Microsoft Cloud Workshop

Microsoft Cloud Workshop

Azure Synapse Analytics and AI

Hands-on lab step-by-step

October 2020

technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Contents

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

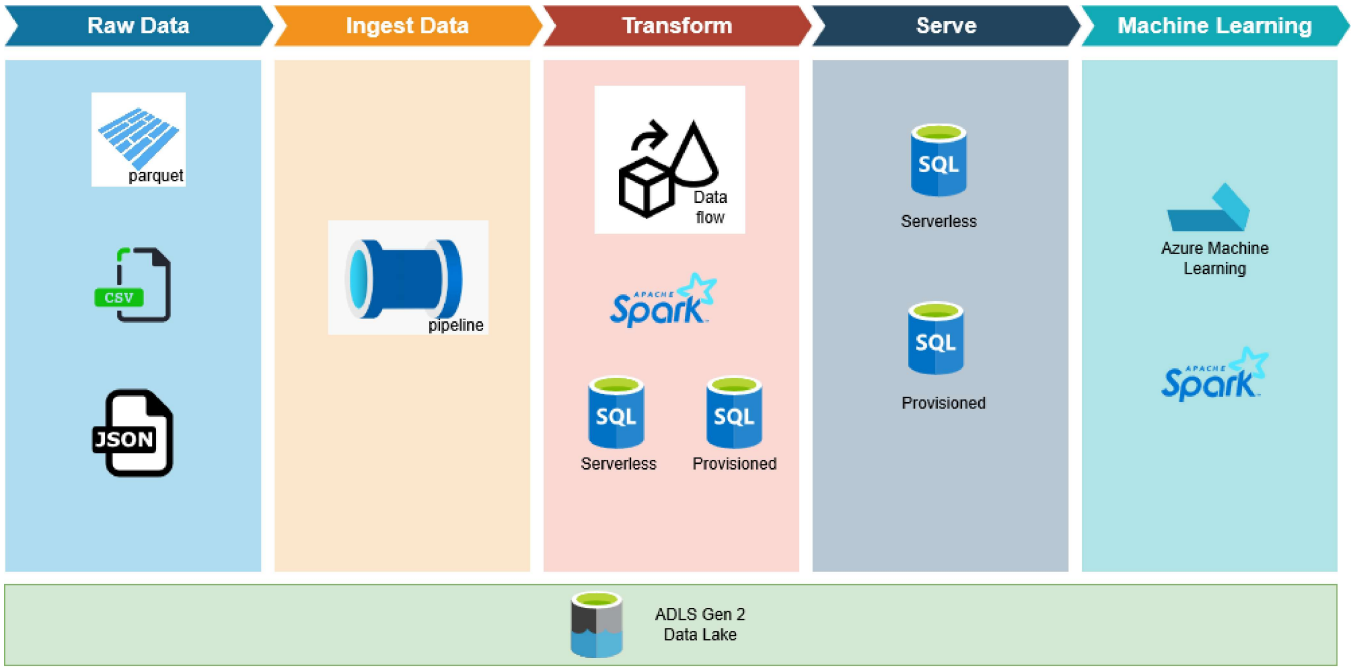
Abstract and learning objectives

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



Architecture diagram explained in the next paragraph.

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
3. Python v.3.7 or newer (https://www.python.org/downloads/)
4. PIP (https://pip.pypa.io/en/stable/installing/#do-i-need-to-install-pip)

5. Visual Studio Code (https://code.visualstudio.com/)
6. Python Extension for Visual Studio Code (https://marketplace.visualstudio.com/items?itemName=ms-python.python)
7. Azure Function Core Tools v.3 (https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=windows%2Ccsharp%2Cbash#v2)
8. Azure Functions Extension for Visual Studio Code (https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions)
9. Postman (https://www.postman.com/downloads/)
10. Ensure the Microsoft.Sql resource provider is registered in your Azure Subscription (https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/resource-providers-and-types).

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}`

Azure Synapse Analytics Resource	To be referred to
Primary Storage Account	`asadatalake{suffix}`
Default file system container	`DefaultFileSystem`
SQL Pool	`SqlPool01`
SQL Serverless Endpoint	`SqlServerless01`
Azure Key Vault	`asakeyvault{suffix}`

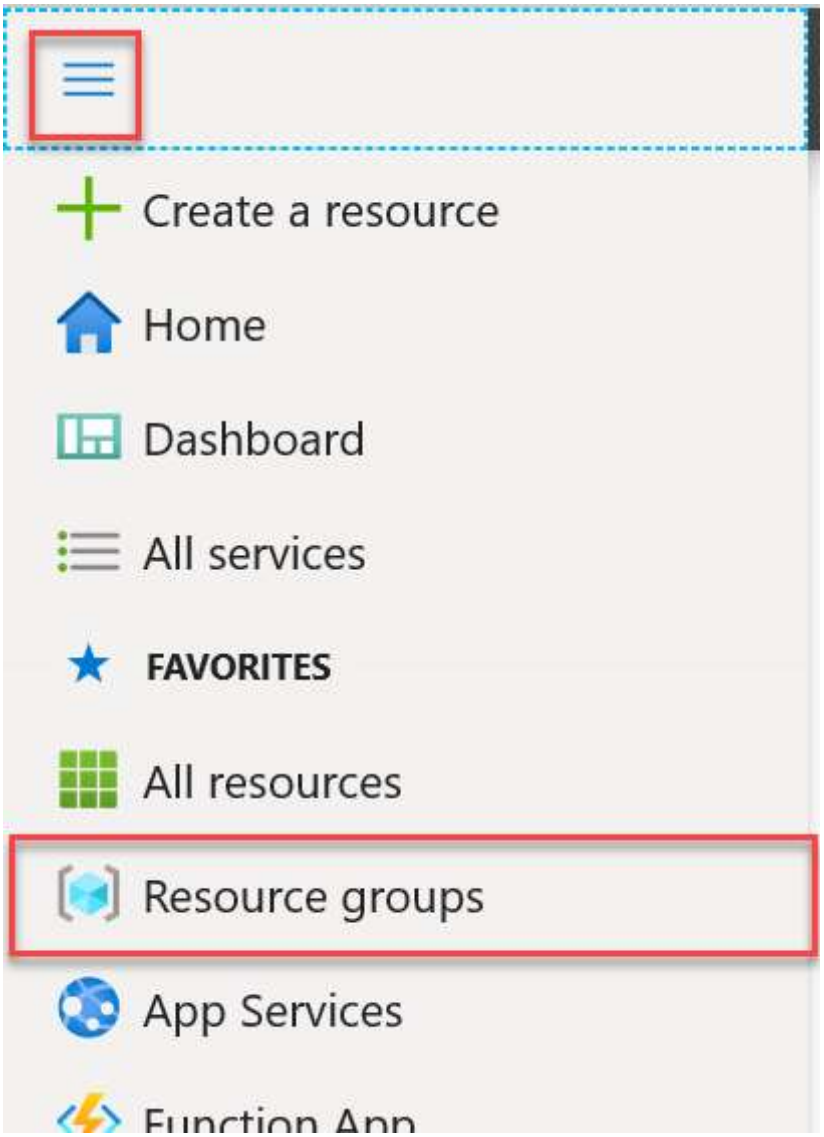
Exercise 1: Accessing the Azure Synapse Analytics workspace

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

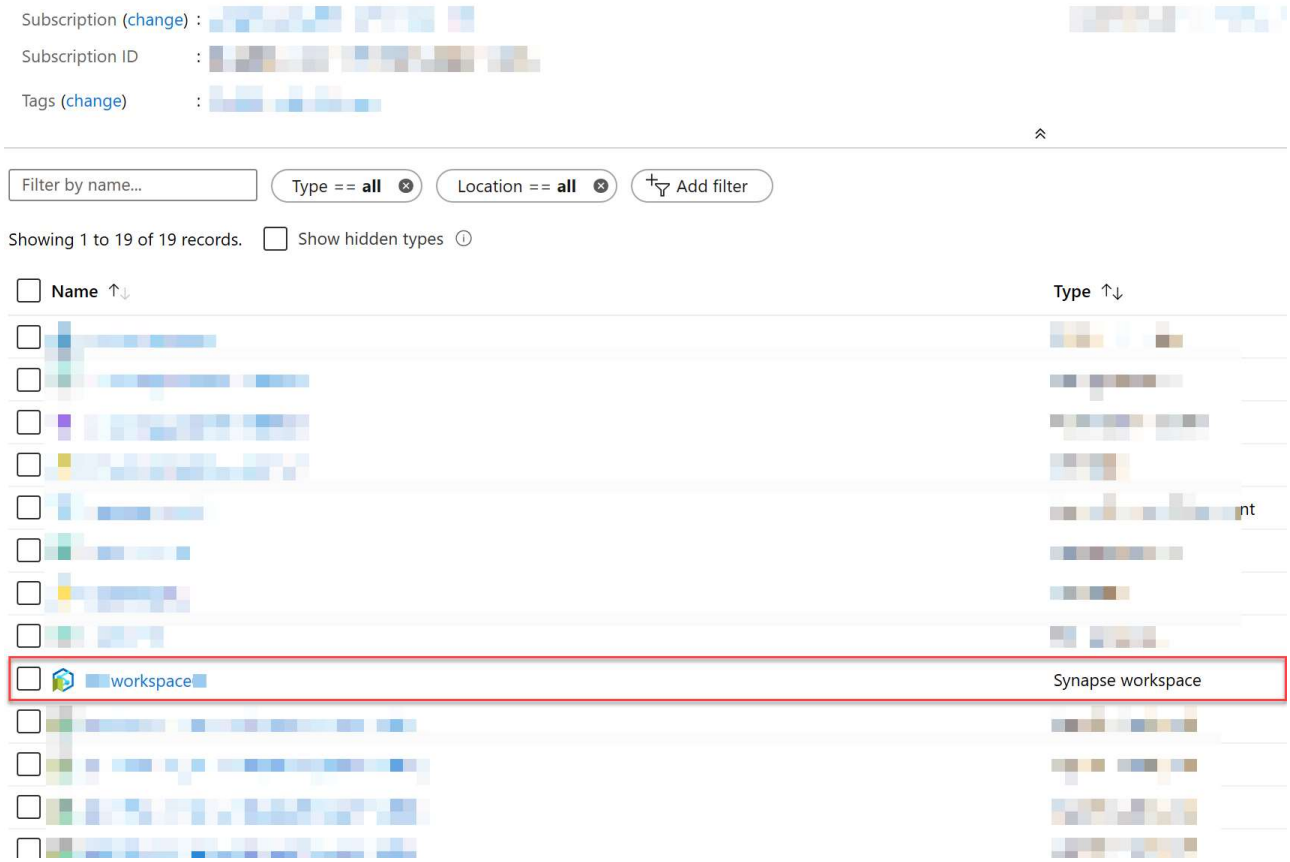
1. Log into the Azure Portal (https://portal.azure.com).
2. Expand the left menu, and select the **Resource groups** item.



The Azure Portal left menu is expanded with the Resource groups item highlighted.

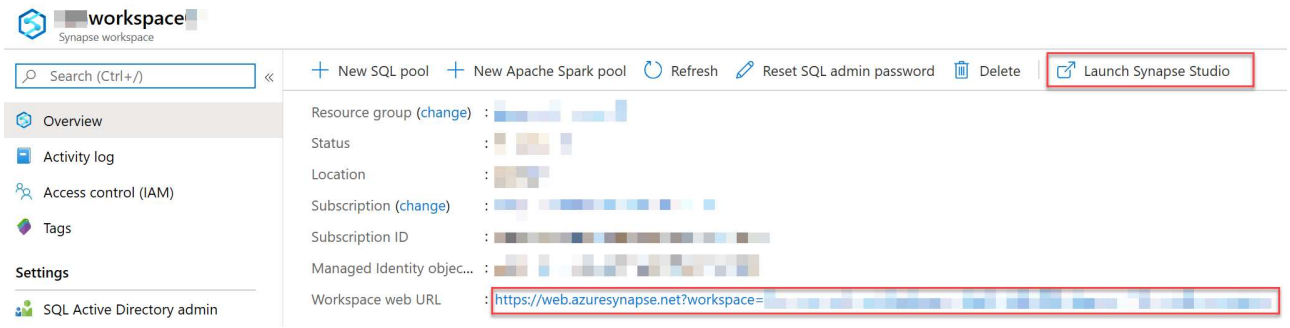
3. From the list of resource groups, select `WorkspaceResourceGroup`.

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



In the resource list, the Synapse Workspace item is selected.

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.



On the Synapse workspace resource screen, the Overview pane is shown with the Launch Synapse Studio button highlighted in the top toolbar. The Workspace web URL value is also highlighted.

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.

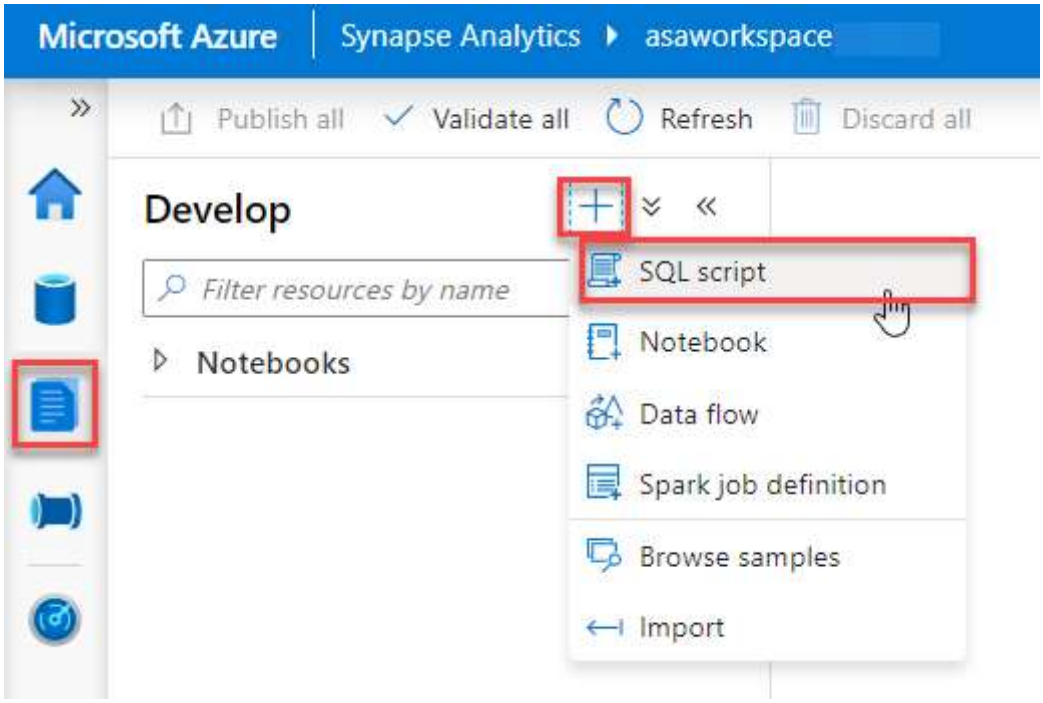
Table Distribution/Partition Type	**Recommended use**
Hash distribution	Tables that are larger than 2 GBs with infrequent insert/update/delete operations, works well for large fact tables in a star schema.
Round robin distribution	Default distribution, when little is known about the data or how it will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

Table Distribution/Partition Type	Recommended use
Replicated tables	Smaller lookup tables, less than 1.5 GB in size.

Task 1: Create the sale table

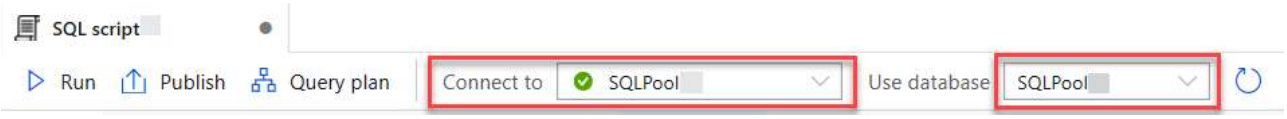
Over the past 5 years, Wide World Importers has amassed over 3 billion rows of sales data. With this quantity of data, the storage consumed would be greater than 2 GB. While we will be using only a subset of this data for the lab, we will design the table for the production environment. Using the guidance 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.



The left menu is expanded with the Develop item selected. The Develop blade has the + button expanded with the SQL script item highlighted.

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



The query tab toolbar menu is displayed with the Connect to set to the SQL Pool.

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, 201
80701, 20180801, 20180901, 20181001, 20181101, 20181201,
      20190101, 20190201, 20190301, 20190401, 20190501, 20190601, 201
90701, 20190801, 20190901, 20191001, 20191101, 20191201)
  )
);
```

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

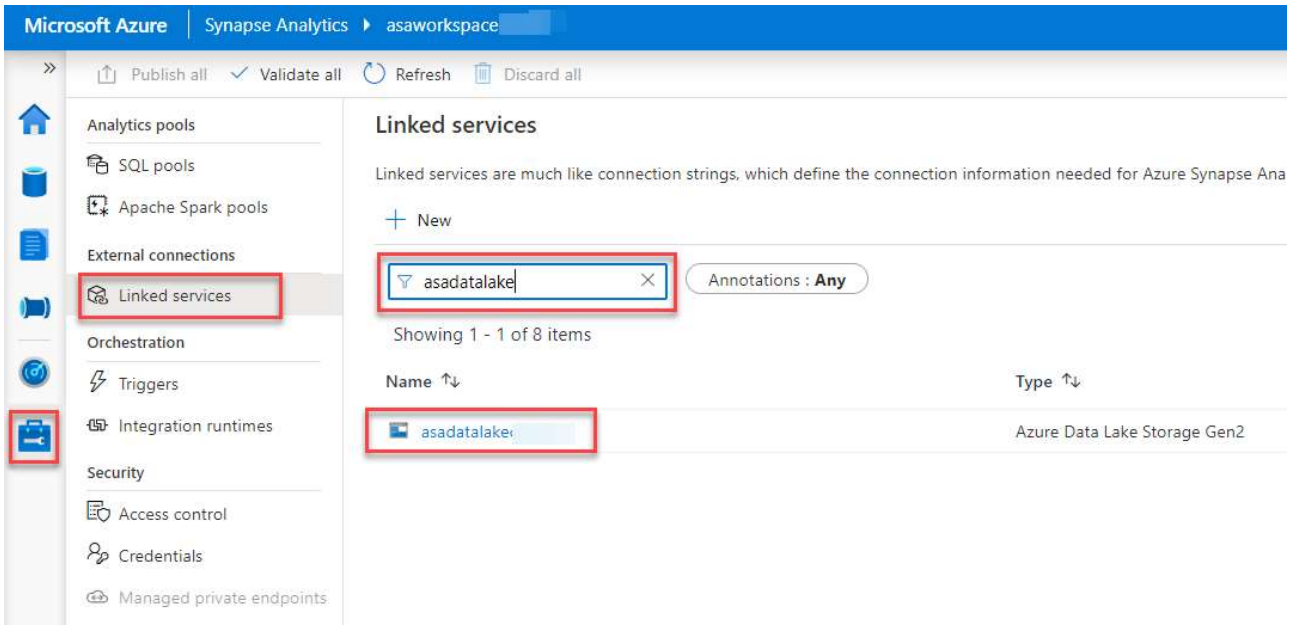
The top toolbar menu is displayed with the Discard all button highlighted.

Task 2: Populate the sale table

> **Note**: *This task involves a long data loading activity (approximately 45 minutes in duration). Once you have triggered the pipeline, please continue to the next task.*

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.



The Manage item is selected from the left menu. The Linked services menu item is selected on the blade. On the Linked services screen the term asadatalake{SUFFIX} is entered in the search box and the asadatalake{SUFFIX} Azure Blob Storage item is selected from the filtered results list.

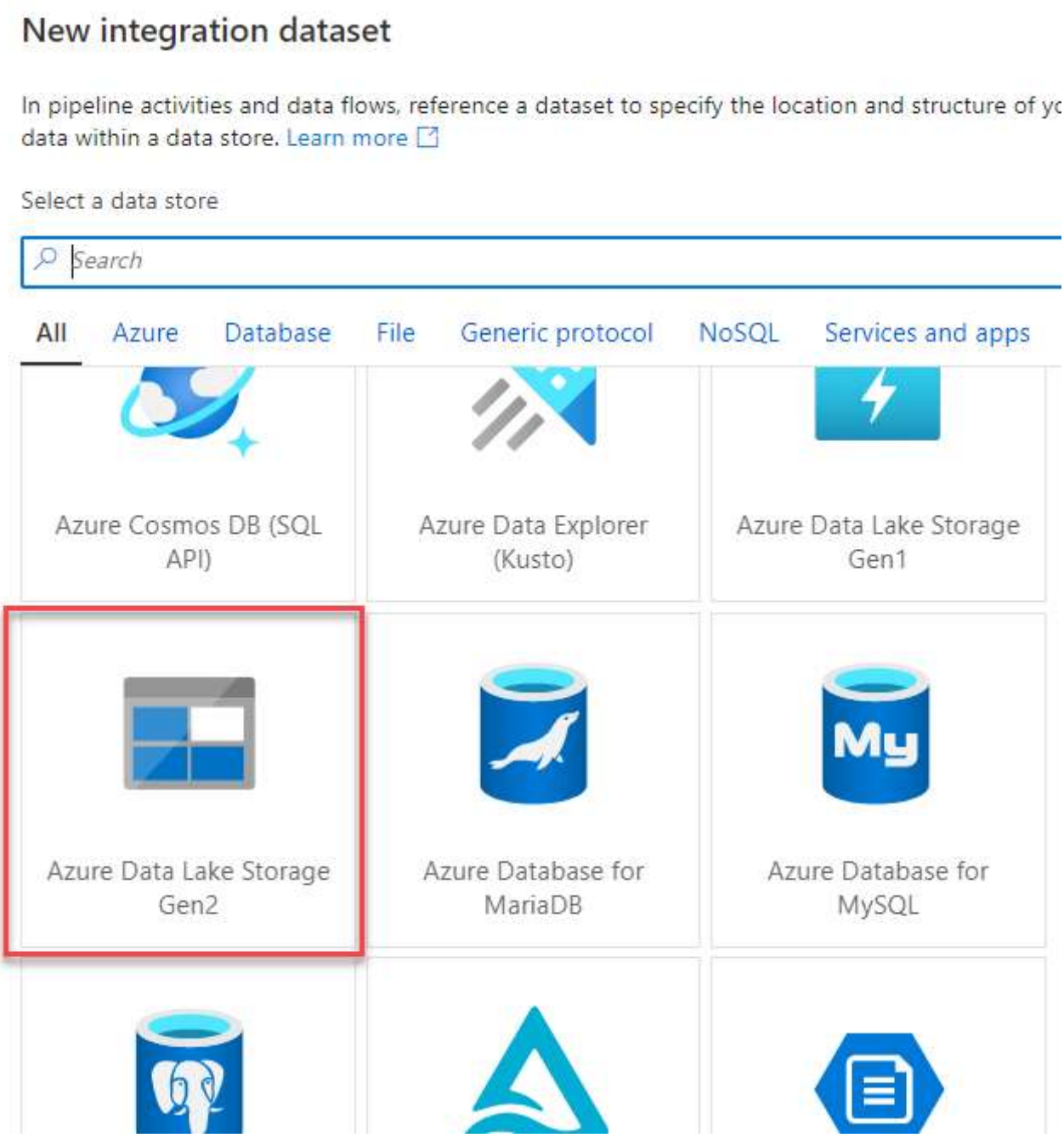
2. The sale data for each day is stored in a separate parquet file which is placed in storage following a known convention. In this lab, we are interested in populating the Sale table with only 2018 and 2019 data. Investigate the structure of the data by selecting the **Data** tab, and in the **Data**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

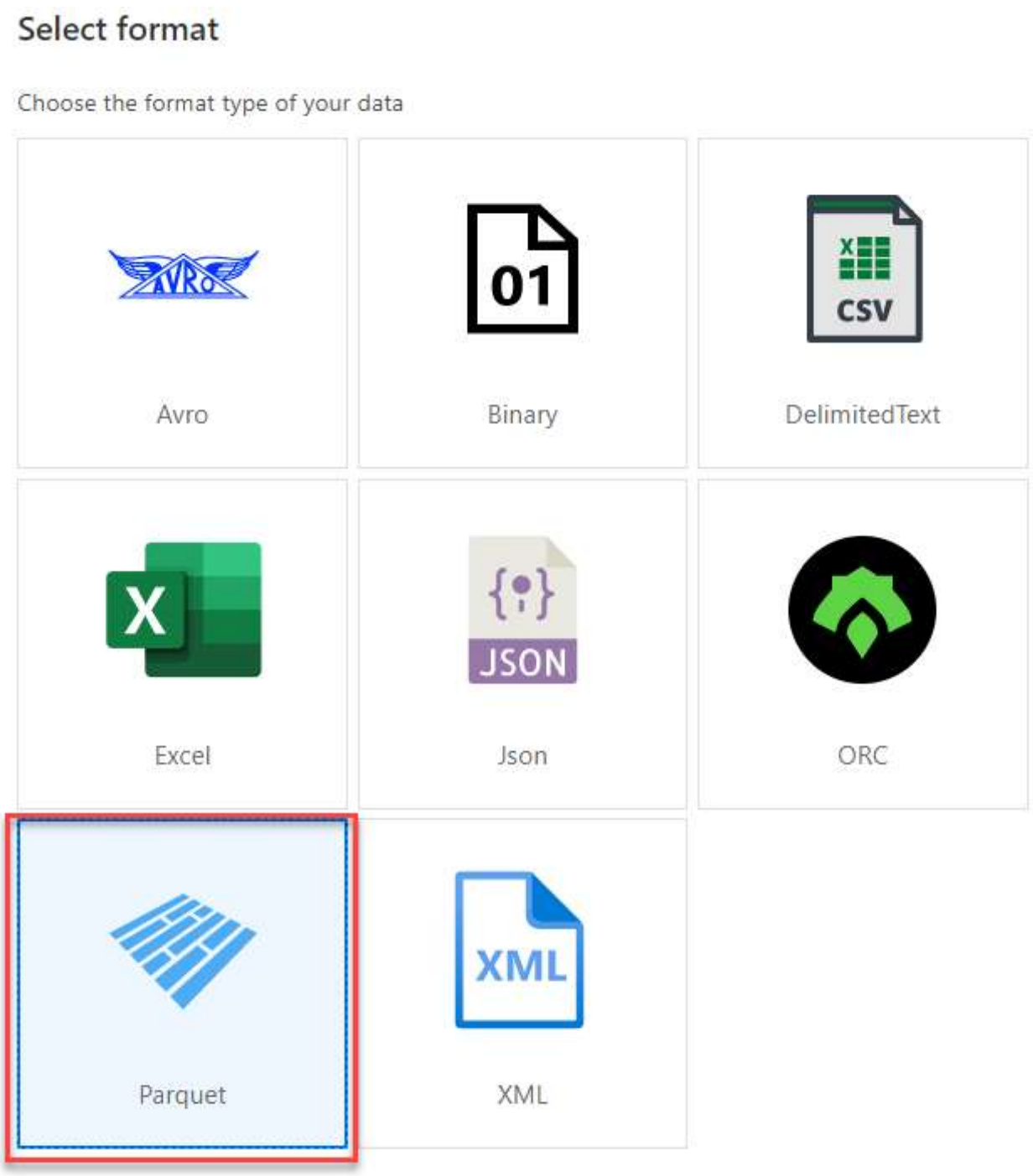
3. Create a new Dataset by selecting **Data** from the left menu, expanding the **+** button on the Data blade and selecting **Integration Dataset**. We will be creating a dataset that will point to the root folder of the sales data in the data lake.

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



The New dataset blade is displayed with the All tab selected, the Azure Data Lake Storage Gen2 item is selected from the list.

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

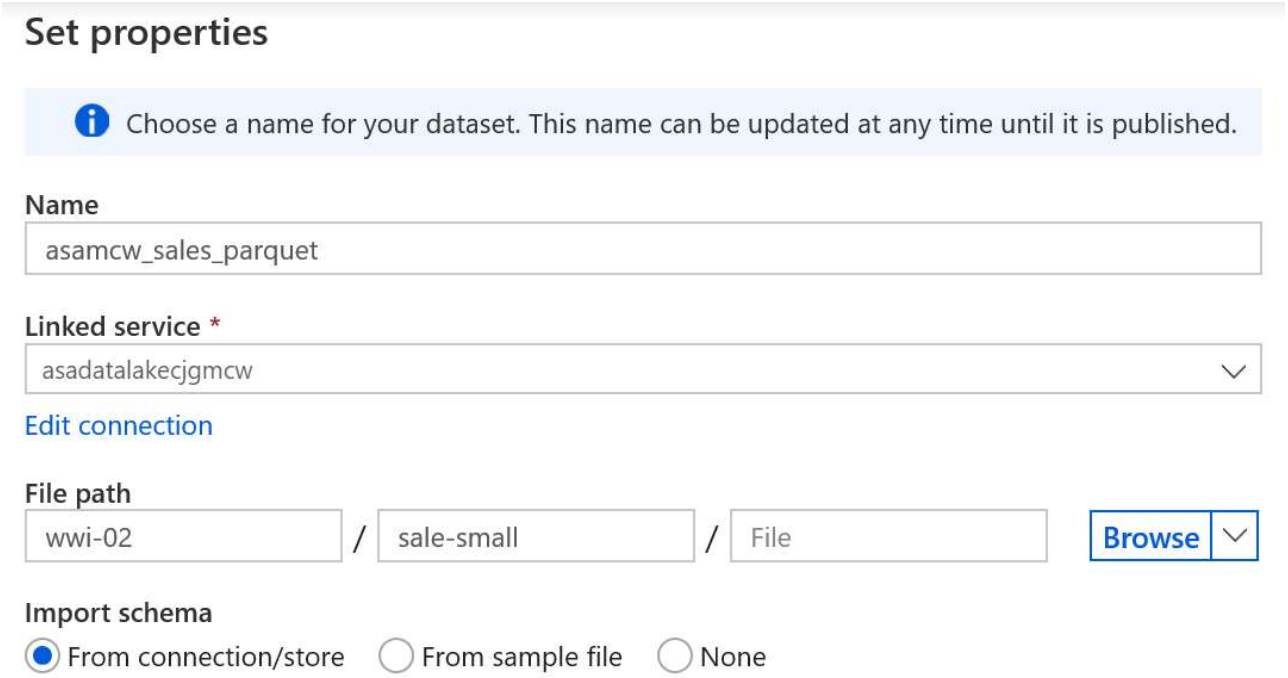


In the Select format screen, the Parquet item is highlighted.

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.**

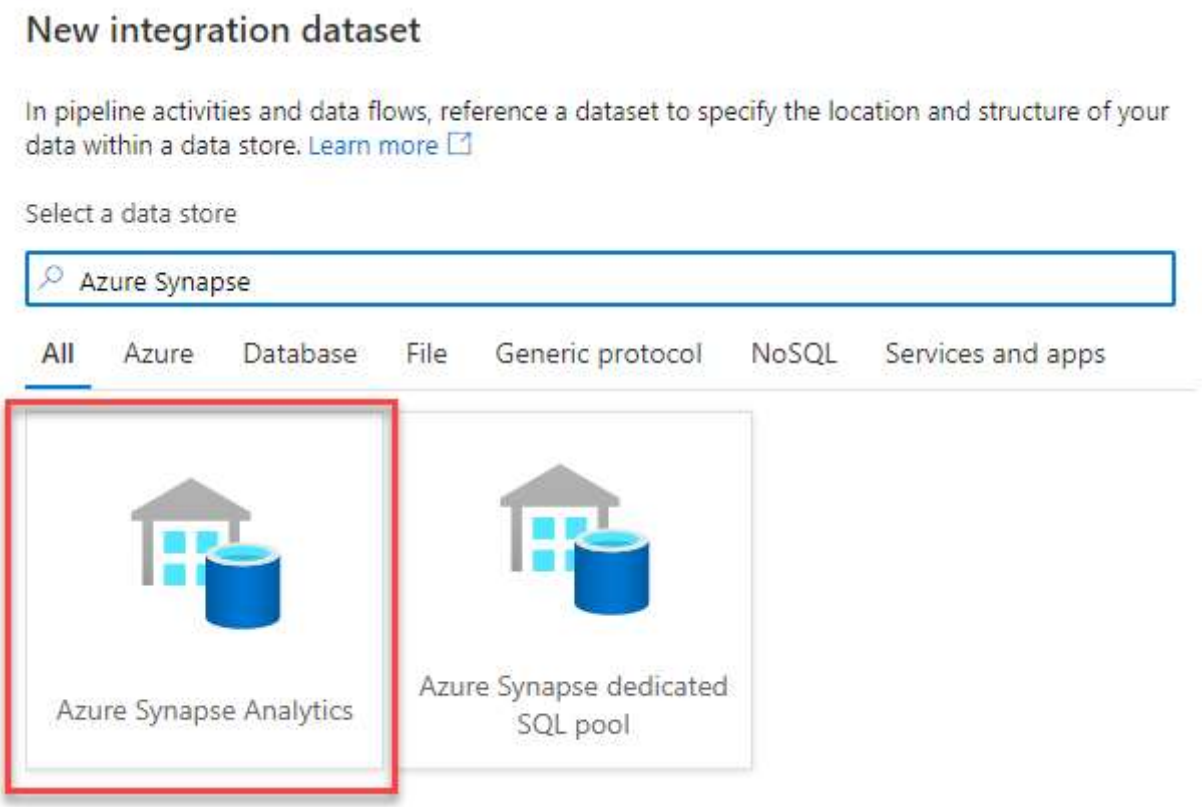
Field	Value
File path - Folder	Enter **sale-small**.
Import schema	**From connection/store**



The Set properties blade is displayed with fields populated with the values from the preceding table.

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 **Integration dataset**.

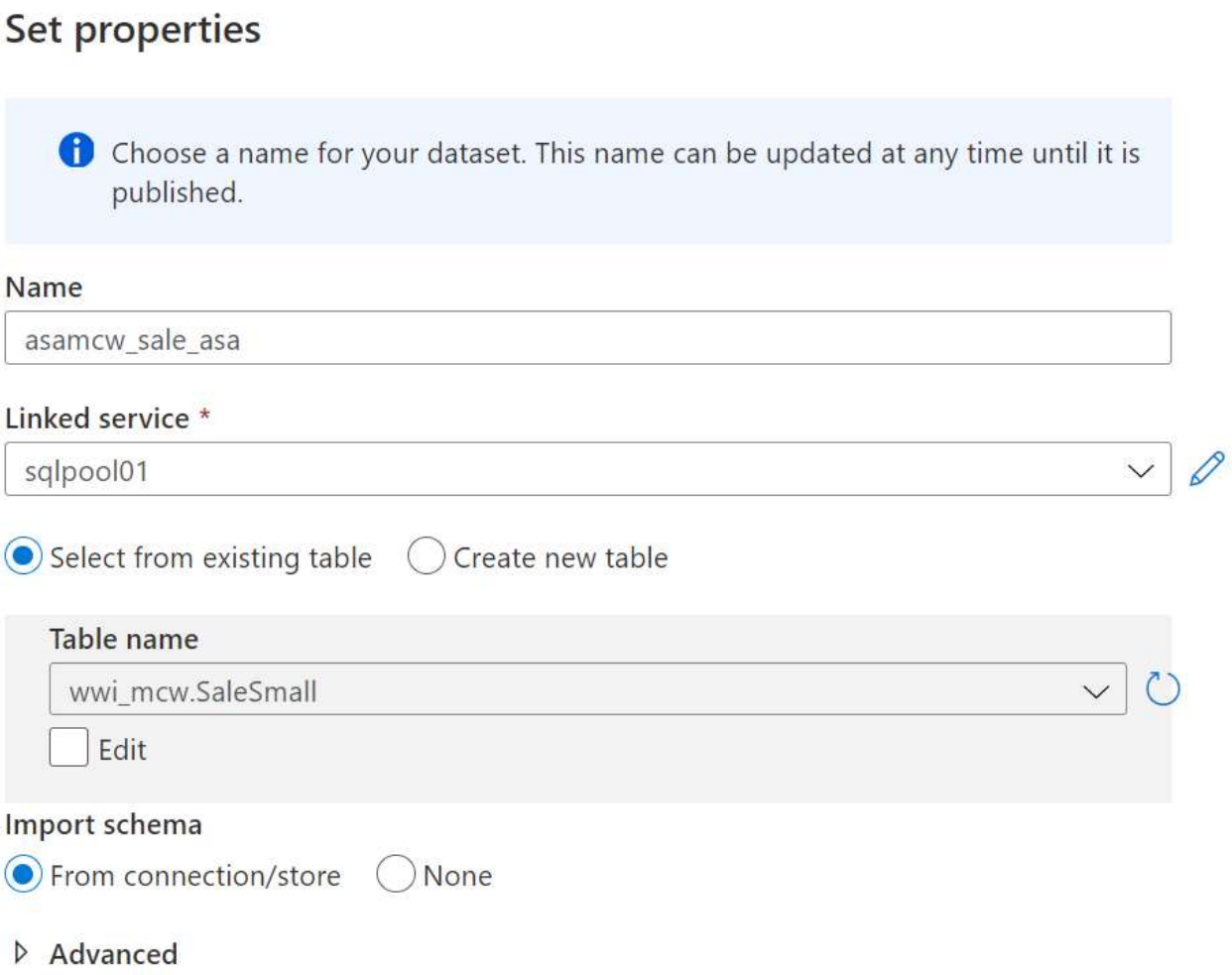
8. On the **New integration dataset** blade, enter **Azure Synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue**.



The New integration dataset form is shown with Azure Synapse entered in the search box and the Azure Synapse Analytics item highlighted.

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**

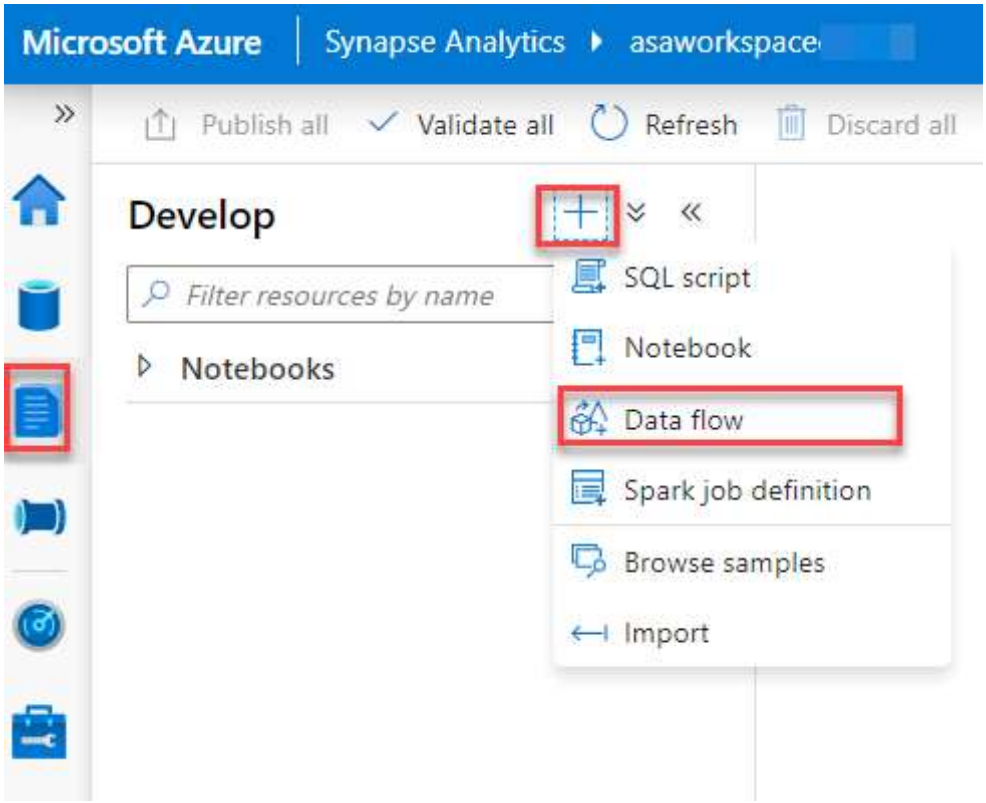


The Set properties blade is populated with the values specified in the preceding table.

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.

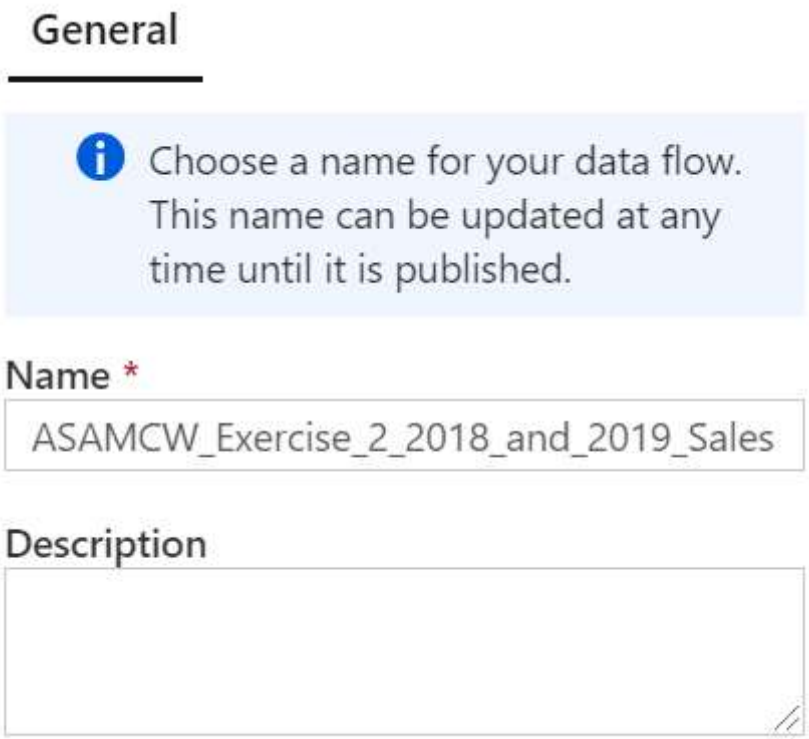
The top toolbar is displayed with the Publish all button highlighted.

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**.



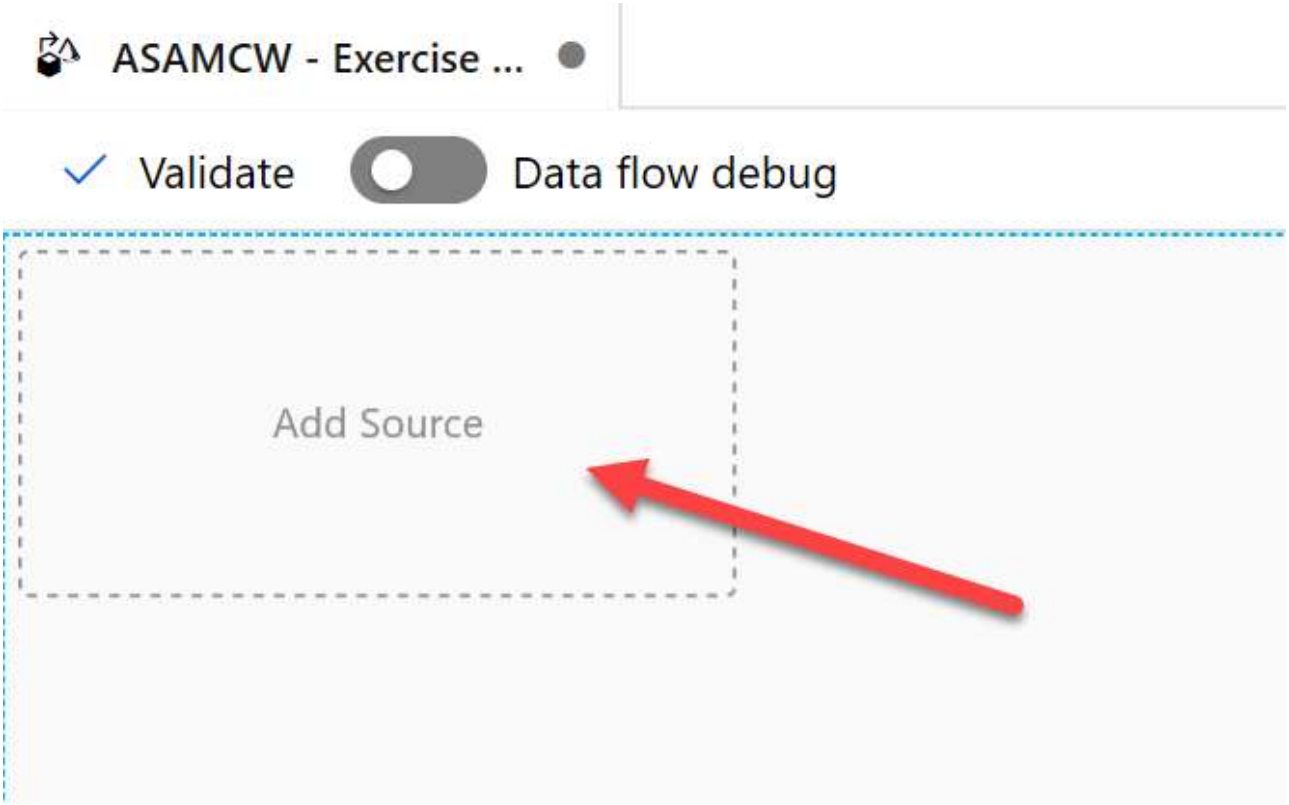
From the left menu, the Develop item is selected. From the Develop blade the + button is expanded with the Data flow item highlighted.

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.



The General tab is displayed with ASAMCW_Exercise_2_2018_and_2019_Sales entered as the name of the data flow.

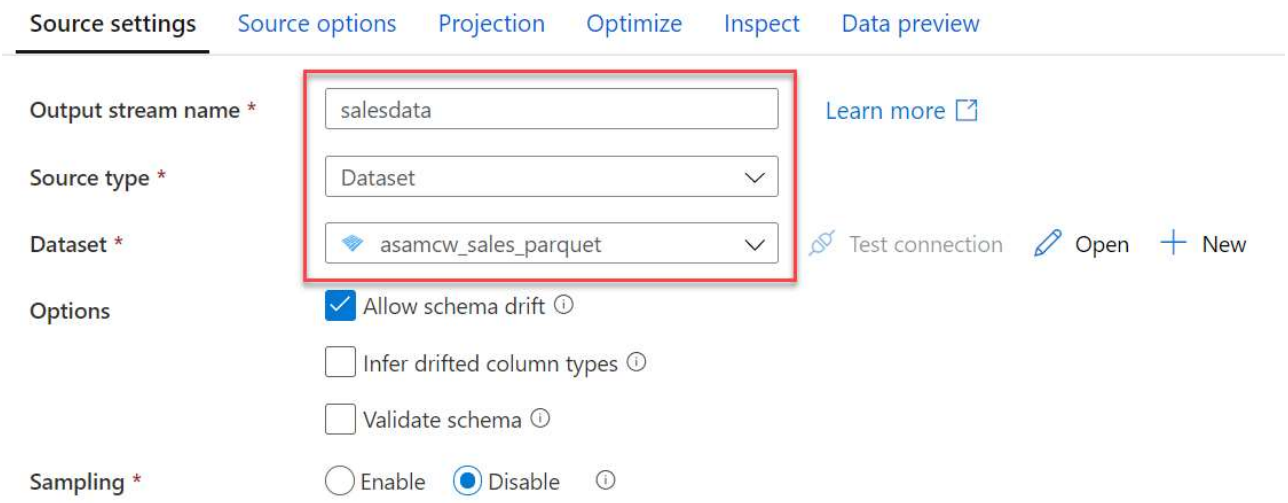
13. In the data flow designer window, select the **Add Source** box.



The Add source box is highlighted in the data flow designer window.

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.**
Source type	**Dataset**
Dataset	**asamcw_sales_parquet**

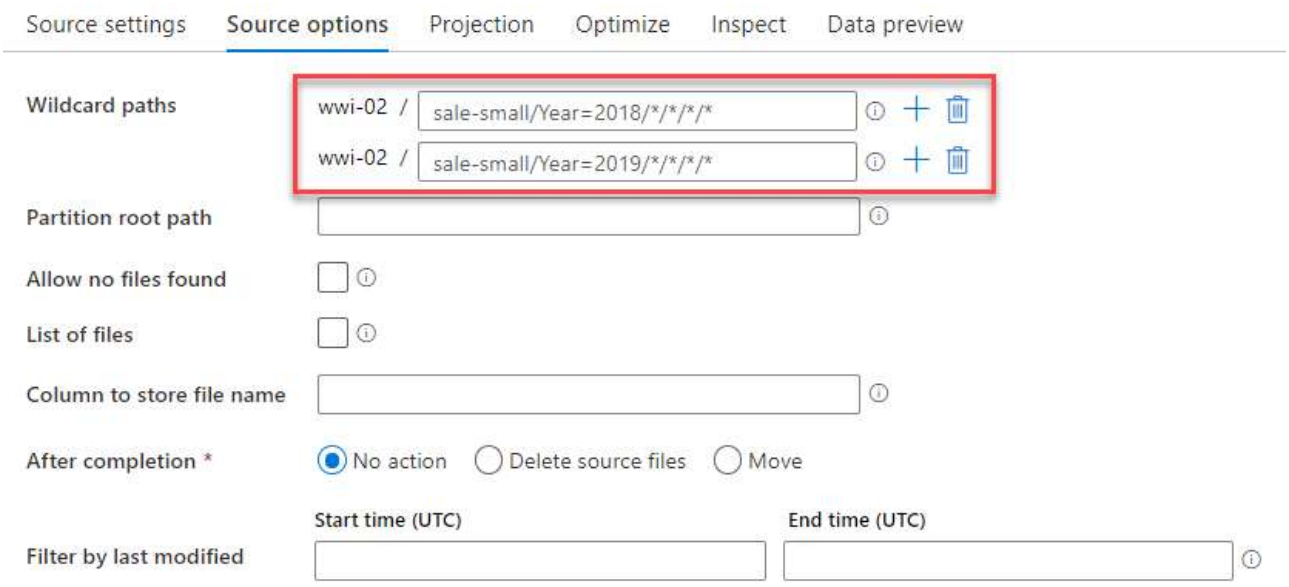


The Source settings tab is selected displaying the Output stream name set to salesdata and the selected dataset being asamcw_sales_parquet.

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:

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

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



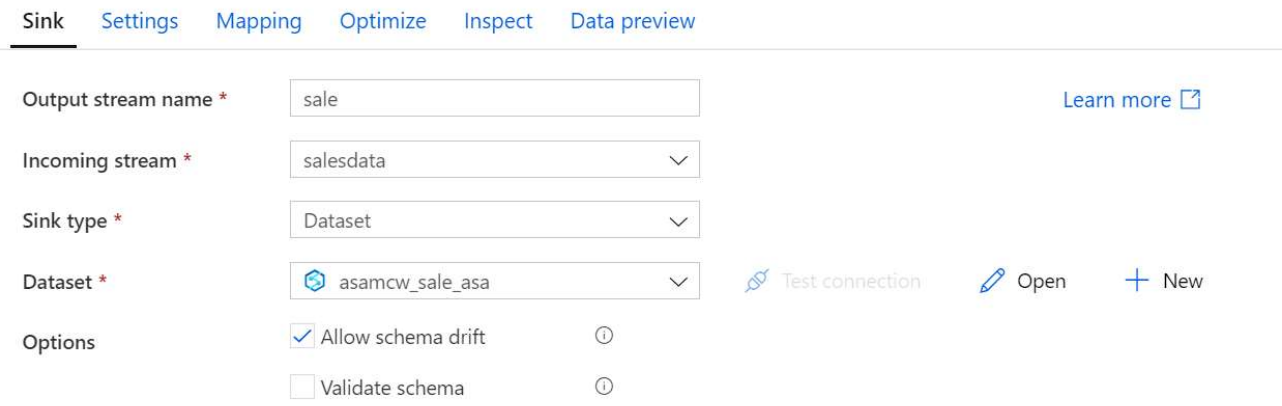
The Source options tab is selected with the above wildcard paths highlighted.

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.

The + button is highlighted toward the bottom right of the source element on the data flow designer.

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**

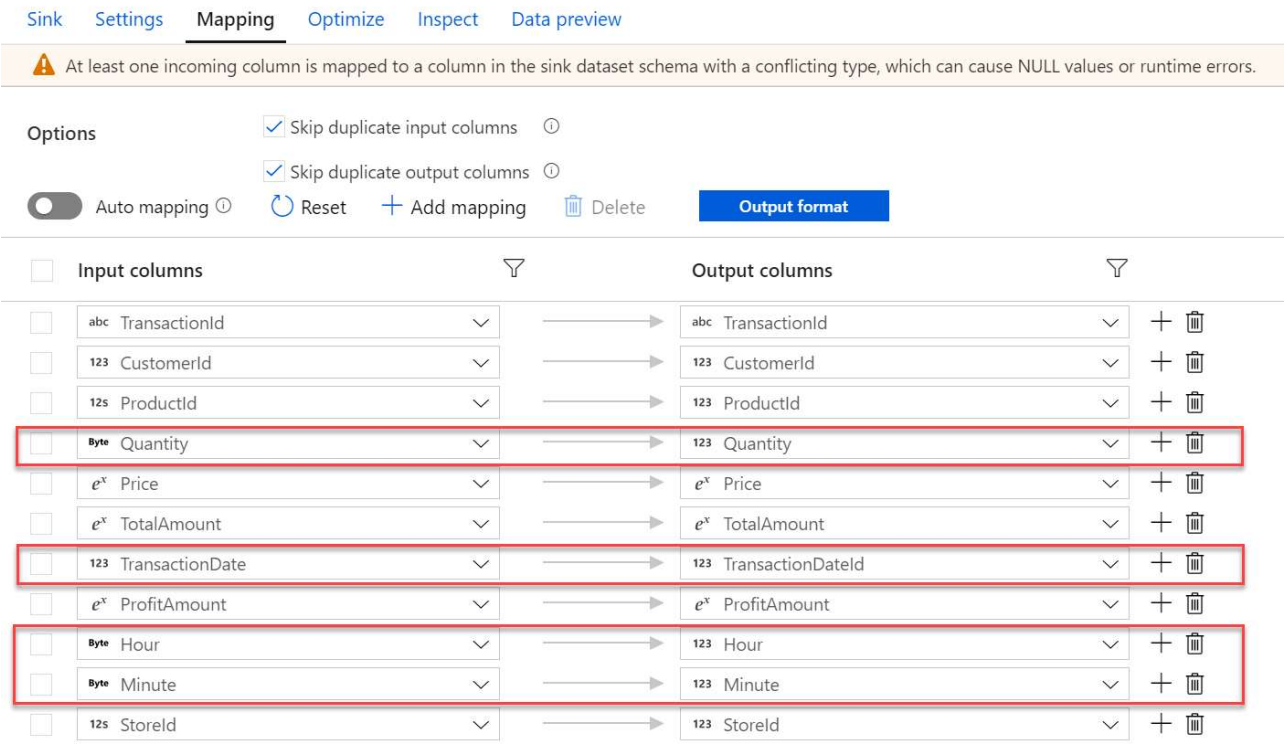


The Sink tab is displayed with the form populated with the values from the preceding table.

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
TransactionDate	TransactionDateId

Input column	Output column
Hour	Hour
Minute	Minute



The Mapping tab is selected with the Auto mapping toggle set to the off position. The + Add mapping button is highlighted along with the mapping entries specified in the preceding table.

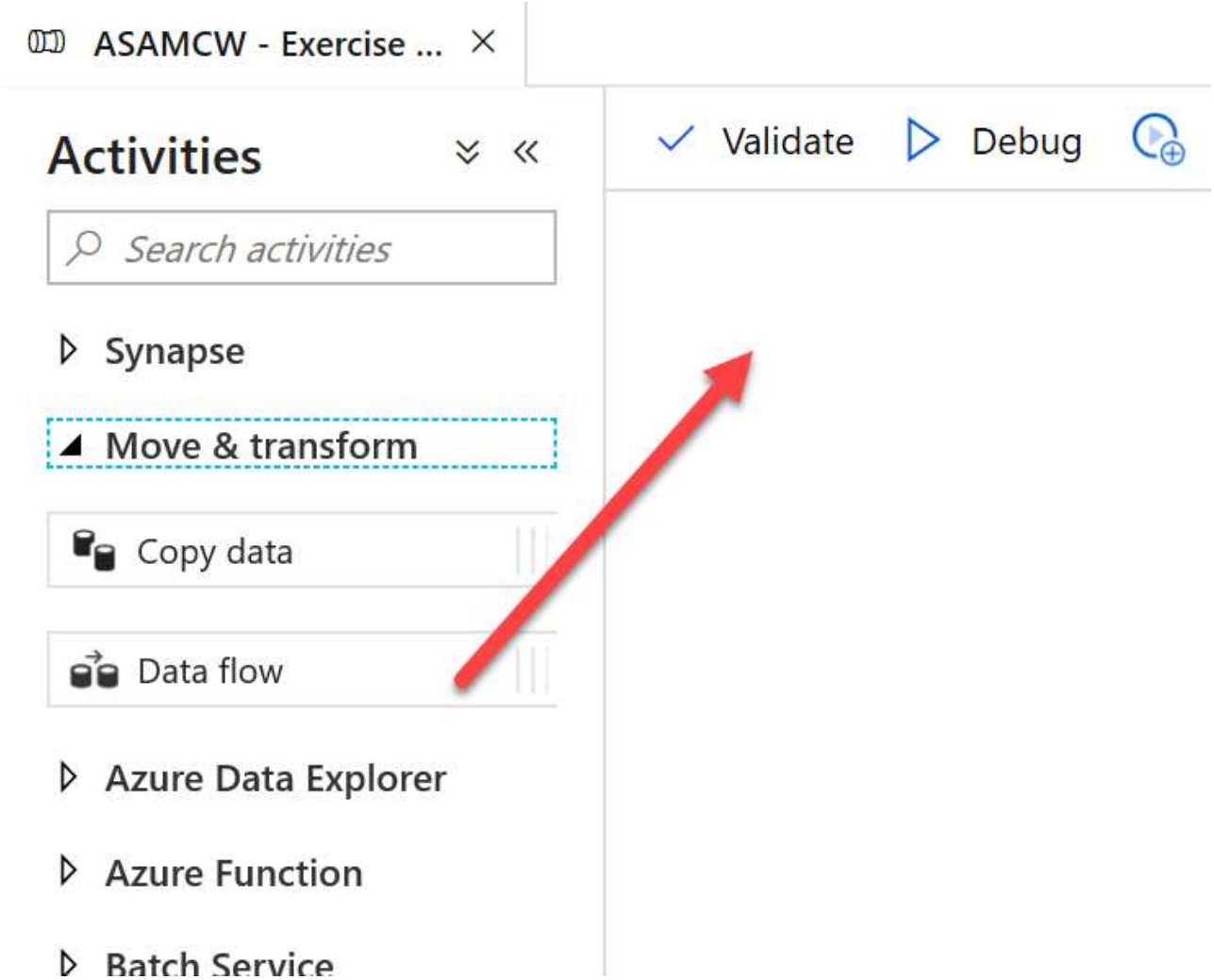
19. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to deploy the new data flow to the workspace.

The top toolbar is displayed with the Publish all button highlighted.

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 **Integrate** blade, expand the **+** button and select **Pipeline**.

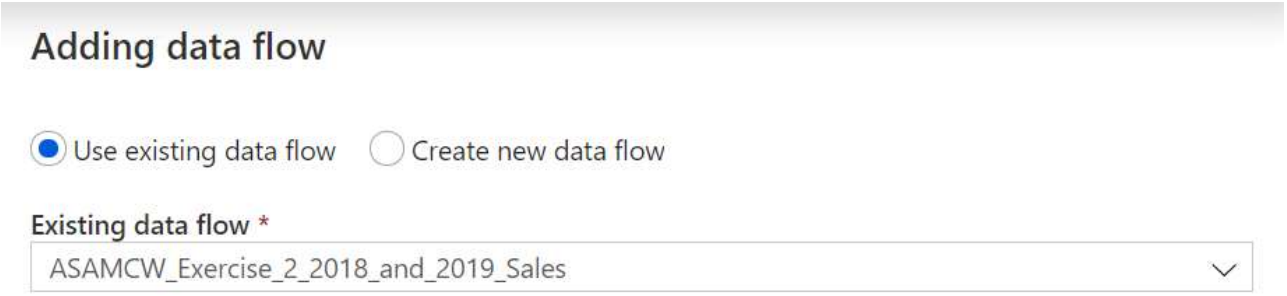
21. On the **Properties** blade, Enter **ASAMCW - Exercise 2 - Copy Sale Data** as the Name of the pipeline.

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



The Activities menu of the pipeline is displayed with the Move and transform section expanded. An arrow indicating a drag operation shows adding a Data flow activity to the design surface of the pipeline.

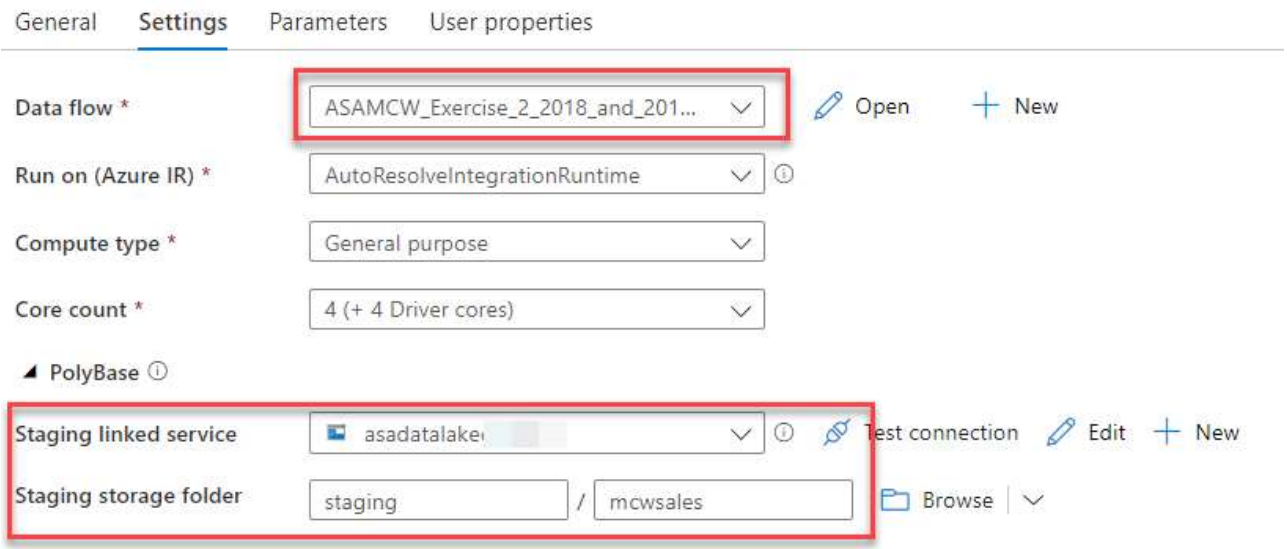
23. In the **Adding data flow** blade, ensure **Use existing data flow** is selected, and choose **ASAMCW_Exercise_2_2018_and_2019_Sales** from the select list and select **OK.**



The Adding data flow blade is displayed populated with the appropriate values.

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**.



The data flow activity Settings tab is displayed with the fields specified in the preceding table highlighted.

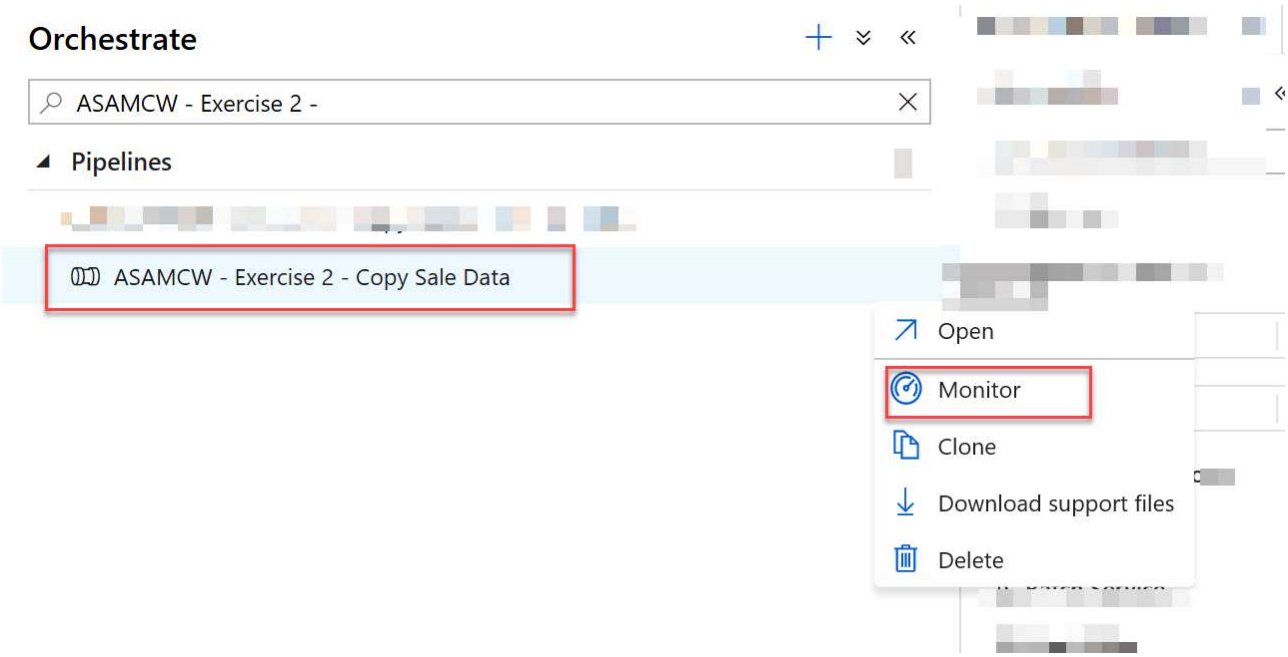
25. In the top toolbar, select **Publish all** to publish the new dataset definitions. When prompted, select the **Publish** button to commit the changes.

The top toolbar is displayed with the Publish all button highlighted.

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.

> **Note**: *This pipeline is processing 667,049,970 rows of 2018 and 2019 sales data. Please proceed to the next task! When the pipeline completes, you will see a notification in Azure Synapse Analytics studio. At that time you can verify your data if you choose (step 29 in this exercise).*

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



In the Integrate blade, the Action menu is displayed with the Monitor item selected on the ASAMCW - Exercise 2 - Copy Sale Data pipeline.

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**.

> **Note**: *Feel free to proceed to the following tasks in this exercise while this pipeline runs.*

On the pipeline runs screen, a successful pipeline run is highlighted in the table.

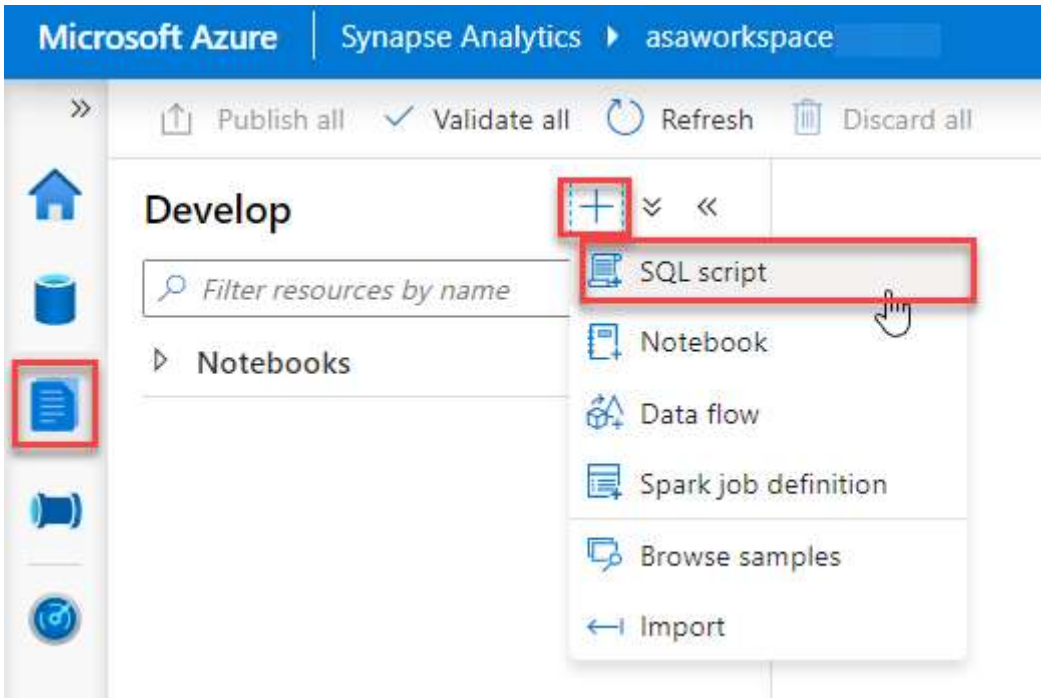
29. Verify the table has populated by creating a new query. Select the **Develop** item from the left menu, and in the **Develop** blade, expand the **+** button, and select **SQL script**. In the query window, be sure to connect to the SQL Pool database (`SQLPool01`), then paste and run the following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



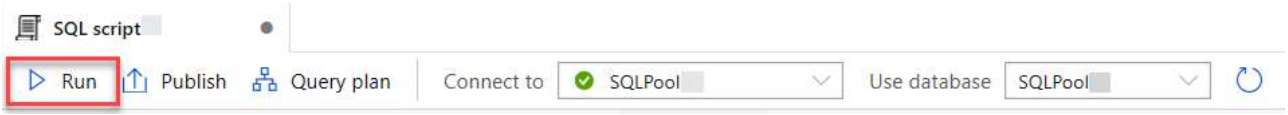
The left menu is expanded with the Develop item selected. The Develop blade has the + button expanded with the SQL script item highlighted.

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

The query tab toolbar menu is displayed with the Connect to set to the SQL Pool.

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
```



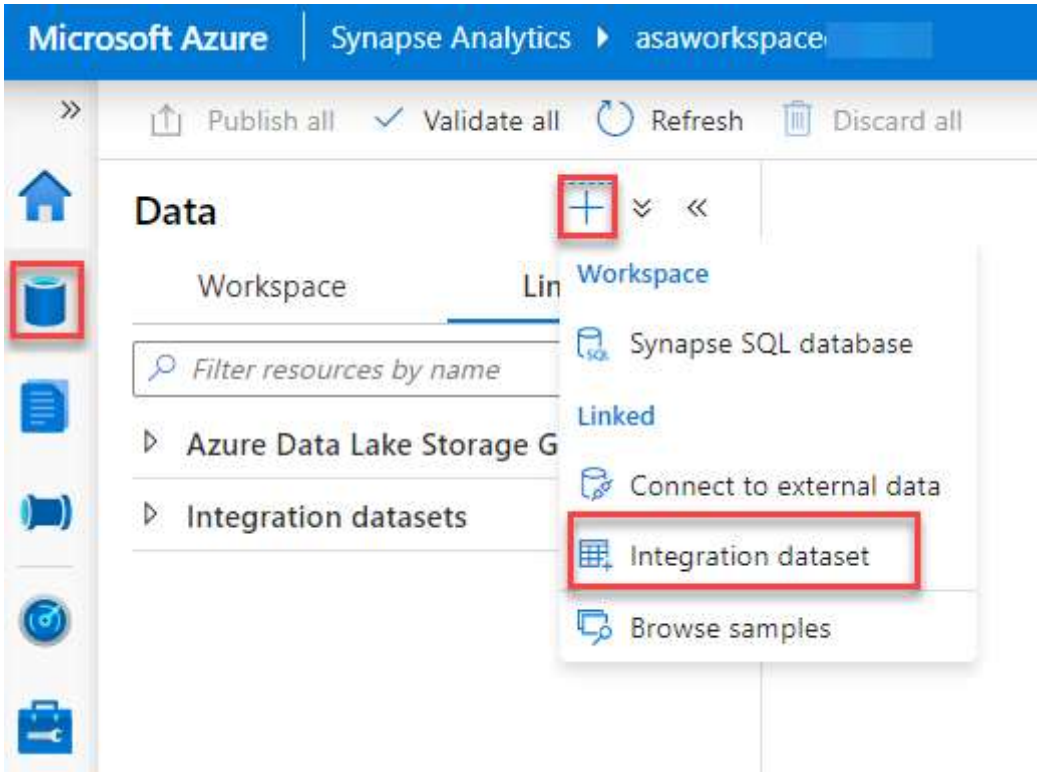
The query tab toolbar is displayed with the Run button selected.

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

The top toolbar menu is displayed with the Discard all button highlighted.

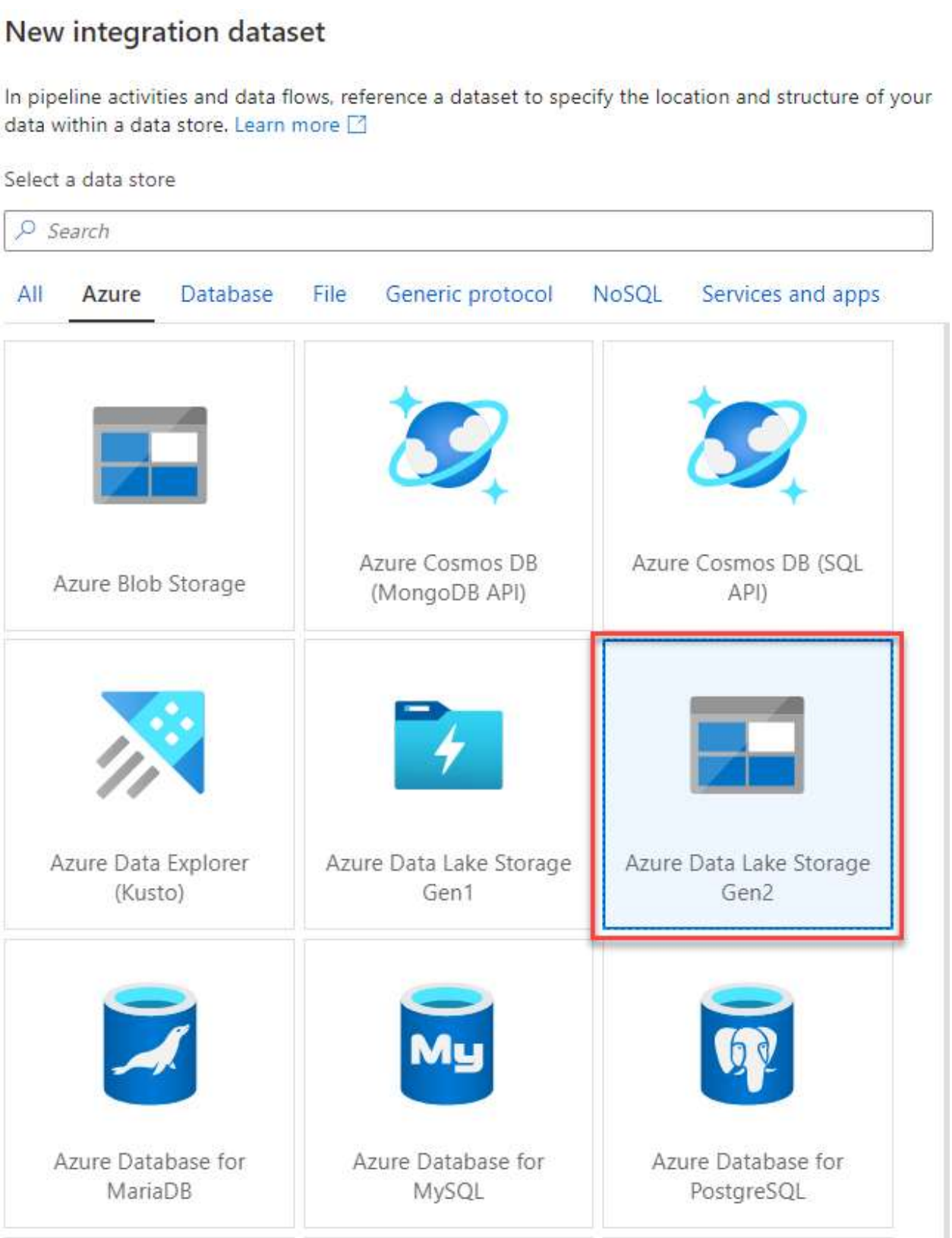
Task 4: Populate the customer information table

1. 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 **Integration Dataset**.



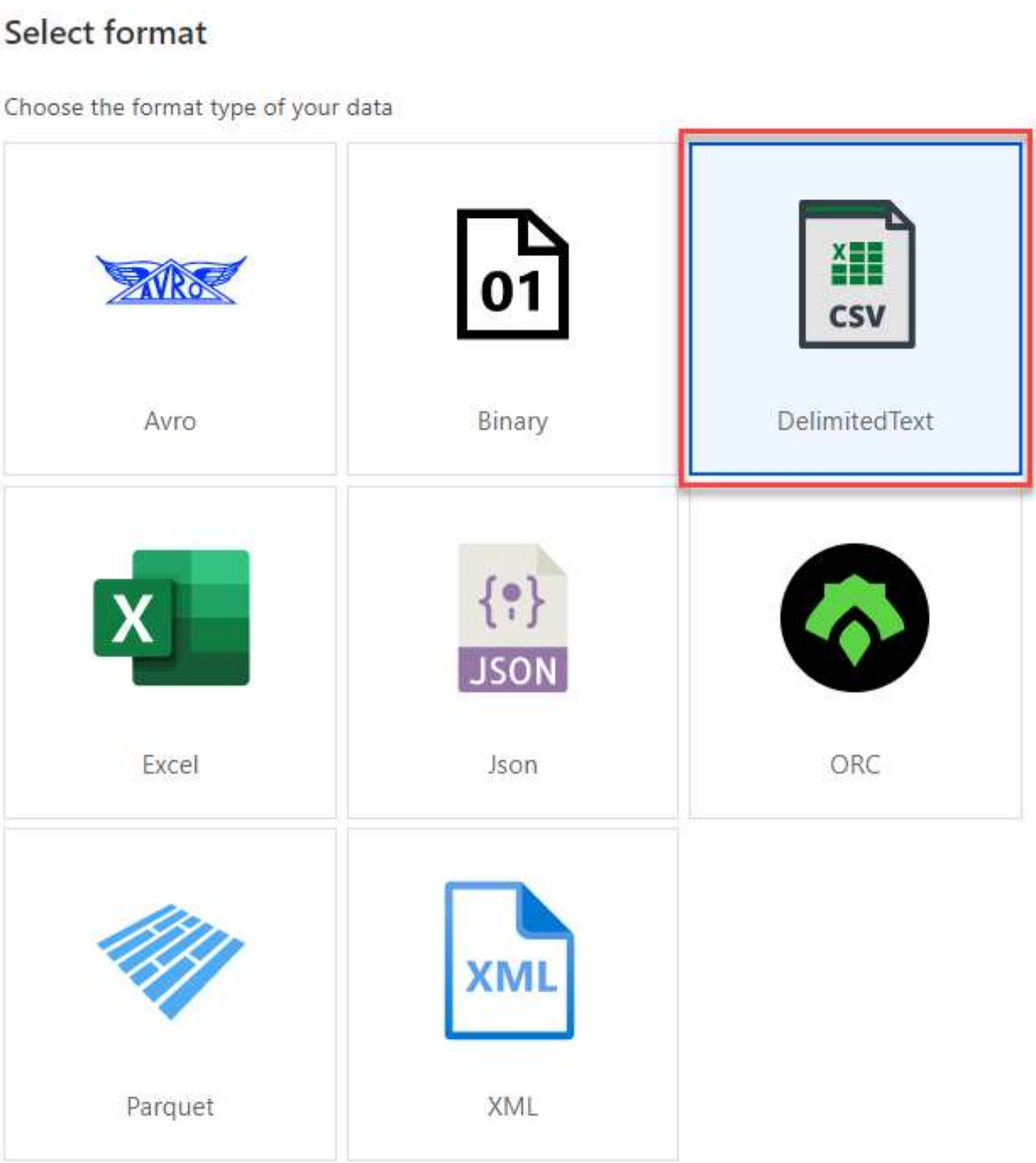
The Data item is selected from the left menu. On the Data blade, the + button is expanded with the Dataset item highlighted.

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



On the New dataset blade, the All tab is selected and the Azure Data Lake Gen2 item is highlighted.

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

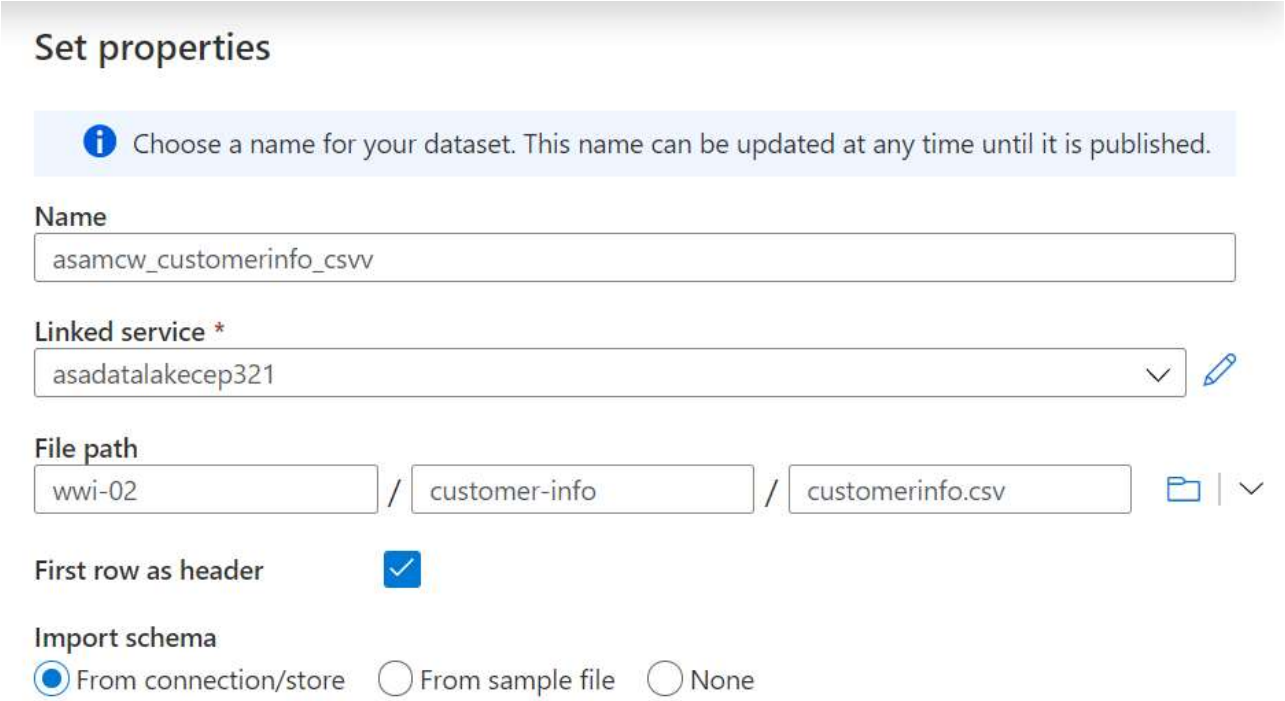


On the Select format blade the CSV Delimited Text item is highlighted.

4. 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.**

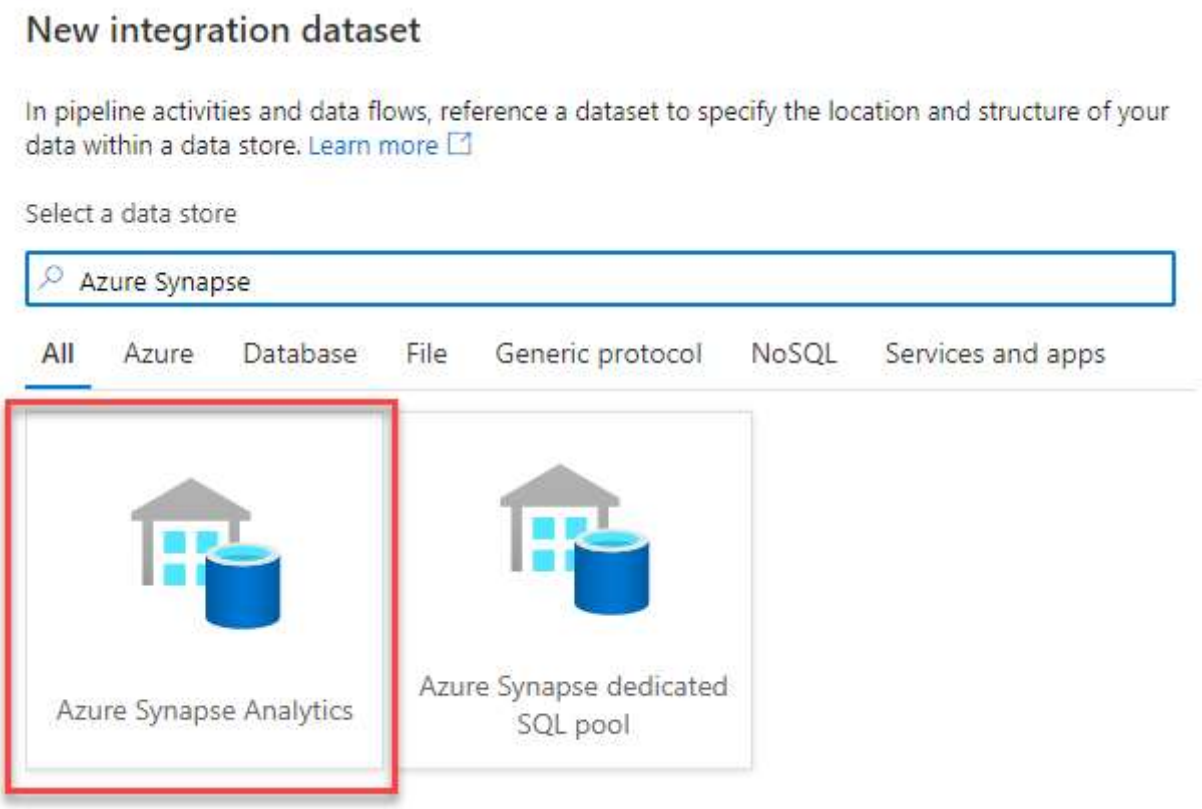
Field	Value
File Path - Directory	Enter **customer-info**.
File Path - File	Enter **customerinfo.csv**.
First row as header	Checked
Import schema	Select **From connection/store**.



The Set properties form is displayed with the values specified in the previous table.

5. 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 and select **Integration dataset**.

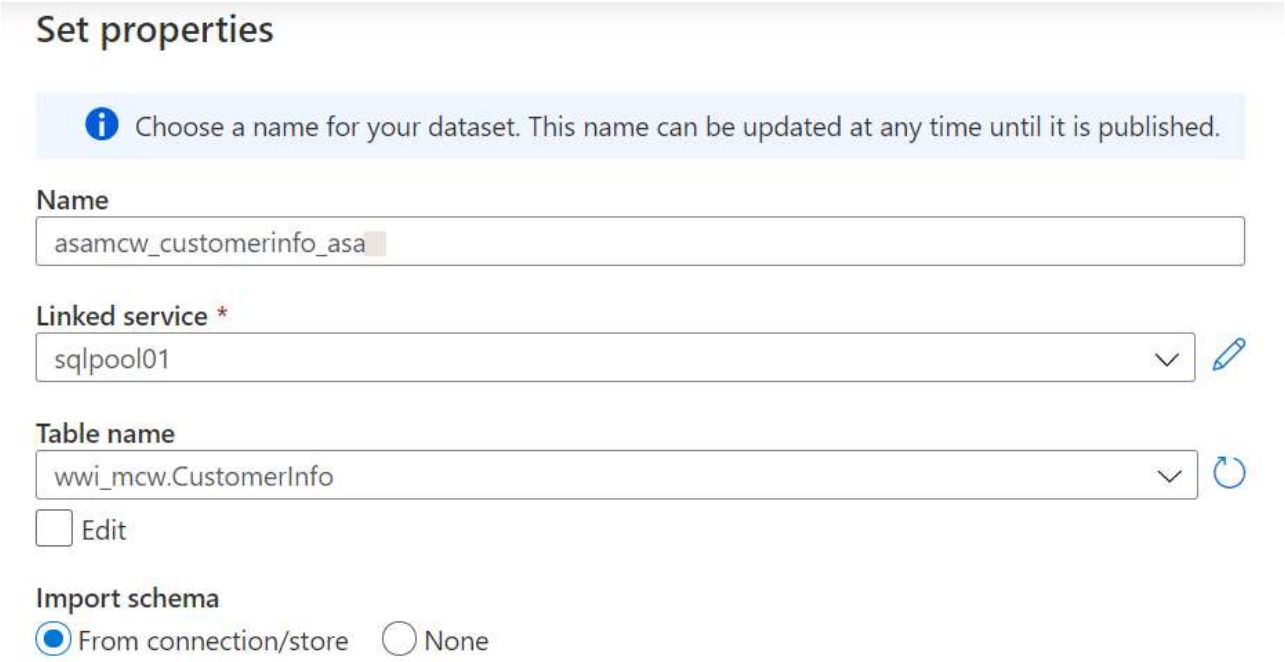
6. On the **New integration dataset** blade, enter **Azure Synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue**.



The New integration dataset form is shown with Azure Synapse entered in the search box and the Azure Synapse Analytics item highlighted.

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

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

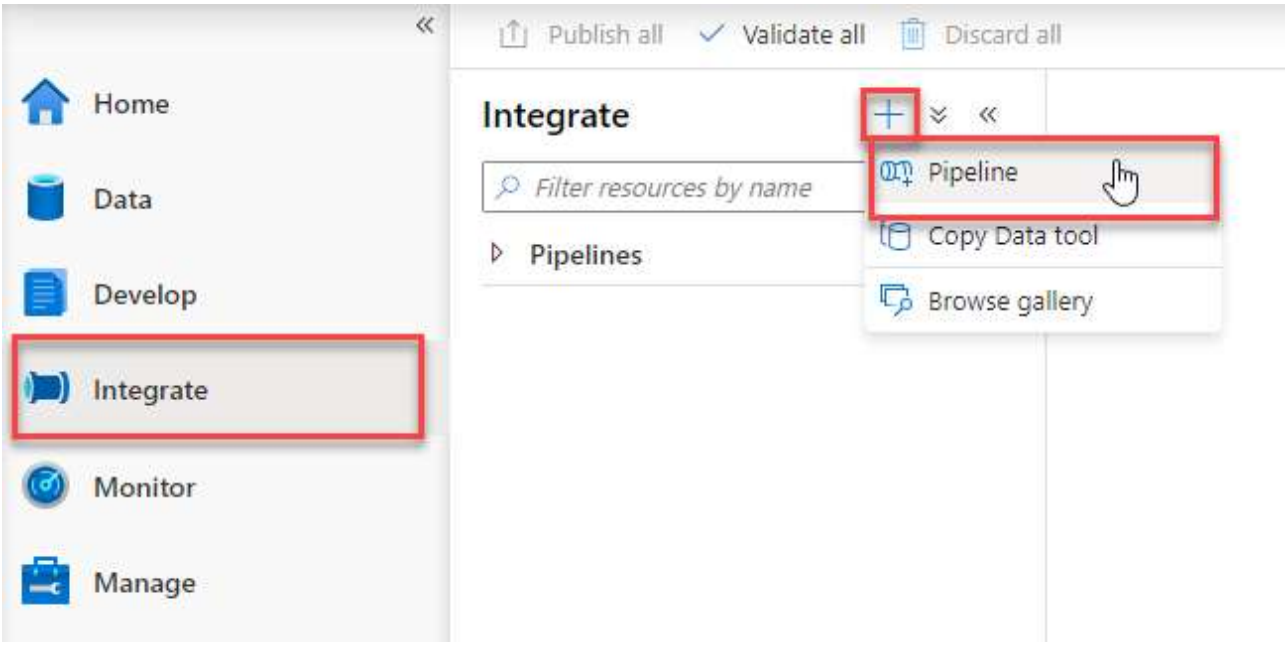


The Set properties blade is populated with the values specified in the preceding table.

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

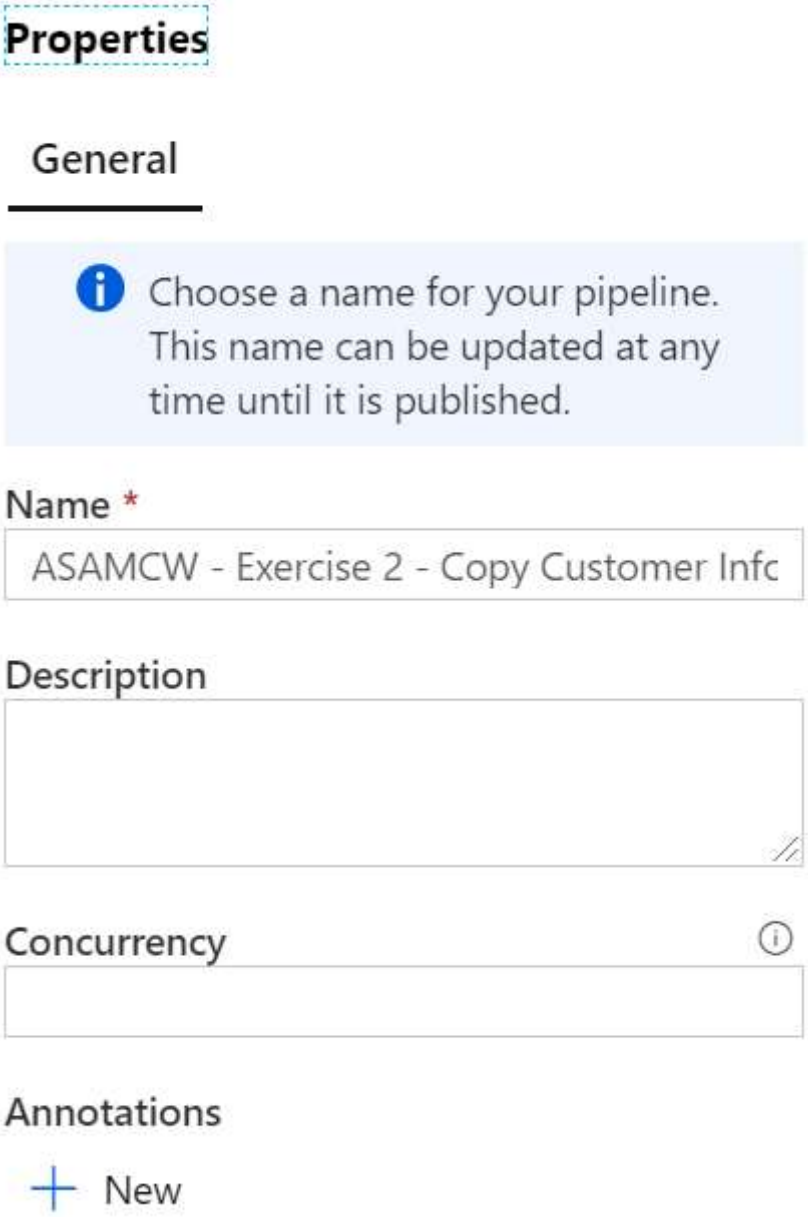
The top toolbar is displayed with the Publish all button highlighted.

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



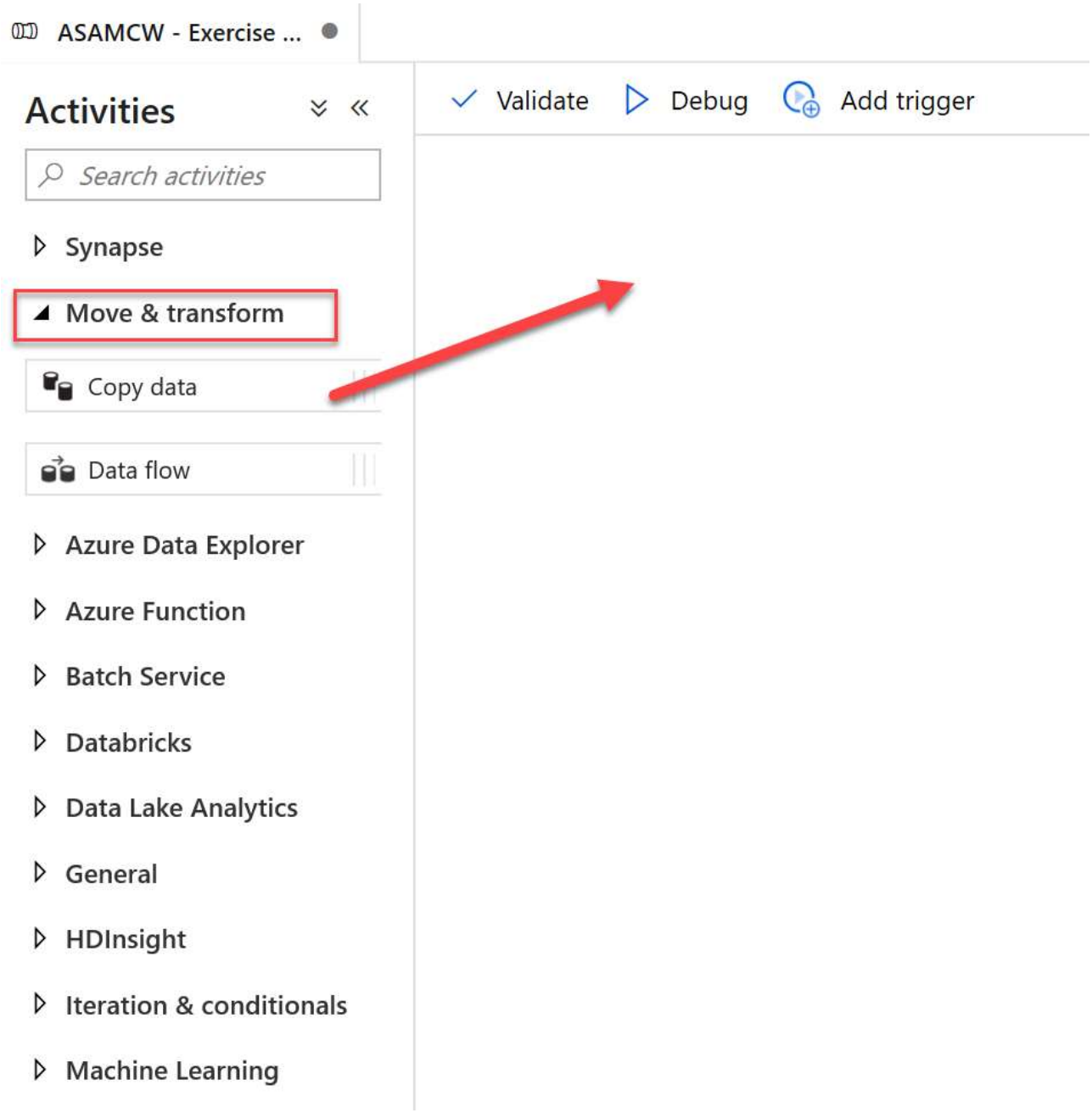
The Integrate menu item is selected from the left menu. On the Integrate blade, the + button is expanded with the Pipeline item highlighted.

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



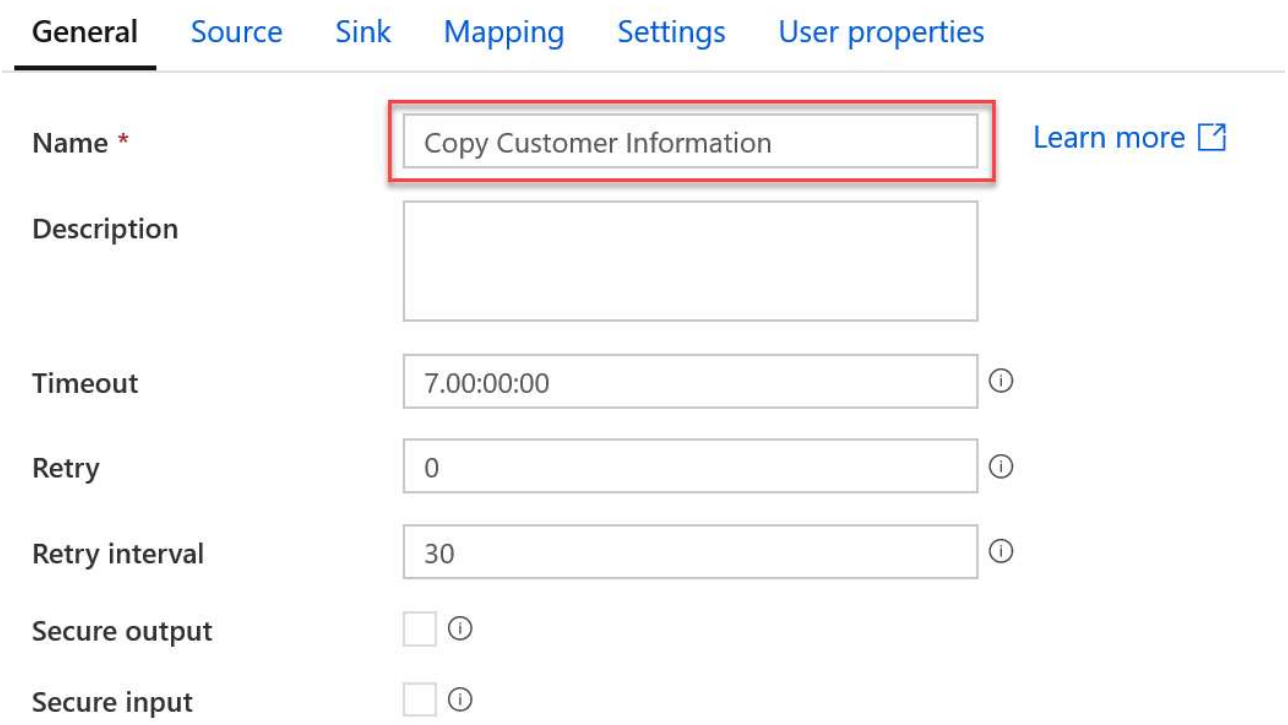
The General tab is shown with the name field populated as described above.

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



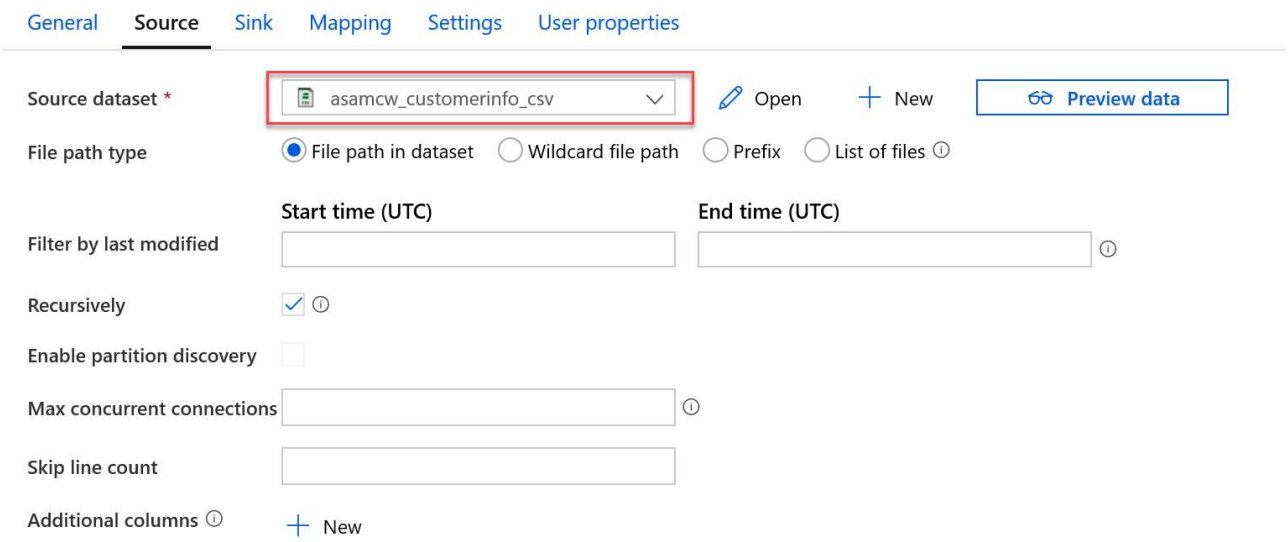
In the Activities menu, the Move and transform section is expanded. An arrow denotes an instance of the Copy data activity being dragged over to the design surface of the pipeline.

12. 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.



The General tab is selected with the Name field set to Copy Customer Information Data.

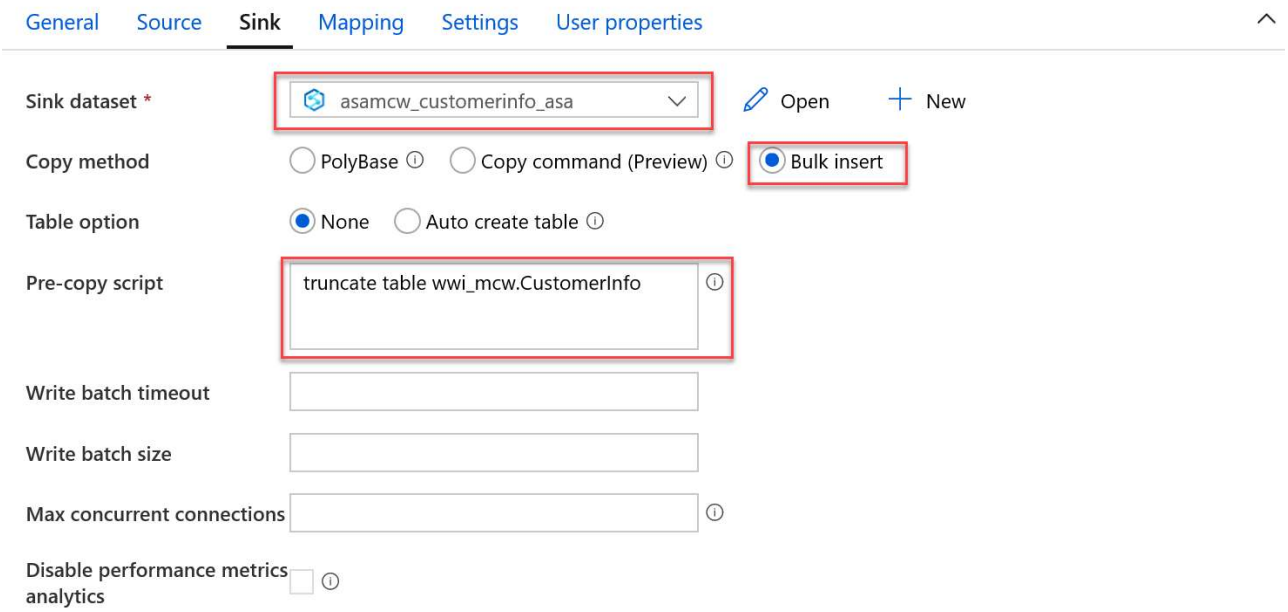
13. Select the **Source** tab in the bottom pane. In the **Source dataset** field, select **asamcw_customerinfo_csv**.



The Source tab is selected with the Source dataset field set to asamcw_customerinfo_csv.

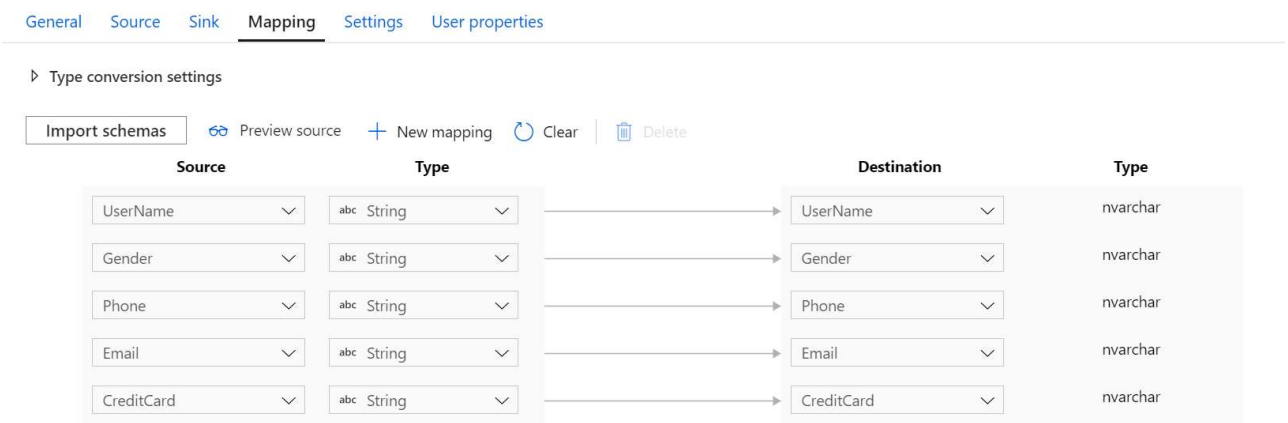
14. Select the **Sink** tab in the bottom pane. In the **Sink dataset** field, select **asamcw_customerinfo_asa**, for the **Copy method** field, select **Bulk insert**, and for **Pre-copy script** enter:

```
truncate table wwi_mcw.CustomerInfo
```



The Sink tab is selected with the Sink dataset field set to asamcw_customerinfo_asa, the Copy method set to Bulk insert, and the Pre-copy script field set to the previous query.

15. 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.



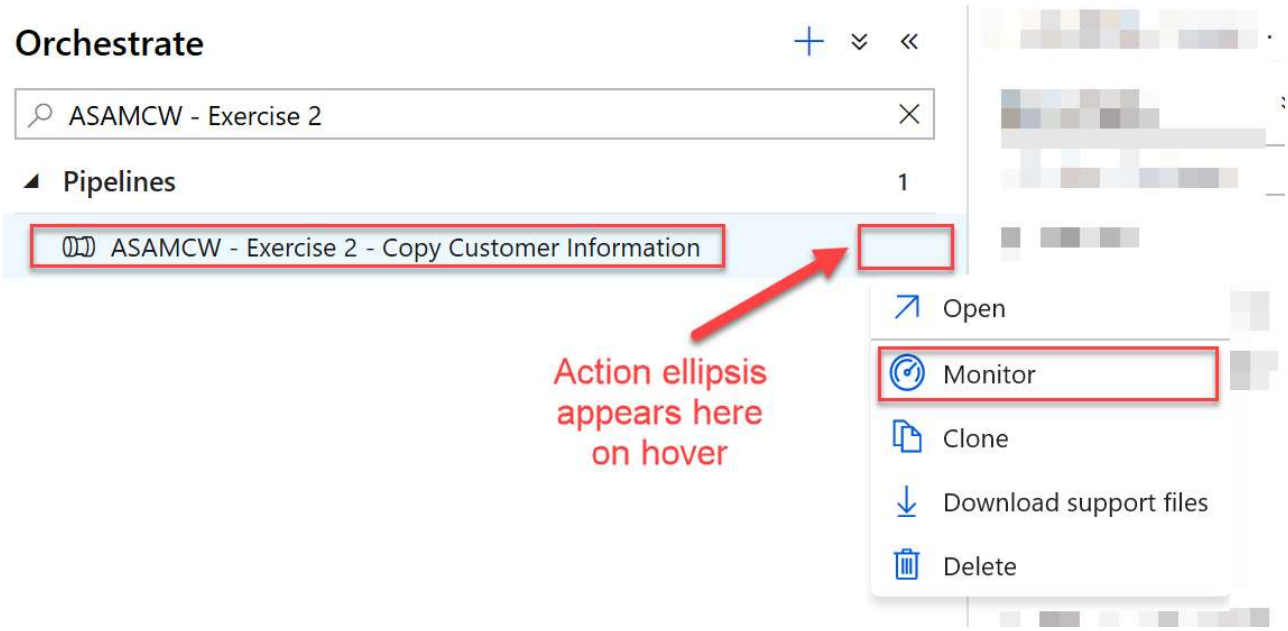
The Mapping tab is selected in the bottom pane. The source to destination field mapping is shown.

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

The top toolbar is displayed with the Publish all button highlighted.

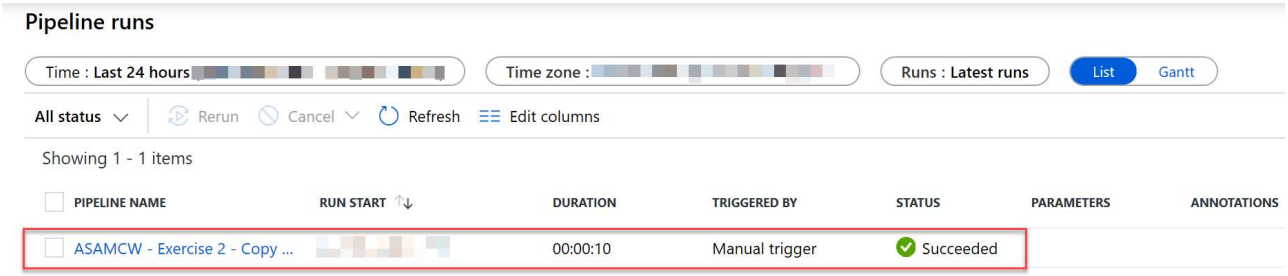
17. Once published, expand the **Add trigger** item on the pipeline designer toolbar, and select **Trigger now**. In the **Pipeline run** blade, select **OK** to proceed with the latest published configuration. You will see notification toast windows indicating the pipeline is running and when it has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

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



In the Integrate blade, the Action menu is displayed with the Monitor item selected on the ASAMCW - Exercise 2 - Copy Customer Information pipeline.

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



On the pipeline runs screen, a successful pipeline run is highlighted in the table.

20. 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

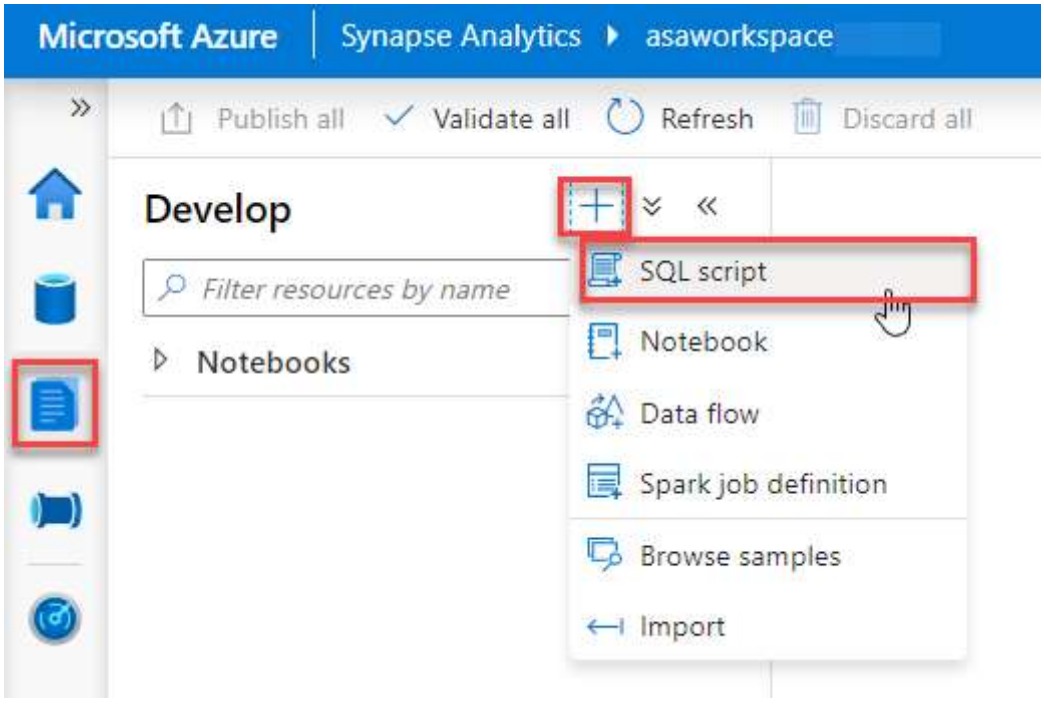
(`SQLPool01`), then paste and run the following query. When complete, select the **Discard all** button from the top toolbar.

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

Task 5: Create the campaign analytics table

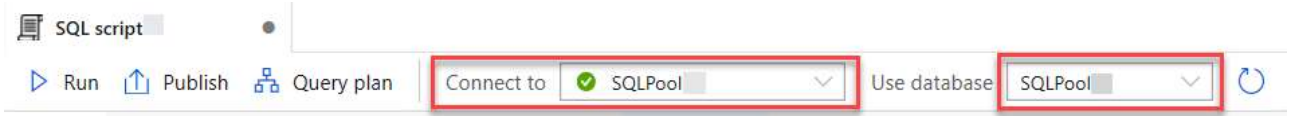
The campaign analytics table will be queried primarily for dashboard and KPI purposes. Performance is a large factor in the design of this table, and as such we can ascertain that we will need a **Clustered Columnstore** table with a **Hash** table distribution based on the **Region** field which will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



The left menu is expanded with the Develop item selected. The Develop blade has the + button expanded with the SQL script item highlighted.

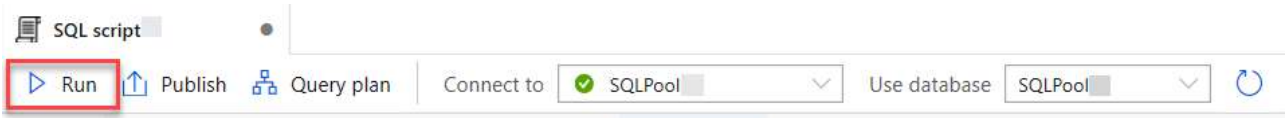
2. In the query tab toolbar menu, ensure you connect to your SQL Pool, `SQLPool01` .



The query tab toolbar menu is displayed with the Connect to set to the SQL Pool.

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
);
```



The query tab toolbar is displayed with the Run button selected.

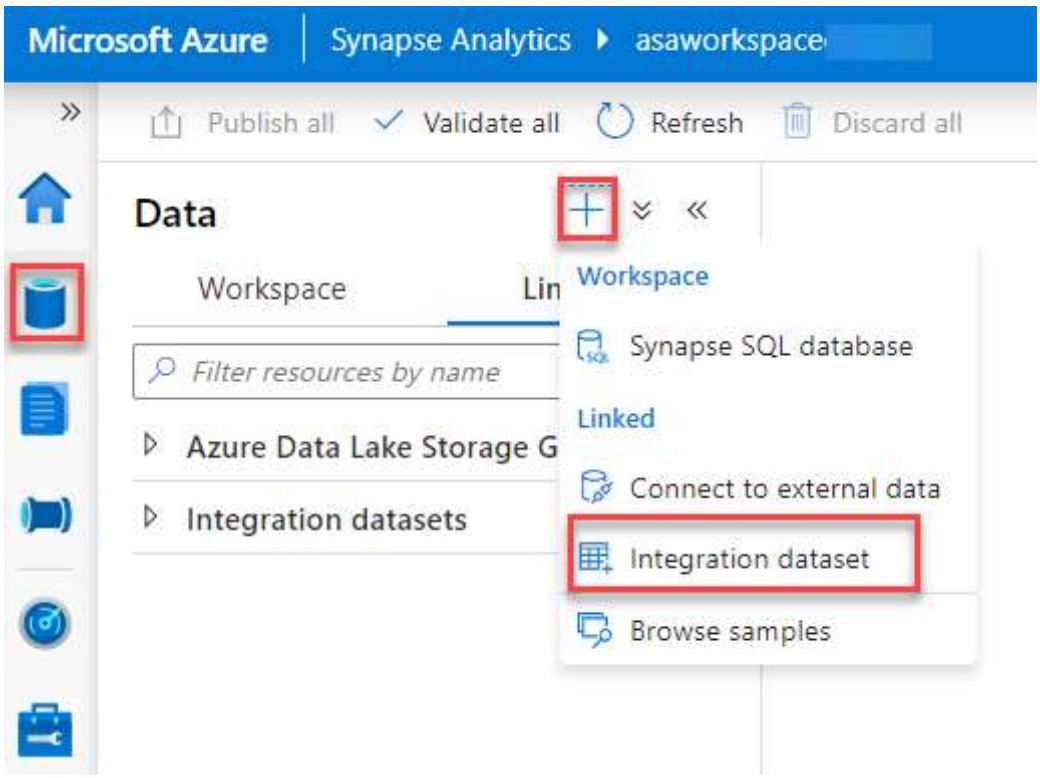
4. At the far right of the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

The top toolbar menu is displayed with the Discard all button highlighted.

Task 6: Populate the campaign analytics table

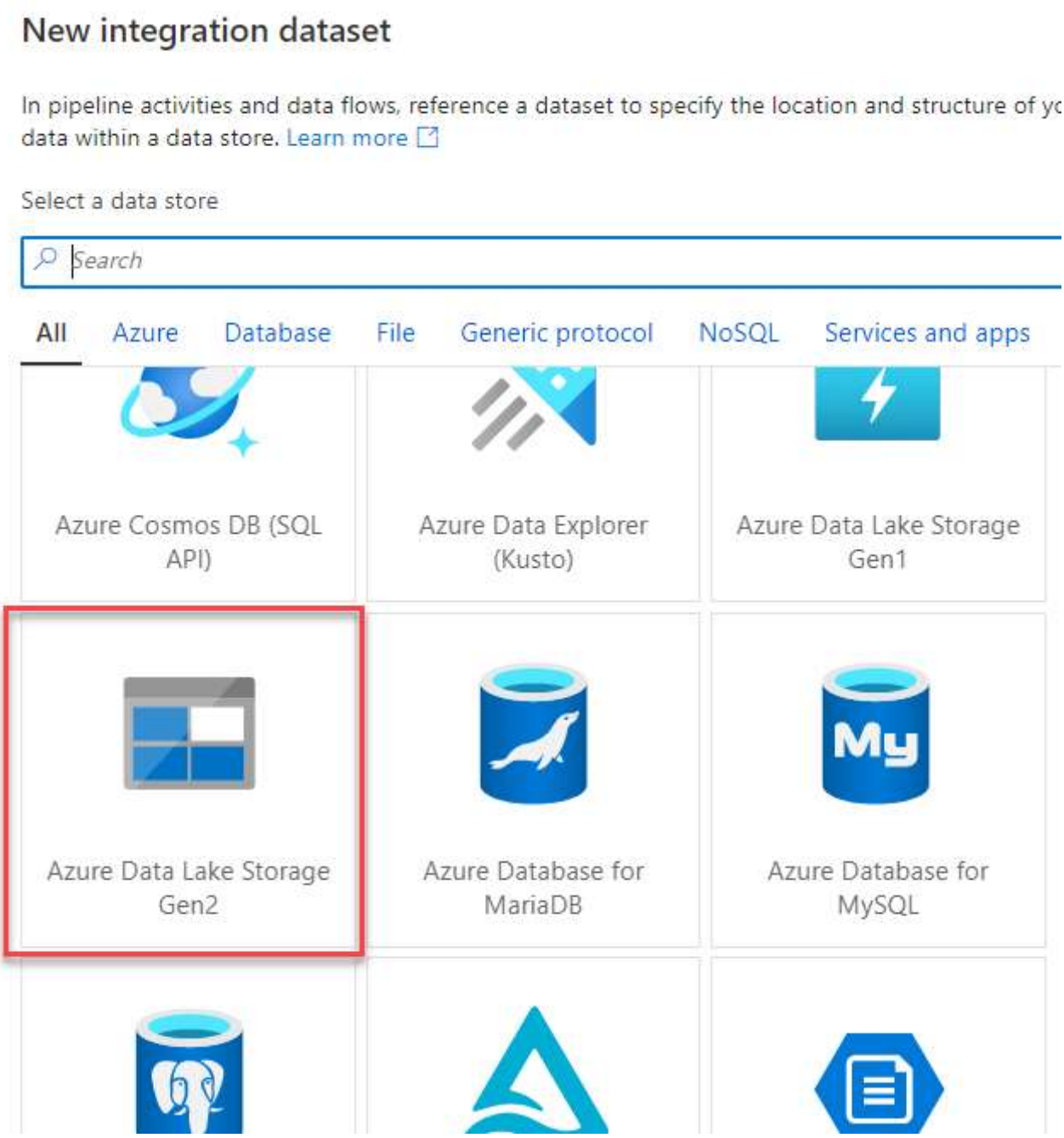
Similar to the customer information table, we will also be populating the campaign analytics table via a CSV file located in the data lake. This will require source and sink datasets to point to the CSV file in storage and the campaign analytics table that you just created in the SQL Pool. The source 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 **Integration dataset**.



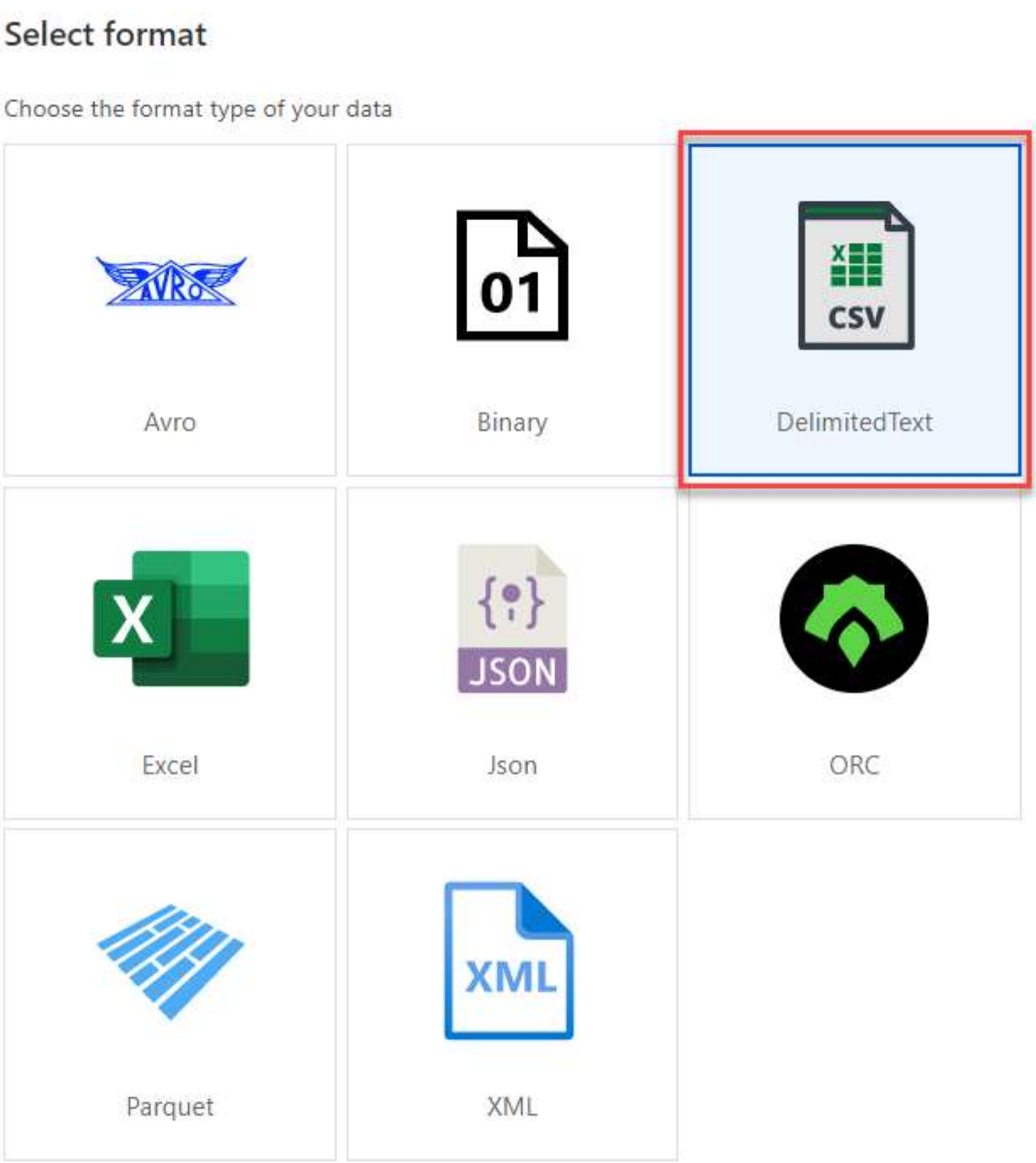
The Data item is selected from the left menu. On the Data blade, the + button is expanded with the Dataset item highlighted.

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



The New dataset blade is displayed with the All tab selected, the Azure Data Lake Storage Gen2 item is selected from the list.

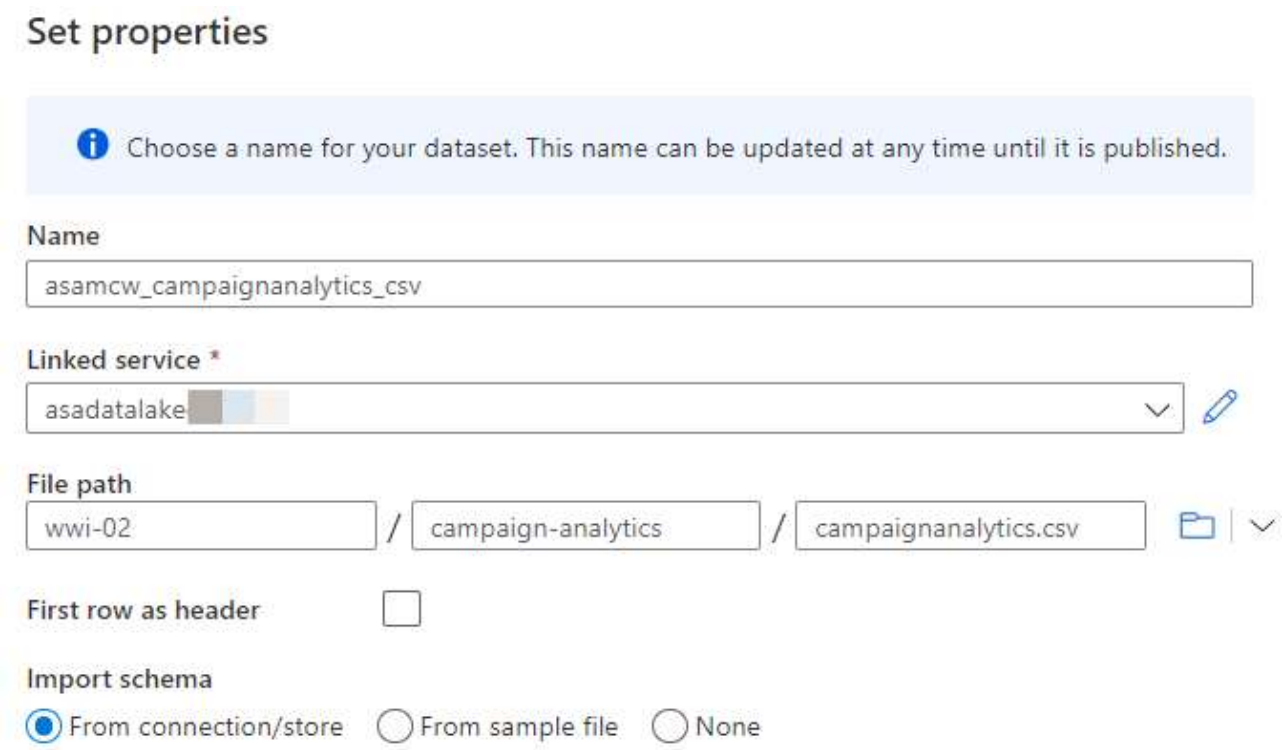
3. On the **Select format** blade, select **CSV Delimited Text.** Select **Continue.**



On the Select format blade the CSV Delimited Text item is highlighted.

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**



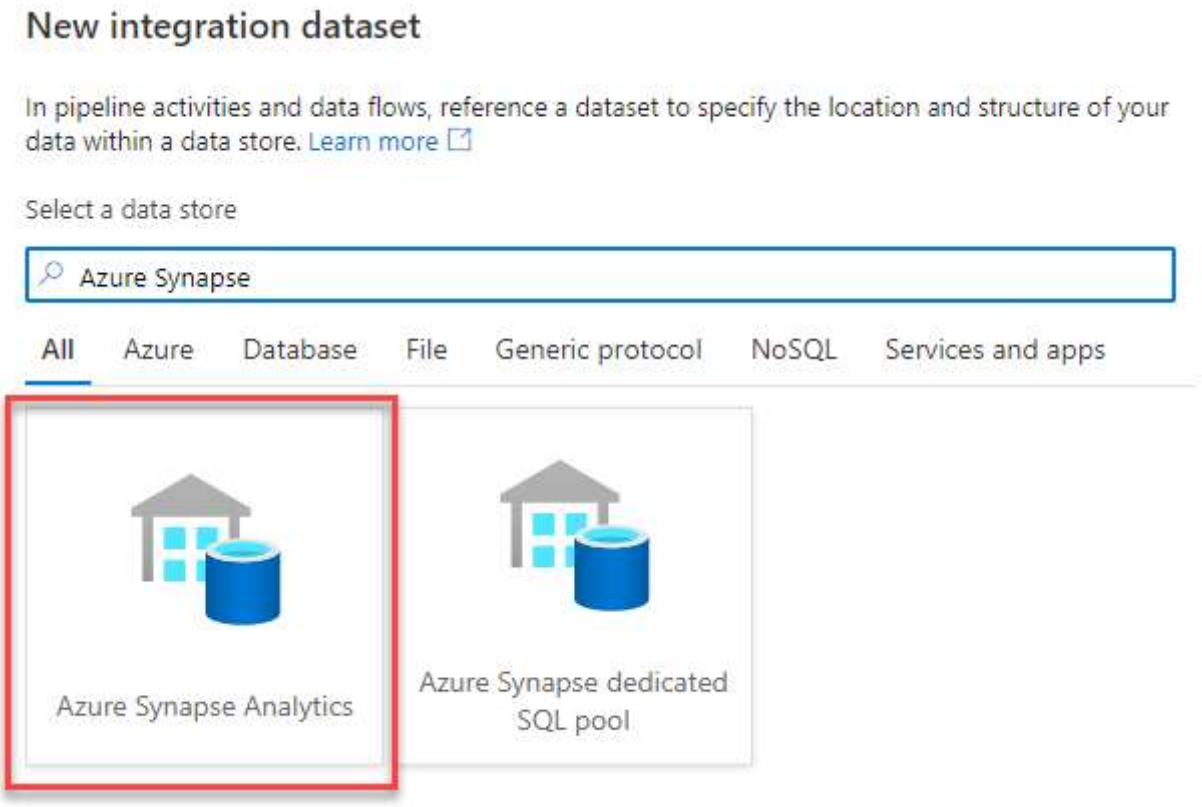
The Set properties form is displayed with the values specified in the previous table.

5. 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 **Integration dataset**.

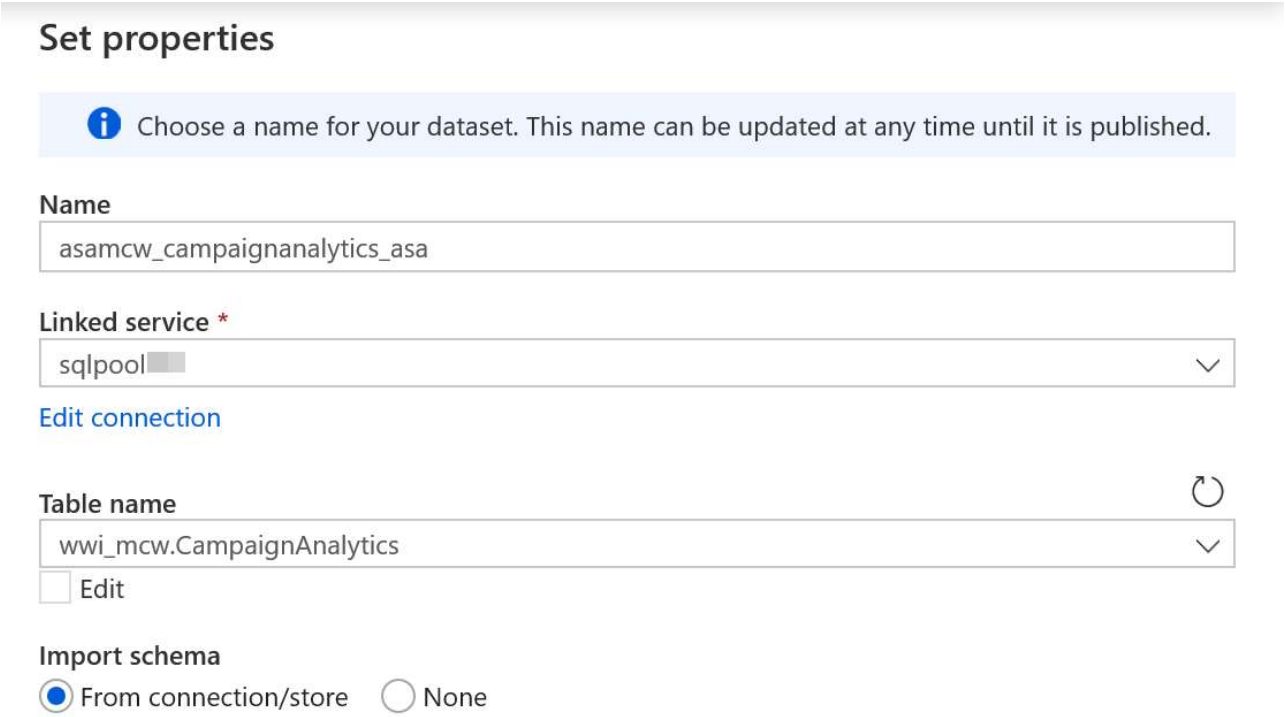
7. On the **New integration dataset** blade, enter **Azure Synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue**.



The New integration dataset form is shown with Azure Synapse entered in the search box and the Azure Synapse Analytics item highlighted.

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**.

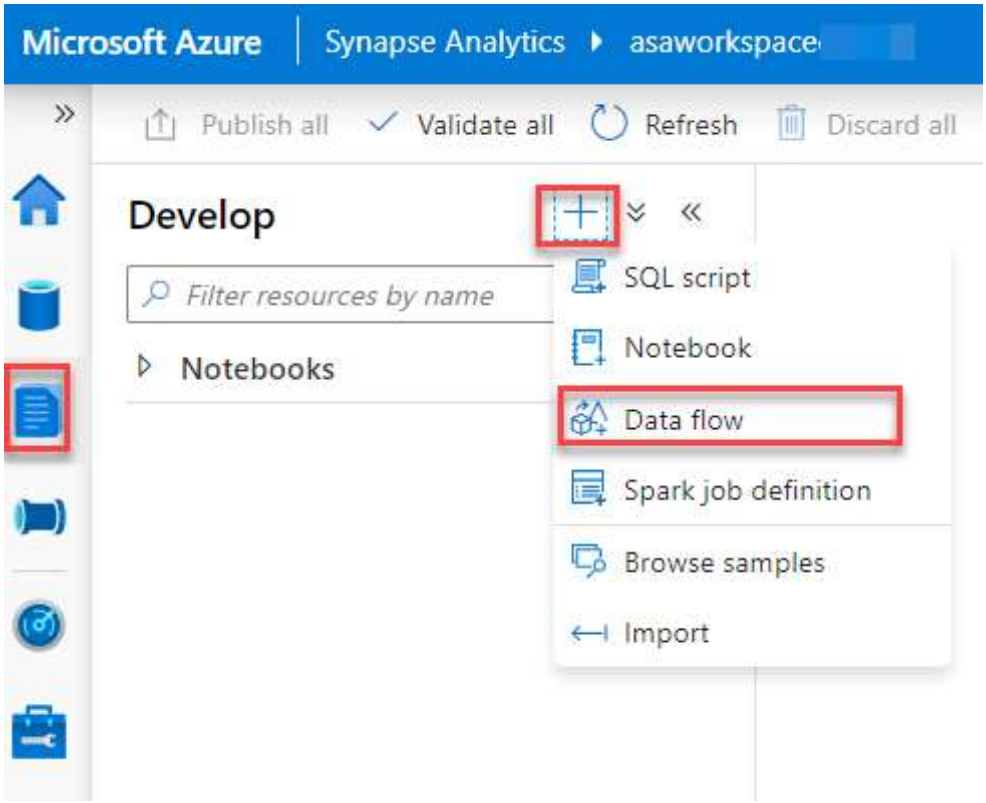


The Set properties blade is populated with the values specified in the preceding table.

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

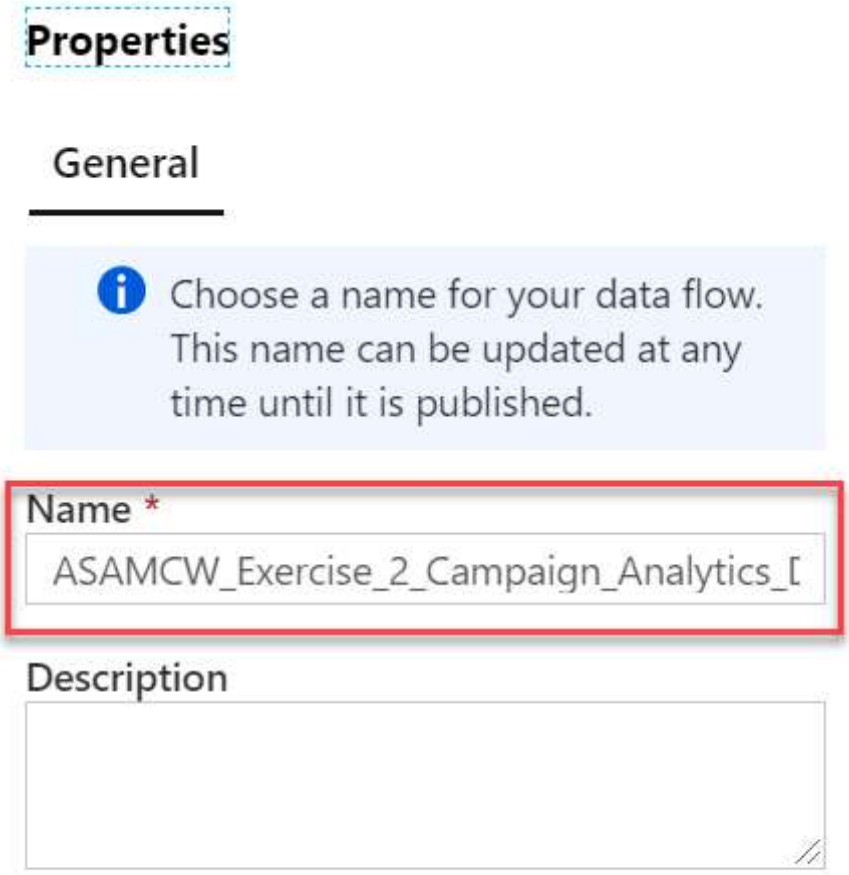
The top toolbar is displayed with the Publish all button highlighted.

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**.



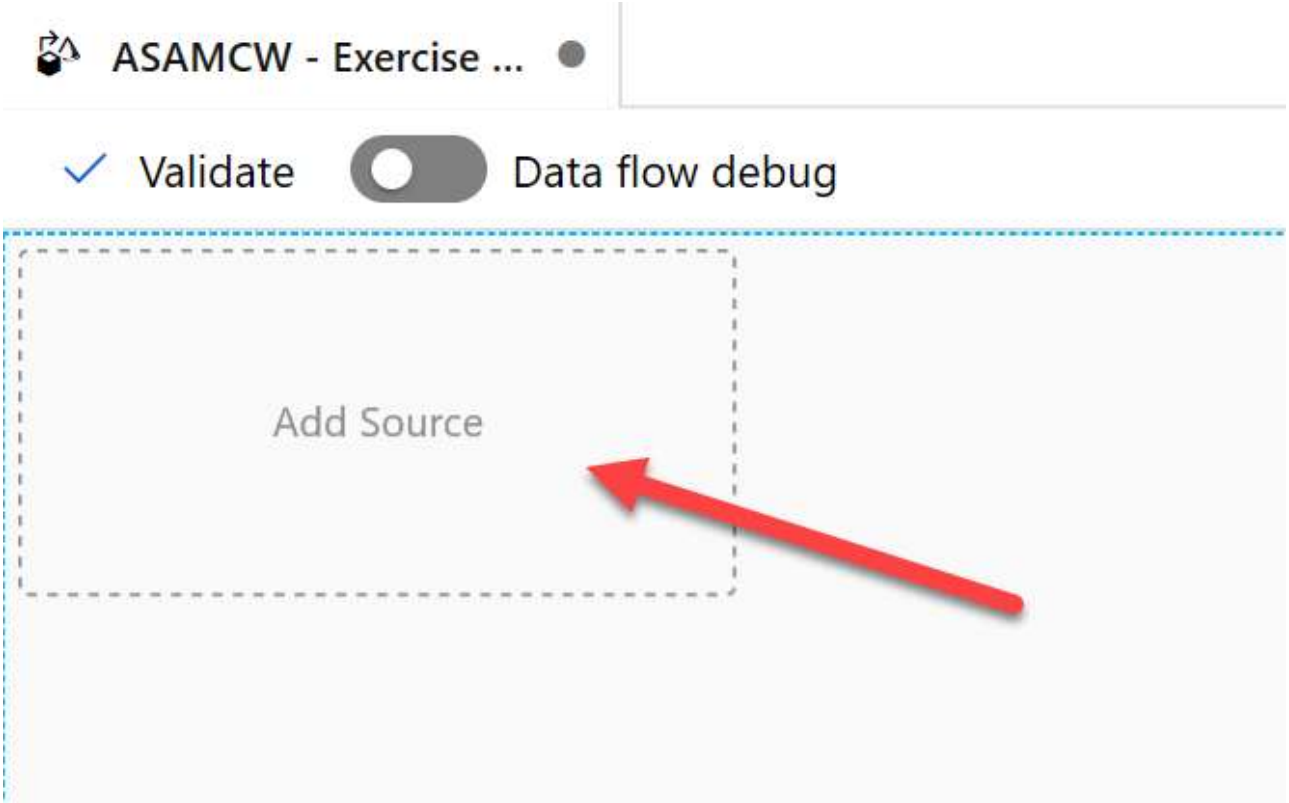
From the left menu, the Develop item is selected. From the Develop blade the + button is expanded with the Data flow item highlighted.

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



The Properties blade is displayed with ASAMCW_Exercise_2_Campaign_Analytics_Data entered as the name of the data flow.

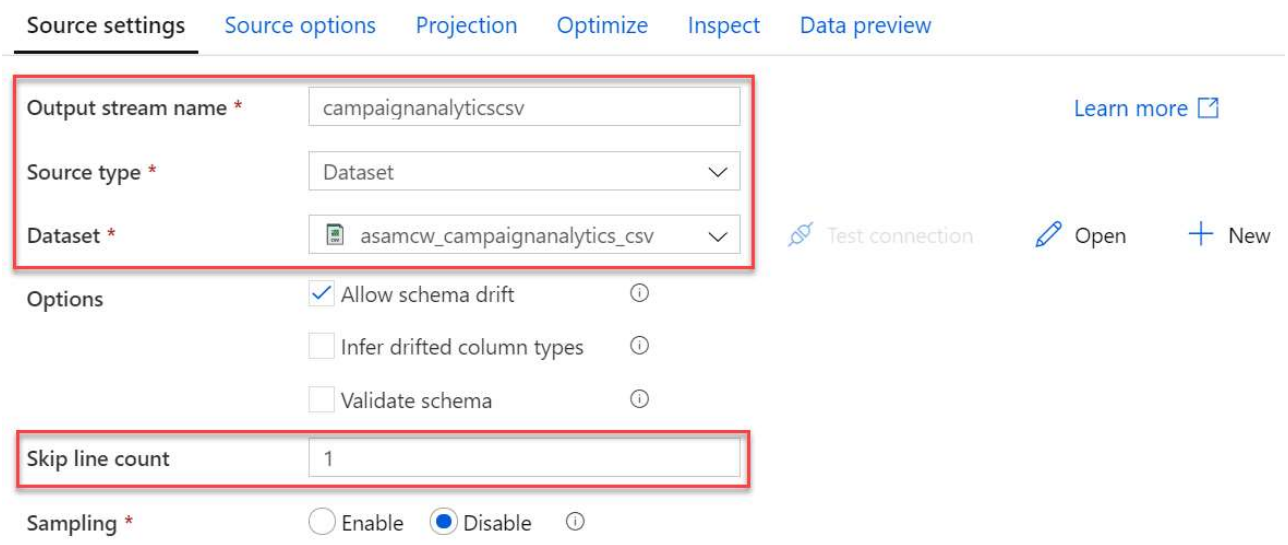
12. In the data flow designer window, select the **Add Source** box.



The Add source box is highlighted in the data flow designer window.

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.**



The Source settings tab is displayed with a form populated with the values defined in the preceding table.

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.

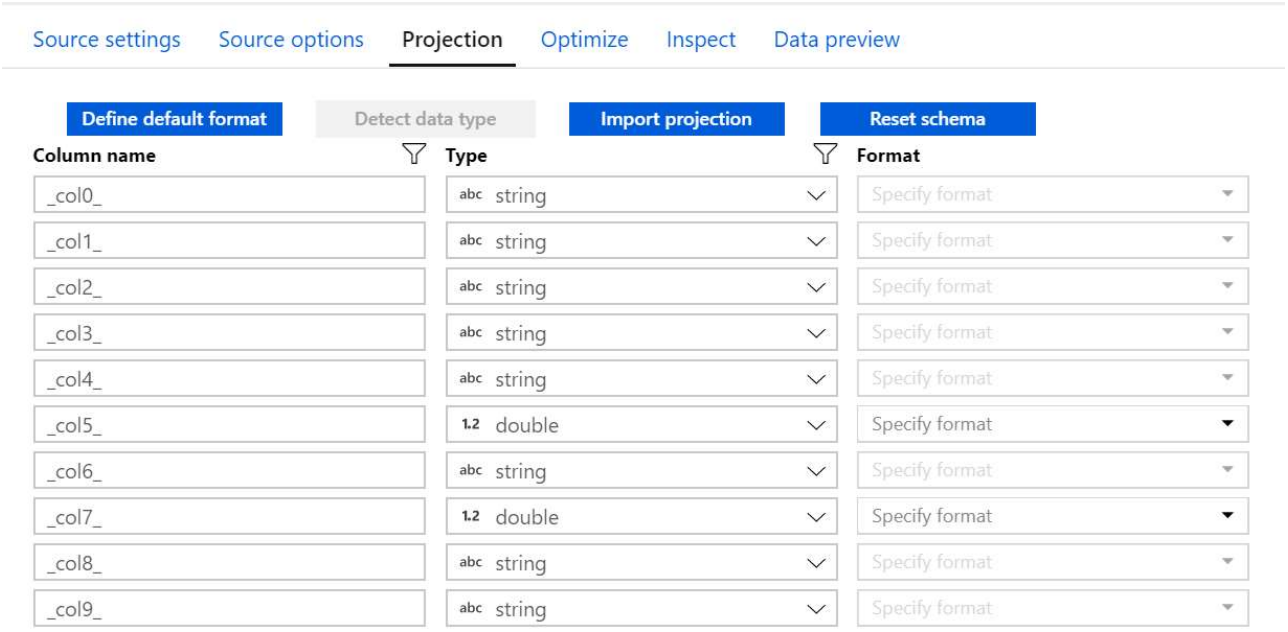
A portion of the dataflow designer toolbar is shown with the Script icon highlighted.

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
```

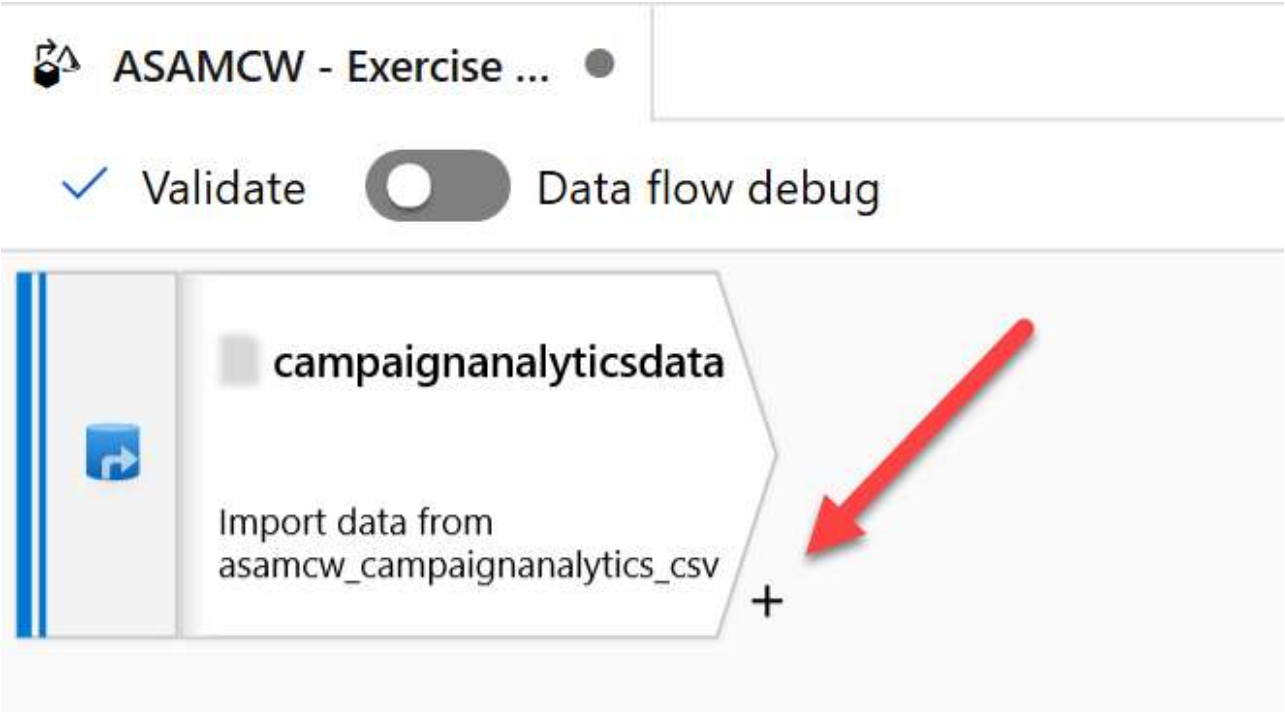
> ***Note**: We are changing the mappings as the source file was corrupted with the wrong headers.*

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



The Projection tab is displayed with columns defined as described in the column mapping script.

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



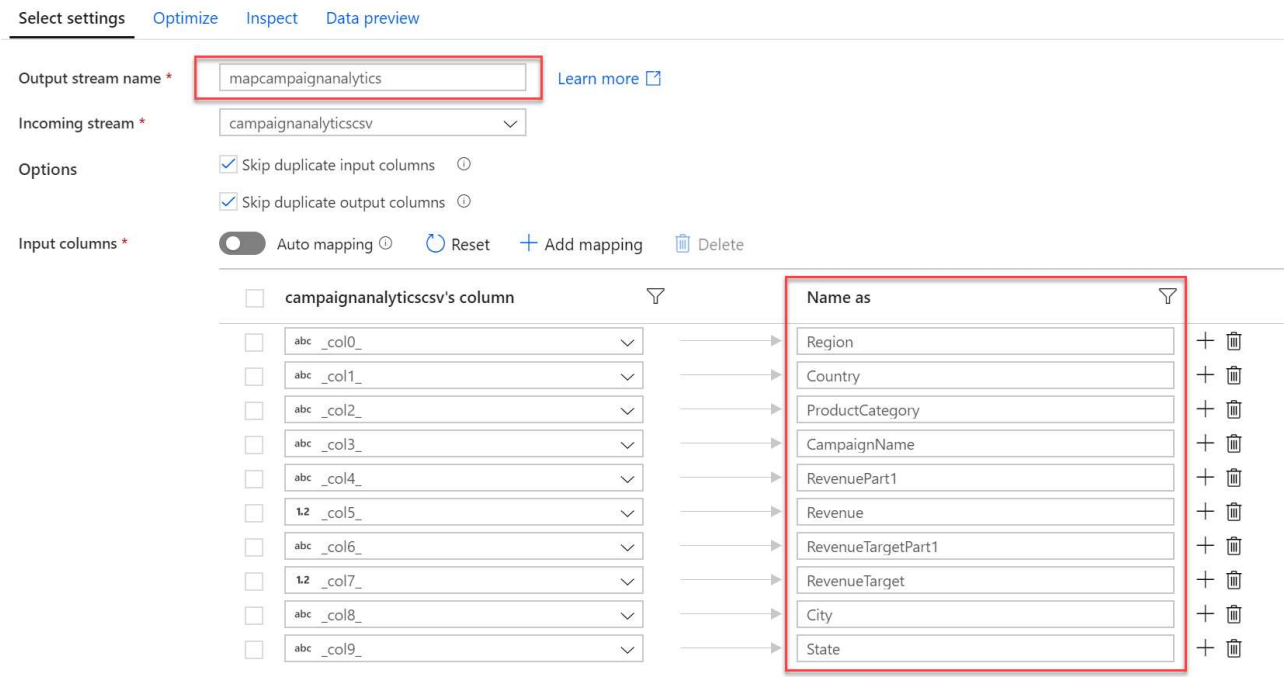
The + button on the bottom right of the campaignanalyticscsv source is highlighted.

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

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

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



The Select settings tab is displayed with the form filled as described in the preceding table.

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

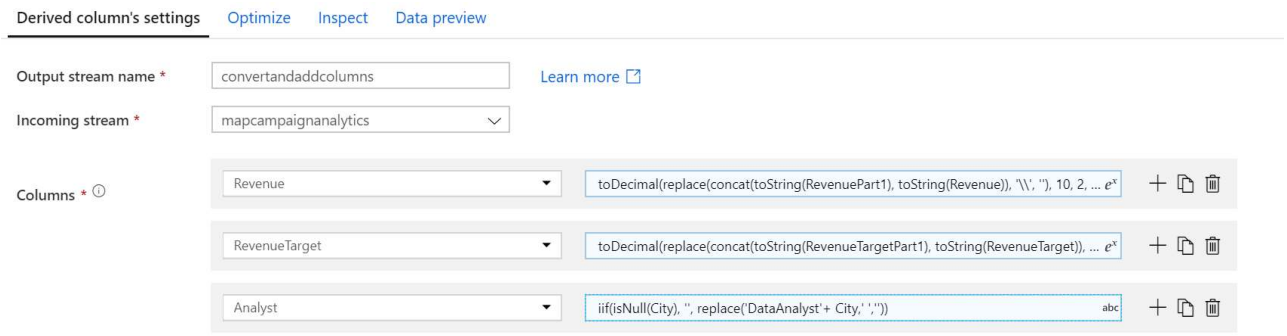
20. Under **Derived column's settings**, configure the following:

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

For **Columns**, add the following (Note: you will need to type in the **Analyst** column and use the **open expression builder** link to enter the expression values):

Column	Expression	Descri
Revenue	**toDecimal(replace(concat(toString(RevenuePart1), toString(Revenue)), '\\', ''), 10, 2,'$###,###.##')** ** \| Concatenate the **RevenuePart1** and **Revenue** fields, replace the invalid `\` character, then convert and format the data to a decimal type. \| \| RevenueTarget \| **toDecimal(replace(concat(toString(RevenueTargetPart1), toString(RevenueTarget)), '\\\\', ''), 10, 2, '$###,###.##')**	Concat **Reven** and **Re** fields, invalid then c format decima

Column	Expression	Descri
Analyst	**iif(isNull(City), ’‘, replace(’DataAnalyst’+ City,’ ‘,”))**	If the c assign string t field, o concat DataAı City va all spa

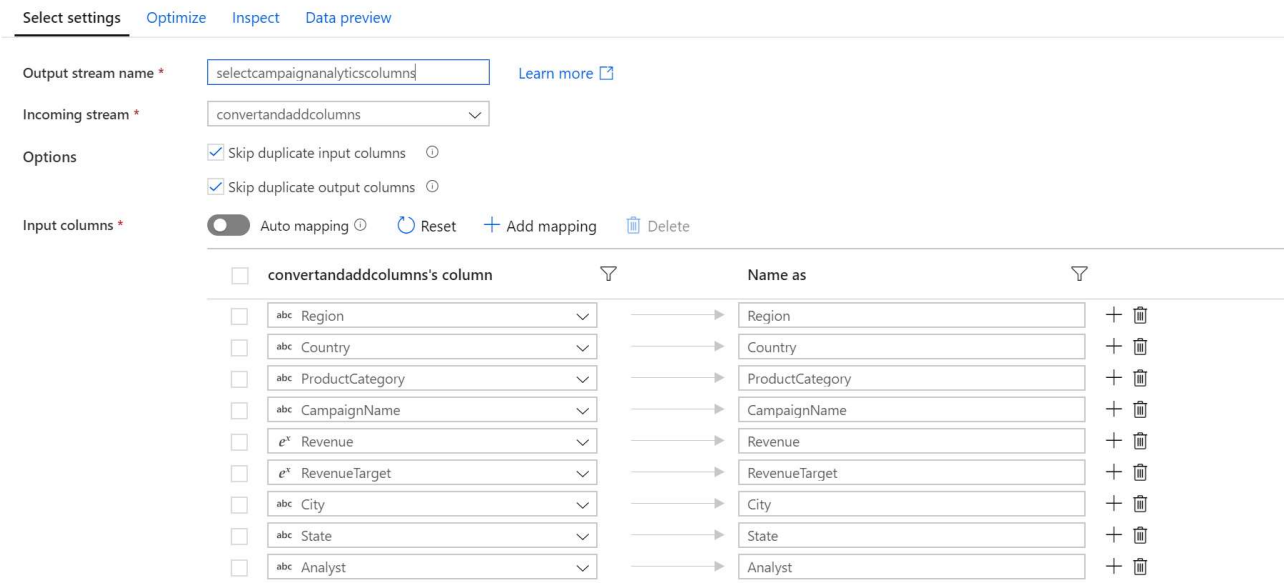


The derived column’s settings are displayed as described.

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

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

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

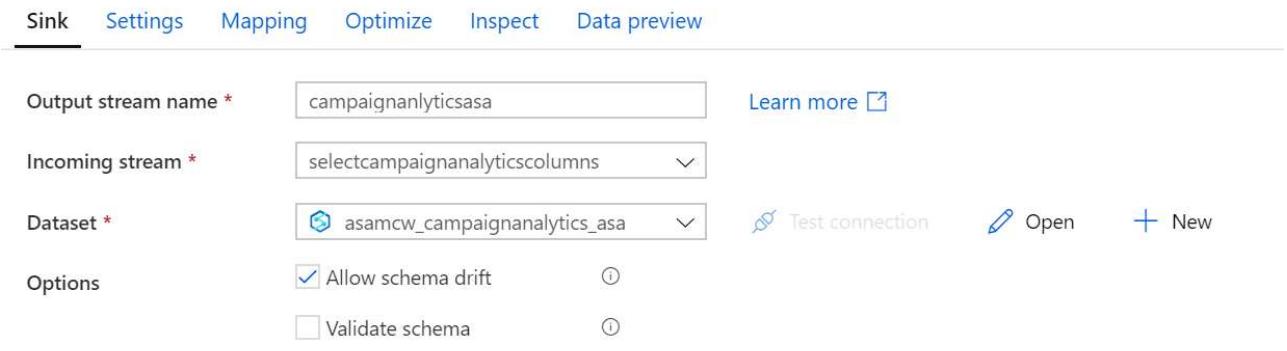


The Select settings are displayed showing the updated column mappings.

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

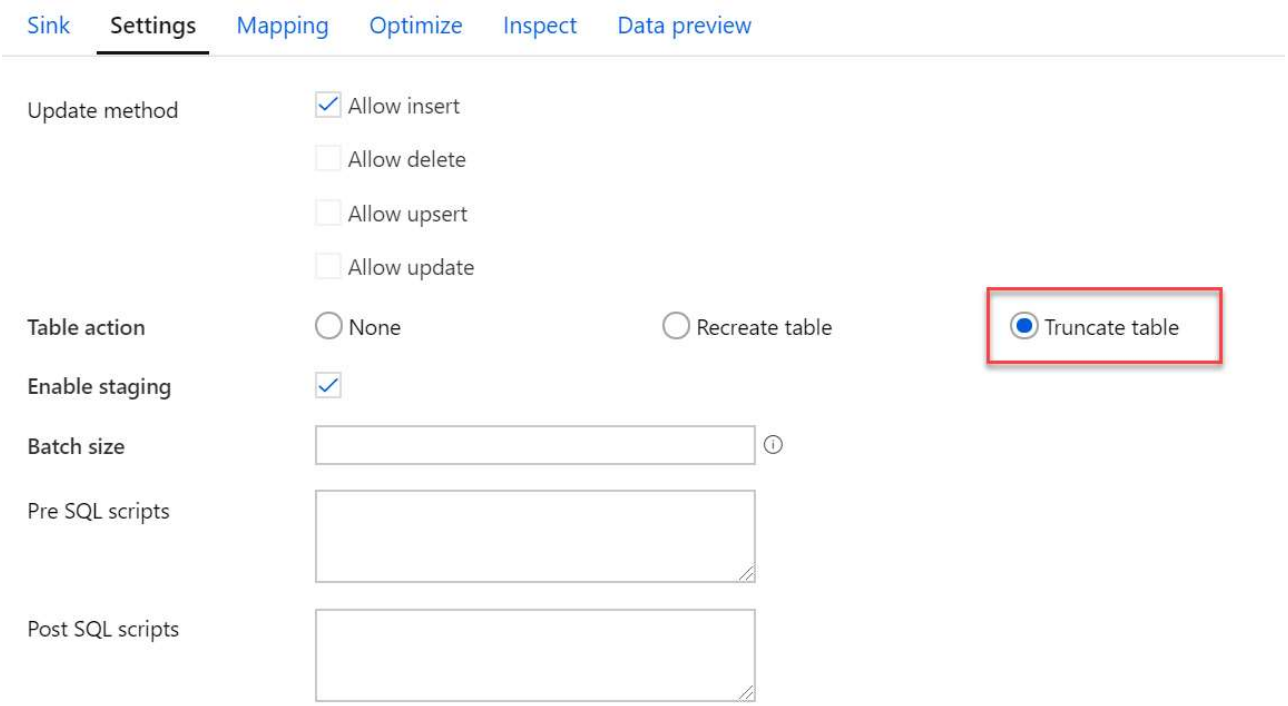
24. In the bottom pane, on the **Sink** tab, configure it as follows:

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



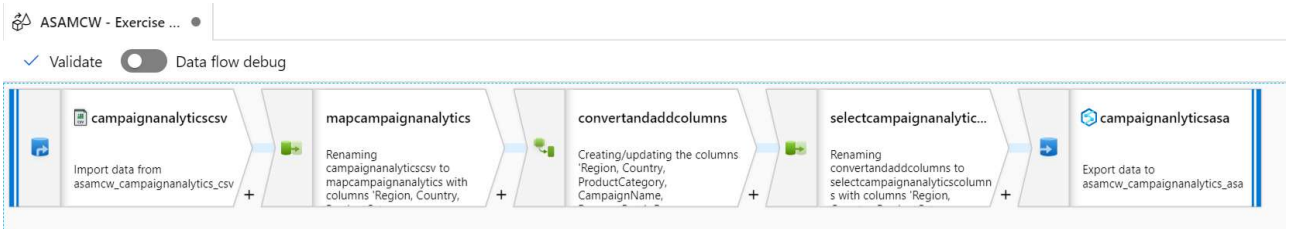
The Sink settings form is displayed populated with the values defined in the previous table.

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



The sink Settings tab is displayed with the Table action set to Truncate table.

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



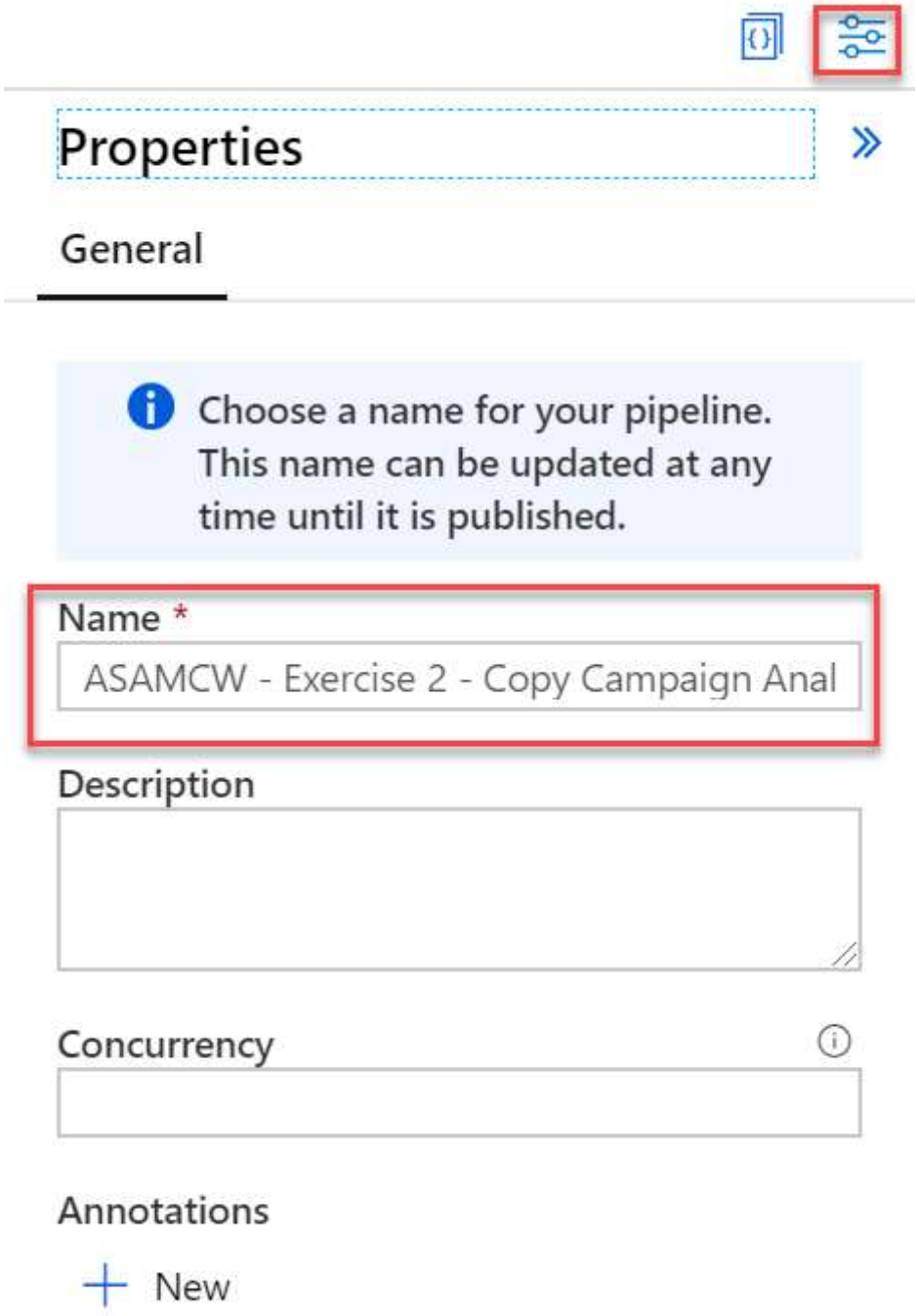
The completed data flow is displayed.

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

Publish all is highlighted.

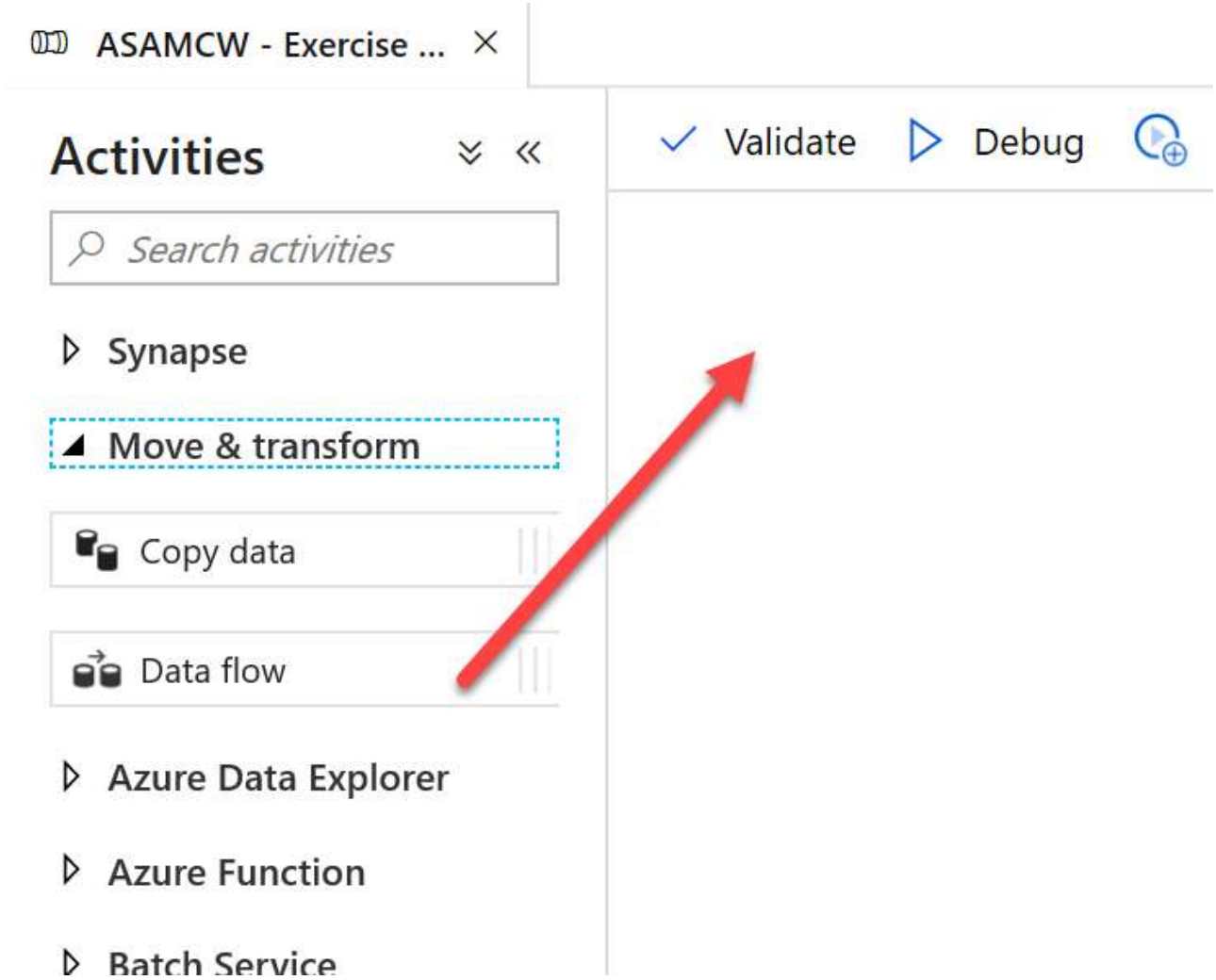
28. Now that the data flow is published, we can use it in a pipeline. Create a new pipeline by selecting **Integrate** from the left menu, then in the **Integrate** blade, expand the **+** button and select **Pipeline**.

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



The pipeline properties blade is displayed with the Name field populated with ASAMCW - Exercise 2 - Copy Campaign Analytics Data.

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

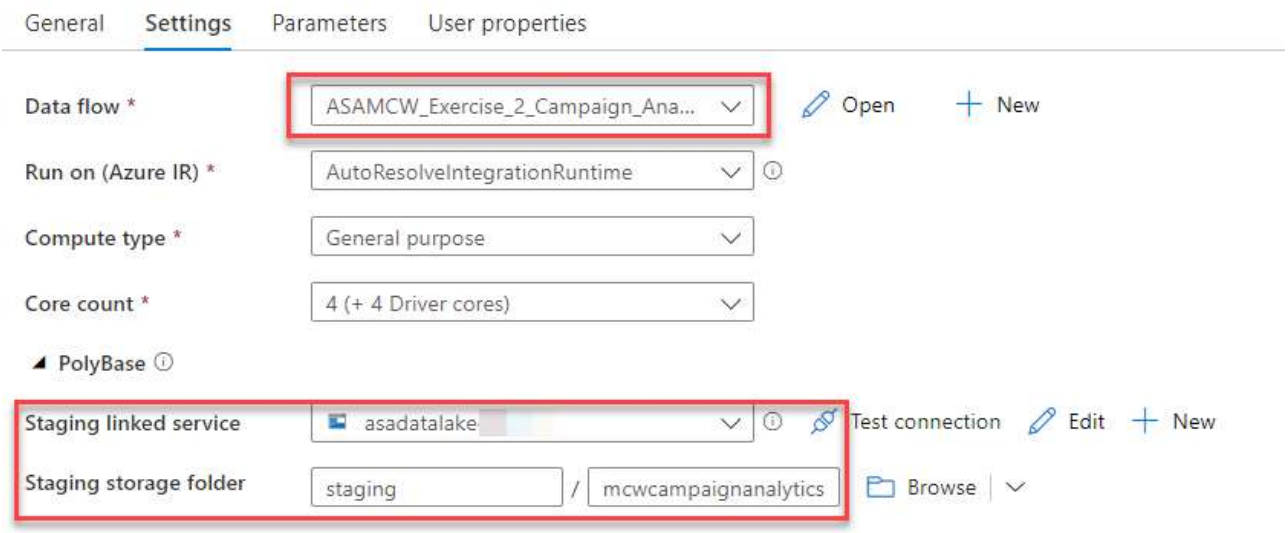


The Activities menu of the pipeline is displayed with the Move and transform section expanded. An arrow indicating a drag operation shows adding a Data flow activity to the design surface of the pipeline.

31. 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.

32. 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.**
Staging storage folder - Directory	Enter **mcwcampaignanalytics.**



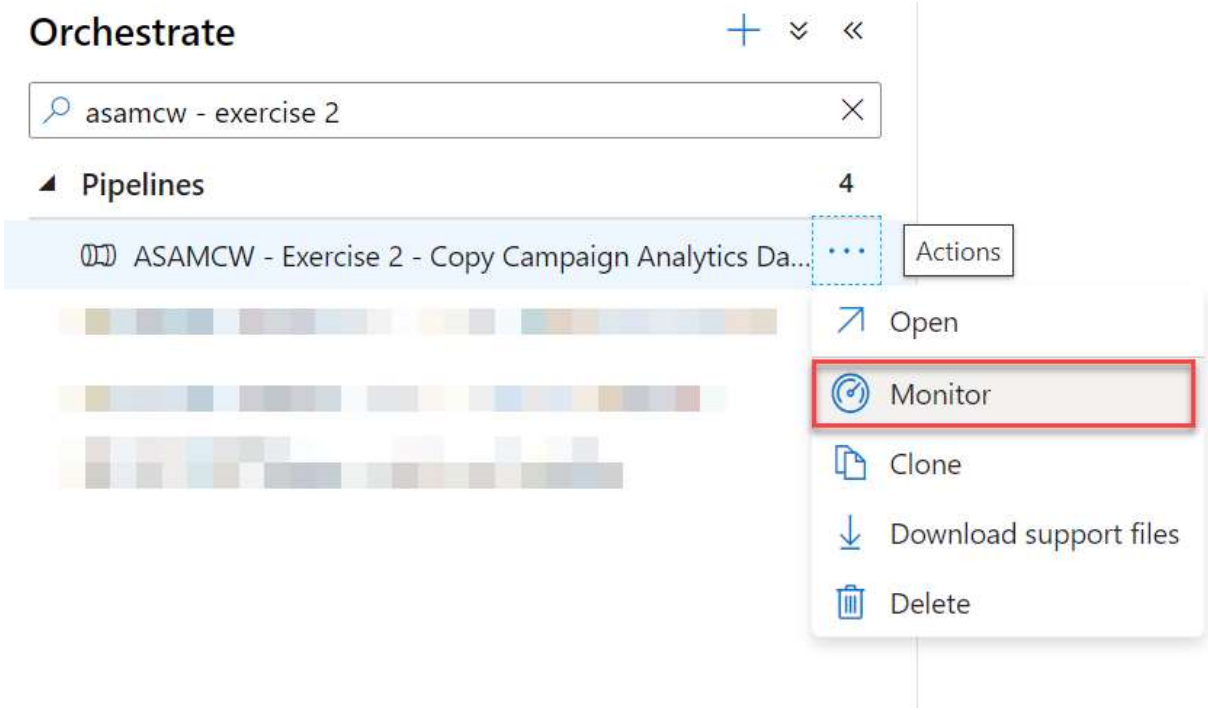
The data flow activity Settings tab is displayed with the fields specified in the preceding table highlighted.

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

The top toolbar is displayed with the Publish all button highlighted.

34. 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.

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



In the Integrate blade, the Action menu is displayed with the Monitor item selected on the ASAMCW - Exercise 2 - Copy Campaign Analytics Data pipeline.

36. 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**.

37. 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 **Integrate**. From the **Integrate** 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 Integrate 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.

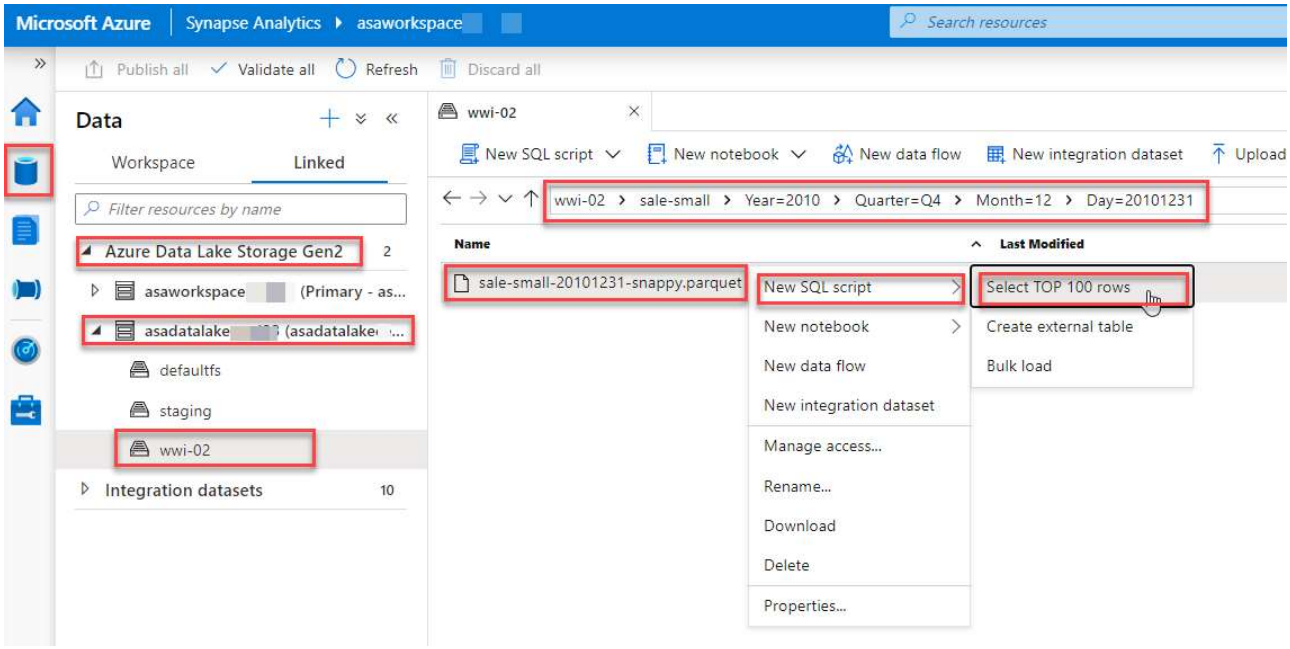
In Azure Synapse Analytics, you have the possibility of using either the Synapse SQL Serverless engine, the big-data Spark engine, or both.

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

Task 1: Query sales Parquet data with Synapse SQL Serverless

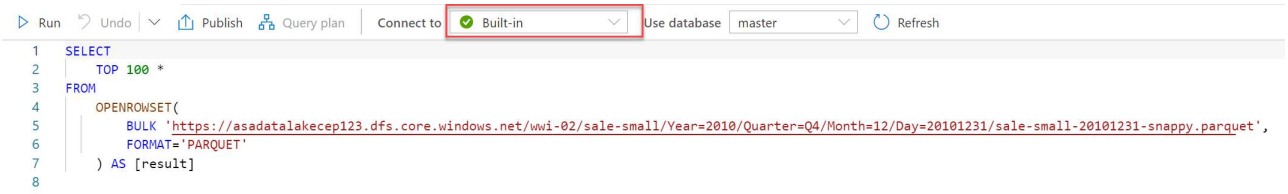
When you query Parquet files using Synapse SQL Serverless, you can explore the data with T-SQL syntax.

1. From the left menu, select **Data**.
2. From the **Data** blade, select the **Linked** tab.
3. Expand **Azure Data Lake Storage Gen2**. 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**.



The Storage accounts section is expanded with the context menu visible on the asadatalake{SUFFIX} account with the Select TOP 100 rows option highlighted.

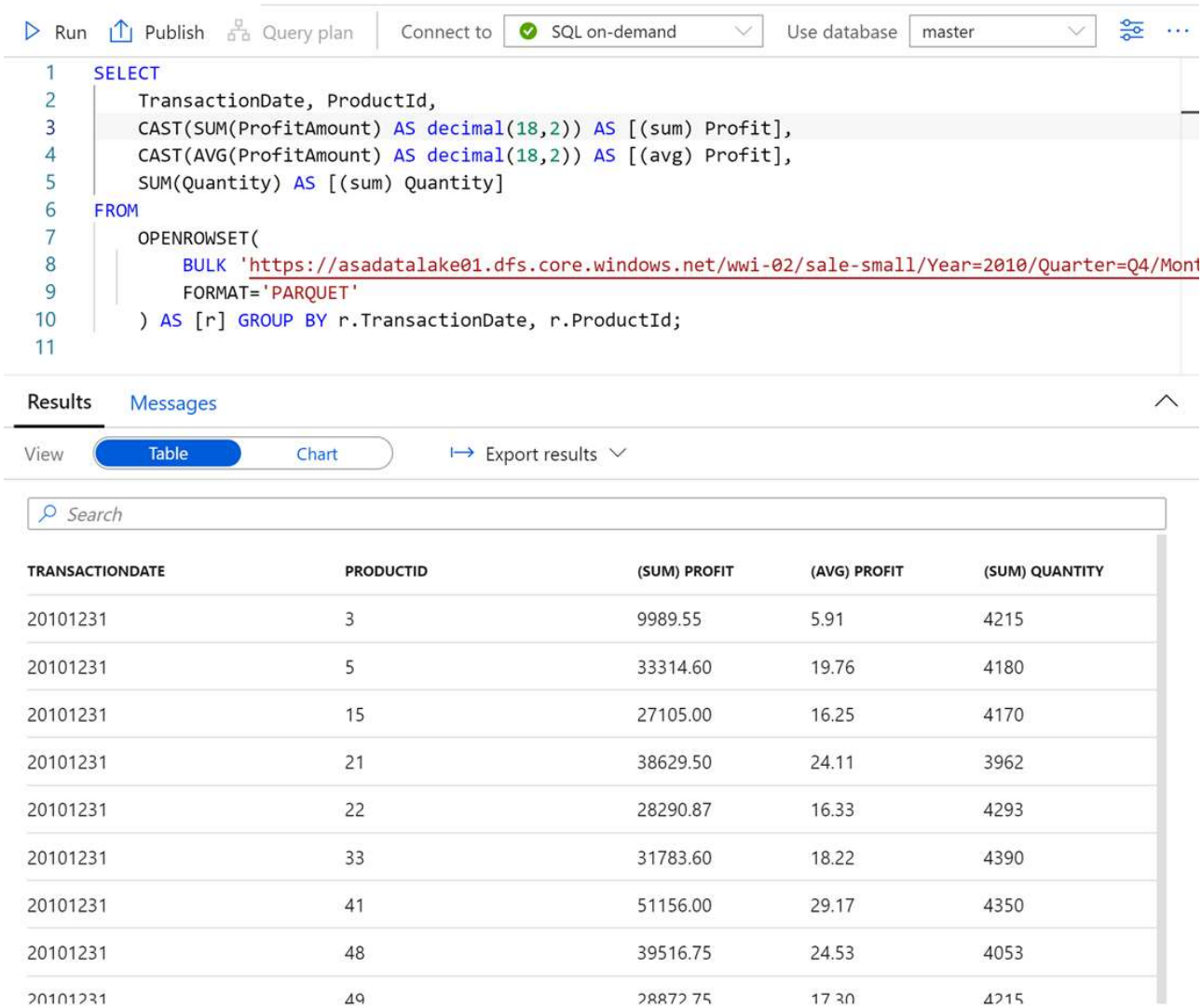
5. Ensure the **Built-in** Synapse SQL Serverless pool is selected in the **Connect to** dropdown list above the query window, then run the query. Data is loaded by the Synapse SQL Serverless endpoint and processed as if was coming from any regular relational database.



The Built-in SQL on-demand connection is highlighted on the query window toolbar.

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;
```



The T-SQL query above is displayed within the query window.

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}`):

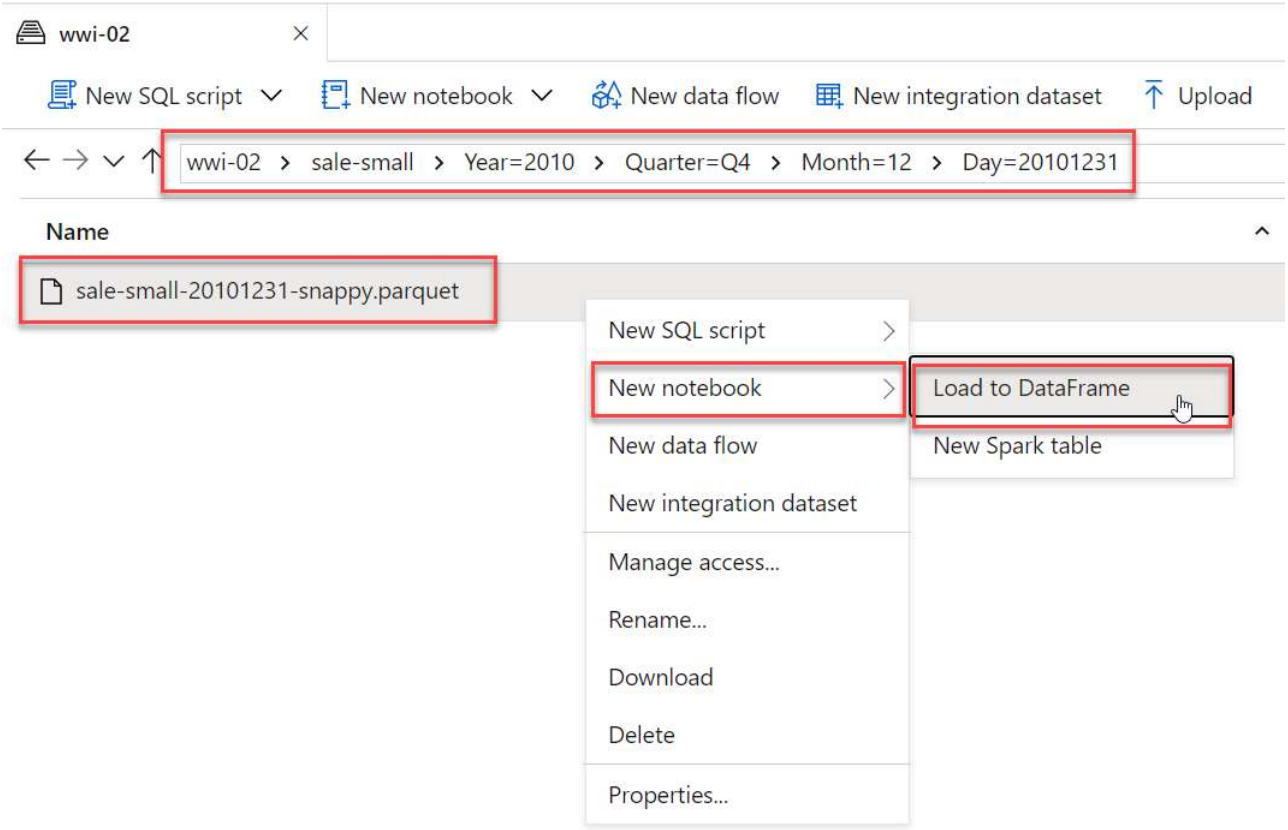
```
SELECT
    COUNT_BIG(*)
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/sale-small/Year=2019/*/*/*/*',
        FORMAT='PARQUET'
    ) AS [r];
```

> *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

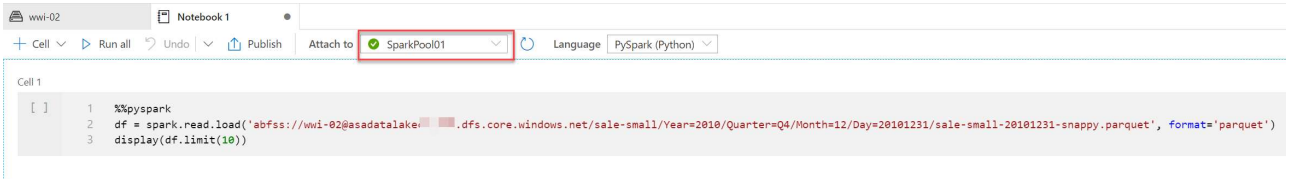
1. Select **Data** from the left menu, select the **Linked** tab, then browse to the data lake storage account `asadatalake{SUFFIX}` to **wwi-02/sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231**, then right-click the Parquet file and select **New notebook** then **Load to DataFrame**.



The Parquet file is displayed with the New notebook and Load to DataFrame menu items highlighted.

2. This will generate a notebook with PySpark code to load the data in a dataframe and display 100 rows with the header.

3. Attach the notebook to a Spark pool.



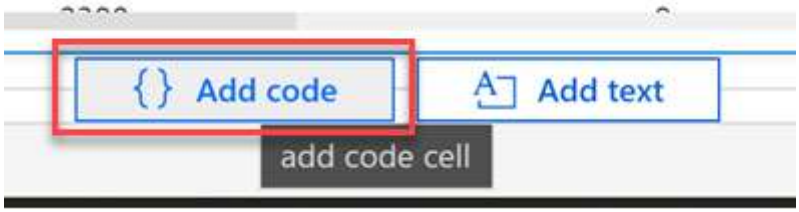
The Spark pool list is displayed.

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.



The Add Code menu option is highlighted.

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

```
df.printSchema()
```

Your output should look like the following:

```
root
    |-- TransactionId: string (nullable = true)
    |-- CustomerId: integer (nullable = true)
    |-- ProductId: short (nullable = true)
    |-- Quantity: short (nullable = true)
    |-- Price: decimal(29,2) (nullable = true)
    |-- TotalAmount: decimal(29,2) (nullable = true)
    |-- TransactionDate: integer (nullable = true)
    |-- ProfitAmount: decimal(29,2) (nullable = true)
    |-- Hour: byte (nullable = true)
    |-- Minute: byte (nullable = true)
    |-- StoreId: short (nullable = true)
```

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

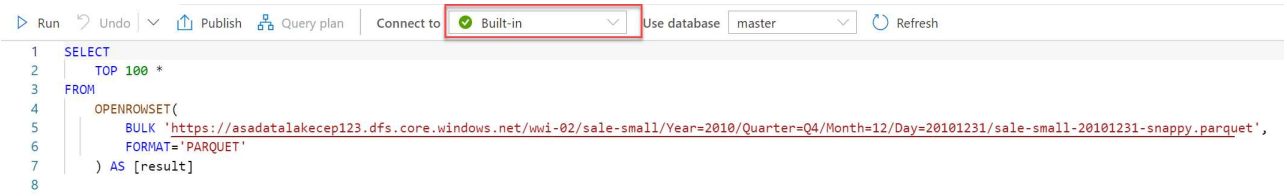
Duration: 15 minutes

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.



The Built-in SQL on-demand connection is highlighted on the query window toolbar.

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.

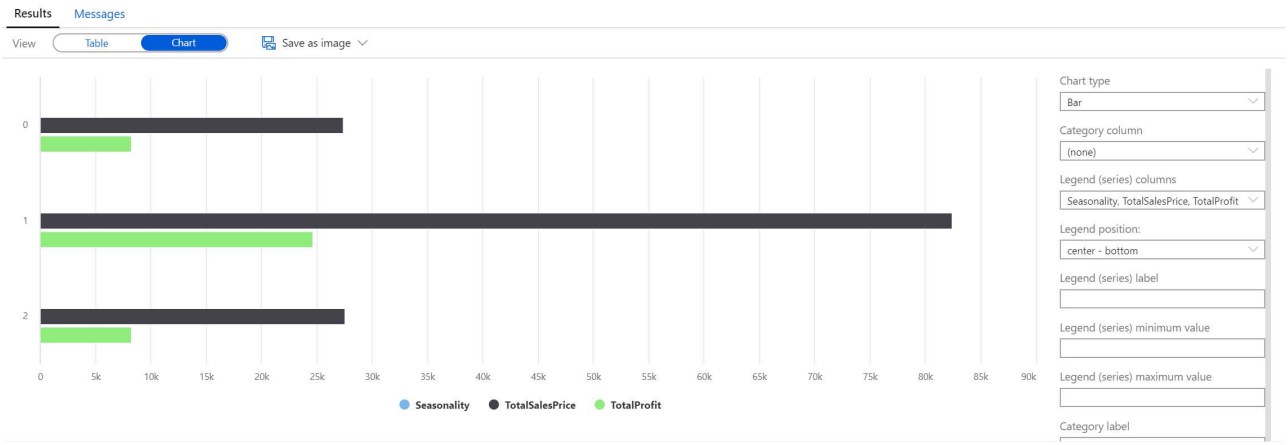
```
SELECT
   csv.*
FROM
    OPENROWSET(
        BULK 'https://asadatalake{SUFFIX}.dfs.core.windows.net/wwi-02/data-generators/generator-product/generator-product.csv',
        FORMAT='CSV',
        FIRSTROW = 1
    ) WITH (
        ProductID INT,
        Seasonality INT,
        Price DECIMAL(10,2),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



The result of the previous aggregation query is displayed as a chart in the Results pane.

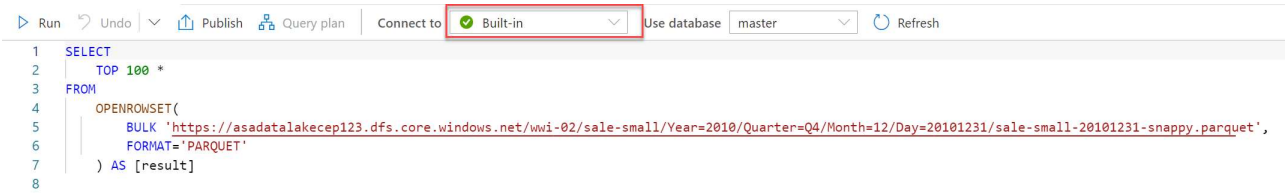
6. At the far right of the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

The top toolbar menu is displayed with the Discard all button highlighted.

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.



The Built-in SQL on-demand connection is highlighted on the query window toolbar.

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/p
roduct-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. At the far right of the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

The top toolbar menu is displayed with the Discard all button highlighted.

Exercise 5: Synapse Pipelines and Cognitive Search (Optional)

Duration: 45 minutes

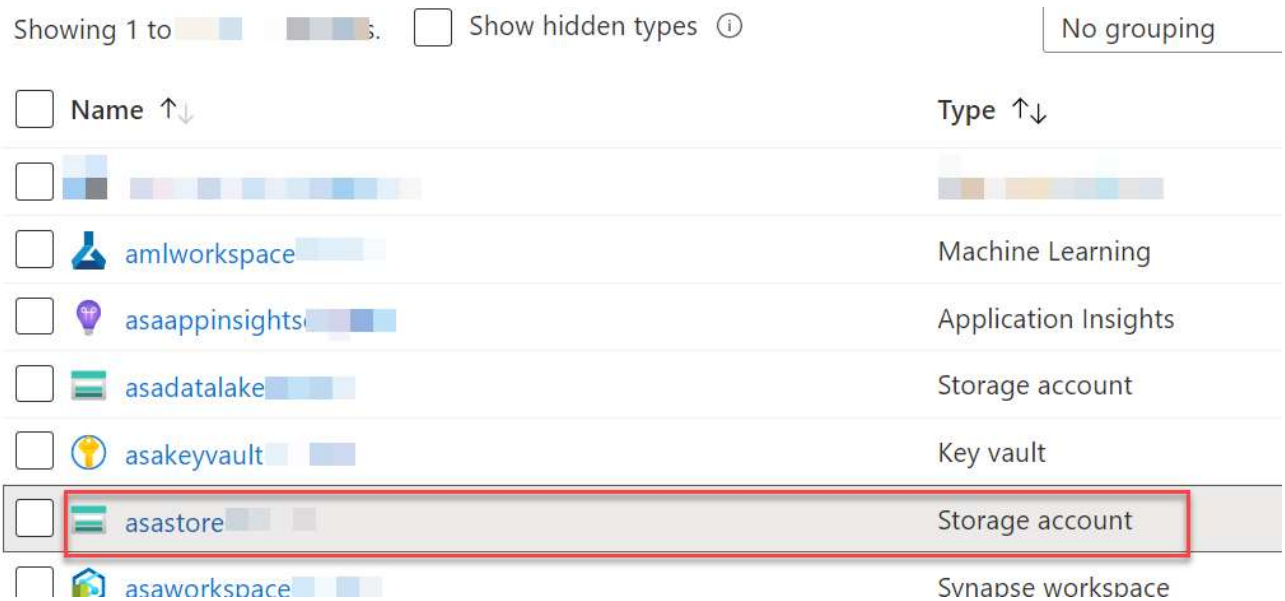
In this exercise you will create a Synapse Pipeline that will orchestrate updating the part prices from a supplier invoice. You will accomplish this by a combination of a Synapse Pipeline with an Azure Cognitive Search Skillset that invokes the Form Recognizer service as a custom skill. The pipeline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

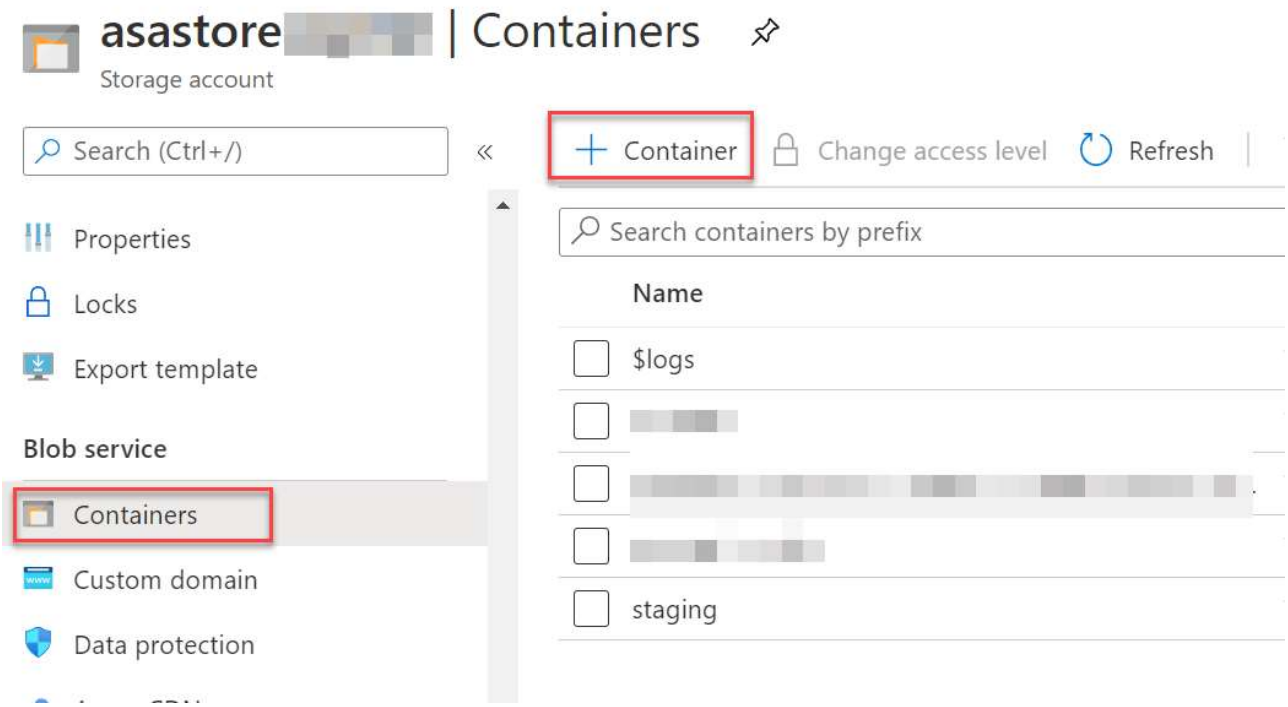
Task 1: Create the invoice storage container

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



The lab resources list is shown with the asastore storage account highlighted.

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



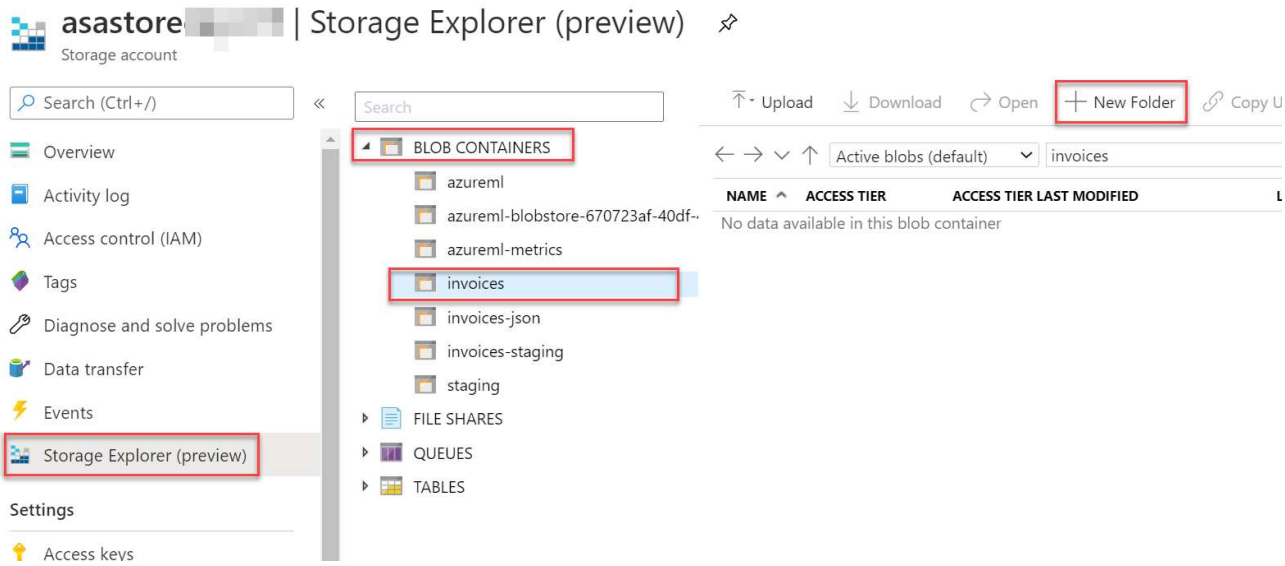
The Containers screen is displayed with Containers selected from the left menu, and + Container selected from the toolbar.

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**.

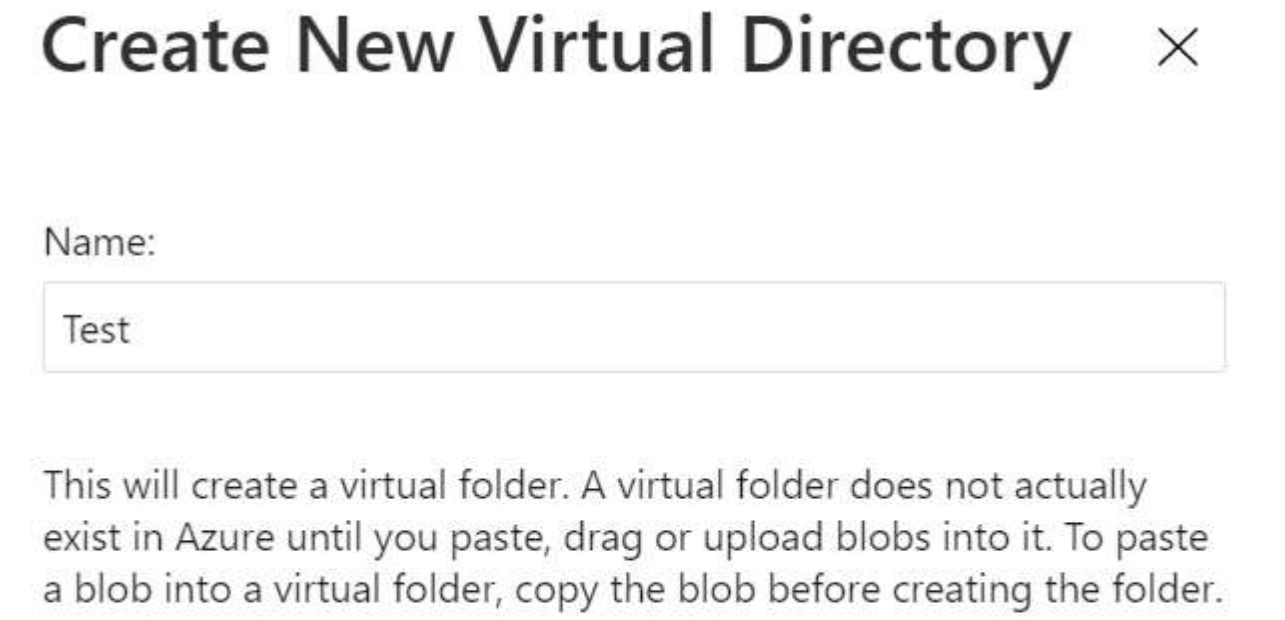
5. From the left menu, select **Storage Explorer (preview)**. Then, in the hierarchical menu, expand the **BLOB CONTAINERS** item.

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



The Storage Explorer (preview) screen is shown with Storage Explorer selected from the left menu. In the hierarchical menu, the BLOB CONTAINERS item expanded with the invoices item selected. The + New Folder button is highlighted in the taskbar menu.

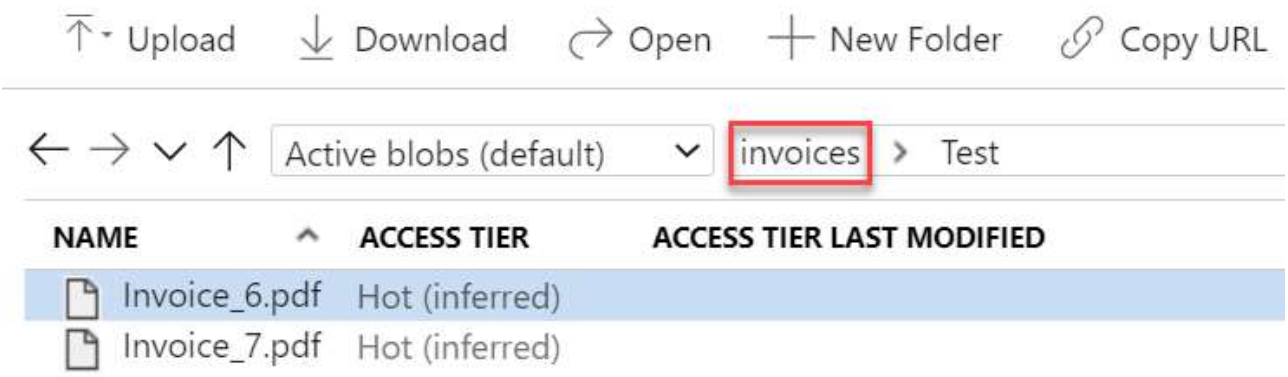
7. In the **Create New Virtual Directory** blade, name the directory **Test**, then select **OK.** This will automatically move you into the new **Test** folder.



The Create New Virtual Directory form is displayed with Test entered in the name field.

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.

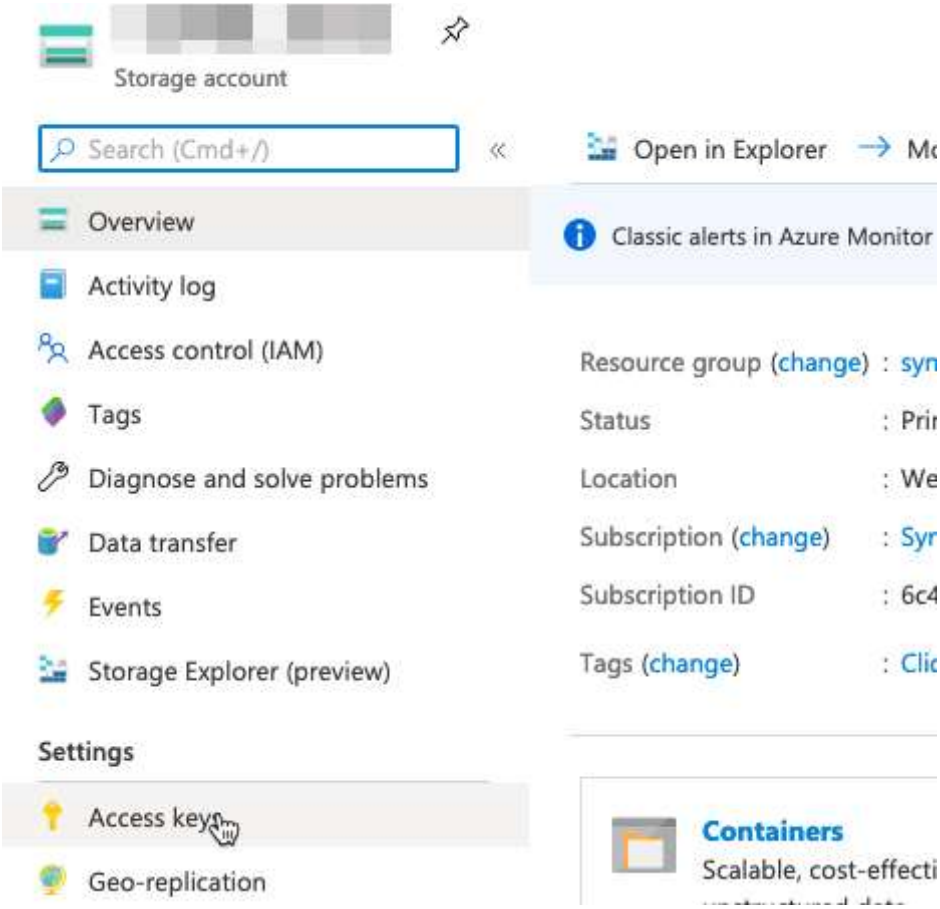


A portion of the Storage Explorer window is displayed with the invoices breadcrumb selected from the location textbox.

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.

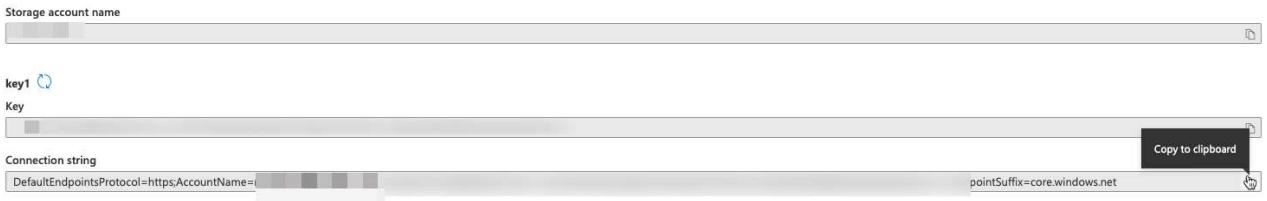
11. From the taskbar, select **Upload**. Upload all invoices located in **Hands-on lab/artifacts/sample_invoices/Train**. These files are Invoice_1.pdf, Invoice_2.pdf, Invoice_3.pdf, Invoice_4.pdf and Invoice_5.pdf.

12. From the left menu, select **Access keys**.



The left menu is displayed with the Access keys link highlighted.

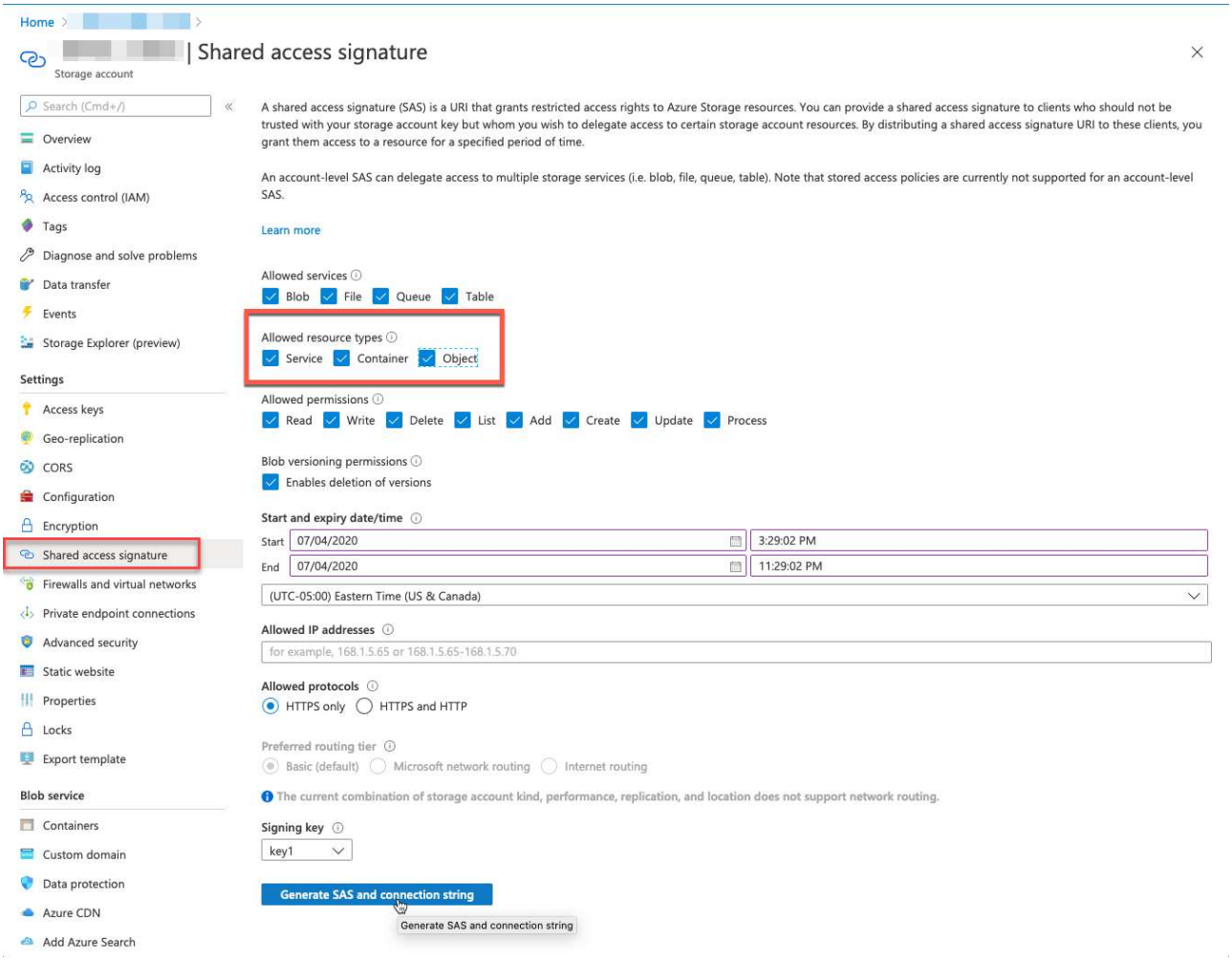
13. Copy the **Connection string** value beneath **key1.** Save it to notepad, Visual Studio Code, or another text file. We'll use this several times



The copy button is selected next to the key1 connection string.

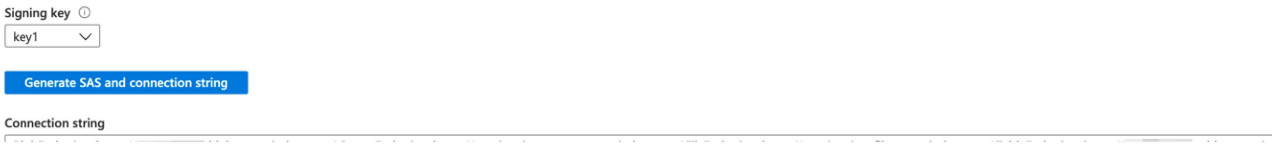
14. From the left menu, beneath **Settings**, select **Shared access signature.**

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



The configuration form is displayed for SAS generation.

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



The SAS form is shown with the shared access signature blob service SAS URL highlighted.

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

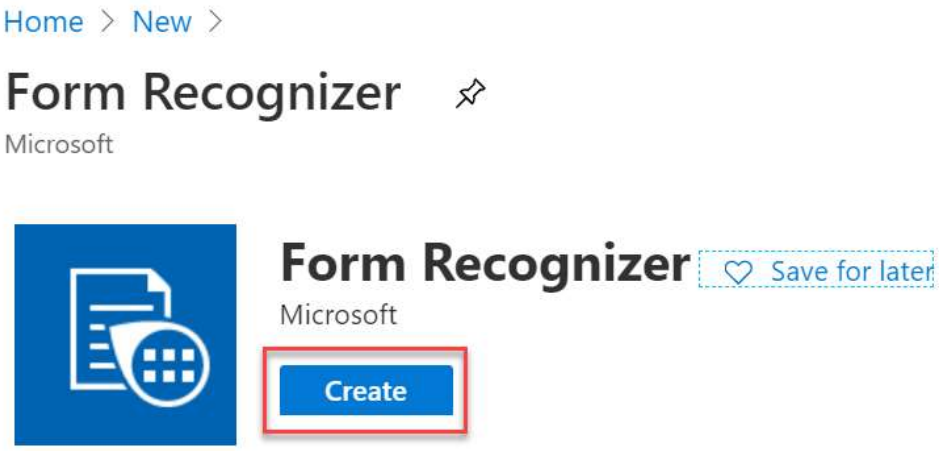
> ***Example**: https://asastore{{suffix}.blob.core.windows.net/invoices?sv=2019-12-12&ss=bfqt&srt …*

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.

The New resource screen is shown with Form Recognizer entered into the search text boxes and selected from the search results.

2. Select **Create**.

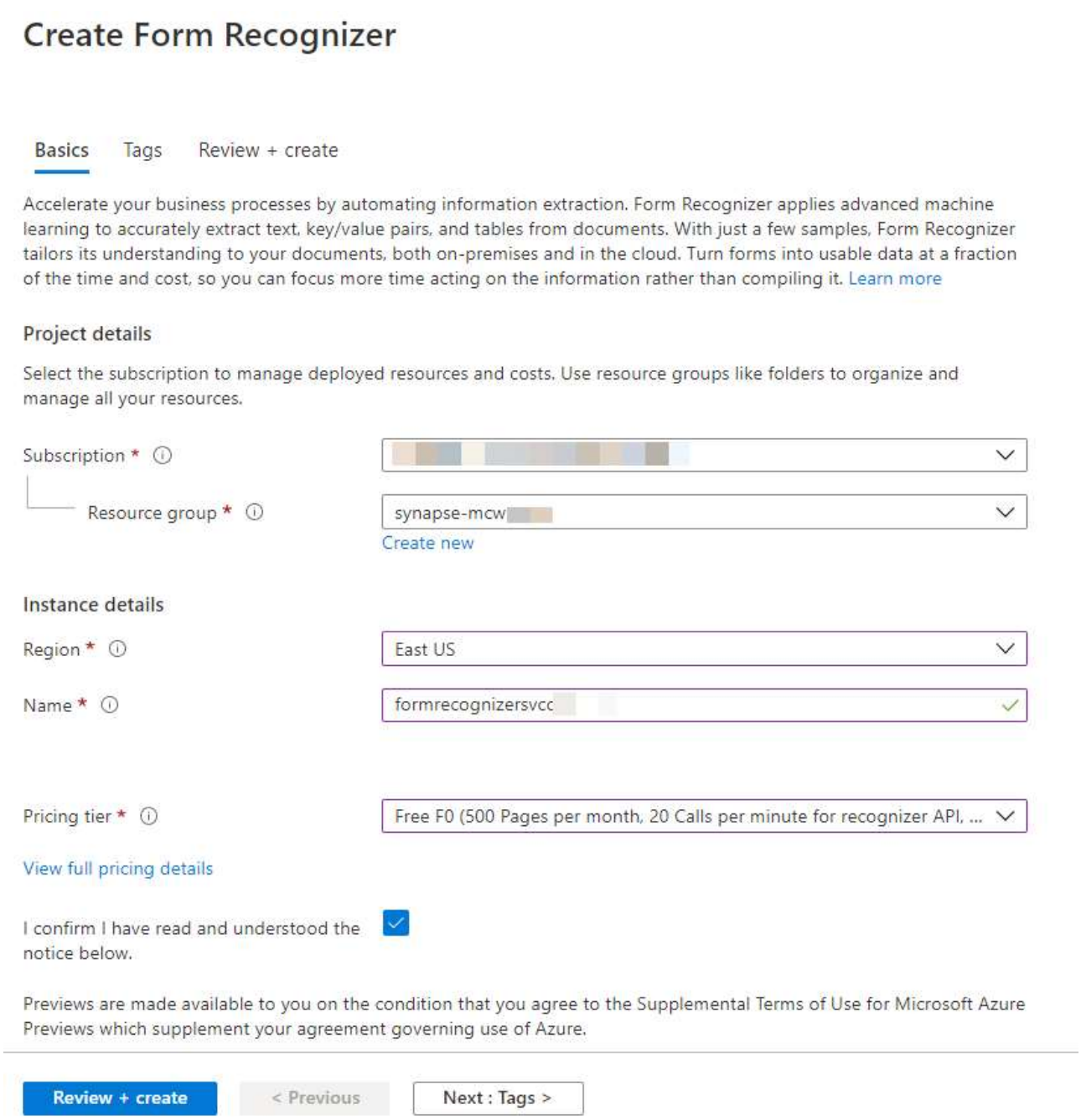


The Form Recognizer overview screen is displayed with the Create button highlighted.

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

Field	Value

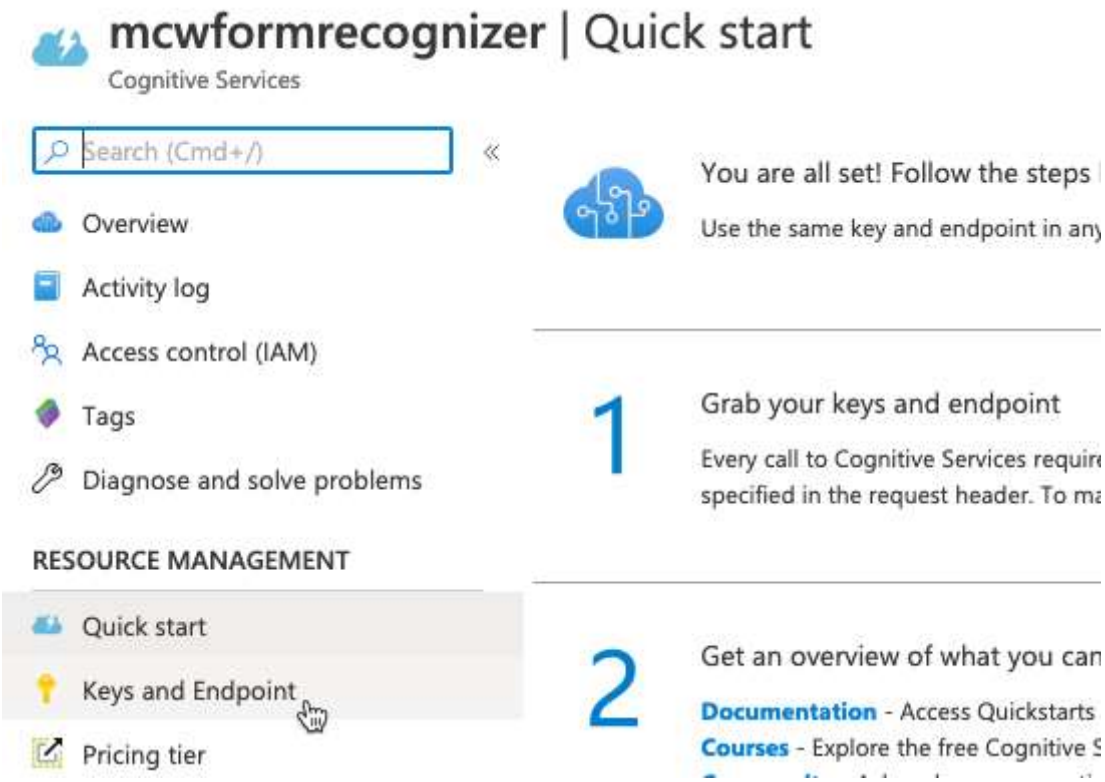
Field	Value
Subscription	Select the lab subscription.
Resource Group	Select the lab resource group
Region	Select the lab region.
Name	Enter a unique name (denoted by the green checkmark indicator) for the form recognition service.
Pricing Tier	Select **Free F0**.
Confirmation checkbox	Checked.



The Form Recognizer configuration screen is displayed populated with the preceding values.

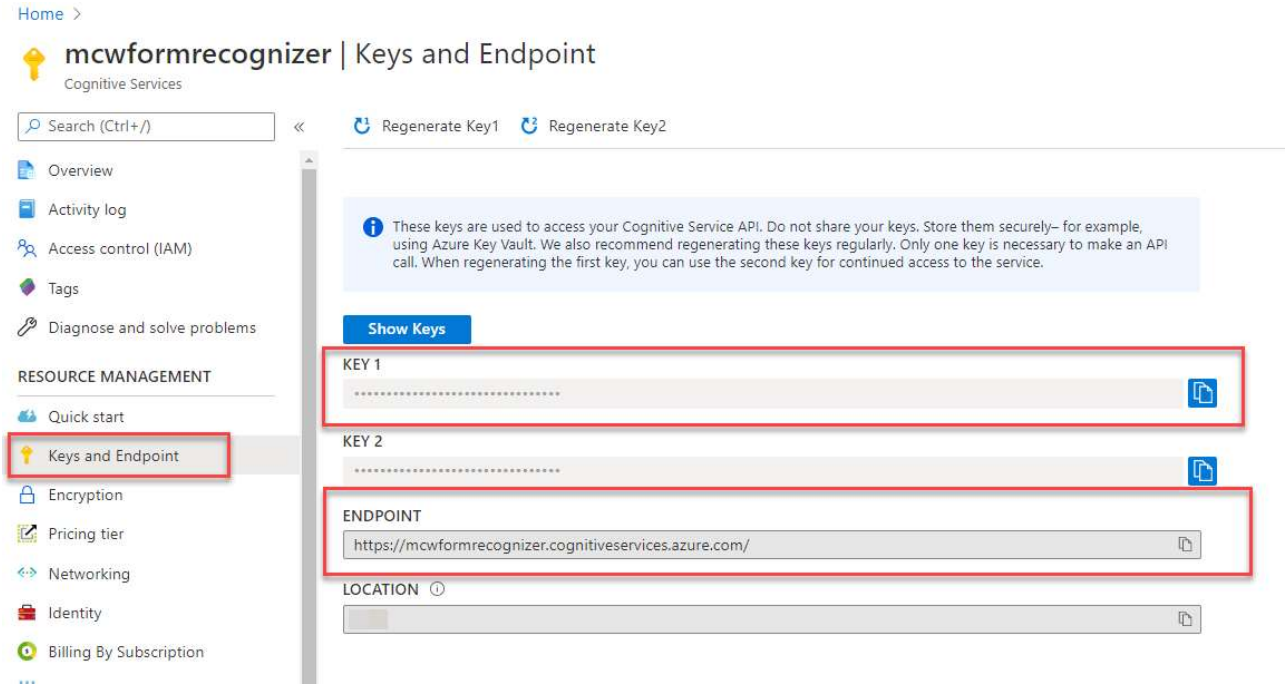
4. Wait for the service to provision then navigate to the resource.

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



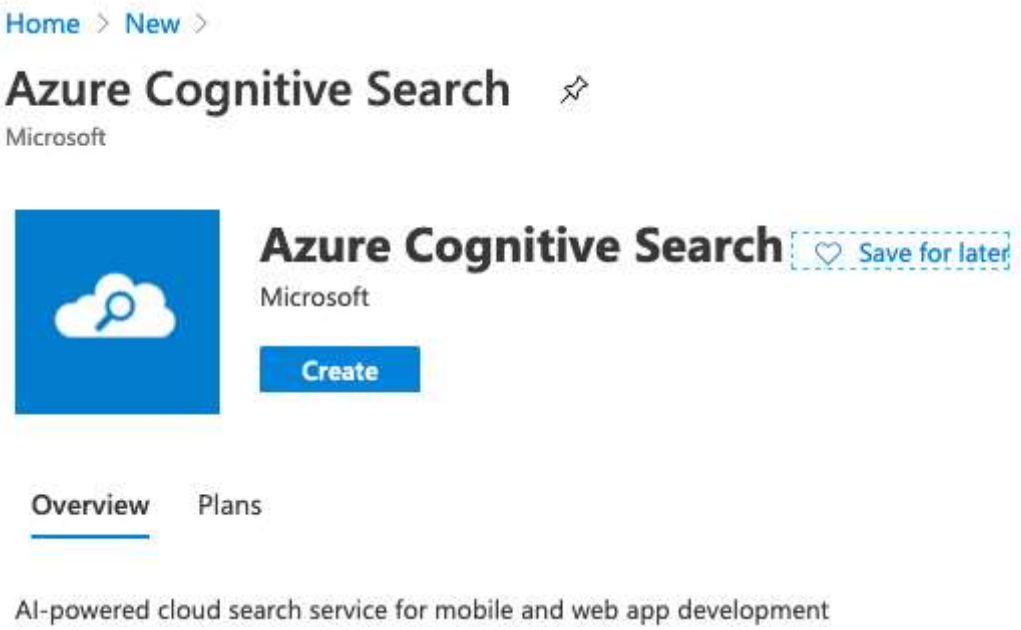
The left side navigation is shown with the Keys and Endpoint item highlighted.

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.



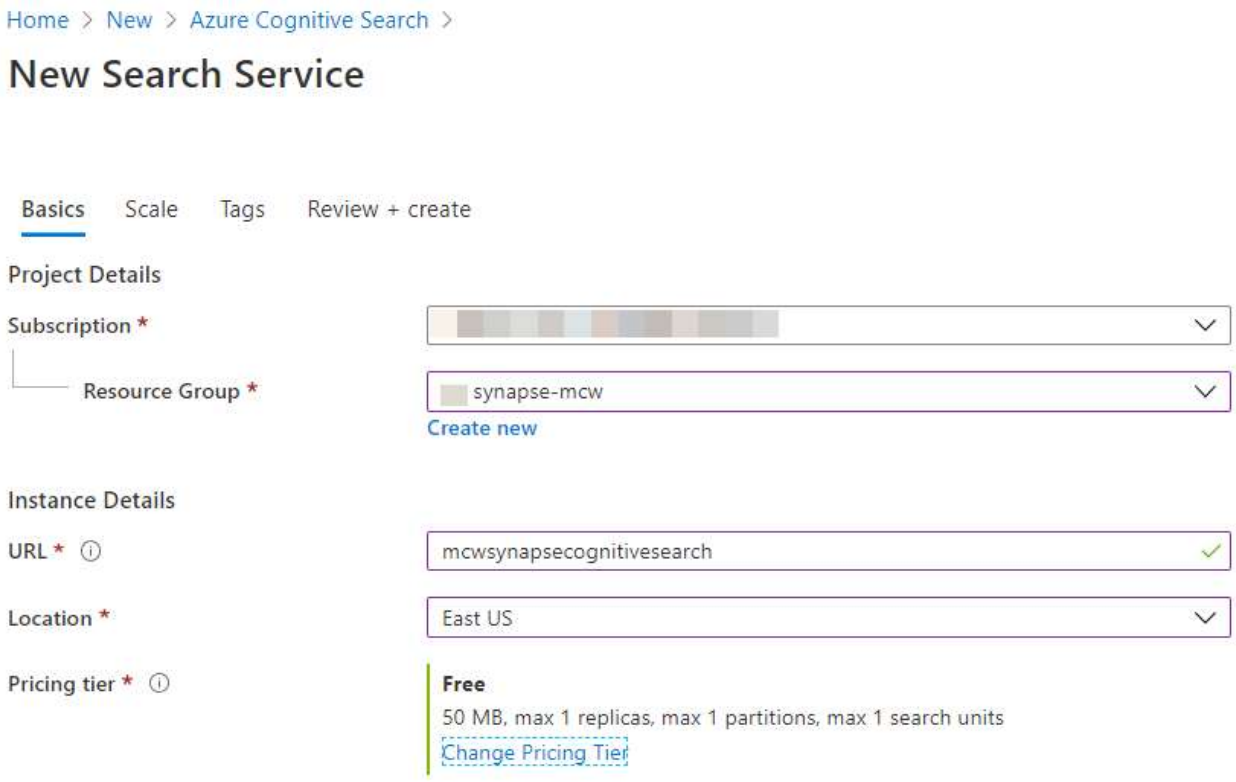
The Keys and Endpoint screen is shown with KEY 1 and ENDPOINT values highlighted.

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



The Azure Cognitive Search overview screen is displayed.

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.**



The configuration screen for Cognitive Search is displayed populated as described above.

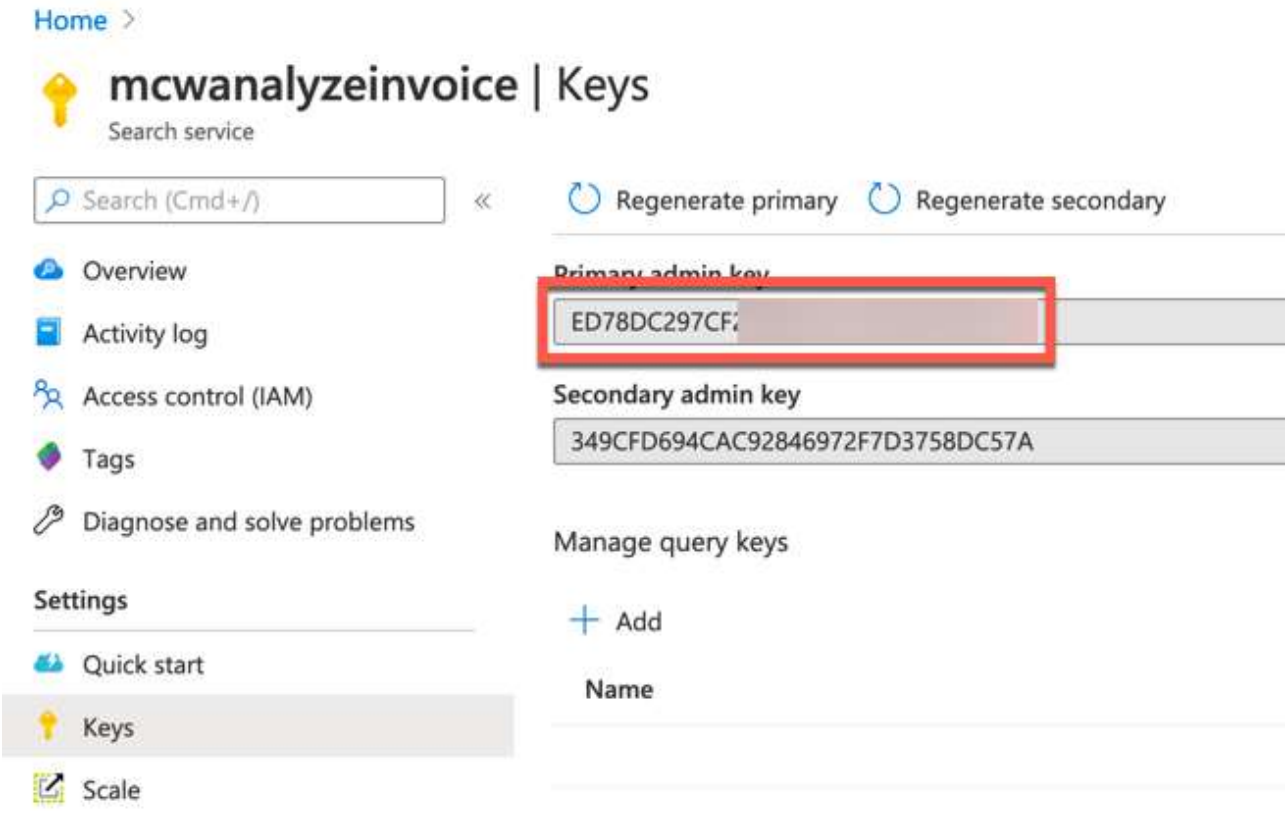
9. Select **Review + create.**

displaying the review + create button

10. Select **Create**.

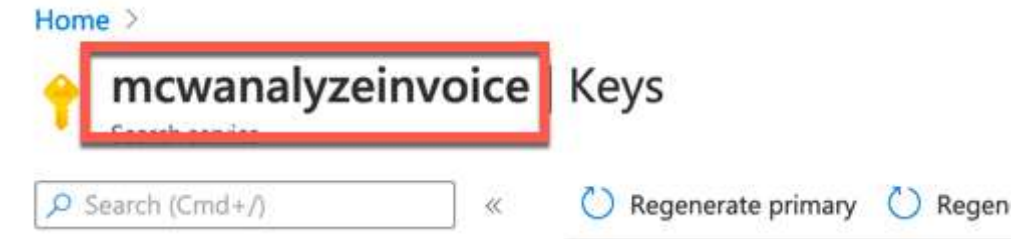
11. Wait for the Search service to be provisioned then navigate to the resource.

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.



They Keys page of the Search service resource is shown with the Primary admin key value highlighted.

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



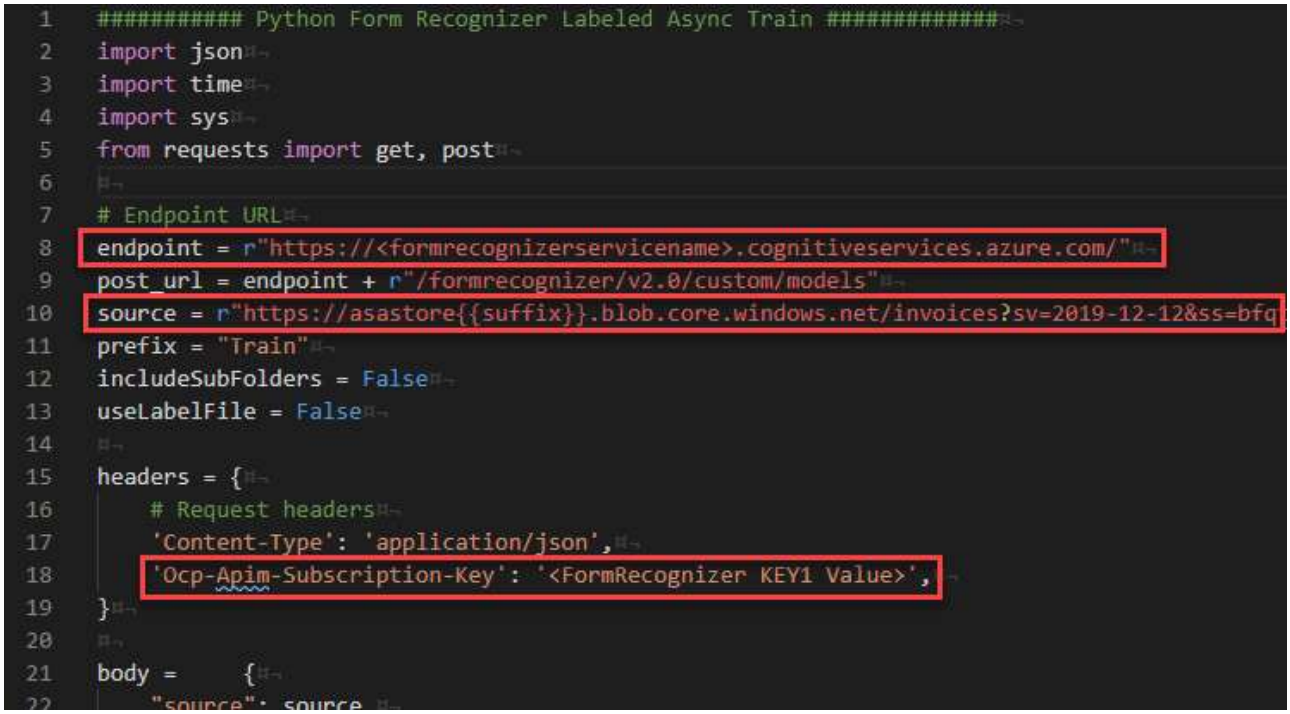
The Search Service name is highlighted on the Keys screen.

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 8, 10, and 18 with the appropriate values indicated below:

    - Line 8: The endpoint of Form Recognizer Service.
    - Line 10: The Blob Service SAS URL storage account with your Train and Test invoice folders.
    - Line 18: The KEY1 value for your Form Recognizer Service.



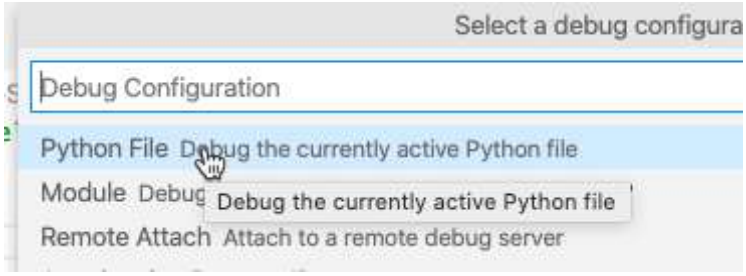
The source code listing of pocformreader.py is displayed with the lines mentioned above highlighted.

17. Save the file.

18. Select Run, then Start Debugging.

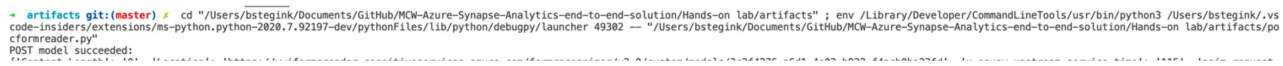
The VS Code File menu is shown with Run selected and Start Debugging highlighted.

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



The Debug Configuration selection is shown with Python File - Debug the currently active Python File highlighted.

20. This process will take a few minutes to complete. 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 value into your text file to use later



A sample output of the python script is shown with a modelId value highlighted.

> ***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***

> ***Note**: If you receive an exception related to SystemExit, this is a known issue in the Python debugger and can be safely ignored. Continue or Terminate the debug execution of the script.*

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**.

   The file structure of the /environment-setup/functions folder is shown.

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.

   *The **init**.py code listing is displayed.*

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.

   The Azure Functions extension panel in VS Code is displayed highlighting the button to publish the function.

   - If prompted for a subscription, select the same subscription as your Synapse workspace.
   - If prompted for the folder to deploy, select **GetInvoiceData**.
   - Choose to **+ Create new Function App in Azure…** (the first one).
   - Give this function a unique name, relative to form recognition.

The Create new function App in Azure dialog is shown with the name populated.

- For the runtime select Python 3.7.

The python runtime version selection dialog is shown with Python 3.7 highlighted.

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

The Region selection dialog is shown.

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 **Function App** resource with the same name.

A resource listing is shown with the Function App highlighted.

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.

The GetInvoiceData function screen is shown with the Get Function Url button highlighted in the taskbar and the URL displayed in a textbox.

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** named **InvoiceKnowledgeStore.postman_collection.json**.

The Postman File menu is expanded with the Import option selected.

The Postman file import screen is displayed with the Upload files button highlighted.

The file selection dialog is shown with the file located in the skillset folder highlighted.

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.

The Collections pane is shown with the Create a KnowledgeStore collection expanded with the four items indicated above.

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.**

In Postman, the ellipsis is expanded next to the Create a KnowledgeStore collection with the edit menu option selected.

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

In the Edit Collection screen, the Variables tab is selected.

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 cognitive search service you created.
search-service-name	The name of the cognitive search service.
storage-account-name	asastore{{suffix}}
storage-connection-string	The connection string from the asastore{{suffix}} storage account.
datasourcename	Enter **invoices**

Variable	Value
indexer-name	Enter **invoice-indexer**
index-name	Enter **invoice-index**
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.

    The Edit Collection Variables screen is shown with a sampling of modified values.

18. Expand the **Create a KnowledgeStore** collection, and select the **Create Index** call, then select the **Body** tab and review the content. For this call, and every subsequent call from Postman - ensure the Content Type is set to **JSON**.

    The Create Index call is selected from the collection, and the Body tab is highlighted.

    The Postman Body tab is selected with the JSON item highlighted.

19. Select “Send”.

    The Postman send button is selected.

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

    The Create Index response is displayed in Postman with the Status of 201 Created highlighted.

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.

    The invoice-indexer is shown with a status of in-progress.

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.

    The execution history of the invoice-indexer is shown as successful.

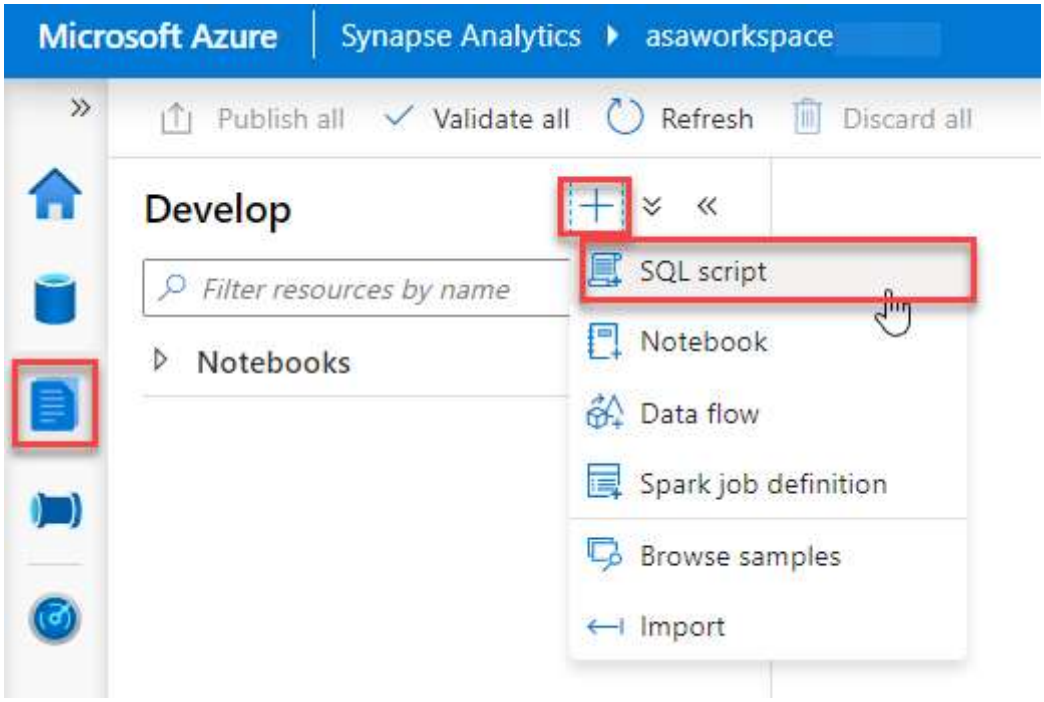
    The invoices-json container is shown with two folders. A JSON file is shown in the blob window.

Task 4: Create the Synapse Pipeline

1. Open your Synapse workspace.

    The Azure Synapse Workspace resource screen is shown with the Launch Synapse Studio button highlighted.

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



The left menu is expanded with the Develop item selected. The Develop blade has the + button expanded with the SQL script item highlighted.

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

The query tab toolbar menu is displayed with the Connect to set to the SQL Pool.

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. At the far right of the top toolbar, select the **Discard all** button as we will not be saving this query. When prompted, choose to **Discard changes**.

The top toolbar menu is displayed with the Discard all button highlighted.

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

The Integrate hub is selected from the left navigation.

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

The + button is expanded with the pipeline option selected.

8. Name your pipeline **InvoiceProcessing**.

The new pipeline properties are shown with InvoiceProcessing entered as the name of the pipeline.

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

The Add trigger button is expanded with the New/Edit option selected.

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

The Add triggers form is displayed with the Choose trigger dropdown expanded and the +New item is selected.

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**.

The new trigger form is displayed with the trigger set to start every 5 minutes.

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.

The pipeline designer is shown with an indicator of a drag and drop operation of the data flow activity.

14. On the **Adding data flow** form, select **Create new data flow** and select **Data flow.**

The Adding data flow form is displayed populated with the preceding values.

15. On the **Properties** blade of the new Data Flow, on **General** tab, enter **NewInvoicesProcessing** in the **Name** field.

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

The NewInvoicesProcessing designer is shown with the Add source box selected.

17. 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.

The Source settings tab is displayed populated with the name of jsonInvoice and the +New button next to the Dataset field is selected.

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

The New dataset blade is displayed with Azure Blob Storage selected.

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

The select format screen is displayed with Json selected as the type.

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

A portion of the Set properties form is displayed populated with the above values.

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

The set properties form is displayed with the file path and import schema fields populated as described.

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

The Set properties form is displayed with the sampleformrecognizer.json selected as the selected file.

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

The Source options tab is shown with the Wildcard paths field populated as specified.

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

The + button is highlighted to the lower right of the source activity.

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

With the + button expanded, Derived column is selected from the list of options.

26. On the **Derived column's settings** tab, provide the output stream name of **RemoveCharFromStrings**. Then for the Columns field, select the following 3 columns and configure them as follows, using the **Open expression builder** link for the expressions:

Column	Expression
productprice	toDecimal(replace(productprice,'', '))||*totalcharges*|*toDecimal*(*replace*(*replace*(*totalcharges*, '','),',','))
quantity	toInteger(replace(quantity,',',''))

The Derived column's settings tab is shown with the fields populated as described.

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

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

    In the Row modifier section, the Alter Row option is selected.

29. 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,"")**

    The Alter row settings tab is shown populated with the values described above.

30. 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.

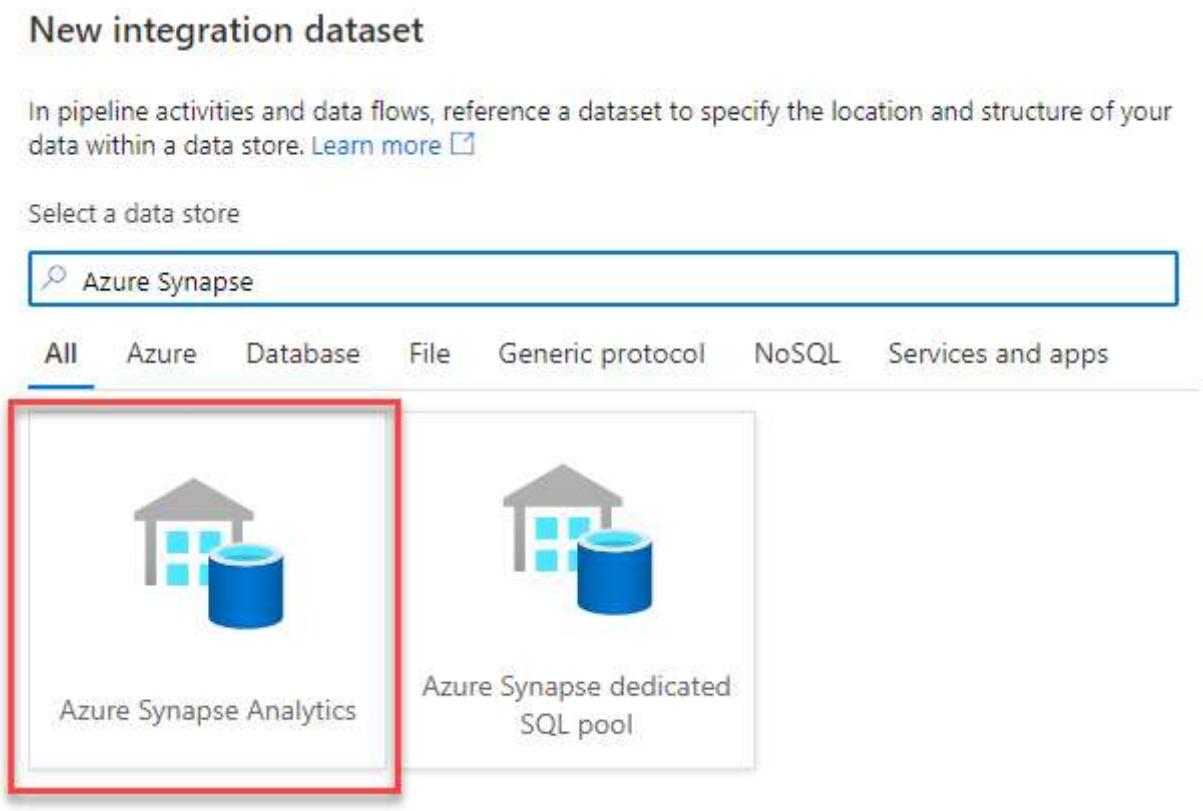
31. Within the **Destination** section, select **Sink.**

    In the activity listing, the sink option is selected from within the Destination section.

32. 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.

    The sink tab is shown with the output stream name set to SQLDatabase and the +New button selected next to the Dataset field.

33. On the **New integration dataset** blade, enter **Azure Synapse** as a search term and select the **Azure Synapse Analytics** item. Select **Continue.**



The New integration dataset form is shown with Azure Synapse entered in the search box and the Azure Synapse Analytics item highlighted.

34. 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**.

The Dataset Set properties form is displayed populated as described.

35. 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**.

The Settings tab of the Sink activity is shown and is populated as described.

36. 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

The Mapping tab is displayed with Auto Mapping disabled and the column mappings from the table above are defined.

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

The InvoiceProcessing tab is selected at the top of the workspace.

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

The data flow activity Settings tab is displayed.

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

The data flow activity Settings tab is displayed with its form populated as indicated above.

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

The Publish All button is selected from the top toolbar.

41. Select **Publish**.

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

The Publishing completed notification is shown.

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

    The Monitor hub is selected from the left menu.

44. 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.

    On the Pipeline runs list, the InvoiceProcessing pipeline is shown as in-progress.

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

    The Pipeline runs list is displayed with the InvoiceProcessing pipeline shown as succeeded.

46. 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
    ```

    show the data in the databases

Exercise 6: Security

Duration: 30 minutes

Task 1: Column level security

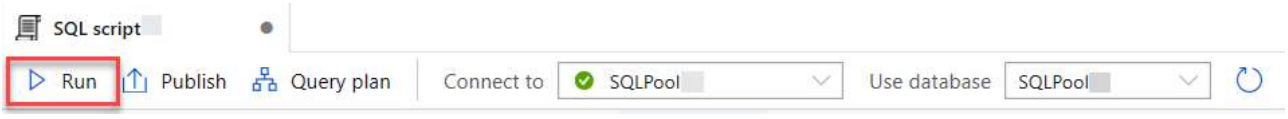
It is important to identify data columns of that hold sensitive information. Types of sensitive information could be social security numbers, email addresses, credit card numbers, financial totals, and more. Azure Synapse Analytics allows you define permissions that prevent users or roles select privileges 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 d
esign and coding of security in applications.
    It ensures column level security by restricting column access to pr
otect sensitive data. */

/* Scenario: In this scenario we will be working with two users. The fi
rst one is the CEO, he has access to all
    data. The second one is DataAnalystMiami, this user doesn't have ac
cess 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 th
at 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 infor
mation in the database
    and a Data Analyst, to whom only required information should be pre
sented.*/

-- Step:2 Verify the existence of the “CEO” and “DataAnalystMiami” user
s in the Datawarehouse.
SELECT Name as [User1] FROM sys.sysusers WHERE name = N'CEO';
SELECT Name as [User2] FROM sys.sysusers WHERE name = N'DataAnalystMiam
i';


-- Step:3 Now let us enforcing column level security for the DataAnalys
tMiami.
/*  The CampaignAnalytics table in the warehouse has information like P
roductID, Analyst, CampaignName, Quantity, Region, State, City, Revenue
Target 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], [R
```

```
egion], [State], [City], [RevenueTarget]) 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 co
lumn
EXECUTE AS USER ='DataAnalystMiami';
select TOP 100 * from wwi_mcw.CampaignAnalytics;
---
-- The following query will succeed since we are not including the Reve
nue column in the query.
EXECUTE AS USER ='DataAnalystMiami';
select [Analyst],[CampaignName], [Region], [State], [City], [RevenueTar
get] from wwi_mcw.CampaignAnalytics;

-- 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 follo
wing 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;
```



The query tab toolbar is displayed with the Run button selected.

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

The top toolbar menu is displayed with the Discard all button highlighted.

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.

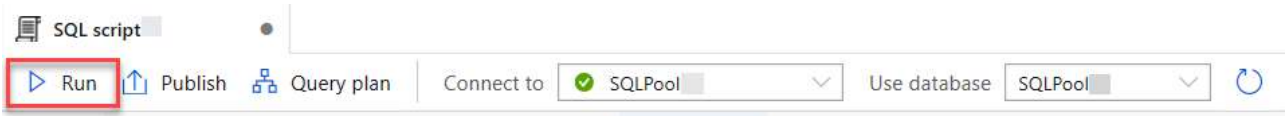
1. Create a new SQL script by selecting **Develop** from the left menu, then in the **Develop** blade, expanding the **+** button and selecting **SQL script**.

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 me
mbership to control access 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 D
ataAnalystSanDiego.
--     Each analyst has jurisdiction across a specific Region. DataAnal
ystMiami on the South East Region
--      and DataAnalystSanDiego on the Far West region.
SELECT DISTINCT Analyst, Region FROM wwi_mcw.CampaignAnalytics order by
Analyst ;

/* Scenario: WWI requires that an Analyst only see the data for their o
wn 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 fi
lter data to a specific Analyst value. */

/* We will define this filter using what is called a Security Predicat
e. This is an inline table-valued function that allows
    us to evaluate additional logic, in this case determining if the An
alyst executing the query is the same as the Analyst
    specified in the Analyst column in the row. The function returns 1
 (will return the row) when a row in the Analyst column is the same as
 the user executing the query (@Analyst = USER_NAME()) or if the user e
xecuting 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 defin
e 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, Dat
aAnalystSanDiego;

-- Step:3 Let us now test the filtering predicate, by selecting data fr
om 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 DataAn
alystMiami

-- 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 execu
te the following.
ALTER SECURITY POLICY SalesFilter
WITH (STATE = OFF);

DROP SECURITY POLICY SalesFilter;
```

```
DROP FUNCTION Security.fn_securitypredicate;
DROP SCHEMA Security;
```



The query tab toolbar is displayed with the Run button selected.

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

The top toolbar menu is displayed with the Discard all button highlighted.

Task 3: Dynamic data masking

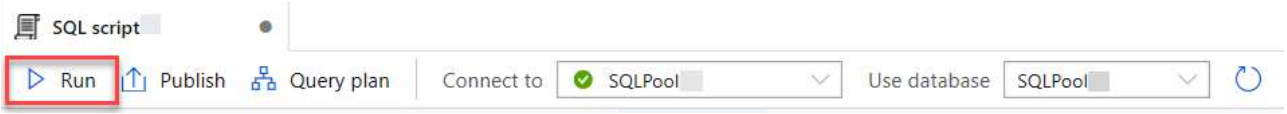
As an alternative to column level security, SQL Administrators also have the option of masking sensitive data. This will result in data being obfuscated when returned in queries. The data is still stored in a pristine state in the table itself. SQL Administrators can grant unmask privileges to users 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 customers
    to designate how much of the sensitive data to reveal with minimal
 impact on the application layer.
    Let see how */

/* Scenario: WWI has identified sensitive information in the CustomerIn
fo table. They would like us to
    obfuscate the CreditCard and Email columns of the CustomerInfo tabl
e 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) ap
plied 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 'CustomerIn
fo' 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 colum
ns.
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 t
he 'CustomerInfo' table.
GRANT SELECT ON wwi_mcw.CustomerInfo TO DataAnalystMiami;

-- Step:6 Logged in as  'DataAnalystMiami' let's execute the select que
ry 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
```



The query tab toolbar is displayed with the Run button selected.

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

The top toolbar menu is displayed with the Discard all button highlighted.

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.

Task 1: Create a SQL Datastore and source Dataset

1. Open the lab resource group, locate and open the **amlworkspace{{suffix}}** Machine Learning resource.

   The lab resource group is shown with the Machine Learning resource selected

2. On the **Overview** screen of the Machine Learning resource, select the **Studio web URL** link.

   The machine learning resource overview screen is selected with the Studio web URL link highlighted.

3. From the left menu of **Azure Machine Learning Studio**, select the **Datastores** item.

   The Machine Learning Studio menu is shown with the Datastores item highlighted

4. On the **Datastores** screen top menu, select **+ New datastore**.

5. On the **New datastore** blade, configure it as follows and select **Create**:

Field	Value
New datastore (name)	sqlpool01
Datastore type	Azure SQL database
Account selection method	From Azure subscription

Field	Value
Subscription ID	Select the lab subscription.
Server name / database name	Select asaworkspace{{suffix}}/SQLPool01.
Authentication type	SQL authentication
User ID	asa.sql.admin
Password	The SQL Admin password you chose when deploying the lab resources.

The new datastore blade is shown populated with the preceding values.

6. From the left menu, select **Datasets**, and with the **Registered datasets** tab selected, expand the **+ Create dataset** button and select **From datastore**.

The Datasets screen is displayed with the +Create dataset button highlighted.

7. In the **Create dataset from datastore** Basic info form, name the dataset **AggregatedProductSeasonality** and select **Next**.

The basic info form is displayed populated with the preceding values.

8. On the **Datastore selection** form, select **Previously created datasource**, choose **sqlpool01** from the list and select the **Select datastore** button.

The Datastore selection form is displayed as described above.

9. In the next **Datastore selection** form, enter the following **SQL query**. Then expand the **Advanced settings** and enter **100** for the **Query timeout (seconds)** value. Select **Next**:

```
SELECT P.ProductId,P.Seasonality,S.TransactionDateId,COUNT(*) as Transa
ctionItemsCount
FROM wwi_mcw.SaleSmall S
JOIN wwi_mcw.Product P ON S.ProductId = P.ProductId
where TransactionDateId between 20190101 and 20191231
GROUP BY P.ProductId ,P.Seasonality,S.TransactionDateId
```

The datastore selection form is displayed populated with the preceding query.

10. The **Settings and preview** data table will be displayed after a few moments. Review this data, then select the **Next** button.

The settings and preview screen is displayed showing a table of data.

11. Review the **Schema** field listing, then select **Next.**

The Schema screen is displayed showing a listing of columns and their types.

12. On the **Confirm details** screen, select **Create**.

The dataset Confirm details screen is displayed showing a summary of the choices from the previous steps.

Task 2: Create compute infrastructure

1. From the left menu of Machine Learning Studio, select **Compute.**

2. On the **Compute** screen with the **Compute instances** tab selected. Choose the **Create** button.

The Azure Machine Learning Studio compute screen is displayed, with the compute instances tab selected, and the Create button highlighted.

3. On the **Create compute instance**, **Select virtual machine** form, configure it as follows, then select **Next**:

Field	Value

Field	Value
Virtual machine type	CPU
Virtual machine size	Search for and select Standard_DS3_v2.

The new compute instance virtual machine form is displayed populated with the preceding values.

4. On the **Configure Settings** form, enter a globally unique **Compute name** of your choice, and select **Create**.

The new compute instance settings form is displayed populated with a compute name

5. Select the **Compute clusters** tab, and select **Create**.

6. On the **New compute cluster**, **Select virtual machine** form, configure the virtual machine as follows, then select **Next**:

Field	Value
Virtual machine priority	Dedicated
Virtual machine type	CPU
Virtual machine size	Search for and select Standard_DS3_v2.

The New compute cluster virtual machine form is displayed with the preceding values.

7. On the **Configure Settings** form, configure it as follows, then select **Create**:

Field	Value
Compute name	automlcluster
Minimum number of nodes	0
Maximum number of nodes	3
Idle seconds before scale down	120

The new compute cluster configure settings form is displayed populated with the preceding values.

Task 3: Use a notebook in AML Studio to prepare data and create a Product Seasonality Classifier model using XGBoost

1. In Azure Machine Learning (AML) Studio, select **Notebooks** from the left menu.
2. In the **Notebooks** pane, select the **Upload** icon from the toolbar.

In Azure Machine Learning Studio, the Notebooks item is selected from the left menu, and the Upload Icon is highlighted in the Notebooks panel.

3. In the **Open** dialog, select **Hands-on lab/artifacts/ProductSeasonality_sklearn.ipynb**. When prompted, check the boxes to **Overwrite if already exists** and **I trust contents of this file** and select **Upload**.

A dialog is displayed with the Overwrite if already exists and the I trust contents of this file checkboxes checked.

4. In the top toolbar of the notebook, expand the **Editors** item, and select **Edit in Jupyter**.

On the notebook toolbar, the Editors item is expanded with the Edit in Jupyter item selected.

5. Review and run each cell in the notebook individually to gain understanding of the functionality being demonstrated.

> ***Note**: Running this notebook in its entirety is required for the next task.*

Task 4: Leverage Automated ML to create and deploy a Product Seasonality Classifier model

1. In Azure Machine Learning (AML) Studio, select **Experiments** from the left menu, then expand the **+ Create** button, and select **Automated ML run**.

The AML Studio Experiments screen is shown with the Create button expanded and the Automated ML run item selected.

2. In the previous task, we registered our PCA dataframe (named **pcadata**) to use with Auto ML. Select **pcadata** from the list and select **Next**.

On the Select dataset screen, the pcadata item is selected from the dataset list.

3. On the **Configure run** screen, select the **Create a new compute** link beneath the **Select compute cluster** field.

4. Back on the **Configure run** form, name the experiment **ProductSeasonalityClassifier**, select **Seasonality** as the **Target column** and select **automlcluster** as the compute cluster. Select **Next**.

The Configure run form is displayed populated with the preceding values.

5. On the **Select task type** screen, select **Classification**, then choose **Finish**.

The Select task type screen is displayed with the Classification item selected.

6. The experiment will then be run. It will take approximately 20-25 minutes for it to complete. Once it has completed, it will display the run results details. In the **Best model summary** box, select the **Algorithm name** link.

The Run is shown as completed and the link below Algorithm name in the Best model summary box is selected.

7. On the Model run screen, select **Deploy** from the top toolbar.

The specific model run screen is shown with the Deploy button selected from the top toolbar.

8. On the **Deploy a model** blade, configure the deployment as follows, then select **Deploy**:

Field	Value
Name	productseasonalityclassifier
Description	Product Seasonality Classifier.

Field	Value
Compute type	Azure Container Instance
Enable authentication	Off

The Deploy a model blade is shown populated with the preceding values.

9. Once deployed, the Model summary will be displayed. You can view the endpoint by selecting the **Deploy status** link.

The successful model deployment was successful and the Deploy status link is highlighted.

10. Review the details of the deployed model service endpoint.

The service endpoint details screen is displayed.

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.

   The Develop menu item is highlighted.

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

   The SQL script context menu item is highlighted.

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

   The connect to option is highlighted in the query toolbar.

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 I
mportance
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.

The run button is highlighted in the query toolbar.

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 a Azure Synapse Pipeline that executes a large number of queries.

7. Select the `Integrate` item from the left menu.

8. **Run** the **Exercise 8 - ExecuteDataAnalystandCEOQueries** 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**.

The add trigger and trigger now menu items are highlighted.

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.

The Monitor Hub icon is selected from the left menu, and the Cancel recursive button is selected on the in progress pipeline.

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 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 I
mportance
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>d
ateadd(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:

SQL query results.

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 W
HERE 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` item from the left menu.
- **Run** the **Exercise 8 - ExecuteDataAnalystandCEOQueries** 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 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 I
mportance
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>d
ateadd(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.

SQL query results showing asa.sql.workload01 queries with a higher importance than those queries from asa.sql.workload02.

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.

   The Develop menu item is highlighted.

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

   The SQL script context menu item is highlighted.

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

   The connect to option is highlighted in the query toolbar.

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

```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_groups wh
ere 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_classifie
rs where  name = 'CEODreamDemo')
BEGIN
    Create Workload Classifier CEODreamDemo with
    ( Workload_Group ='CEODemo',MemberName='asa.sql.workload02',IMPORTA
NCE = 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 `Integrate` item from the left menu. **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:

    The script results show that each session was executed with below normal importance with two queries being run in parallel.

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 (gua
ranteed 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',IMPORTAN
CE = 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 `Integrate` item from the left menu. **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.

The SQL results pane is shown with multiple queries being run in parallel.

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.

   The Develop menu item is highlighted.

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

   The SQL script context menu item is highlighted.

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

   The connect to option is highlighted in the query toolbar.

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

   ```
   SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed' and ses
   sion_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 `Integrate` item from the left menu. **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*):

   Active query results.

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:

The query results are displayed.

> *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 `Integrate` item from the left menu. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline.

The add trigger and trigger now menu items are highlighted.

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.

   The pipeline runs blade is displayed within the Monitor hub.

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

   The Cancel option is highlighted.

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 `Integrate` item from the left menu. **Run** the **Exercise 8 - Execute Business Analyst Queries** Pipeline.

   The add trigger and trigger now menu items are highlighted.

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.

   The SQL requests blade is displayed within the Monitor hub.

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.

   The request content link is displayed over a 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.