

LAB05 : Use Azure Synapse Link for Cosmos DB

Azure Synapse Link for Azure Cosmos DB is a cloud-native *hybrid transactional analytical processing* (HTAP) technology that enables you to run near-real-time analytics over operational data stored in Azure Cosmos DB from Azure Synapse Analytics.

This lab will take approximately **30** minutes to complete.

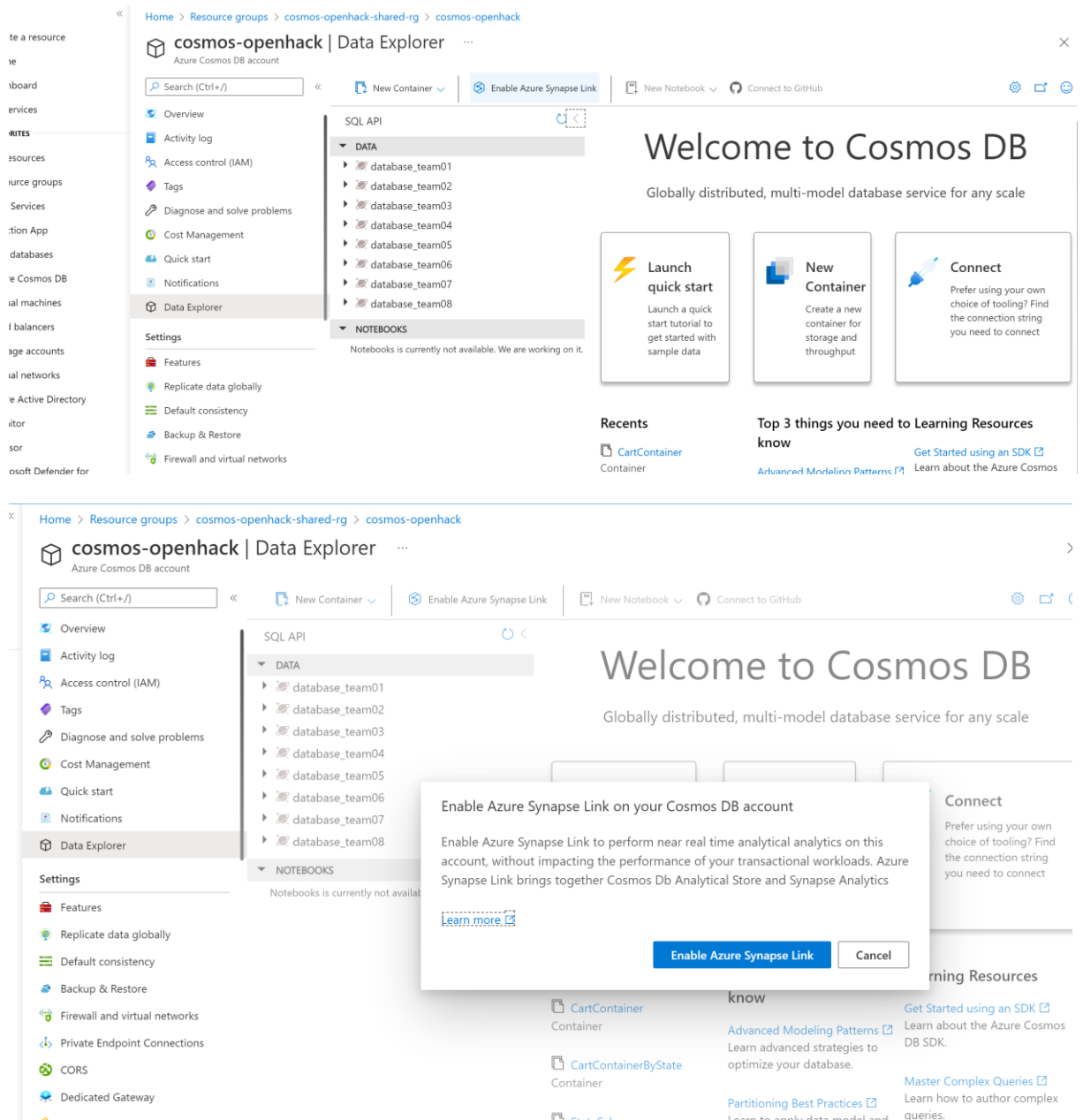
To explore Azure Synapse Link for Cosmos DB, you'll need an Azure Synapse Analytics workspace and an Azure Cosmos DB account.

Configure Synapse Link in Cosmos DB

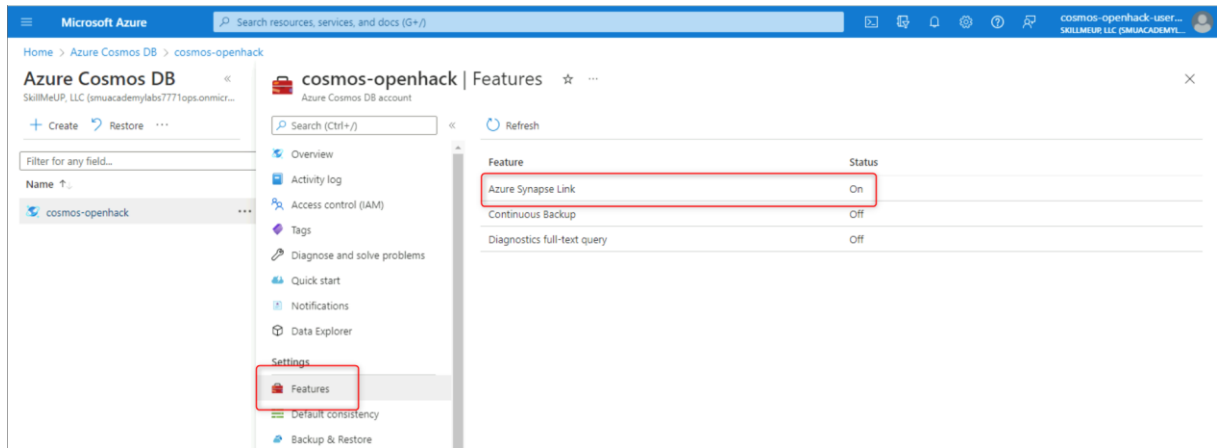
Before you can use Synapse Link for Cosmos DB, you must enable it in your Azure Cosmos DB account and configure a container as an analytical store.

Enable the Synapse Link feature in your Cosmos DB account

1. In the [Azure portal](#), browse to the **cosmos-openhack-shared-rg** resource group that was created by the setup script, and identify your **cosmos-openhack** Cosmos DB account.
2. Open your Azure Cosmos DB account, and select the **Data Explorer** page on the left side of its blade.
3. At the top of the **Data Explorer** page, use the **Enable Azure Synapse Link** button to enable Synapse Link. (in this workshop lab the synapse link will be already enabled as this take few minutes)

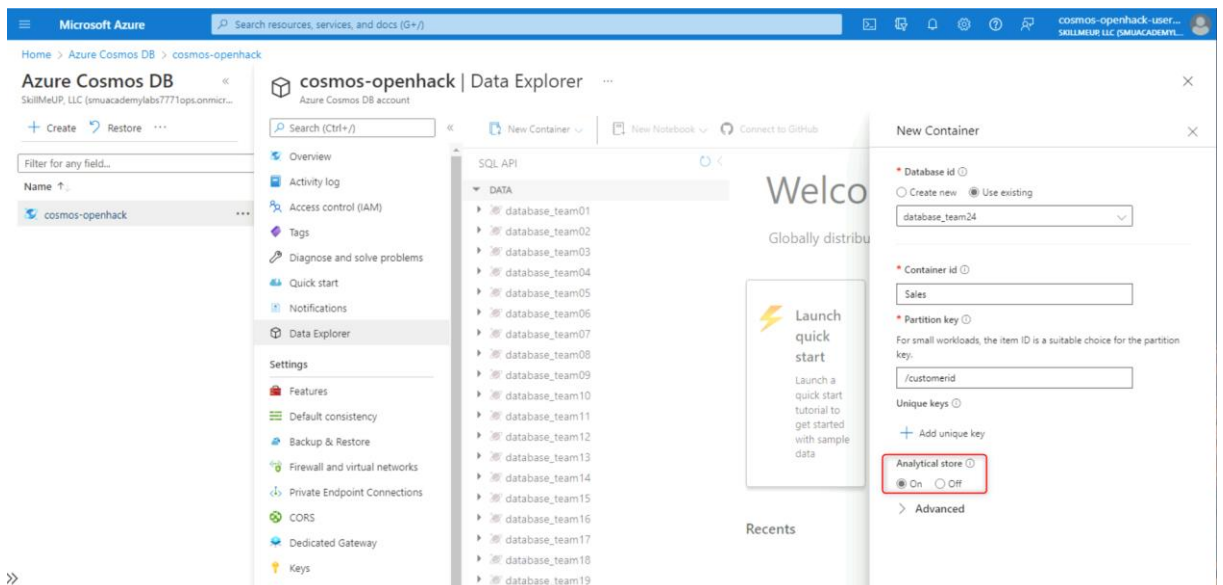


4. On the left side of the page, in the **Settings** section, select the **Features** page and verify the status of the **Azure Synapse Link** feature. If it is *Pending*, use the **Refresh** button at the top of the page to refresh the view until it is *On*.



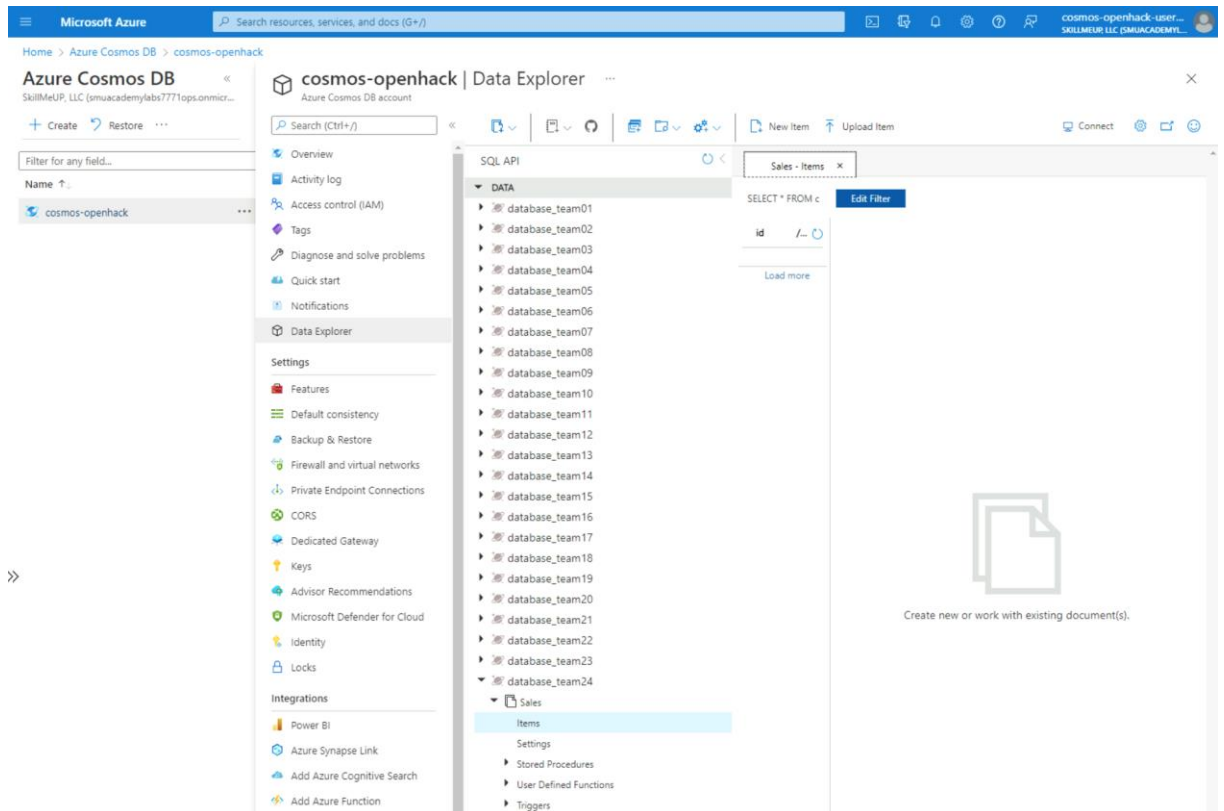
Create an analytical store container

- Return to the **Data Explorer** page, and use the **new Container** button (or tile) to create a new container with the following settings:
 - Database id:** (use existing) `database_teamxx`
 - Container id:** Sales
 - Partition key:** /customerid
 - Analytical store:** On



Note: In this scenario, **customerid** is used for partition key as it's likely to be used in many queries to retrieve customer and sales order information in a hypothetical application, it has relatively high cardinality (number of unique values), so it will allow the container to scale as the number of customers and sales orders grows.

- After the container has been created, in the **SQL API** pane, expand the **database_teamXX** database and its **Sales** folder; and then select the **Items** folder.



- Use the **New Item** button to create a new customer item based on the following JSON. Then save the new item (some additional metadata fields will be added when you save the item).

```
{
  "id": "S043701",
  "orderdate": "2019-07-01",
  "customerid": 123,
  "customerdetails": {
    "customername": "Christy Zhu",
    "customeremail": "christy12@adventure-works.com"
  },
  "product": "Mountain-100 Silver, 44",
  "quantity": 1,
  "price": 3399.99
}
```

- Add a second item with the following JSON:

```
{
  "id": "S043704",
  "orderdate": "2019-07-01",
  "customerid": 124,
  "customerdetails": {
    "customername": "Julio Ruiz",
    "customeremail": "julio1@adventure-works.com"
  }
}
```

```
},  
"product": "Mountain-100 Black, 48",  
"quantity": 1,  
"price": 3374.99  
}
```

5. Add a third item with the following JSON:

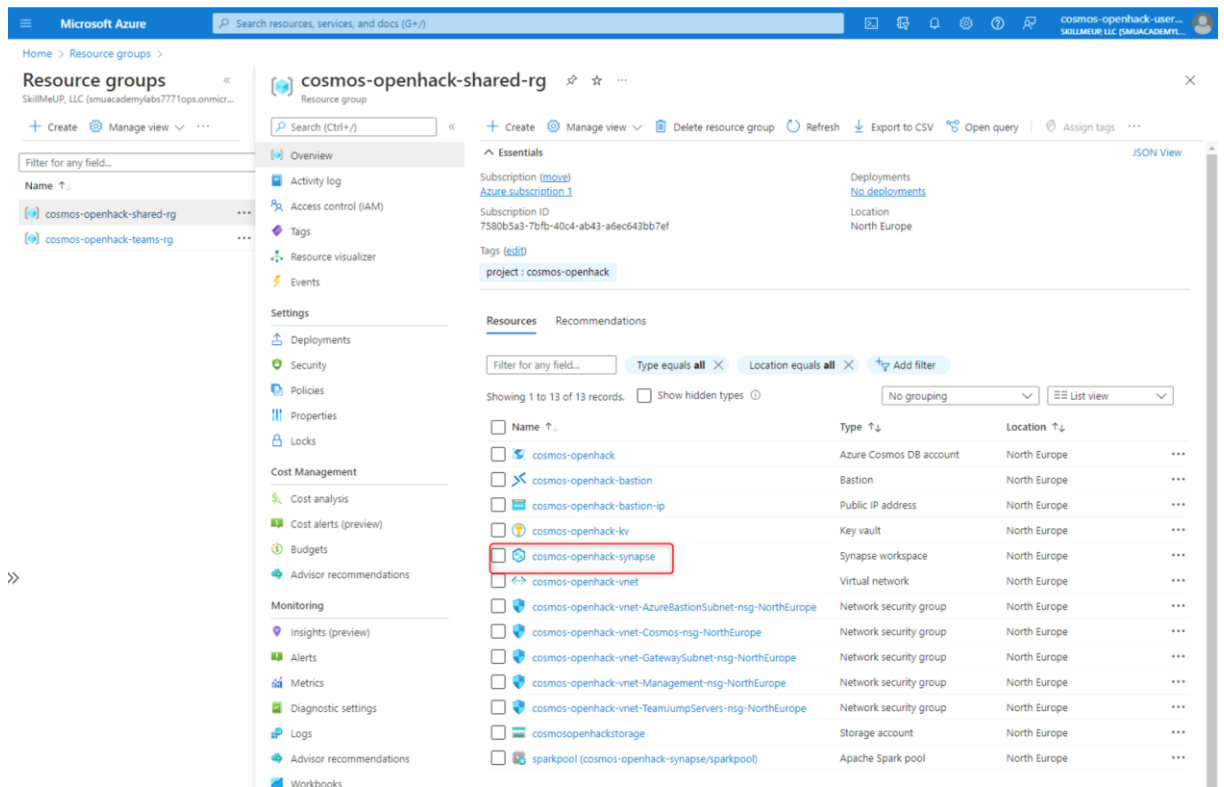
```
{  
  "id": "S043707",  
  "orderdate": "2019-07-02",  
  "customerid": 125,  
  "customerdetails": {  
    "customername": "Emma Brown",  
    "customeremail": "emma3@adventure-works.com"  
  },  
  "product": "Road-150 Red, 48",  
  "quantity": 1,  
  "price": 3578.27  
}
```

Note: In reality, the analytical store would contain a much larger volume of data, written to the store by an application. These few items will be sufficient to demonstrate the principle in this exercise.

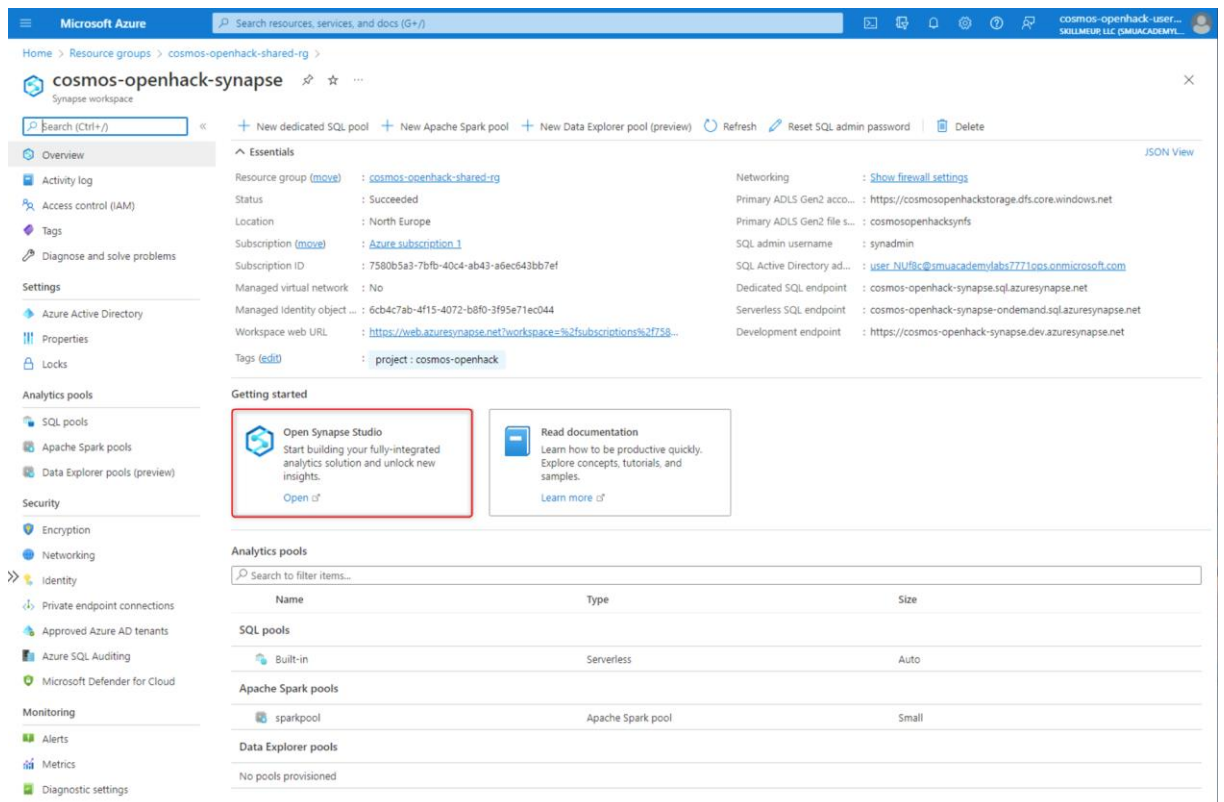
Configure Synapse Link in Azure Synapse Analytics

Now that you have prepared your Azure Cosmos DB account, you can configure Azure Synapse link for Cosmos DB in your Azure Synapse Analytics workspace.

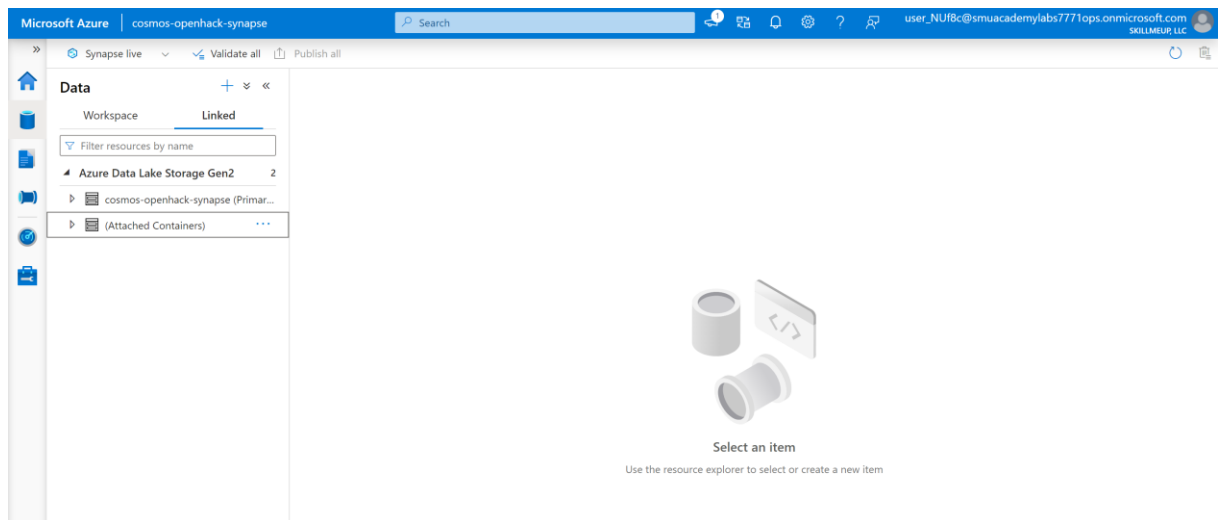
1. In the Azure portal, close the blade for your Cosmos DB account if it is still open, and return to the **cosmos-openhack-shared-rg** resource group.



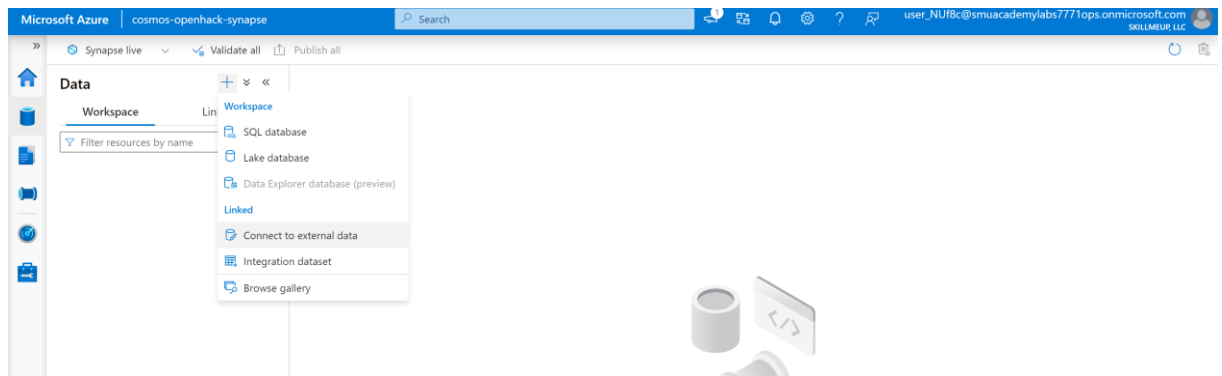
2. Open the **cosmos-openhack-synapse** Synapse workspace
3. On the **Overview** page, in the **Open Synapse Studio** card, select **Open** to open Synapse Studio in a new browser tab; signing in if prompted.

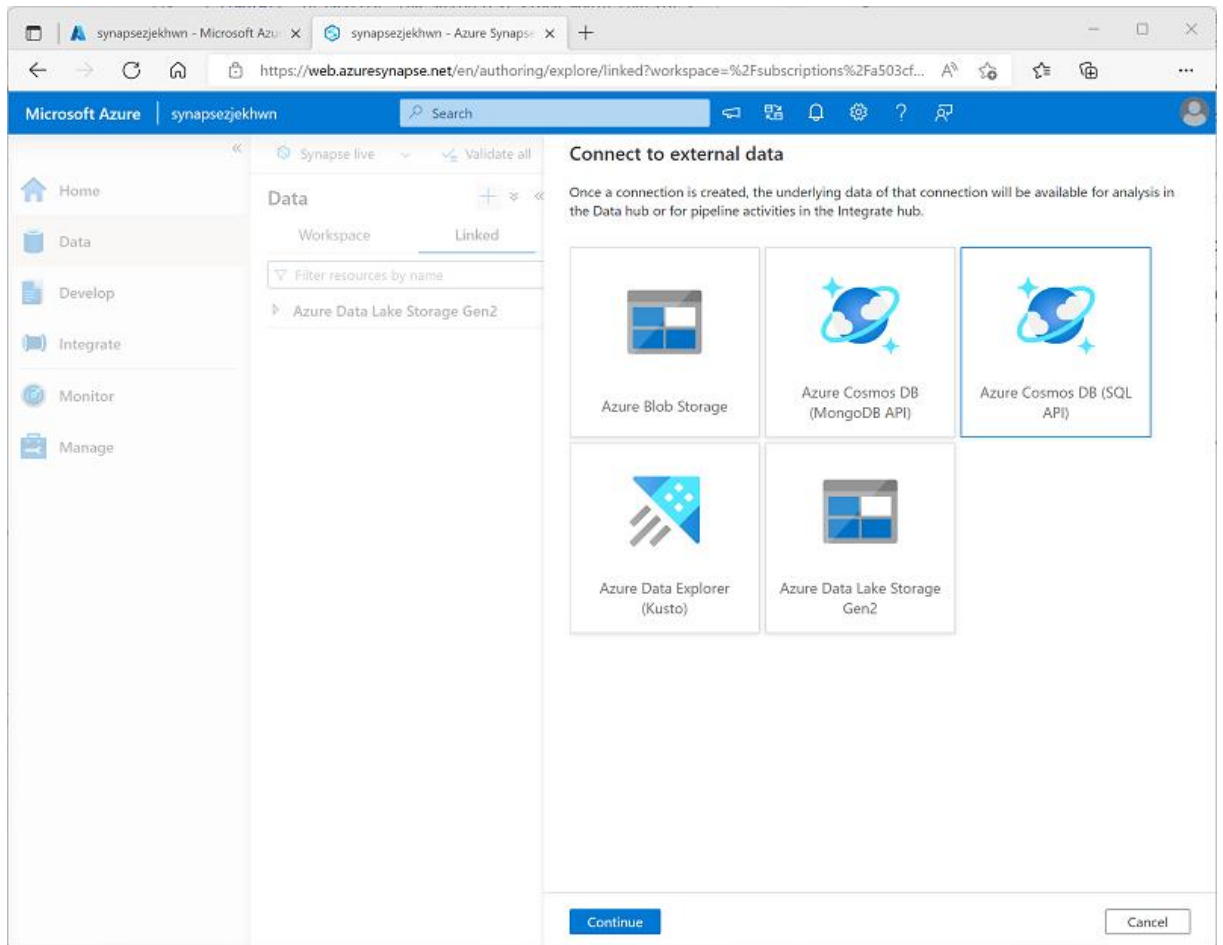


4. On the left side of Synapse Studio, use the >> icon to expand the menu - this reveals the different pages within Synapse Studio.
5. On the **Data** page, view the **Linked** tab. Your workspace should already include a link to your Azure Data Lake Storage Gen2 storage account, but no link to your Cosmos DB account



6. In the + menu, select **Connect to external data**, and then select **Azure Cosmos DB (SQL API)**.







7. Continue, and create a new Cosmos DB connection with the following settings:

- **Name** : SalesCosmos
- **Description**: Sales Cosmos DB SQL API database
- **Connect via integration runtime**: AutoResolveIntegrationRuntime
- **Authentication type**: Account key
- **Connection string**: *selected*
- **Account selection method**: From subscription
- **Azure subscription**: *select your Azure subscription*
- **Azure Cosmos DB account name**: *select your **cosmos-openhack** account*
- **Database name**: database_teamxx (where xx is your db number)

New linked service


 Azure Cosmos DB (SQL API) [Learn more](#) 

 Choose a name for your linked service. This name cannot be updated later.

Name *

SalesCosmos

Description

Connect via integration runtime * 

 AutoResolveIntegrationRuntime

Authentication type

Account key

Connection string


Azure Key Vault

Account selection method 

☒ From Azure subscription ☐ Enter manually

Azure subscription 

Azure subscription 1 (7580b5a3-7bfb-40c4-ab43-a6ec643bb7ef)

Azure Cosmos DB account name * 

cosmos-openhack

Database name *


database_team01

Additional connection properties


 New

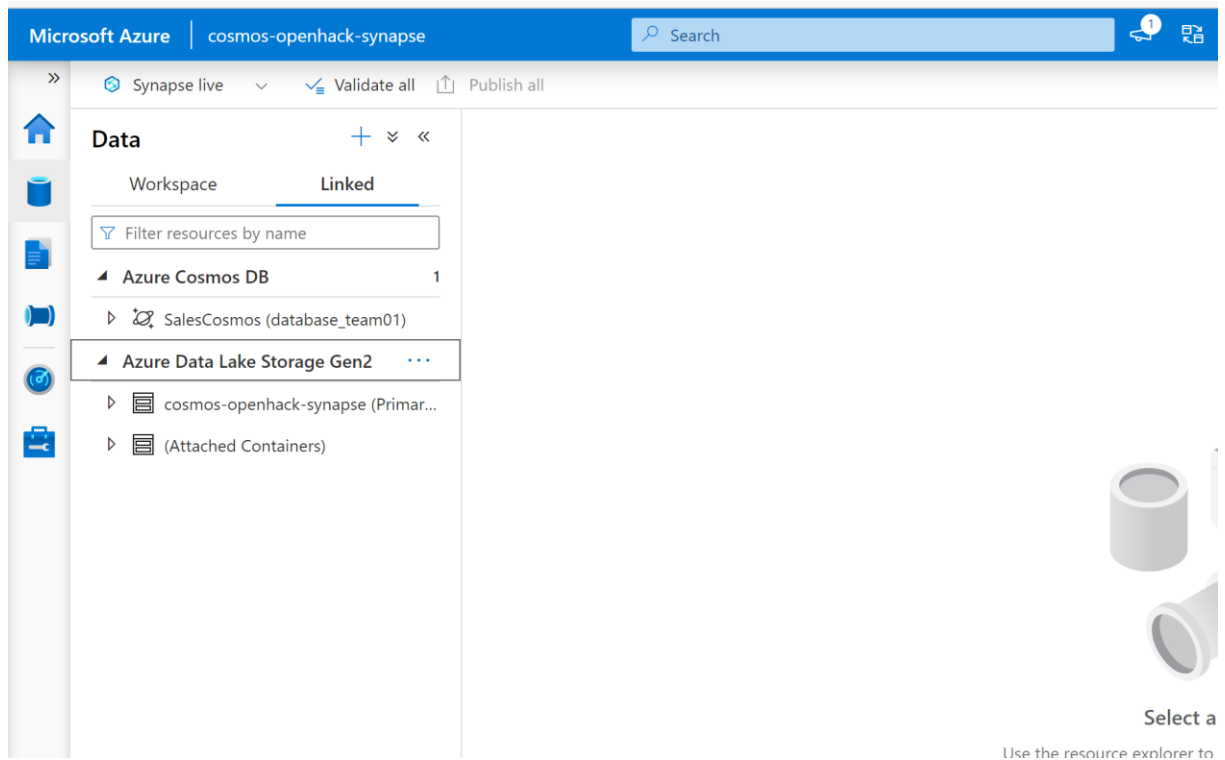
Create

Back

 Test connection

Cancel

- After creating the connection, use the  button at the top right of the **Data** page to refresh the view until an **Azure Cosmos DB** category is listed in the **Linked** pane



9. Expand the **Azure Cosmos DB** category to see the connection you created and the **Sales** container it contains

>>

Synapse live

Validate all

Publish all

Home

Storage

Files

Network

Monitoring

Tools

Data

+ ><

Workspace

Linked

Filter resources by name

▲ Azure Cosmos DB1

▲ SalesCosmos (database_team01)

▶ CartContainer

▶ CartContainerByState

▶ consoleLeases

▶ FoodCollection

▶ materializedViewLeases

▶ Sales...

▶ StateSales

▲ Azure Data Lake Storage Gen22

▶ cosmos-openhack-synapse (Primar...

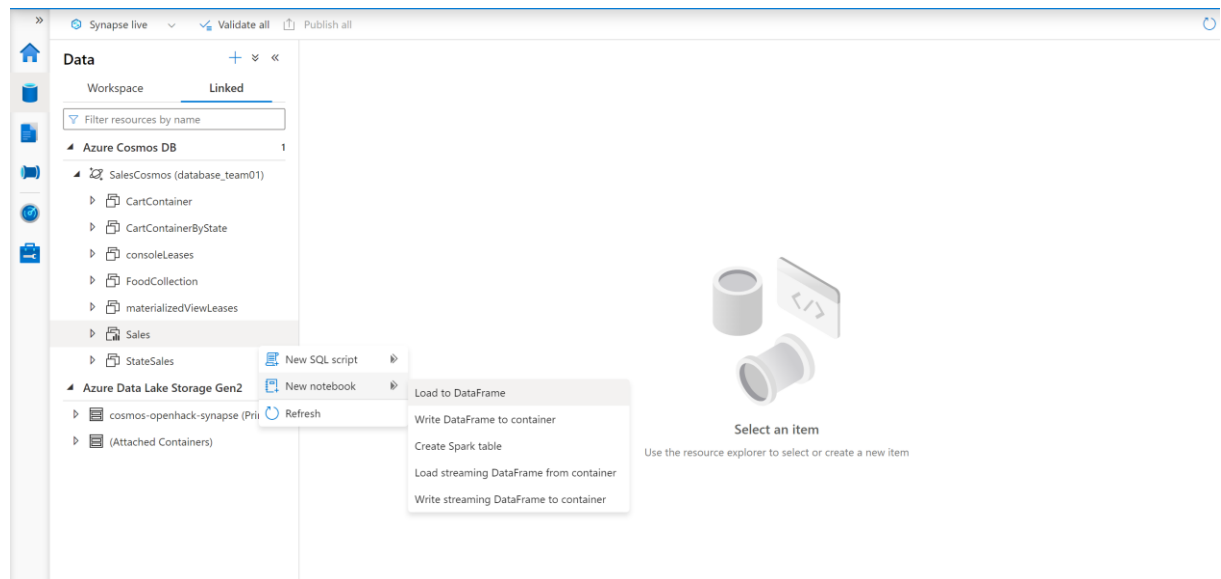
▶ (Attached Containers)

Query Cosmos DB from Azure Synapse Analytics

Now you're ready to query your Cosmos DB database from Azure Synapse Analytics.


Query Cosmos DB from a Spark pool

1. In the **Data** pane, select the **Sales** container, and in its ... menu, select **New Notebook** > **Load to DataFrame**



2. In the new **Notebook 1** tab that opens, in the **Attach to** list, select your Spark pool (**sparkpool**). Then use the ► **Run all** button to run all of the cells in the notebook (there's currently only one!).

Since this is the first time you've run any Spark code in this session, the Spark pool must be started. This means that the first run in the session can take a few minutes. Subsequent runs will be quicker.

3. While you are waiting for the Spark session to initialize, review the code that was generated (you can use the **Properties** button, which looks similar to , on the right end of the toolbar to close the **Properties** pane so you can see the code more clearly). The code should look similar to this:

```
# Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the DataFrame
# To select a preferred list of regions in a multi-region Cosmos DB account, add .option("spark.cosmos.preferredRegions", "<Region1><Region2>")

df = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "AdventureWorksSQL")\
    .option("spark.cosmos.container", "Sales")\
    .load()
```

```
display(df.limit(10))
```

The screenshot shows a Synapse notebook interface. On the left, the 'Data' pane lists linked resources: 'Azure Cosmos DB' (1) and 'Azure Data Lake Storage Gen2' (2). The 'SalesCosmos (database_team01)' container is expanded, showing items like 'CartContainer', 'consoleLeases', 'FoodCollection', 'materializedViewLeases', 'Sales', and 'StateSales'. The main notebook area shows a code cell with the following PySpark code:

```
1 # Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the D
2 # To select a preferred list of regions in a multi-region Cosmos DB account, add .option("sp
3
4 df = spark.read\
5     .format("cosmos.olap")\
6     .option("spark.synapse.linkedService", "SalesCosmos")\
7     .option("spark.cosmos.container", "Sales")\
8     .load()
9
10 display(df.limit(10))
```

The code cell has a status bar indicating 'Job execution Succeeded' and 'Spark 2 executors 8 cores'. Below the code, the output is displayed as a table with 4 columns: '_rid', '_ts', 'id', and 'orderda'. The table contains 3 rows of data:

_rid	_ts	id	orderda
Pww5APVWXmMDAAAAAAAAA...	1656491775	SO43707	2019-07
Pww5APVWXmMBAAAAAAAAA...	1656491713	SO43701	2019-07
Pww5APVWXmMCAAAAAAAAAA...	1656491749	SO43704	2019-07

- When the code has finished running, and then review the output beneath the cell in the notebook. The results should include three records; one for each of the items you added to the Cosmos DB database. Each record includes the fields you entered when you created the items as well as some of the metadata fields that were automatically generated.
- Under the results from the previous cell, use the **+ Code** icon to add a new cell to the notebook, and then enter the following code in it:

```
customer_df = df.select("customerid", "customerdetails")
display(customer_df)
```
- Use the **▶** icon to the left of the cell to run it, and view the results; which should be similar to this:

Synapse live Validate all Publish all 1

Data + ✕ <<

Workspace **Linked**

Filter resources by name

- Azure Cosmos DB 1
 - SalesCosmos (database_team01)
 - CartContainer
 - CartContainerByState
 - consoleLeases
 - FoodCollection
 - materializedViewLeases
 - Sales
 - StateSales
- Azure Data Lake Storage Gen2 2

Notebook 1

Run all Undo Publish Outline Attach to sparkpool Language PySpark (Python)

Ready

_rid	_ts	id	orderdate
Pww5APVWXmMDAAAAAAAAA...	1656491775	SO43707	2019-07
Pww5APVWXmMBAAAAAAAAA...	1656491713	SO43701	2019-07
Pww5APVWXmMCAAAAAAAAAA...	1656491749	SO43704	2019-07

ML

```

1 customer_df = df.select("customerid", "customerdetails")
2 display(customer_df)
3

```

[4] ✓ 1 sec - Command executed in 1 sec 417 ms by user_NUf8c on 5:04:47 PM, 6/29/22

Job execution Succeeded Spark 2 executors 8 cores [View in monitoring](#) [Open Spark UI](#)

View Table Chart Export results

customerid	customerdetails
123	▶ {"customername":"Christy Zhu","customeremail":"chr..."
124	▶ {"customername":"Julio Ruiz","customeremail":"juli..."
125	▶ {"customername":"Emma Brown","customeremail":"emma..."

- This query created a new dataframe containing only the **customerid** and **customerdetails** columns. Observe that the **customerdetails** column contains the JSON structure for the nested data in the source item. In the table of results that is displayed, you can use the ► icon next to the JSON value to expand it and see the individual fields it contains

MI 🔗 🗨 ... 🗑

```

1 # Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the D
2 # To select a preferred list of regions in a multi-region Cosmos DB account, add .option("sp
3
4 df = spark.read\
5     .format("cosmos.olap")\
6     .option("spark.synapse.linkedService", "SalesCosmos")\
7     .option("spark.cosmos.container", "Sales")\
8     .load()
9
10 display(df.limit(10))

```

[2] ✓ 2 sec - Command executed in 2 sec 780 ms by user_NUf8c on 11:12:07 AM, 6/29/22

> **Job execution** Succeeded **Spark** 2 executors 8 cores
[View in monitoring](#) [Open Spark UI](#)

View Table Chart
[Export results](#)

customerid	customerdetails	product	qua
125	▼ {"customername":"Emma Brown", customername: ""Emma Brown"" customeremail: ""emma3@adven	Road-150 Red, 48	1
123	▶ {"customername":"Christy Zhu","c	Mountain-100 Silver, 44	1
124	▶ {"customername":"Julio Ruiz","cus	Mountain-100 Black, 48	1

+ Code + Markdown

8. Add another new code cell and enter the following code:

```

from pyspark.sql import functions as F
custdetails_df =
customer_df.select("customerid",F.json_tuple(F.to_json(F.col("customerdetail
ls")), "customername", "customeremail"))
display(custdetails_df)

```

Run the cell and review the results, which should include the **customername** and **customeremail** from the **customerdetails** value as columns:

1
from pyspark.sql import functions as F
2
custdetails_df = customer_df.select("customerid",F.json_tuple(F.to_json(F.col("customerdetail
3
4
display(custdetails_df)
5

[6]
✓ 2 sec - Command executed in 1 sec 840 ms by user_NUf8c on 5:07:50 PM, 6/29/22

>
Job execution Succeeded
Spark 2 executors 8 cores
[View in monitoring](#)
[Open Spark UI](#)

..

View

Table
Chart

[Export results](#)

customerid	c0	c1
123	Christy Zhu	christy12@adventure-works.com
124	Julio Ruiz	julio1@adventure-works.com
125	Emma Brown	emma3@adventure-works.com

Spark enables you to run complex data manipulation code to restructure and explore the data from Cosmos DB. In this case, the PySpark language includes a **sql** library that provides functions for manipulating JSON data.

9. Add another new code cell and enter the following code:

```
%%sql

-- Create a logical database in the Spark metastore
CREATE DATABASE salesdb;

USE salesdb;

-- Create a table from the Cosmos DB container
CREATE TABLE salesorders using cosmos.olap options (
    spark.synapse.linkedService 'CosmosSales',
    spark.cosmos.container 'Sales'
);

-- Query the table
SELECT *
    FROM salesorders;
```

10. Run the new cell to create a new database containing a table that includes data from the Cosmos DB analytical store.

11. Add another new code cell, and then enter and run the following code:

```
%%sql

SELECT id, orderdate, customerdetails.customername, product
FROM salesorders
    ORDER BY id;
```

The results from this query should resemble this:

[M](#) [L](#) [F](#) [...](#) [W](#)

> | ✓

```

1  %%sql
2  SELECT id, orderdate, customerdetails.customername, product
3  FROM salesorders
4  ORDER BY id;
5

```

[14]
✓ 3 sec - Command executed in 2 sec 854 ms by user_NUf8c on 5:13:04 PM, 6/29/22

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

[View in monitoring](#)
[Open Spark UI](#)

...

View

Table

Chart

[Export results](#)

⌵

	orderdate	customername	product
13701	2019-07-01	Christy Zhu	Mountain-100
13704	2019-07-01	Julio Ruiz	Mountain-100
13707	2019-07-02	Emma Brown	Road-150 Red

⌵

[+ Code](#)
[+ Markdown](#)

Observe that when using Spark SQL, you can retrieve subelements of a JSON structure as properties.

12. Keep the **Notebook 1** tab open - you'll return to it later.

Query Cosmos DB from a serverless SQL pool

1. In the **Data** pane, select the **Sales** container, and in its ... menu, select **New SQL script** > **Select TOP 100 rows**.
2. In the **SQL script 1** tab that opens, hide the **Properties** pane and view the code that has been generated, which should look similar to this:

```

IF (NOT EXISTS(SELECT * FROM sys.credentials WHERE name = 'cosmosxxxxxxxx'))
THROW 50000, 'As a prerequisite, create a credential with Azure Cosmos DB key
in SECRET option:
CREATE CREDENTIAL [cosmosxxxxxxxx]
WITH IDENTITY = ''SHARED ACCESS SIGNATURE'', SECRET = ''<Enter your Azure
Cosmos DB key here>'', 0
GO

SELECT TOP 100 *
FROM OPENROWSET(PROVIDER = 'CosmosDB',
CONNECTION = 'Account=cosmosxxxxxxxx;Database=AdventureWorks',
OBJECT = 'Sales',
SERVER_CREDENTIAL = 'cosmosxxxxxxxx'

```

```
) AS [Sales]
```

The SQL pool requires a credential to use when accessing Cosmos DB, which is based on an authorization key for your Cosmos DB account. The script includes an initial `IF (NOT EXISTS(...` statement that checks for this credential, and throws an error if it does not exist.

Replace the `IF (NOT EXISTS(...` statement in the script with the following code to create a credential, replacing `cosmosxxxxxxx` with the name of your Cosmos DB account:

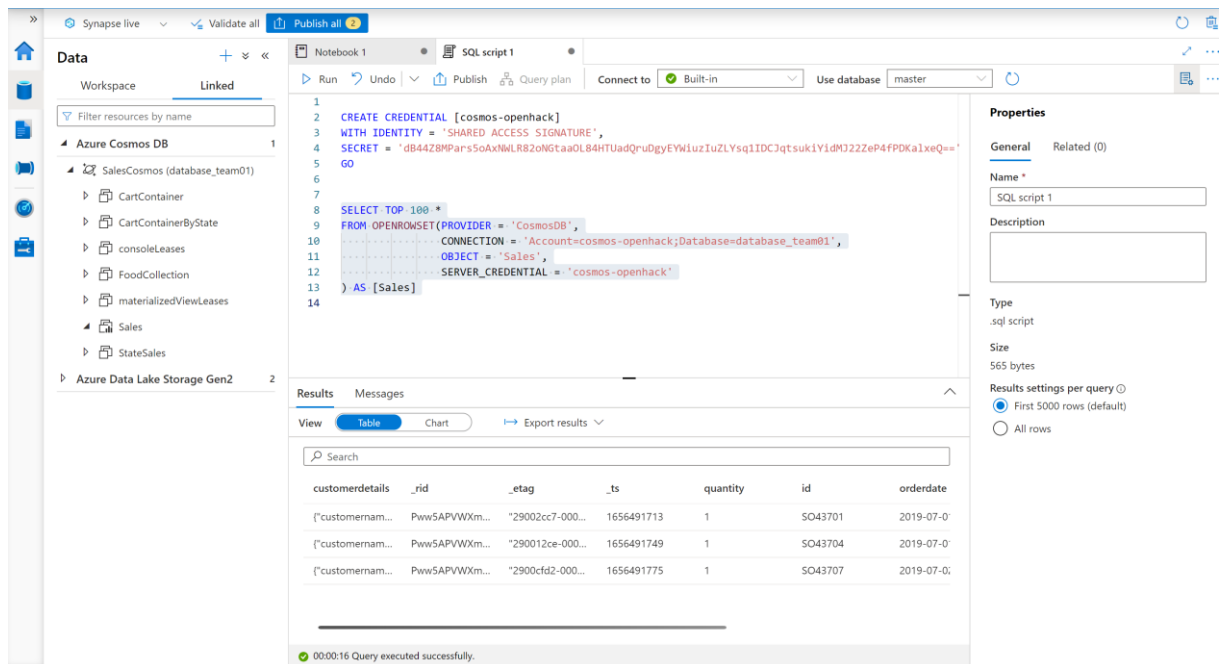
```
CREATE CREDENTIAL [cosmosxxxxxxx]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
SECRET = '<Enter your Azure Cosmos DB key here>'
GO
```

The whole script should now resemble the following:

```
CREATE CREDENTIAL [cosmosxxxxxxx]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
SECRET = '<Enter your Azure Cosmos DB key here>'
GO

SELECT TOP 100 *
FROM OPENROWSET(PROVIDER = 'CosmosDB',
                CONNECTION =
                'Account=cosmosxxxxxxx;Database=database_team01',
                OBJECT = 'Sales',
                SERVER_CREDENTIAL = 'cosmosxxxxxxx'
) AS [Sales]
```

3. Switch to the browser tab containing the Azure portal (or open a new tab and sign into the Azure portal at <https://portal.azure.com>). Then in the **cosmos-openhack-shared-rg** resource group, open your **cosmos-openhack** Azure Cosmos DB account.
4. In the pane on the left, in the **Settings** section, select the **Keys** page. Then copy the **Primary Key** value to the clipboard.
5. Switch back to the browser tab containing the SQL script in Azure Synapse Studio, and paste the key into the code replacing the **<Enter your Azure Cosmos DB key here>** placeholder so that the script looks similar to this:



- Use the **Run** button to run the script, and review the results, which should include three records; one for each of the items you added to the Cosmos DB database.

Now that you have created the credential, you can use it in any query against the Cosmos DB data source.

- Replace all of the code in the script (both the CREATE CREDENTIAL and SELECT statements) with the following code (substituting *cosmosxxxxxxx* with the name of your Azure Cosmos DB account):

```

SELECT *
FROM OPENROWSET(PROVIDER = 'CosmosDB',
                  CONNECTION = 'Account=cosmos-openhack;
Database=database_teamxx',
                  OBJECT = 'Sales',
                  SERVER_CREDENTIAL = 'cosmos-openhack'
)
WITH (
    OrderID VARCHAR(10) '$.id',
    OrderDate VARCHAR(10) '$.orderdate',
    CustomerID INTEGER '$.customerid',
    CustomerName VARCHAR(40) '$.customerdetails.customername',
    CustomerEmail VARCHAR(30) '$.customerdetails.customeremail',
    Product VARCHAR(30) '$.product',
    Quantity INTEGER '$.quantity',
    Price FLOAT '$.price'
)
AS sales
ORDER BY OrderID;

```

- Run the script and review the results, which should match the schema defined in the WITH clause:

9. Keep the **SQL script 1** tab open - you'll return to it later.

Verify data modifications in Cosmos DB are reflected in Synapse

1. Leaving the browser tab containing Synapse Studio open, switch back to the tab containing the Azure portal, which should be open at the **Keys** page for your Cosmos DB account.
2. On the **Data Explorer** page, expand the **database_teamxxx** database and its **Sales** folder; and then select the **Items** folder.
3. Use the **New Item** button to create a new customer item based on the following JSON. Then save the new item (some additional metadata fields will be added when you save the item).

```
{
  "id": "S043708",
  "orderdate": "2019-07-02",
  "customerid": 126,
  "customerdetails": {
    "customername": "Samir Nadoy",
    "customeremail": "samir1@adventure-works.com"
  },
  "product": "Road-150 Black, 48",
  "quantity": 1,
  "price": 3578.27
}
```

4. Return to the Synapse Studio tab and in the **SQL Script 1** tab, re-run the query. Initially, it may show the same results as before, but wait a minute or so and then re-run the query again until the results include the sale to Samir Nadoy on 2019-07-02.
5. Switch back to the **Notebook 1** tab and re-run the last cell in the Spark notebook to verify that the sale to Samir Nadoy is now included in the query results.