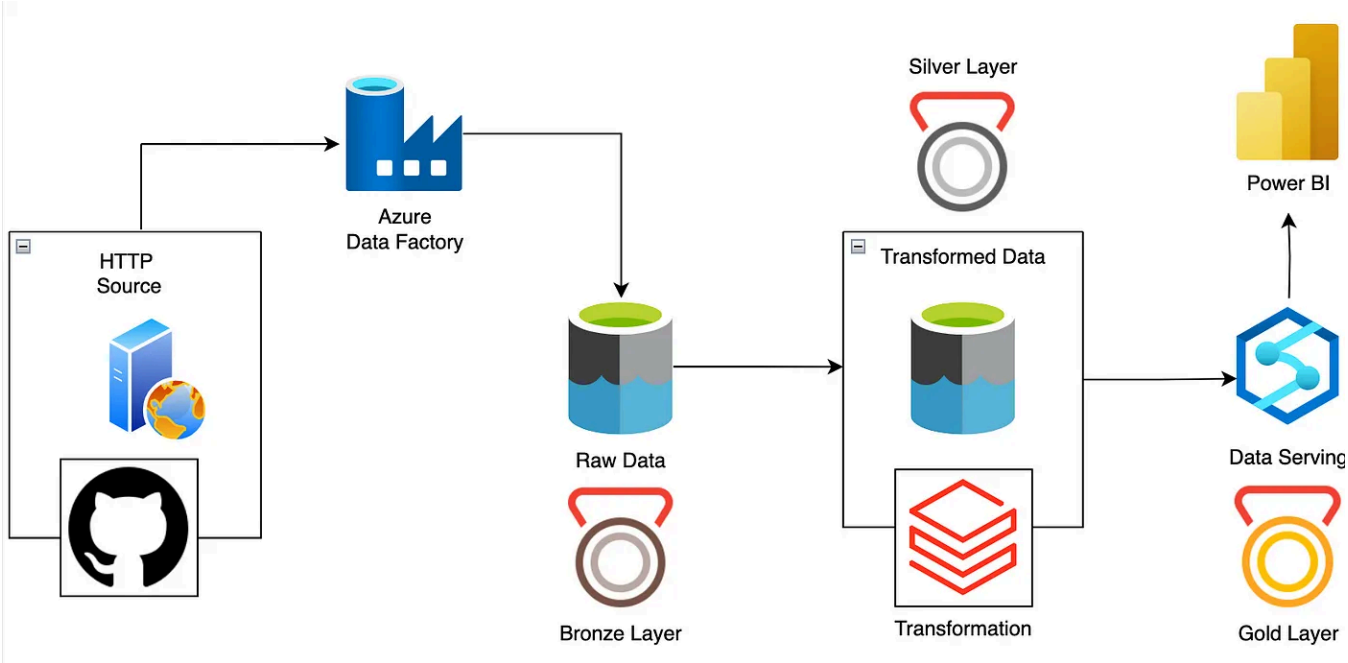
Azure End-To-End Data Engineering Project

Suraj Badrinarayan S
9 min read · Jan 14, 2025



This is an end-to-end Azure Data Engineering Project where I upload data from GitHub HTTP source to bronze layer through Azure Data factory. Transform the data in the silver layer using Databricks and provide the data to downstream (Analytics, Data Science) using Azure Synapse Analytics.

I have followed **Medallion** architecture where:

- Raw data is uploaded to bronze layer using parameters and dynamic pipelines.
- Raw data is Transformed in the solver layer using Databricks.

- Finally, the Transformed data is served into gold layer, where views of the data are created for downstream analysis.

ABOUT THE DATASET:

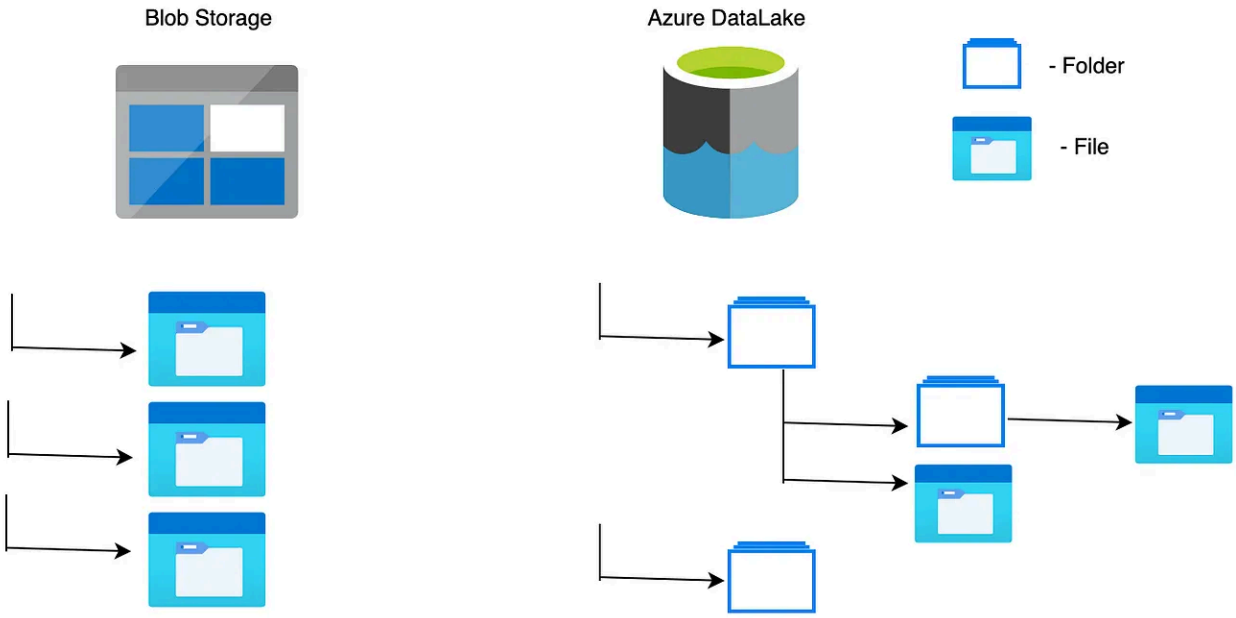
This is an open-source dataset from Kaggle named — **Adventure Works**.

Here is a quick description of the columns in the dataset.

- AdventureWorks_Calendar.csv
- AdventureWorks_Customers.csv
- AdventureWorks_Product_Categories.csv
- AdventureWorks_Product_Subcategories.csv
- AdventureWorks_Products.csv
- AdventureWorks_Returns.csv
- AdventureWorks_Sales_2015.csv
- AdventureWorks_Sales_2016.csv
- AdventureWorks_Sales_2017.csv
- AdventureWorks_Territories.csv

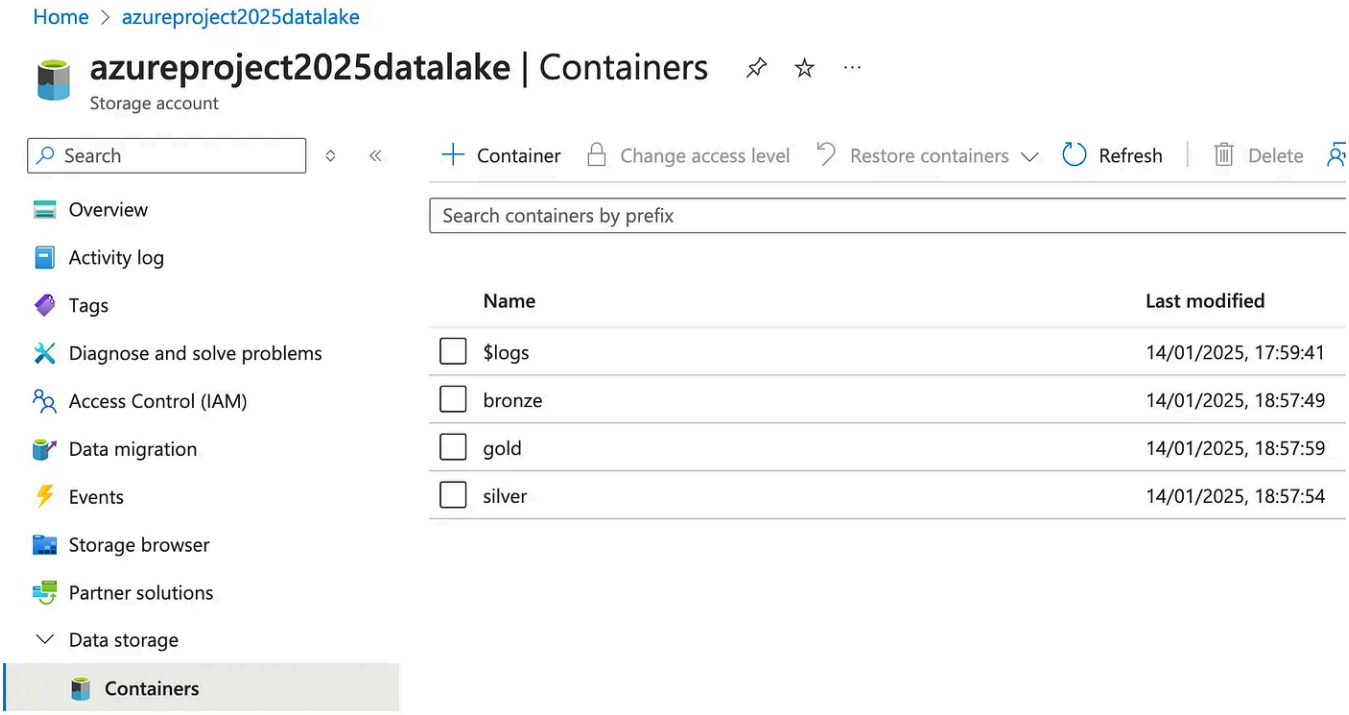
I have used Azure DataLake Storage to store the data.

Why ADLS Gen 2?



Storage in Blog Vs. ADLS

In ADLS, we can store data in the form of hierarchies. While creating the storage account, make sure to enable "**Hierarchical Namespace**" to create ADLS, else a blob storage will be created by default.

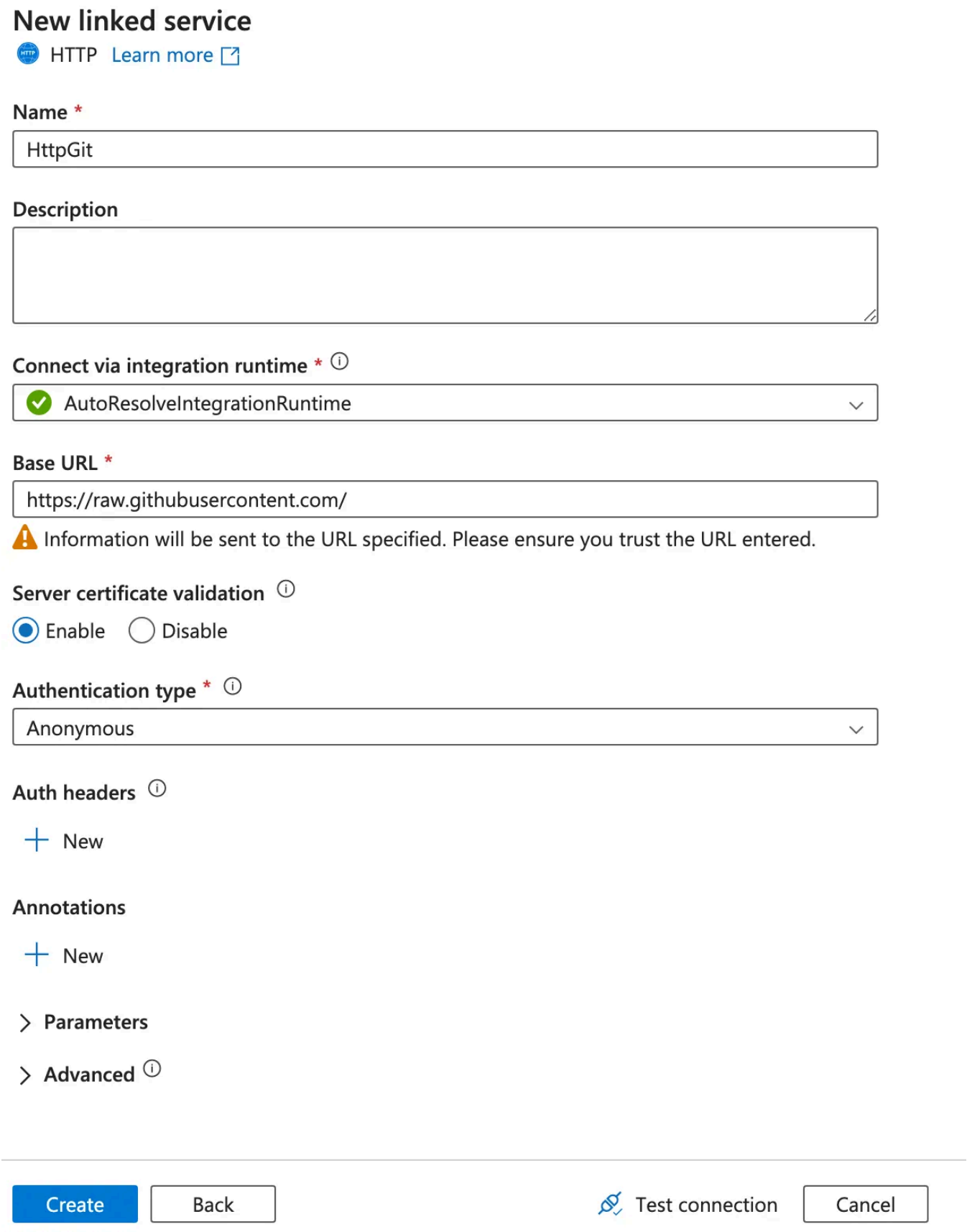


Containers in Storage Account

I have created 3 containers to follow **Medallion** architecture (bronze, silver and gold)

THE BRONZE LAYER:

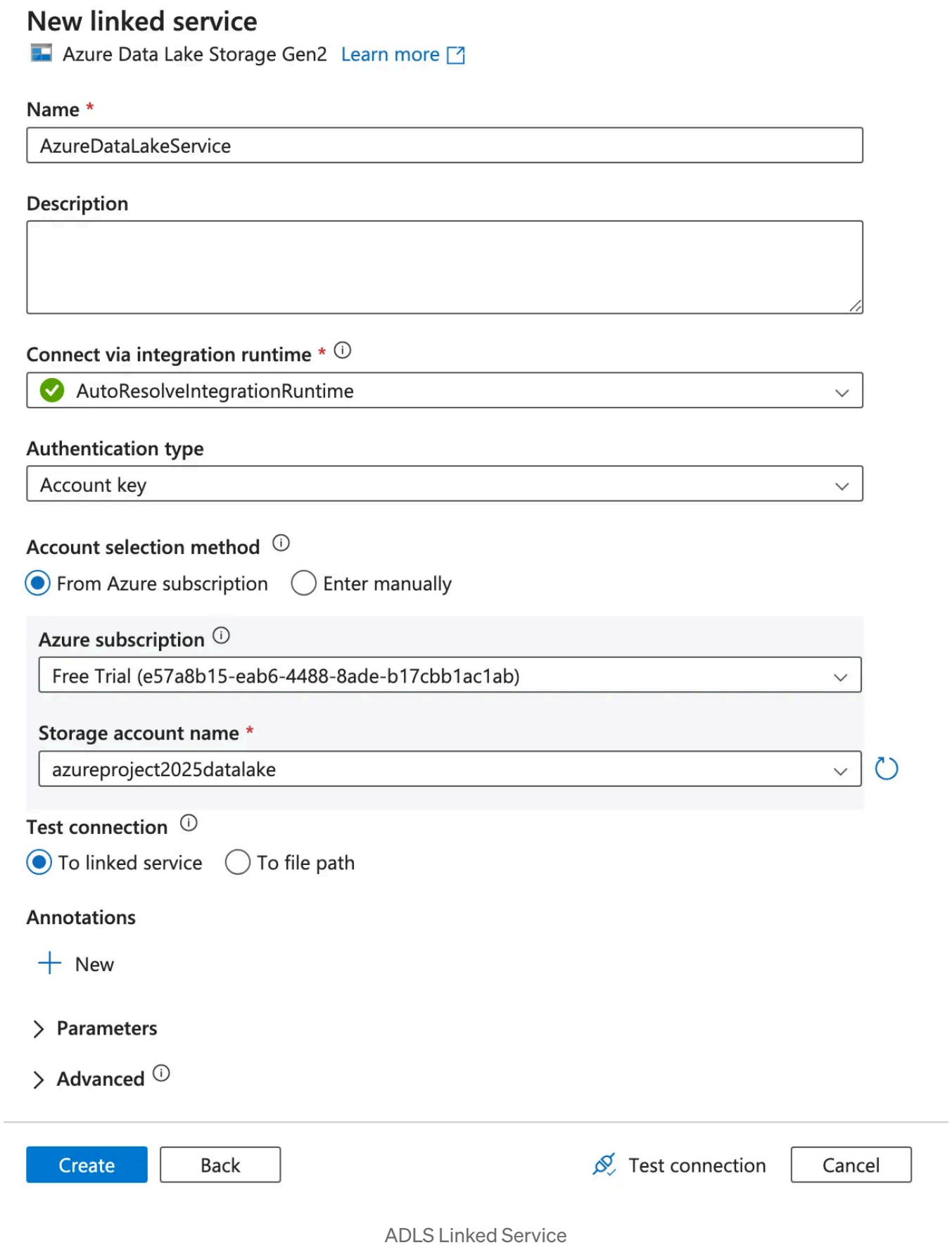
Now, I will load the data from GitHub to the bronze layer (raw data). I am first creating a static loading of data from GitHub to bronze, then I will create dynamic pulling of all files from a folder to bronze.



HTTP Linked Service

I have created an HTTP Linked Service by giving the base URL of products.csv file from GitHub's Raw data.

Now, I will create a Azure DataLake Storage Linked Service to connect to ADLS.

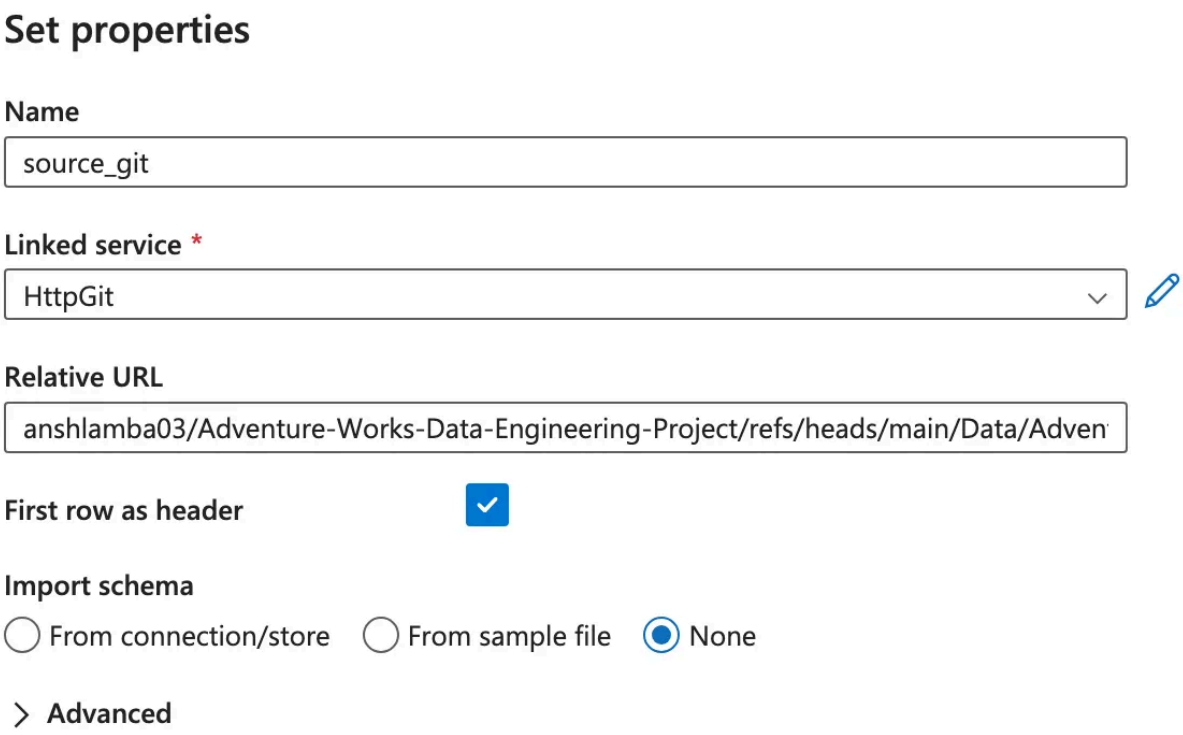


ADLS Linked Service

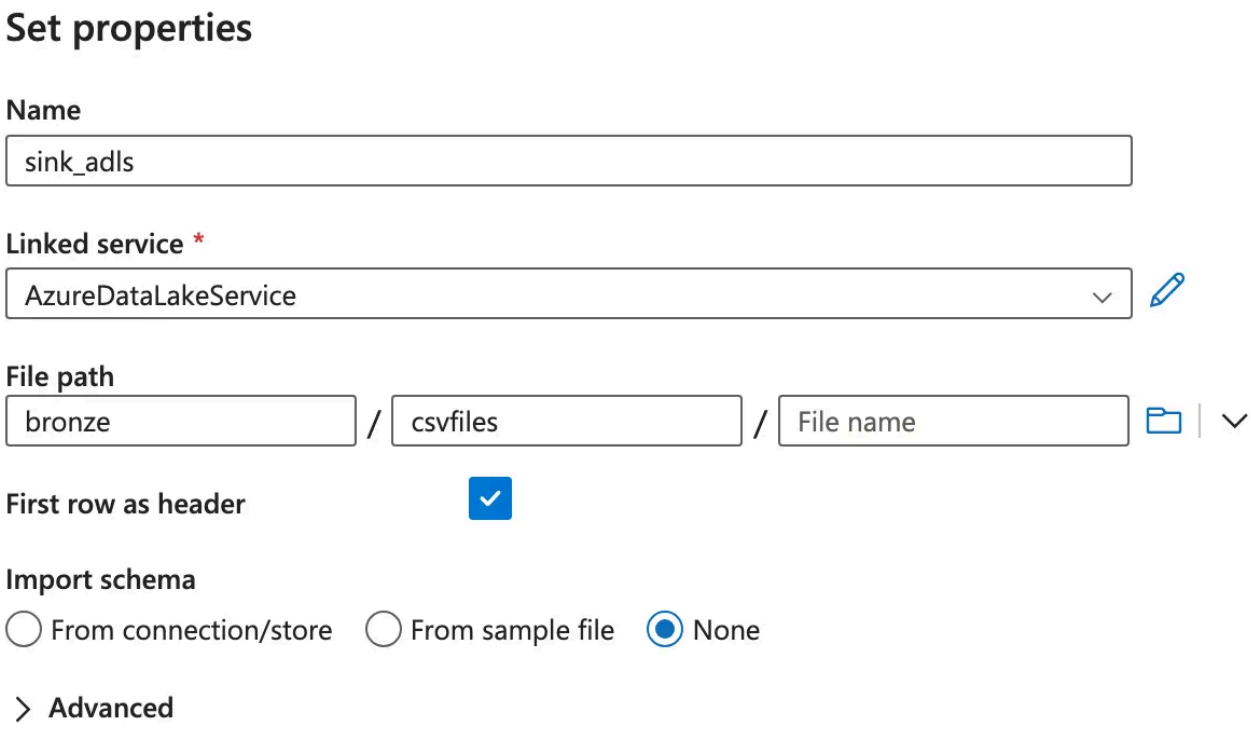
COPY ACTIVITY:

Copy Activity requires the following information:

- Source: It requires the name of the dataset, linked service and relative path of the source dataset. (For data stored in ADLS, for GitHub we will be using an HTTP Linked Service)
- Sink: It requires the name of the dataset, linked service and relative path of the source dataset. (To store data in ADLS)

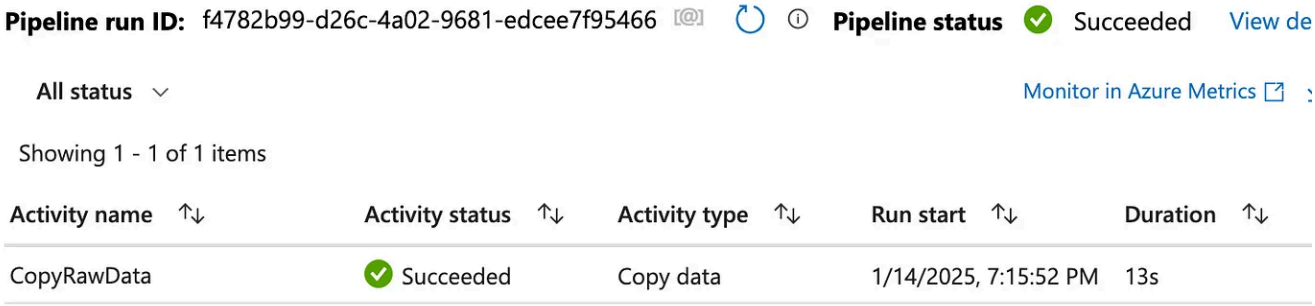


Source Dataset from git

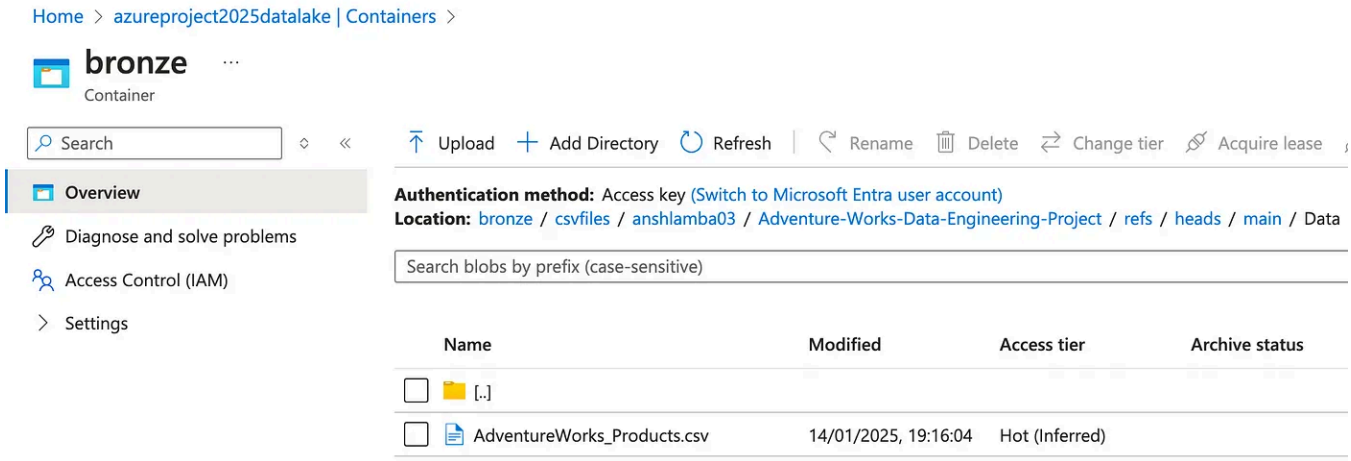


Sink Dataset for ADLS

After connecting Copy Activity to Source and Sink, now I will run the debug option to execute this Pipeline.



Successful Execution of Pipeline

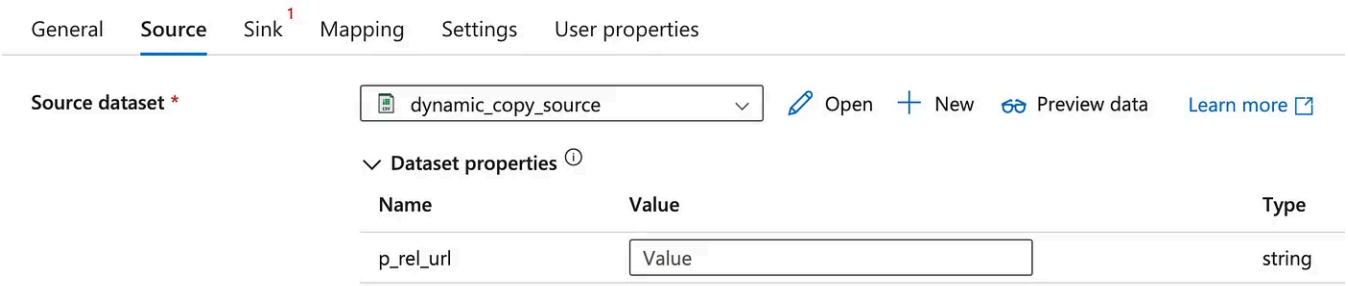


Data Successfully Copied to the bronze folder

Now, I will be creating a Copy Activity which will be parameterised.

I created a new Pipeline where I dragged a Copy Activity.

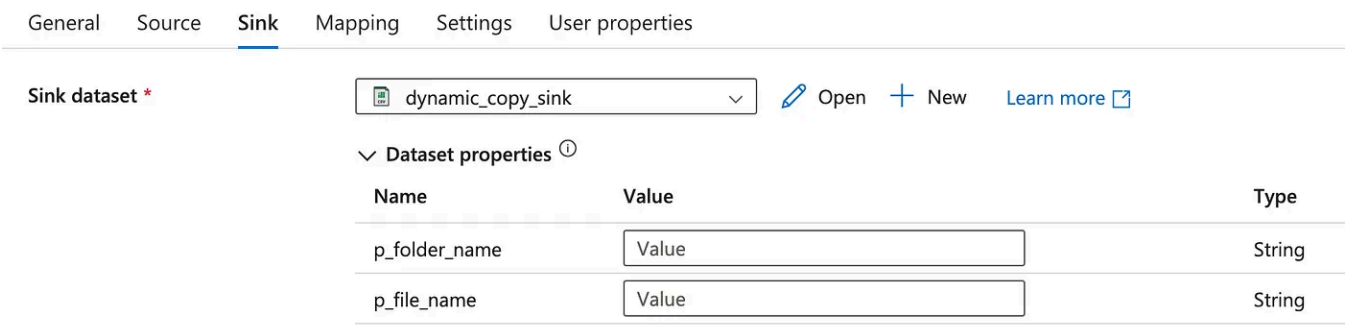
The source will be parameterised in the following way:



Parameterised Source Dataset

The input (each rel_url will be coming from a ForEach Activity) of the Copy Activity receives a p_rel_url which will dynamically change upon each Copy step.

The sink will be parameterised in the following way:



Parameterised Sink Dataset

I will now create a json file which will contain the following fields:

- p_rel_url
- p_sink_folder
- p_sink_file

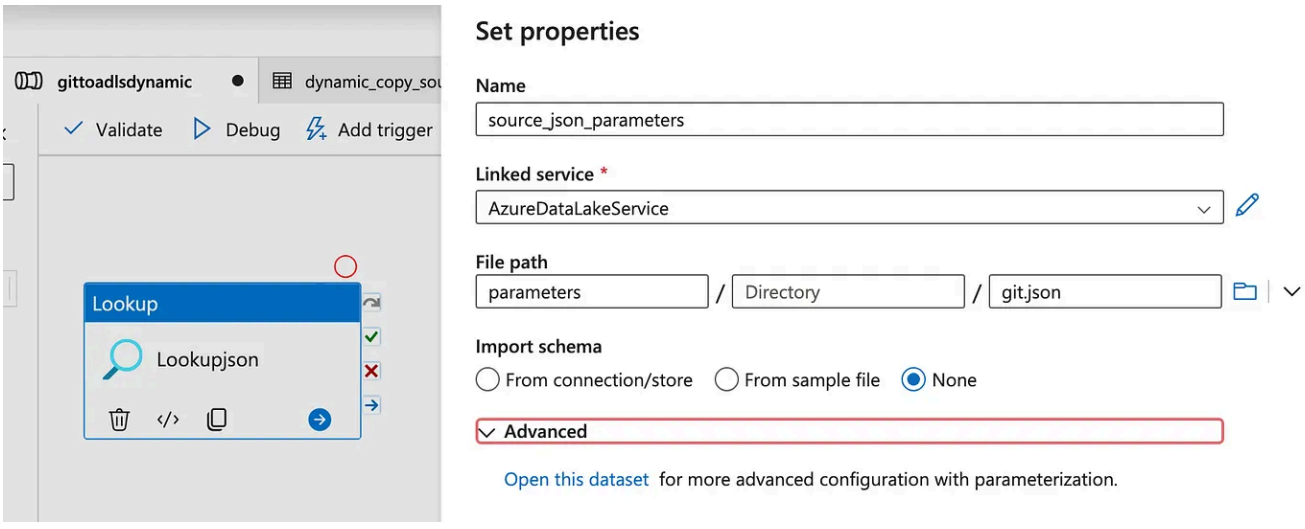
Sample json file snippet:

```
[
    {
        "p_rel_url" : "anshlamba03/Adventure-Works-Data-Engineering-Project/main/Data/AdventureWorks_Product_Categories.csv",
        "p_sink_folder" : "AdventureWorks_Product_Categories",
        "p_sink_file" : "AdventureWorks_Product_Categories.csv"
    },
    {
        "p_rel_url" : "anshlamba03/Adventure-Works-Data-Engineering-Project/main/Data/AdventureWorks_Calendar.csv",
        "p_sink_folder" : "AdventureWorks_Calendar",
        "p_sink_file" : "AdventureWorks_Calendar.csv"
    },
```

json file with Parameters

This git.json will be uploaded in the StorageLake under parameters folder.

I will be creating an Activity called LookUp (To get the information inside git.json file)



LookUp Activity to Read git.json

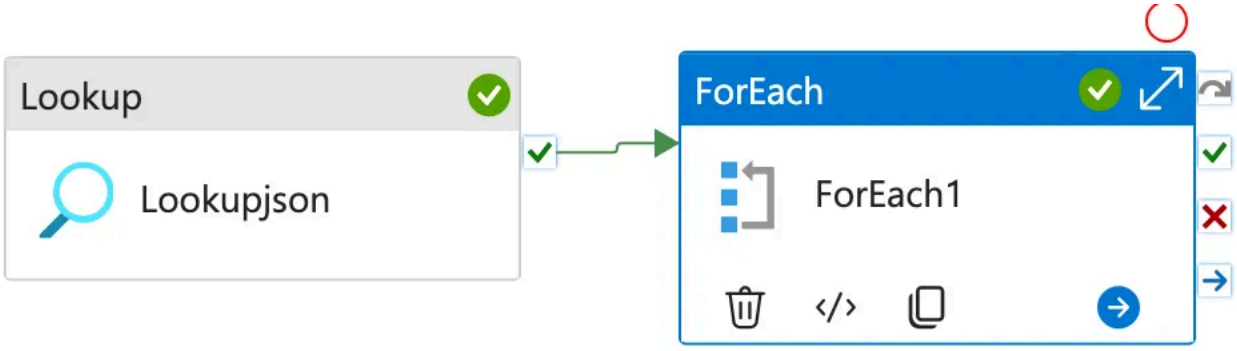
Now, the output of this LookUp Activity will be directed to FarEach Activity. (make sure to uncheck the “First Row” to read the entire data in the json file).

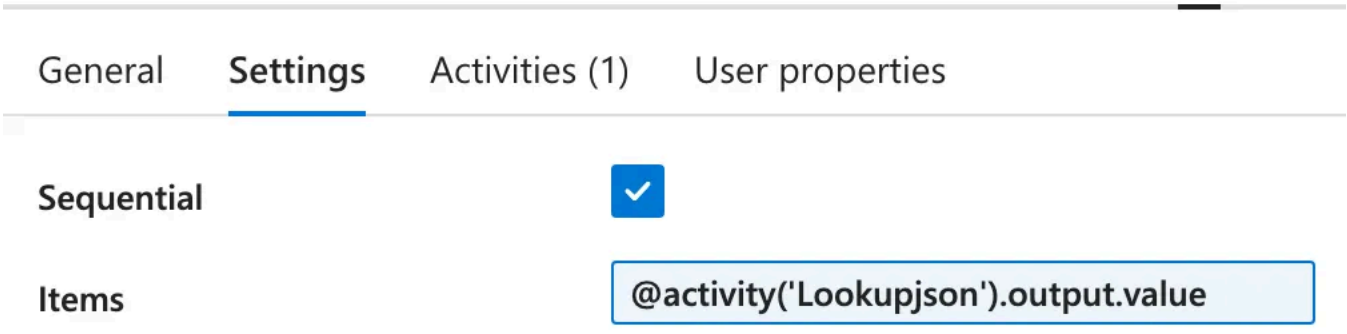
Output

Copy to clipboard


```
{
    "count": 10,
    "value": [
        {
            "p_rel_url": "anshlamba03/Adventure-Works-Data-Engineering-Project/main/Data/AdventureWorks_Product_Categories.csv",
            "p_sink_folder": "AdventureWorks_Product_Categories",
            "p_sink_file": "AdventureWorks_Product_Categories.csv"
        },
        {
            "p_rel_url": "anshlamba03/Adventure-Works-Data-Engineering-Project/main/Data/AdventureWorks_Calendar.csv",
            "p_sink_folder": "AdventureWorks_Calendar",
            "p_sink_file": "AdventureWorks_Calendar.csv"
        },
        {
            "p_rel_url": "anshlamba03/Adventure-Works-Data-Engineering-Project/main/Data/AdventureWorks_Customers.csv",
            "p_sink_folder": "AdventureWorks_Customers",
            "p_sink_file": "AdventureWorks_Customers.csv"
        },
```

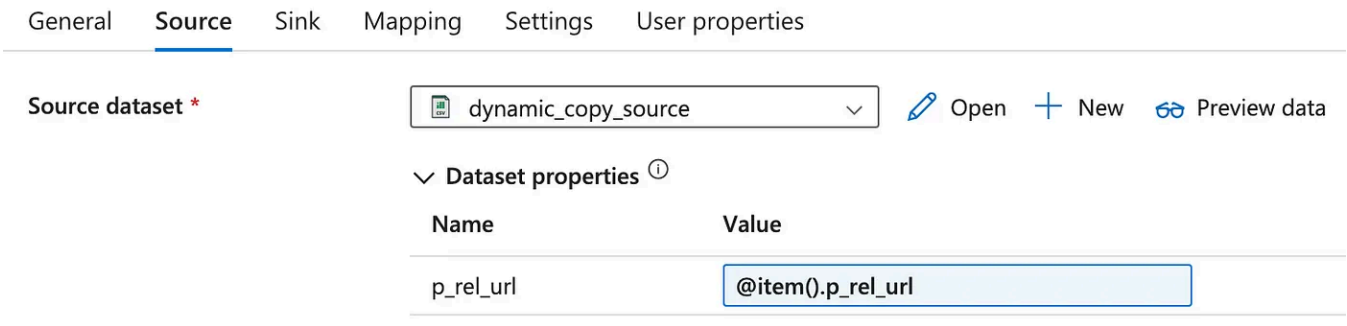

Output of LookUp Activity



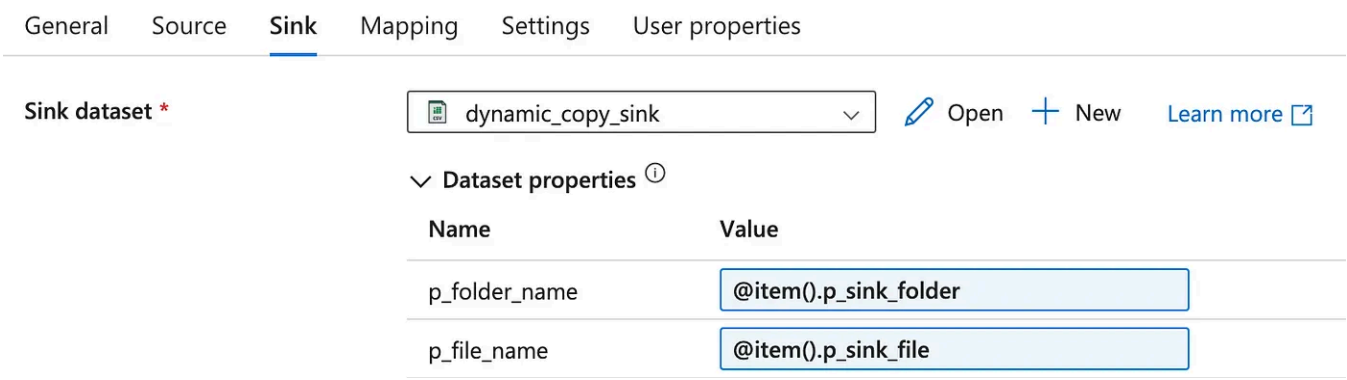


I have now connected the Lookupjson Activity to ForEach1 such that the output (all file parameters) will be fed to ForEach Activity (output of Lookup.value).

Inside ForEach Activity canvas, I will paste the Copy Activity which I have created already and input the parameters as following:



Source: Input of p_rel_url from ForEach Item

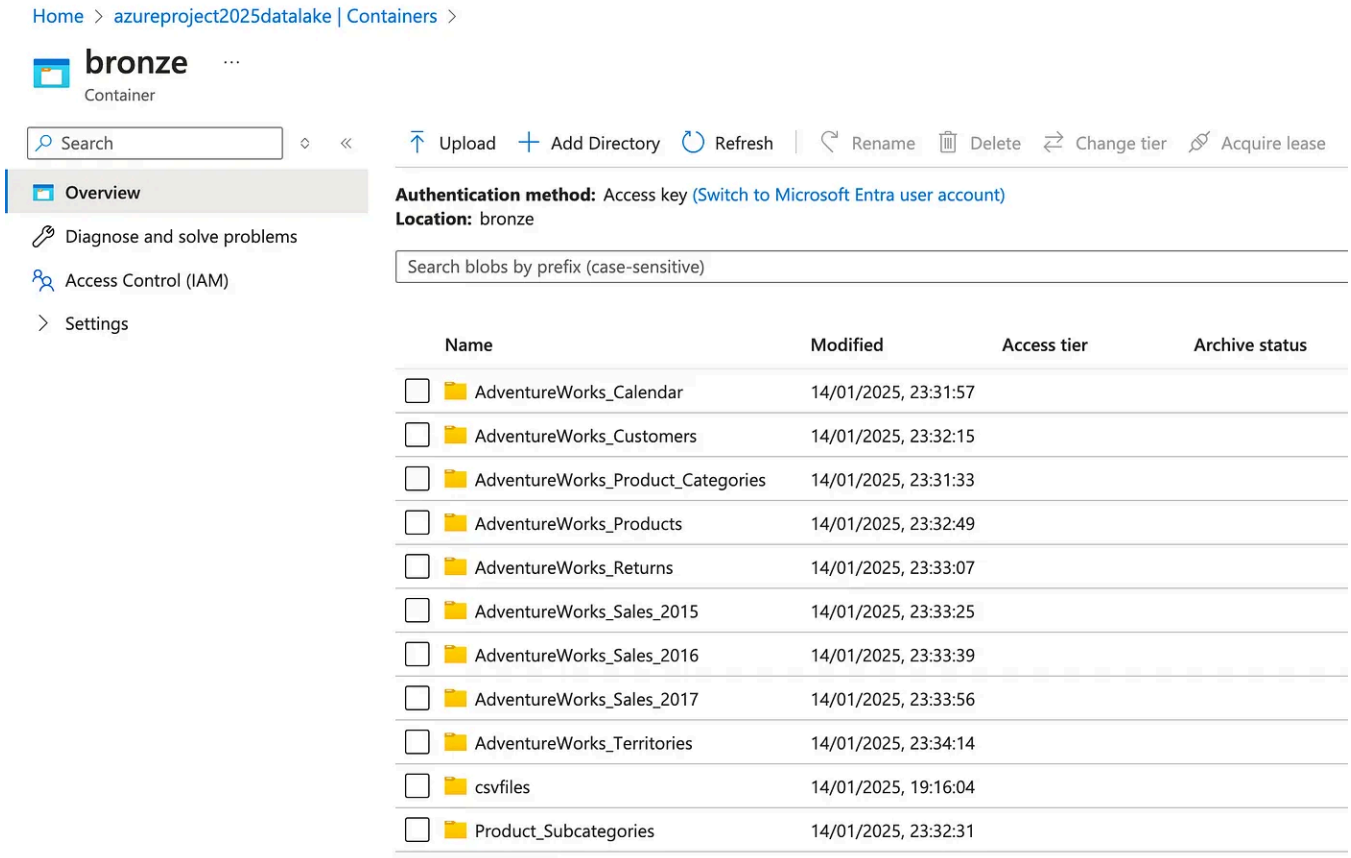


Sink: Input of p_folder_name and p_file_name from ForEach Item

Now, I will exit the ForEach Canvas and will run the debug option.

dynamiccopy	Succeeded	Copy data	1/14/2025, 11:31:45 PM	14s
dynamiccopy	Succeeded	Copy data	1/14/2025, 11:31:18 PM	26s
ForEach1	Succeeded	ForEach	1/14/2025, 11:31:18 PM	3m 0s
Lookupjson	Succeeded	Lookup	1/14/2025, 11:31:09 PM	7s

The Pipeline is Executed Successfully



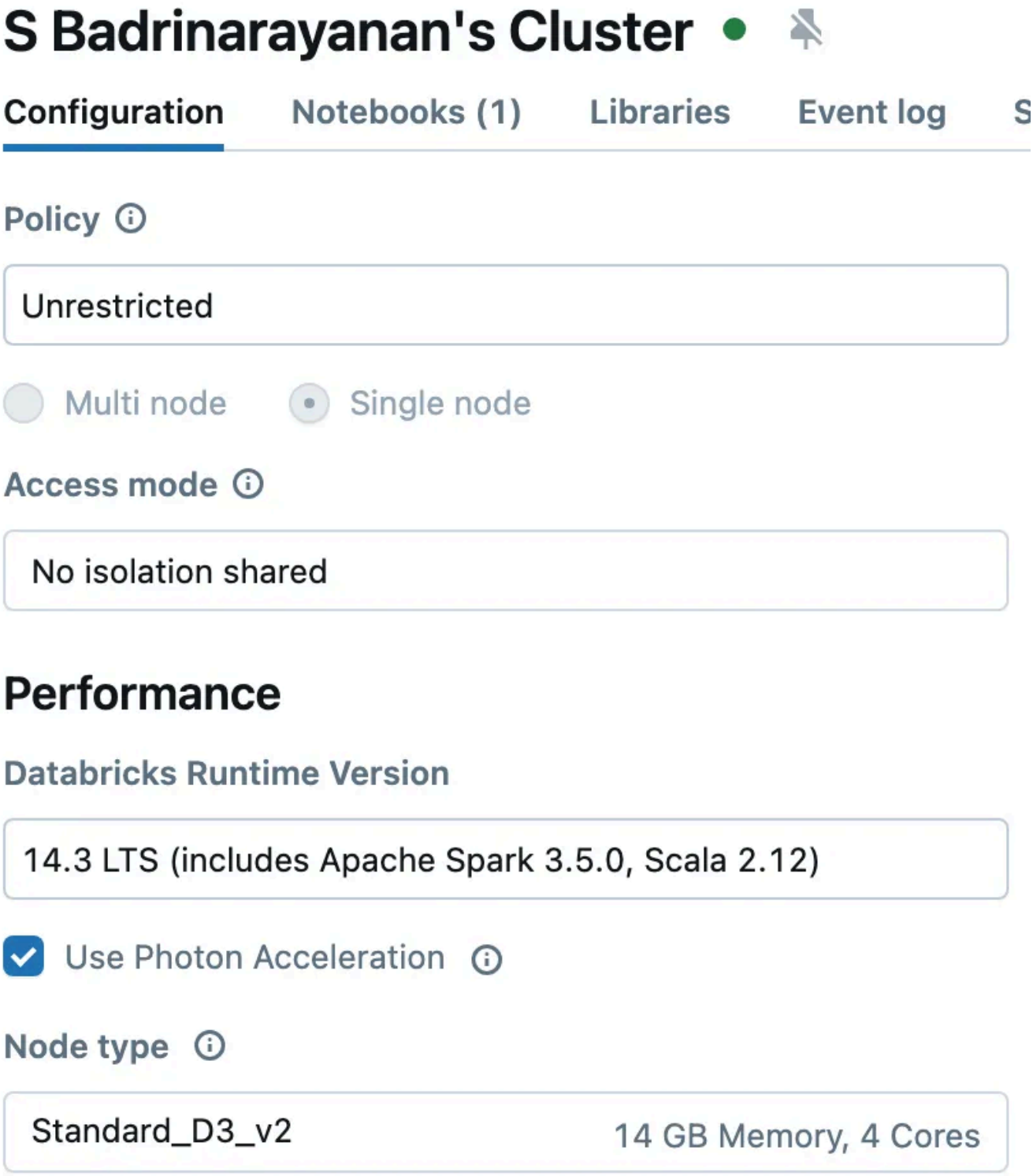
All the Files are now Uploaded in the bronze container

The **BRONZE** layer of the **Medallion** architecture is now completed with a parameterised approach.

THE SILVER LAYER:

Now I will be creating a Databricks Resource under the Resource Group (Azure-Project).

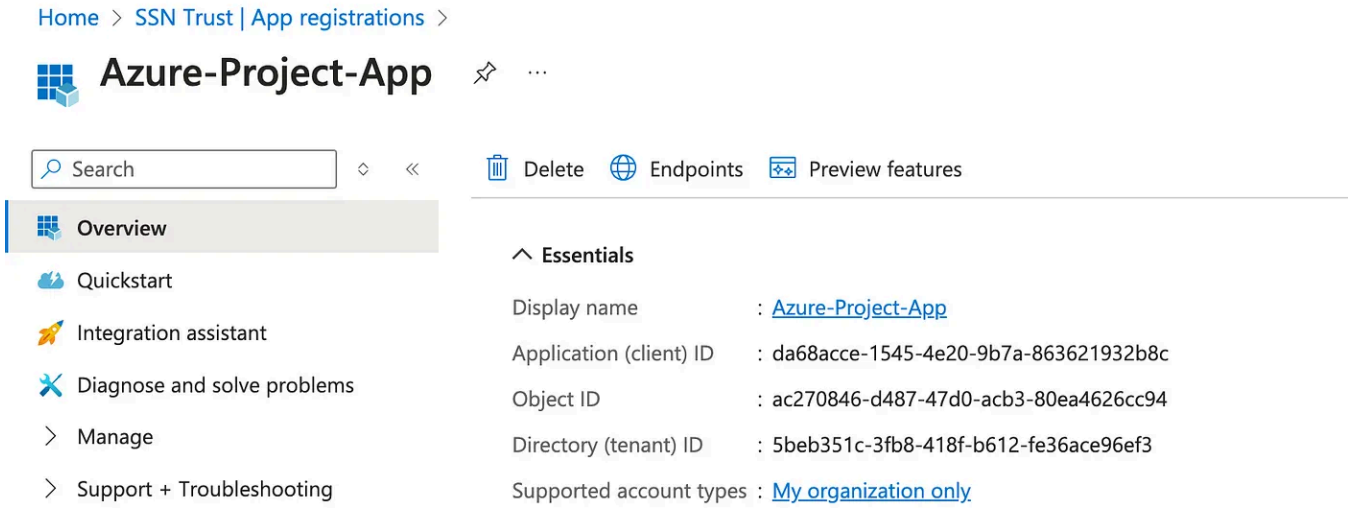
The compute information of the Databricks Cluster is as follows:



Compute Information of Cluster

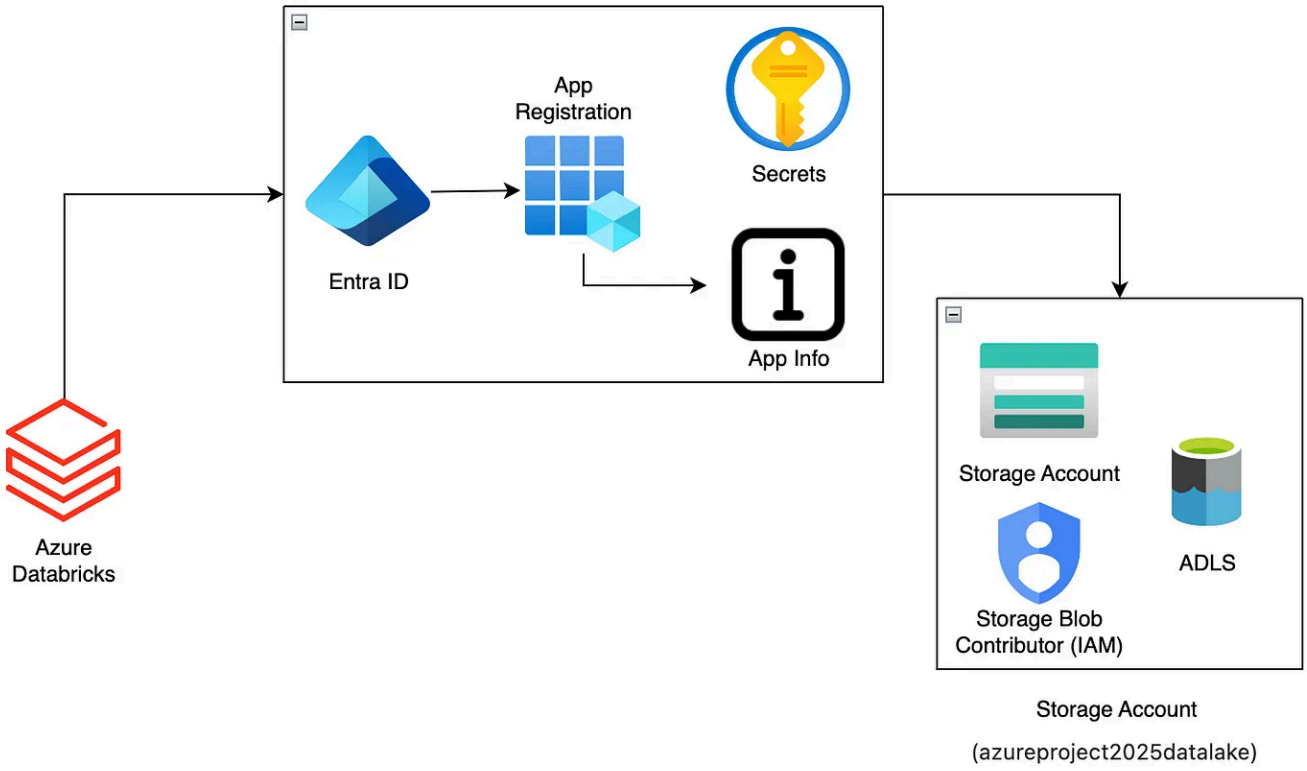
I will now create a notebook (silver_notebook) under a folder (Databricks) inside the Workspace tab.

I will now register an application in Microsoft Entra ID, so that I can connect Databricks to my Storage Account by giving the **Credentials** of the application.



Application Details to connect Databricks and ADLS

Here is a simple flow diagram to connect Azure Databricks with ADLS:



Connection of Azure Databricks and ADLS

I have completed the following steps for connection:

- Copy Paste the **information** regarding the Application
- Create a secret and Copy the **Value** of the secret for secure connection
- Grant **Storage Blob Data Contributor** to Storage Account through **IAM** and add the Application as the member.

- Apply the connecting **Credentials** in the Databrick notebook

```
▶ 2

spark.conf.set("fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.
oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net", "<application-id>")
spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net", service_credential)
spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.net", "https://login.
microsoftonline.com/<directory-id>/oauth2/token")
```

Code for Connecting Azure DataBricks with ADLS

```
▶ ✓ 4 minutes ago (5s)  3

df_cal = spark.read.format('csv')\
            .option("header",True)\
            .option("inferSchema",True)\
            .load('abfss://bronze@azureproject2025datalake.dfs.core.windows.net/AdventureWorks_Calendar')


df_cus = spark.read.format('csv')\
            .option("header",True)\
            .option("inferSchema",True)\
            .load('abfss://bronze@azureproject2025datalake.dfs.core.windows.net/AdventureWorks_Customers')
```

Sample Loading of Data

If the code is running properly but you see red underlines in your editor, it likely means there are **syntax checking issues** within the editor/IDE. These do not affect the execution of the code.

Now I will be performing some Transformations and will push the Transformed data into the silver layer.

PYSPARK TRANSFORMATIONS:

```
05:47 PM (<1s)                                                    7

df_cal = df_cal.withColumn('Month',month(col('Date'))).withColumn('Year',year(col('Date')))

df_cal:  pyspark.sql.dataframe.DataFrame = [Date: date, Month: integer ... 1 more field]
```

```
05:47 PM (1s)                                                     8

df_cal.display()

(1) Spark Jobs
```

Table +

	Date	Month	Year
1	2015-01-01	1	2015
2	2015-01-02	1	2015

Transformation of df_cal column

I have created a new column called ‘Month’ and ‘Year’ by extracting the month and year of the Date column.

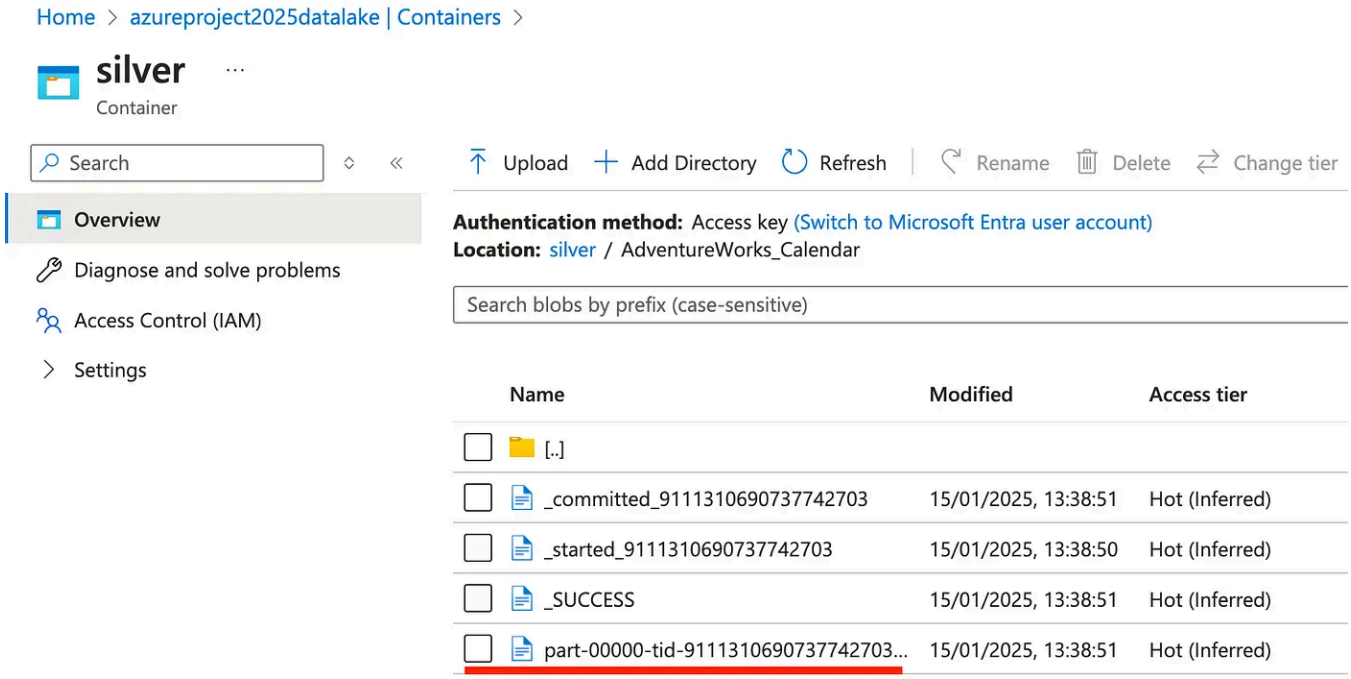
withColumn function is used to create or modify a column. It:

- Creates a column if we provide different column name.
- Modifies a column if we provide same column name.

Now, I will write the df_cal (Transformed data) into the silver container.

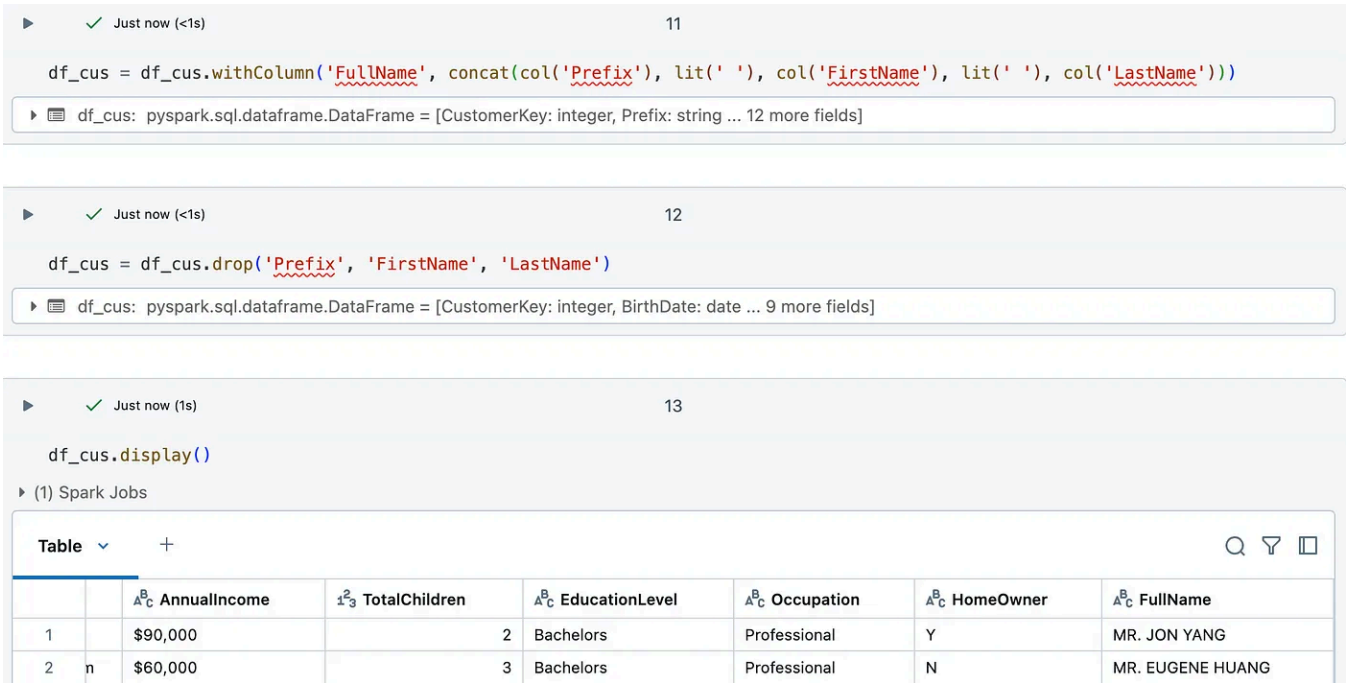
```
Just now (1s)                                                     9

df_cal.write.format('parquet')\
        .mode('append')\
        .option("path","abfss://silver@azureproject2025datalake.dfs.core.windows.net/AdventureWorks_Calendar")\
        .save()

(1) Spark Jobs
```

Load the data to silver container



Data Successfully Loaded into the silver container

Now I will be performing Transformations on df_cus data.



Creating FullName Column using concat() function

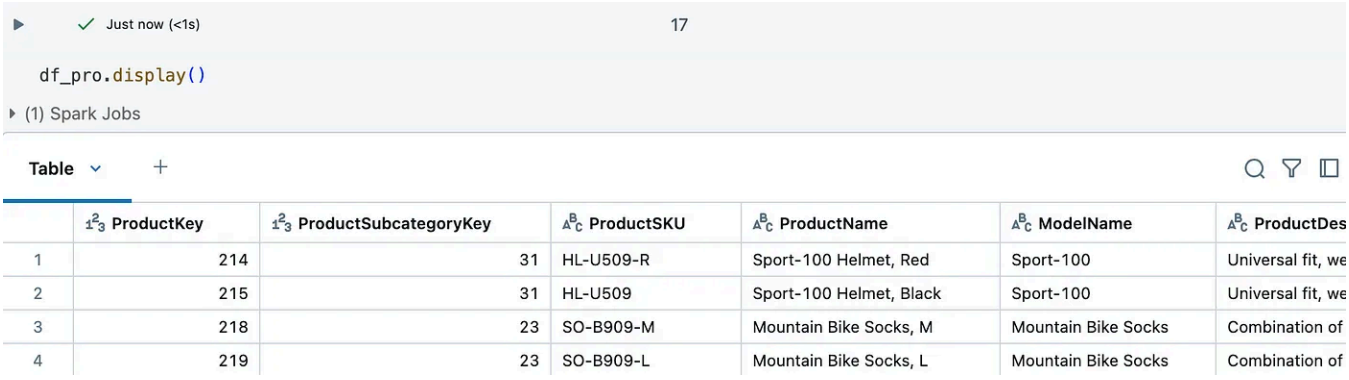
Here I have used the following functions:

- concat: concats different columns
- lit: since (‘ ‘) is a space (i.e a constant) I have used lit function.

- drop: I have dropped prefix, first and last name because I have combined them into a single column.

I will now write df_cus data into silver container and the code is same as df_cal except that the dataframe name only must be changed.

Now, I will Transform df_pro data.



	ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductDes
1	214	31	HL-U509-R	Sport-100 Helmet, Red	Sport-100	Universal fit, we
2	215	31	HL-U509	Sport-100 Helmet, Black	Sport-100	Universal fit, we
3	218	23	SO-B909-M	Mountain Bike Socks, M	Mountain Bike Socks	Combination of
4	219	23	SO-B909-L	Mountain Bike Socks, L	Mountain Bike Socks	Combination of

Sample Snippet of df_pro Data

I will perform a Transformation that retrieves the size of a product from **ProductSKU.**

Notice that some of the records in **ProductSKU** doesn't have any size to it. In that case we will get **NULL** values. I will try to fill the **NULL** with the value **Not Applicable.**

1 minute ago (<1s) 18

```
df_pro = df_pro.withColumn('ProductSize',split(col('ProductSKU'),'-')[2])
```

df_pro: pyspark.sql.dataframe.DataFrame = [ProductKey: integer, ProductSubcategoryKey: integer ... 9 more fields]

Just now (<1s) 19

```
df_pro = df_pro.fillna({'ProductSize': 'Not Applicable'})
```

df_pro: pyspark.sql.dataframe.DataFrame = [ProductKey: integer, ProductSubcategoryKey: integer ... 9 more fields]

Just now (<1s) 20

```
df_pro.display()
```

(1) Spark Jobs

Table +

		ProductColor	ProductSize	ProductStyle	ProductCost	ProductPrice
1		Red	R	0	13.0863	34.99
2		Black	Not Applicable	0	12.0278	33.6442
3		White	M	U	3.3963	9.5
4		White	L	U	3.3963	9.5
5		Blue	B	0	12.0278	33.6442
6		Multi	Not Applicable	U	5.7052	8.6442

Created ProductSize and filled NULL with Not Applicable

I will now write the df_pro into the silver container.

I will analyse data in df_sales using some Transformations.

Just now (1s) 26

```
df_sales.groupby(col('TerritoryKey')).agg(sum(col('OrderQuantity')).alias('TotalQuantity')).display()
```

(2) Spark Jobs

Table +

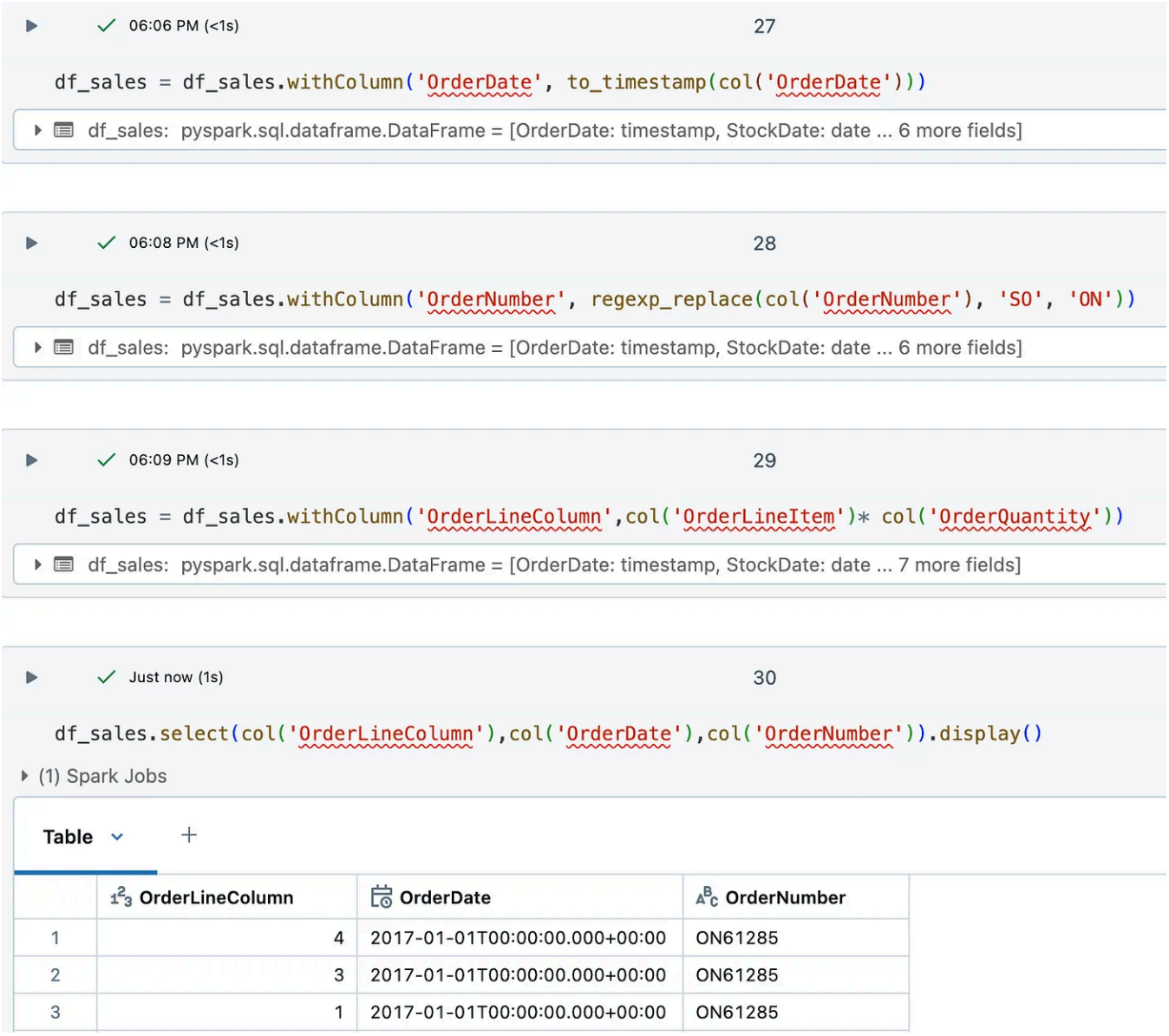
	TerritoryKey	TotalQuantity
1	1	12513
2	6	10894
3	3	30
4	5	49
5	9	17951

GroupBy Function

GroupBy Aggregation:

- I **grouped** the data in accordance with **TerritoryKey** to see the performance of Orders based on Territories.

- I have performed a **sum** aggregation on **OrderQuantity** column to see the total number of orders.
- Finally I have used **alias** function to rename the sum(OrderQuantity) column.

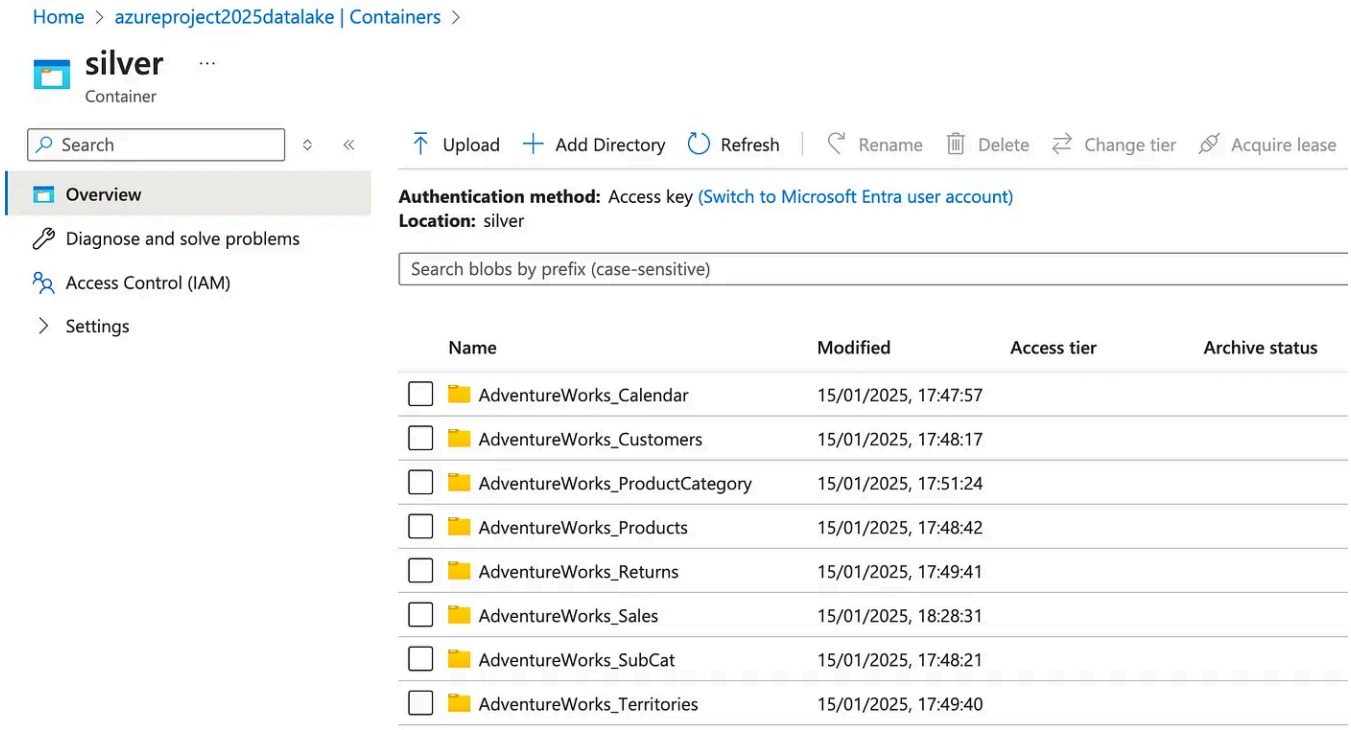


I have made the following Transformations to df_sales data:

- **regexp_replace** function replaces a string with the string we want. In this case I have replaces SO with ON (Order Number).
- Performed arithmetic operation on columns where **OrderLineColumn** is obtained by multiplying **OrderLineItem** and **OrderQuantity.**
- Sometimes downstream professionals require date information in timestamps. So, I have converted the **OrderDate** into timestamp.

- Finally, I have used **Select** function to show only the changed and modified columns instead of the whole data frame.

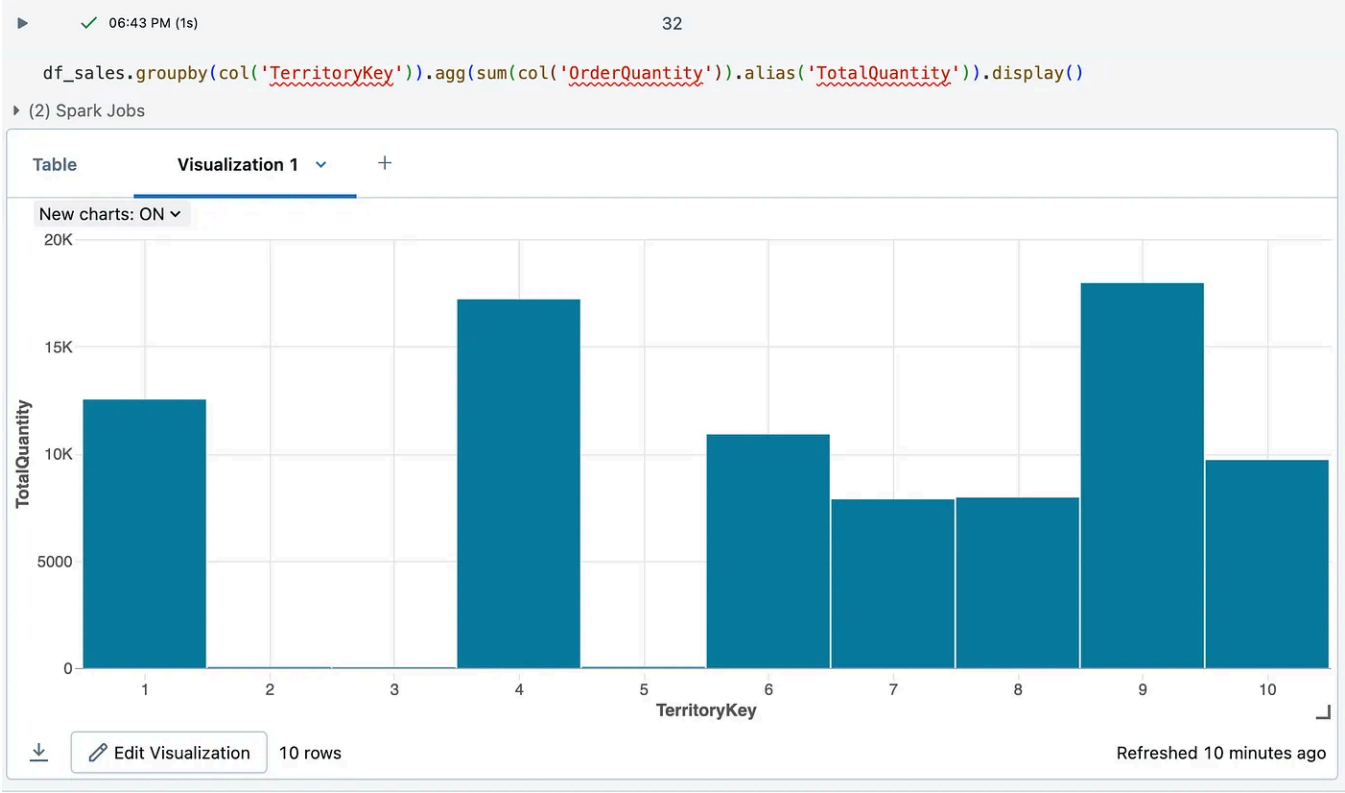
Now, I will write all the remaining data frames to silver container.



Transformed data in silver container

VISUALISATION IN DATABRICKS:

Scenario 1:



Territory-wise Performance

I have done a Territory-wise analysis where we can understand which territory is best performing in sales.

Scenario 2:

Month-wise and Year-wise Sales Analysis

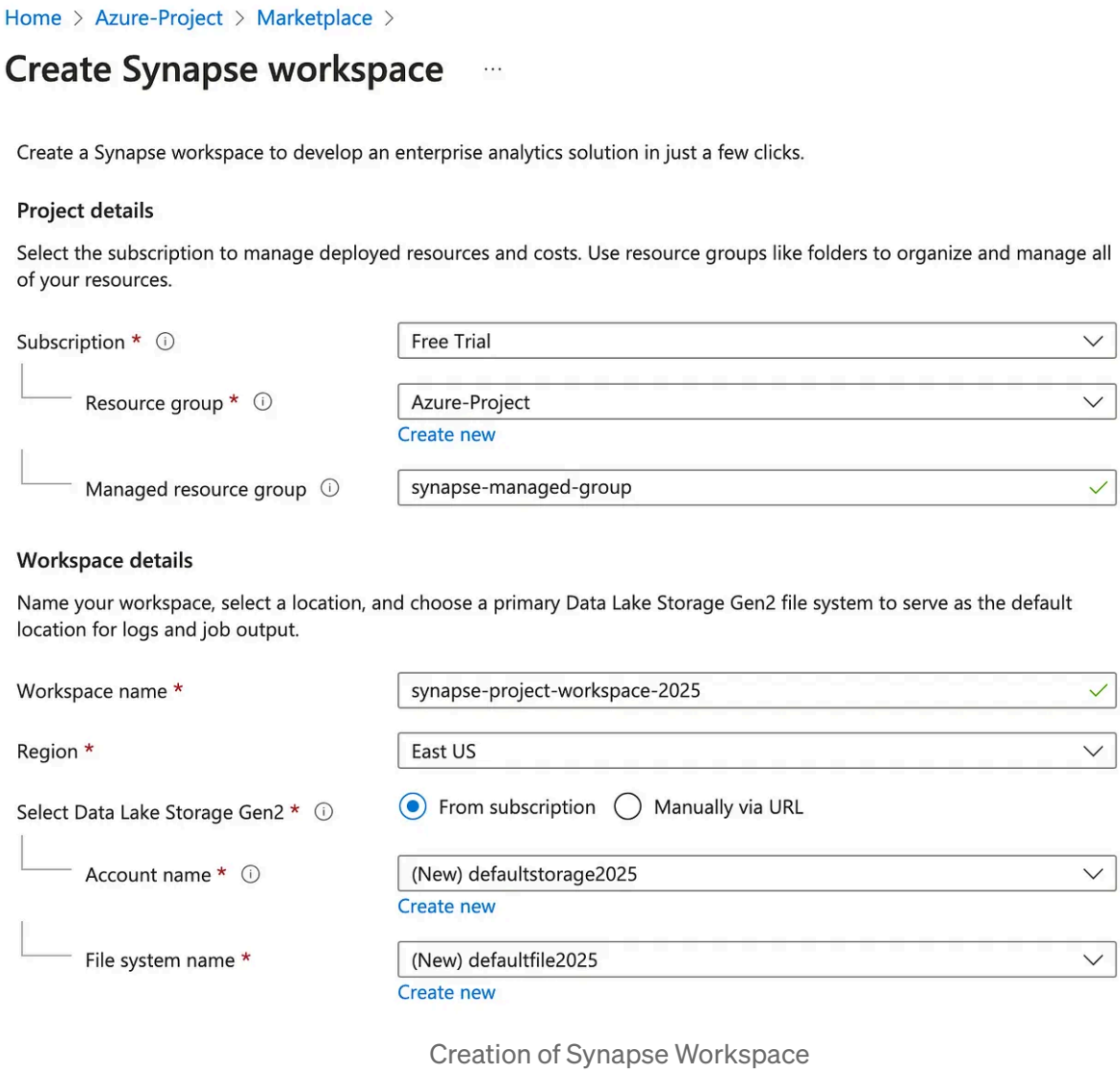
Here I have performed groupby on both **OrderYear** and **OrderMonth** such that the stakeholders can easily hover over a specific month and see the total quantity sold by the distributors.

SCENARIO 3:

Here, we can see that most of the Stock manufacturing in the year 2004 were did in first few months and the Stock manufacturing in the year 2003 were did in last months of the year.

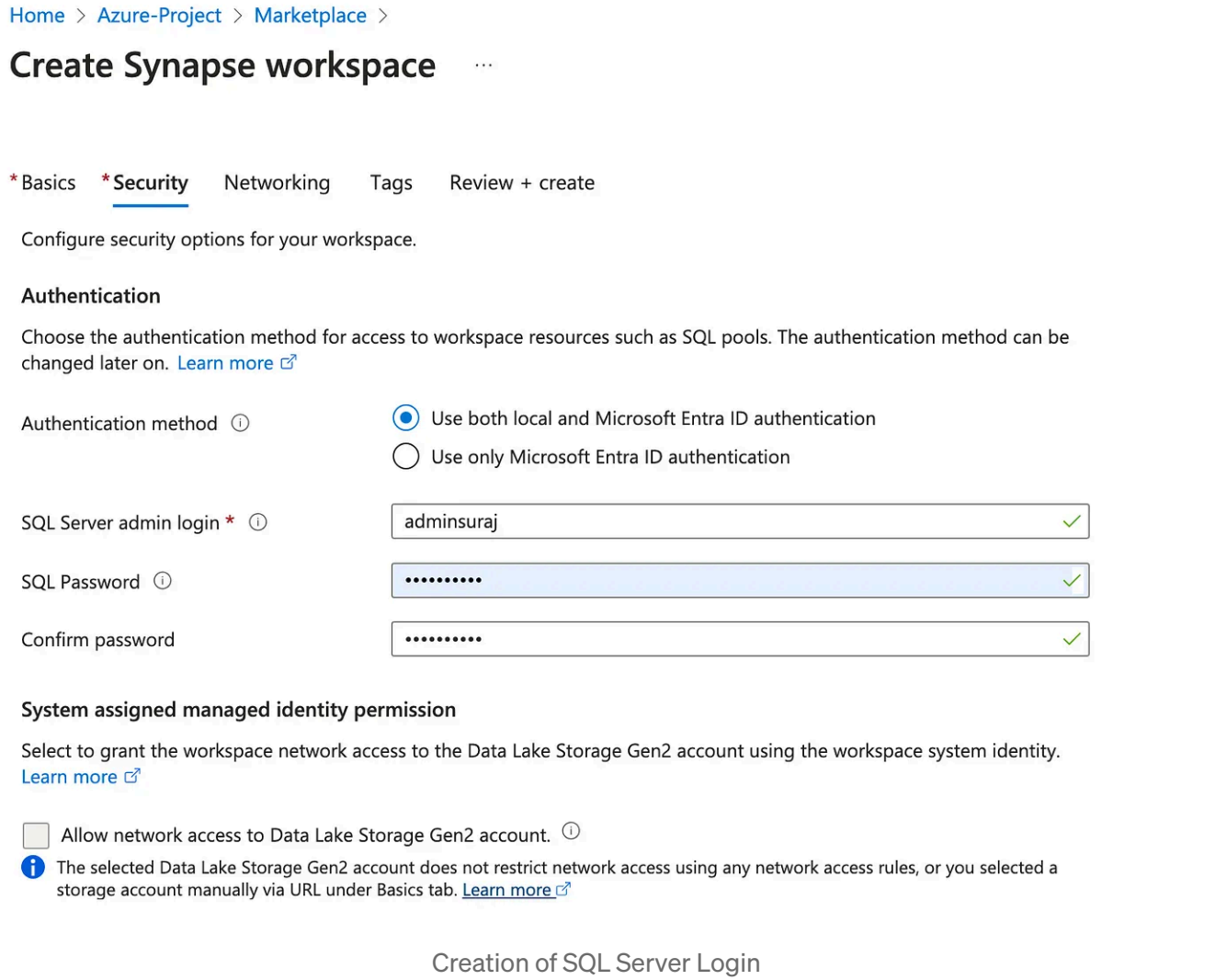
The **SILVER** layer of the **Medallion** architecture is now completed where all the Transformation and Visualisation is done in Databricks and is written into the silver container.

For the gold layer presentation, I will be using Azure Synapse Analytics.

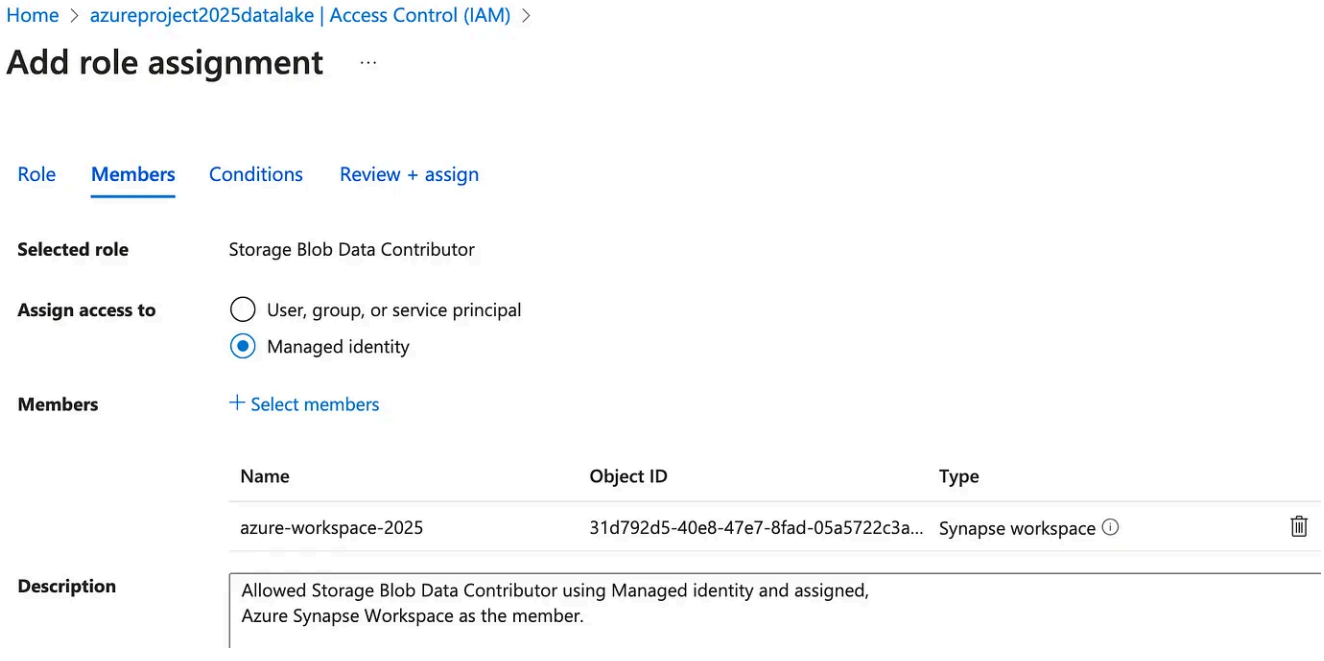


Creation of Synapse Workspace

I have created a default Storage Account and default File System which will be using by Synapse Analytics.



Creation of SQL Server Login

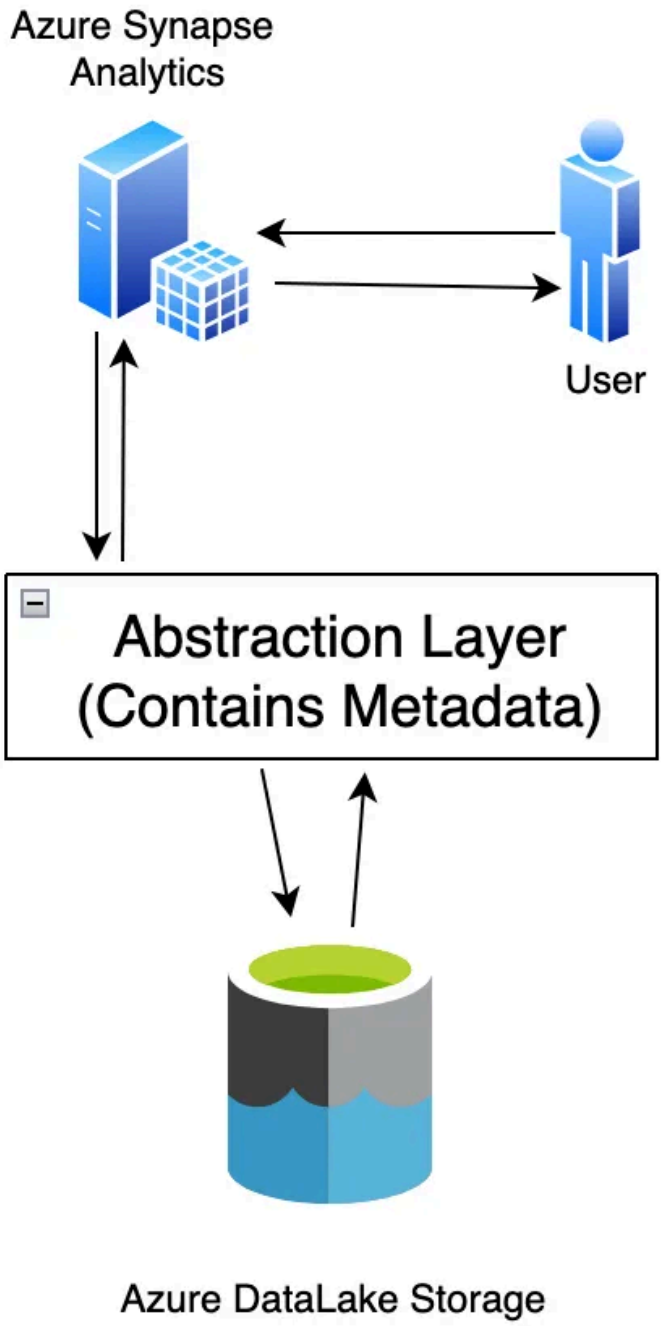


Creation of Managed Identity

Now, I have created a managed identity under the Storage blob data contributor.

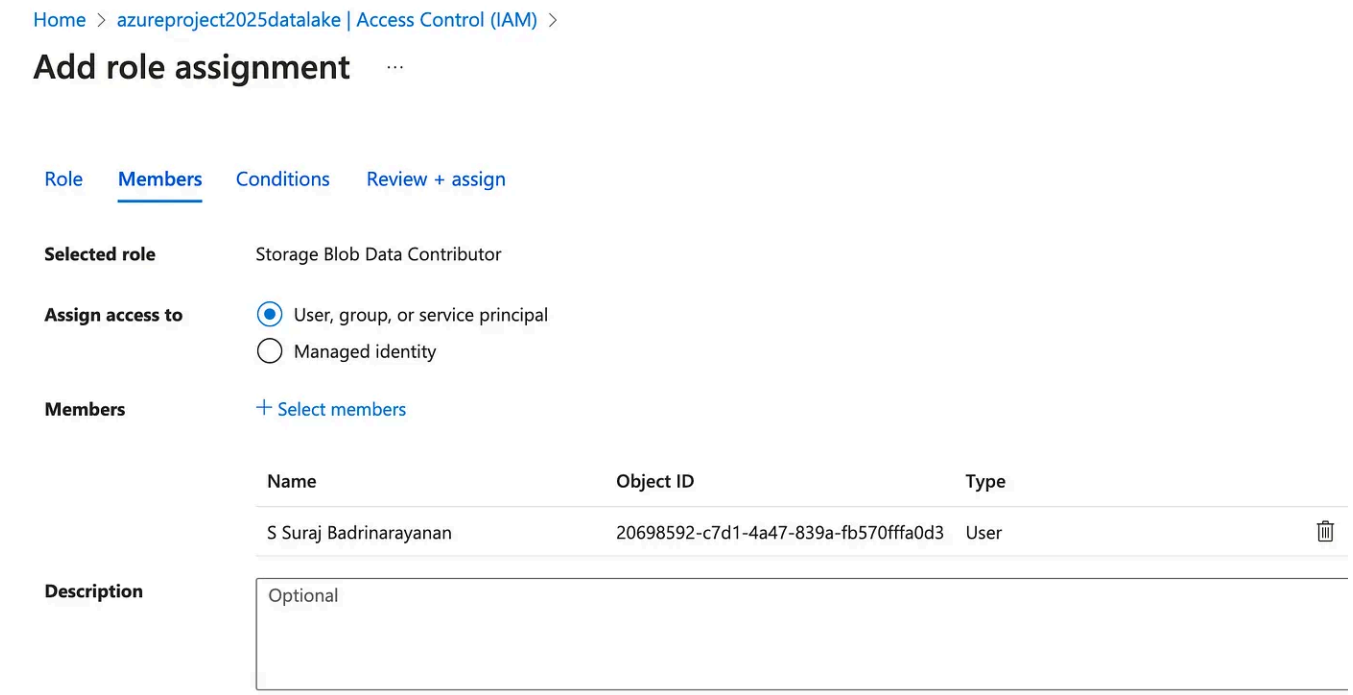
Managed Identity helps in linking of Azure Resources with each other. Now Synapse Analytics can access data from ADLS.

Azure Synapse Analytics helps us in implementing **lakehouse** concept.



The above diagram represents an **abstraction layer** over Azure Data Lake Storage, allowing SQL Server and users to query data using **metadata**.

Synapse Analytics retrieves data from Azure Data Lake Storage (ADLS) and enables the implementation of SQL queries, advanced analytics, and other functionalities, making it easier to process, analyse, and visualise large-scale data efficiently.



IAM to Query the Data as a User

```
SELECT * FROM
    OPENROWSET (
        BULK 'https://azureproject2025datalake.dfs.core.windows.net/silver/AdventureWorks_Calendar/',
        FORMAT = 'PARQUET'
    ) as query1
```

Results Messages

View Table Chart Export results

Search

Date	Month	Year
2015-01-01	1	2015
2015-01-02	1	2015
2015-01-03	1	2015
2015-01-04	1	2015

I have now queried the data in ADLS, using SQL Syntax and displayed the data in parquet file format in the form of a table.

OPENROWSET() function created an abstraction layer on top of ADLS and made me perform SQL Queries on parquet file format.

Now, I will create a gold schema where I will store all the Views inside it.

```sql
-- CREATE VIEW CALENDAR
CREATE VIEW gold.calendar AS
SELECT
    *
FROM
    OPENROWSET(
        BULK 'https://azureproject2025datalake.blob.core.windows.net/silver/AdventureWorks_Calendar/',
        FORMAT = 'PARQUET'
    ) AS QUER1;

-- CREATE VIEW CUSTOMERS
CREATE VIEW gold.customers AS
SELECT
    *
FROM
    OPENROWSET(
        BULK 'https://azureproject2025datalake.blob.core.windows.net/silver/AdventureWorks_Customers/',
        FORMAT = 'PARQUET'
    ) AS QUER1;
```

Successfully Created Views for all Data in the silver Layer

Now the **stakeholders**, **managers** or **data analysts** can query the data as if it was a SQL Server, but in reality, all the data are retrieved from ADLS using an Abstraction Layer.

Now, I have successfully created the gold schema in Azure Synapse Analytics and completed the requirements in accordance with the **Medallion** Architecture.

Azure Synapse Analytics simplifies data processing by allowing you to query large datasets directly from Azure Data Lake without the need for expensive SQL databases. Synapse makes it easy to run SQL queries on big data, providing fast insights while keeping expenses low, making it a powerful tool for modern businesses.

I hope you found this guide helpful! If you enjoyed the blog or have any thoughts to share, feel free to leave a like or drop a comment on my Medium blog — I'd love to hear your feedback!

Edit profile