Exercise - Integrate a Notebook within Azure Synapse Pipelines

In this unit, you create an Azure Synapse Spark notebook to analyze and transform data loaded by a mapping data flow and store the data in a data lake. You create a parameter cell that accepts a string parameter that defines the folder name for the data the notebook writes to the data lake. You then add this notebook to a Synapse pipeline and pass the unique pipeline run ID to the notebook parameter so that you can later correlate the pipeline run with the data saved by the notebook activity. Finally, you use the Monitor hub in Synapse Studio to monitor the pipeline run, obtain the run ID, then locate the corresponding files stored in the data lake.

About Apache Spark and notebooks

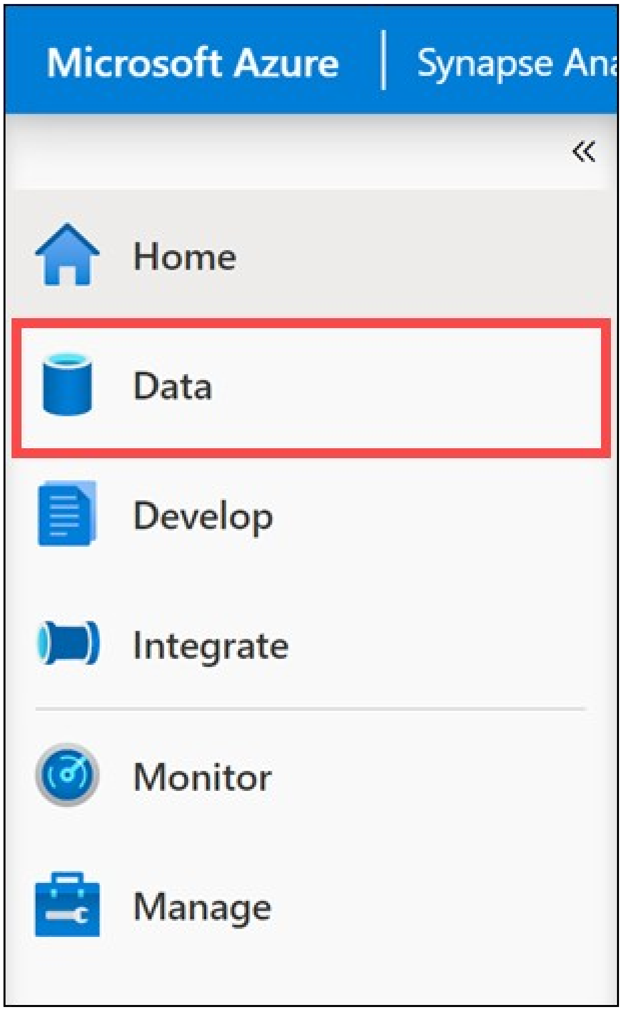
Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud.

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

Create a Synapse Spark notebook

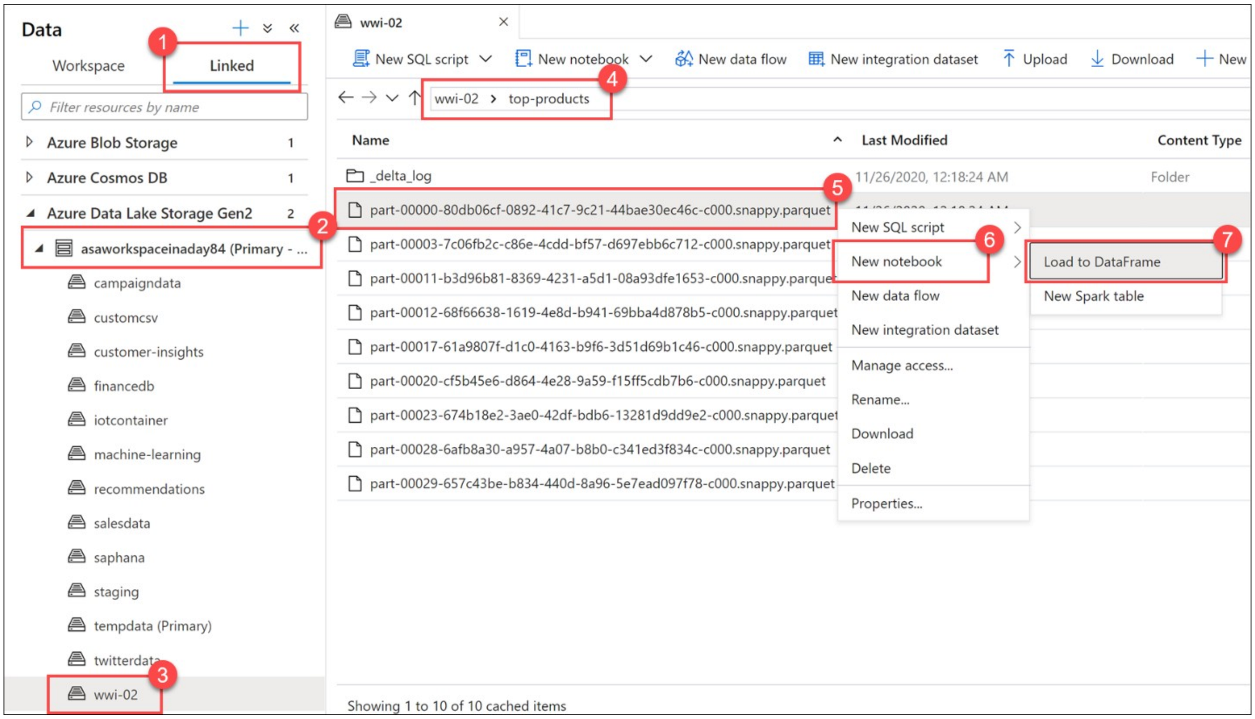
Suppose you created a Mapping Data flow in Synapse Analytics to process, join, and import user profile data. Now you want to find the top 5 products for each user, based on which ones are both preferred and top, and have the most purchases in the past 12 months. Then, you want to calculate the top 5 products overall.

In this step, you create a Synapse Spark notebook to make these calculations. 1. Open Synapse Analytics Studio, and then navigate to the Data hub.



Data hub

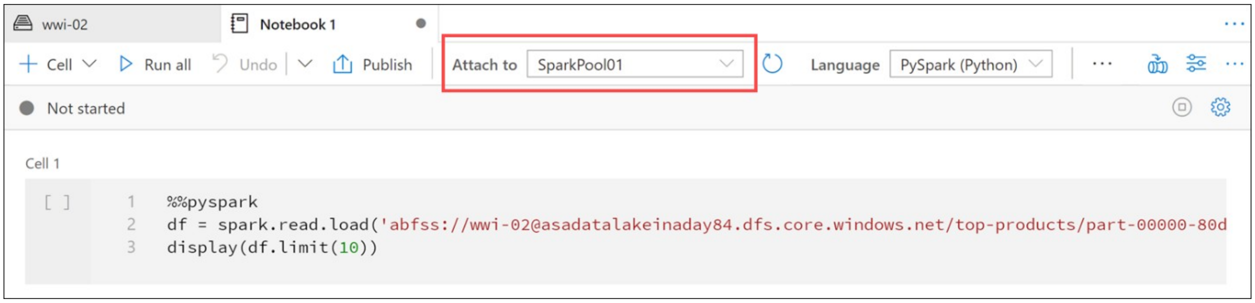
2. Select the **Linked** tab **(1)** and expand the **primary data lake storage account (2)** underneath the **Azure Data Lake Storage Gen2**. Select the **wwi-02** container **(3)** and open the **top-products** folder **(4)**. Right-click on any Parquet file **(5)**, select the **New notebook** menu item **(6)**, then select **Load to DataFrame (7)**. If you don't see the folder, select Refresh above.



Name	Last Modified	Content Type
_delta_log	11/26/2020, 12:18:24 AM	Folder
part-00000-80db06cf-0892-41c7-9c21-44bae30ec46c-c000.snappy.parquet		
part-00003-7c06fb2c-c86e-4cdd-bf57-d697ebb6c712-c000.snappy.parquet		
part-00011-b3d96b81-8369-4231-a5d1-08a93dfe1653-c000.snappy.parquet		
part-00012-68f66638-1619-4e8d-b941-69bba4d878b5-c000.snappy.parquet		
part-00017-61a9807f-d1c0-4163-b9f6-3d51d69b1c46-c000.snappy.parquet		
part-00020-cf5b45e6-d864-4e28-9a59-f15ff5cdb7b6-c000.snappy.parquet		
part-00023-674b18e2-3ae0-42df-bdb6-13281d9dd9e2-c000.snappy.parquet		
part-00028-6afb8a30-a957-4a07-b8b0-c341ed3f834c-c000.snappy.parquet		
part-00029-657c43be-b834-440d-8a96-5e7ead097f78-c000.snappy.parquet		

The Parquet file and new notebook option are highlighted

3. Make sure the notebook is attached to your Spark pool.



Attach to SparkPool01

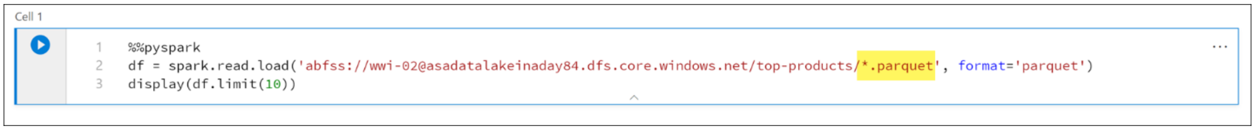
Language PySpark (Python)

```
%%pyspark
df = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/part-00000-80d
display(df.limit(10))
```

The attach to Spark pool menu item is highlighted

4. Replace the Parquet file name with **.parquet (1)* to select all Parquet files in the *top-products* folder. For example, the path should be similar to:

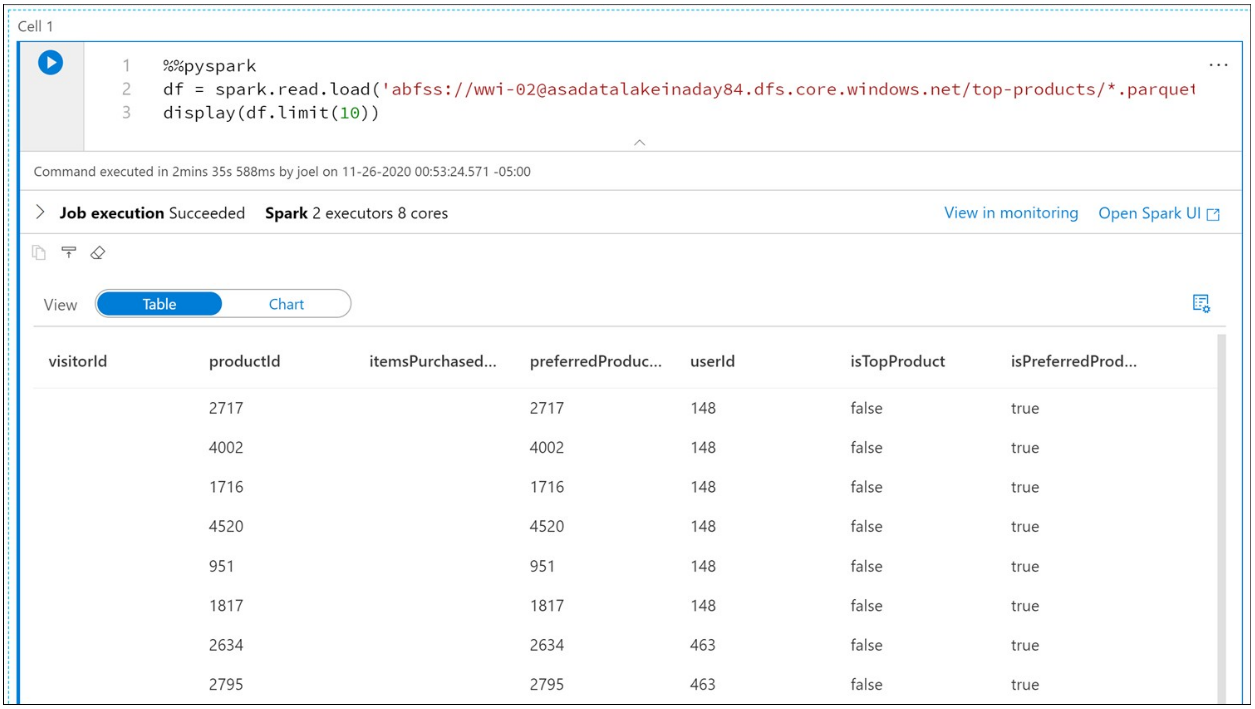
abfss://wwi-02@YOUR_DATALAKE_NAME.dfs.core.windows.net/top-products/*.parquet.



```
%%pyspark
df = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet', format='parquet')
display(df.limit(10))
```

The filename is highlighted.

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



Cell 1

```
%%pyspark
df = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet
display(df.limit(10))
```

Command executed in 2mins 35s 588ms by joel on 11-26-2020 00:53:24.571 -05:00

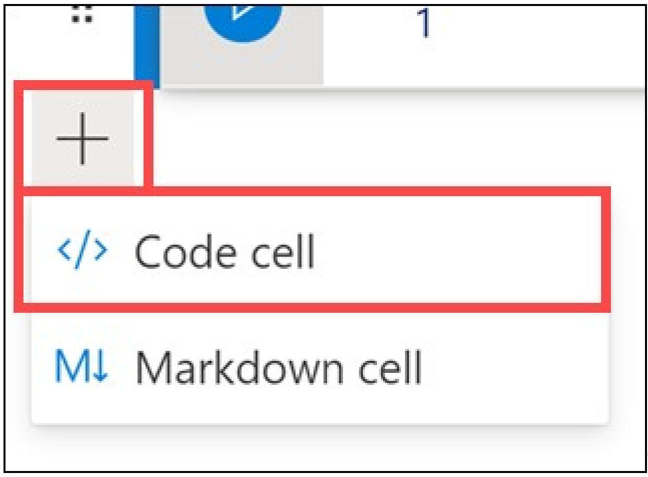
Job execution Succeeded **Spark** 2 executors 8 cores

visitorId	productId	itemsPurchased...	preferredProduc...	userId	isTopProduct	isPreferredProd...
	2717		2717	148	false	true
	4002		4002	148	false	true
	1716		1716	148	false	true
	4520		4520	148	false	true
	951		951	148	false	true
	1817		1817	148	false	true
	2634		2634	463	false	true
	2795		2795	463	false	true

The cell results are displayed.

Note: **The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 3-5 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.**

6. Create a new cell underneath by selecting the + button nd selecting the Code cell item. The + button is located beneath the notebook cell on the left. Alternatively, you can also expand the + Cell menu in the Notebook toolbar and select the Code cell item.



The Add Code menu option is highlighted.

7. Enter and execute the following in the new cell to populate a new dataframe called *top_purchases*, create a new temporary view named *top_purchases* , and show the first 100 rows:

```
topPurchases = df.select(
    "UserId", "ProductId",
    "ItemsPurchasedLast12Months", "IsTopProduct",
    "IsPreferredProduct")

# Populate a temporary view so we can query from SQL
topPurchases.createOrReplaceTempView("top_purchases")

topPurchases.show(100)
```

The output should look similar to the following:

```
|   833|     1087|                      null|       false|              true|
```

8. Execute the following in a new cell to create a new temporary view by using SQL:

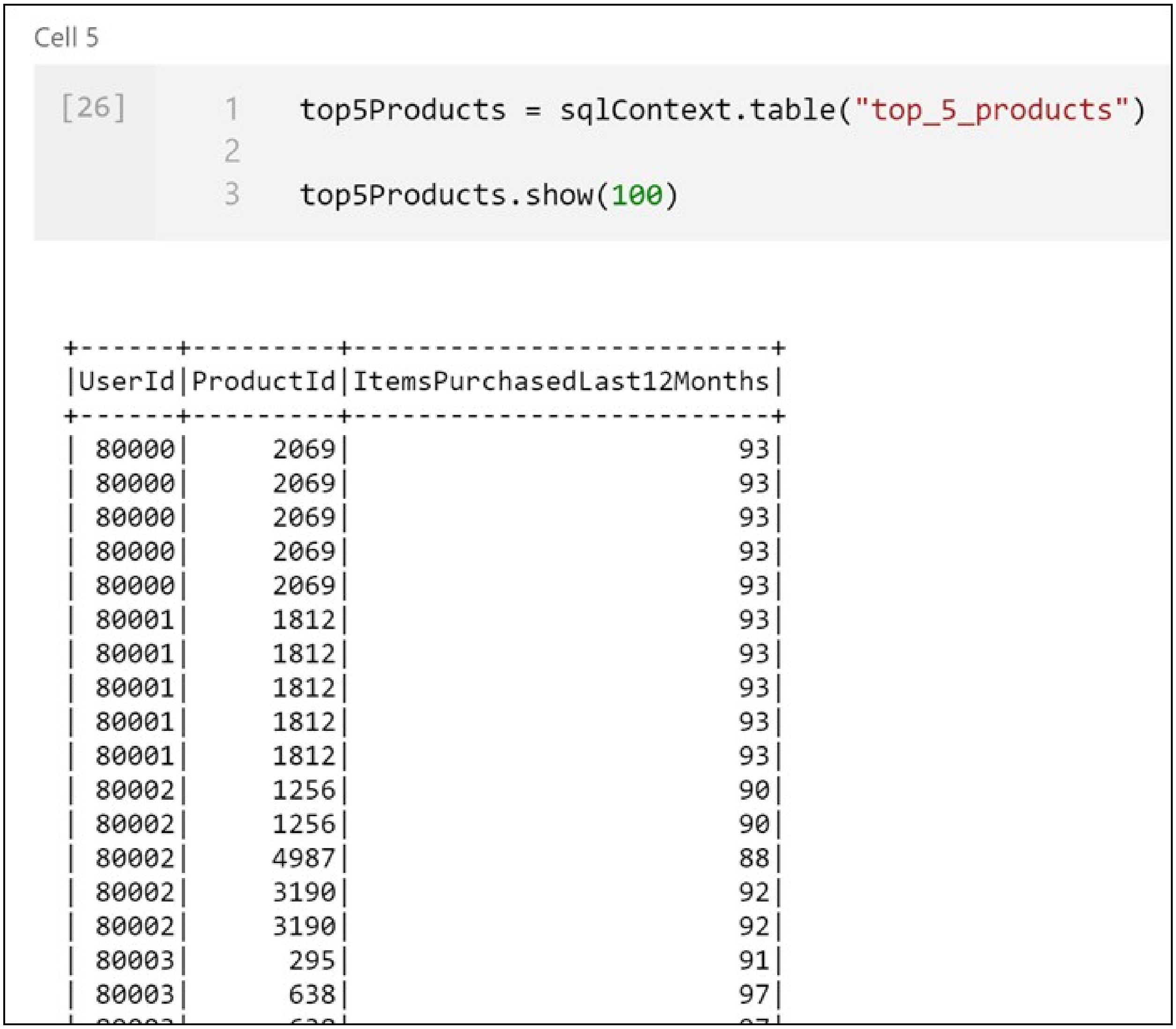
```
%%sql

CREATE OR REPLACE TEMPORARY VIEW top_5_products
```

```
AS
    select UserId, ProductId, ItemsPurchasedLast12Months
    from (select *,
                row_number() over (partition by UserId order by ItemsPurchasedLa
st12Months desc) as seqnum
        from top_purchases
        ) a
    where seqnum <= 5 and IsTopProduct == true and IsPreferredProduct = true
    order by a.UserId
```

Note that there is no output for the above query. **The query uses the *top_purchases* temporary view as a source and applies a *row_number()* over method to apply a row number for the records for each user where *ItemsPurchasedLast12Months* is greatest. The *where* clause filters the results so we only retrieve up to five products where both *IsTopProduct* and *IsPreferredProduct* are set to true. This gives us the top five most purchased products for each user where those products are *also* identified as their favorite products, according to their user profile stored in Azure Cosmos DB. 9. Execute the following in a new cell to create and display a new DataFrame that stores the results of the *top_5_products* temporary view you created in the previous cell:**

```
top5Products = sqlContext.table("top_5_products")

top5Products.show(100)
```

You should see an output similar to the following, which displays the top five preferred products per user:



Cell 5

[26]

```
top5Products = sqlContext.table("top_5_products")

top5Products.show(100)
```

```
+------+---------+--------------------------+
|UserId|ProductId|ItemsPurchasedLast12Months|
+------+---------+--------------------------+
| 80000|     2069|                        93|
| 80000|     2069|                        93|
| 80000|     2069|                        93|
| 80000|     2069|                        93|
| 80000|     2069|                        93|
| 80001|     1812|                        93|
| 80001|     1812|                        93|
| 80001|     1812|                        93|
| 80001|     1812|                        93|
| 80001|     1812|                        93|
| 80002|     1256|                        90|
| 80002|     1256|                        90|
| 80002|     4987|                        88|
| 80002|     3190|                        92|
| 80002|     3190|                        92|
| 80003|      295|                        91|
| 80003|      638|                        97|
```

The top five preferred products are displayed per user.

10. Calculate the top five products overall, based on those that are both preferred by customers and purchased the most. To do this, execute the following in a new cell:

```
top5ProductsOverall = (top5Products.select("ProductId","ItemsPurchasedLast12Months")
    .groupBy("ProductId")
    .agg( sum("ItemsPurchasedLast12Months").alias("Total") )
    .orderBy( col("Total").desc() )
    .limit(5))

top5ProductsOverall.show()
```

In this cell, we grouped the top five preferred products by product ID, summed up the total items purchased in the last 12 months, sorted that value in descending order, and returned the top five results. Your output should be similar to the following:

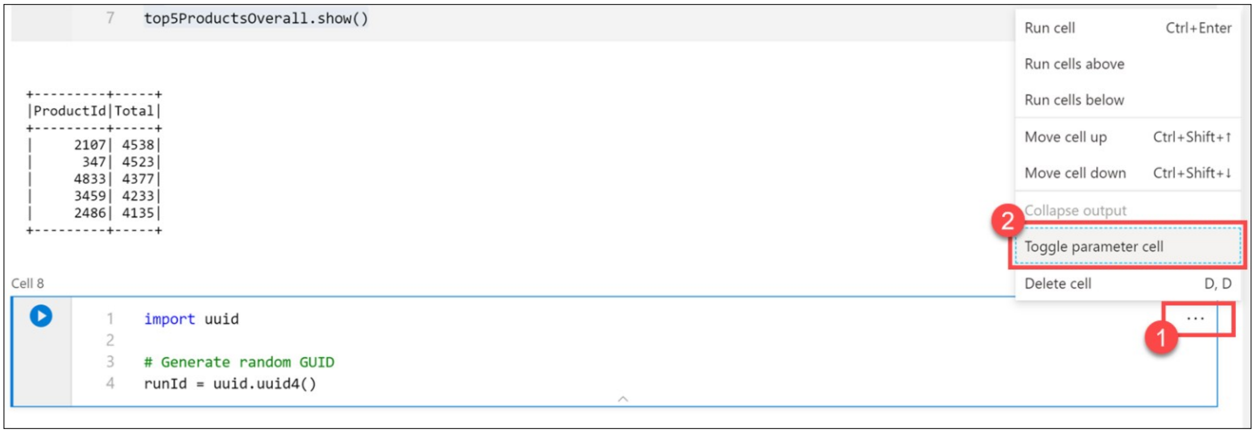
```
+---------+-----+
|ProductId|Total|
+---------+-----+
|     2107| 4538|
|     4833| 4533|
|      347| 4523|
|     3459| 4233|
|     4246| 4155|
+---------+-----+
```

Create a parameter cell

Azure Synapse pipelines look for the parameters cell and treat this cell as defaults for the parameters passed in at execution time. The execution engine will add a new cell beneath the parameters cell with input parameters in order to overwrite the default values. When a parameters cell isn't designated, the injected cell will be inserted at the top of the notebook. 1. We are going to execute this notebook from a pipeline. We want to pass in a parameter that sets a *runId* variable value that will be used to name the Parquet file. Execute the following in a new cell:

```
import uuid

# Generate random GUID
runId = uuid.uuid4()
```

We are using the *uuid* library that comes with Spark to generate a random GUID. We want to override the *runId* variable with a parameter passed in by the pipeline. To do this, we need to toggle this as a parameter cell. 2. Select the actions ellipses **(...)** on the top-right corner of the cell **(1)**, then select **Toggle parameter cell (2)**.



```
top5ProductsOverall.show()
```

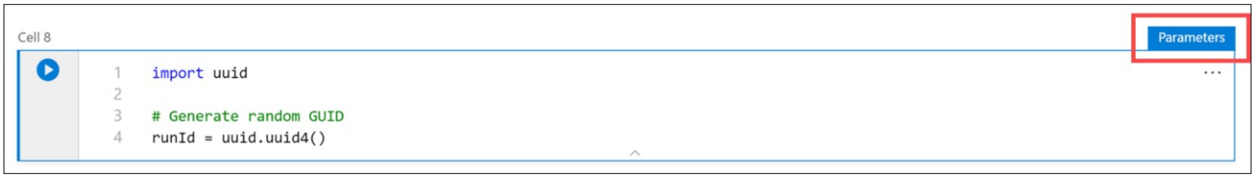
```
+---------+-----+
|ProductId|Total|
+---------+-----+
|     2107| 4538|
|      347| 4523|
|     4833| 4377|
|     3459| 4233|
|     2486| 4135|
+---------+-----+
```

Cell 8

```
import uuid

# Generate random GUID
runId = uuid.uuid4()
```

The menu item is highlighted.

After toggling this option, you will see the **Parameters** tag on the cell



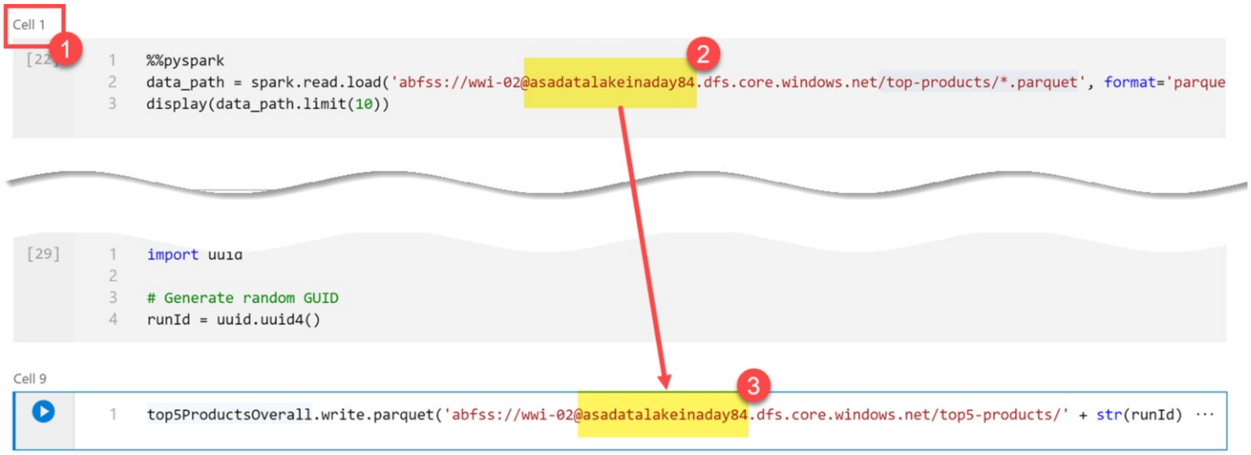
Cell 8

```
import uuid

# Generate random GUID
runId = uuid.uuid4()
```

The cell is configured to accept parameters.
3. Paste the following code in a new cell to use the *runId* variable as the Parquet filename in the */top5-products/ p*ath in the primary data lake account. **Replace** YOUR DATALAKE NAME in the path with the name of your primary data lake account. To find this, scroll up to **Cell 1** at the top of the page **(1)**. Copy the data lake storage account from the path **(2)**. Paste this value as a replacement for YOUR DATALAKE NAME in the path **(3)** inside the new cell, then execute the cell

```
%%pyspark
```

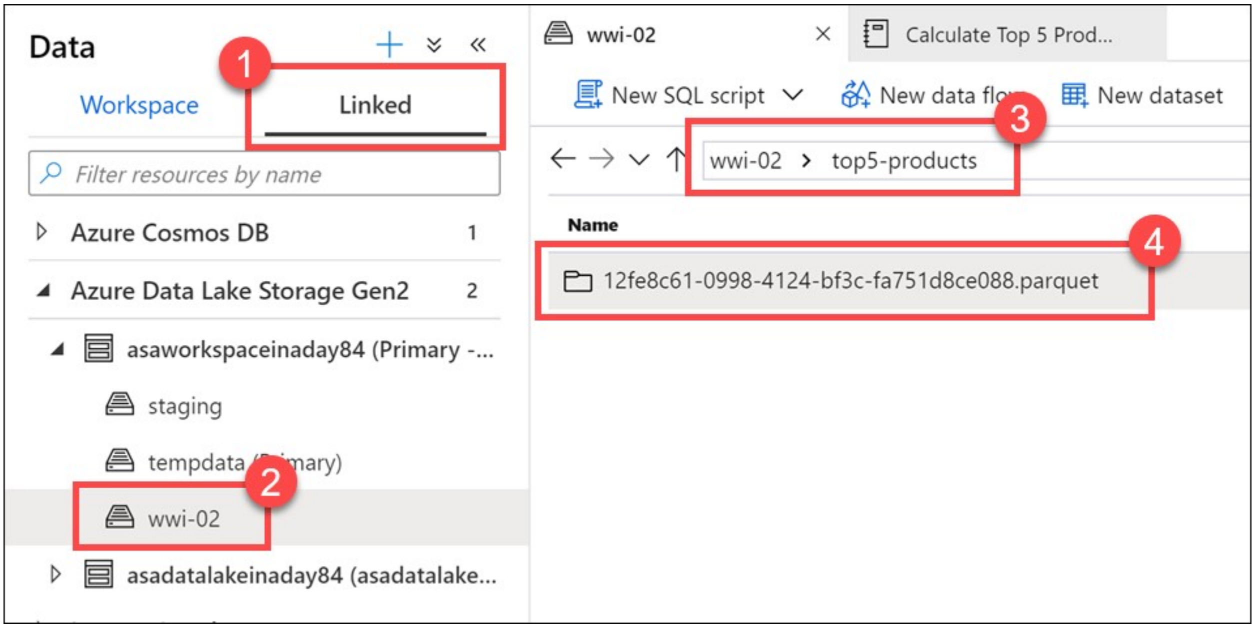
```
top5ProductsOverall.write.parquet('abfss://wwi-02@YOURDATALAKE_NAME.dfs.core.windows.net/top5-products/' + str(runId) + '.parquet')
```



```
Cell 1
[22]
%%pyspark
data_path = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet', format='parque
display(data_path.limit(10))

[29]
import uuid

# Generate random GUID
runId = uuid.uuid4()

Cell 9
top5ProductsOverall.write.parquet('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top5-products/' + str(runId) ···
```

The path is updated with the name of the primary data lake account

4. Verify that the file was written to the data lake. Navigate to the **Data** hub and select the **Linked** tab **(1)**. Expand the primary data lake storage account and select the **wwi-02** container **(2)**. Navigate to the **top5-products** folder **(3)**. You should see a folder for the Parquet file in the directory with a GUID as the file name **(4)**.

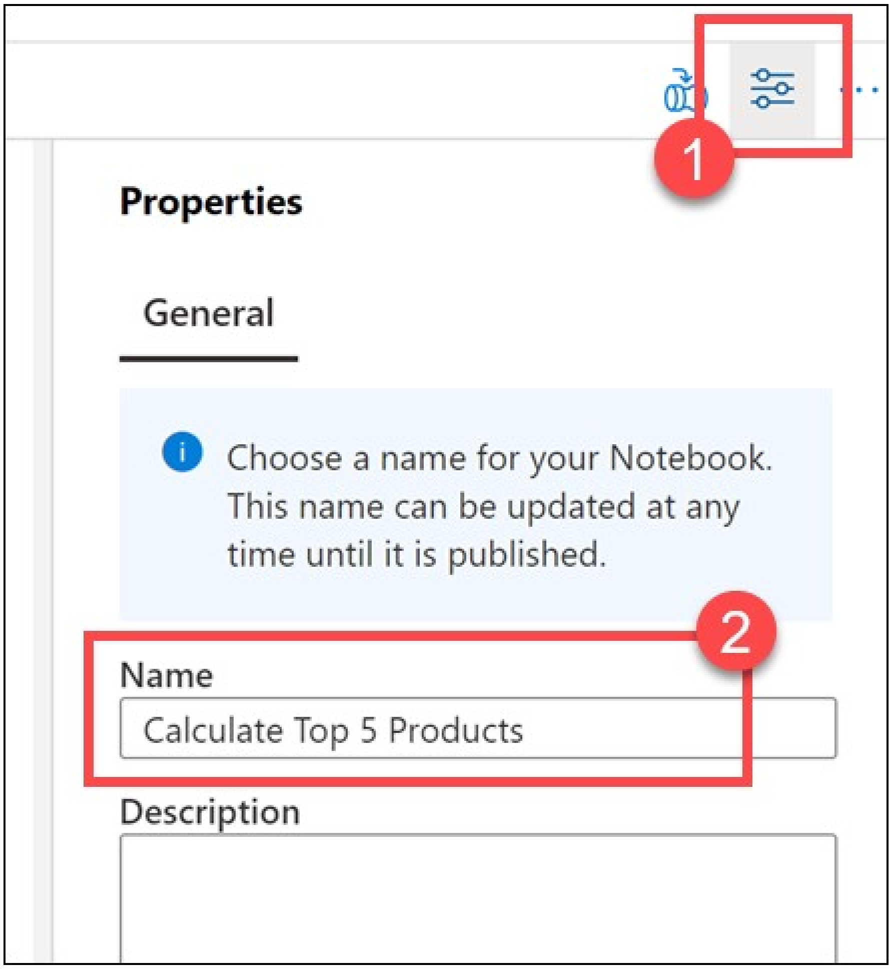


The parquet file is highlighted.

The Parquet write method on the dataframe in the Notebook cell created this directory since it did not previously exist. Add the Notebook to a Synapse pipeline

Referring back to the Mapping Data Flow we mentioned at the beginning of the exercise, suppose you want to execute this notebook after the Data Flow runs as part of your orchestration process. To do this, you add this notebook to a pipeline as a new Notebook activity.

1. Return to the notebook. Select the Properties button (1) at the top-right corner of the notebook, then enter *Calculate Top 5 Products* for the Name (2).



Properties

General

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

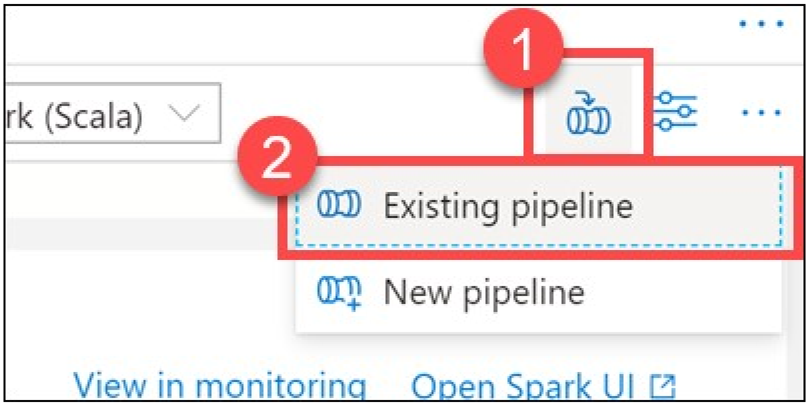
Name

Calculate Top 5 Products

Description

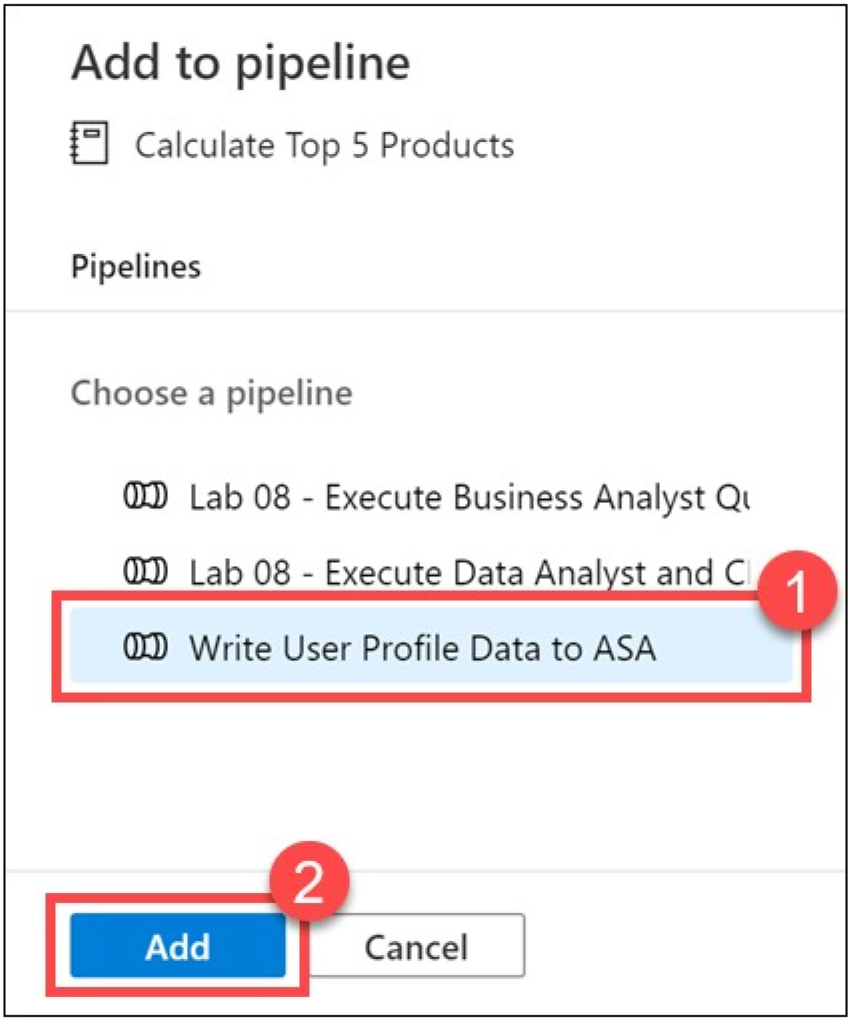
The properties blade is displayed.

2. Select the Add to pipeline button (1) at the top-right corner of the notebook, then select Existing pipeline (2)



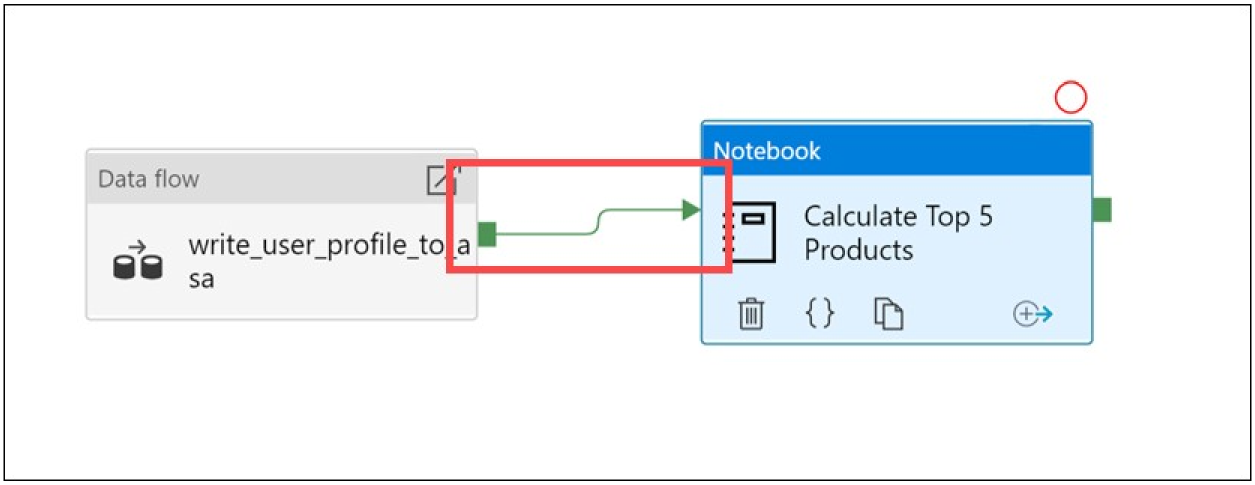
The add to pipeline button is highlighted.

3. Select the **Write User Profile Data to ASA** pipeline (1), then select **Add** *2).


Add to pipeline
Calculate Top 5 Products
Pipelines
Choose a pipeline
Lab 08 - Execute Business Analyst Qu
Lab 08 - Execute Data Analyst and Cl
Write User Profile Data to ASA
1
2
Add
Cancel

The pipeline is selected.

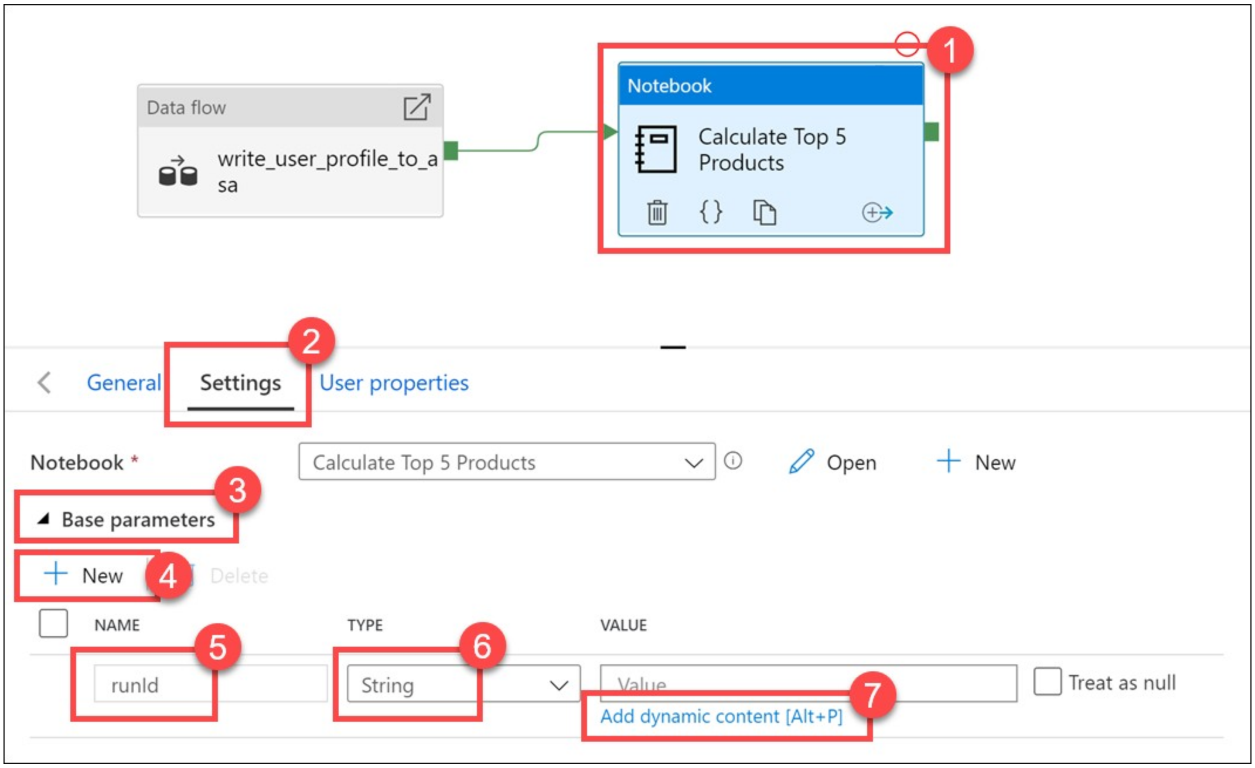
4. Synapse Studio adds the Notebook activity to the pipeline. Rearrange the Notebook activity so it sits to the right of the Data flow activity. Select the Data flow activity and drag a Success activity pipeline connection green box to the Notebook activity.



The green arrow is highlighted.

The Success activity arrow instructs the pipeline to execute the Notebook activity after the Data flow activity successfully runs.

5. Select the Notebook activity (1), select the Settings tab (2), expand Base parameters (3), and select + New (4). Enter *runId* in the Name field (5). Select String for the Type (6). For the Value, select Add dynamic content (7)



Notebook * : Calculate Top 5 Products

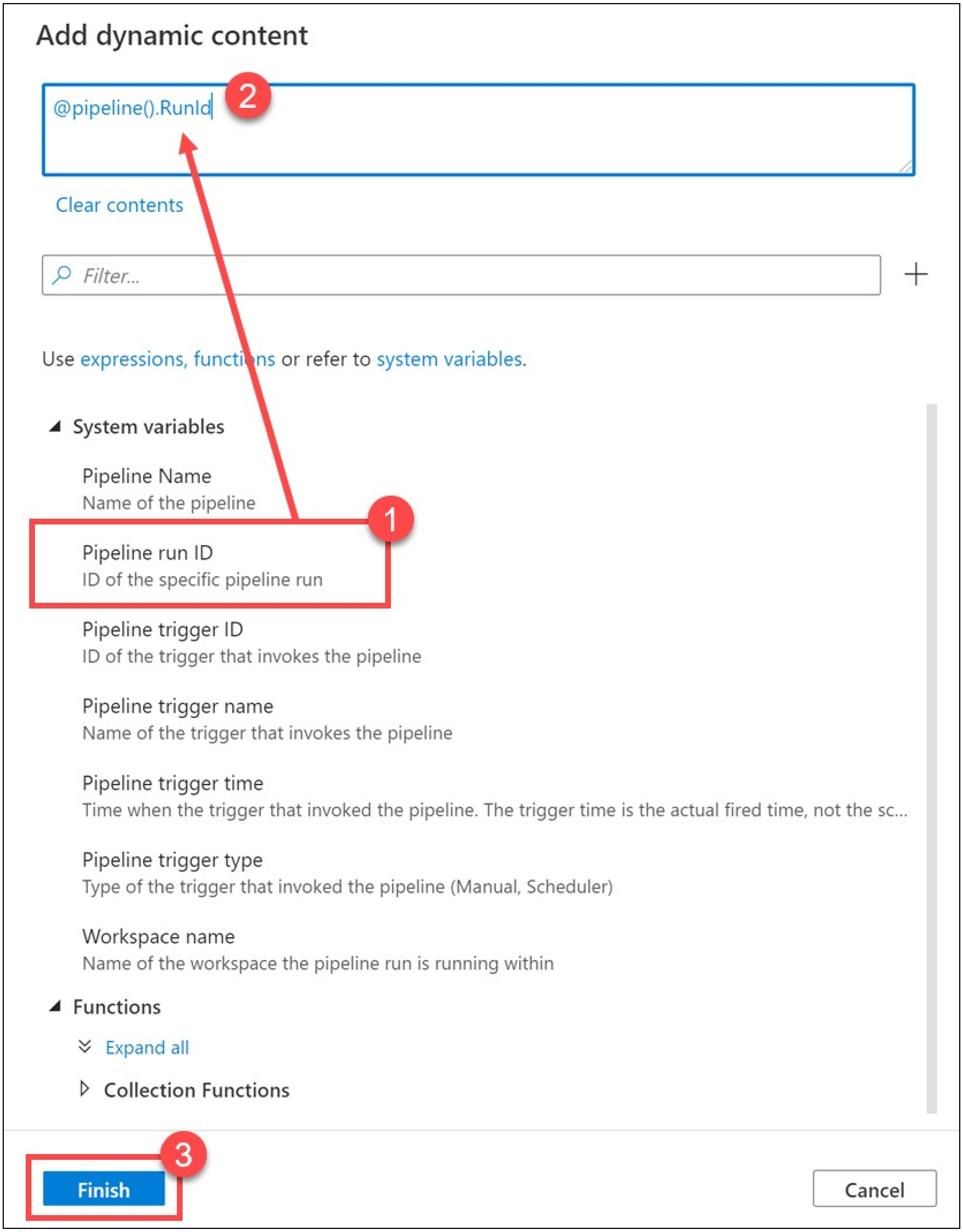
Base parameters

+ New

NAME	TYPE	VALUE
runId	String	Value

The settings are displayed.

6. Select **Pipeline run ID** under **System variables (1)**. This adds *@pipeline().RunId* to the dynamic content box **(2)**. Select **Finish (3)** to close the dialog.



Add dynamic content

@pipeline().RunId

Use expressions, functions or refer to system variables.

◢ System variables

Pipeline Name
Name of the pipeline

Pipeline run ID
ID of the specific pipeline run

Pipeline trigger ID
ID of the trigger that invokes the pipeline

Pipeline trigger name
Name of the trigger that invokes the pipeline

Pipeline trigger time
Time when the trigger that invoked the pipeline. The trigger time is the actual fired time, not the sc...

Pipeline trigger type
Type of the trigger that invoked the pipeline (Manual, Scheduler)

Workspace name
Name of the workspace the pipeline run is running within

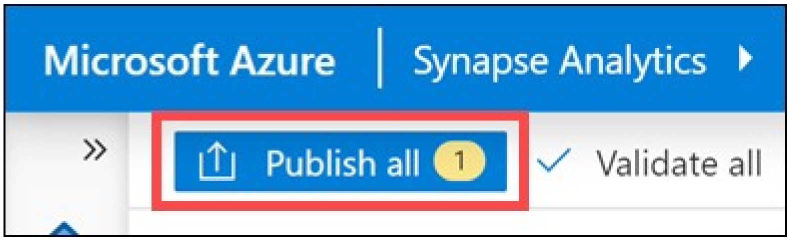
◢ Functions

▷ Collection Functions

The dynamic content form is displayed.

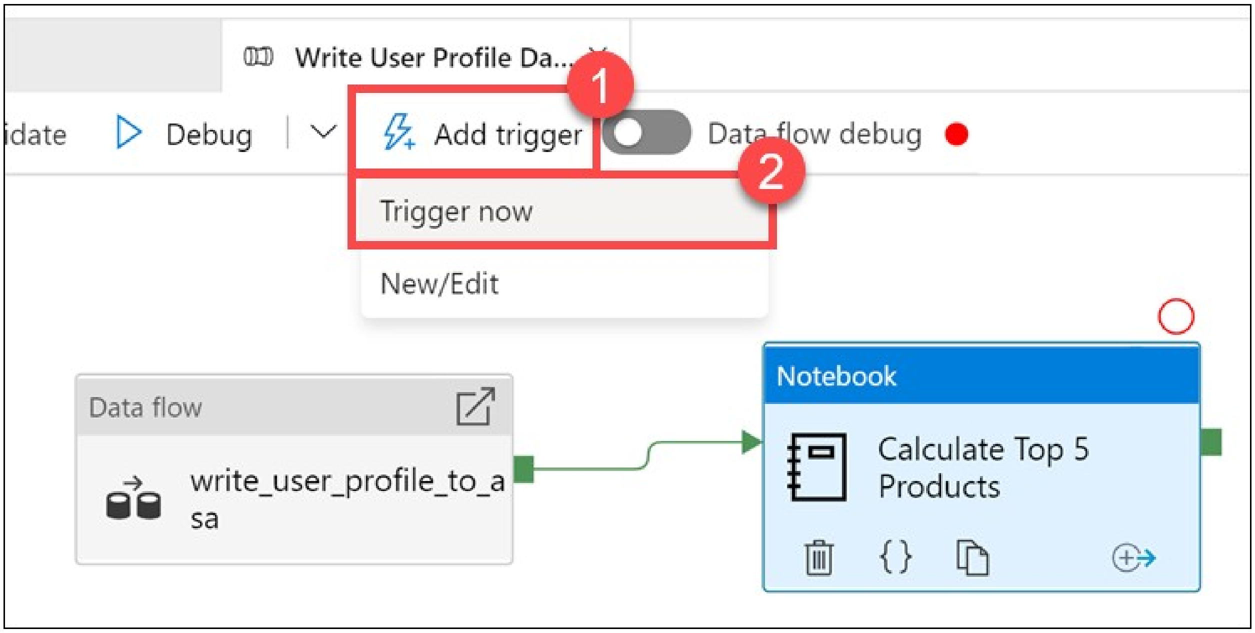
The Pipeline run ID value is a unique GUID assigned to each pipeline run. We will use this value for the name of the Parquet file by passing this value in as the *runId* Notebook parameter. We can then look through the pipeline run history and find the specific Parquet file created for each pipeline run.

7. Select Publish all then Publish to save your changes



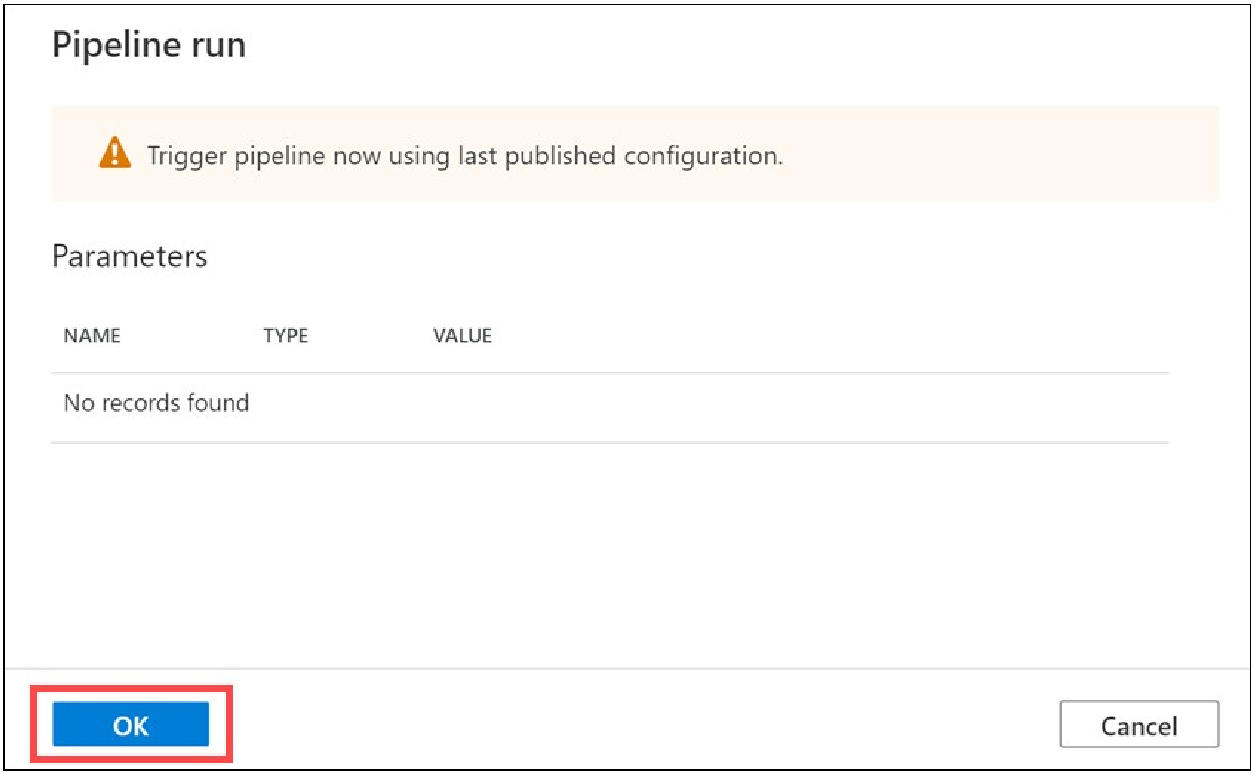
Publish all is highlighted.

8. After publishing is complete, select Add trigger (1), then Trigger now (2) to run the updated pipeline.



The trigger menu item is highlighted.

9. Select OK to run the trigger.



Pipeline run

⚠ Trigger pipeline now using last published configuration.

Parameters

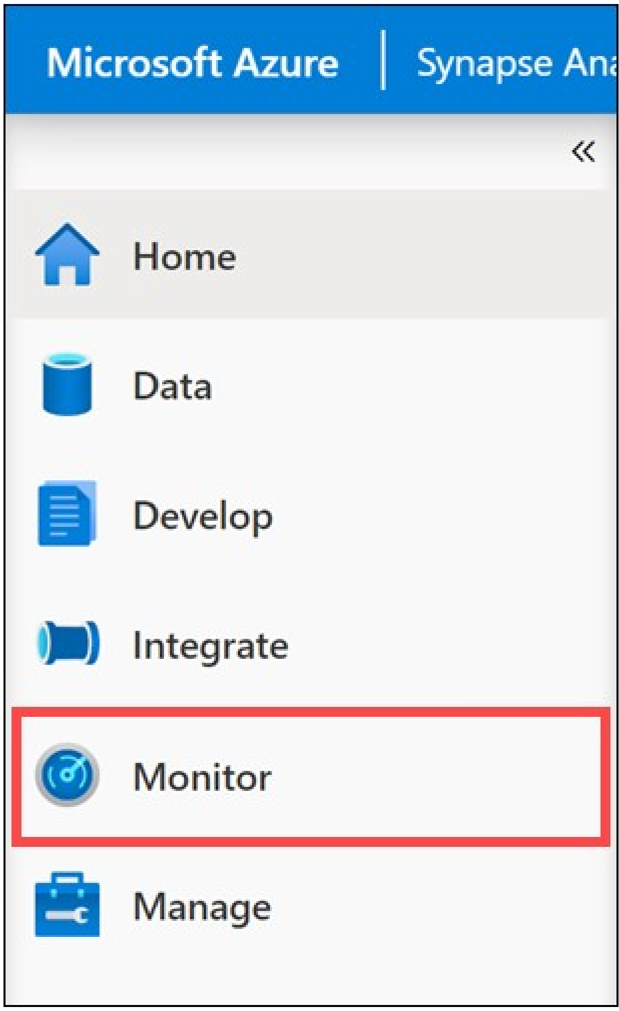
NAME	TYPE	VALUE
No records found		

The OK button is highlighted.

Monitor the pipeline run

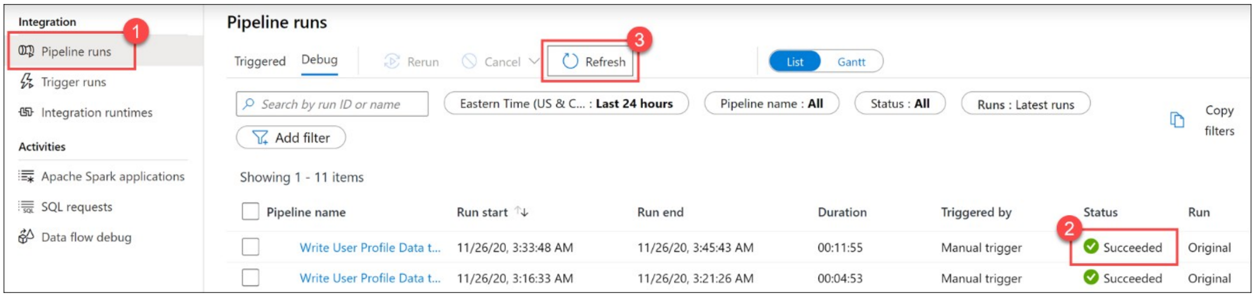
The Monitor hub lets you monitor current and historical activities for SQL, Apache Spark, and Pipelines.

1. Navigate to the Monitor hub.



The Monitor hub menu item is selected.

2. Select **Pipeline runs (1)** and wait for the pipeline run to successfully complete **(2)**. You may need to refresh **(3)** the view.

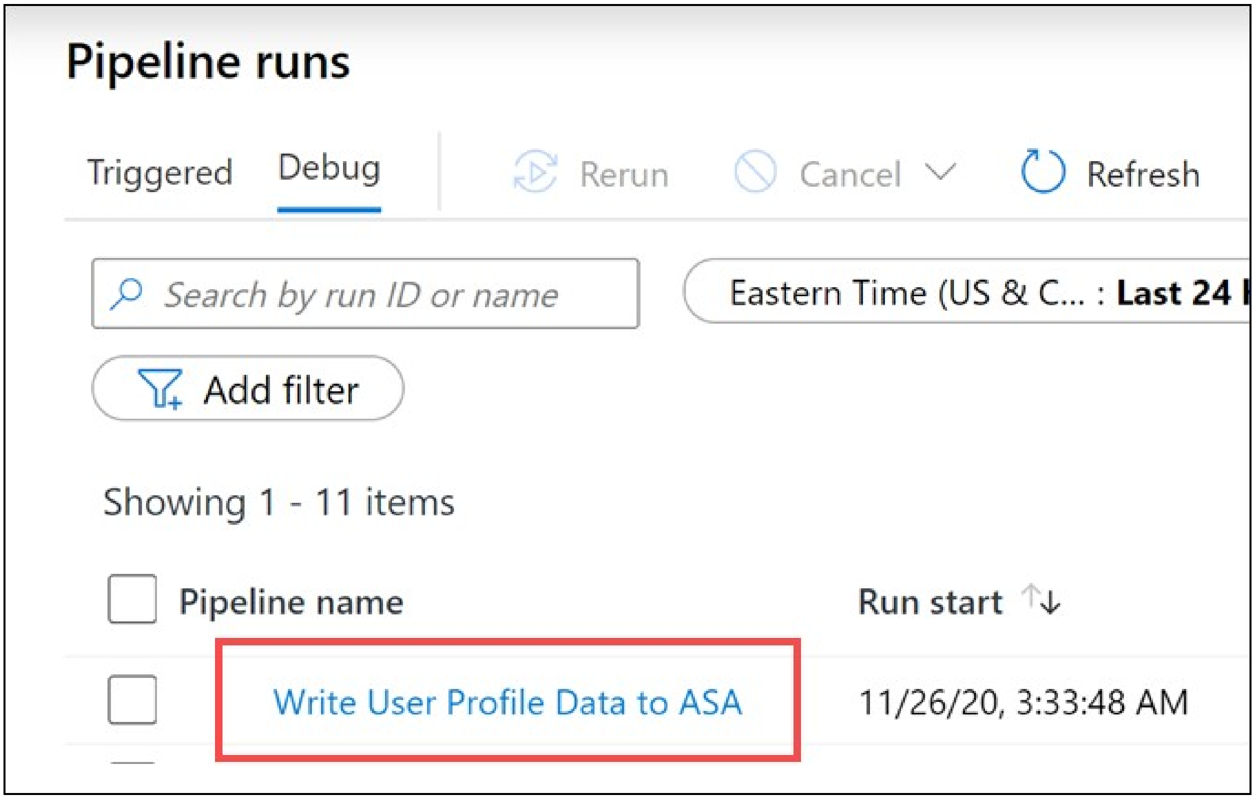


Showing 1 - 11 items

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run
Write User Profile Data t...	11/26/20, 3:33:48 AM	11/26/20, 3:45:43 AM	00:11:55	Manual trigger	✓ Succeeded	Original
Write User Profile Data t...	11/26/20, 3:16:33 AM	11/26/20, 3:21:26 AM	00:04:53	Manual trigger	✓ Succeeded	Original

The pipeline run succeeded.

3. Select the name of the pipeline to view the pipeline's activity runs.



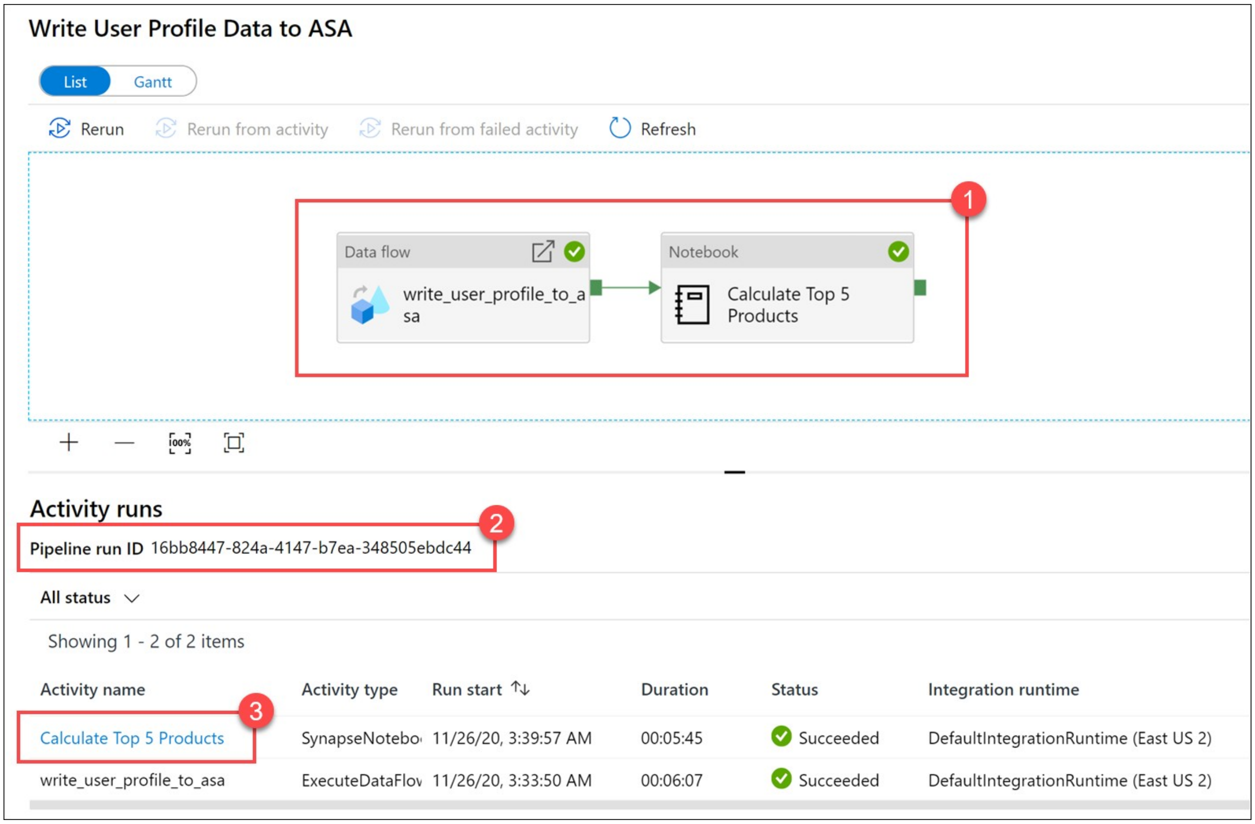
Pipeline runs

Showing 1 - 11 items

□	Pipeline name	Run start
□	Write User Profile Data to ASA	11/26/20, 3:33:48 AM

The pipeline name is selected.

4. Notice both the **Data flow** activity, and the new **Notebook** activity **(1)**. Make note of the **Pipeline run ID** value **(2)**. We will compare this to the Parquet file name generated by the notebook. Select the **Calculate Top 5 Products** Notebook name to view its details **(3)**.



Write User Profile Data to ASA

Activity runs

Pipeline run ID 16bb8447-824a-4147-b7ea-348505ebdc44

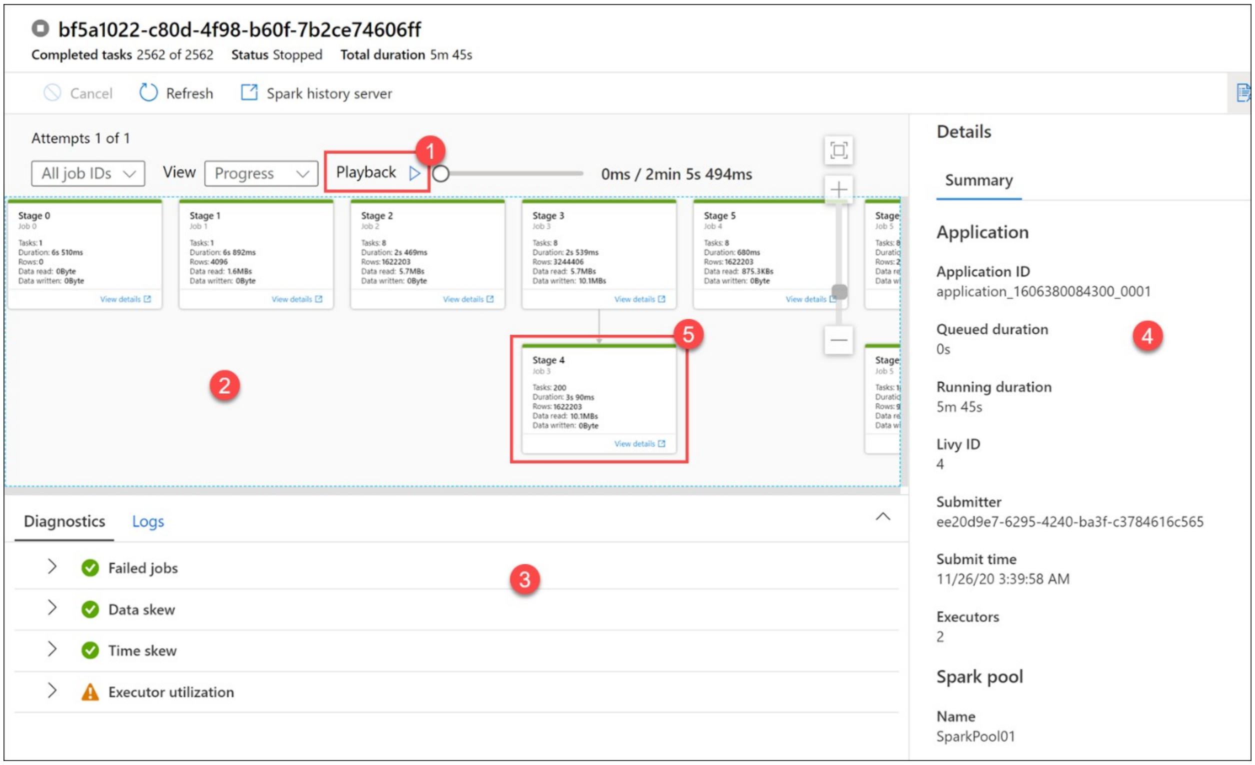
All status

Showing 1 - 2 of 2 items

Activity name	Activity type	Run start ↑↓	Duration	Status	Integration runtime
Calculate Top 5 Products	SynapseNotebo	11/26/20, 3:39:57 AM	00:05:45	✓ Succeeded	DefaultIntegrationRuntime (East US 2)
write_user_profile_to_asa	ExecuteDataFlov	11/26/20, 3:33:50 AM	00:06:07	✓ Succeeded	DefaultIntegrationRuntime (East US 2)

The pipeline run details are displayed.

5. Here we see the Notebook run details. You can select the Playback button (1) to watch a playback of the progress through the jobs (2). At the bottom, you can view the Diagnostics and Logs with different filter options (3). To the right, we can view the run details, such as the duration, Livy ID, Spark pool details, etc. Select the View details link on a job to view its details (5).



bf5a1022-c80d-4f98-b60f-7b2ce74606ff

Completed tasks 2562 of 2562 Status Stopped Total duration 5m 45s

Stage 0
Job 0
Tasks: 1
Duration: 6s 510ms
Rows: 0
Data read: 0Byte
Data written: 0Byte

Stage 1
Job 1
Tasks: 1
Duration: 6s 892ms
Rows: 4096
Data read: 1.6MBs
Data written: 0Byte

Stage 2
Job 2
Tasks: 8
Duration: 2s 469ms
Rows: 1622203
Data read: 5.7MBs
Data written: 0Byte

Stage 3
Job 3
Tasks: 8
Duration: 2s 539ms
Rows: 3244406
Data read: 5.7MBs
Data written: 10.1MBs

Stage 5
Job 4
Tasks: 8
Duration: 680ms
Rows: 1622203
Data read: 875.3KBs
Data written: 0Byte

Stage 4
Job 3
Tasks: 200
Duration: 3s 90ms
Rows: 1622203
Data read: 10.1MBs
Data written: 0Byte

Diagnostics Logs

- Failed jobs
- Data skew
- Time skew
- Executor utilization

Details

Summary

Application

Application ID
application_1606380084300_0001

Queued duration
0s

Running duration
5m 45s

Livy ID
4

Submitter
ee20d9e7-6295-4240-ba3f-c3784616c565

Submit time
11/26/20 3:39:58 AM

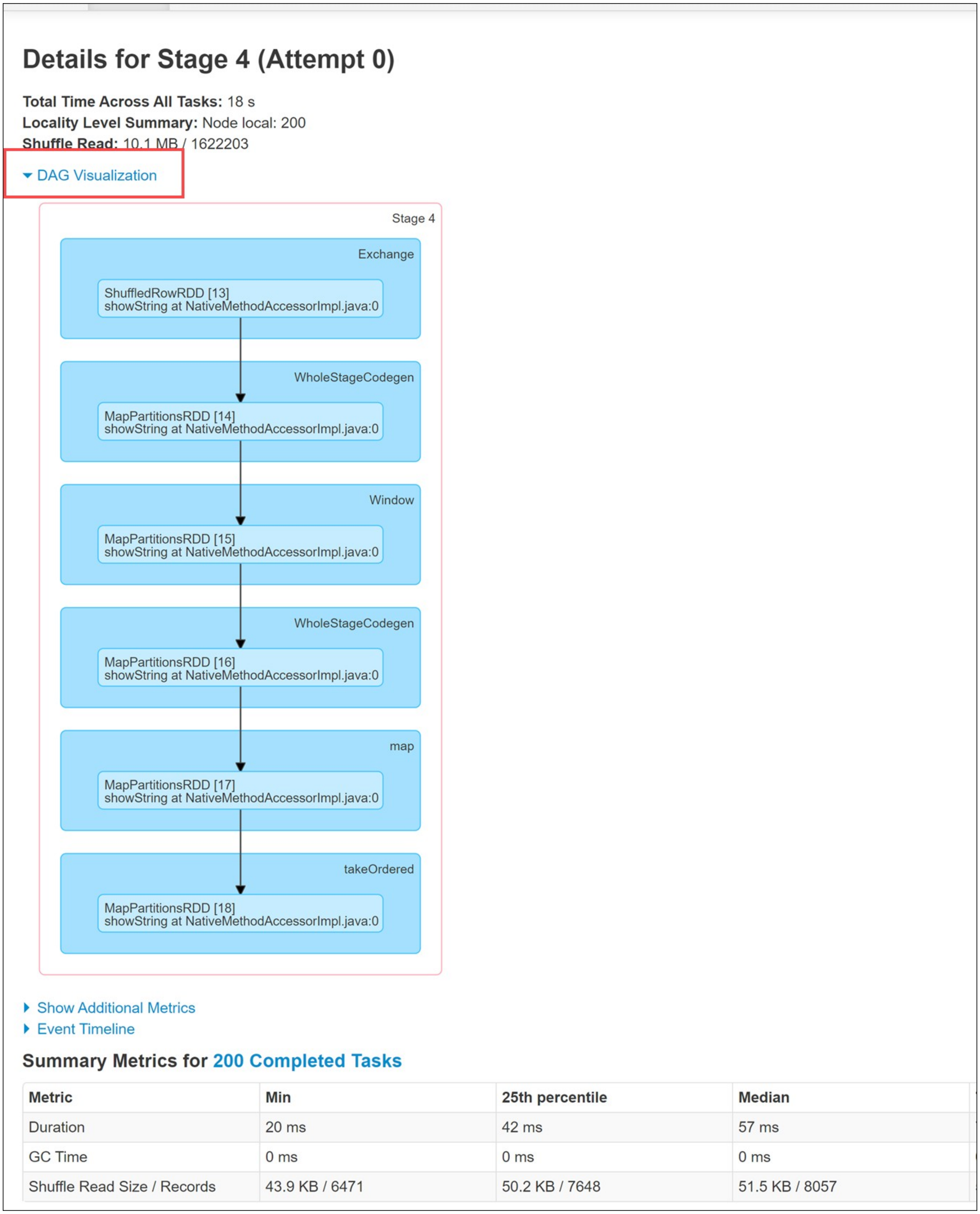
Executors
2

Spark pool

Name
SparkPool01

The run details are displayed.

6. The Spark application UI opens in a new tab where we can see the stage details. Expand the **DAG Visualization** to view the stage details.



Details for Stage 4 (Attempt 0)

Total Time Across All Tasks: 18 s
Locality Level Summary: Node local: 200
Shuffle Read: 10.1 MB / 1622203

▾ DAG Visualization

Stage 4
Exchange
ShuffledRowRDD [13]
showString at NativeMethodAccessorImpl.java:0
WholeStageCodegen
MapPartitionsRDD [14]
showString at NativeMethodAccessorImpl.java:0
Window
MapPartitionsRDD [15]
showString at NativeMethodAccessorImpl.java:0
WholeStageCodegen
MapPartitionsRDD [16]
showString at NativeMethodAccessorImpl.java:0
map
MapPartitionsRDD [17]
showString at NativeMethodAccessorImpl.java:0
takeOrdered
MapPartitionsRDD [18]
showString at NativeMethodAccessorImpl.java:0

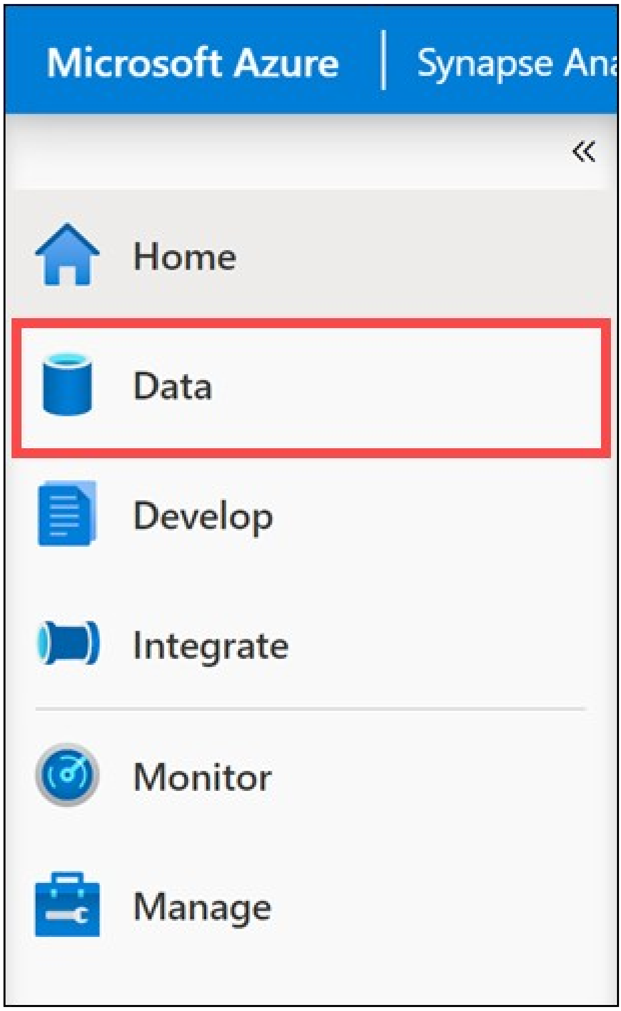
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median
Duration	20 ms	42 ms	57 ms
GC Time	0 ms	0 ms	0 ms
Shuffle Read Size / Records	43.9 KB / 6471	50.2 KB / 7648	51.5 KB / 8057

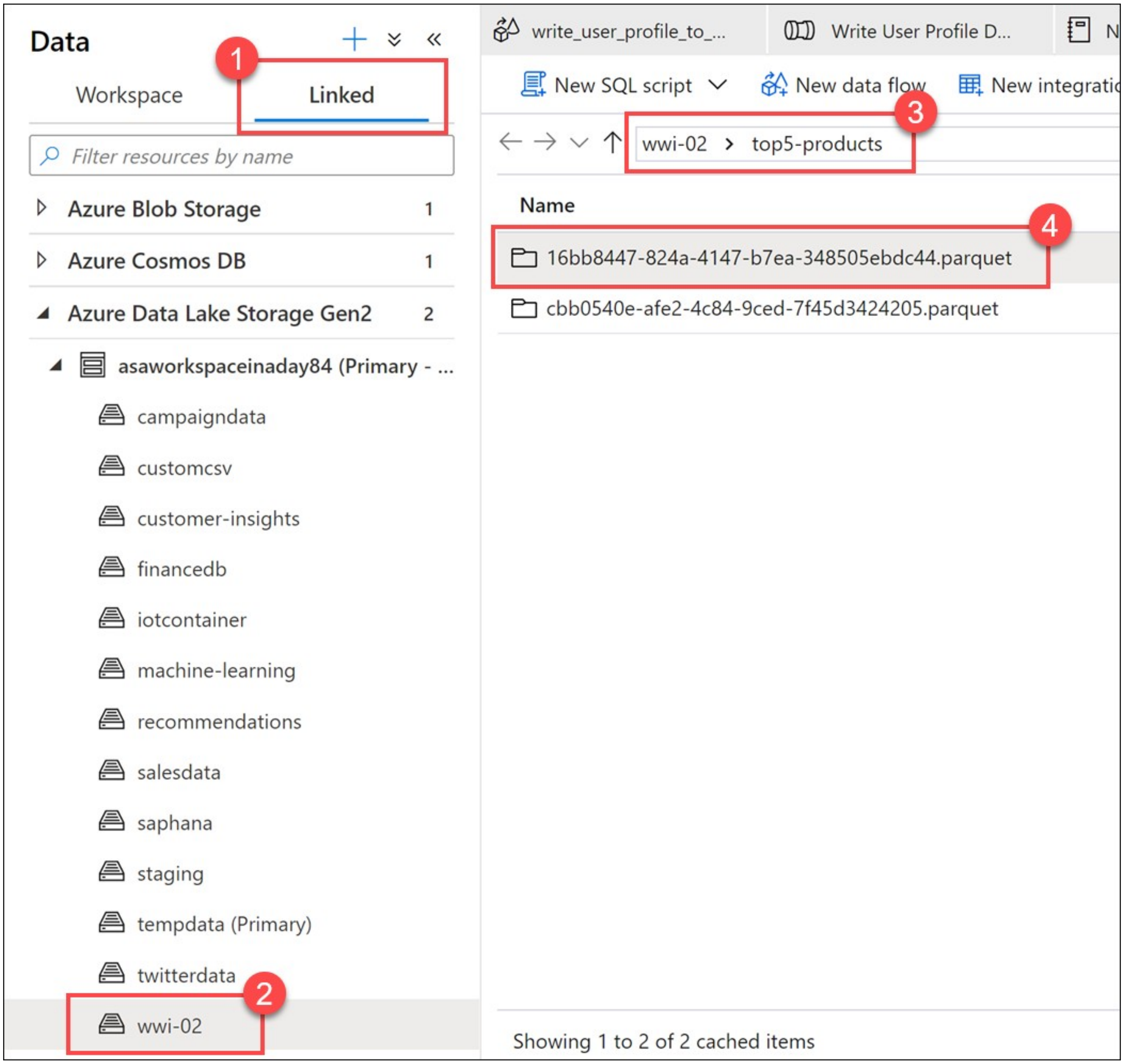
The Spark stage details are displayed.

7. Navigate back to the **Data** hub.



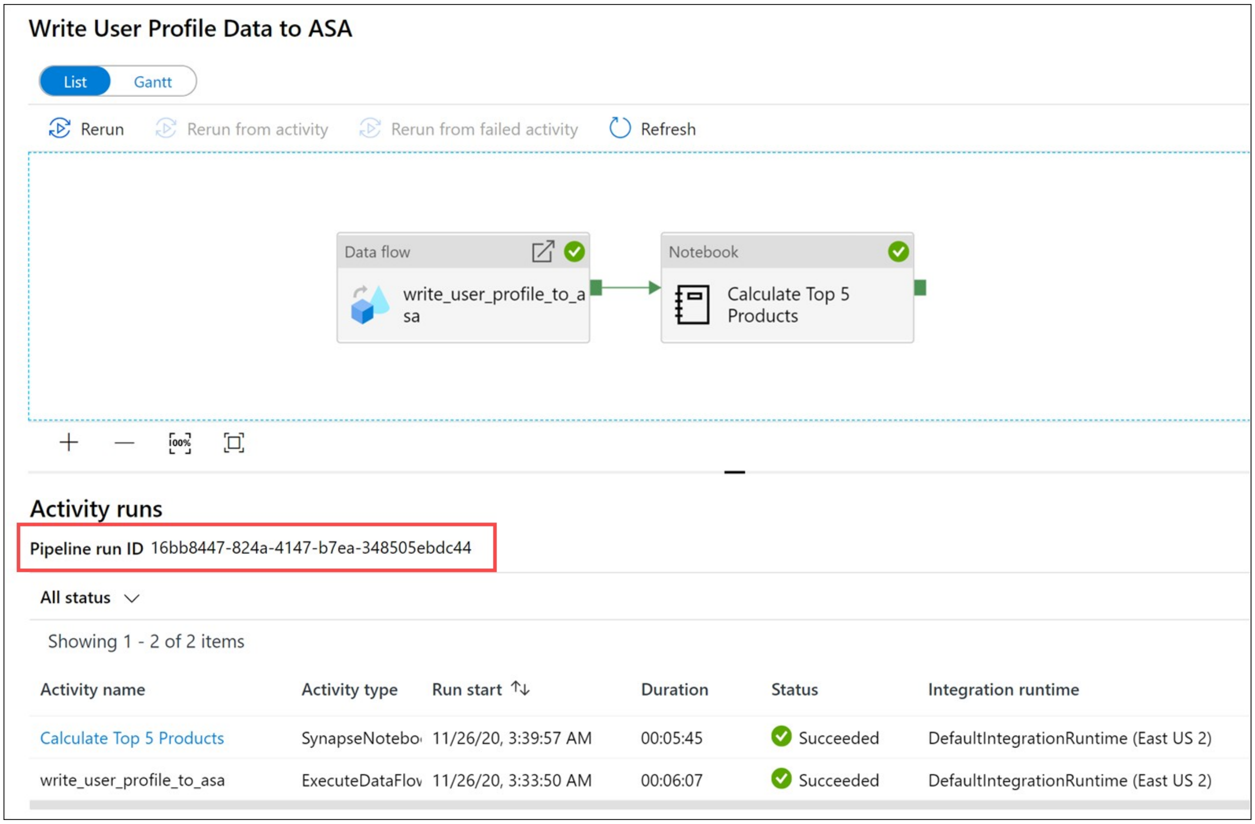
Data hub

8. Select the **Linked** tab **(1)**, select the **wwi-02** container **(2)** on the primary data lake storage account, navigate to the **top5-products** folder **(3)**, and verify that a folder exists for the Parquet file whose name matches the **Pipeline run ID**.



The file is highlighted.

As you can see, we have a file whose name matches the Pipeline run ID we noted earlier:


Write User Profile Data to ASA
List
Gantt
Rerun
Rerun from activity
Rerun from failed activity
Refresh
Data flow
write_user_profile_to_asa
Notebook
Calculate Top 5 Products

Activity runs

Pipeline run ID 16bb8447-824a-4147-b7ea-348505ebdc44

All status

Showing 1 - 2 of 2 items

Activity name	Activity type	Run start	Duration	Status	Integration runtime
Calculate Top 5 Products	SynapseNotebo	11/26/20, 3:39:57 AM	00:05:45	✓ Succeeded	DefaultIntegrationRuntime (East US 2)
write_user_profile_to_asa	ExecuteDataFlov	11/26/20, 3:33:50 AM	00:06:07	✓ Succeeded	DefaultIntegrationRuntime (East US 2)

The Pipeline run ID is highlighted.

These values match because we passed in the Pipeline run ID to the *runId* parameter on the Notebook activity.