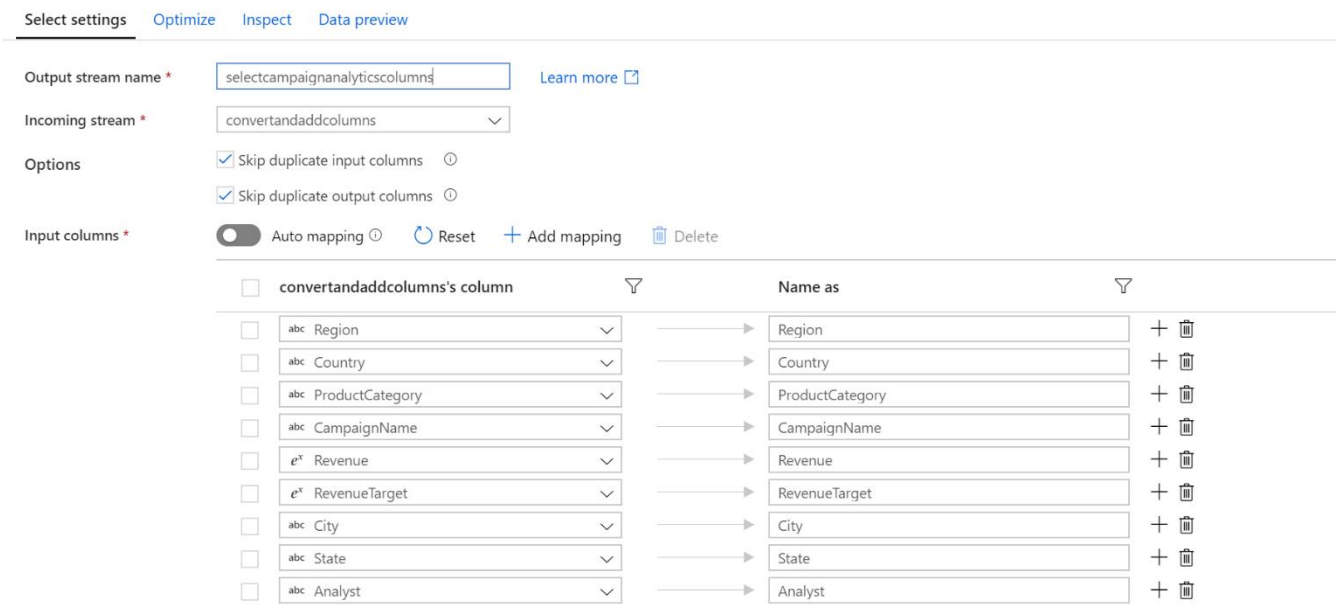
Column	Expression	Description
RevenueTarget	**toDecimal(replace(concat(toString(RevenueTargetPart1), toString(RevenueTarget)), '\\', ''), 10, 2, '$###,###.##')**	Concatenate the **RevenueTargetPart1** and **RevenueTarget** fields, replace the invalid \ character, then convert and format the data to a decimal type.
Analyst	**iif(isNull(City), '', replace('DataAnalyst'+ City,' ',''))**	If the city field is null, assign an empty string to the Analyst field, otherwise concatenate DataAnalyst to the City value, removing all spaces.



23. Select the **+** to the right of the **convertandaddcolumns** step, then select the **Select** schema modifier from the context menu.

24. Under **Select settings**, configure the following:

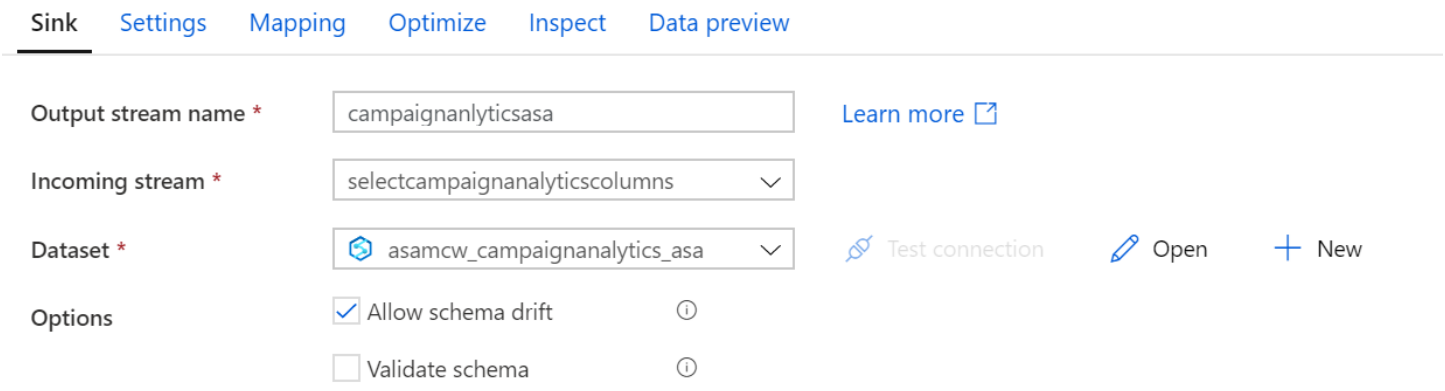
Field	Value
Output stream name	Enter **selectcampaignanalyticscolumns**.
Input columns	Delete the **RevenuePart1** and **RevenueTargetPart1** columns.



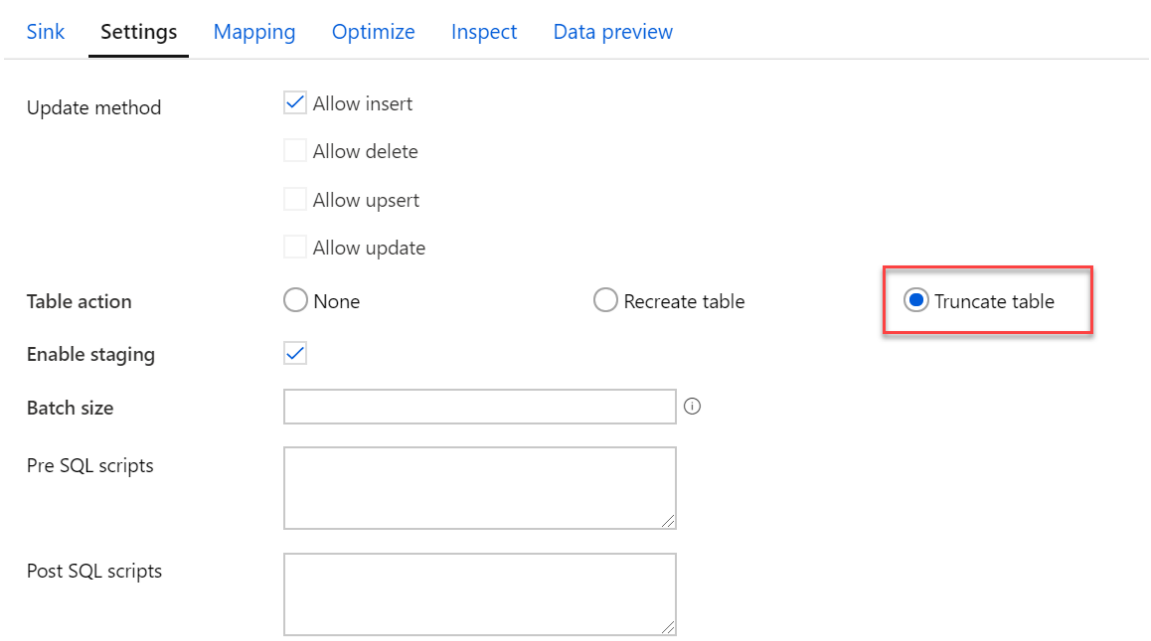
25. Select the **+** to the right of the **selectcampaignanalyticscolumns** step, then select the **Sink** destination from the context menu.

26. In the bottom pane, on the **Sink** tab, configure it as follows:

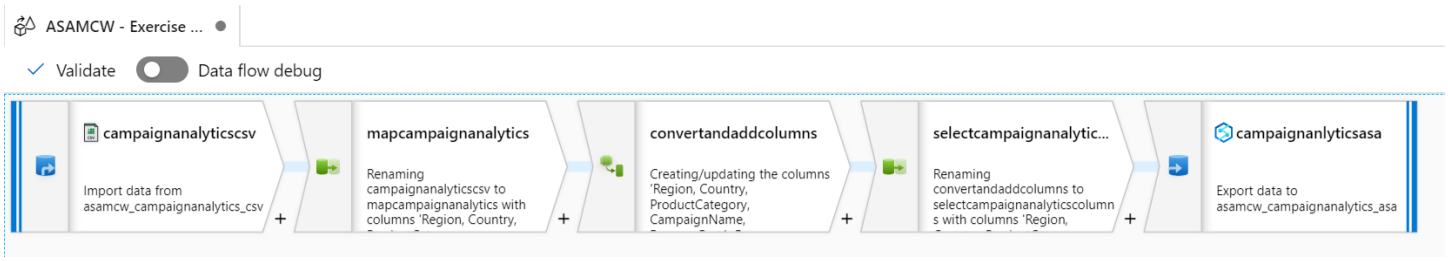
Field	Value
Output stream name	Enter **campaignanlyticsasa**.
Dataset	**asamcw_campaignanalytics_asa**



27. Select **Settings** tab, and for **Table action** select **Truncate table**.



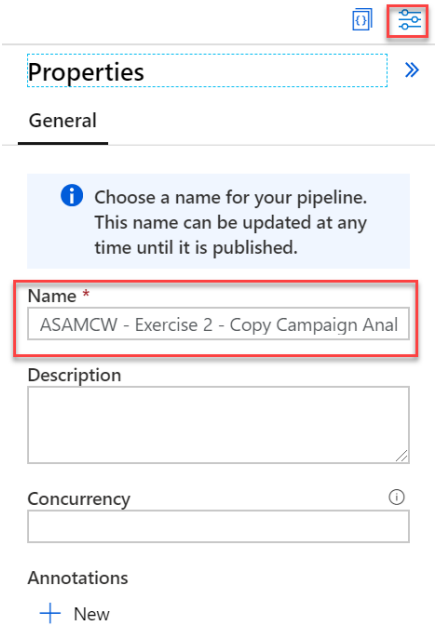
28. Your completed data flow should look similar to the following:



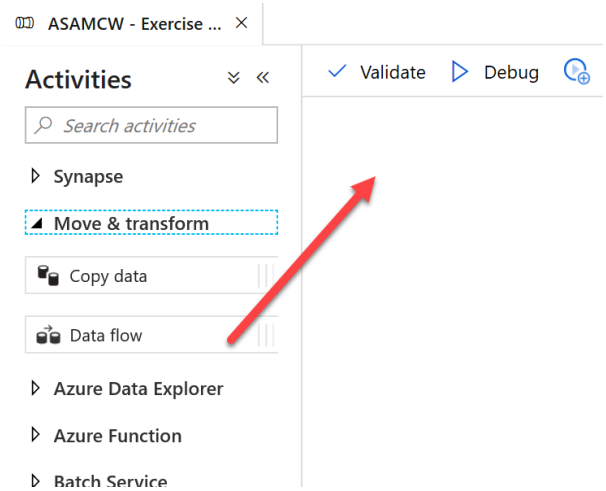
29. Select **Publish all** to save your new data flow.

30. Now that the data flow is published, we can use it in a pipeline. Create a new pipeline by selecting **Integrate** from the left menu, then in the **Integrate** blade, expand the **+** button and select **Pipeline**.

31. In the **Properties** pane on the right side of the pipeline designer. Enter **ASAMCW - Exercise 2 - Copy Campaign Analytics Data** in the **Name** field.



32. From the **Activities** menu, expand the **Move & transform** section and drag an instance of **Data flow** to the design surface of the pipeline.

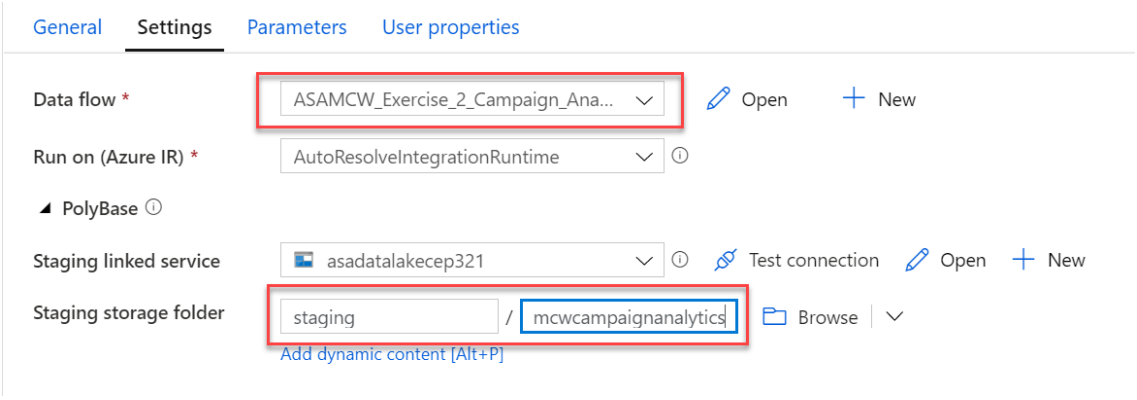


33. In the **Adding data flow** blade, select the data flow **ASAMCW_Exercise_2_Campaign_Analytics_Data**, then **Finish**. Select the Mapping Data Flow activity on the design surface.

34. In the bottom pane, select the **Settings** tab and set the form fields to the following values:

Field	Value
Data flow	**ASAMCW_Exercise_2_Campaign_Analytics_Data**
Staging linked service	**asadatalake{SUFFIX}**
Staging storage folder - Container	Enter **staging**.

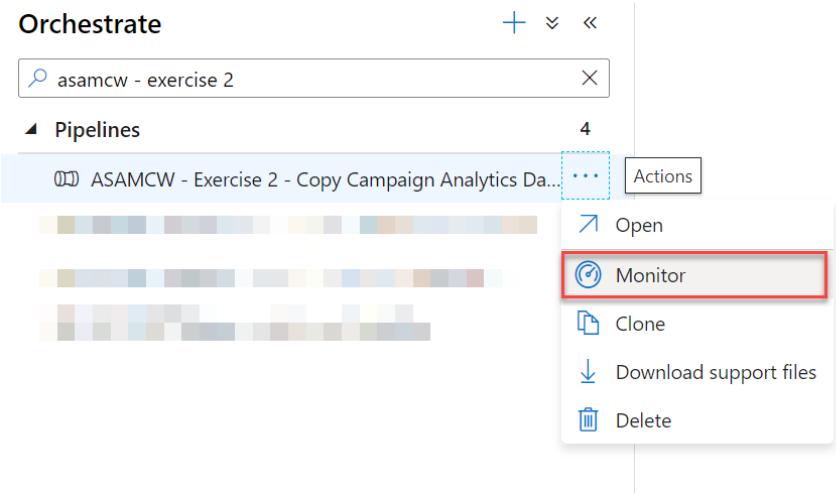
Field	Value
Staging storage folder - Directory	Enter **mcwcampaignanalytics**.



35. In the top toolbar, select **Publish all** to publish the new pipeline. When prompted, select the **Publish** button to commit the changes.

36. Once published, expand the **Add trigger** item on the pipeline designer toolbar, and select **Trigger now**. In the **Pipeline run** blade, select **OK** to proceed with the latest published configuration. You will see notification toast window indicating the pipeline is running and when it has completed.

37. View the status of the pipeline run by locating the **ASAMCW - Exercise 2 - Copy Campaign Analytics Data** pipeline in the Orchestrate blade. Expand the actions menu, and select the **Monitor** item.



38. You should see a run of the pipeline we created in the **Pipeline runs** table showing as in progress. You will need to refresh this table from time to time to see updated progress. Once it has completed. You should see the pipeline run displayed with a Status of **Succeeded**.

39. Verify the table has populated by creating a new query. Select the **Develop** item from the left menu, and in the **Develop** blade, expand the **+** button, and select **SQL script**. In the query window, be sure to connect to the SQL Pool database (`SQLPool01`), then paste and run the following query. When complete, select the **Discard all** button from the top toolbar.

```
select count(Region) from wwi_mcw.CampaignAnalytics;
```

Task 7: Populate the product table

When the lab environment was provisioned, the **wwi_mcw.Product** table and datasets required for its population were created. Throughout this exercise, you have gained experience creating datasets, data flows, and pipelines. The population of the product table would be repetitive, so we will simply trigger an existing pipeline to populate this table.

1. From the left menu, select **Orchestrate ~Integrate~**. From the **Orchestrate** blade, expand the **Pipelines** section and locate and select the **ASAMCW - Exercise 2 - Copy Product Information** pipeline.

2. Expand the **Add trigger** item on the pipeline designer toolbar, and select **Trigger now**. In the **Pipeline run** blade, select **OK** to proceed with the latest published configuration. You will see notification toast windows indicating the pipeline is running and when it has completed.

3. View the status of the pipeline run by locating the **ASAMCW - Exercise 2 - Copy Product Information** pipeline in the Orchestrate blade. Expand the actions menu, and select the **Monitor** item.

4. You should see a run of the pipeline we created in the **Pipeline runs** table showing as in progress (or succeeded). Once it has completed. You should see the pipeline run displayed with a Status of **Succeeded**.

5. Verify the table has populated by creating a new query. Select the **Develop** item from the left menu, and in the **Develop** blade, expand the **+** button, and select **SQL script**. In the query window, be sure to connect to the SQL Pool database (`SQLPool01`), then paste and run the following query. When complete, select the **Discard all** button from the top toolbar.

```
select * from wwi_mcw.Product;
```

Exercise 3: Exploring raw parquet

Duration: 30 minutes

Understanding data through data exploration is one of the core challenges faced today by data engineers and data scientists. Depending on the underlying structure of the data as well as the specific requirements of the exploration process, different data processing engines will offer varying degrees of performance, complexity, and flexibility.

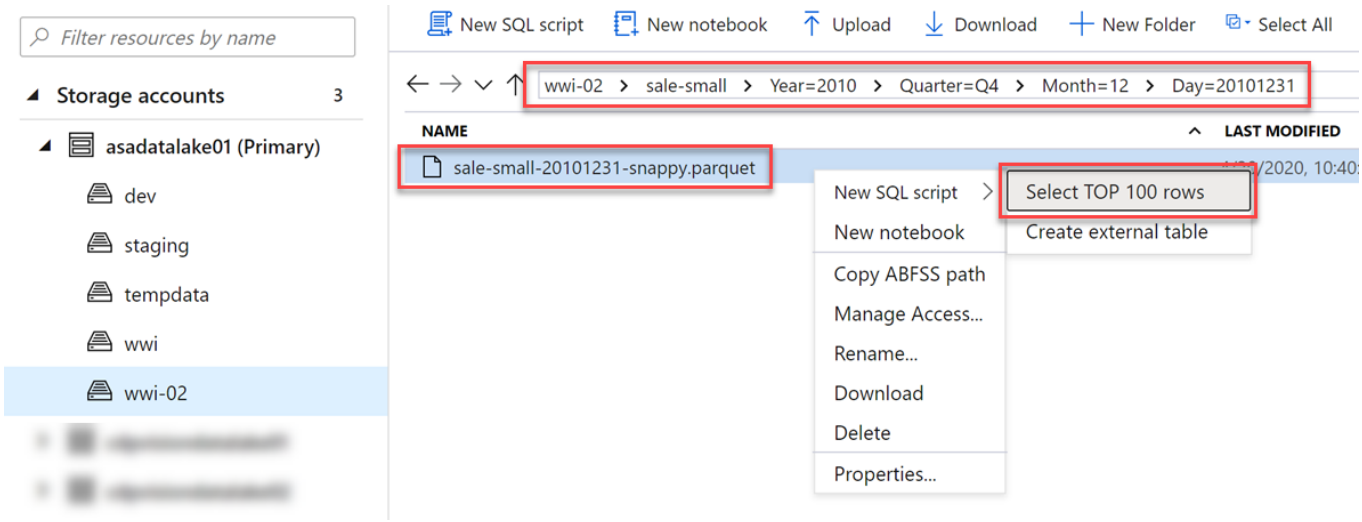
In Azure Synapse Analytics, you have the possibility of using either the Synapse SQL Serverless engine, the big-data Spark engine, or both.

In this exercise, you will explore the data lake using both options.

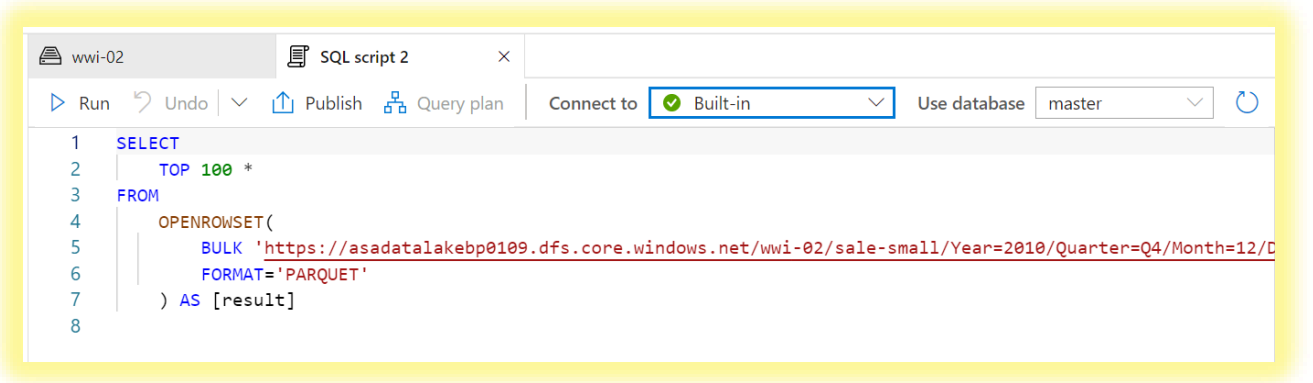
Task 1: Query sales Parquet data with Synapse SQL Serverless

When you query Parquet files using Synapse SQL Serverless, you can explore the data with T-SQL syntax.

1. From the left menu, select **Data**.
2. From the **Data** blade, select the **Linked** tab.
3. Expand **Storage accounts**. Expand the `asadatalake{SUFFIX}` ADLS Gen2 account and select **wwi-02**.
4. Navigate to the **wwi-02/sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231** folder. Right-click on the **sale-small-20101231-snappy.parquet** file, select **New SQL script**, then **Select TOP 100 rows**.



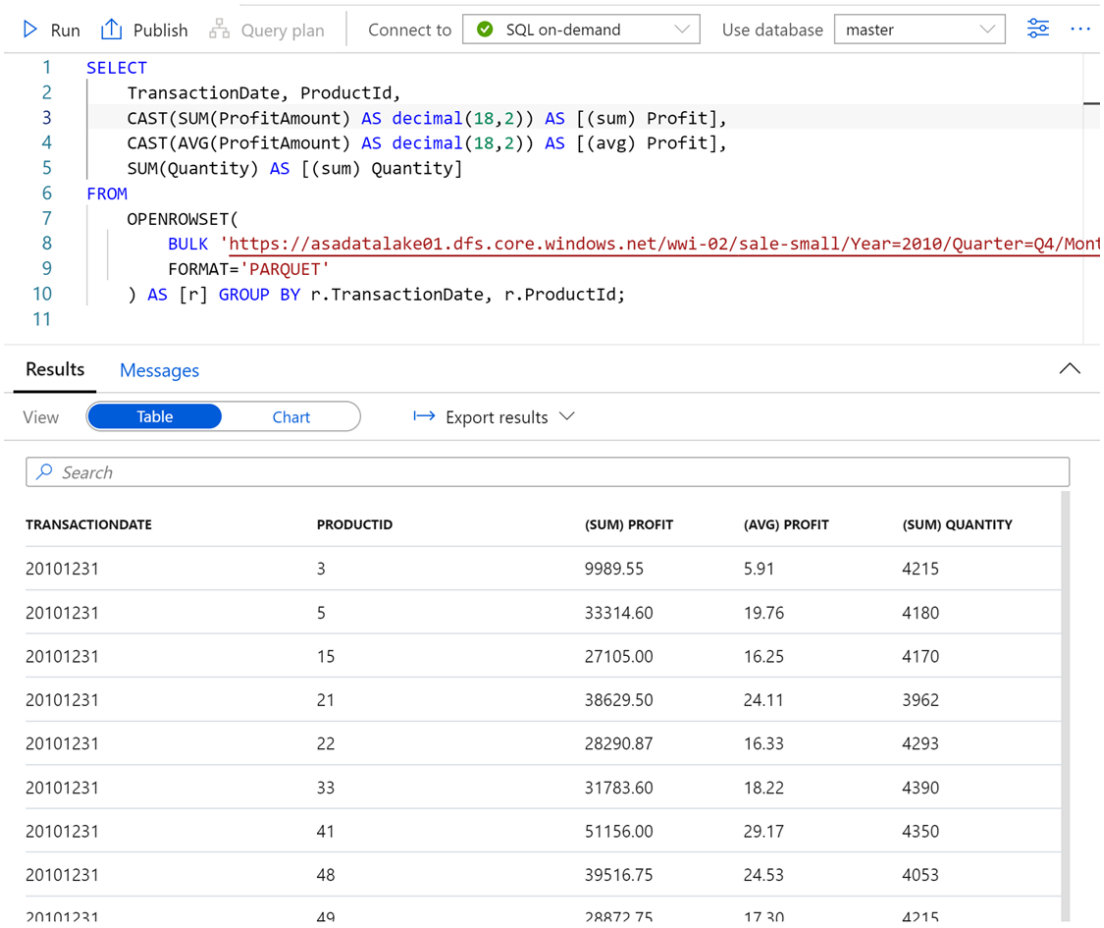
5. Ensure **Built-in** is selected in the **Connect to** dropdown list above the query window, then run the query. Data is loaded by the Synapse SQL Serverless endpoint and processed as if was coming from any regular relational database.



6. Modify the SQL query to perform aggregates and grouping operations to better understand the data. Replace the query with the following, making sure that the file path in **OPENROWSET** matches your current file path, be sure to substitute `asadatalake{SUFFIX}` for the

appropriate value in your environment:

```
SELECT
    TransactionDate, ProductId,
    CAST(SUM(ProfitAmount) AS decimal(18,2)) AS [(sum) Profit],
    CAST(AVG(ProfitAmount) AS decimal(18,2)) AS [(avg) Profit],
    SUM(Quantity) AS [(sum) Quantity]
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231/sale-small-20101231-snappy.parquet',
        FORMAT='PARQUET'
    ) AS [r] GROUP BY r.TransactionDate, r.ProductId;
```



TRANSACTIONDATE	PRODUCTID	(SUM) PROFIT	(AVG) PROFIT	(SUM) QUANTITY
20101231	3	9989.55	5.91	4215
20101231	5	33314.60	19.76	4180
20101231	15	27105.00	16.25	4170
20101231	21	38629.50	24.11	3962
20101231	22	28290.87	16.33	4293
20101231	33	31783.60	18.22	4390
20101231	41	51156.00	29.17	4350
20101231	48	39516.75	24.53	4053
20101231	49	28872.75	17.30	4215

7. Now let's figure out how many records are contained within the Parquet files for 2019 data. This information is important for planning how we optimize for importing the data into Azure Synapse Analytics. To do this, replace your query with the following (be sure to update the name of your data lake in BULK statement, by replacing `asadatalake{SUFFIX}`):

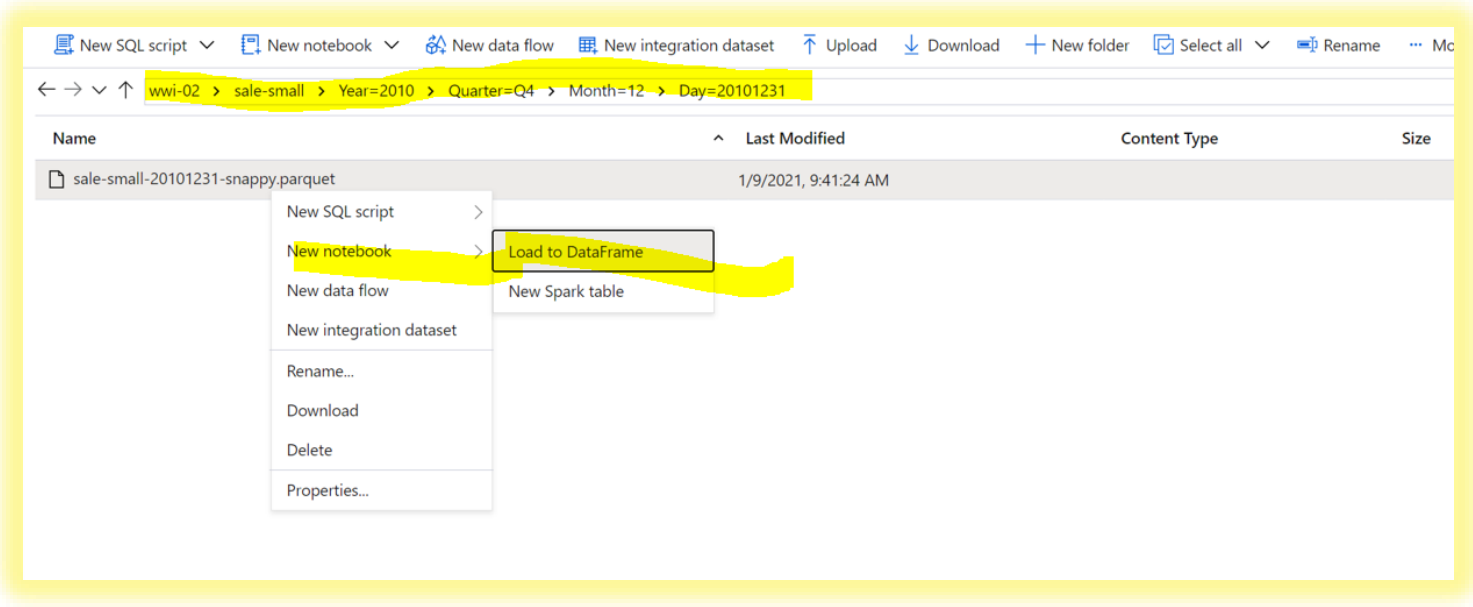
```
SELECT
    COUNT_BIG(*)
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/sale-
small/Year=2019/*/*/*/*',
        FORMAT='PARQUET') AS [r];
```

Notice how we updated the path to include all Parquet files in all subfolders of `sale-small/Year=2019`.

The output should be **339507246** records.

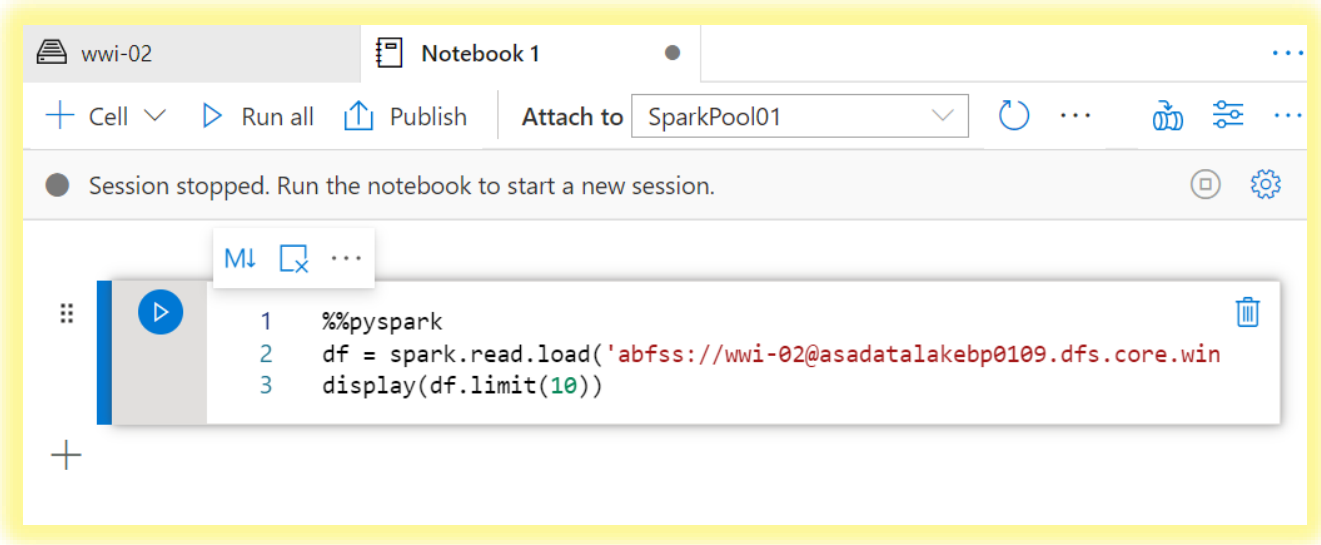
Task 2: Query sales Parquet data with Azure Synapse Spark

1. Select **Data** from the left menu, select the **Linked** tab, then browse to the data lake storage account `asadatalake{SUFFIX}` to **wwi-02/sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231**, then right-click the Parquet file and select New notebook.



2. This will generate a notebook with PySpark code to load the data in a dataframe and display 10 rows with the header.

3. Attach the notebook to a Spark pool – **SparkPool01**.

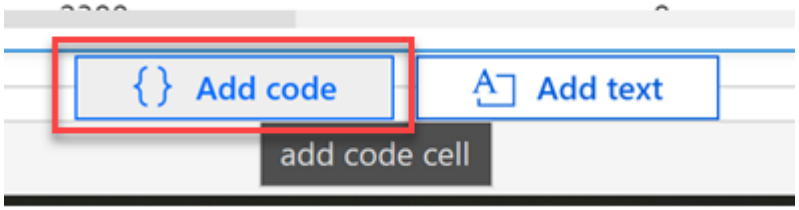


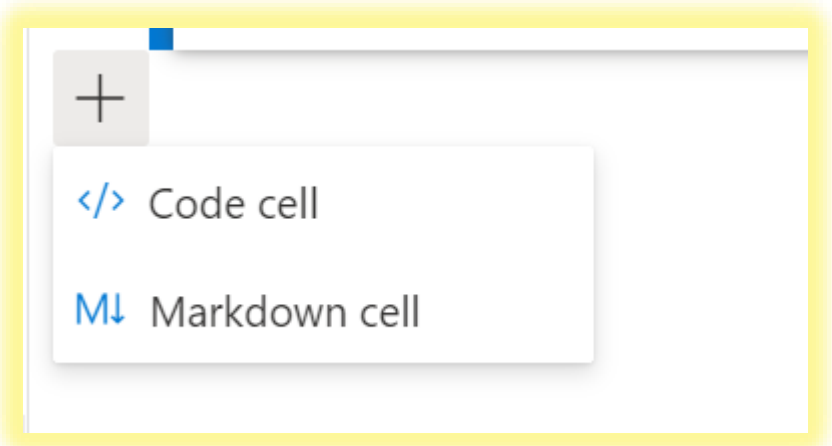
4. Select **Run all** on the notebook toolbar to execute the notebook.

   Note: The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 5 minutes.

   Note: To run just the cell, either hover over the cell and select the *Run cell* icon to the left of the cell, or select the cell then type **Ctrl+Enter** on your keyboard.

5. Create a new cell underneath by selecting **{} Add code** when hovering over the blank space at the bottom of the notebook.

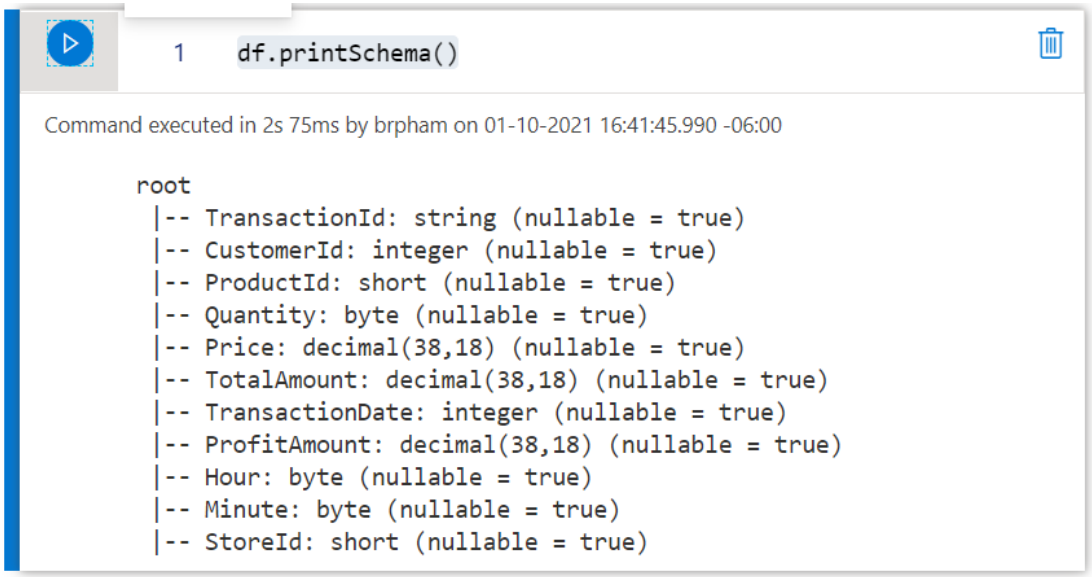




6. The Spark engine can analyze the Parquet files and infer the schema. To do this, enter the following in the new cell:

```
data_path.printSchema()
df.printSchema()
```

Your output should look like the following:



7. Now let's use the dataframe to perform the same grouping and aggregate query we performed with the SQL Serverless pool. Create a new cell and enter the following:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

profitByDateProduct = (df.groupBy("TransactionDate", "ProductId").agg(
round(sum("ProfitAmount"),2).alias("(sum)Profit"),
round(avg("ProfitAmount"),2).alias("(avg)Profit"),
sum("Quantity").alias("(sum)Quantity")
).orderBy("TransactionDate", "ProductId")
)
profitByDateProduct.show(100)
```

We import required Python libraries to use aggregation functions and types defined in the schema to successfully execute the query.

Exercise 4: Exploring raw text based data with Azure Synapse SQL Serverless

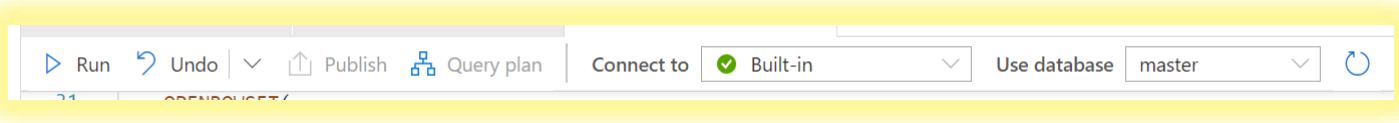
Duration: 15 minutes

A common format for exporting and storing data is with text based files. These can delimited text files such as CSV as well as JSON structured data files. Azure Synapse Analytics also provides ways of querying into these types of raw files to gain valuable insights into the data without having to wait for them to be processed.

Task 1: Query CSV data

1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

2. Ensure **Built-in** is selected in the **Connect to** dropdown list above the query window.



3. In this scenario, we will be querying into the CSV file that was used to populate the product table. This file is located in the `asadatalake{SUFFIX}` account at: **wwi-02/data-generators/generator-product.csv**. We will select all data from this file. Copy and paste the following query into the query window and select **Run** from the query window toolbar menu. Remember to replace `asadatalake{SUFFIX}` with your storage account name.

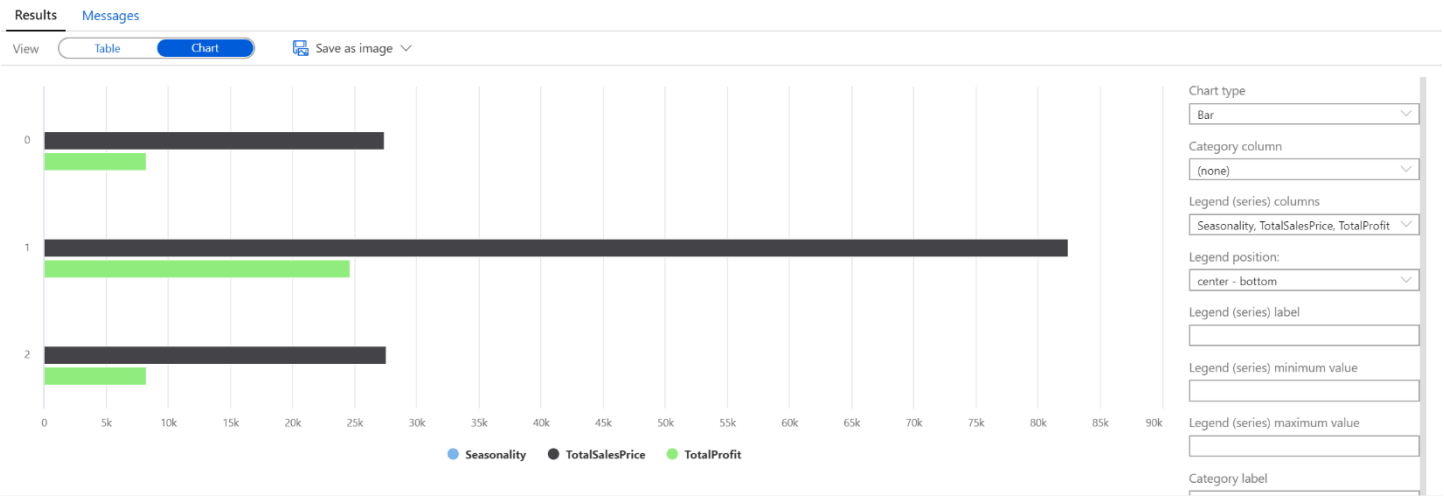
```
SELECT csv.*
 FROM
     OPENROWSET(
         BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/data-
 generators/generator-product/generator-product.csv',
         FORMAT='CSV',
         FIRSTROW = 1
     ) WITH (
         ProductID INT,
         Seasonality INT,
         Price DECIMAL(10,2),
         Profit DECIMAL(10,2)
     ) as csv
```

Note: In this query we are querying only a single file. Azure Synapse Analytics allows you to query across a series of CSV files (structured identically) by using wildcards in the path to the file(s).

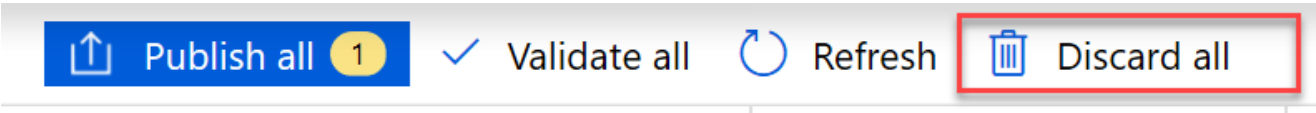
4. You are also able to perform aggregations on this data. Replace the query with the following, and select **Run** from the toolbar menu. Remember to replace `asadatalake{SUFFIX}` with your storage account name.

```
SELECT
    Seasonality,
    SUM(Price) as TotalSalesPrice,
    SUM(Profit) as TotalProfit
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/data-
generators/generator-product/generator-product.csv',
        FORMAT='CSV',
        FIRSTROW = 1
    ) WITH (
        ProductID INT,
        Seasonality INT,
        Price DECIMAL(10,2),
        Profit DECIMAL(10,2)
    ) as csv
GROUP BY csv.Seasonality
```

5. After you have run the previous query, switch the view on the **Results** tab to **Chart** to see a visualization of the aggregation of this data. Feel free to experiment with the chart settings to obtain the best visualization!



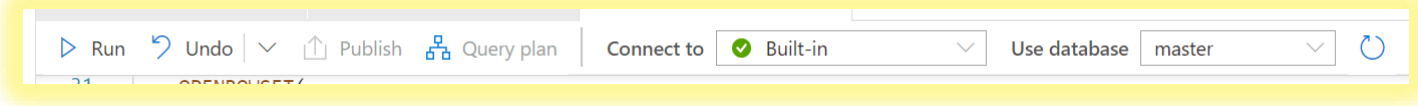
6. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Task 2: Query JSON data

1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

2. Ensure **Built-in** is selected in the **Connect to** dropdown list above the query window.

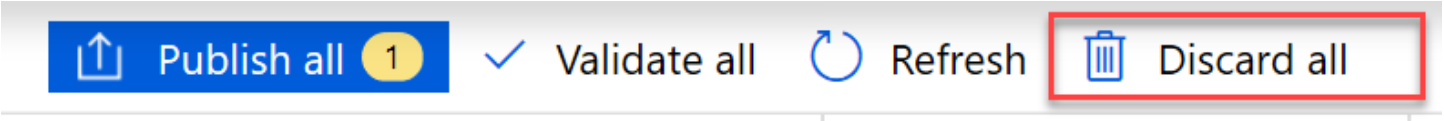


3. Replace the query with the following, remember to replace `asadatalake{SUFFIX}` with the name of your storage account:

```
SELECT
    products.*
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/product-json/json-
data/*.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent NVARCHAR(200)
    ) AS [raw]
```

```
CROSS APPLY OPENJSON(jsonContent)
WITH (
    ProductId INT,
    Seasonality INT,
    Price DECIMAL(10,2),
    Profit DECIMAL(10,2)
) AS products
```

4. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Exercise 5: Synapse Pipelines and Cognitive Search

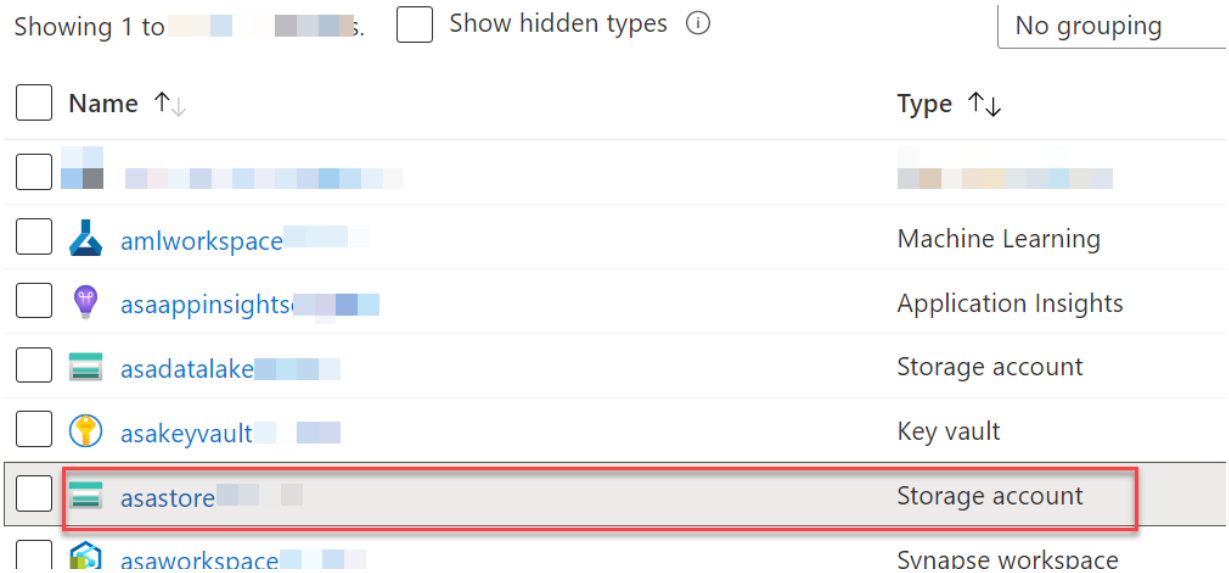
Duration: 45 minutes

In this exercise you will create a Synapse Pipeline that will orchestrate updating the part prices from a supplier invoice. You will accomplish this by a combination of a Synapse Pipeline with an Azure Cognitive Search Skillset that invokes the Form Recognizer service as a custom skill. The pipeline will work as follows:

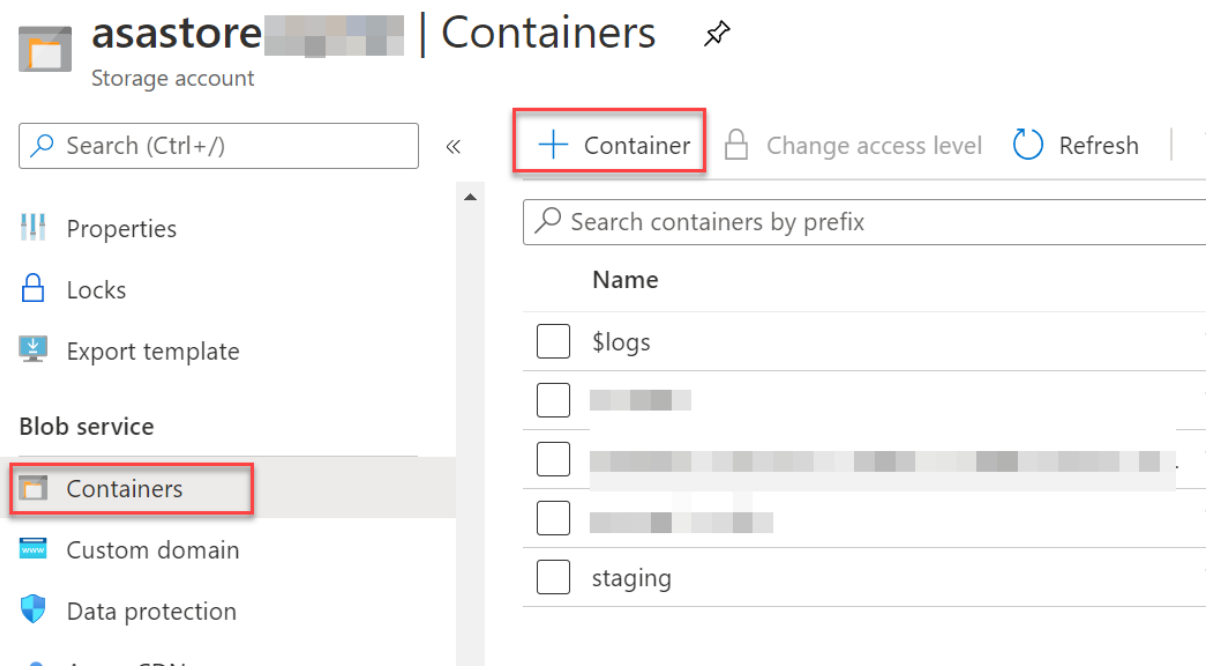
- Invoice is uploaded to Azure Storage.
- An Azure Cognitive Search index is started
- The index of any new or updated invoices invokes an Azure Cognitive Search skillset.
- The first skill in the skillset invokes an Azure Function, passing it the URL to the PDF invoice.
- The Azure Function invokes the Form Recognizer service, passing it the URL and SAS token to the PDF invoice. Forms recognizer returns the OCR results to the function.
- The Azure Function returns the results to skillset. The skillset then extracts only the product names and costs and sends that to a configure knowledge store that writes the extracted data to JSON files in Azure Blob Storage.
- The Synapse pipeline reads these JSON files from Azure Storage in a Data Flow activity and performs an upsert against the product catalog table in the Synapse SQL Pool.

Task 1: Create the invoice storage container

1. In the Azure Portal, navigate to the lab resource group and select the **asastore{suffix}** storage account.



2. From the left menu, beneath **Blob service**, select **Containers**. From the top toolbar menu of the **Containers** screen, select **+ Container**.

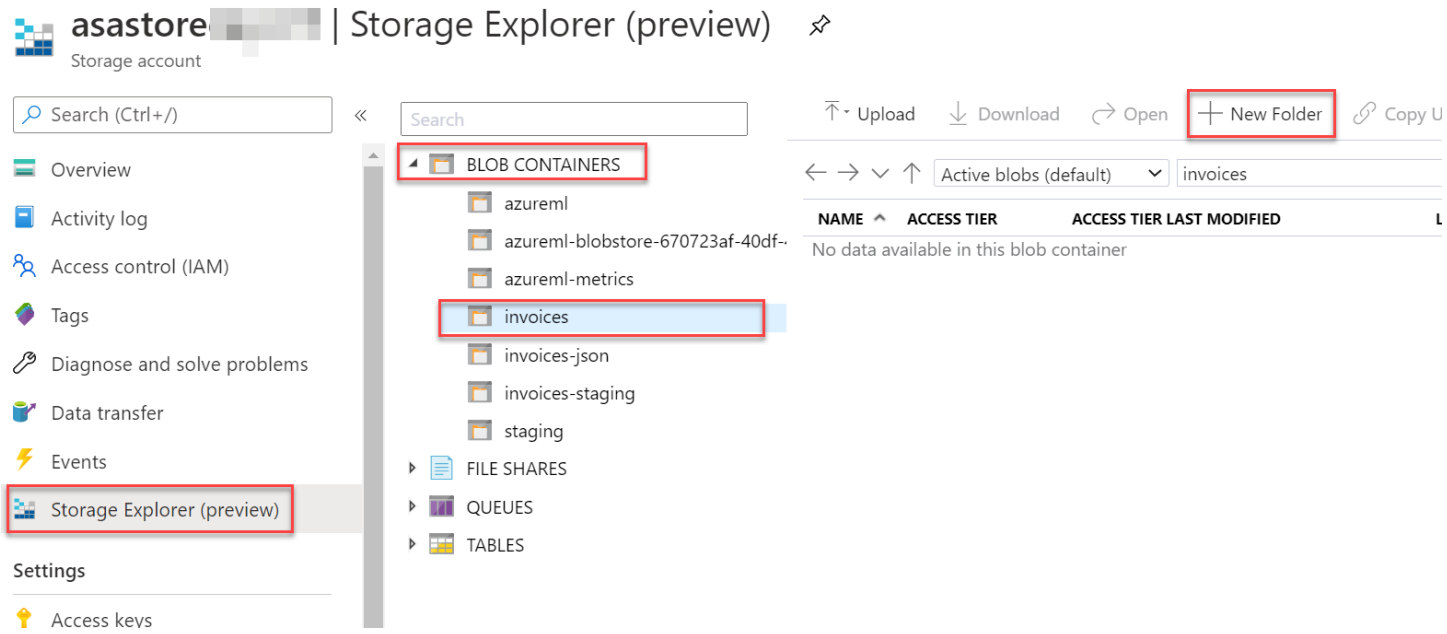


3. On the **New container** blade, name the container **invoices**, and select **Create**, we will keep the default values for the remaining fields.

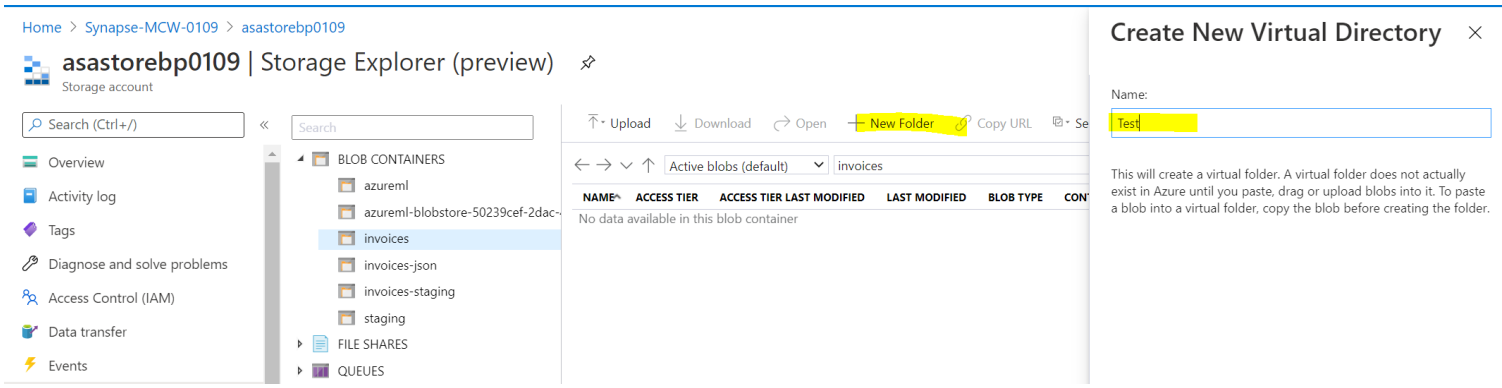
4. Repeat steps 2 and 3, and create two additional containers named **invoices-json** and **invoices-staging**.

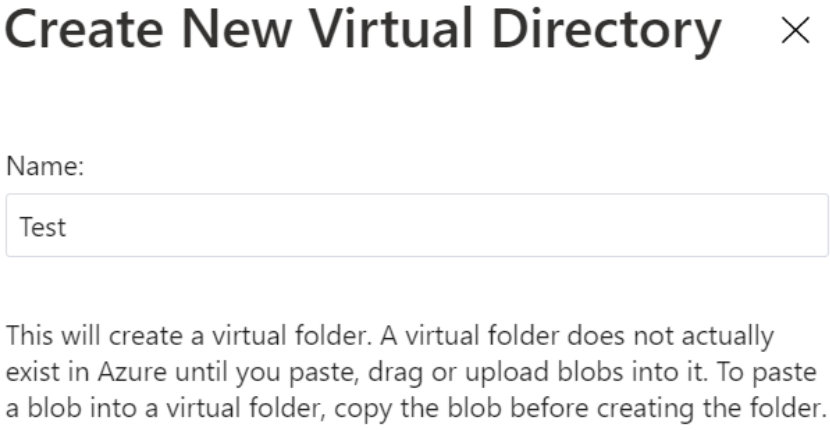
5. From the left menu, select **Storage Explorer (preview)**. Then, in the hierarchical menu, expand the **BLOB CONTAINERS** item.

6. Beneath **BLOB CONTAINERS**, select the **invoices** container, then from the taskbar menu, select **+ New Folder**

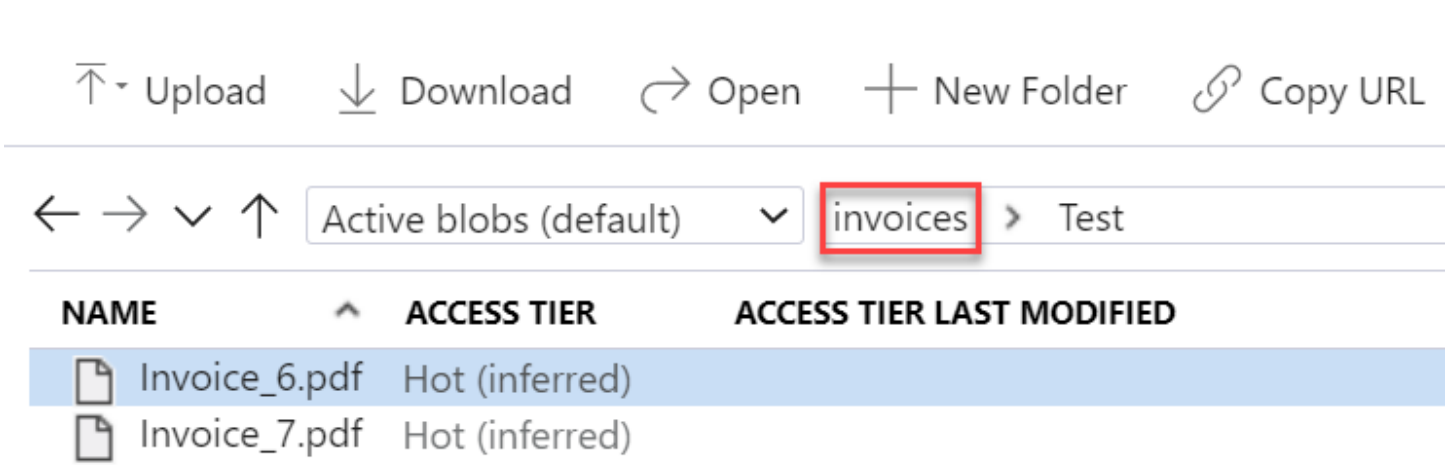


7. In the **Create New Virtual Directory** blade, name the directory **Test**, then select **OK**. This will automatically move you into the new **Test** folder.

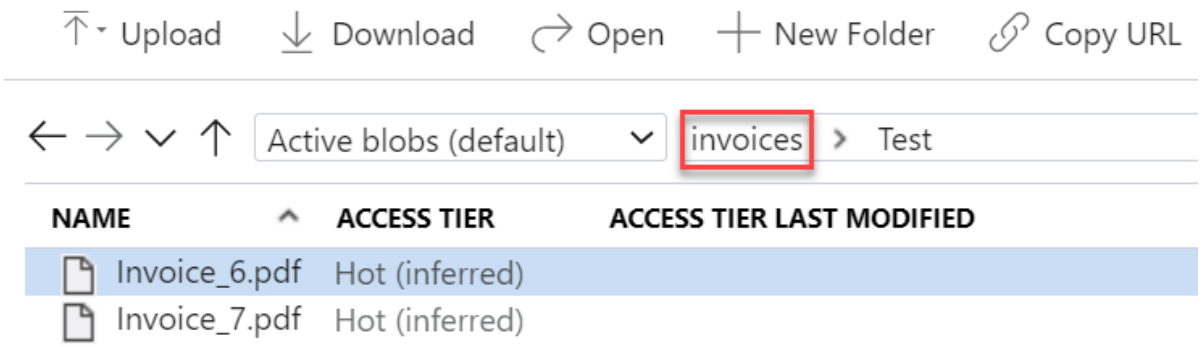




8. From the taskbar, select **Upload**. Upload all invoices located in **Hands-on lab/artifacts/sample_invoices/Test**. These files are Invoice_6.pdf and Invoice_7.pdf.



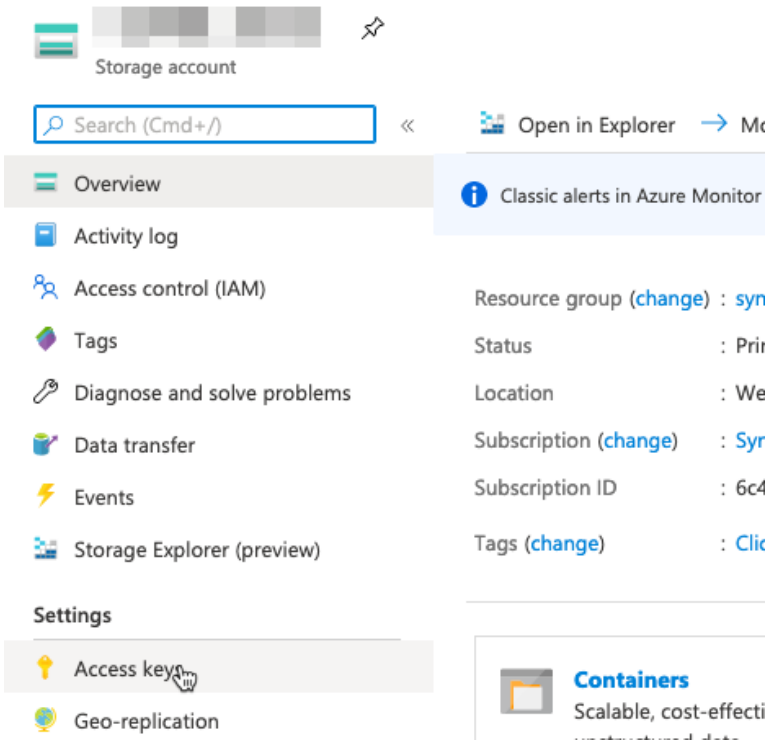
9. Return to the root **invoices** folder by selecting the **invoices** breadcrumb from the location textbox found beneath the taskbar.



10. From the taskbar, select **+ New Folder** once again. This time creating a folder named **Train**. This will automatically move you into the new **Train** folder.

11. From the taskbar, select **Upload**. Upload all invoices located in **Hands-on lab/artifacts/sample_invoices/Train**. These files are Invoice_1.pdf, Invoice_2.pdf, Invoice_3.pdf, Invoice_4.pdf and Invoice_5.pdf.

12. From the left menu, select **Access keys**. Click **Show keys** button.

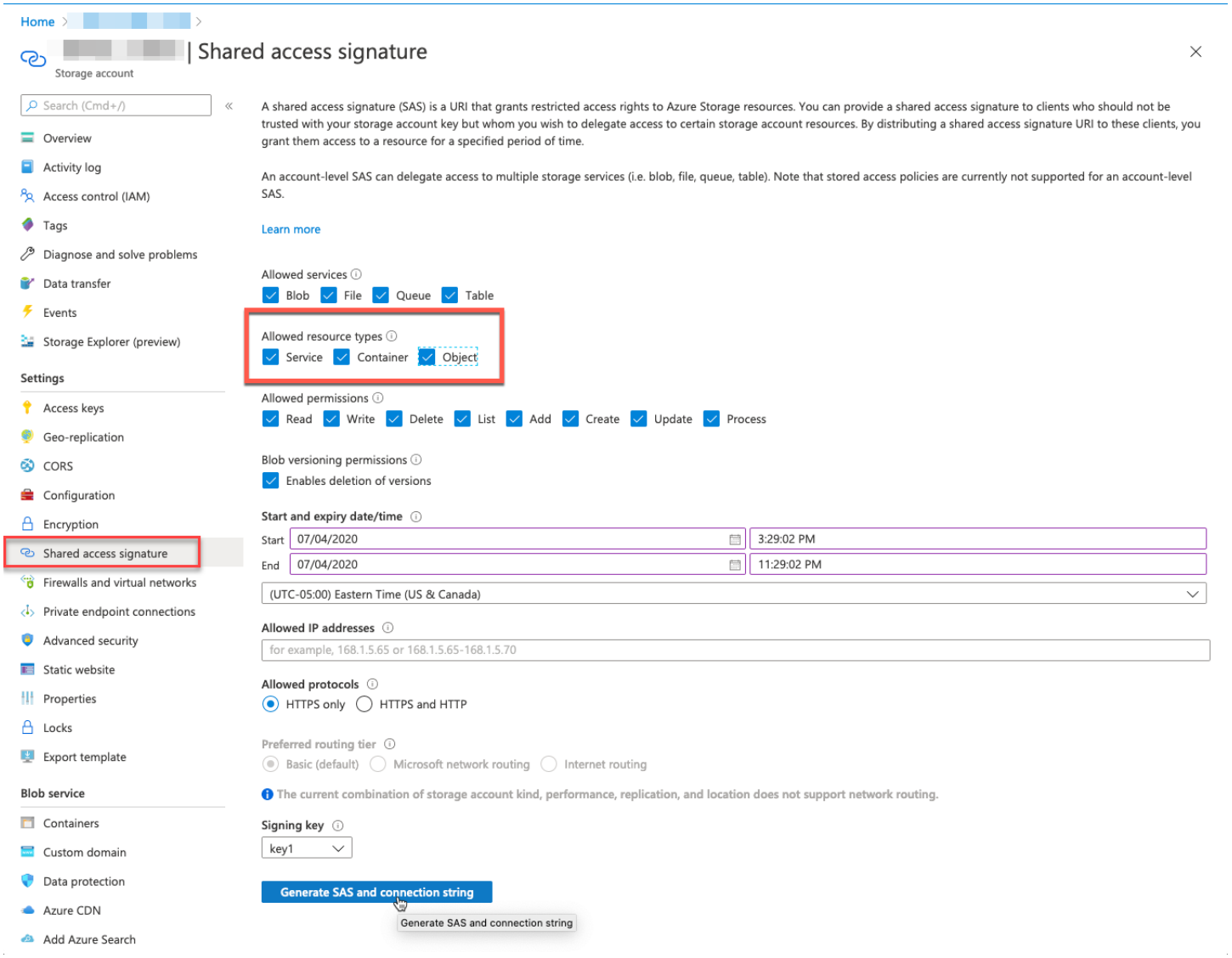


13. Copy the Connection string under **key1**. Save it to notepad, Visual Studio Code, or another text file. We'll use this several times

14. From the left menu, beneath **Settings**, select **Shared access signature**.

15. Make sure all the checkboxes are selected and choose **Generate SAS and connection string**.

    For 'End' date, extend 7 days from today.



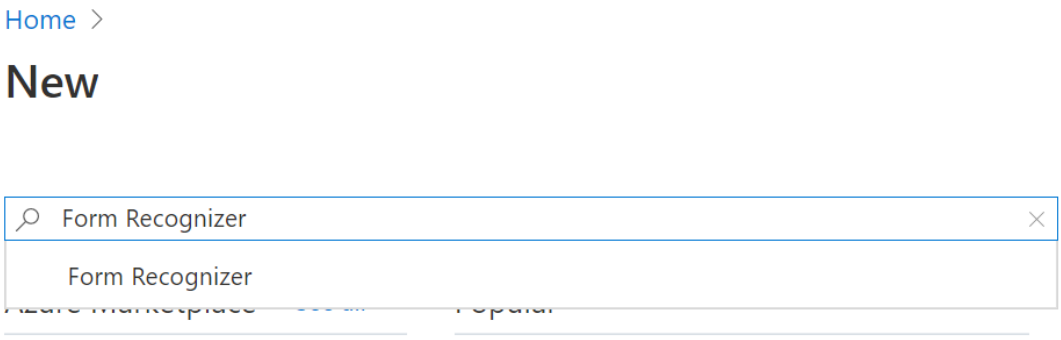
16. Copy the generated **Blob service SAS URL** to the same text file as above.

17. Modify the SAS URL that you just copied and add the **invoices** container name just before the '**?**' character.

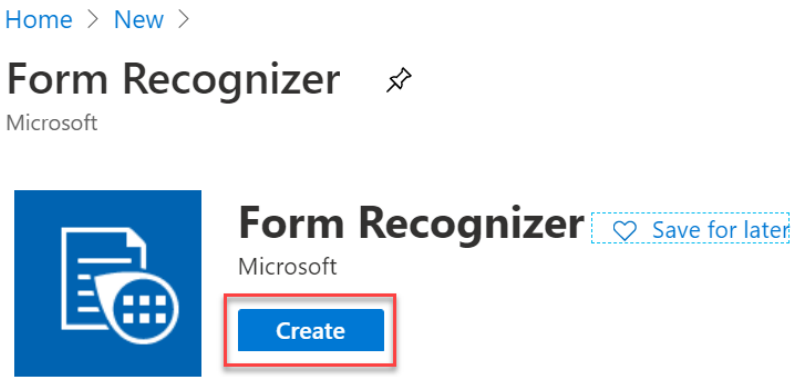
Example: https://asastore{{suffix}.blob.core.windows.net/invoices?sv=2019-12-12&ss=bfqt&srt ...

Task 2: Create and train an Azure Forms Recognizer model and setup Cognitive Search

1. Browse to your Azure Portal homepage, select **+ Create a resource**, then search for and select **Form Recognizer** from the search results.

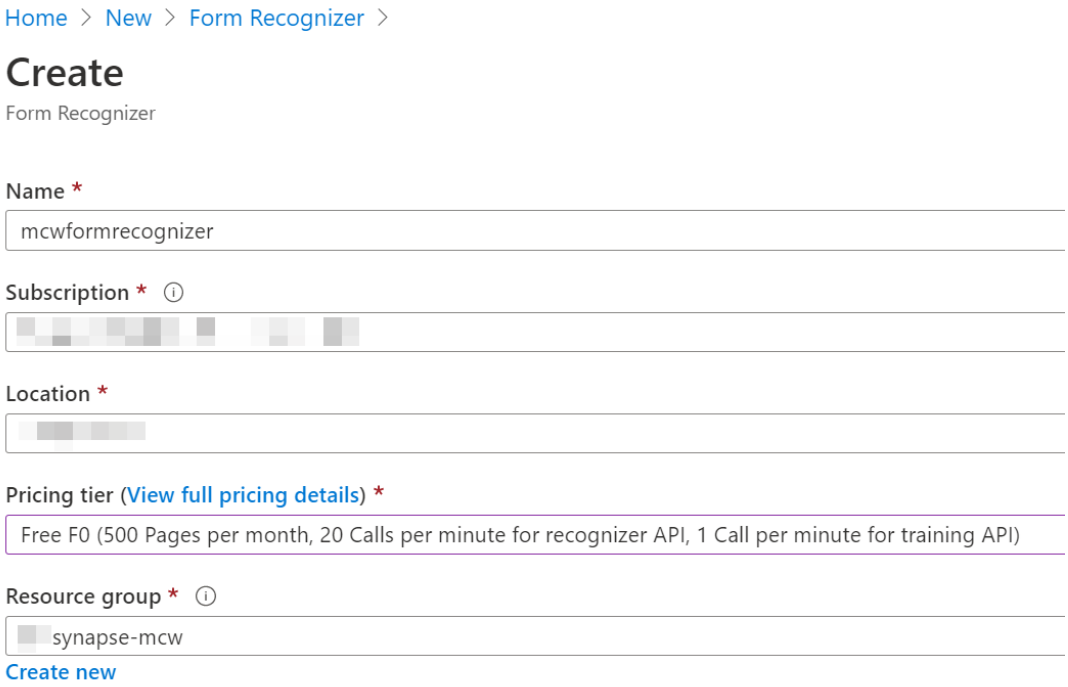


2. Select **Create**.



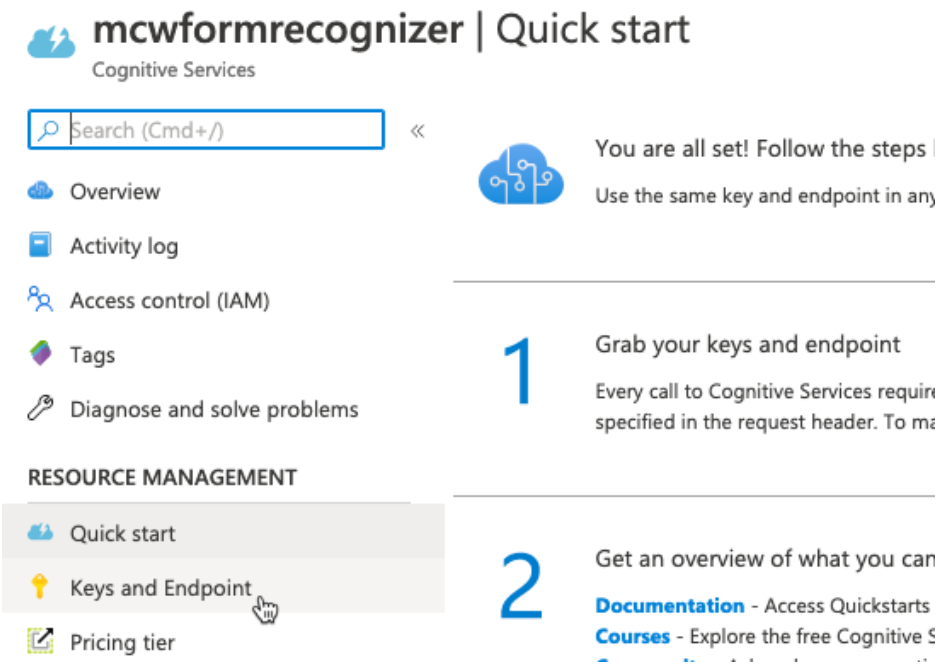
3. Enter the following configuration settings, then select **Create**:

Field	Value
Name	Enter a unique name (denoted by the green checkmark indicator) for the form recognition service.
Subscription	Select the lab subscription.
Location	Select the lab region.
Pricing	Select **Free F0**.
Confirmation checkbox	Checked.

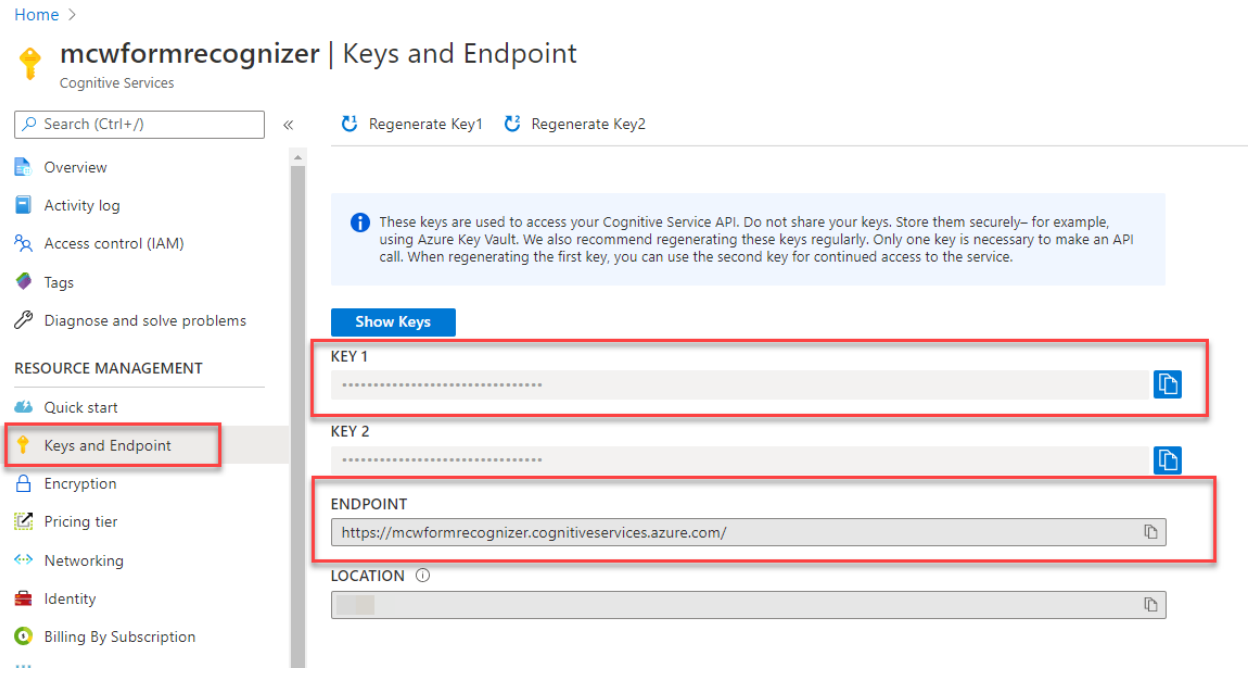


4. Wait for the service to provision then navigate to the resource.

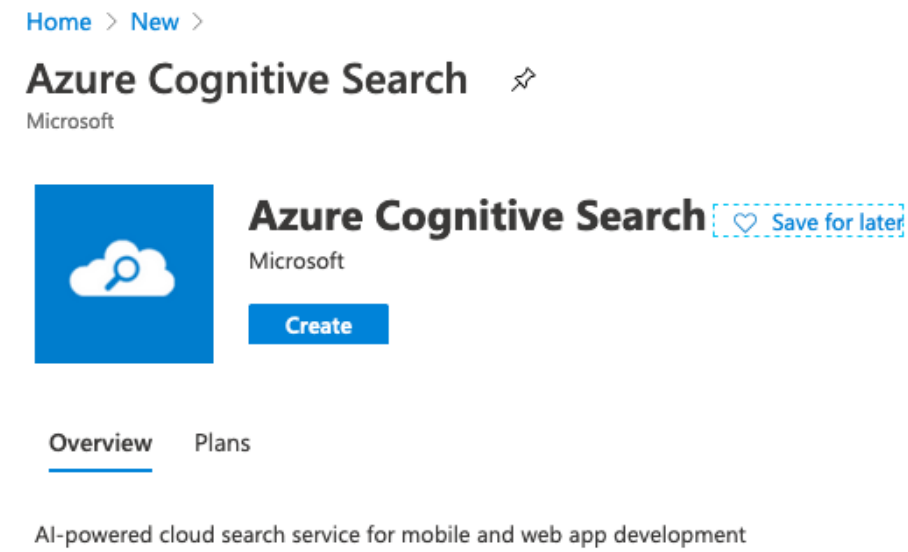
5. From the left menu, select **Keys and Endpoint**. Click the **Show Keys** button.



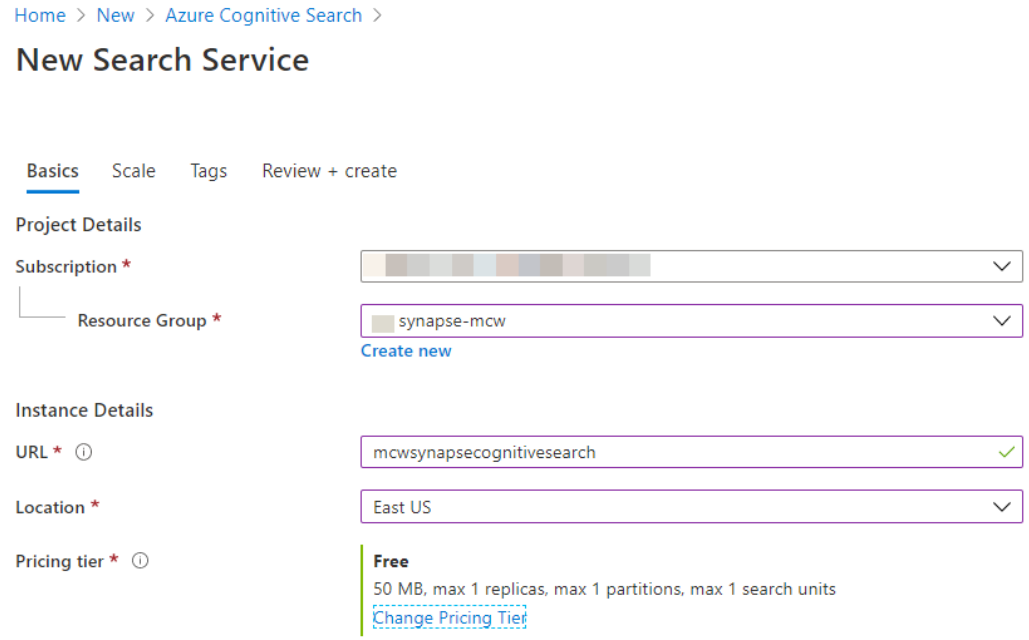
6. Copy and Paste both KEY 1 and the ENDPOINT values. Put these in the same location as the storage connection string you copied earlier



7. Browse to your Azure Portal homepage, select **+ Create a new resource**, then search for and create a new instance of **Azure Cognitive Search**.



8. Choose the subscription and the resource group you've been using for this lab. Set the URL of the Cognitive Search Service to a unique value, relating to search. Then, switch the pricing tier to **Free**.

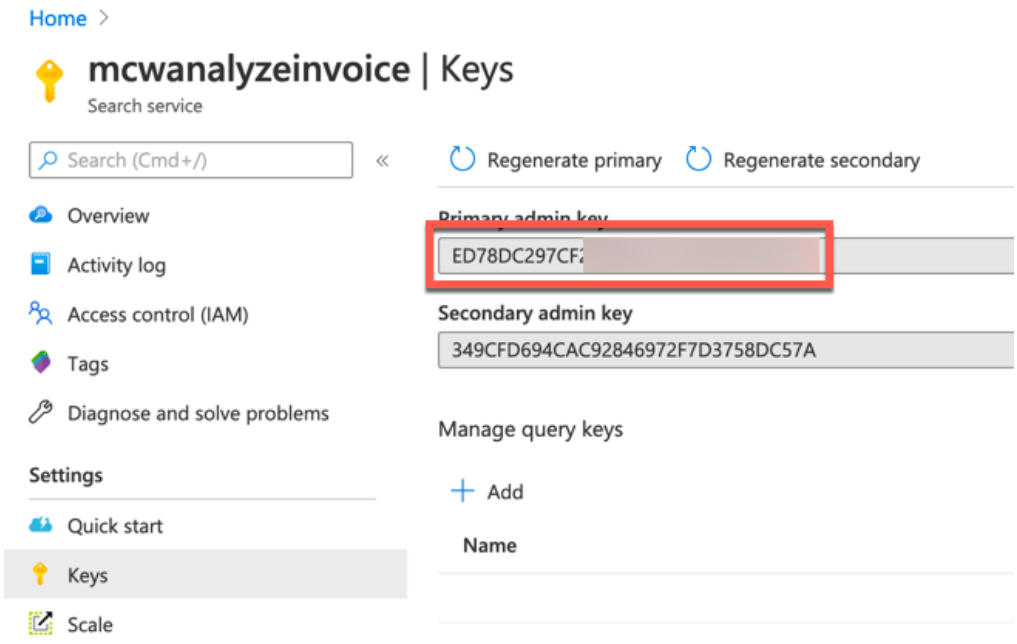


9. Select **Review + create**.

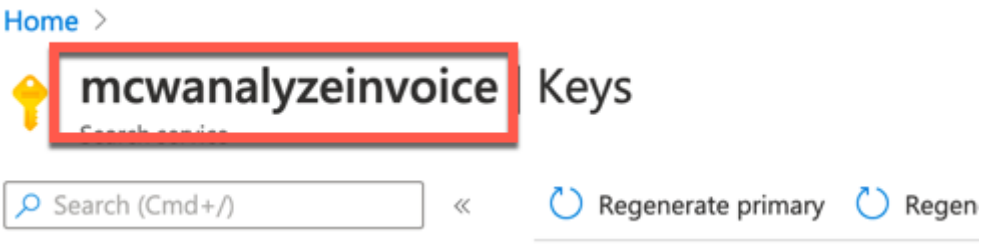
10. Select **Create**.

11. Wait for the Search service to be provisioned then navigate to the resource.

12. From the left menu, select **Keys**, copy the **Primary admin key** and paste it into your text document. Also make note of the name of your search service resource.



13. Also make note of the name of your search service in the text document.



14. Open Visual Studio Code.

15. From the **File** menu, select **Open file** then choose to open **Hands-on lab/artifacts/pocformreader.py**.

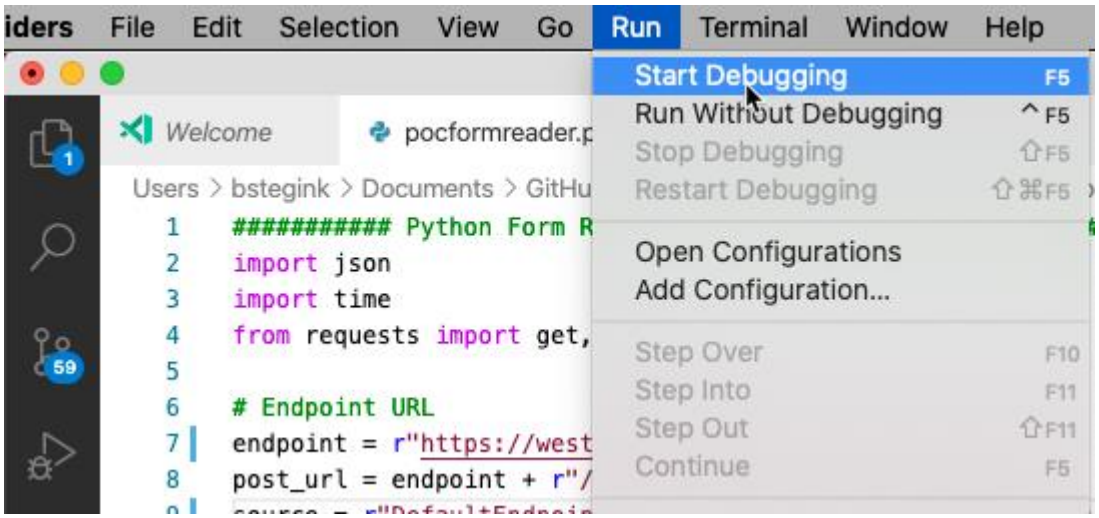
16. Update Lines 7, 9, and 17 with the appropriate values indicated below:

    - Line 7: The endpoint of Azure Cognitive Services.
    - Line 9: The Blob Service SAS URL storage account with your Train and Test invoice folders.
    - Line 17: The key for your Azure Cognitive Service endpoint.

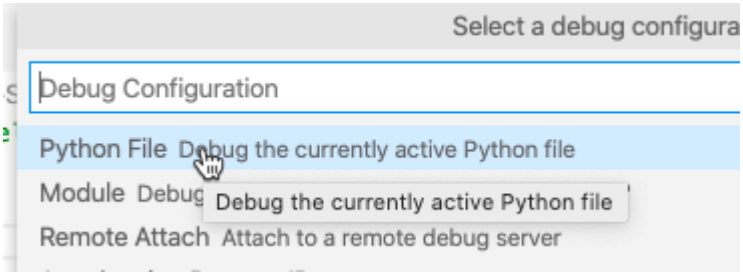
```
1   ########### Python Form Recognizer Labeled Async Train #############
2   import json
3   import time
4   from requests import get, post
5
6   # Endpoint URL
7   endpoint = r"https://wwiformsreader.cognitiveservices.azure.com/"
8   post_url = endpoint + r"/formrecognizer/v2.0/custom/models"
9   source = r"https://mcwinvoices.blob.core.windows.net/?sv=2019-10-10&ss=bfqt&srt=sco&sp=rwdlacupx&s
10  prefix = "train"
11  includeSubFolders = False
12  useLabelFile = False
13
14  headers = {
15      # Request headers
16      'Content-Type': 'application/json',
17      'Ocp-Apim-Subscription-Key': '8a537                              ',
18  }
19
20  body =      {
21      "source": source,
```

17. Save the file.

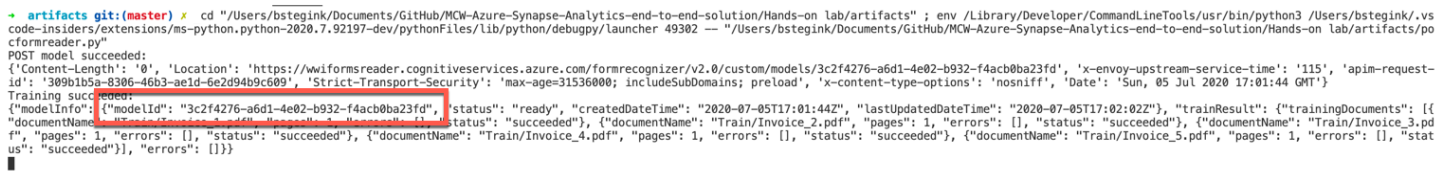
18. Select Run, then Start Debugging.



19. In the **Debug Configuration**, select to debug the **Python File - Debug the currently active Python File** value.



20. When it completes, you should see an output similar to what is seen in the screenshot below. The output should also contain a modelID. Copy and paste this into your text file to use later

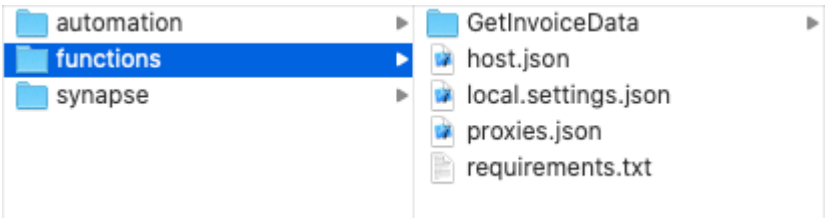


Note: If you receive an error stating the **requests** module is not found, from the terminal window in Visual Studio code, execute: **pip install requests**

Task 3: Configure a skillset with Form Recognizer

1. Open a new instance of Visual Studio Code.

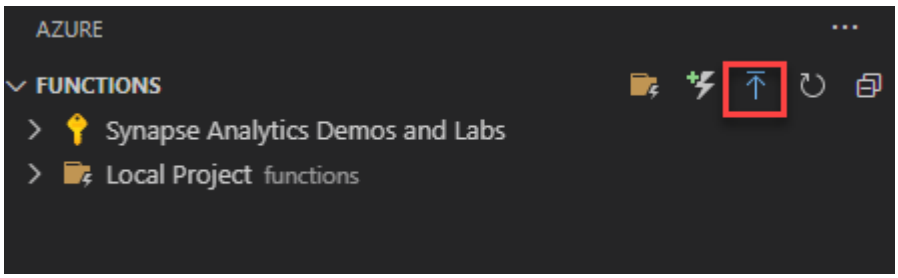
2. In Visual Studio Code open the folder **Hands-on lab/environment-setup/functions**.



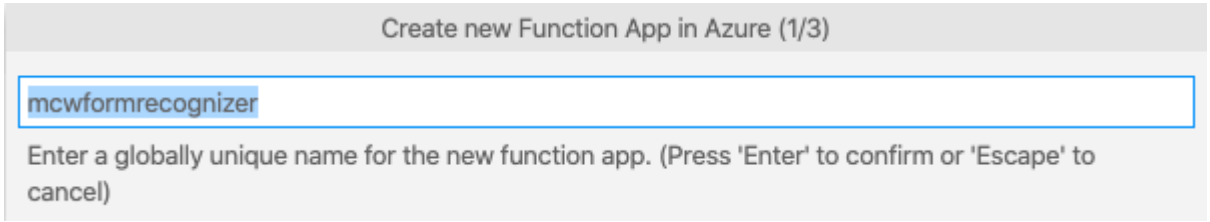
3. In the **GetInvoiceData/__init__.py** file, update lines 66, 68, 70, and 73 with the appropriate values for your environment, the values that need replacing are located between **<<** and **>>** values.

```
64
65      # Endpoint URL for your form recognizer service
66      endpoint = r"https://<<formreaderservc>>.cognitiveservices.azure.com/"
67      #Key for the Form Recognizer Service
68      apim_key = "<<Form Recognizer Key>>"
69      #Insert the ID for the form recognizer model
70      model_id = "<<Model ID for Form Recognizer>>"
71      post_url = endpoint + "/formrecognizer/v2.0/custom/models/%s/analyze" % model_id
72      #Azure storage connection string where files will be uploaded
73      connectionstring = "<<Azure Blob Storage Connection String>>"
74      path = fullpath.split(".net/",1)
75      path = path[1]
```

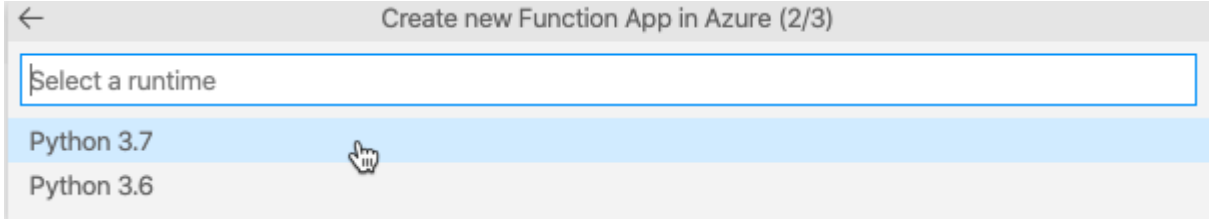
4. Use the Azure Functions extension to publish to a new Azure function. If you don't see the Azure Functions panel, go to the **View** menu, select **Open View...** and choose **Azure**. If the panel shows the **Sign-in to Azure** link, select it and log into Azure. Select the **Publish** button at the top of the panel.



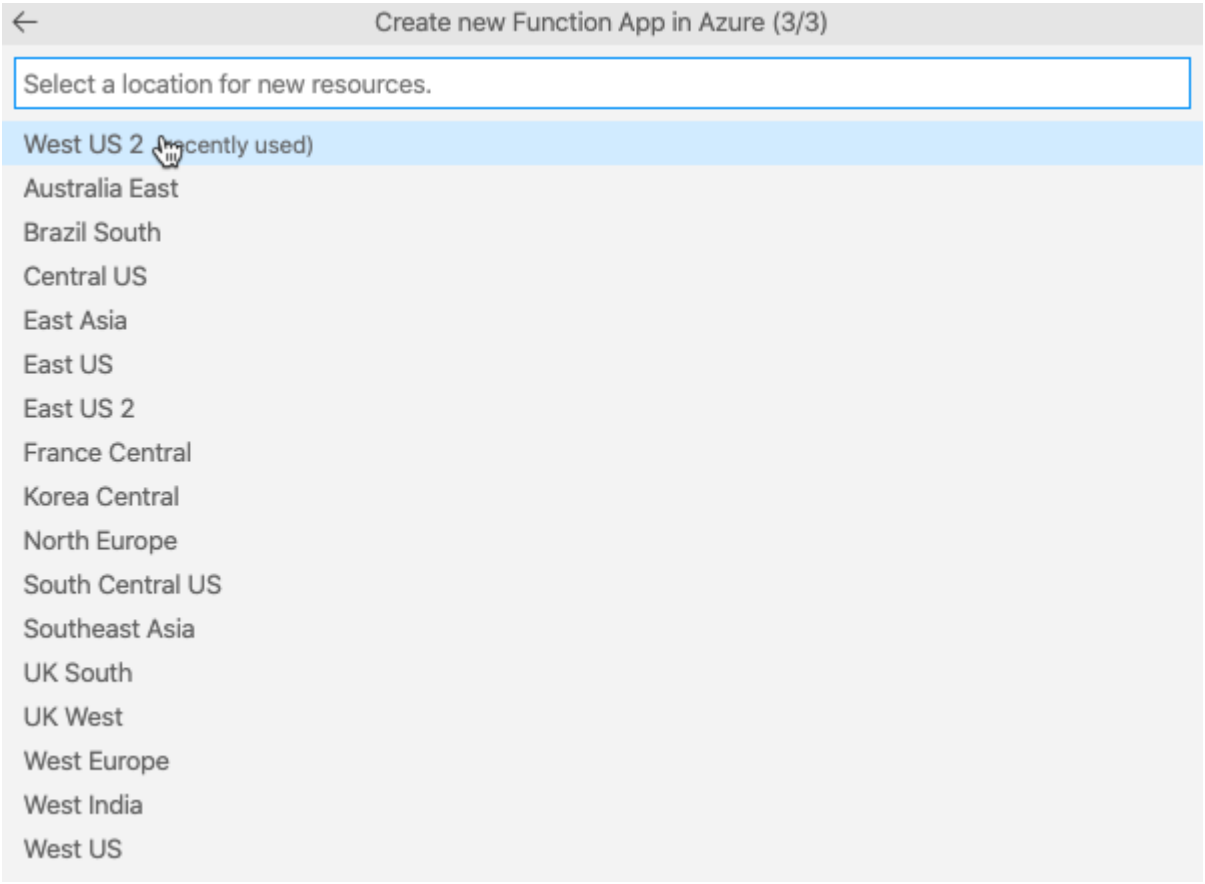
   - Select the same subscription as your Synapse workspace.
   - Choose to **Create new Function App in Azure...**.
   - Give this function a unique name, relative to form recognition.



   - For the runtime select Python 3.7.

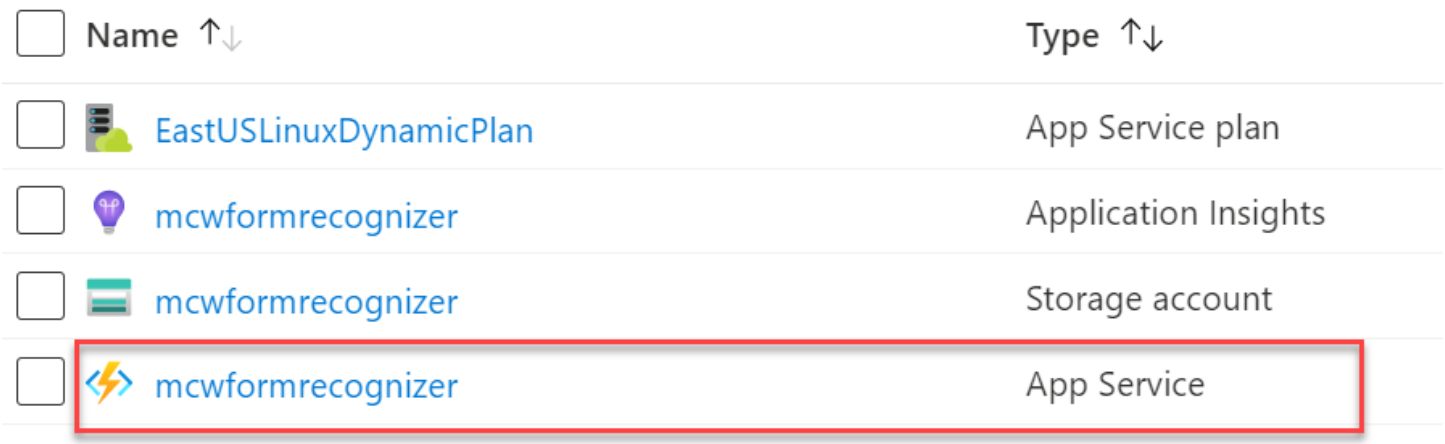


   - Deploy the function to the same region as your Synapse workspace.



5. Once publishing has completed, return to the Azure Portal and search for a resource group that was created with the same name as the Azure Function App.

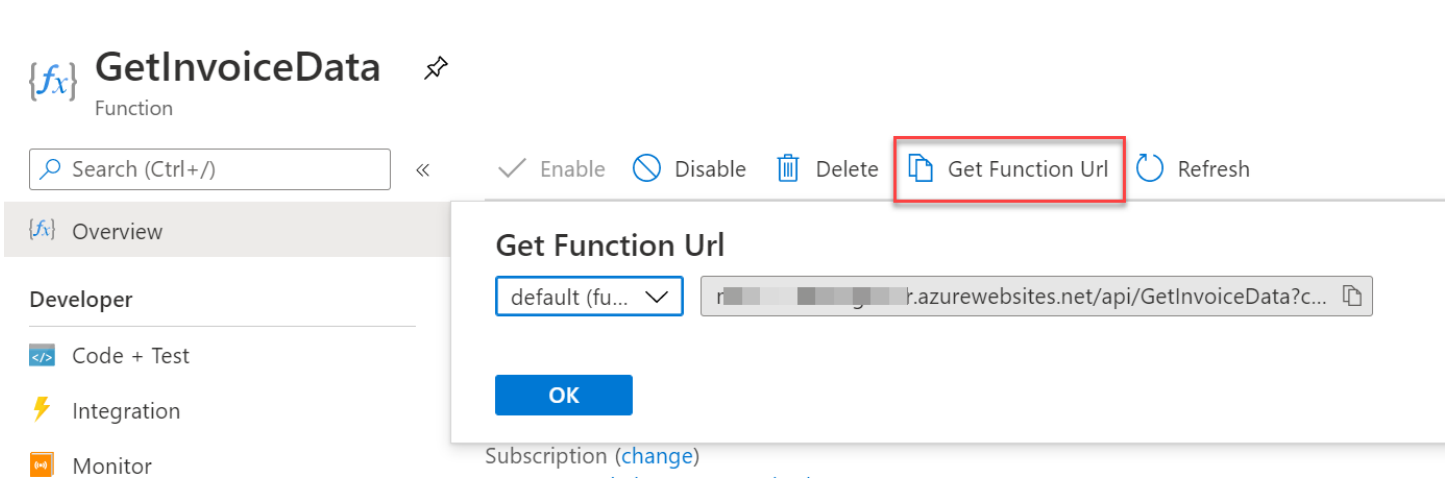
6. Within this resource group, open the **App Service** resource with the same name.



7. From the left menu, beneath the **Functions** heading, select **Functions**.

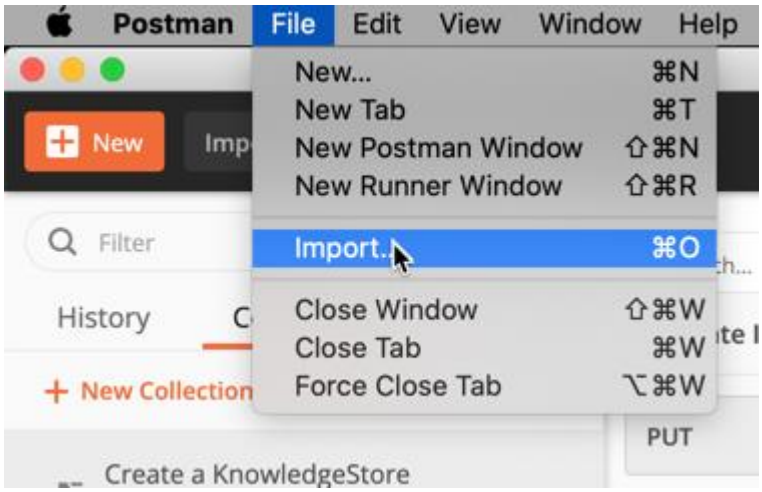
8. From the Functions listing, select **GetInvoiceData**.

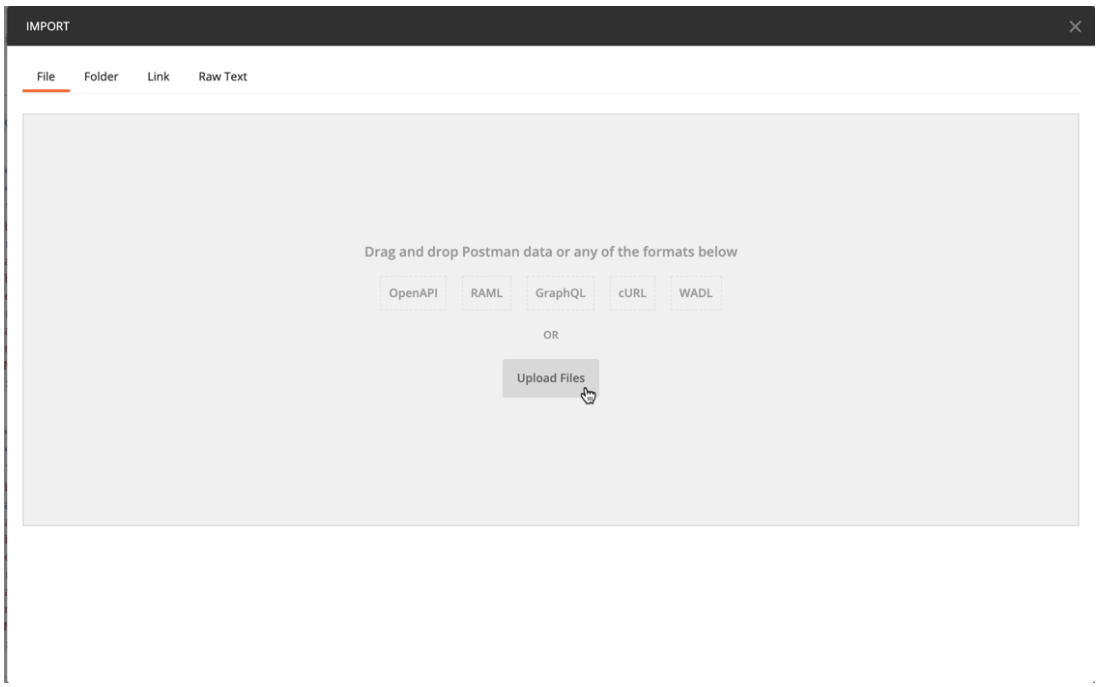
9. From the toolbar menu of the **GetInvoiceData** screen, select the **Get Function Url** item, then copy this value to your text document for later reference.

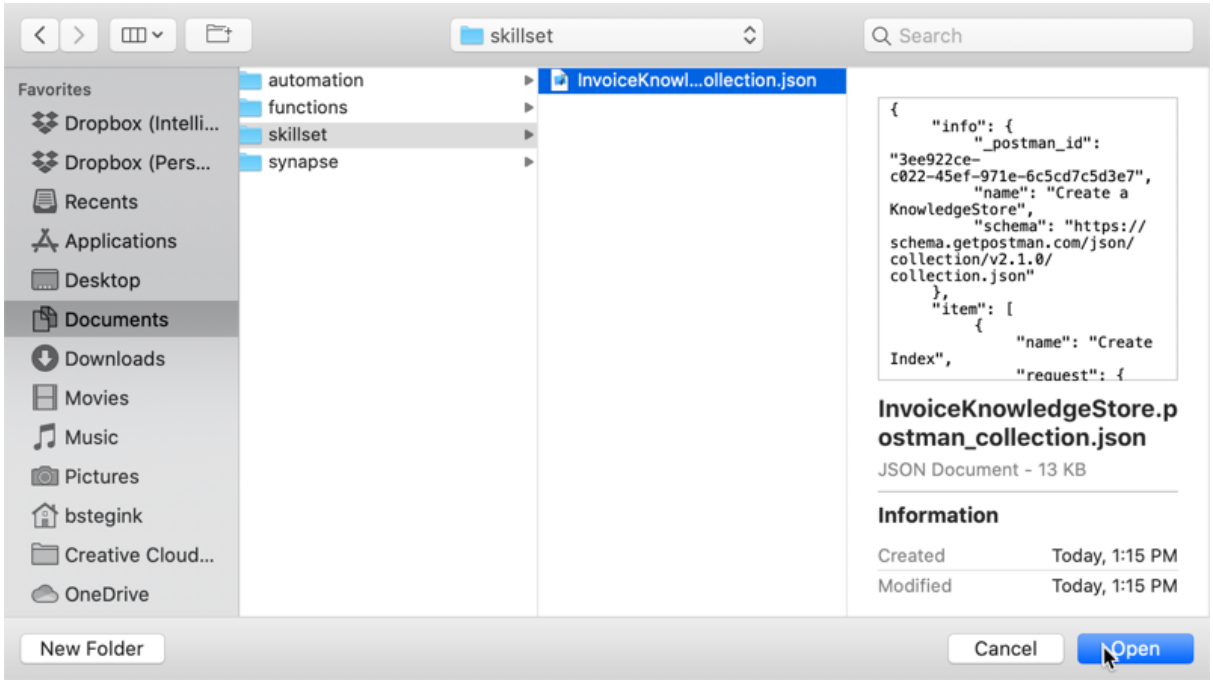


10. Now that we have the function published and all our resources created, we can create the skillset. This will be accomplished using **Postman**. Open Postman.

11. From the **File** menu, select **Import** and choose to import the postman collection from **Hands-on lab/environment-setup/skillset**.

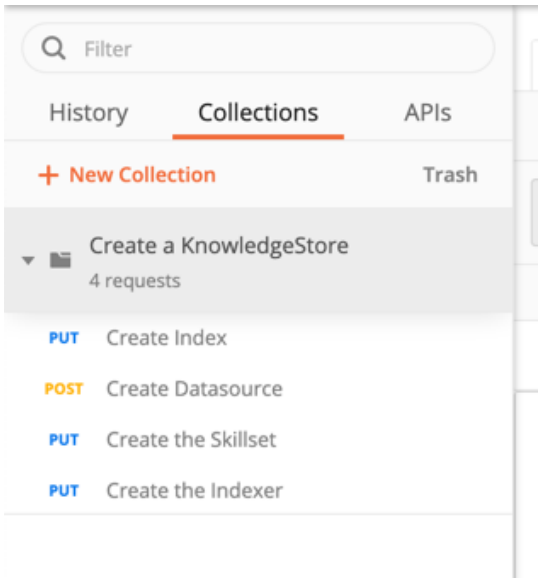




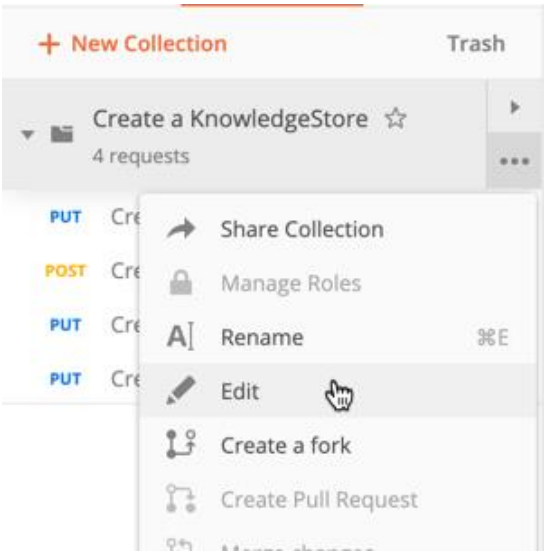


12. Select Import.

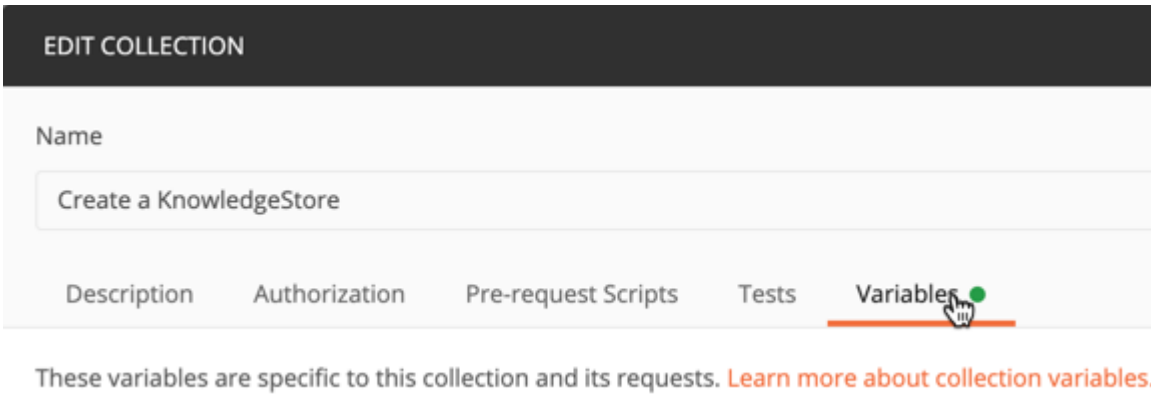
13. In Postman, the Collection that was imported will give you 4 items in the **Create a KnowledgeStore** collection. These are: Create Index, Create Datasource, Create the skillset, and Create the Indexer.



14. The first thing we need to do, is edit some properties that will affect each of the calls in the collection. Hover over the **Create a KnowledgeStore** collection, and select the ellipsis button **...**, and then select **Edit**.



15. In the Edit Collection screen, select the **Variables** tab.

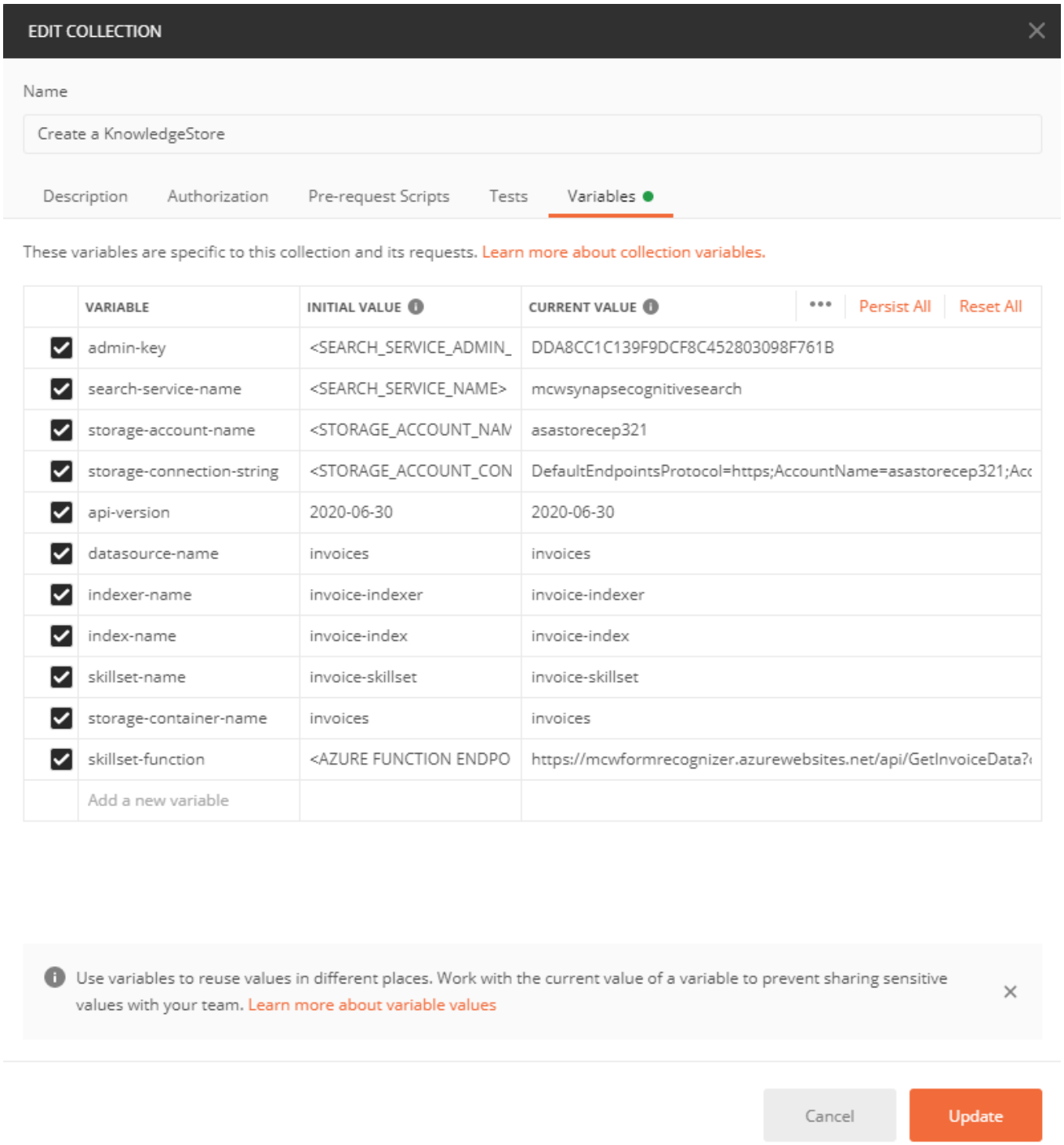


16. We are going to need to edit each one of these variables to match the following:

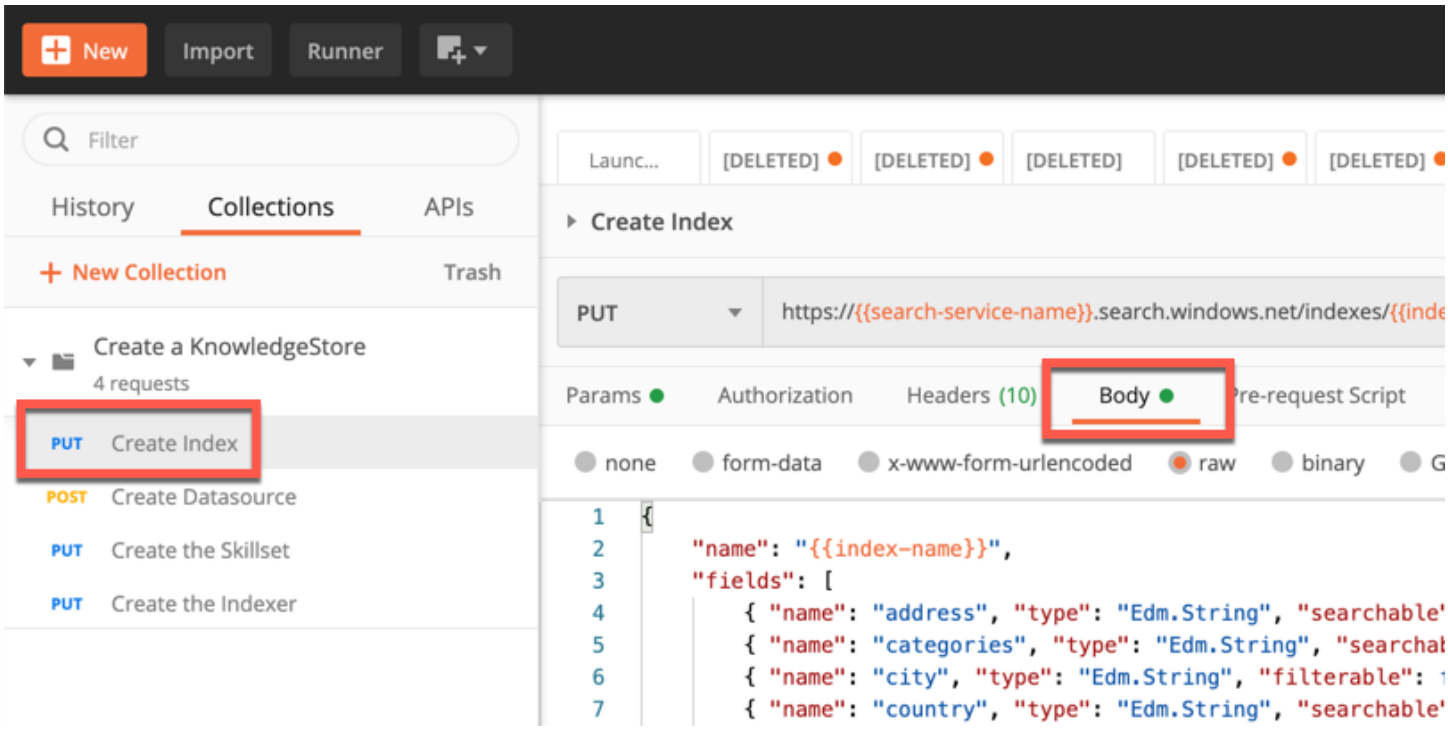
Variable	Value
admin-key	The key from the search service you created.
search-service-name	The name of the search service.
storage-account-name	asastore{{suffix}}
storage-connection-string	The connection string from the asastore{{suffix}} storage account.
datasourcename	Enter **invoices**
indexer-name	Enter **invoice-indexer**
index-name	Enter **invoice-index**

Variable	Value
skillset-name	Enter **invoice-skillset**
storage-container-name	Enter **invoices**
skillset-function	Enter function URL from the function you published.

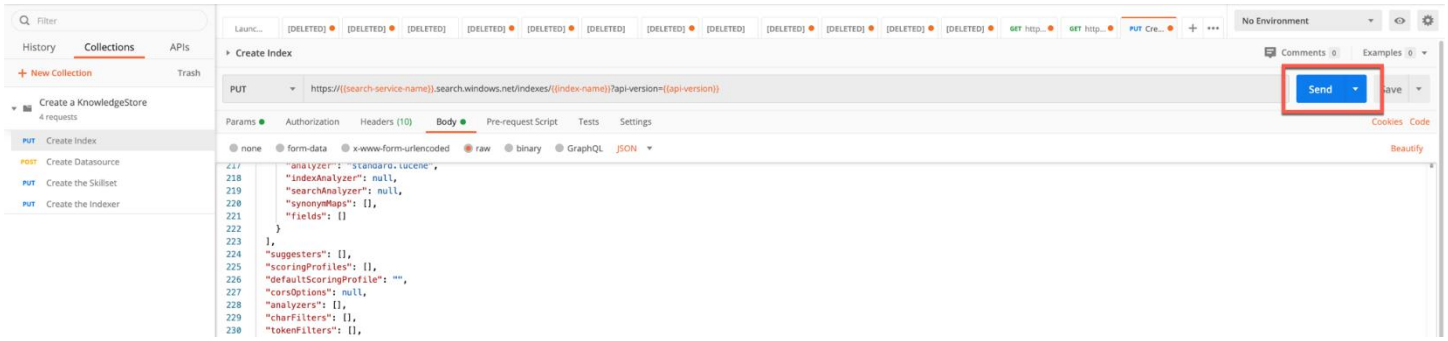
17. Select **Update** to update the collection with the modified values.



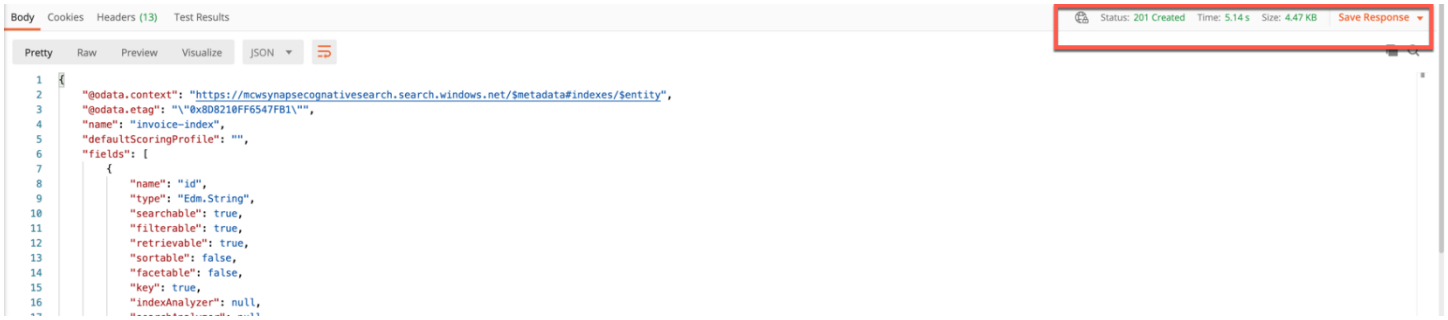
18. Select the **Create Index** call from the collection, then select the **Body** tab and review the content.



19. Select "Send".

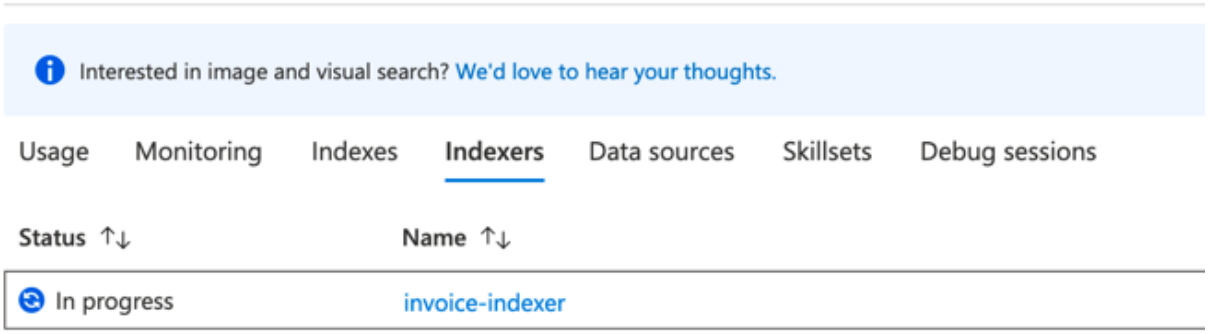


20. You should get a response that the index was created.

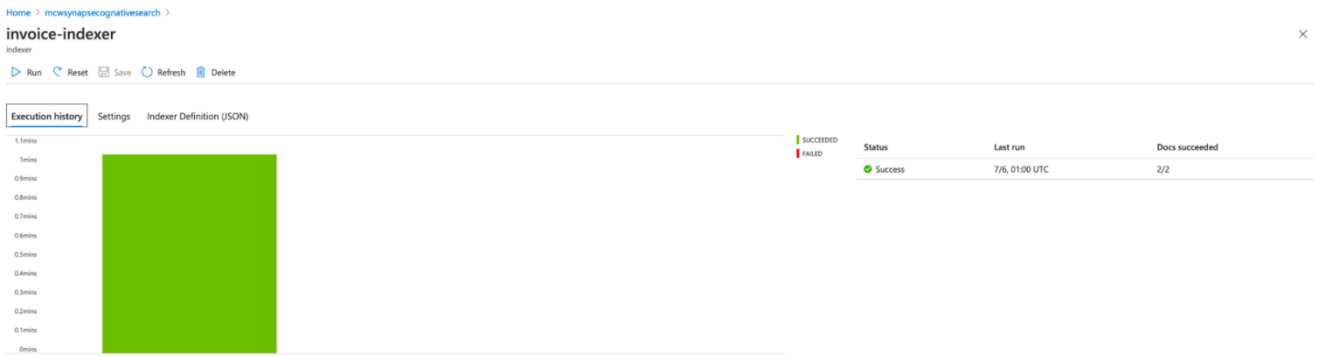


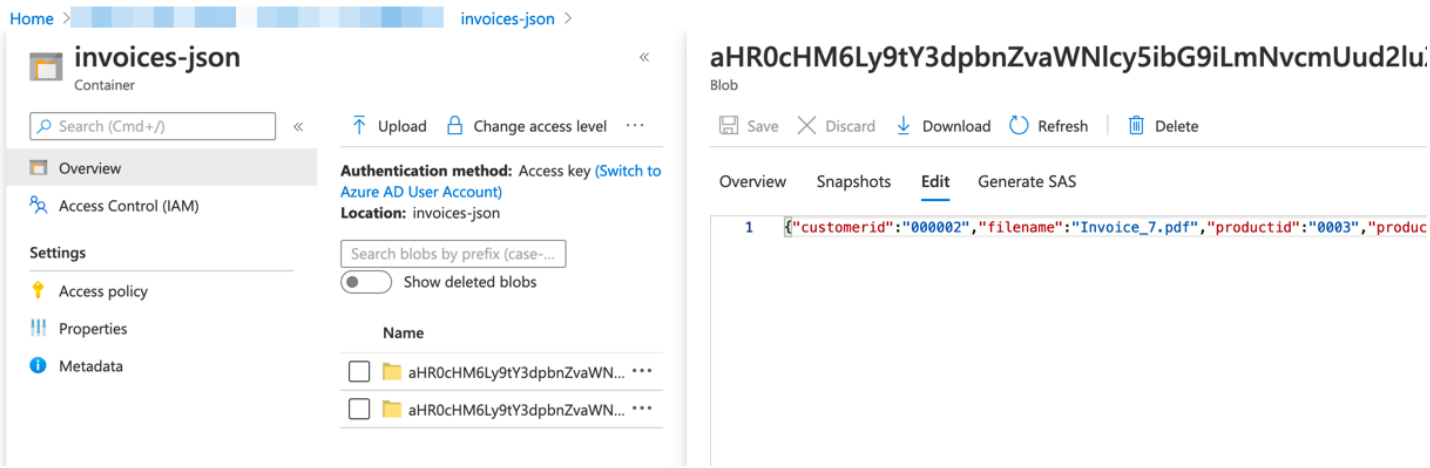
21. Do the same steps for the **Create Datasource, Create the Skillset, and Create the indexer** calls.

22. After you Send the Indexer request, if you navigate to your search service you should see your indexer running, indicated by the in-progress indicator. It will take a couple of minutes to run.



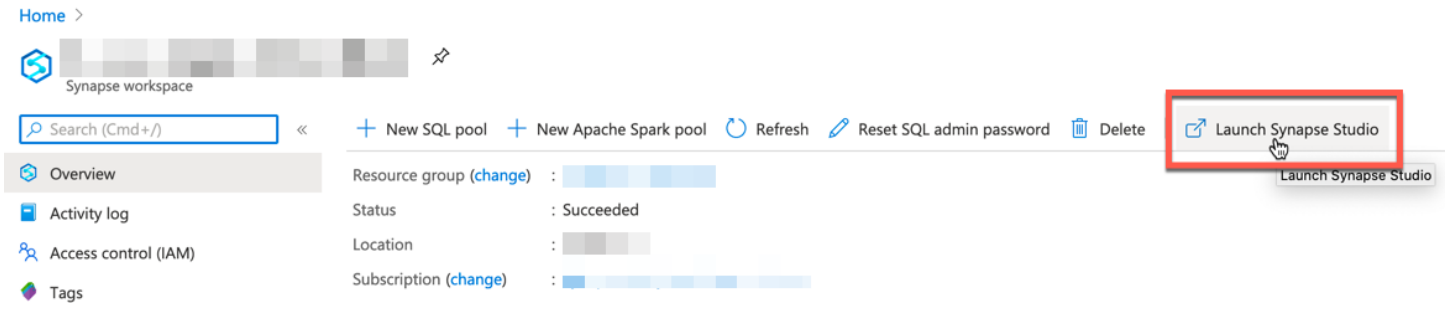
23. Once the indexer has run, it will show two successful documents. If you go to your Blob storage account, **asastore{suffix}** and look in the **invoices-json** container you will see two folders with .json documents in them.



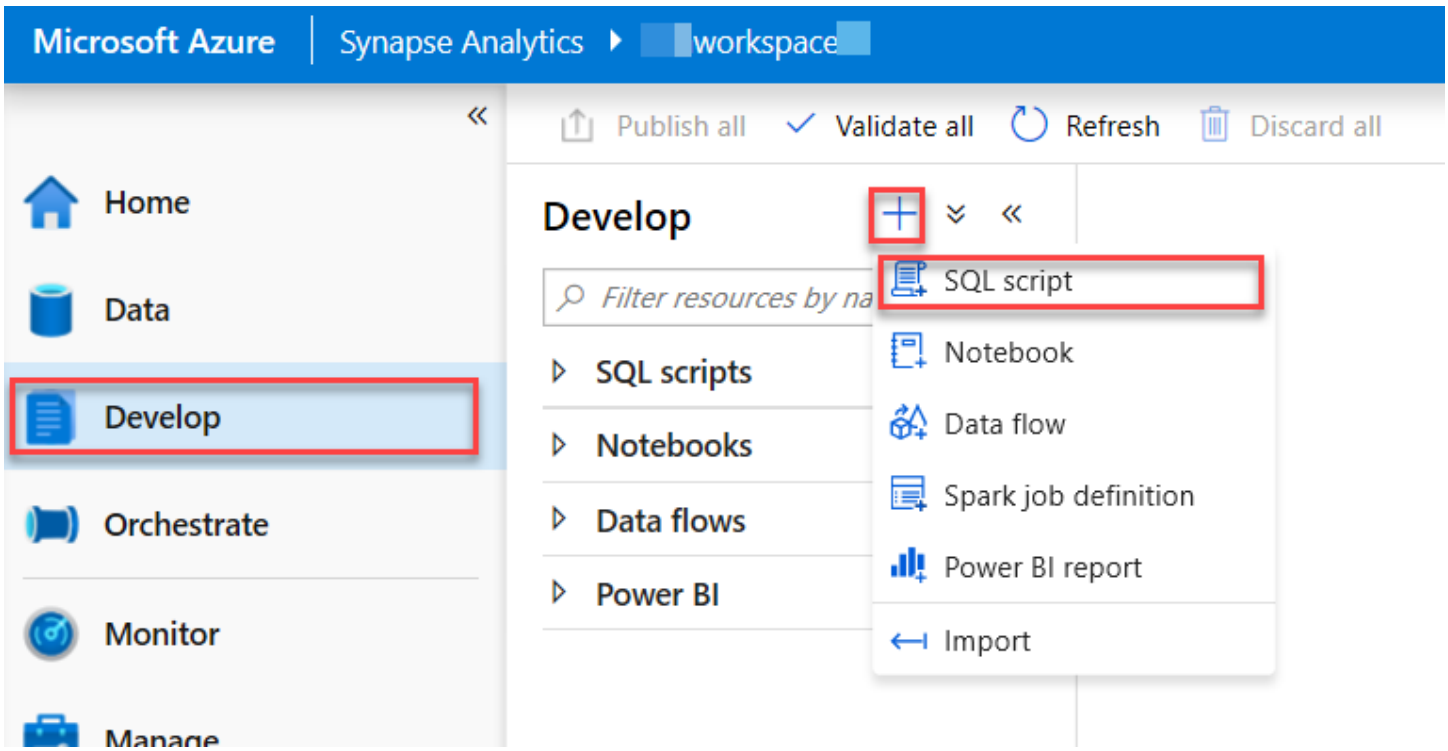


Task 4: Create the Synapse Pipeline

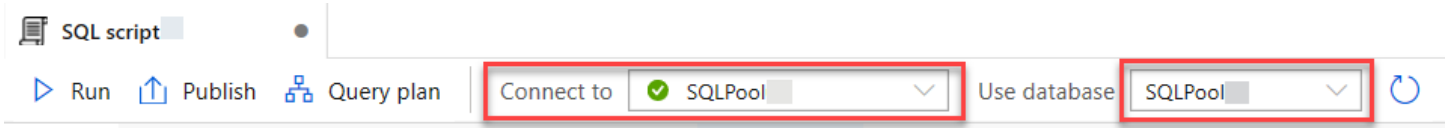
1. Open your Synapse workspace.



2. Expand the left menu and select the **Develop** item. From the **Develop** blade, expand the **+** button and select the **SQL script** item.



3. In the query tab toolbar menu, ensure you connect to your SQL Pool, `SQLPool01`.

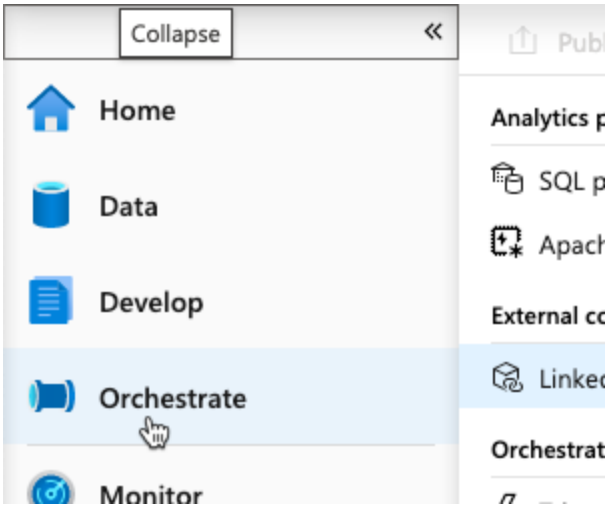


4. In the query window, copy and paste the following query to create the invoice information table. Then select the **Run** button in the query tab toolbar.

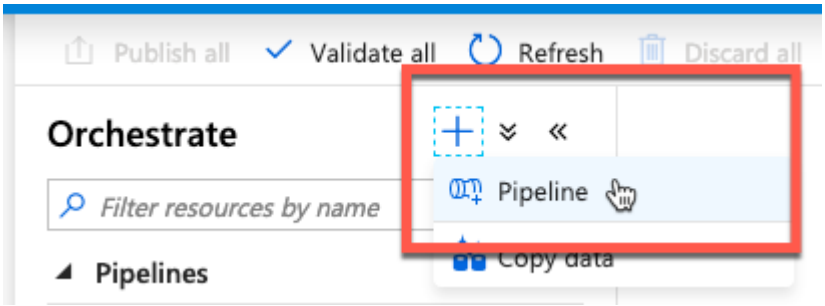
```
CREATE TABLE [wwi_mcw].[Invoices]
(
  [TransactionId] [uniqueidentifier]  NOT NULL,
  [CustomerId] [int]  NOT NULL,
  [ProductId] [smallint]  NOT NULL,
  [Quantity] [tinyint]  NOT NULL,
  [Price] [decimal](9,2)  NOT NULL,
  [TotalAmount] [decimal](9,2)  NOT NULL
);
```

5. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

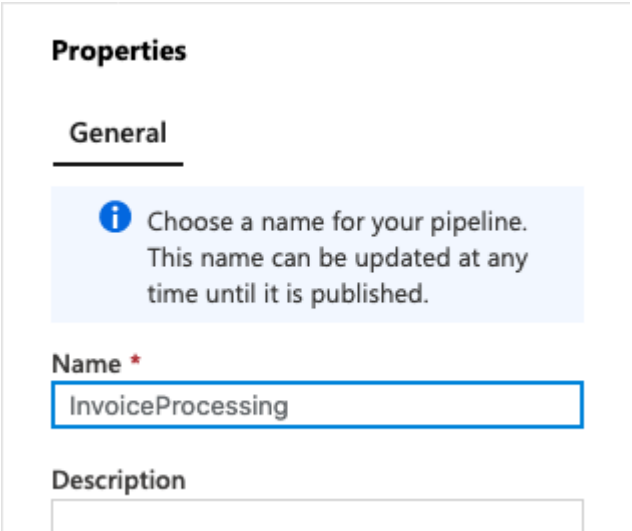
6. Select the **Orchestrate** hub from the left navigation.



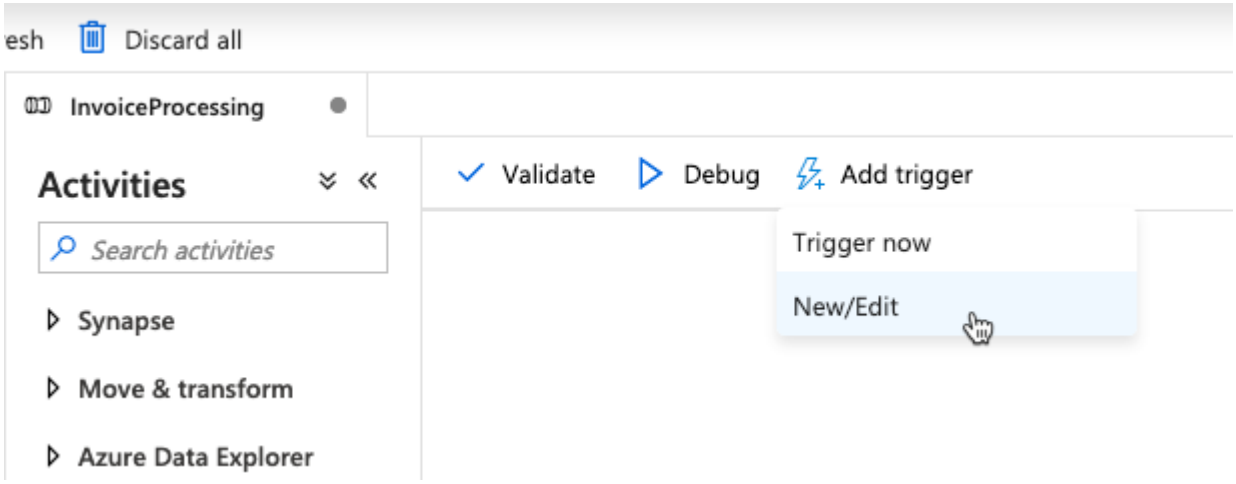
7. In the Orchestrate blade, expand the **+** button and then select **Pipeline** to create a new pipeline.



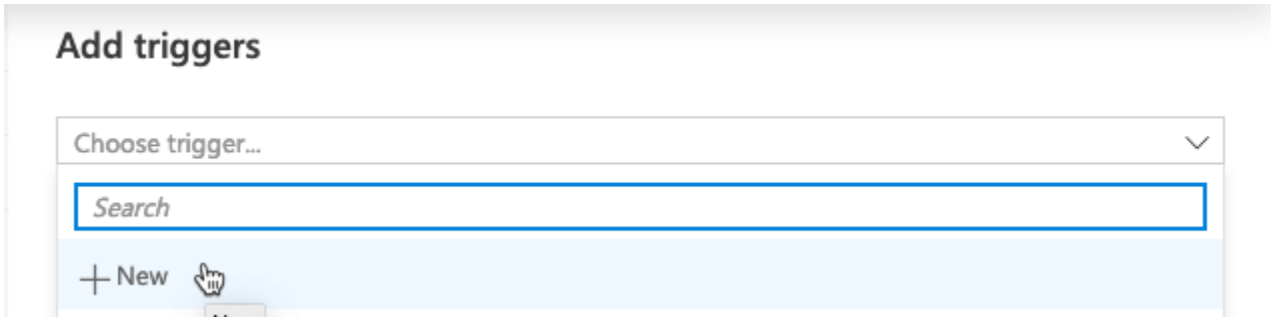
8. Name your pipeline **InvoiceProcessing**.



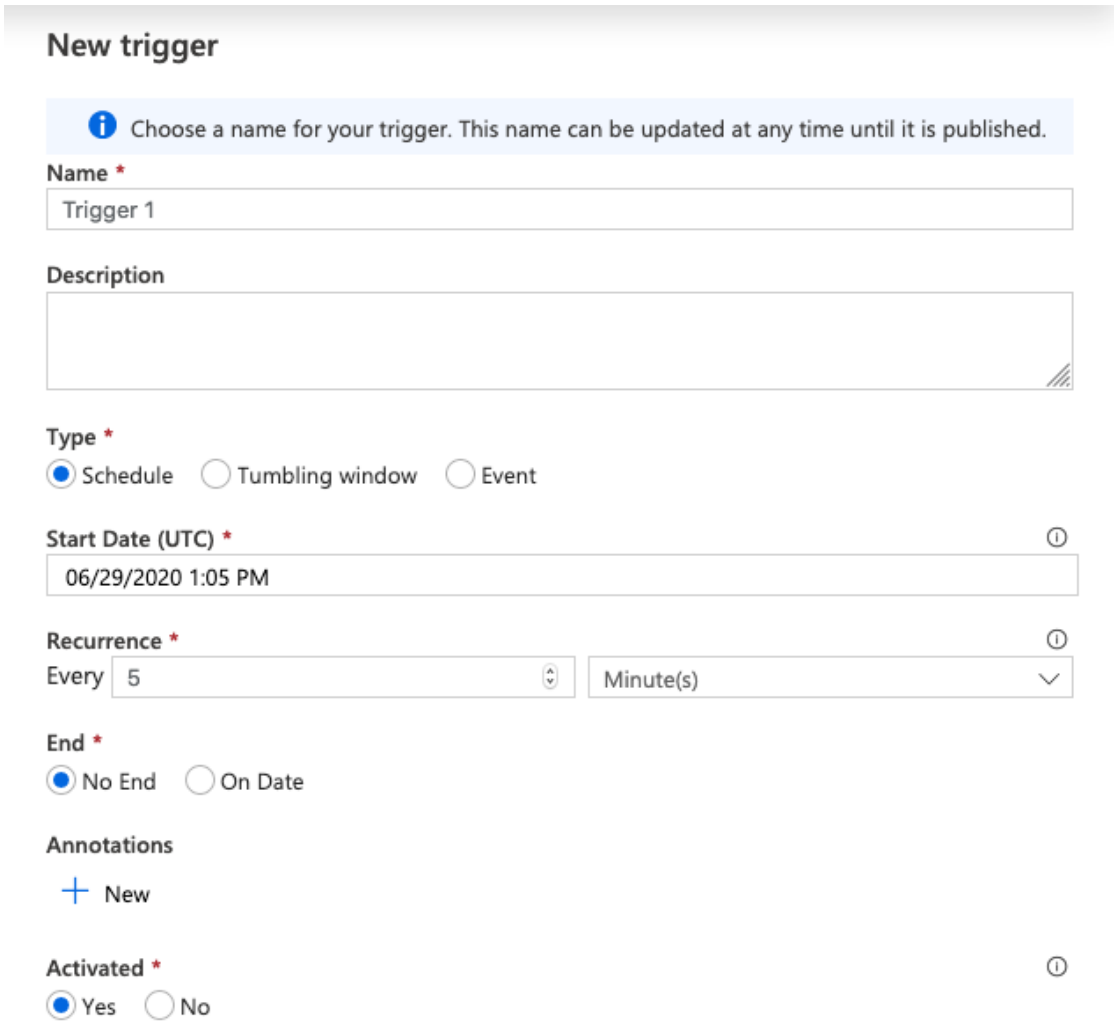
9. On the pipeline taskbar, select **Add trigger** then choose **New/Edit** to create an event to start the pipeline.



10. On the Add triggers form, select **+New** from the **Choose trigger** dropdown.

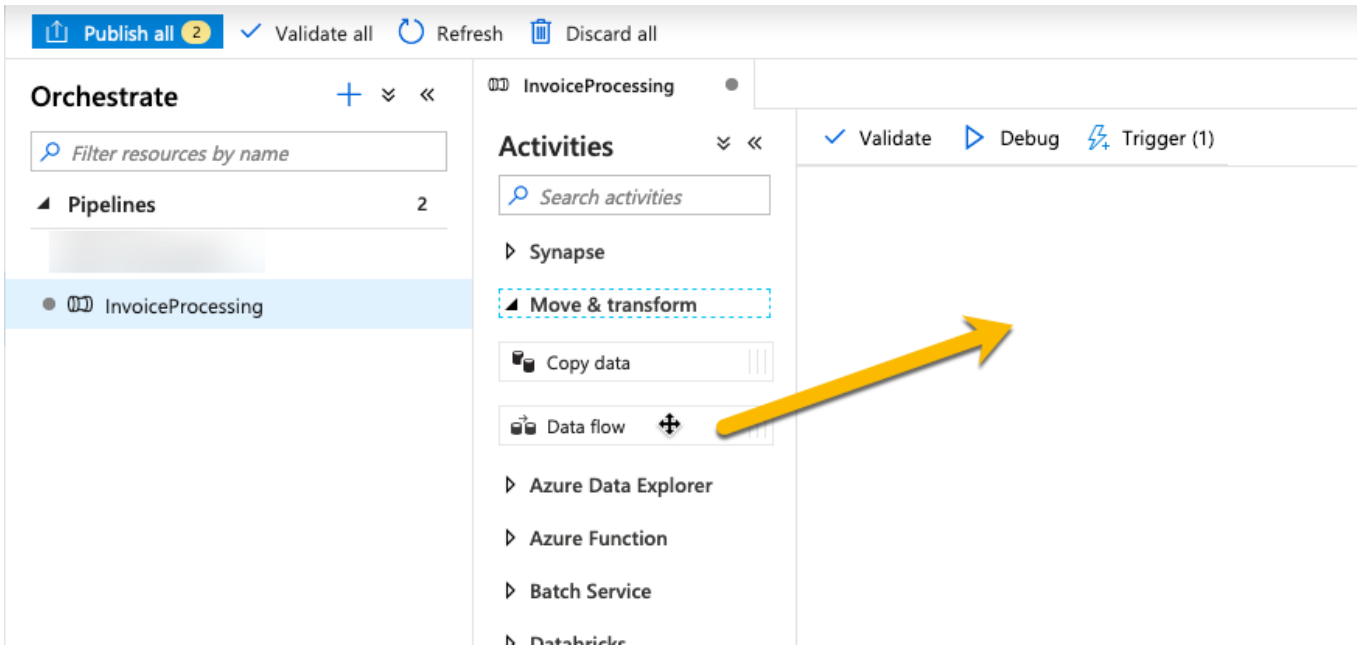


11. For this exercise, we're going to do a schedule. However, in the future you'll also be able to use an event-based trigger that would fire off new JSON files being added to blob storage. Set the trigger to start every 5 minutes, then select **OK**.

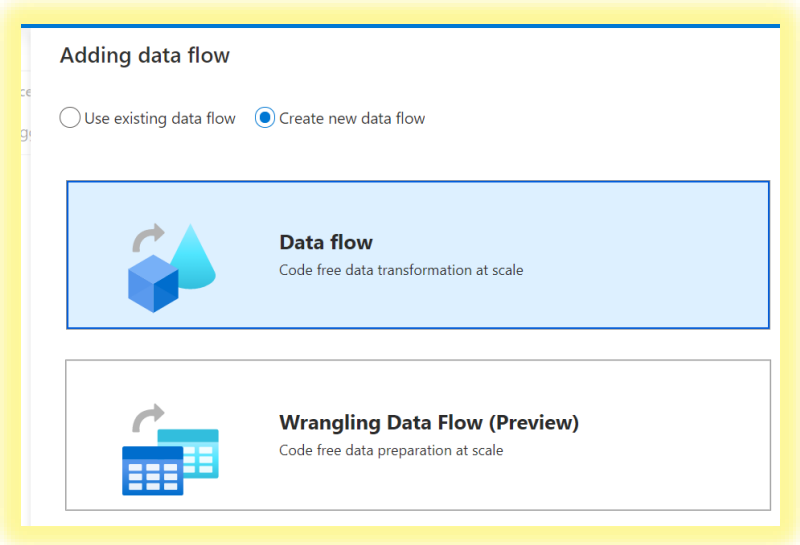


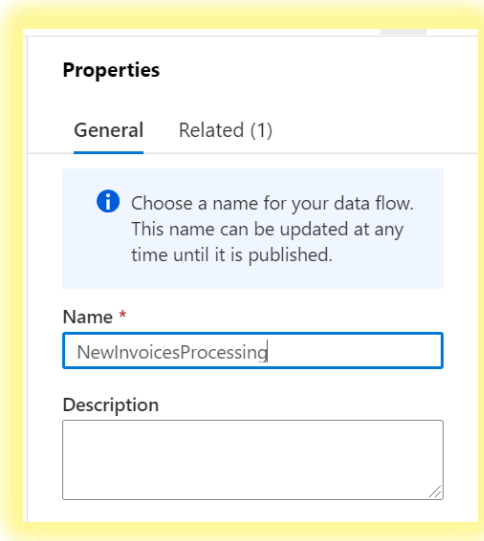
12. Select **OK** on the Run Parameters form, nothing needs to be done here.

13. Next, we need to add a Data Flow to the pipeline. Under Activities, expand **Move & transform** then drag and drop a **Data flow** onto the designer canvas.

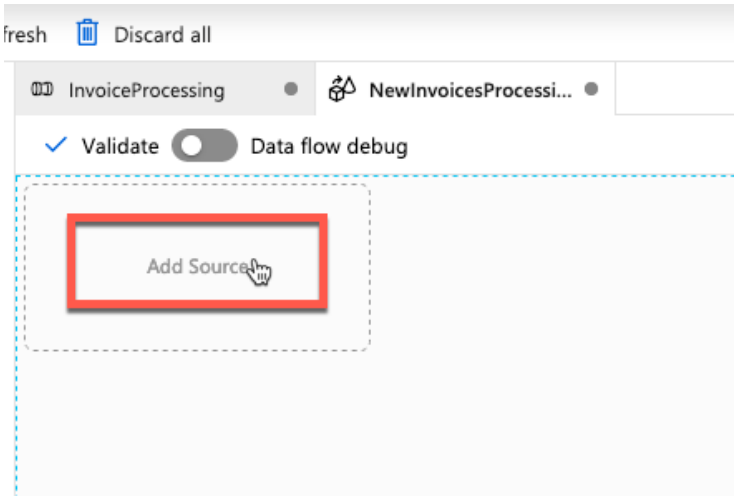


14. On the **Adding data flow** form, select **Create new data flow** and name it **NewInvoicesProcessing**.

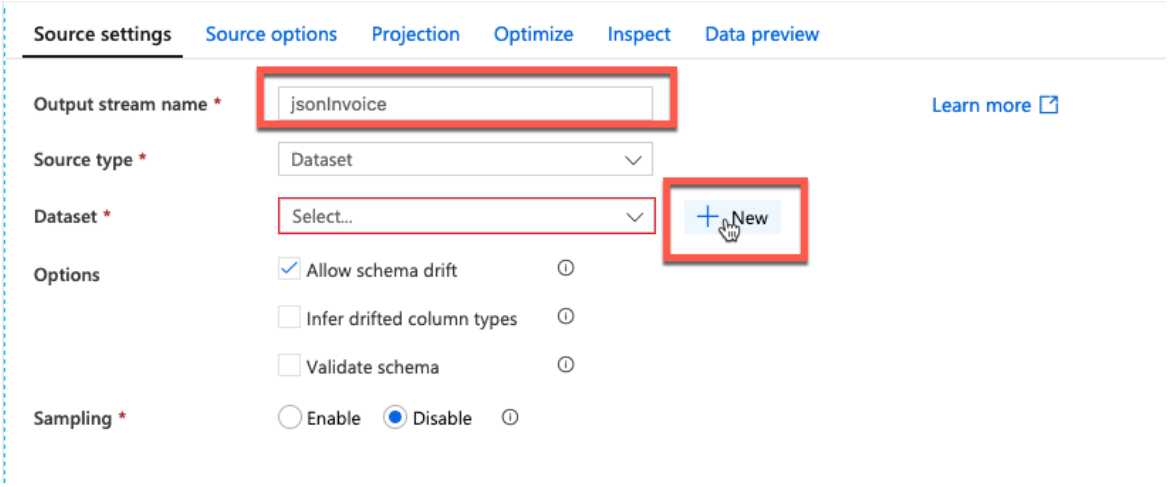




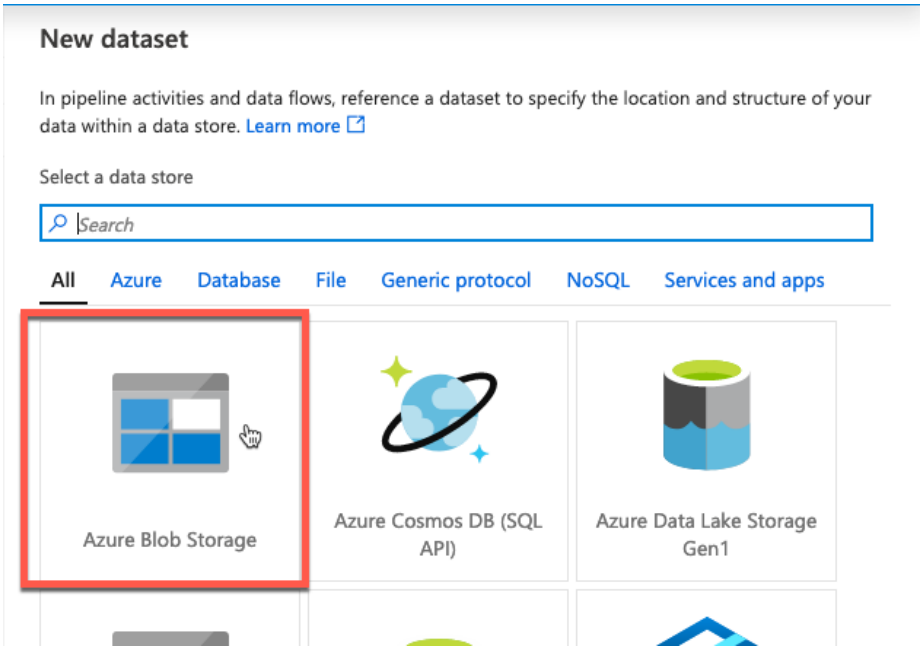
15. On the **NewInvoicesProcessing** data flow design canvas. Select the **Add source** box.



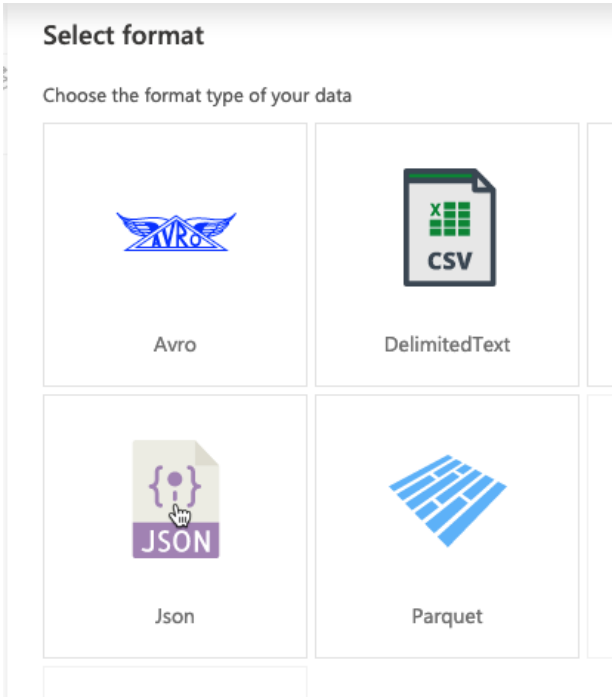
16. In the bottom pane, name the output stream **jsonInvoice**, leave the source type as **Dataset**, and keep all the remaining options set to their defaults. Select **+New** next to the Dataset field.



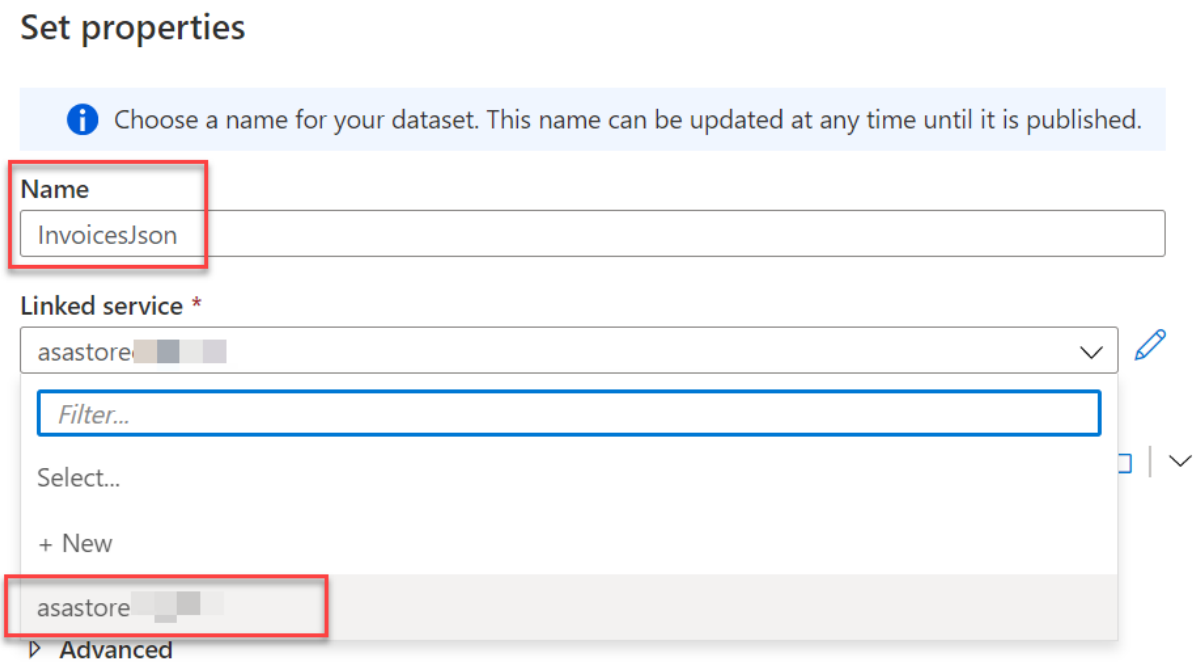
17. In the **New dataset blade**, select **Azure Blob Storage** then select **Continue**.



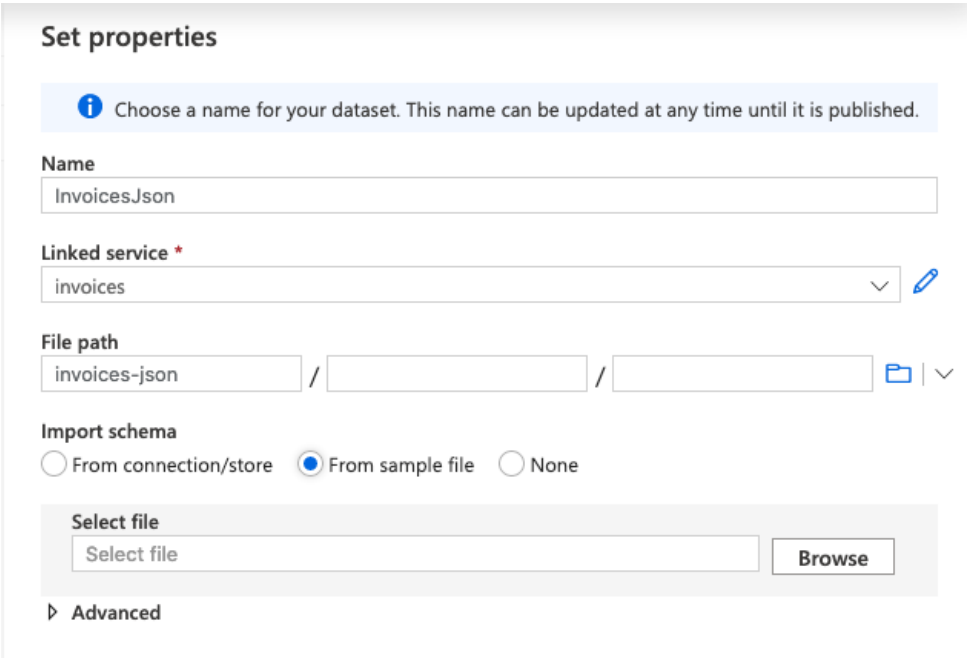
18. On the **Select format** blade, select **Json** then select **Continue**.



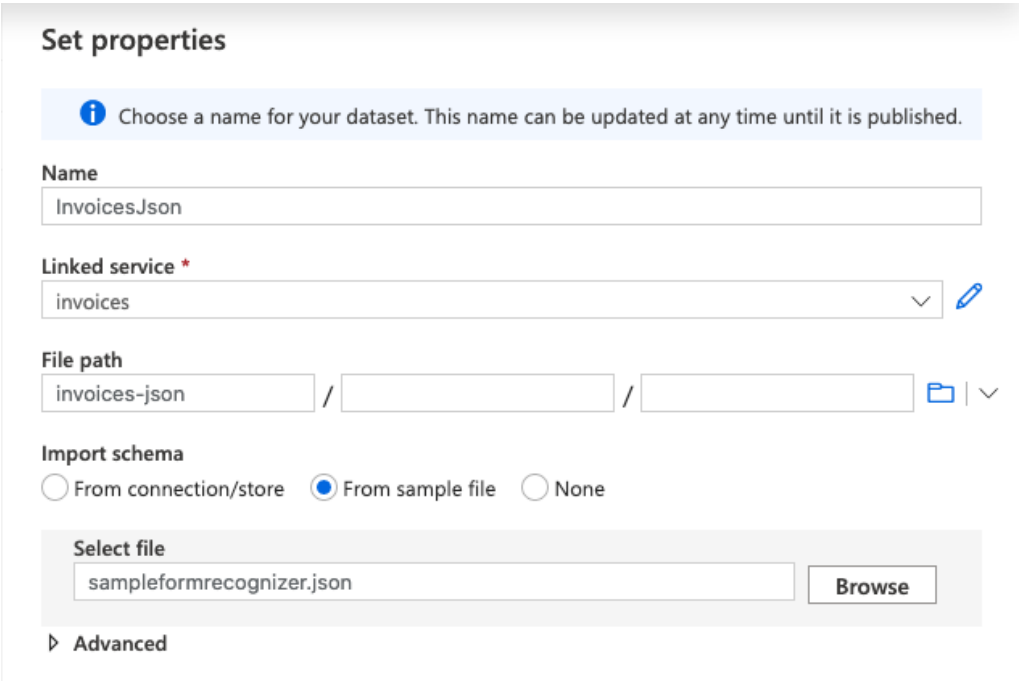
19. On the **Set properties** screen, name the dataset **InvoicesJson** then for the linked service field, choose the Azure Storage linked service **asastore{suffix}**.



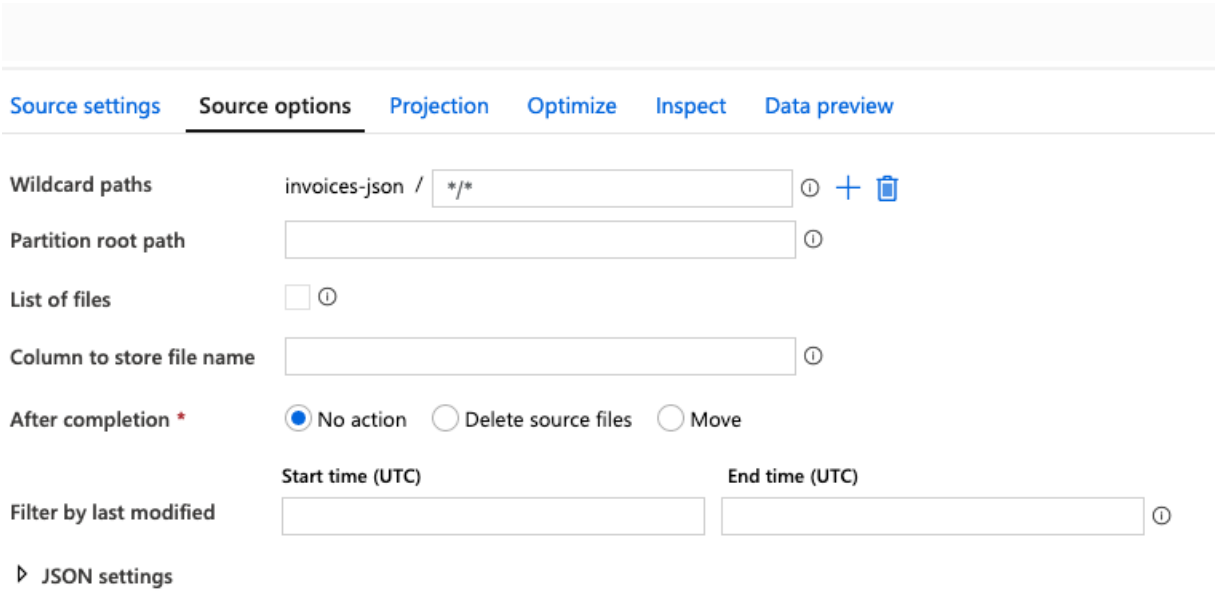
20. For the file path field, enter **invoices-json** and set the import schema field to **From sample file**.



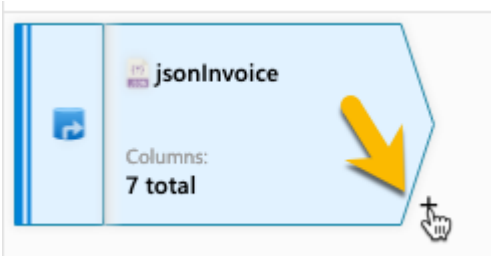
21. Select **Browse** and select the file located at **Hands-on lab/environment-setup/synapse/sampleformrecognizer.json** and select **OK**.



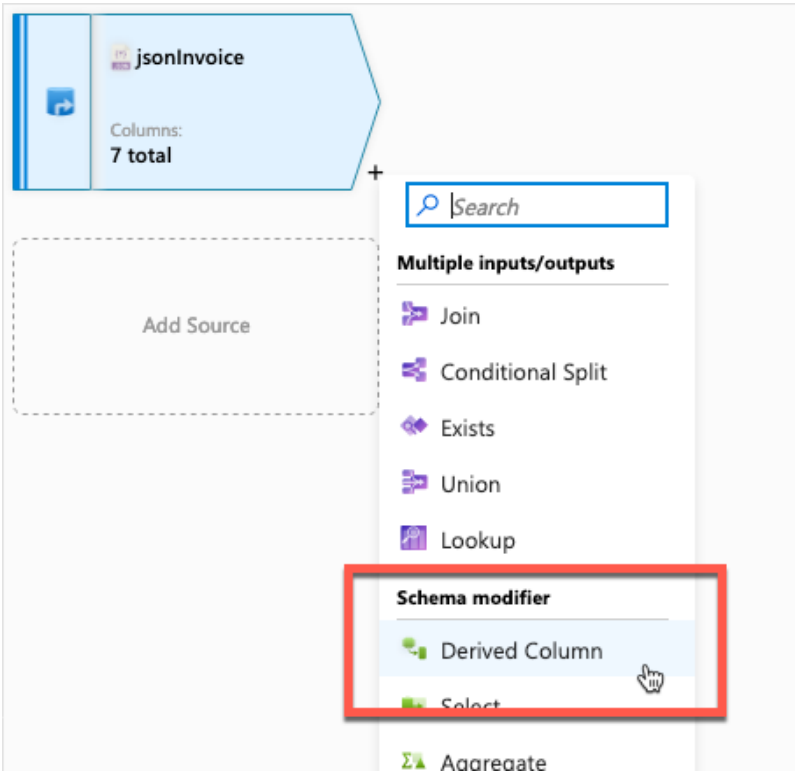
22. Select the **Source options** tab on the bottom pane. Add */* to the Wildcard paths field.



23. On the Data flow designer surface, select **+** to the lower right of the source activity to add another step in your data flow.

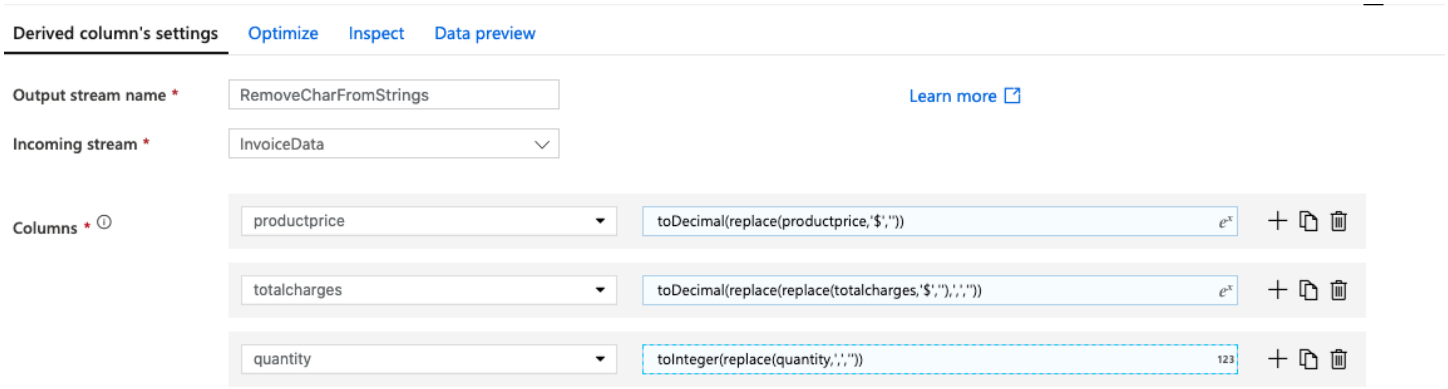


24. From the list of options, select **Derived column** from beneath the **Schema modifier** section.



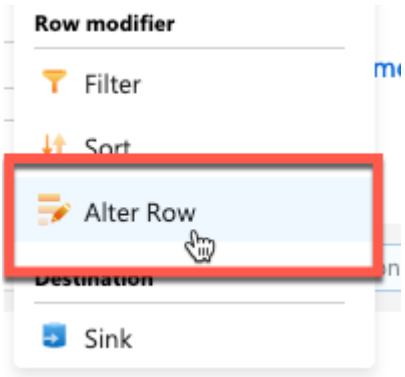
25. On the **Derived column's settings** tab, provide the output stream name of **RemoveCharFromStrings**. Then for the Columns field, add 3 columns and configure them as follows:

Column	Expression
productprice	toDecimal(replace(productprice,'$',"))
totalcharges	toDecimal(replace(replace(totalcharges,'$',"),',',"))
quantity	toInteger(replace(quantity,',',"))

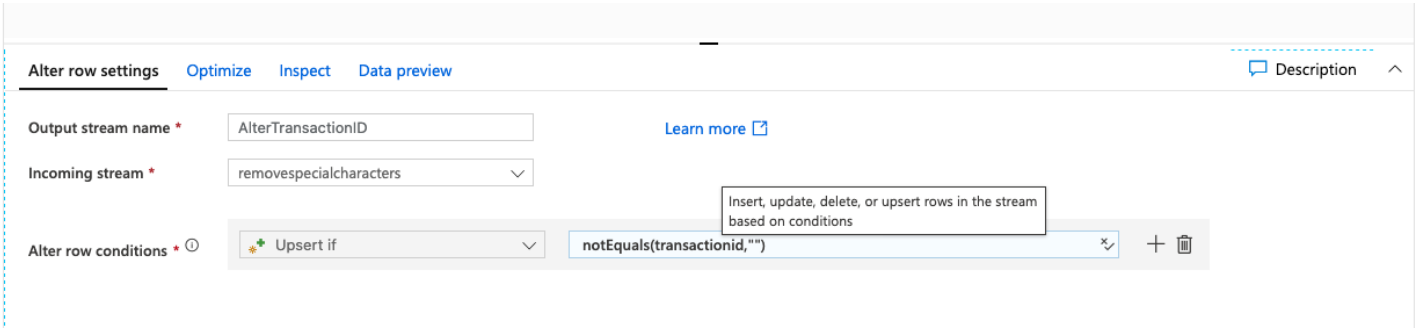


26. Return to the Data flow designer, select the **+** next to the derived column activity to add another step to your data flow.

27. This time select the **Alter Row** from beneath the **Row modifier** section.

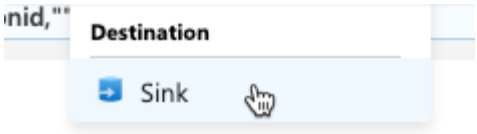


28. On the **Alter row settings** tab on the bottom pane, Name the Output stream **AlterTransactionID**, and leave the incoming stream set to the default value. Change **Alter row conditions** field to **Upsert If** and then set the expression to **notEquals(transactionid,"")**

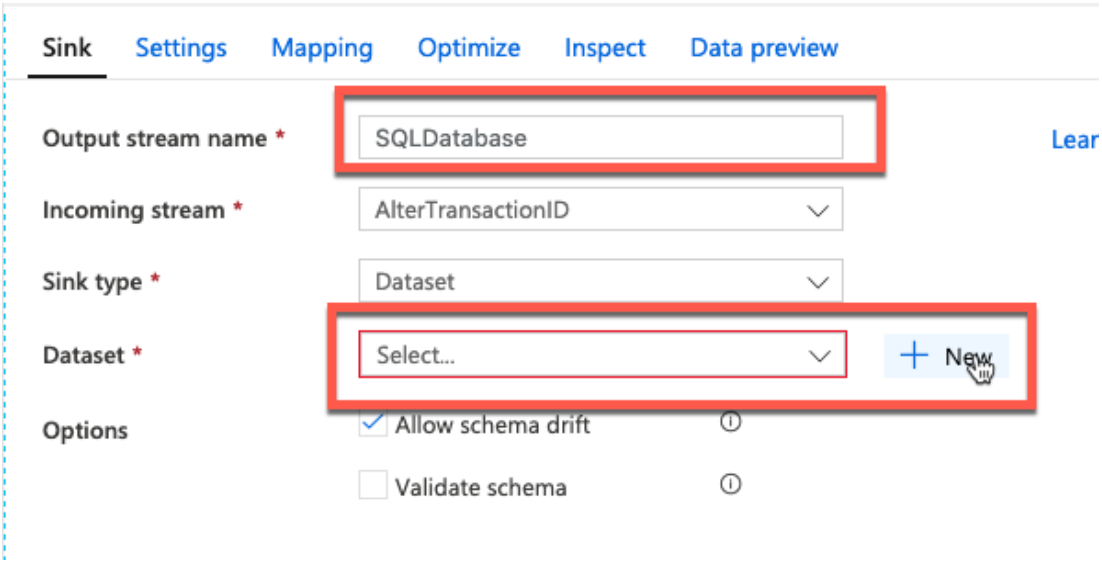


29. Return to the Data flow designer, select the **+** to the lower right of the **Alter Row** activity to add another step into your data flow.

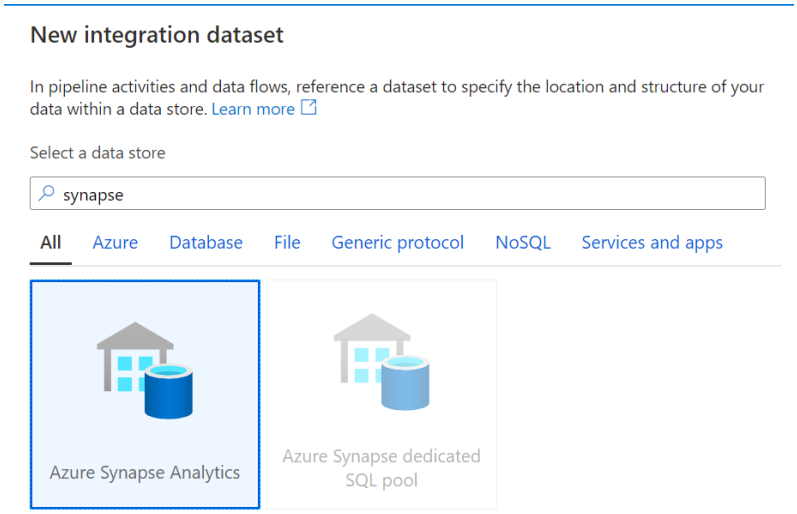
30. Within the **Destination** section, select **Sink**.



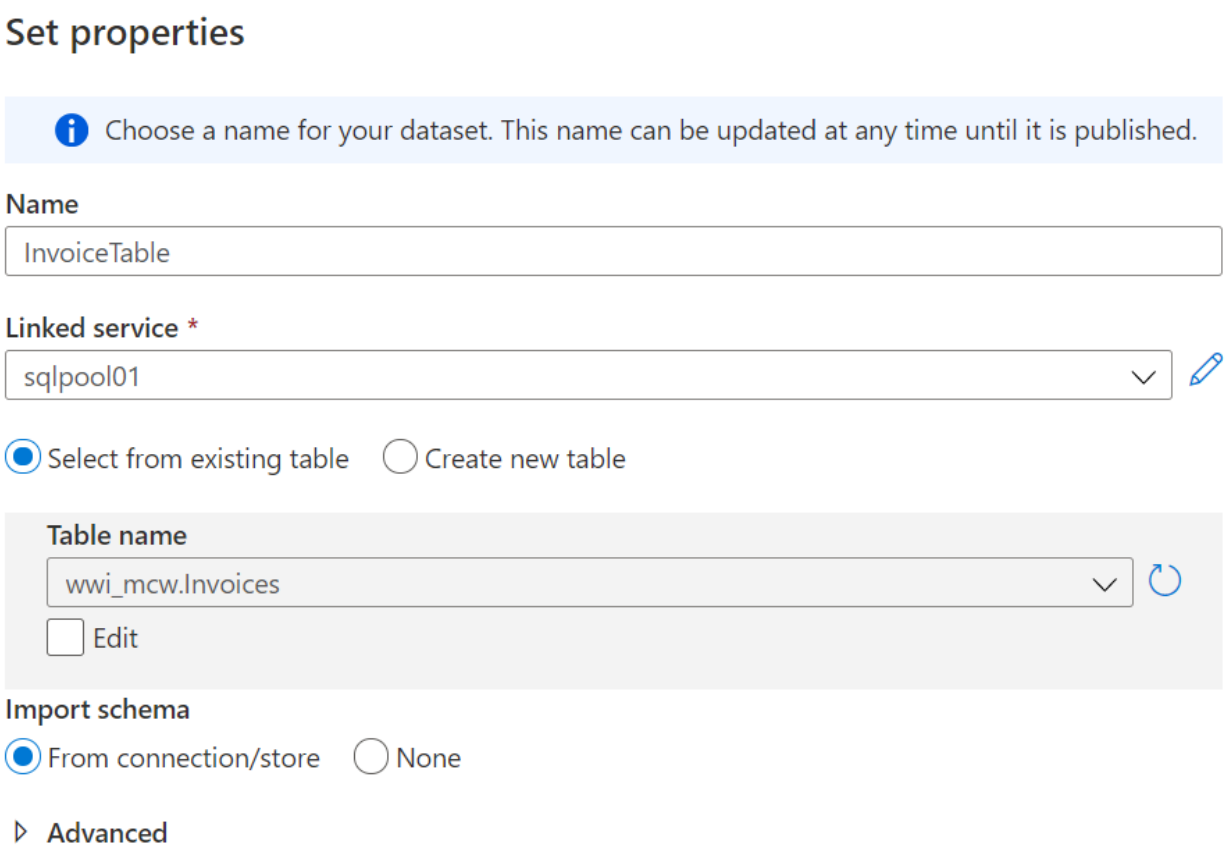
31. On the bottom pane, with the **Sink** tab selected, name the Output stream name **SQLDatabase** and leave everything else set to the default values. Next to the **Dataset** field, select **+New** to add a new Dataset.



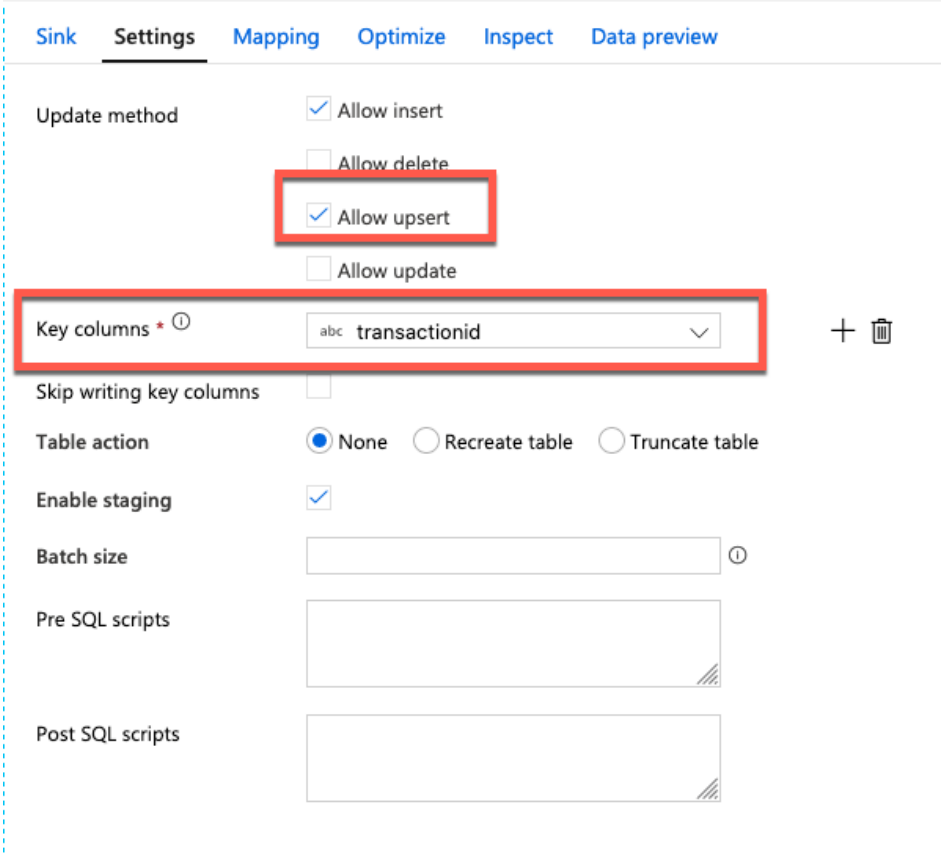
32. On the **New dataset** blade, select the **Azure** tab. Select **Azure Synapse Analytics** and select **Continue**.



33. Set the name of the Dataset to **InvoiceTable** and choose the **sqlpool01** Linked service. Choose **Select from existing table** and choose the **wwi_mcw.Invoices** table. If you don't see it in the list of your table names, select the **Refresh** button and it should show up. Select **OK**.

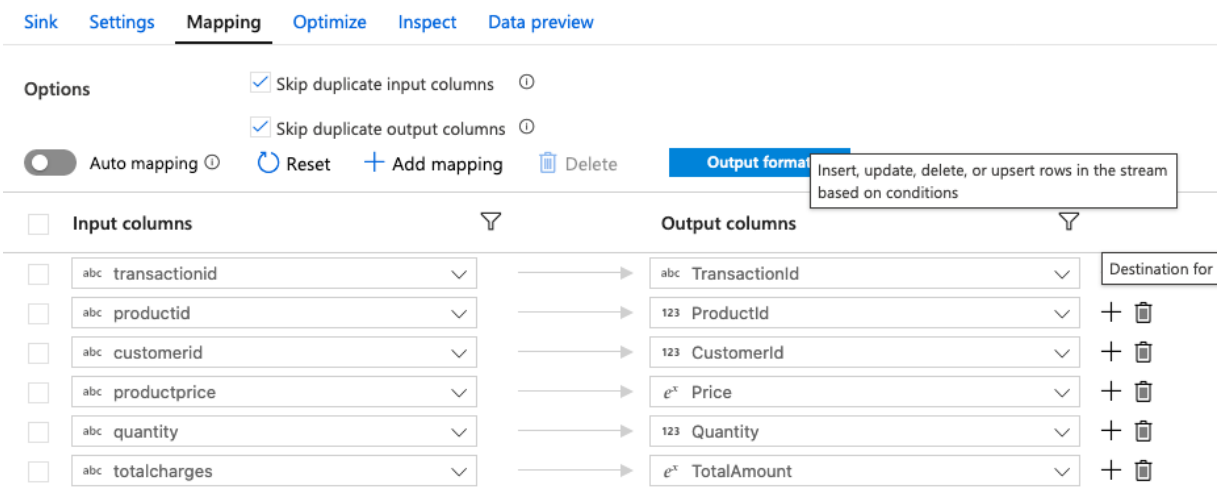


34. In the bottom pane, with the Sink activity selected on the data flow designer, select the **Settings** tab and check the box to **Allow upsert**. Set the **Key columns** field to **transactionid**.

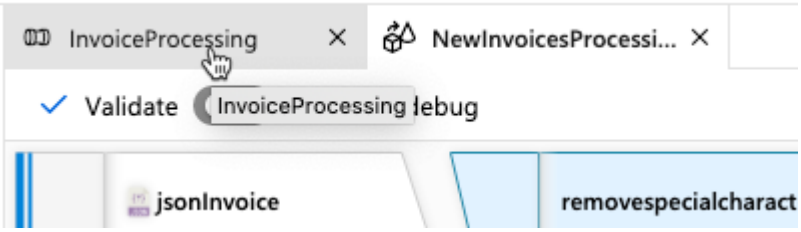


35. Select the **Mapping** tab, disable the **Auto mapping** setting and configure the mappings between the json file and the database. Select **+ Add mapping** then choose **Fixed mapping** to add the following mappings:

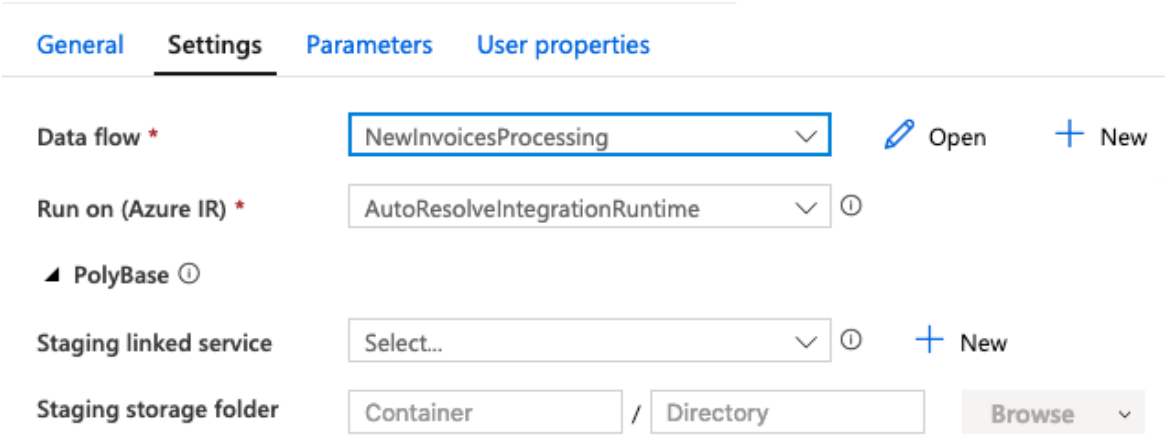
Input column	Output column
transactionid	TransactionId
productid	ProductId
customerid	CustomerId
productprice	Price
quantity	Quantity
totalcharges	TotalAmount



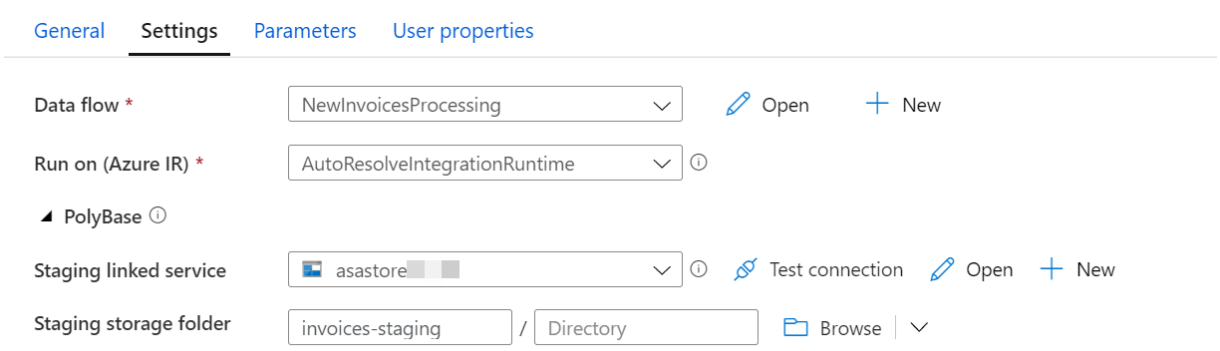
36. Return to the **InvoiceProcessing** pipeline by selecting its tab at the top of the workspace.



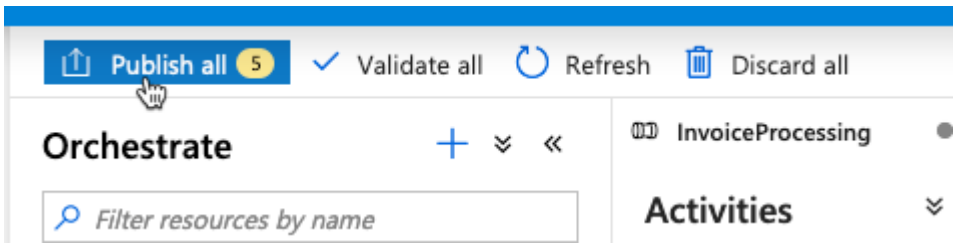
37. Select the data flow activity on the pipeline designer surface, then in the bottom pane, select the **Settings** tab.



38. Under the **PolyBase** settings, set the **Staging linked service** to the **asastore{suffix}** linked service. Enter **invoices-staging** as the **Storage staging folder**.

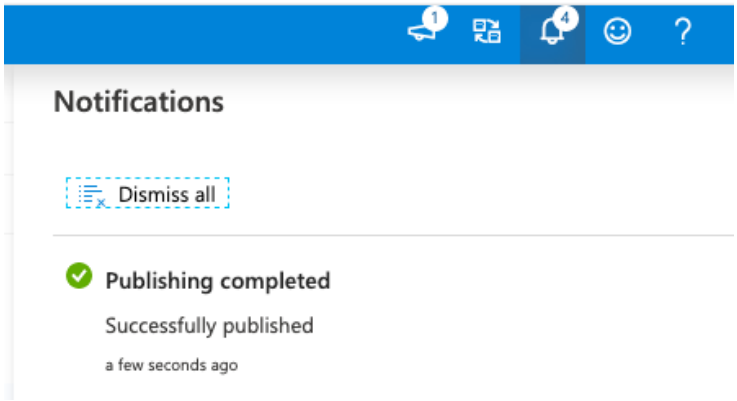


39. Select **Publish All** from the top toolbar.

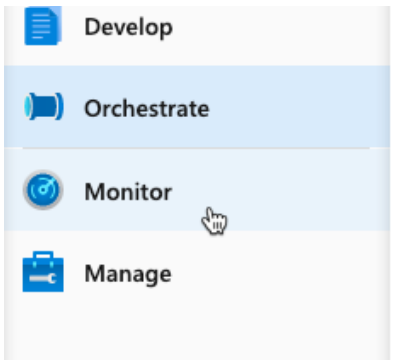


40. Select **Publish**.

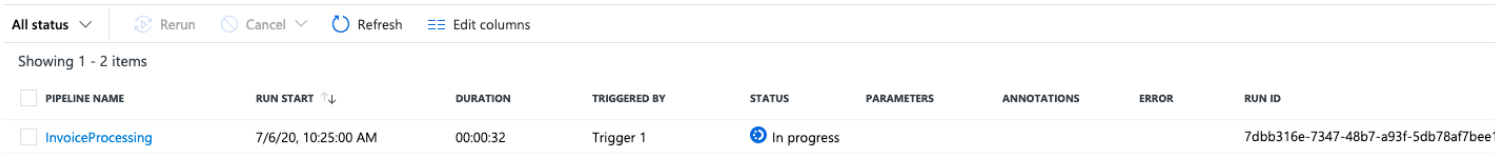
41. Within a few moments, you should see a notification that Publishing completed.



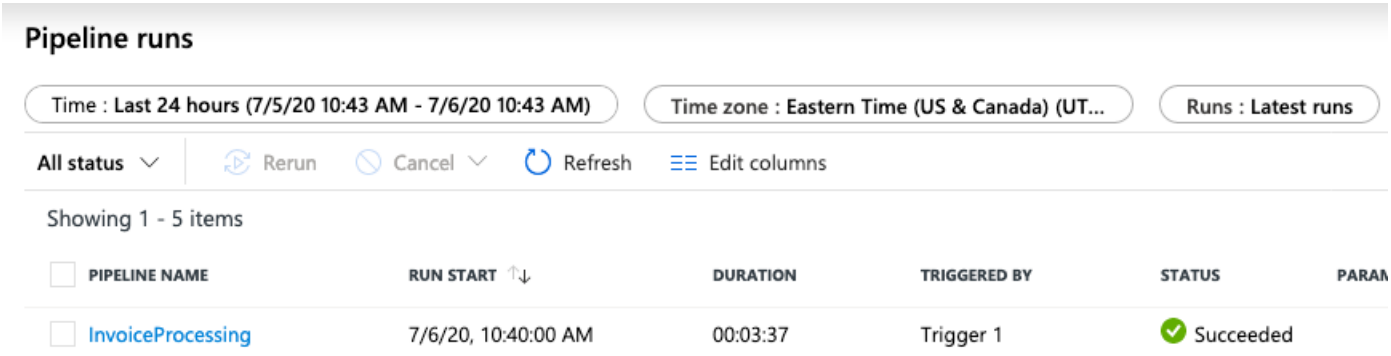
42. From the left menu, select the **Monitor** hub, then ensure the **Pipeline runs** option is selected from the hub menu.



43. In approximately 5 minutes, you should see the **InvoiceProcessing** pipeline begin processing. You may need to refresh this list to see it appear, a refresh button is located in the toolbar.

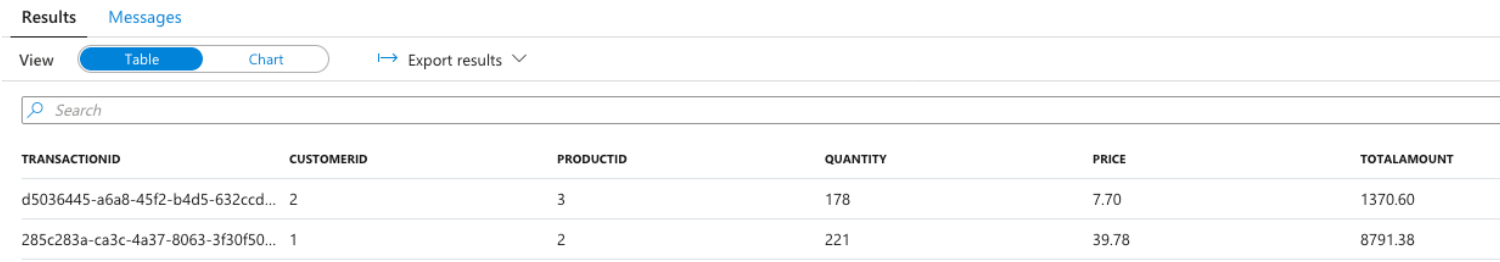


44. After about 3 or 4 minutes it will complete. You may need to refresh the list to see the completed pipeline.



45. From the left menu, select the **Develop** hub, then expand the **+** button an choose **SQL Script**. Ensure the proper database is selected, then run the following query to verify the data from the two test invoices.

```
SELECT * FROM wwi_mcw.Invoices
```



Exercise 6: Security

Duration: 30 minutes

Task 1: Column level security

It is important to identify data columns of that hold sensitive information. Types of sensitive information could be social security numbers, email addresses, credit card numbers, financial totals, and more. Azure Synapse Analytics allows you define permissions that prevent users or roles select privileges on specific columns.

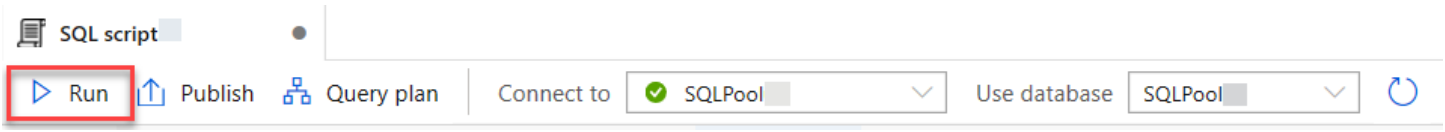
1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

2. Copy and paste the following query into the query window. Then, step through each statement group by highlighting all queries between each comment block in the query window, and

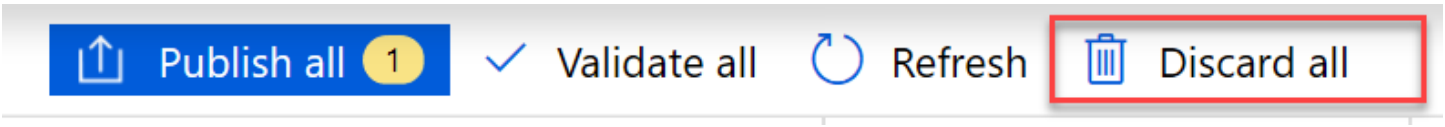
selecting **Run** from the query window toolbar menu. The query is documented inline. Ensure you are connected to **SQLPool01** when running the queries.

```
/* Column-
level security feature in Azure Synapse simplifies the design and coding of security in applications.
    It ensures column level security by restricting column access to protect sensitive data. */

/* Scenario: In this scenario we will be working with two users. The first one is the CEO, he has access
to all
data. The second one is DataAnalystMiami, this user doesn't have access to the confidential Revenue
Column in the CampaignAnalytics table. Follow this lab, one step at a time to see how Column-
level security removes access to the
    Revenue column to DataAnalystMiami */

--Step 1: Let us see how this feature in Azure Synapse works.
-- Before that let us have a look at the Campaign Analytics table.
select  Top 100 * from wwi_mcw.CampaignAnalytics
where City is not null and state is not null

/* Consider a scenario where there are two users.
    A CEO, who is an authorized  personnel with access to all the information in the database
    and a Data Analyst, to whom only required information should be presented.*/

SELECT Name as [User1] FROM sys.sysusers WHERE name = N'CEO';
SELECT Name as [User2] FROM sys.sysusers WHERE name = N'DataAnalystMiami';

-- Step:3 Now let us enforce column level security for the DataAnalystMiami.
/* The CampaignAnalytics table in the warehouse has information like ProductID, Analyst, CampaignName, Qua
ntity, Region, State, City, RevenueTarget and Revenue.
    The Revenue generated from every campaign is classified and should be hidden from DataAnalystMiami.
*/

REVOKE SELECT ON wwi_mcw.CampaignAnalytics FROM DataAnalystMiami;
GRANT SELECT ON wwi_mcw.CampaignAnalytics([Analyst], [CampaignName], [Region], [State], [City], [RevenueTa
rget]) TO DataAnalystMiami;
-- This provides DataAnalystMiami access to all the columns of the Sale table but Revenue.

-- Step:4 Then, to check if the security has been enforced,
-- we execute the following query with current User As 'DataAnalystMiami', this will result in an error
-- since DataAnalystMiami doesn't have select access to the Revenue column
EXECUTE AS USER ='DataAnalystMiami';
select TOP 100 * from wwi_mcw.CampaignAnalytics;

-- The following query will succeed since we are not including the Revenue column in the query.
EXECUTE AS USER ='DataAnalystMiami';
select [Analyst],[CampaignName], [Region], [State], [City], [RevenueTarget] from wwi_mcw.CampaignAnalytics
;

EXECUTE AS USER ='DataAnalystMiami';
select [Analyst],[CampaignName], [Region], [State], [City], [RevenueTarget], [Revenue] from wwi_mcw.Campai
gnAnalytics;
```

```
-- Step:5 Whereas, the CEO of the company should be authorized with all the information present in the
-- warehouse. To do so, we execute the following query.
Revert;
GRANT SELECT ON wwi_mcw.CampaignAnalytics TO CEO;  --Full access to all columns.

-- Step:6 Let us check if our CEO user can see all the information that is present.
-- Assign Current User As 'CEO' and the execute the query
EXECUTE AS USER ='CEO'
select * from wwi_mcw.CampaignAnalytics
Revert;
```



3. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Task 2: Row level security

In many organizations it is important to filter certain rows of data by user. In the case of WWI, they wish to have data analysts only see their data. In the campaign analytics table, there is an Analyst column that indicates to which analyst that row of data belongs. In the past, organizations would create views for each analyst - this was a lot of work and unnecessary overhead. Using Azure Synapse Analytics, you can define row level security that compares the user executing the query to the Analyst column, filtering the data so they only see the data destined for them.

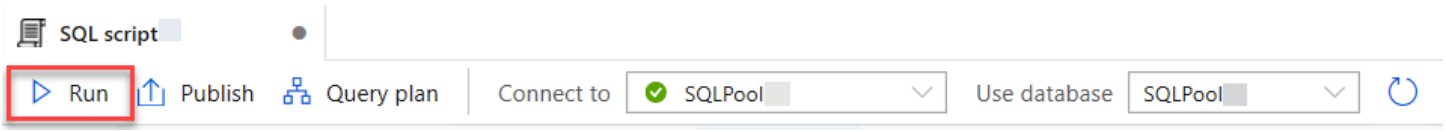
1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

2. Copy and paste the following query into the query window. Then, step through each statement group by highlighting all queries between each comment block in the query window, and selecting **Run** from the query window toolbar menu. The query is documented inline.

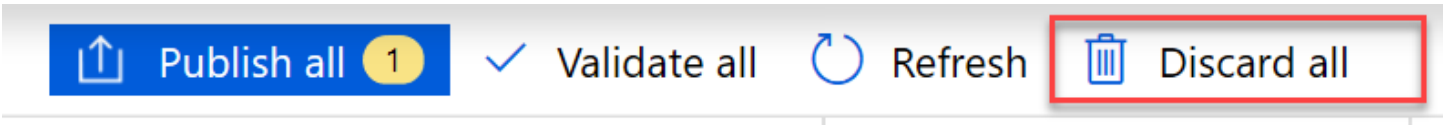
```
/* Row level Security (RLS) in Azure Synapse enables us to use group membership to control acc
ess to rows in a table.
Azure Synapse applies the access restriction every time the data access is attempted from any
user.
Let see how we can implement row level security in Azure Synapse.*/

-- Row-Level Security (RLS), 1: Filter predicates

-- Step:1 The Sale table has two Analyst values: DataAnalystMiami and DataAnalystSanDiego.
-- Each analyst has jurisdiction across a specific Region.
-- DataAnalystMiami on the South East Region and DataAnalystSanDiego on the Far West region.
```

```sql
SELECT DISTINCT Analyst, Region FROM wwi_mcw.CampaignAnalytics order by Analyst ;

/* Scenario: WWI requires that an Analyst only see the data for their own data from their own
region. The CEO should see ALL data.
    In the Sale table, there is an Analyst column that we can use to filter data to a specific
 Analyst value. */

/* We will define this filter using what is called a Security Predicate. This is an inline tab
le-valued function that allows
    us to evaluate additional logic, in this case determining if the Analyst executing the que
ry is the same as the Analyst
    specified in the Analyst column in the row. The function returns 1 (will return the row) w
hen a row in the Analyst column is the same as the user executing the query (@Analyst = USER_N
AME()) or if the user executing the query is the CEO user (USER_NAME() = 'CEO')
    whom has access to all data.
*/

-- Review any existing security predicates in the database
SELECT * FROM sys.security_predicates

-- Step:2 Create a new Schema to hold the security predicate,
-- then define the predicate function. It returns 1 (or True)
-- when a row should be returned in the parent query.
GO
CREATE SCHEMA Security
GO
CREATE FUNCTION Security.fn_securitypredicate(@Analyst AS sysname)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
    WHERE @Analyst = USER_NAME() OR USER_NAME() = 'CEO'
GO

-- Now we define security policy that adds the filter predicate to the Sale table.
-- This will filter rows based on their login name.
CREATE SECURITY POLICY SalesFilter
ADD FILTER PREDICATE Security.fn_securitypredicate(Analyst)
ON wwi_mcw.CampaignAnalytics
WITH (STATE = ON);

------ Allow SELECT permissions to the Sale Table.------
GRANT SELECT ON wwi_mcw.CampaignAnalytics TO CEO, DataAnalystMiami, DataAnalystSanDiego;

-- Step:3 Let us now test the filtering predicate,
-- by selecting data from the Sale table as 'DataAnalystMiami' user.
EXECUTE AS USER = 'DataAnalystMiami'
SELECT * FROM wwi_mcw.CampaignAnalytics;
revert;
-- As we can see, the query has returned rows here Login name is DataAnalystMiami
```

```
-- Step:4 Let us test the same for  'DataAnalystSanDiego' user.
EXECUTE AS USER = 'DataAnalystSanDiego';
SELECT * FROM wwi_mcw.CampaignAnalytics;
revert;
-- RLS is working indeed.

-- Step:5 The CEO should be able to see all rows in the table.
EXECUTE AS USER = 'CEO';
SELECT * FROM wwi_mcw.CampaignAnalytics;
revert;
-- And he can.

--Step:6 To disable the security policy we just created above, we execute the following.
ALTER SECURITY POLICY SalesFilter
WITH (STATE = OFF);

DROP SECURITY POLICY SalesFilter;
DROP FUNCTION Security.fn_securitypredicate;
DROP SCHEMA Security;
```



3. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Task 3: Dynamic data masking

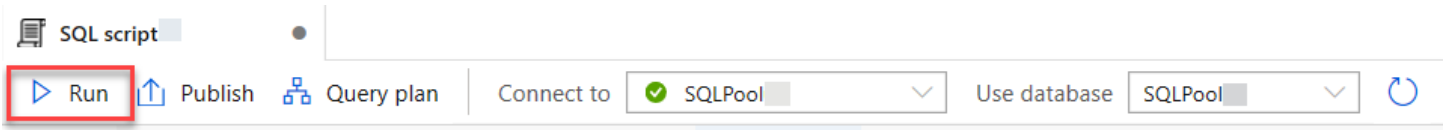
As an alternative to column level security, SQL Administrators also have the option of masking sensitive data. This will result in data being obfuscated when returned in queries. The data is still stored in a pristine state in the table itself. SQL Administrators can grant unmask privileges to users that have permissions to see this data.

1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

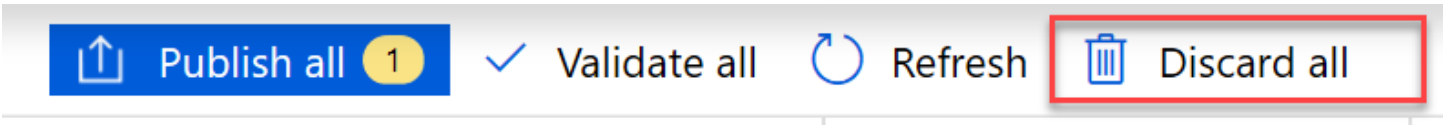
2. Copy and paste the following query into the query window. Then, step through each statement group by highlighting all queries between each comment block in the query window, and selecting **Run** from the query window toolbar menu. The query is documented inline.

```
----- Dynamic Data Masking (DDM) ---------

/*  Dynamic data masking helps prevent unauthorized access to sensitive data by enabling cu
stomers
    to designate how much of the sensitive data to reveal with minimal impact on the applic
ation layer.
    Let see how */

/* Scenario: WWI has identified sensitive information in the CustomerInfo table. They would
 like us to
    obfuscate the CreditCard and Email columns of the CustomerInfo table to DataAnalysts */

-- Step:1 Let's first get a view of CustomerInfo table.
SELECT TOP (100) * FROM wwi_mcw.CustomerInfo;

-- Step:2 Let's confirm that there are no Dynamic Data Masking (DDM) applied on columns.
SELECT c.name, tbl.name as table_name, c.is_masked, c.masking_function
FROM sys.masked_columns AS c
JOIN sys.tables AS tbl
    ON c.[object_id] = tbl.[object_id]
WHERE is_masked = 1
    AND tbl.name = 'CustomerInfo';
-- No results returned verify that no data masking has been done yet.

-- Step:3 Now let's mask 'CreditCard' and 'Email' Column of 'CustomerInfo' table.
ALTER TABLE wwi_mcw.CustomerInfo
ALTER COLUMN [CreditCard] ADD MASKED WITH (FUNCTION = 'partial(0,"XXXX-XXXX-XXXX-",4)');
GO

ALTER TABLE wwi_mcw.CustomerInfo
ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()');
GO
-- The columns are successfully masked.

-- Step:4 Let's see Dynamic Data Masking (DDM) applied on the two columns.
SELECT c.name, tbl.name as table_name, c.is_masked, c.masking_function
FROM sys.masked_columns AS c
JOIN sys.tables AS tbl
    ON c.[object_id] = tbl.[object_id]
WHERE is_masked = 1
    AND tbl.name ='CustomerInfo';

-- Step:5 Now, let's grant SELECT permission to 'DataAnalystMiami'
-- on the 'CustomerInfo' table.
GRANT SELECT ON wwi_mcw.CustomerInfo TO DataAnalystMiami;

-- Step:6 Logged in as  'DataAnalystMiami' let's execute the select query and
-- view the result.
EXECUTE AS USER = 'DataAnalystMiami';
SELECT * FROM wwi_mcw.CustomerInfo;

-- Step:7 Let's remove the data masking using UNMASK permission
```

```
GRANT UNMASK TO DataAnalystMiami;
EXECUTE AS USER = 'DataAnalystMiami';
SELECT *
FROM wwi_mcw.CustomerInfo;
revert;
REVOKE UNMASK TO DataAnalystMiami;

----step:8 Reverting all the changes back to as it was.
ALTER TABLE wwi_mcw.CustomerInfo
ALTER COLUMN CreditCard DROP MASKED;
GO
ALTER TABLE wwi_mcw.CustomerInfo
ALTER COLUMN Email DROP MASKED;
GO
```



3. From the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.



Exercise 7: Machine Learning

Duration: 60 minutes

Using Azure Synapse Analytics, data scientists are no longer required to use separate tooling to create and deploy machine learning models.

In this exercise, you will create multiple machine learning models. You will learn how to consume these models in your notebook. You will also deploy a model as a web service to Azure Container Instances and consume the service - without ever having to leave the Azure Synapse Analytics workspace.

Task 1: Analyze data with Apache Spark

1. Select **Develop** from the left menu, from the **Develop** menu, expand the **Notebooks** section and select 'Import'.

2. Browse to '' and import 'NYC Taxi - Analyze.ipynb' notebook.

Task 2: Training, consuming, and deploying models

1. Select **Develop** from the left menu, from the **Develop** menu, expand the **Notebooks** section and select 'Import'.
2. Browse to '' and import 'NYC Taxi - Spark MLlib.ipynb' notebook.

Note: Please note that each of these tasks will be addressed through several cells in the notebook.

Exercise 8: Monitoring

Duration: 45 minutes

Azure Synapse Analytics provides a rich monitoring experience within the Azure portal to surface insights regarding your data warehouse workload.

You can monitor active SQL requests using the SQL requests area of the Monitor Hub. This includes details like the pool, submitter, duration, queued duration, workload group assigned, importance, and the request content.

Pipeline runs can be monitored using the Monitor Hub and selecting Pipeline runs. Here you can filter pipeline runs and drill in to view the activity runs associated with the pipeline run and monitor the running of in-progress pipelines.

Task 1: Workload importance

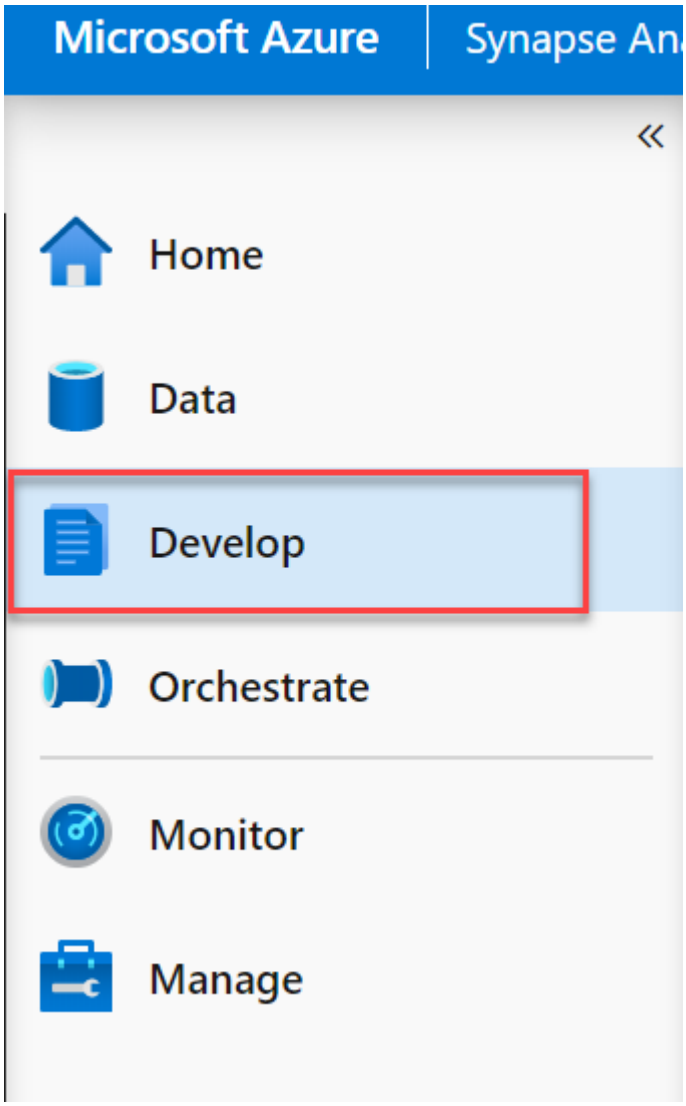
Running mixed workloads can pose resource challenges on busy systems. Solution architects seek ways to separate classic data warehousing activities (such as loading, transforming, and querying data) to ensure that enough resources exist to hit SLAs.

Synapse SQL pool workload management in Azure Synapse consists of three high-level concepts: workload classification, workload importance and workload isolation. These capabilities give you more control over how your workload utilizes system resources.

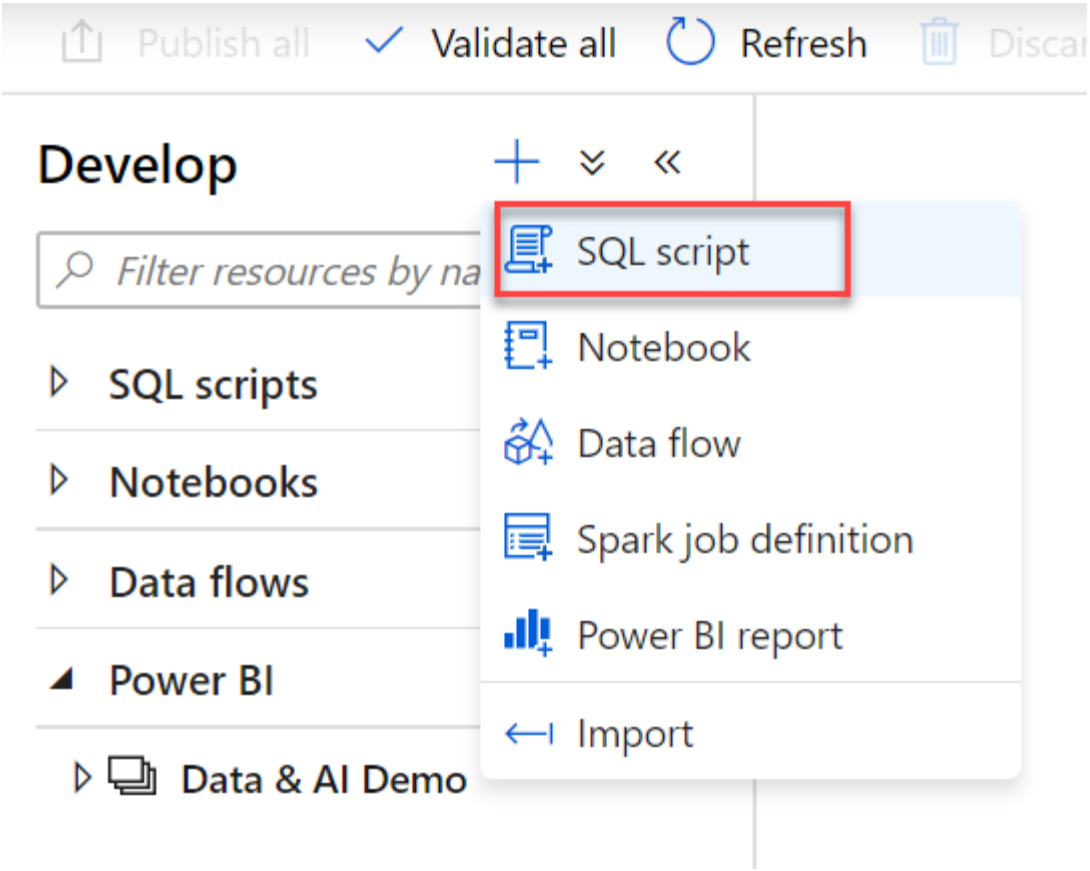
Workload importance influences the order in which a request gets access to resources. On a busy system, a request with higher importance has first access to resources. Importance can also ensure ordered access to locks.

Setting importance in Synapse SQL for Azure Synapse allows you to influence the scheduling of queries. Queries with higher importance will be scheduled to run before queries with lower importance. To assign importance to queries, you need to create a workload classifier.

1. Navigate to the **Develop** hub.



2. From the **Develop** menu, select the + button and choose **SQL Script** from the context menu.

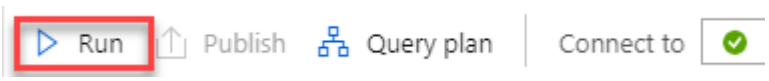


3. In the toolbar menu, connect to the **SQL Pool** database to execute the query.

4. In the query window, replace the script with the following to confirm that there are no queries currently being run by users logged in as `asa.sql.workload01`, representing the CEO of the organization or `asa.sql.workload02` representing the data analyst working on the project:

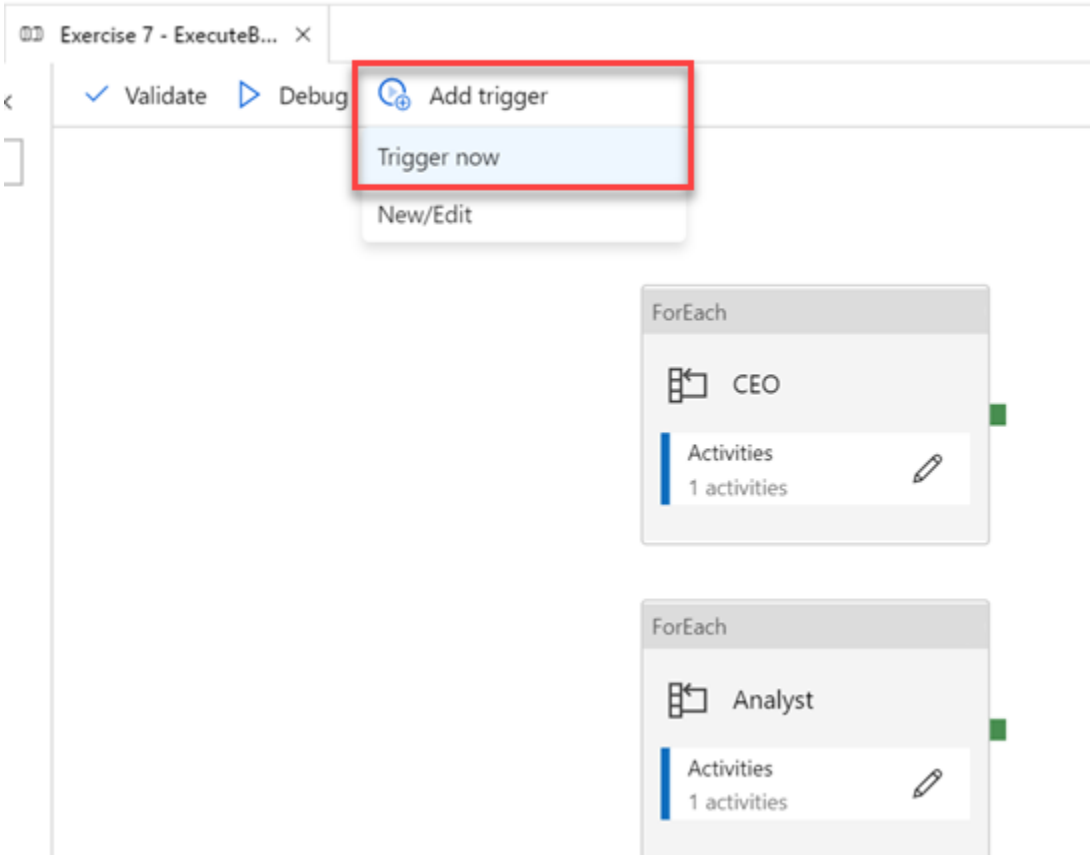
```
-- First, let's confirm that there are no queries currently
-- being run by users logged in as CEONYC or AnalystNYC.


SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
--and submit_time>dateadd(minute,-2,getdate())
ORDER BY submit_time ,s.login_name
```

5. Select **Run** from the toolbar menu to execute the SQL command.

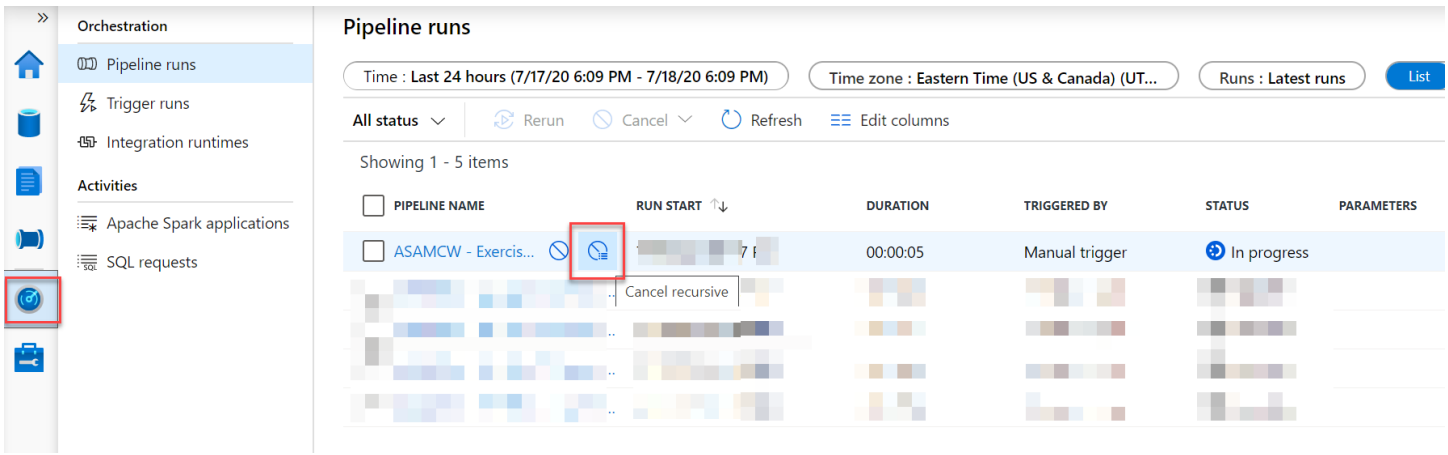


6. Next, you will flood the system with queries and see what happens for `asa.sql.workload01` and `asa.sql.workload02`. To do this, we'll run an Azure Synapse Pipeline that executes a large number of queries.
7. Select the **`Integrate`** Tab.
8. **Run** the **Exercise 8 - Execute Data Analyst and CEO Queries** Pipeline, which will run the `asa.sql.workload01` and `asa.sql.workload02` queries. You can run the pipeline with the Debug option if you have an instance of the Integration Runtime running.

9. Select **Add trigger**, then **Trigger now**. In the dialog that appears, select **OK**. **Let this pipeline run for 30 seconds to 1 minute, then proceed to the next step**.



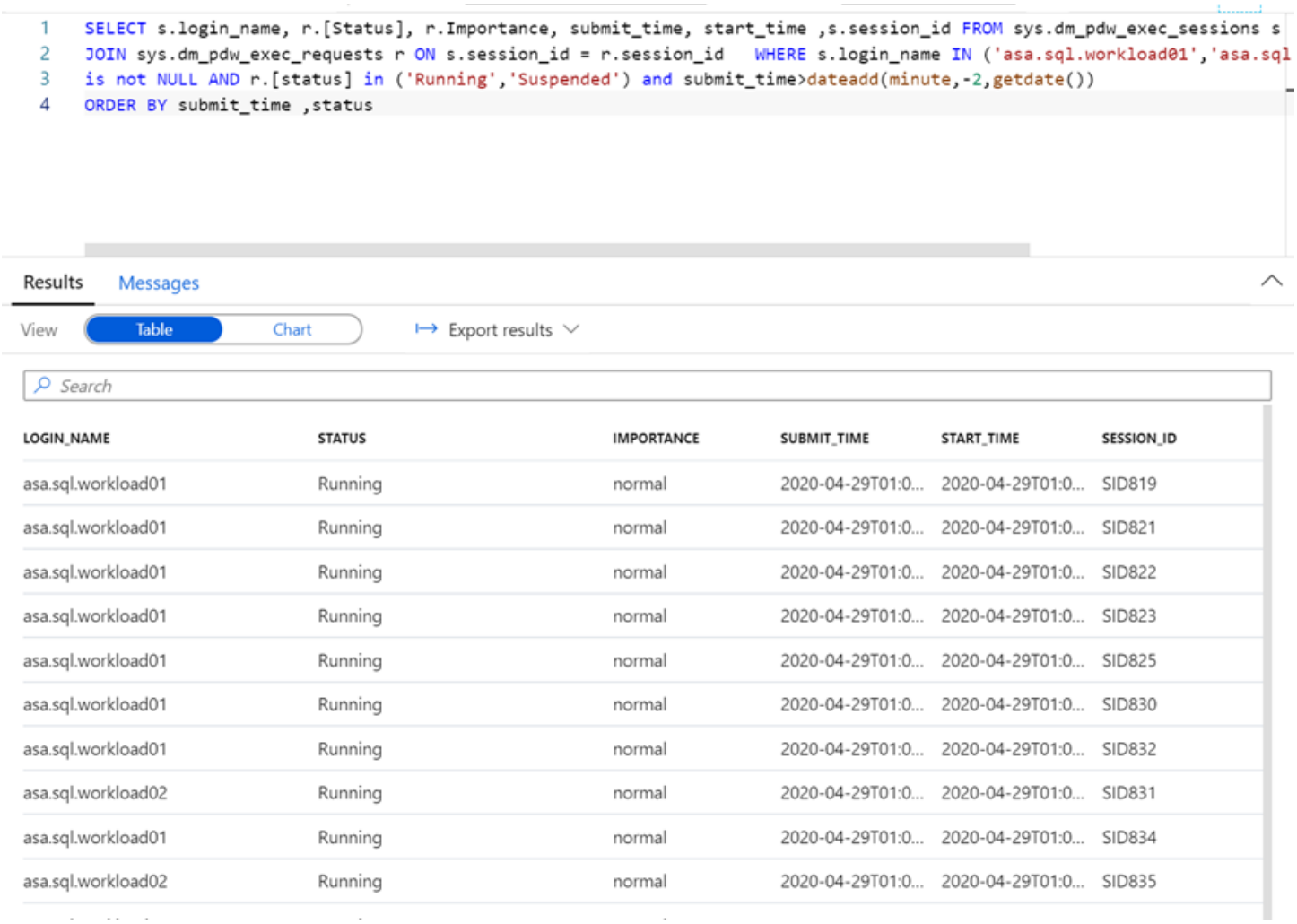
10. From the left menu, select the **Monitor** hub. Hover over the link of the in-progress pipeline, and select the **Cancel recursive** icon that displays.



11. From the left menu, select the **Develop** hub and return to your SQL script. Let's see what happened to all the queries that flooded the system. In the query window, replace the script with the following:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM s
ys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-
4,getdate())
ORDER BY submit_time ,status
```

12. Select **Run** from the toolbar menu to execute the SQL command. You should see an output similar to the following:



```
1  SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
2  JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id   WHERE s.login_name IN ('asa.sql.workload01','asa.sql
3  is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-2,getdate())
4  ORDER BY submit_time ,status
```

Results | Messages

View: Table | Chart — Export results

Search

LOGIN_NAME	STATUS	IMPORTANCE	SUBMIT_TIME	START_TIME	SESSION_ID
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID819
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID821
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID822
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID823
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID825
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID830
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID832
asa.sql.workload02	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID831
asa.sql.workload01	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID834
asa.sql.workload02	Running	normal	2020-04-29T01:0...	2020-04-29T01:0...	SID835

13. Intermittently perform the preceding query until all queries have been run and no results are returned.

14. We will give our `asa.sql.workload01` user queries priority by implementing the **workload importance** feature. In the query window, replace the script with the following:

```
IF EXISTS (SELECT * FROM sys.workload_management_workload_classifiers WHERE name = 'CEO')
BEGIN
    DROP WORKLOAD CLASSIFIER CEO;
END
CREATE WORKLOAD CLASSIFIER CEO
  WITH (WORKLOAD_GROUP = 'largerc'
  ,MEMBERNAME = 'asa.sql.workload01',IMPORTANCE = HIGH);
```

15. Select **Run** from the toolbar menu to execute the SQL command.

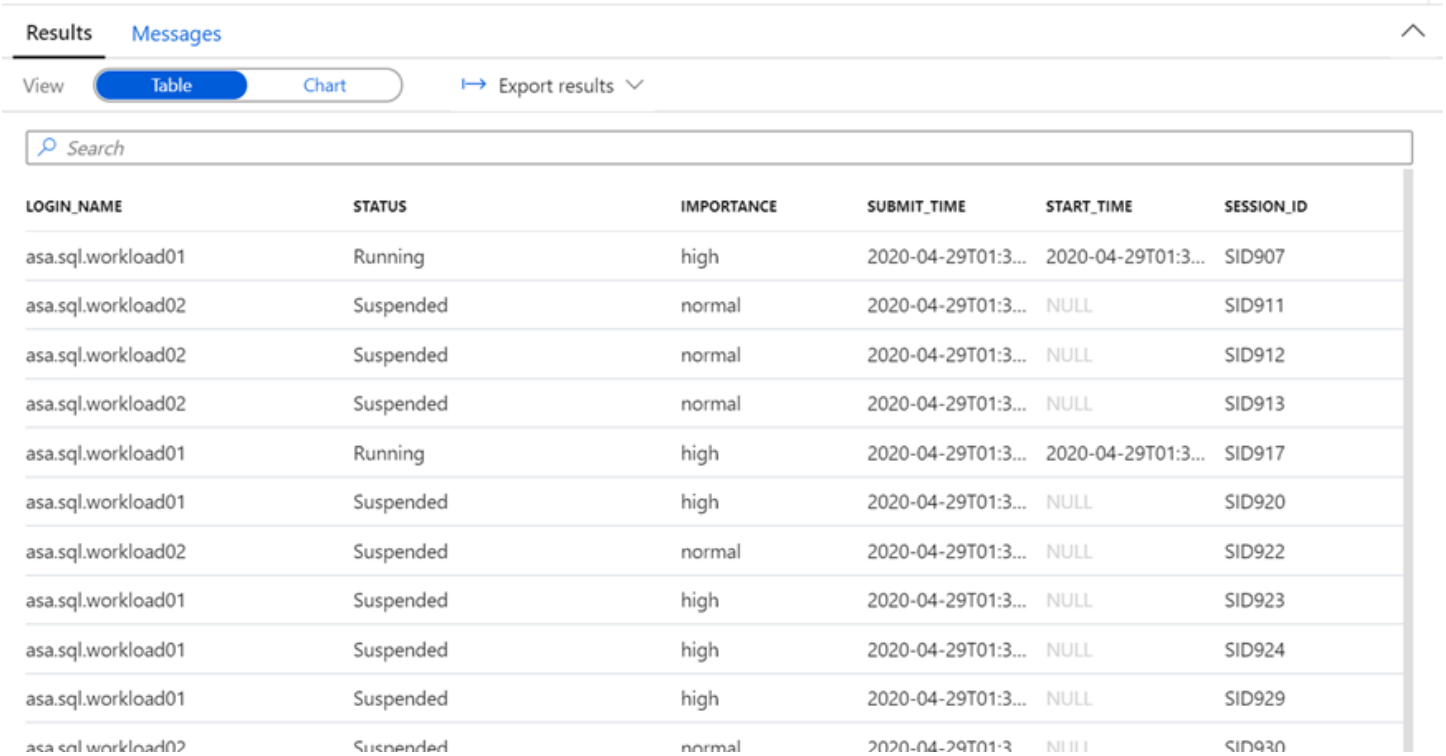
16. Let's flood the system again with queries and see what happens this time for `asa.sql.workload01` and `asa.sql.workload02` queries. To do this, we'll run an Azure Synapse Pipeline that runs a large number queries. **Similar to before, run this pipeline for about 30 seconds to 1 minute**.
    - **Select** the `Integrate` Tab.
    - **Run** the **Exercise 8 - Execute Data Analyst and CEO Queries** Pipeline, which will run the `asa.sql.workload01` and `asa.sql.workload02` queries.

17. In the query window, replace the script with the following to see what happens to the `asa.sql.workload01` queries this time:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM s
ys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-
2,getdate())
ORDER BY submit_time ,status desc
```

18. Select **Run** from the toolbar menu to execute the SQL command. You should see an output similar to the following that shows query executions for the `asa.sql.workload01` user having a **high** importance. Also note that the 'asa.sql.workload02' queries are in **Suspended** status while the high priority queries are being run.

```
SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id   WHERE s.login_name IN ('asa.sql.workload01','asa.sql.
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-2,getdate())
ORDER BY submit_time ,status desc
```



Results | Messages

View: Table | Chart — Export results

Search

LOGIN_NAME	STATUS	IMPORTANCE	SUBMIT_TIME	START_TIME	SESSION_ID
asa.sql.workload01	Running	high	2020-04-29T01:3...	2020-04-29T01:3...	SID907
asa.sql.workload02	Suspended	normal	2020-04-29T01:3...	NULL	SID911
asa.sql.workload02	Suspended	normal	2020-04-29T01:3...	NULL	SID912
asa.sql.workload02	Suspended	normal	2020-04-29T01:3...	NULL	SID913
asa.sql.workload01	Running	high	2020-04-29T01:3...	2020-04-29T01:3...	SID917
asa.sql.workload01	Suspended	high	2020-04-29T01:3...	NULL	SID920
asa.sql.workload02	Suspended	normal	2020-04-29T01:3...	NULL	SID922
asa.sql.workload01	Suspended	high	2020-04-29T01:3...	NULL	SID923
asa.sql.workload01	Suspended	high	2020-04-29T01:3...	NULL	SID924
asa.sql.workload01	Suspended	high	2020-04-29T01:3...	NULL	SID929
asa.sql.workload02	Suspended	normal	2020-04-29T01:3	NULL	SID930

Task 2: Workload isolation

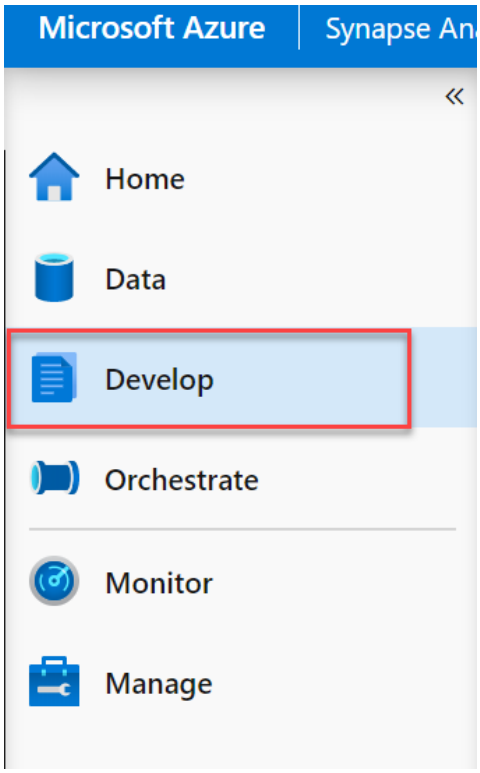
Workload isolation means resources are reserved, exclusively, for a workload group. Workload groups are containers for a set of requests and are the basis for how workload management, including workload isolation, is configured on a system. A simple workload management configuration can manage data loads and user queries.

In the absence of workload isolation, requests operate in the shared pool of resources. Access to resources in the shared pool is not guaranteed and is assigned on an importance basis.

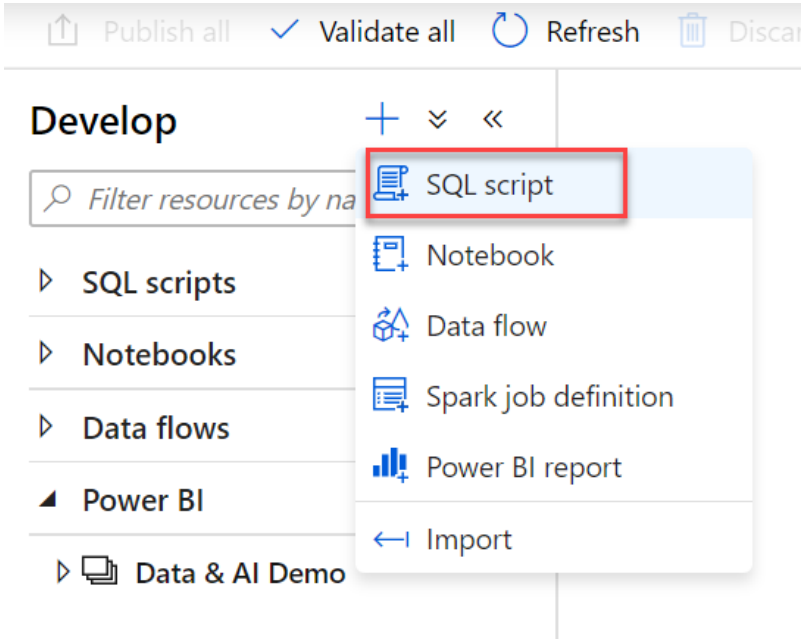
Configuring workload isolation should be done with caution as the resources are allocated to the workload group even if there are no active requests in the workload group. Over-configuring isolation can lead to diminished overall system utilization.

Users should avoid a workload management solution that configures 100% workload isolation: 100% isolation is achieved when the sum of `min_percentage_resource` configured across all workload groups equals 100%. This type of configuration is overly restrictive and rigid, leaving little room for resource requests that are accidentally misclassified. There is a provision to allow one request to execute from workload groups not configured for isolation.

1. Navigate to the **Develop** hub.



2. From the **Develop** menu, select the + button and choose **SQL Script** from the context menu.



3. In the toolbar menu, connect to the **SQL Pool** database to execute the query.

4. In the query window, replace the script with the following:

```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_groups where name = 'CEODemo'
)

BEGIN
    Create WORKLOAD GROUP CEODemo WITH
```

```
    ( MIN_PERCENTAGE_RESOURCE = 50        -- integer value
    ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 25 --
    ,CAP_PERCENTAGE_RESOURCE = 100
    )
END
```

The code creates a workload group called `CEODemo` that reserves resources exclusively for the workload group. In this example, a workload group with a `MIN_PERCENTAGE_RESOURCE` set to 50% and `REQUEST_MIN_RESOURCE_GRANT_PERCENT` set to 25% is guaranteed 2 concurrent queries.

5. Select **Run** from the toolbar menu to execute the SQL command.

6. In the query window, replace the script with the following to create a workload Classifier called `CEODreamDemo` that assigns a workload group and importance to incoming requests:

```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_classifiers where  name = 'CE
ODreamDemo')
BEGIN
    Create Workload Classifier CEODreamDemo with
    ( Workload_Group ='CEODemo',MemberName='asa.sql.workload02',IMPORTANCE = BELOW_NORMAL);
END
```

7. Select **Run** from the toolbar menu to execute the SQL command.

8. In the query window, replace the script with the following to confirm that there are no active queries being run by `asa.sql.workload02`:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```

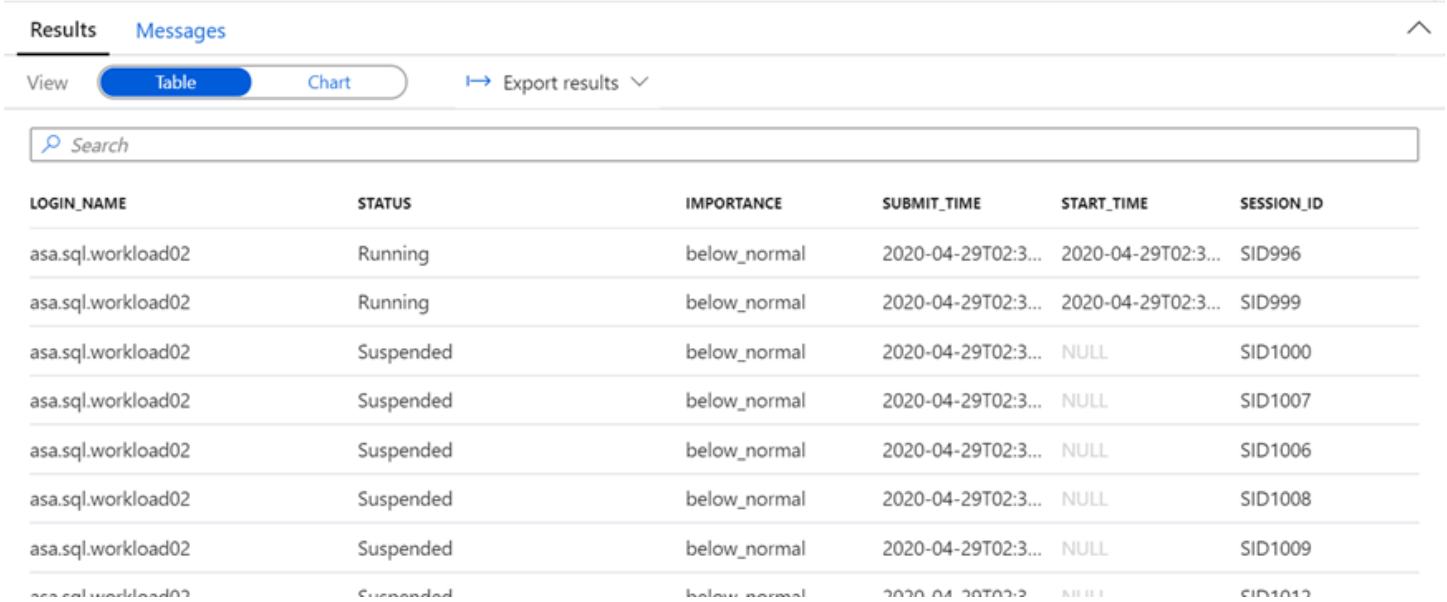
9. Let's flood the system with queries and see what happens for `asa.sql.workload02`. To do this, we will run an Azure Synapse Pipeline that runs a large number of queries. Select the `Orchestrate` Tab. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline, which will run the `asa.sql.workload02` queries. **Let this pipeline run for 30 seconds to 1 minute, then cancel the run recursively**.

10. In the query window, replace the script with the following to see what happened to all the `asa.sql.workload02` queries that were flooded into the system:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```

11. Select **Run** from the toolbar menu to execute the SQL command. You should see an output similar to the following that shows the importance for each session set to `below_normal` and two queries being run in parallel:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is  not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```



Results | Messages

View: Table | Chart — Export results

Search

LOGIN_NAME	STATUS	IMPORTANCE	SUBMIT_TIME	START_TIME	SESSION_ID
asa.sql.workload02	Running	below_normal	2020-04-29T02:3...	2020-04-29T02:3...	SID996
asa.sql.workload02	Running	below_normal	2020-04-29T02:3...	2020-04-29T02:3...	SID999
asa.sql.workload02	Suspended	below_normal	2020-04-29T02:3...	NULL	SID1000
asa.sql.workload02	Suspended	below_normal	2020-04-29T02:3...	NULL	SID1007
asa.sql.workload02	Suspended	below_normal	2020-04-29T02:3...	NULL	SID1006
asa.sql.workload02	Suspended	below_normal	2020-04-29T02:3...	NULL	SID1008
asa.sql.workload02	Suspended	below_normal	2020-04-29T02:3...	NULL	SID1009

12. In the query window, replace the script with the following to set 3.25% minimum resources per request:

```
IF  EXISTS (SELECT * FROM sys.workload_management_workload_classifiers where group_name = '
CEODemo')
BEGIN
    Drop Workload Classifier CEODreamDemo
    DROP WORKLOAD GROUP CEODemo
    --- Creates a workload group 'CEODemo'.
        Create  WORKLOAD GROUP CEODemo WITH
    (MIN_PERCENTAGE_RESOURCE = 26 -- integer value
        ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 3.25 -
- factor of 26 (guaranteed more than 4 concurrencies)
    ,CAP_PERCENTAGE_RESOURCE = 100
    )
    --- Creates a workload Classifier 'CEODreamDemo'.
    Create Workload Classifier CEODreamDemo with
    (Workload_Group ='CEODemo',MemberName='asa.sql.workload02',IMPORTANCE = BELOW_NORMAL);
END
```

Note: Configuring workload containment implicitly defines a maximum level of concurrency. With a CAP_PERCENTAGE_RESOURCE set to 60% and a REQUEST_MIN_RESOURCE_GRANT_PERCENT set to 1%, up to a 60-concurrency level is allowed for the workload group. Consider the method included below for determining the maximum concurrency:

[Max Concurrency] = [CAP_PERCENTAGE_RESOURCE] / [REQUEST_MIN_RESOURCE_GRANT_PERCENT]

13. Let's flood the system again and see what happens for `asa.sql.workload02`. To do this, we will run an Azure Synapse Pipeline that runs a large number of queries. Select the `Orchestrate` Tab. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline, which will run the `asa.sql.workload02` queries.
14. In the query window, replace the script with the following to see what happened to all of the `asa.sql.workload02` queries that flooded the system, note that many more queries are now being performed in parallel for asa.sql.workload02:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is  not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```

15. Select **Run** from the toolbar menu to execute the SQL command.

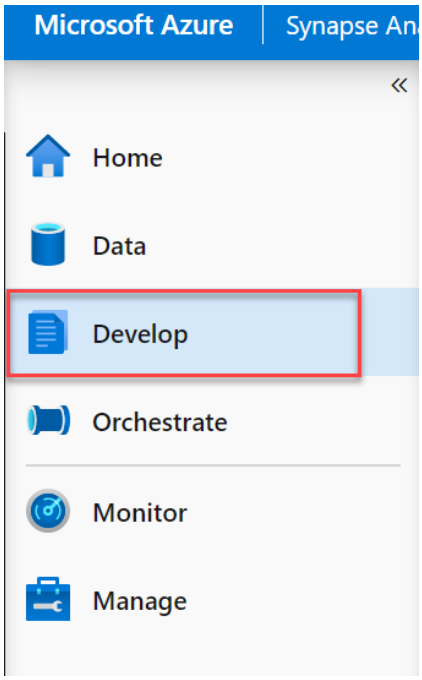
LOGIN_NAME	STATUS	IMPORTANCE	SUBMIT_TIME	START_TIME	SESSION_ID
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.4030000	2020-07-18T22:40:17.4330000	SID690
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.4330000	2020-07-18T22:40:17.4800000	SID692
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.4330000	2020-07-18T22:40:17.4970000	SID693
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.4330000	2020-07-18T22:40:17.4630000	SID691
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.4800000	2020-07-18T22:40:17.5430000	SID695
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.6200000	2020-07-18T22:40:17.7470000	SID699
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:17.6670000	2020-07-18T22:40:17.8870000	SID698
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:27.7770000	2020-07-18T22:40:27.9330000	SID701
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:36.0600000	2020-07-18T22:40:36.2300000	SID704
asa.sql.workload02	Running	below_normal	2020-07-18T22:40:36.1670000	2020-07-18T22:40:36.3570000	SID705

Task 3: Monitoring with Dynamic Management Views

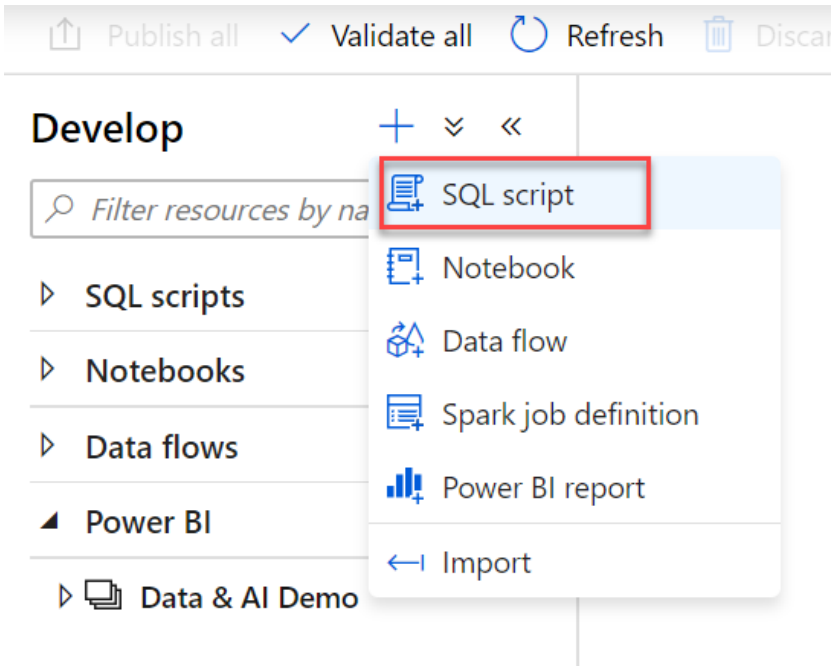
For a programmatic experience when monitoring SQL Analytics via T-SQL, the service provides a set of Dynamic Management Views (DMVs). These views are useful when actively troubleshooting and identifying performance bottlenecks with your workload.

All logins to your data warehouse are logged to `sys.dm_pdw_exec_sessions`. This DMV contains the last 10,000 logins. The `session_id` is the primary key and is assigned sequentially for each new logon.

1. Navigate to the **Develop** hub.



2. From the **Develop** menu, select the + button and choose **SQL Script** from the context menu.



3. In the toolbar menu, connect to the **SQL Pool** database to execute the query.

4. In the query window, replace the script with the following:

```
SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed' and session_id <> session_id();
```

All queries executed on SQL pool are logged to `sys.dm_pdw_exec_requests`. This DMV contains the last 10,000 queries executed. The `request_id` uniquely identifies each query and is the primary key for this DMV. The `request_id` is assigned sequentially for each new query and is prefixed with `QID`,

which stands for query ID. Querying this DMV for a given `session_id` shows all queries for a given logon.

5. Select **Run** from the toolbar menu to execute the SQL command.

6. Let's flood the system with queries to create operations to monitor. To do this, we will run a Azure Synapse Pipeline which triggers queries. Select the `Orchestrate` Tab. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline, which will run / trigger `asa.sql.workload02` queries. **Let this pipeline run for 30 seconds to 1 minute, then cancel the run recursively**.

7. In the query window, replace the script with the following:

```
SELECT *
FROM sys.dm_pdw_exec_requests
WHERE status not in ('Completed','Failed','Cancelled')
  AND session_id <> session_id()
ORDER BY submit_time DESC;
```

8. Select **Run** from the toolbar menu to execute the SQL command. You should see a list of sessions in the query results similar to the following. **Note the `Request_ID` of a query** in the results that you would like to investigate (*keep this value in a text editor for a later step*):

```
1  SELECT *
2  FROM sys.dm_pdw_exec_requests
3  WHERE status not in ('Completed','Failed','Cancelled')
4    AND session_id <> session_id()
5  ORDER BY submit_time DESC;
```

Results | Messages

View: Table | Chart — Export results

Search

REQUEST_ID	SESSION_ID	STATUS	SUBMIT_TIME	START_TIME	END_COMPILE_TIME	END_TIME	TOTAL_ELAPSED_TII
QID6386	SID1070	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5375
QID6388	SID1071	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5375
QID6390	SID1072	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5375
QID6379	SID1065	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5484
QID6381	SID1066	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5484
QID6373	SID1062	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5500
QID6376	SID1064	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5500
QID6377	SID1063	Running	2020-04-29T02:5...	2020-04-29T02:5...	2020-04-29T02:5...	NULL	5500

9. As an alternative, you can execute the following SQL command to find the top 10 longest running queries.

```
SELECT TOP 10 *
FROM sys.dm_pdw_exec_requests
ORDER BY total_elapsed_time DESC;
```

10. To simplify the lookup of a query in the `sys.dm_pdw_exec_requests` table, use `LABEL` to assign a comment to your query, which can be looked up in the `sys.dm_pdw_exec_requests` view. To test using the labels, replace the script in the query window with the following:

```
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query');
```

11. Select **Run** from the toolbar menu to execute the SQL command.

12. In the query window, replace the script with the following to filter the results with the label, `My Query`.

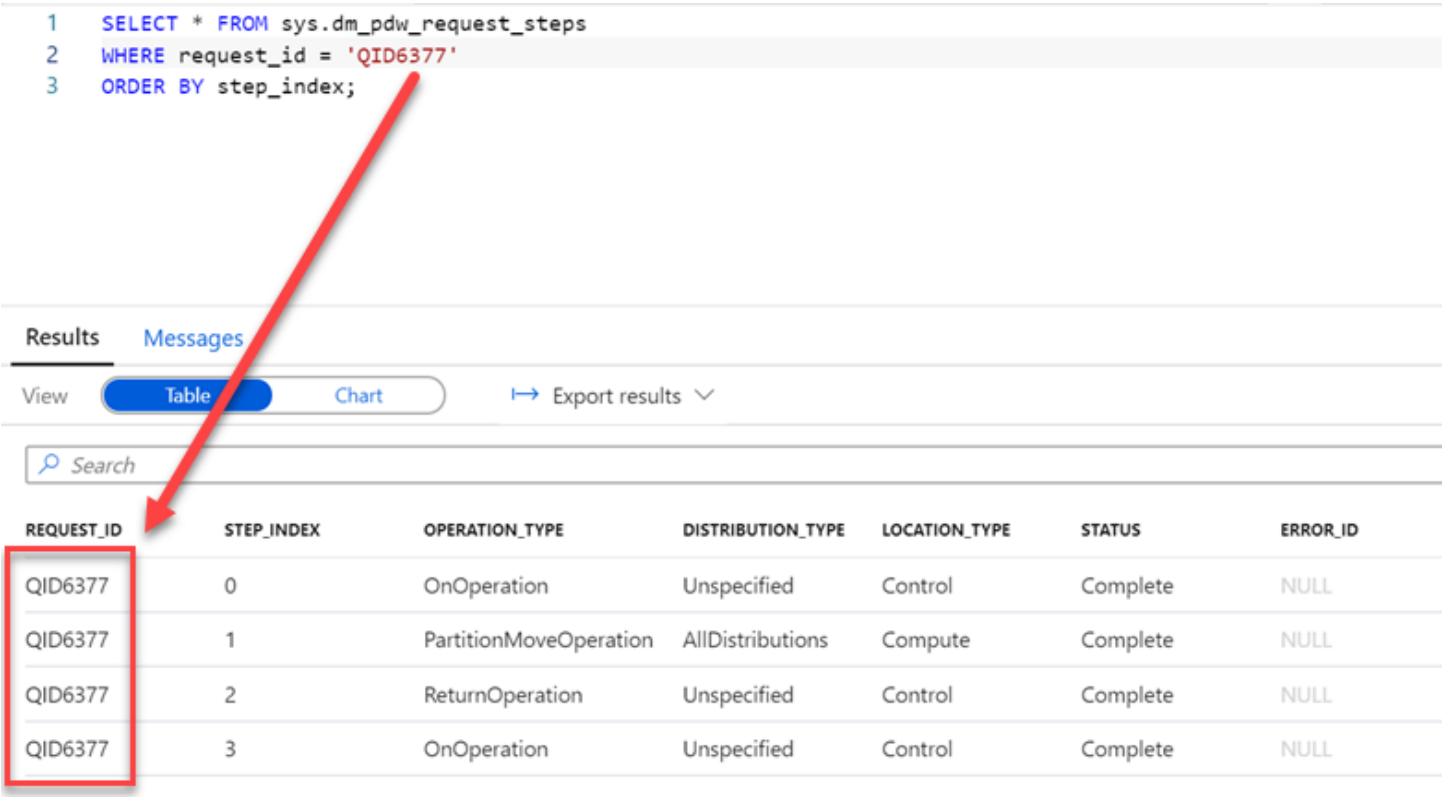
```
-- Find a query with the Label 'My Query'
-- Use brackets when querying the label column, as it is a key word
SELECT  *
FROM sys.dm_pdw_exec_requests
WHERE [label] = 'My Query';
```

13. Select **Run** from the toolbar menu to execute the SQL command. You should see the previously run query in the results view.

14. In the query window, replace the script with the following to retrieve the query's distributed SQL (DSQL) plan from `sys.dm_pdw_request_steps`. **Be sure to replace** the `QID#####` with the `Request_ID` you noted in Step 8:

```
SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = 'QID####'
ORDER BY step_index;
```

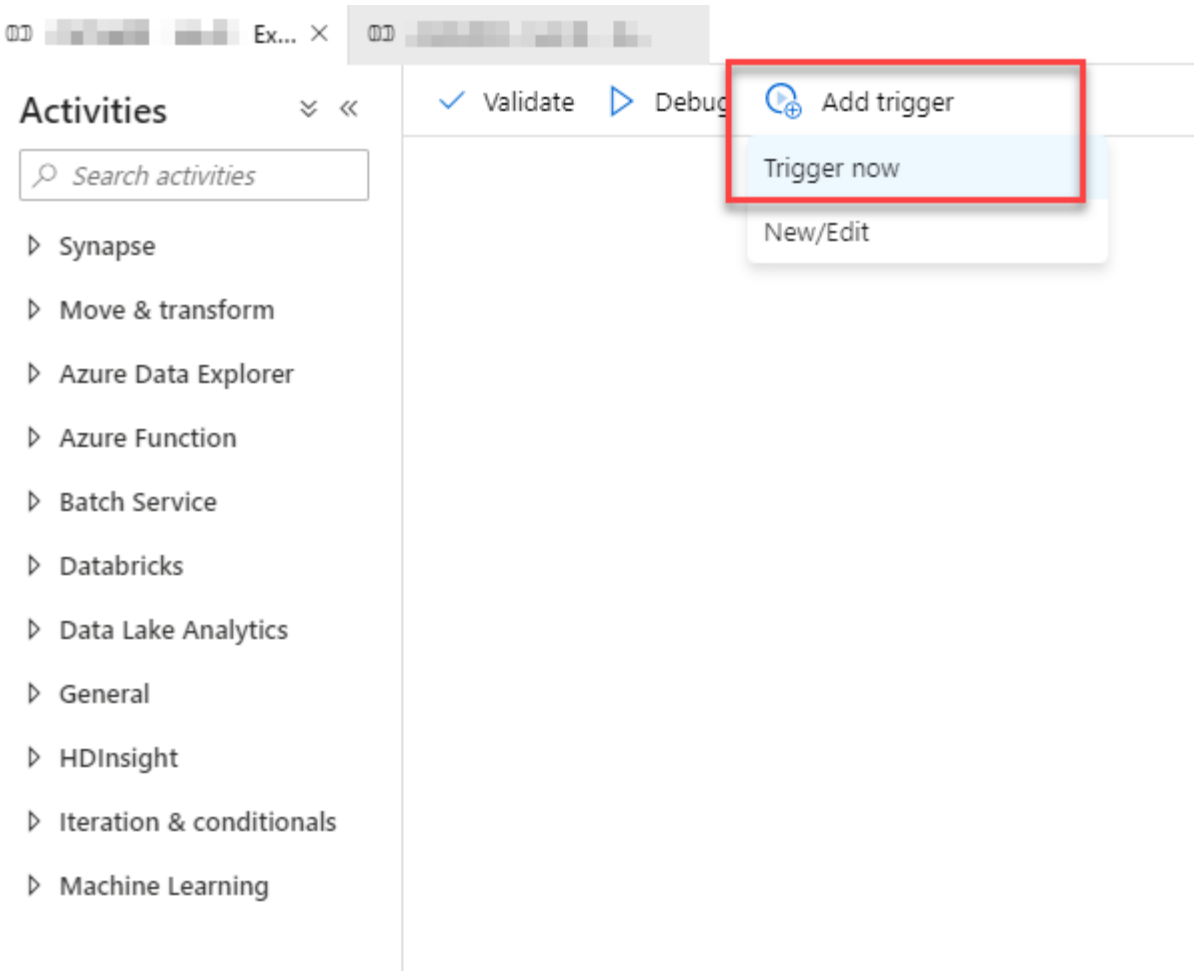
15. Select **Run** from the toolbar menu to execute the SQL command. You should see results showing the distributed query plan steps for the specified request:



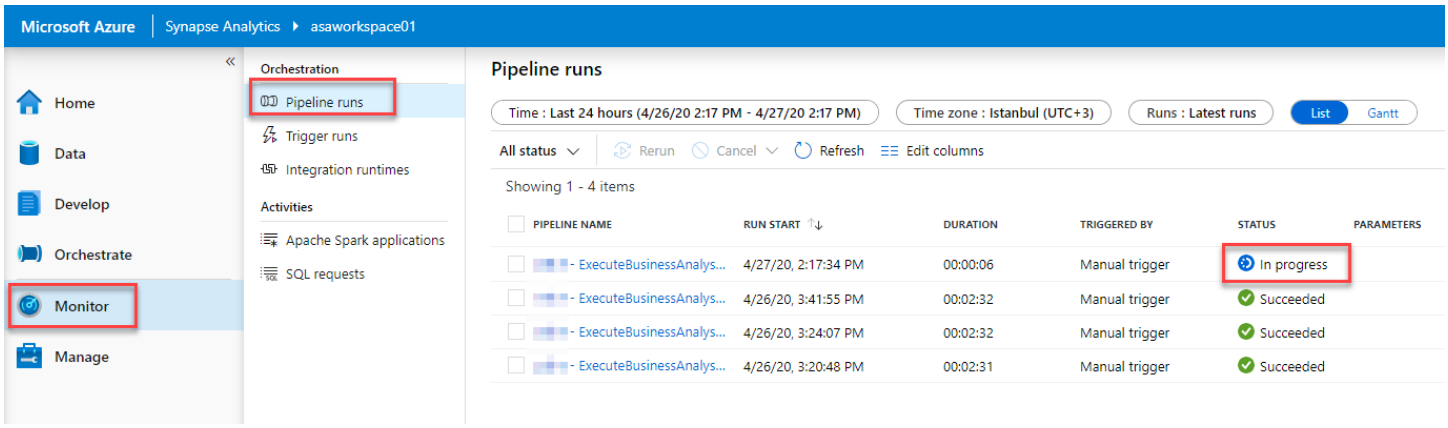
When a DSQL plan is taking longer than expected, the cause can be a complex plan with many DSQL steps or just one step taking a long time. If the plan is many steps with several move operations, consider optimizing your table distributions to reduce data movement.

Task 4: Orchestration Monitoring with the Monitor Hub

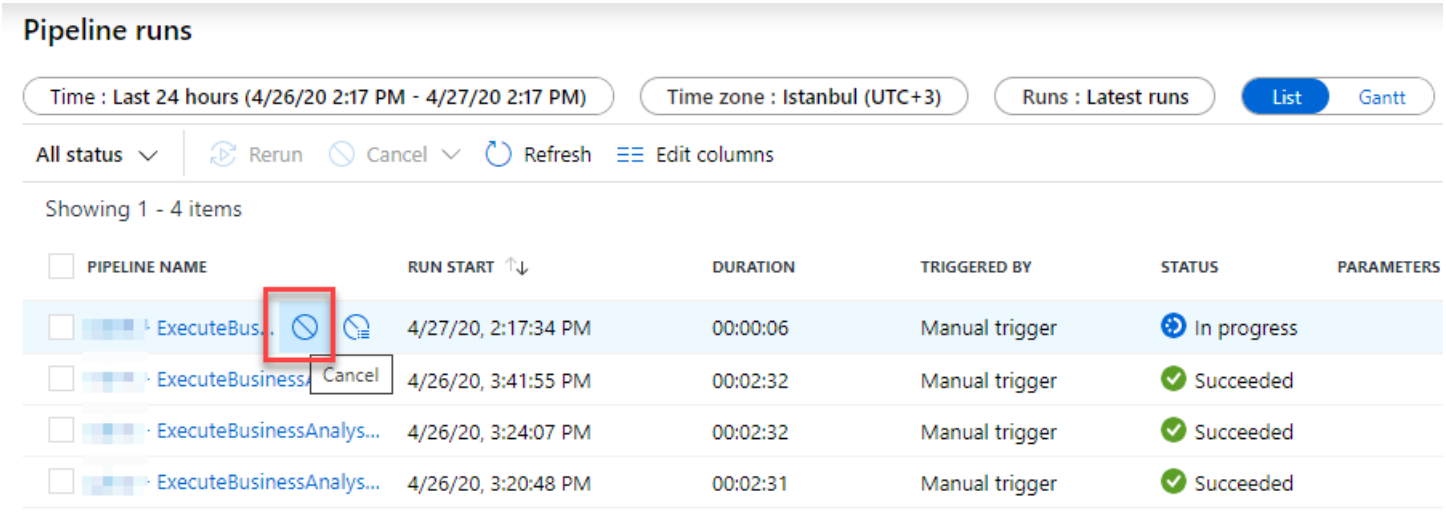
1. Let's run a pipeline to monitor its execution in the next step. To do this, select the `Orchestrate` Tab. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline.



2. Navigate to the `Monitor` hub. Then select **Pipeline runs** to get a list of pipelines that ran during the last 24 hours. Observe the Pipeline status.

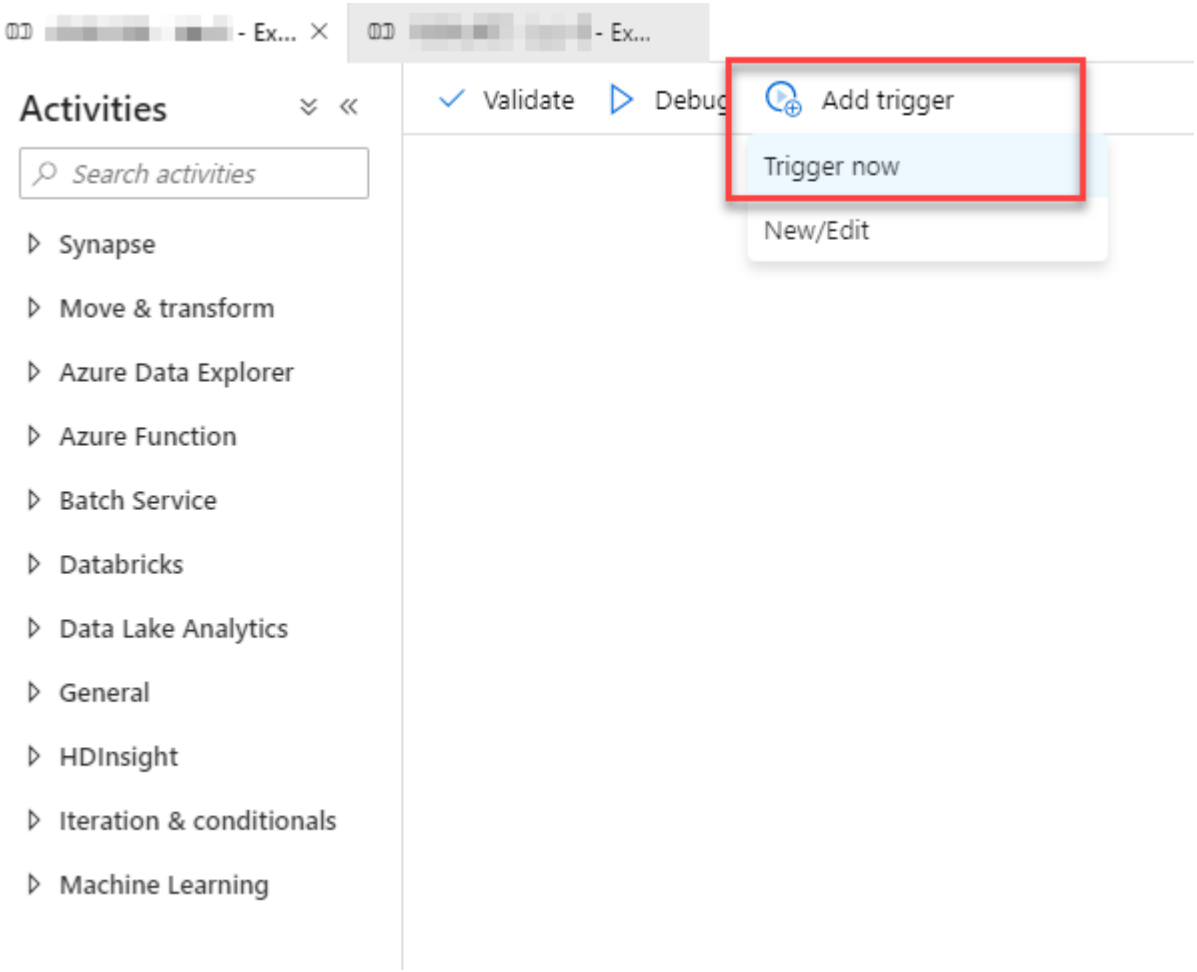


3. Hover over the running pipeline and select **Cancel** to cancel the execution of the current instance of the pipeline.



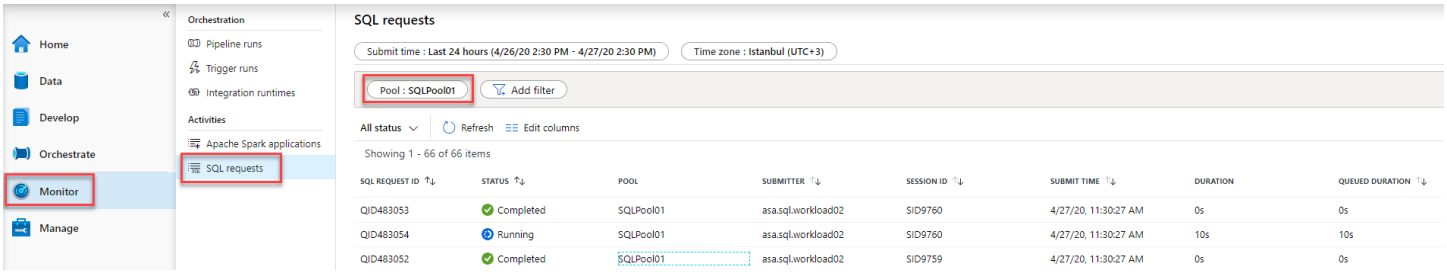
Task 5: Monitoring SQL Requests with the Monitor Hub

1. Let's run a pipeline to monitor its execution in the next step. To do this, select the `Orchestrate` Tab. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline.

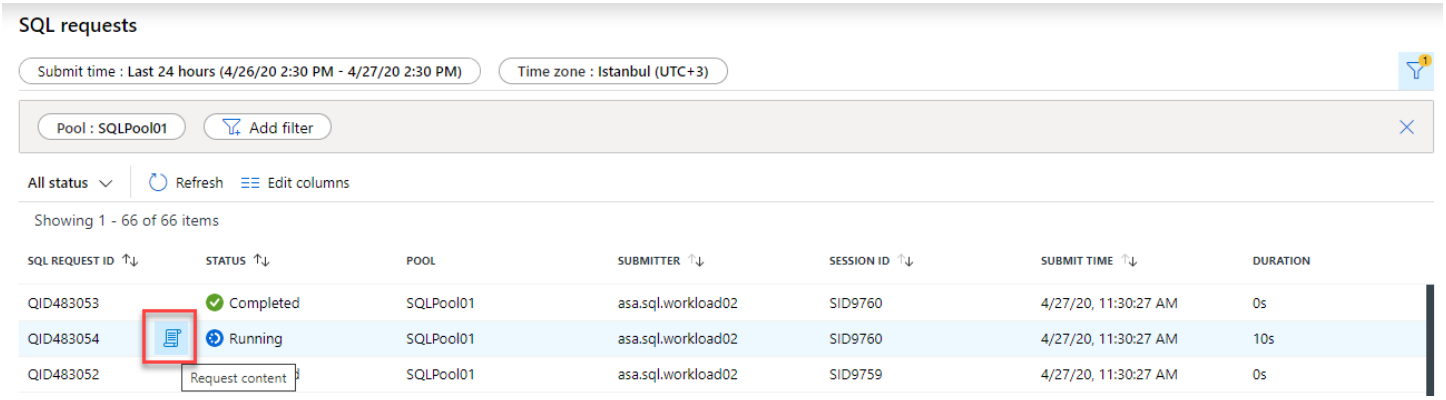


2. Navigate to the `Monitor` hub. Then select **SQL requests** to get a list of SQL requests that ran during the last 24 hours.

3. Select the **Pool** filter and select your SQL Pool. Observe the `Request Submitter`, `Submit Time`, `Duration`, and `Queued Duration` values.



4. Hover onto a SQL Request log and select `Request Content` to access the actual T-SQL command executed as part of the SQL Request.



5. You may now return to the **Monitor** hub and cancel the in-progress pipeline run.

After the hands-on lab

Duration: 5 minutes

Task 1: Delete the resource group

1. In the Azure Portal, open the resource group for this lab. Select **Delete** from the top toolbar menu.

2. In the Azure Portal, open the resource group with the same name as your Function App. Select **Delete** from the top toolbar menu.

3. Open the Cloud Shell and issue the following command to remove the lab files:

   ```
   Remove-Item -Path .\Synapse-MCW -recurse -force
   ```

You should follow all steps provided *after* attending the Hands-on lab.

https://github.com/microsoft/AzureTrailblazerAcademy

DefaultEndpointsProtocol=https;AccountName=asastorebp0109;AccountKey=dN4kmu0rkKYUXmeMifTICCPRr8nzqsmTvbRNltSNRkcrt6mn+aicrq/+4b4C1qpxzq8xo1FIrLJV5+lRQSfkhw==;EndpointSuffix=core.windows.net

https://asastorebp0109.blob.core.windows.net/?sv=2019-12-12&ss=bfqt&srt=sco&sp=rwdlacupx&se=2022-01-20T23:06:39Z&st=2021-01-20T15:06:39Z&spr=https&sig=HG5dwNfWHDxDuRwXVHSNRStzSqFH05ittWG%2BWt1xdJA%3D

https://asastorebp0109.blob.core.windows.net/invoices?sv=2019-12-12&ss=bfqt&srt=sco&sp=rwdlacupx&se=2022-01-20T23:06:39Z&st=2021-01-20T15:06:39Z&spr=https&sig=HG5dwNfWHDxDuRwXVHSNRStzSqFH05ittWG%2BWt1xdJA%3D

Form Recognizer

mcwformrecognizerbp0109

https://mcwformrecognizerbp0109.cognitiveservices.azure.com/

f17f214b1e304c658f0a79c9a956d87d

Cognitive Search

mcwsynapsecognitivesearchbp0109

1D0FEEE953C80EF467F66A4CEB34A996

"modelId": "f23480f9-9bda-46ab-bcaf-7f8cb7f7866e"

https://mcwformrecognizer2.azurewebsites.net/api/GetInvoiceData?code=sNWEFc7uJgRgzXAGovgxMH2VjuagNP2euovQ8IJBtHEltE4hSuExvw==

Data Science Virtual Machine - Windows 2019

Microsoft

Data Science Virtual Machine - Windows 2019

Add to Favorites

Microsoft

★★★★★ 5.0 (1 ratings)

Create | Start with a pre-set configuration

Overview | Plans | Usage Information + Support | Reviews

The **'Data Science Virtual Machine (DSVM)'** is a 'Windows Server 2019 with Containers' VM & includes popular tools for data exploration, analysis, modeling development.

Highlights:

- Anaconda Python
- SQL Server 2019 Dev. Edition - With In-Database R and Python analytics
- Microsoft Office 365 ProPlus BYOL - Shared Computer Activation
- Julia
- Jupyter notebooks
- Visual Studio Community Ed. + Python, R & node.js tools
- Power BI Desktop
- Deep learning tools e.g. TensorFlow, Chainer
- ML algorithm libraries e.g. xgboost, Vowpal Wabbit
- Azure SDKs + libraries for various Azure Cloud offerings. Integration tools are included for:

1. Azure Machine Learning
2. Azure Data Factory
3. Stream Analytics
4. SQL Data Warehouse
5. Hadoop + Apache Spark (HDICluster)
6. Data Lake
7. Blob storage
8. ML & Data Science tutorials as Jupyter notebooks

This image is pre-configured with Nvidia drivers, CUDA Toolkit, & cuDNN library for GPU workloads available if using NC class VM SKUs.

More offers from Microsoft

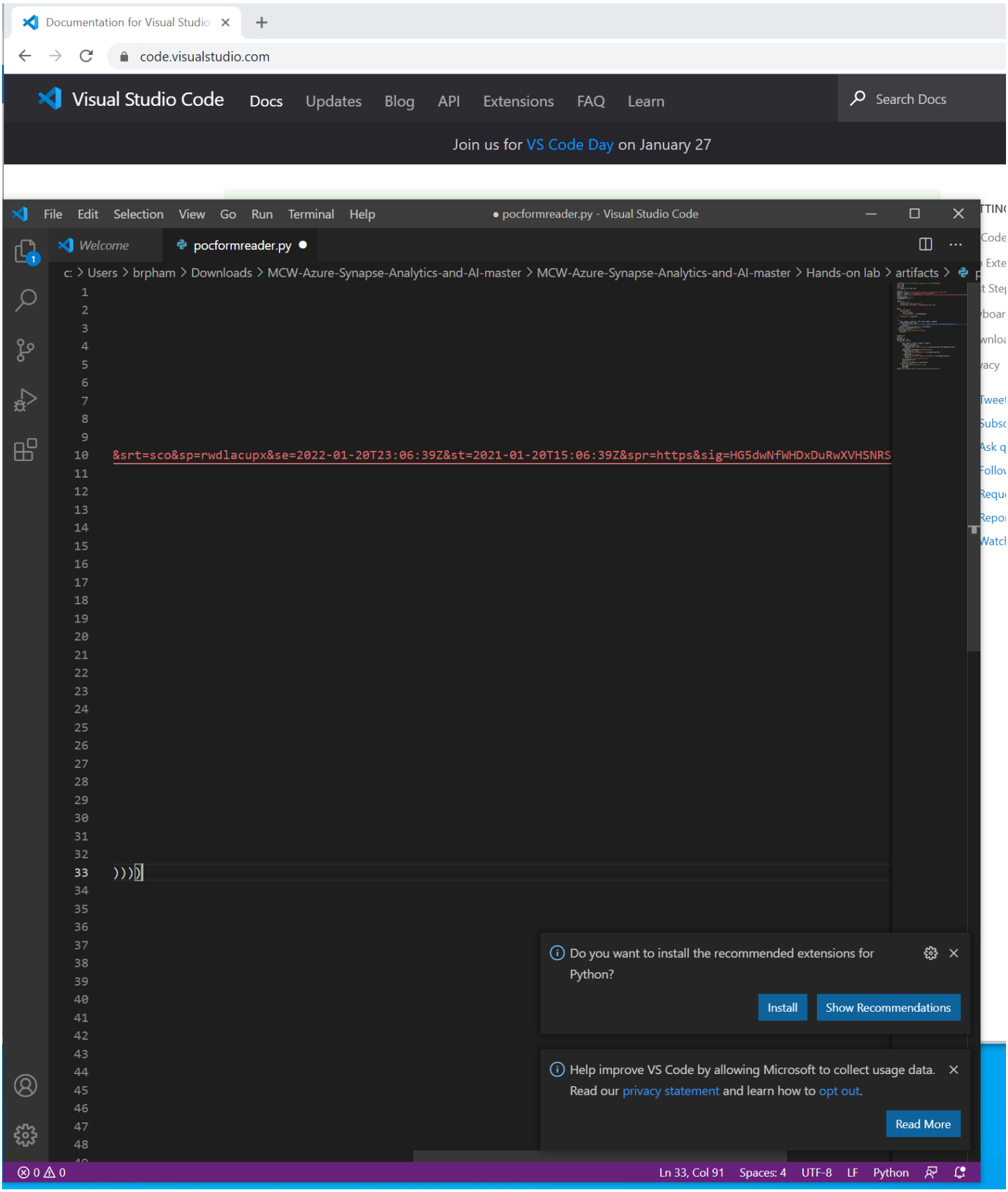
Data Science Virtual Machine- Ubuntu 18.04

Microsoft

★☆☆☆☆ 1.0 (1 ratings)

Virtual Machine

Data Science Virtual Machine - Ubuntu 18.04


Documentation for Visual Studio
code.visualstudio.com
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
Learn
Search Docs
Join us for VS Code Day on January 27
File
Edit
Selection
View
Go
Run
Terminal
Help
• pocformreader.py - Visual Studio Code
Welcome
pocformreader.py
c: > Users > brpham > Downloads > MCW-Azure-Synapse-Analytics-and-AI-master > MCW-Azure-Synapse-Analytics-and-AI-master > Hands-on lab > artifacts >
&srt=sco&sp=rwdlacupx&se=2022-01-20T23:06:39Z&st=2021-01-20T15:06:39Z&spr=https&sig=HG5dwNfWHDxDuRwXVHSNRS
))))
Do you want to install the recommended extensions for Python?
Install
Show Recommendations
Help improve VS Code by allowing Microsoft to collect usage data.
Read our privacy statement and learn how to opt out.
Read More
Ln 33, Col 91
Spaces: 4
UTF-8
LF
Python

Create a virtual machine

Changing Basic options may reset selections you have made. Review all options prior to creating the virtual machine.

Basics | Disks | Networking | Management | Advanced | Tags | Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Microsoft Azure Internal Consumption BP

Resource group * : Synapse-MCW-0109

Create new

Instance details

Virtual machine name * : vmvisualstudiobp0109

Region * : (US) East US

Availability options : Availability zone

Availability zone * : 1

Image * : Visual Studio 2019 Community (latest release) on Windows 10 Enterprise N (x64

See all images

Azure Spot instance : ☐

Size * : Standard_B4ms - 4 vcpus, 16 GiB memory ($132.86/month)

See all sizes

Administrator account

Username * : serveradmin

Password * : ••••••••••••

Confirm password * : ••••••••••••

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * : ○ None ◉ Allow selected ports

Select inbound ports * : RDP (3389)

This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Licensing

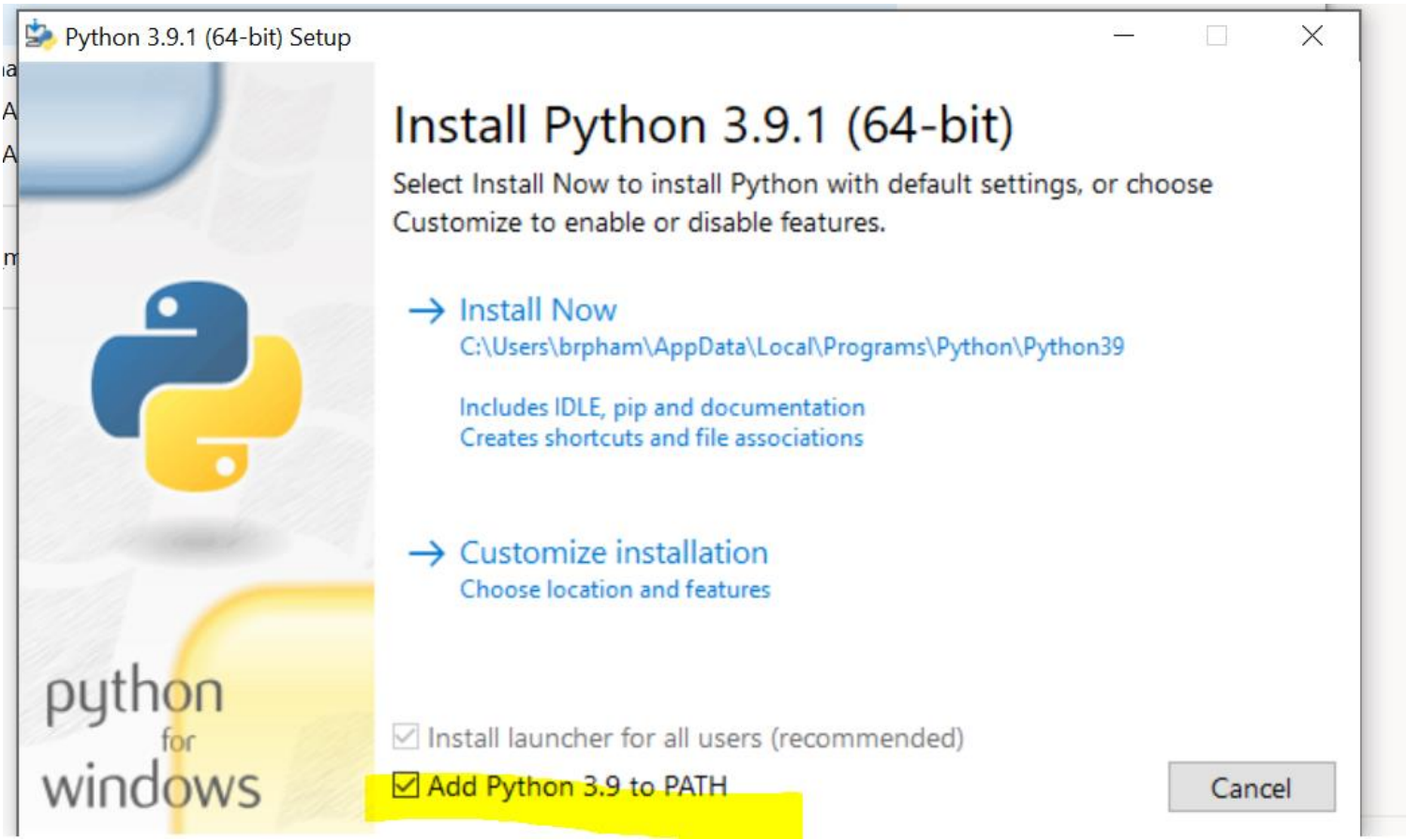
License type * : Windows server

Save up to 49% with a license you already own using Azure Hybrid Benefit. Learn more

Would you like to use an existing Windows Server license? * : ☐

Review Azure hybrid benefit compliance

Review + create | < Previous | Next : Disks >



[Download Python | Python.org](https://www.python.org/downloads/)

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

py get-pip.py

pip install requests

POST model succeeded:

{'Content-Length': '0', 'Location': 'https://mcwformrecognizerbp0201.cognitiveservices.azure.com/formrecognizer/v2.0/custom/models/f3dd73bb-6310-45b8-8bef-581b4eed7e99', 'x-envoy-upstream-service-time': '212', 'apim-request-id': 'f6952d55-c390-46d9-9e2d-aec22131f3a3', 'Strict-Transport-Security': 'max-age=31536000; includeSubDomains; preload', 'x-content-type-options': 'nosniff', 'Date': 'Mon, 01 Feb 2021 21:57:02 GMT'}

Training succeeded:

{"modelInfo": {"modelId": "f3dd73bb-6310-45b8-8bef-581b4eed7e99", "status": "ready", "createdDateTime": "2021-02-01T21:57:02Z", "lastUpdatedDateTime": "2021-02-01T21:57:18Z"}, "trainResult": {"trainingDocuments": [{"documentName": "Train/Invoice_1.pdf", "pages": 1, "errors": [], "status": "succeeded"}, {"documentName": "Train/Invoice_2.pdf", "pages": 1, "errors": [], "status": "succeeded"}, {"documentName": "Train/Invoice_3.pdf", "pages": 1, "errors": [], "status": "succeeded"}, {"documentName": "Train/Invoice_4.pdf", "pages": 1, "errors": [], "status": "succeeded"}, {"documentName": "Train/Invoice_5.pdf", "pages": 1, "errors": [], "status": "succeeded"}], "errors": []}}

PS C:\Users\brpham\Downloads\MCW-Azure-Synapse-Analytics-and-AI-master\MCW-Azure-Synapse-Analytics-and-AI-master\Hands-on lab\artifacts>

:; cd 'c:\Users\brpham\Downloads\MCW-Azure-Synapse-Analytics-and-AI-master\MCW-Azure-Synapse-Analytics-and-AI-master\Hands-on lab\artifacts'; & 'C:\Users\brpham\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\brpham\.vscode\extensions\ms-python.python-2021.1.502429796\pythonFiles\lib\python\debugpy\launcher' '61321' '--' 'c:\Users\brpham\Downloads\MCW-Azure-Synapse-Analytics-and-AI-master\MCW-Azure-Synapse-Analytics-and-AI-master\Hands-on lab\artifacts\pocformreader.py'