

Azure End to End Data engineering project (github data)

1) Setup Azure: Use new email get 30 days free with \$200 credit

2) Create resource group → storage account

Storage account name * awstorageaccount2

Region * (Asia Pacific) Southeast Asia Deploy to an Azure Extended Zone

Primary service Azure Blob Storage or Azure Data Lake Storage Gen 2

Performance * Standard: Recommended for most scenarios (general-purpose v2 account)
Premium: Recommended for scenarios that require low latency.

Redundancy * Locally-redundant storage (LRS)

Previous Next Review + create Give feedback

Minimum TLS version Version 1.2

Permitted scope for copy operations (preview) From any storage account

Hierarchical Namespace

Hierarchical namespace, complemented by Data Lake Storage Gen2 endpoint, enables file and directory semantics, accelerates big data analytics workloads, and enables access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace

Access protocols

Blob and Data Lake Gen2 endpoints are provisioned by default. [Learn more](#)

Enable SFTP

Enable network file system v2

Previous Next Review + create Give feedback

Network connectivity

You can connect to your storage account either publicly, via public IP addresses or service endpoints, or privately, using a private endpoint.

Network access * Enable public access from all networks
 Enable public access from selected virtual networks and IP addresses
 Disable public access and use private access
Enabling public access from all networks might make this resource available publicly. Unless public access is required, we recommend using a more restricted access type. [Learn more](#)

Private endpoint

Create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the storage account or private link center.

Previous Next Review + create Give feedback

Storage account name awstorageaccount2

Location (Asia Pacific) Southeast Asia

Primary endpoint https://awstorageaccount2.blob.core.windows.net

Previous Next Review + create Give feedback

3) Create Containers (bronze, silver, gold) inside storage account

The screenshot shows the 'New container' dialog box on the right side of the Azure portal. The 'Name' field is set to 'bronze'. The 'Anonymous access level' dropdown is set to 'Private (no anonymous access)'. A note below states: 'The access level is set to private because anonymous access is disabled on this storage account.' At the bottom, there is a 'Create' button.

The screenshot shows the 'Containers' list in the Azure Storage account. It lists four containers: '\$logs', 'bronze', 'gold', and 'silver'. The 'gold' and 'silver' containers were created in the previous step. The 'bronze' container was created in this step. The 'gold' and 'silver' containers have 'Private' anonymous access level, while 'bronze' has 'Available' anonymous access level.

Name	Last modified	Anonymous access level	Lease state
\$logs	4/26/2025, 8:27:21 PM	Private	Available
bronze	4/26/2025, 11:01:58 PM	Private	Available
gold	4/27/2025, 12:46:20 PM	Private	Available
silver	4/27/2025, 12:46:10 PM	Private	Available

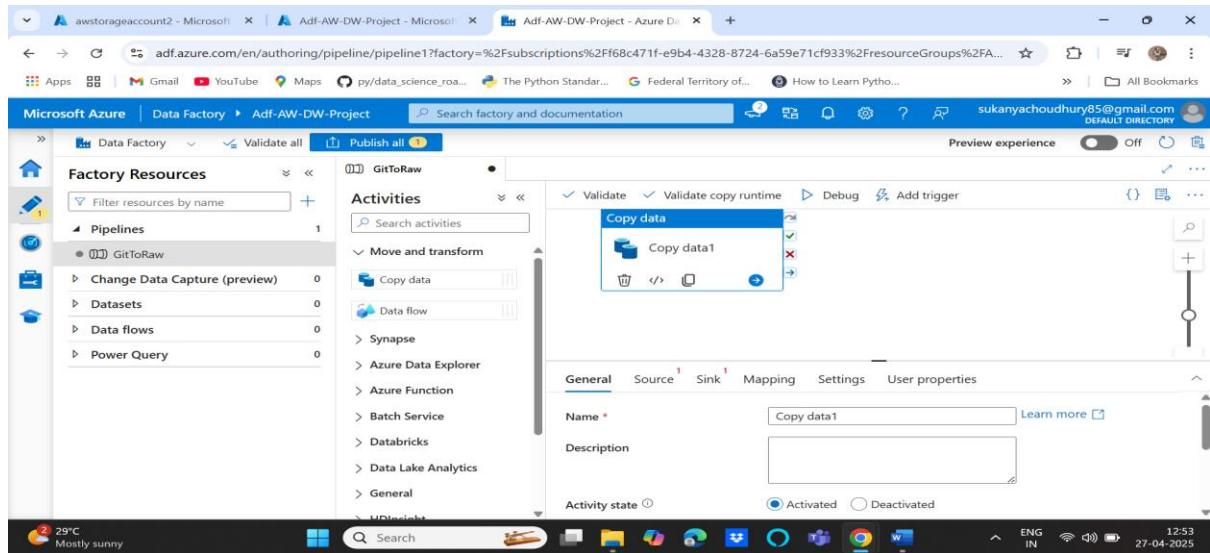
Phase 1 : Data Extraction from github using Data Factory pipelines

1) Create Data Factory “Adf-AW-DW-Project” as below:

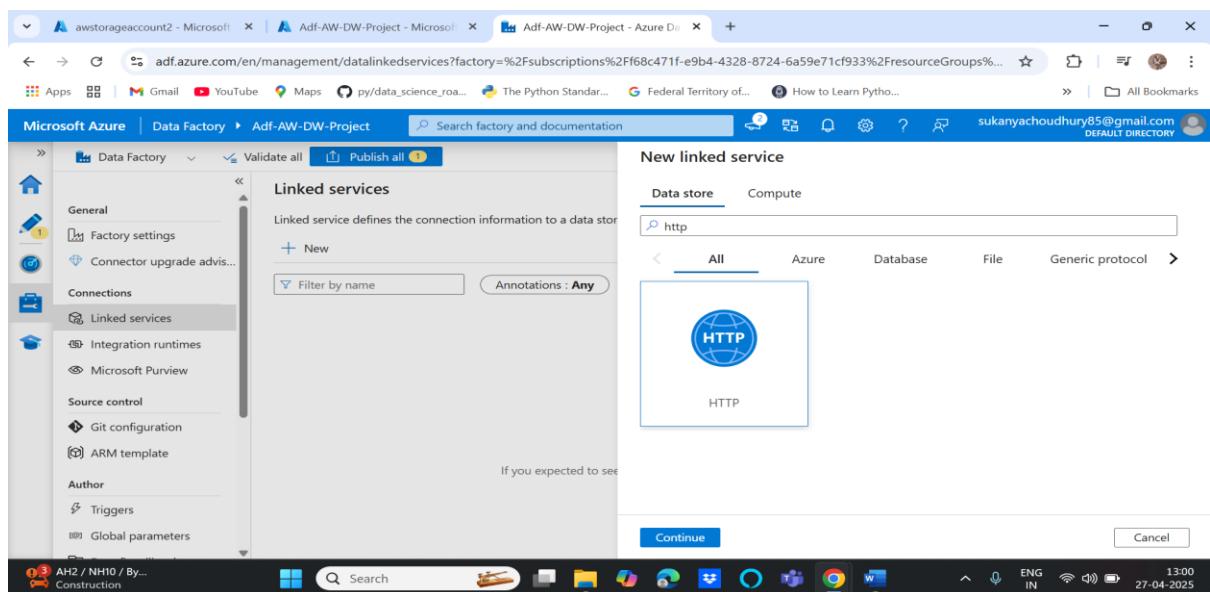
The screenshot shows the 'Create Data Factory' wizard. In the 'Subscription' section, 'Azure subscription 1' is selected. In the 'Resource group' section, 'AW_DW_Project' is selected. Under 'Instance details', the 'Name' is 'Adf-AW-DW-Project', 'Region' is 'Southeast Asia', and 'Version' is 'V2'. At the bottom, there are 'Previous', 'Next', and 'Review + create' buttons.

2) Create a pipeline in data factory

DF → Author → Pipeline → new pipeline and then move and transform → Copy data



3) Create linked services for the pipeline to work e.g. for source data source create http linked service to fetch the github data set and give base url as the github main url



This screenshot shows the detailed configuration for the 'HTTP' linked service. It includes fields for 'Description', 'Connect via integration runtime' (set to 'AutoResolveIntegrationRuntime'), 'Base URL' (set to 'https://raw.githubusercontent.com'), a note about trusting the URL, 'Server certificate validation' (set to 'Enable'), and 'Authentication type' (set to 'Anonymous'). At the bottom are 'Create', 'Back', 'Test connection', and 'Cancel' buttons.

For destination(sink), create Azure Data Lake Gen2 as the linked service

New linked service

Connect via integration runtime: AutoResolveIntegrationRuntime

Authentication type: Account key

Account selection method: From Azure subscription (selected)

Azure subscription: Azure subscription 1 (f68c471f-e9b4-4328-8724-6a59e71cf933)

Storage account name: awstorageaccount2

Test connection: Connection successful

4) Create source and sink dataset inside copy pipeline

For source its http->CSV file format as it's http type data source with a file format of csv then give the linked service created above with the relative URL as below

Select format

Choose the format type of your data

Avro

Binary

DelimitedText

Excel

JSON

ORC

Name: ds_http_AW

Linked service: HttpGitLinkedService

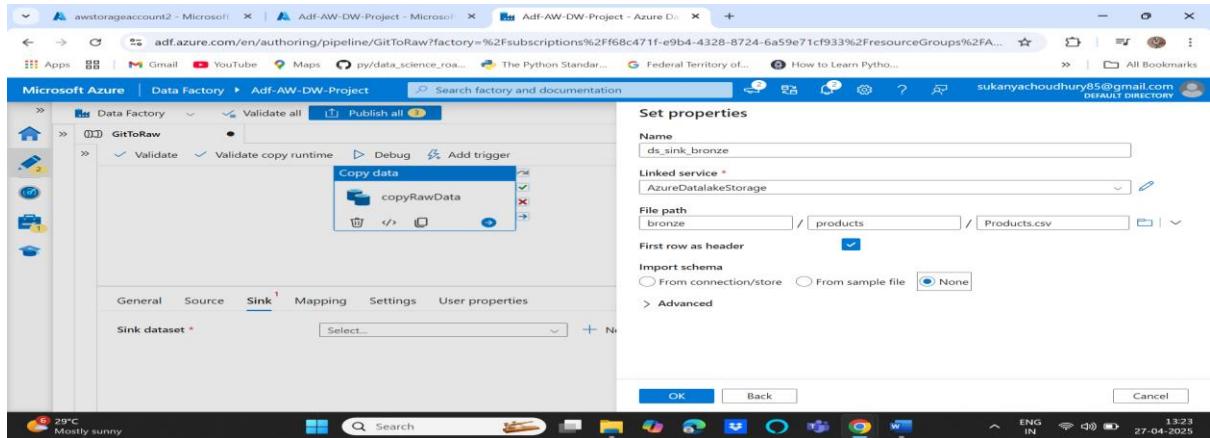
Relative URL: anshlambaodgit/Adventure-Works-Data-Engineering-Project/refs/heads/main/Data/Ad

First row as header:

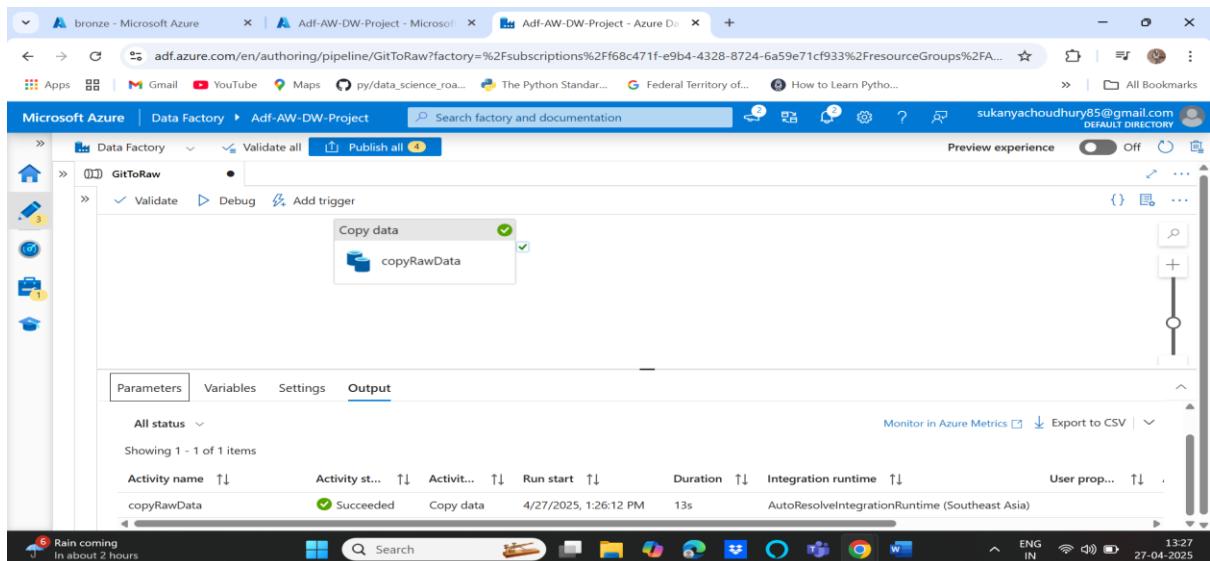
Import schema: From connection/store From sample file None

OK Back Cancel

For sink again same process but the datasource type will azure data lake storage gen2 and format is csv. Then again select the linked service created for destination with the folder where you need to store it. Eg: here, bronze container/products and Products.csv



Validate and debug to run the pipeline and then publish all to save the changes



Products.csv is created under bronze container:

Home > awstorageaccount2 | Containers >

bronze Container

Search Upload Add Directory Refresh Rename Delete Change tier Acquire lease Break lease Give feedback

Overview Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bronze / products

Diagnose and solve problems
Access Control (IAM)

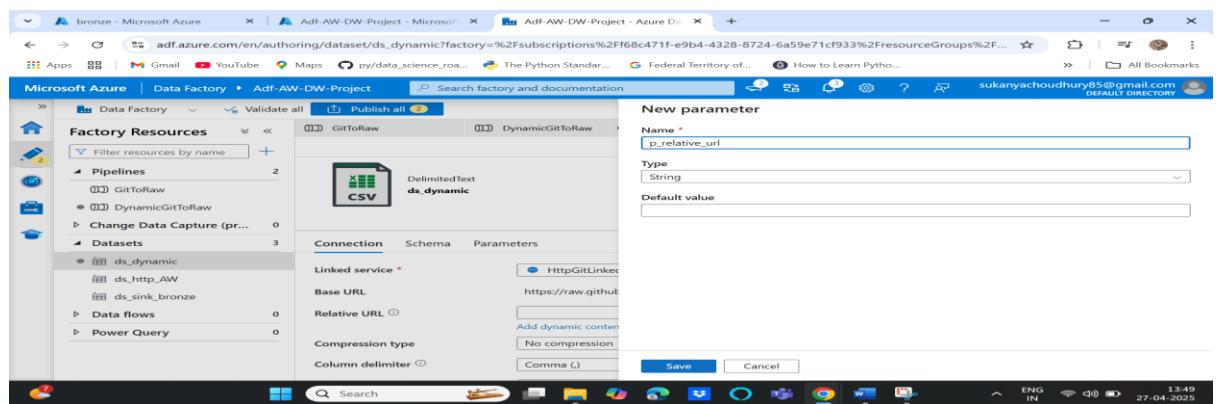
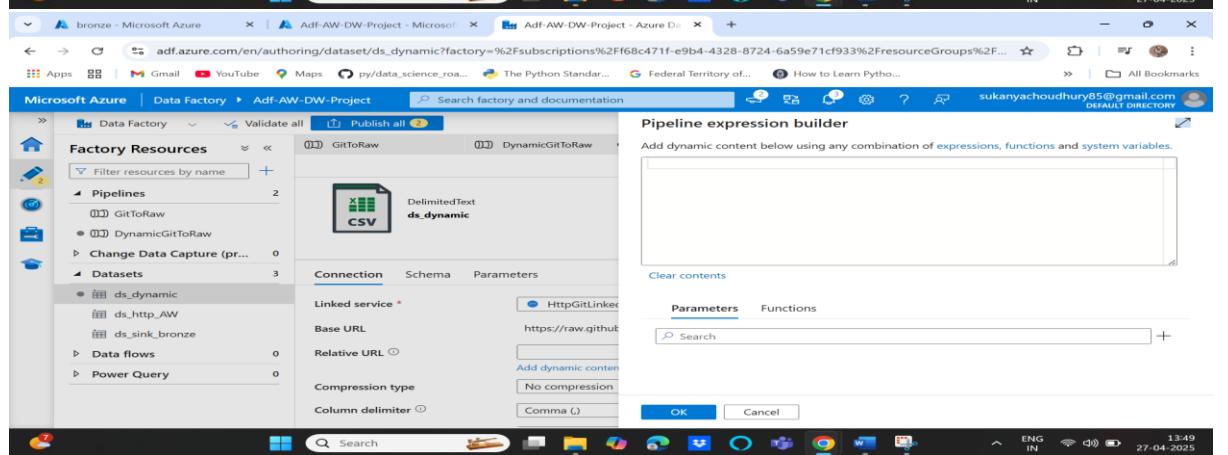
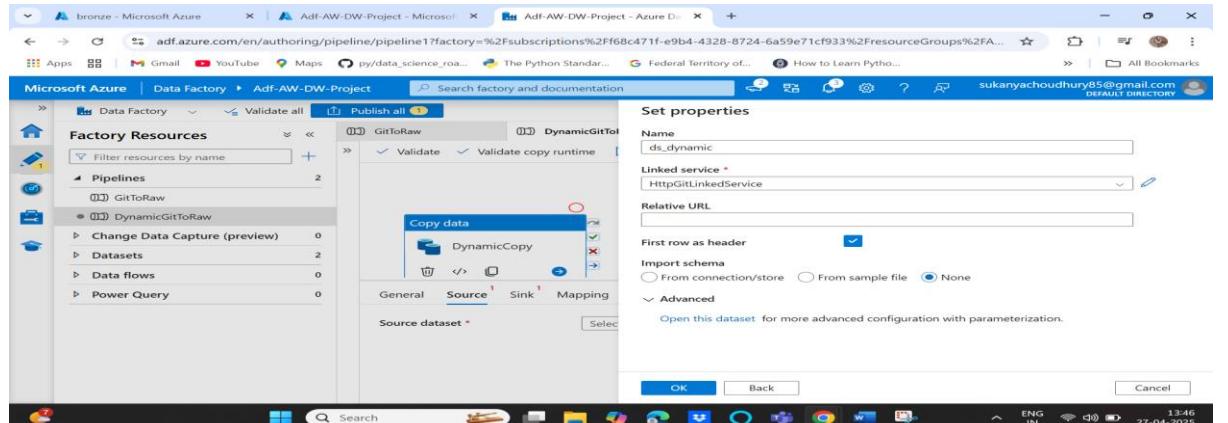
Search blobs by prefix (case-sensitive) Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size
[..]					
Products.csv	4/27/2025, 1:26:24 PM	Hot (Inferred)		Block blob	56.7

- 5) Dynamic pipelines to fetch multiple files using iteration and conditionals under author tab in Df. Create Parameters for the dynamic part of the pipeline e.g.: the relative URL for each file will change , the destination folder as well as the filename will change so we need to create 3 parameters and use a parameterized copy activity and a single data source to fetch all the files.

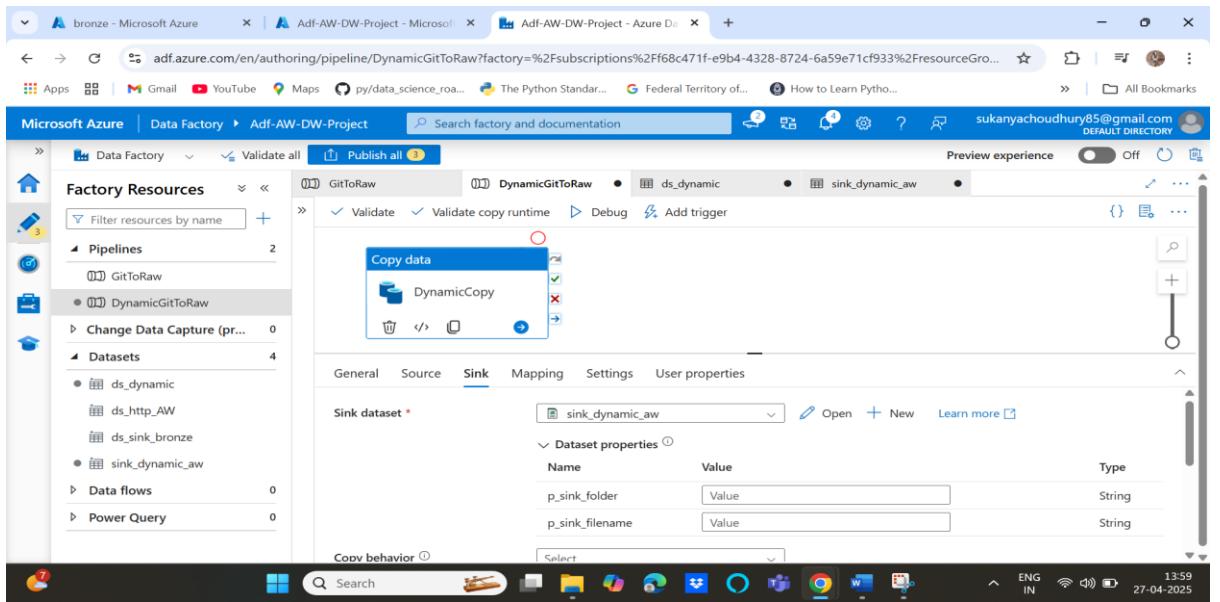
i) DF → Author → Pipeline → New Pipeline → Copy activity → source dataset → http → csv → source linked service create above in step 3 as the base URL is same for all so same linked service will be used

Instead of giving a hardcoded value for relative URL, we need to use parameters so for that click advanced → open this data set → Relative URL → add dynamic content → parameters—new parameter and then select the parameter on expression folder



Similarly for sink dataset , create parameters for folder and filename by going to sink tab of copy activity → add new → data lake gen2 → csv → select linked service in step 3 → instead of hardcoding the folders and filename → advanced → open this data set → folder, file → add dynamic content and create two parameters as shown below:

Ultimately copy activity looks like below with dynamic source and sink data sources:



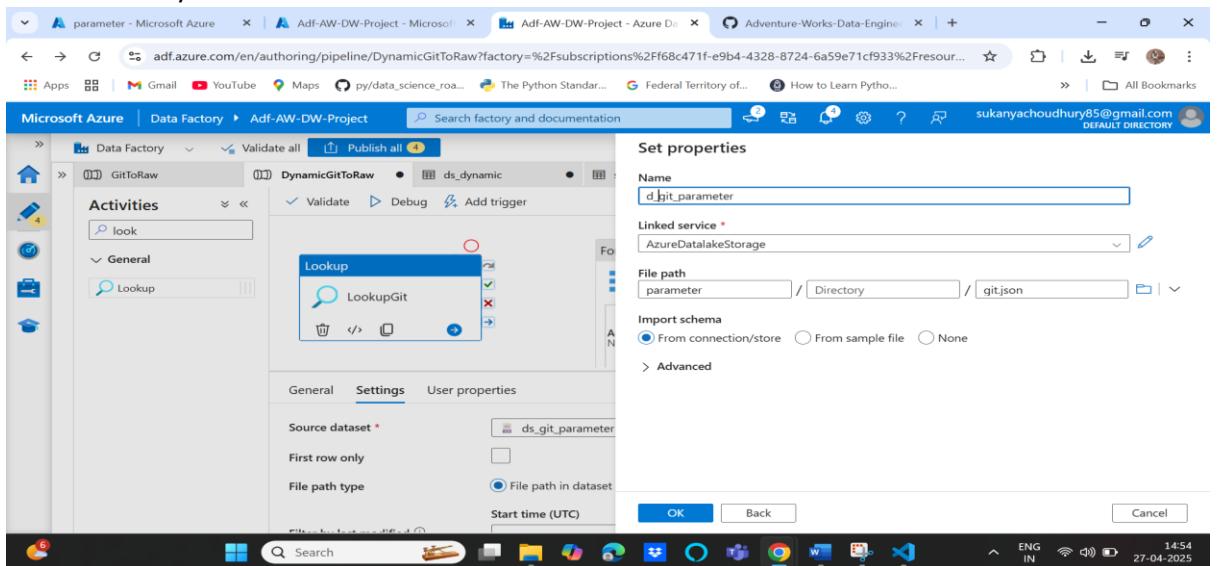
ii) Now, we need to pass values to these parameters:

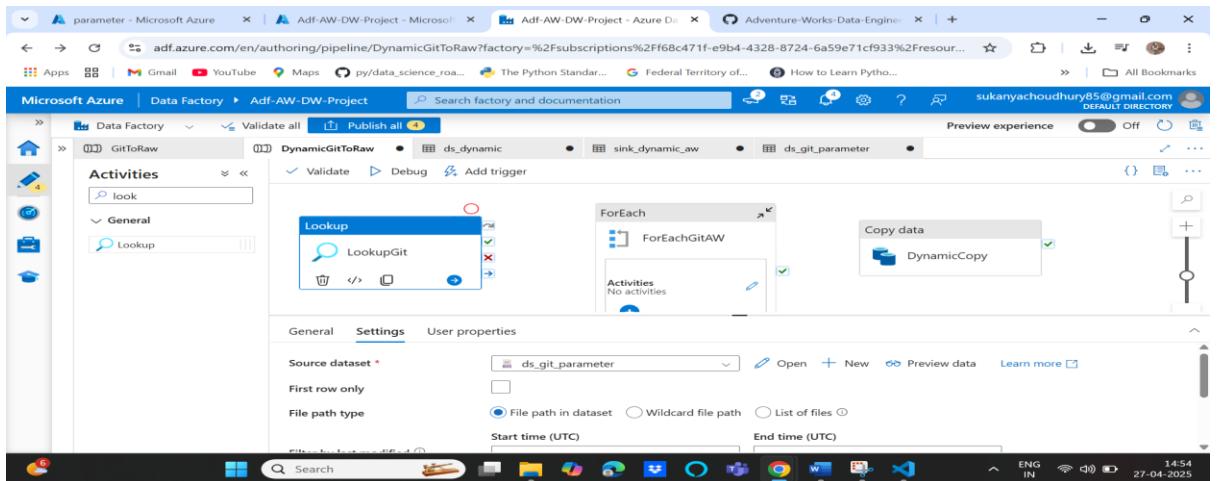
Select the pipeline under activities → Iteration and conditionals → select for each activity and add it to the pipeline

In settings of the foreach activity pass a json file with a dictionary of parameters and their values in the items tab.

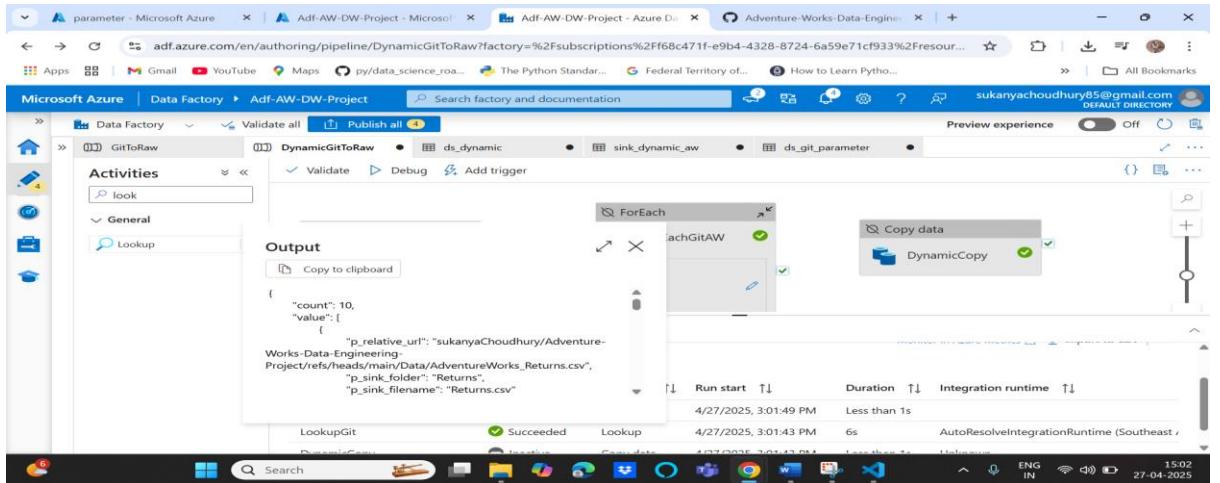
To do that first save the json file e.g. git.json inside a container in the storage account then create a lookup activity to get the git.json data by using a data lake → json data set type in the lookup activity and linked service is same as before as its using the same storage account and then select the folder and json file to connect

Lookup activity will fetch the json data which has parameters that we need to pass into the foreach activity

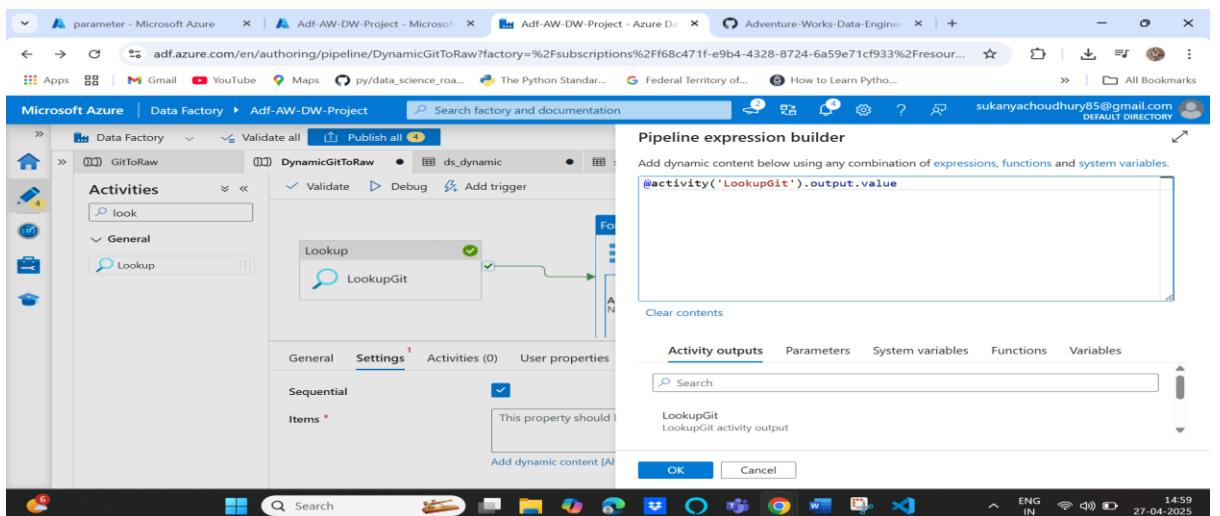


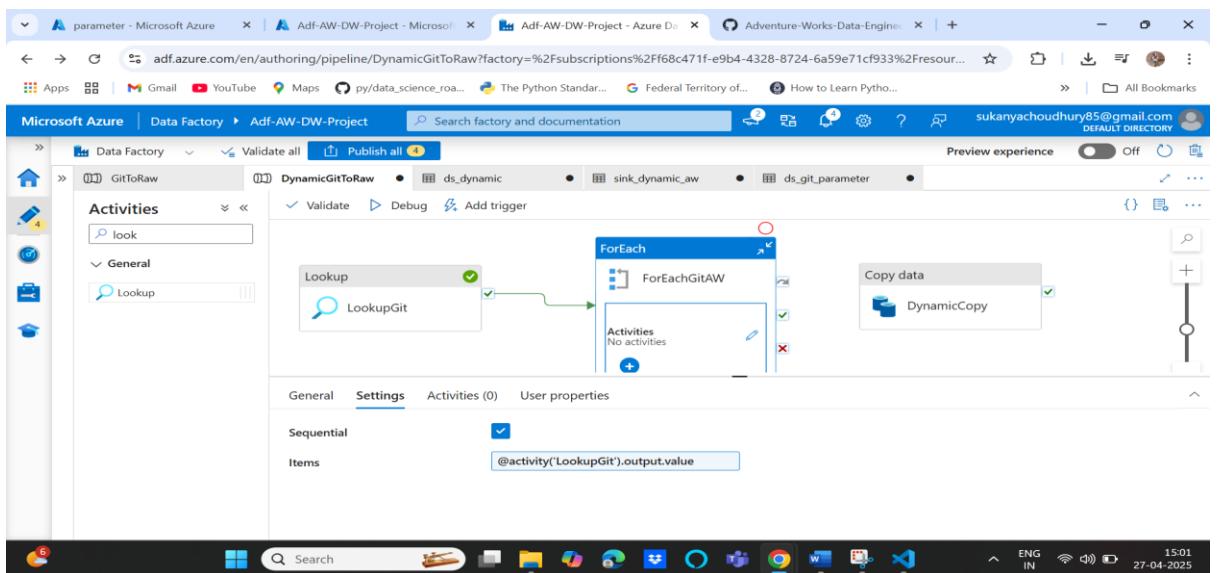


After this debug and test this particular activity only by disconnecting others, You can see that it's passed as a dictionary and all the values are under the value key passed as a list so we need to pass this value key so that the list is passed as parameter:

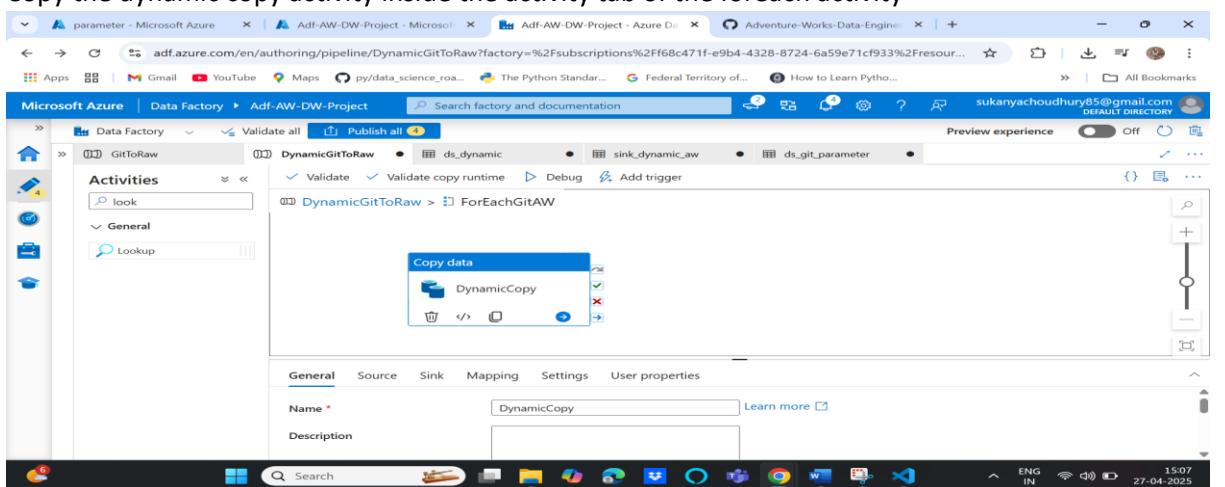


Next is to connect lookup git on success with the foreach activity and then in items, add dynamic content and select the lookupgit output and add .value to it since we need to get the value array:

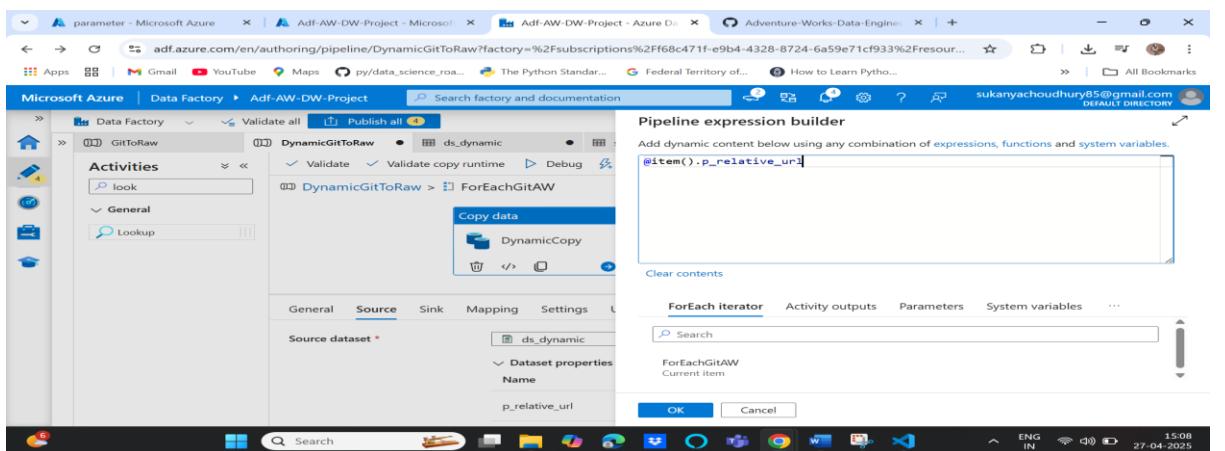


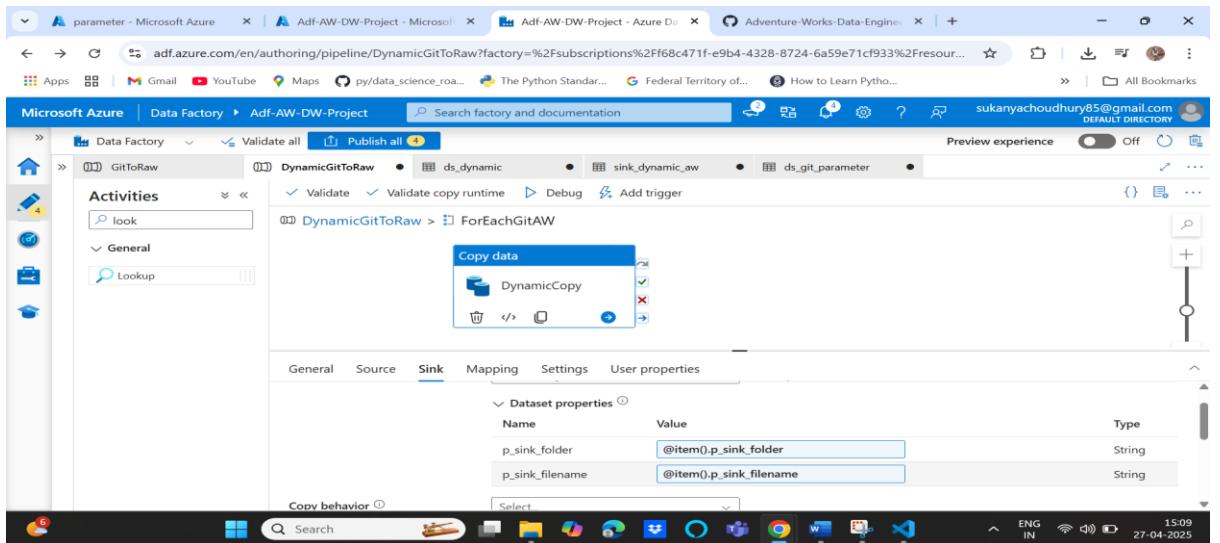


Copy the dynamic copy activity inside the activity tab of the foreach activity

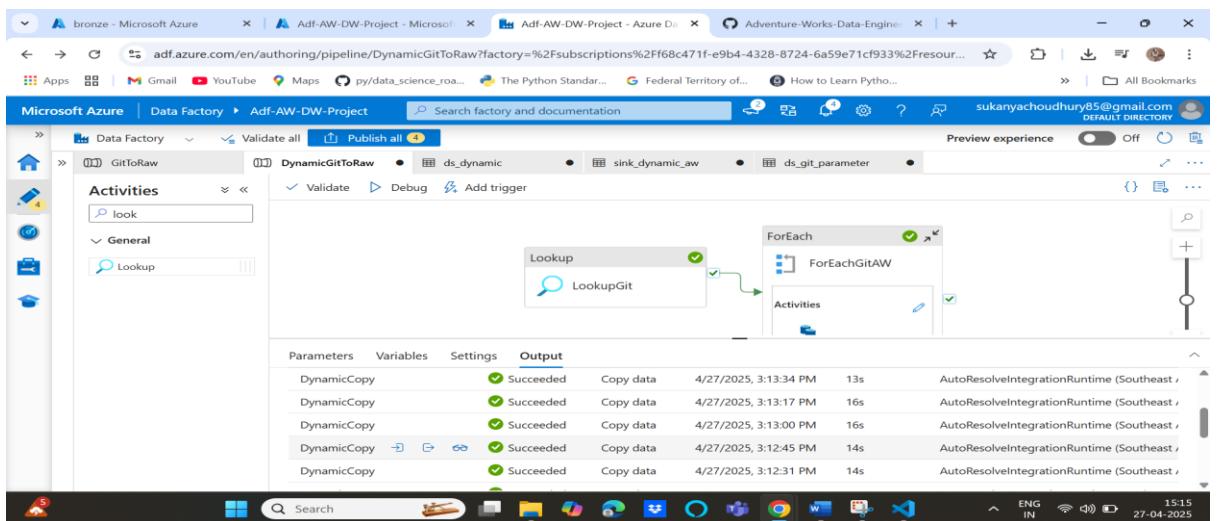


Then pass the values for each source and sink parameters:

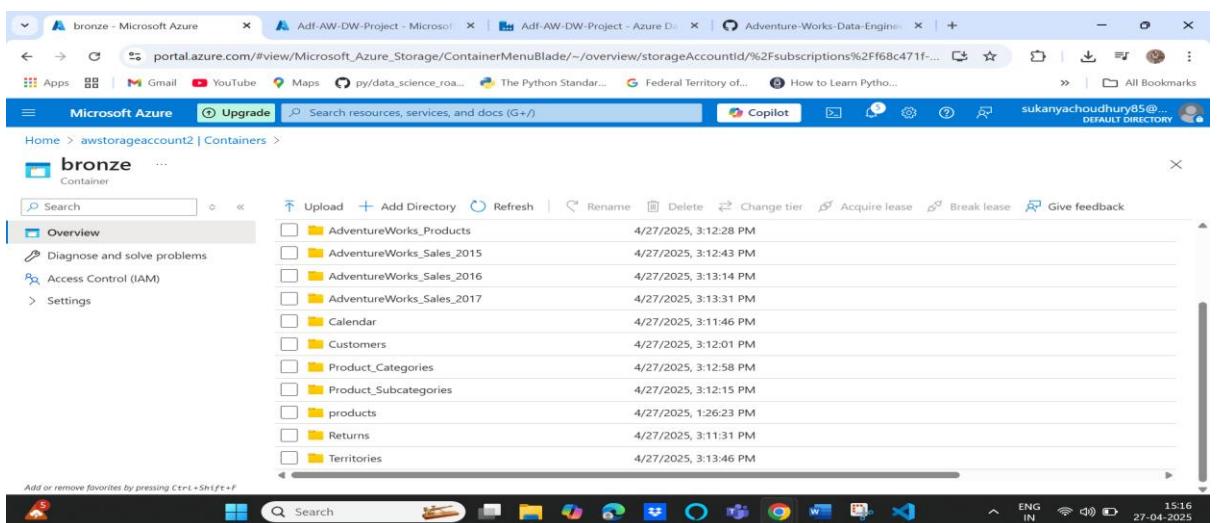




After debug the pipeline and it should be able to fetch all 10 items

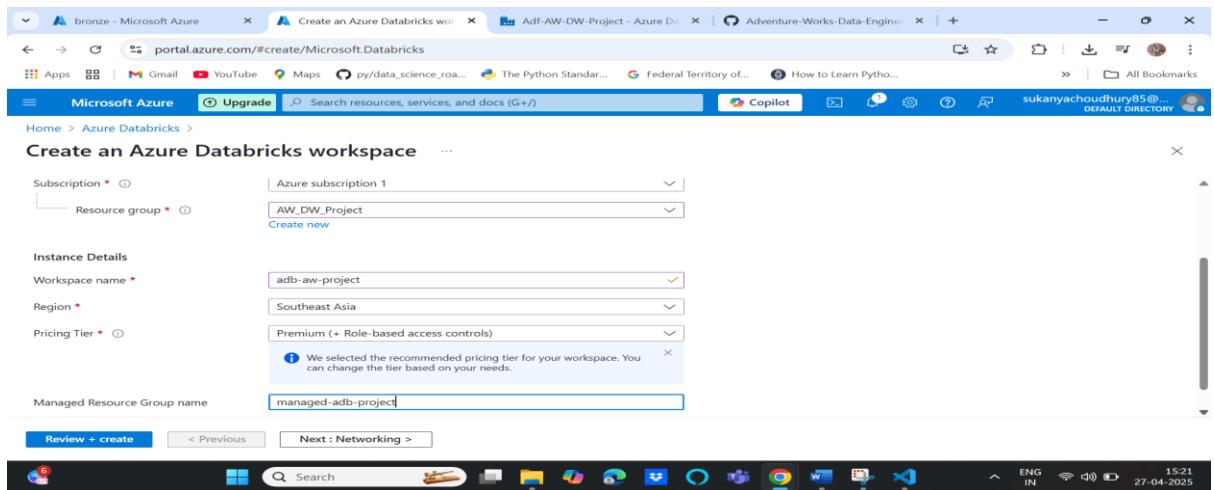


Validate the bronze container to see all files:

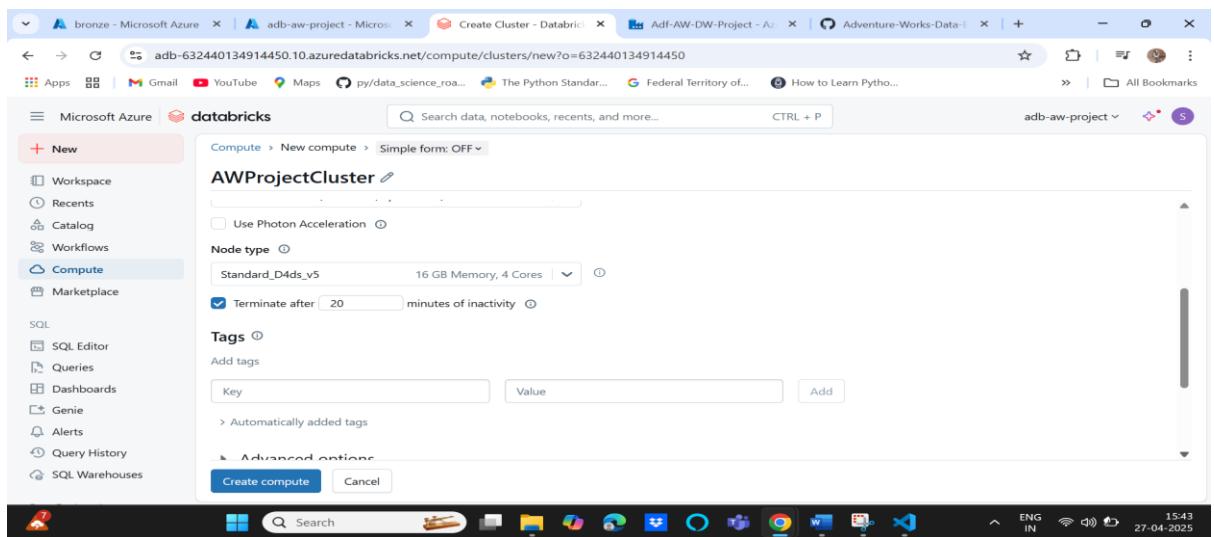
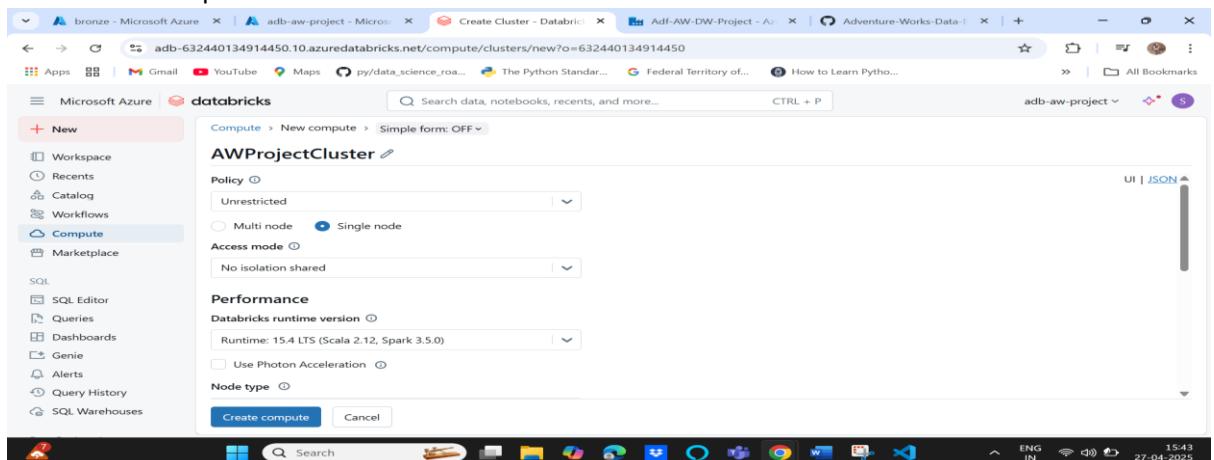


Phase 2: Data manipulation using Databricks and load into silver layer

1) Create Databricks, "adb-aw-project":



2) Create a new compute:



- 3) Create app registration for databricks to access data from our bronze layer in Datalake

Microsoft Entra Id → App Registrations

The screenshot shows the Microsoft Azure portal interface. The left sidebar has 'App registrations' selected under 'Default Directory'. The main area displays a message about the deprecation of ADAL and AAD Graph. Below it, there are tabs for 'All applications', 'Owned applications', 'Deleted applications', and 'Applications from personal account'. A search bar and filter options are present. At the bottom, there are buttons for 'View all applications in the directory' and 'View all applications from personal account'.

The screenshot shows the 'Register an application' page. It requires entering a 'Name' (awproject_app), which is highlighted. Below it, 'Supported account types' are listed with 'Accounts in this organizational directory only (Default Directory only - Single tenant)' selected. A note states that by proceeding, you agree to the Microsoft Platform Policies. A 'Register' button is at the bottom.

Save the details for this app key: App ID and Directory ID which would be needed later on

- 4) Create secret key and save its value to be used as secret key in Jupiter notebook

The screenshot shows the 'Certificates & secrets' section for the 'awproject_app'. It lists 'Client secrets (0)'. A 'New client secret' button is visible. On the right, a 'Add a client secret' dialog is open, asking for a 'Description' (awproject) and an 'Expires' date (Recommended: 180 days (6 months)).

5) Assigning contributor role to a storage account application

Select storage account → access control(IAM) → add a role → add role assignment

→ Storage blob data contributor → select members → application name(awproject_app) → Review+assign

Name	Description	Type	Category	Details
Defender CSPM Storage Data Scanner	Grants access to read blobs and files. This role is used by the data scanner of Defender CSPM.	BuiltinRole	None	View
Defender for Storage Data Scanner	Grants access to read blobs and update index tags. This role is used by the data scanner of ...	BuiltinRole	None	View
Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data	BuiltinRole	Storage	View

6) Write transformation code: Create a workspace → folder → new notebook → Add headings using markup from dropdown

SILVER LAYER SCRIPT

```
%md
###DATA_LOADING
```

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

Data access: code taken from **Azure service principal python code from**
<https://learn.microsoft.com/en-us/azure/databricks/connect/storage/azure-storage>

Replace client id, directory id, secret key, storage account appropriately

```
spark.conf.set("fs.azure.account.auth.type", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type", "org.apache.hadoop.fs.azure.OAuth2ClientCredentialsProvider")
spark.conf.set("fs.azure.account.oauth2.client.id", "7c3782cf-1c0f-4852-95d3-fa514b65cc4d")
spark.conf.set("fs.azure.account.oauth2.client.secret", "Lfd8Q-YNE0jIPsFFCA2-aJHnNKzpsEgQOMBL7")
spark.conf.set("fs.azure.account.oauth2.client.endpoint", "https://login.microsoftonline.com/6be32f9d-dc44-4bd9-8c13-e01cdf1a5bcc/oauth2/token")
```

DATA LOADING

7) Reading data from bronze layer

```
calendar=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/Calendar')
Products=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/AdventureWorks_Products')
Sales_2015=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/AdventureWorks_Sales_2015')
Sales_2016=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/AdventureWorks_Sales_2016')
Sales_2017=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/AdventureWorks_Sales_2017')
Customers=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/Customers')
Returns=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net>Returns')
Territories=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/Territories')
Product_Categories=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/Product_Categories')
Product_SubCategories=spark.read.format('csv').option("header",True).option("inferSchema",True).load('abfss://bronze@awstorageaccount2.dfs.core.windows.net/Product_SubCategories')
```

8) Transformations

The screenshot shows a Databricks notebook titled "TRANSFORMATIONS". The code in cell 8 imports `pyspark.sql.functions` and `pyspark.sql.types`. Cell 9 contains a complex multi-line command to create a DataFrame `df_cal` with various date-related columns like Month, MonthName, Year, DayName, Week, Day, and DayofWeek. The notebook interface includes a sidebar with workspace navigation and a bottom taskbar.

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

df_cal=calendar.withColumn('Month',month(col('Date')))\n    .withColumn('MonthName',monthname(col('Date')))\n    .withColumn('Year',year(col('Date')))\n    .withColumn('DayName',date_format(col('Date'), 'EEEE'))\n    .withColumn('Week',weekofyear(col('Date')))\n    .withColumn('Day',dayofmonth(col('Date')))\n    .withColumn('DayofWeek',dayofweek(col('Date')))
```

The screenshot shows the same Databricks notebook after running the transformation code. Cell 10 displays the resulting DataFrame `df_cal` using the `display` method. The table shows 10 rows of data for January 2015, with columns including Date, Month, MonthName, Year, DayName, Week, Day, and DayofWeek. The table view includes standard data manipulation tools like sorting and filtering.

	Date	Month	MonthName	Year	DayName	Week	Day	DayofWeek
1	2015-01-01	1	Jan	2015	Thursday	1	1	
2	2015-01-02	1	Jan	2015	Friday	1	2	
3	2015-01-03	1	Jan	2015	Saturday	1	3	
4	2015-01-04	1	Jan	2015	Sunday	1	4	
5	2015-01-05	1	Jan	2015	Monday	2	5	
6	2015-01-06	1	Jan	2015	Tuesday	2	6	
7	2015-01-07	1	Jan	2015	Wednesday	2	7	
8	2015-01-08	1	Jan	2015	Thursday	2	8	
9	2015-01-09	1	Jan	2015	Friday	2	9	
10	2015-01-10	1	Jan	2015	Saturday	2	10	

9) Write into silver layer

The screenshot shows the final step of the notebook. Cell 12 contains the command `calendar.write.format('parquet')` followed by `.mode('append').option("path", "abfss://silver@awstorageaccount2.dfs.core.windows.net/Calendar").save()`. The notebook interface shows the command has run 4 minutes ago.

```
calendar.write.format('parquet')\n    .mode('append').option("path", "abfss://silver@awstorageaccount2.dfs.core.windows.net/Calendar").save()
```

10) Customers transformation

The screenshot shows a Databricks notebook titled "silver_layer" running on a Python cluster. The notebook contains two runs:

- Run 16: `Customers=Customers.withColumn('fullName',concat_ws(' ',col('Prefix'),col('FirstName'),col('LastName')))`
- Run 17: `Customers.display()`

The resulting table displays customer names with their first and last names concatenated with a space and their prefix.

allIncome	TotalChildren	EducationLevel	Occupation	HomeOwner	fullName
1	2	Bachelors	Professional	Y	MR. JON YANG
2	3	Bachelors	Professional	N	MR. EUGENE HUANG
3	3	Bachelors	Professional	Y	MR. RUBEN TORRES
4	0	Bachelors	Professional	N	MS. CHRISTY ZHU
5	5	Bachelors	Professional	Y	MRS. ELIZABETH JOHNS...

11) Products Transformation

The screenshot shows a Databricks notebook titled "silver_layer" running on a Python cluster. The notebook contains two runs:

- Run 24: `Products= Products.withColumn('ProductSKU',split(col('ProductSKU'),'-')[0])\n.withColumn('ProductName',split(col('ProductName'),'-')[0])`
- Run 25: `Products.display()`

The resulting table displays product keys and names.

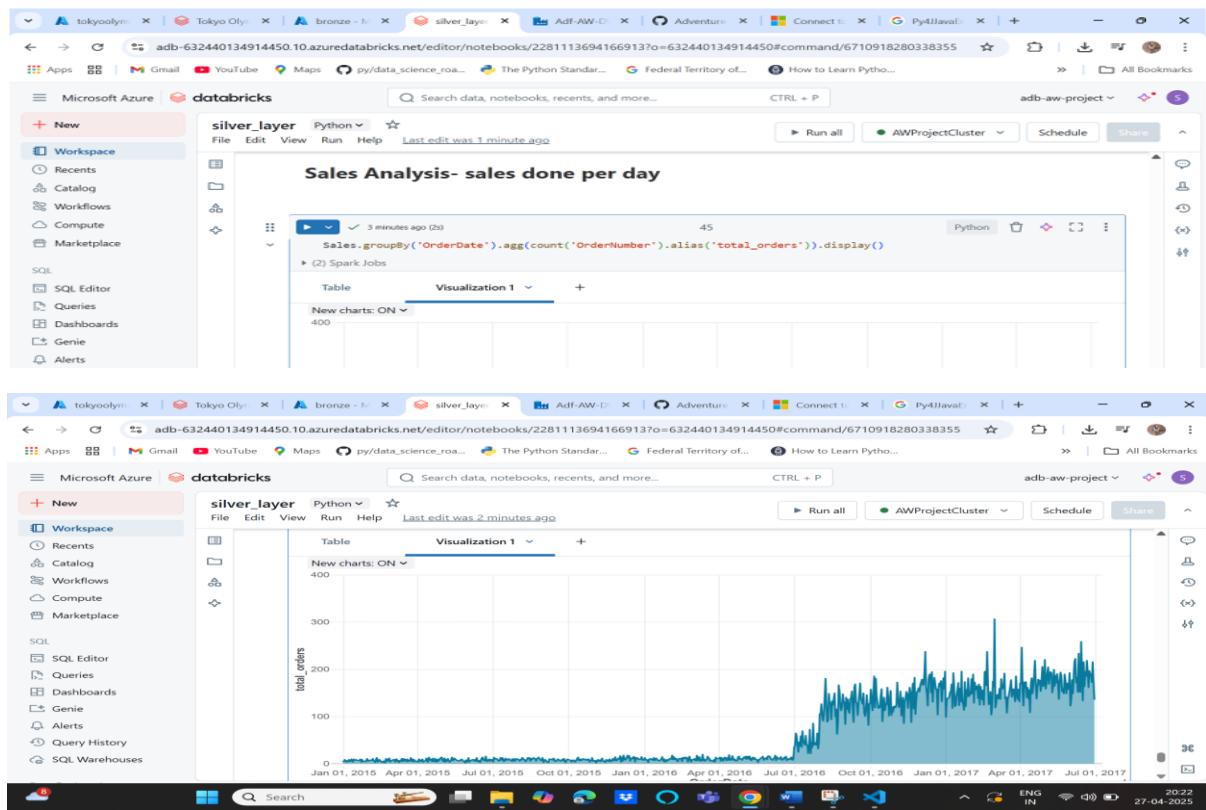
ProductKey	ProductSubCategoryKey	ProductSKU	ProductName	ModelName	Prod...
214	31	HL	Sport-100	Sport-100	Universal

12) Sales Transformation

The screenshot shows a Databricks notebook titled "silver_layer" running on a Python cluster. The notebook contains two runs:

- Run 38: `Sales= Sales.withColumn('StockDate',to_timestamp(col('StockDate')))`
- Run 40: `Sales= Sales.withColumn('OrderNumber',regexp_replace(col('OrderNumber'),'S','T'))`

The resulting table displays sales data with converted stock dates and modified order numbers.

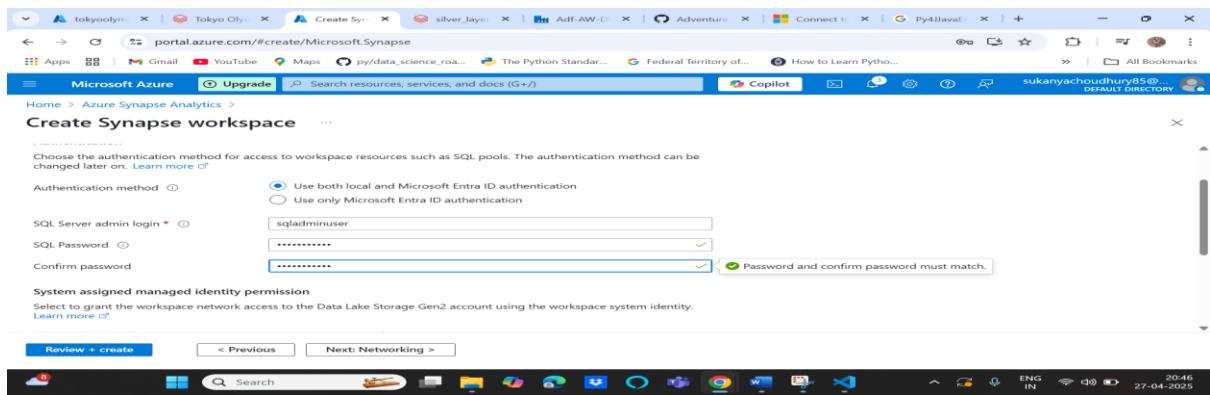


All data after transformation are written into silver container of the data lake

Phase 3: Data Storage into Synapse Analytics

- 1) Create Synapse Analytics Workspace(awproject-sa) and save the DB details, username and pwd

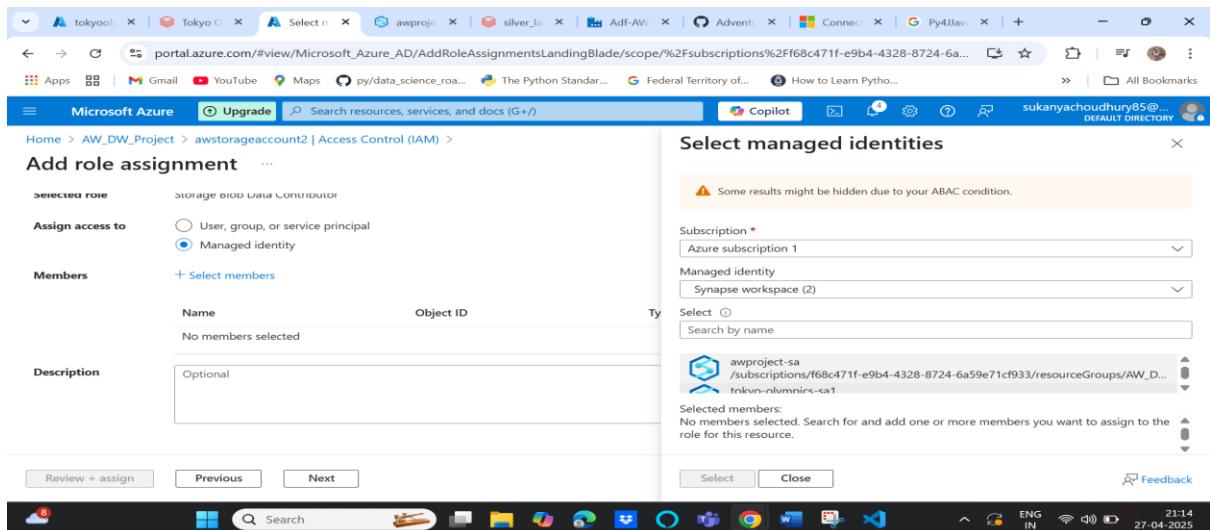
Usually a new storage account is created while creating synapse analytics workspace as shown in data lake storage gen2



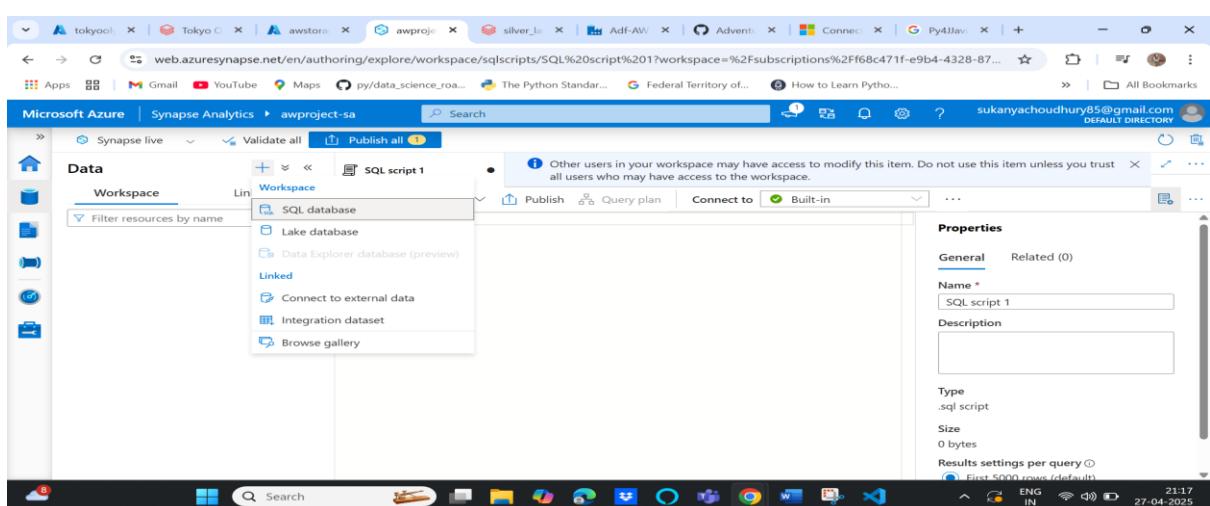
2) System managed identity: allow synapse to connect with data lake to get data

Go to the original/external storage account → IAM → Add role → Storage blob data contributor

→ Next → check Managed Identity → Addmembers → Synapse Workspace



3) Create a synapse database first as we need to run SQL queries on it for data modelling transformations etc. SA workspace → Data tab → + sign → SQL database



- 4) After creating db, go to IAM again on storage account and then give yourself access to access the data

The screenshot shows the 'Add role assignment' page in the Microsoft Azure portal. Under 'Selected role', 'Storage Blob Data Contributor' is selected. Under 'Assign access to', 'User, group, or service principal' is selected. In the 'Members' section, 'Sukanya Choudhury(Guest)' is listed under 'Selected members'. The 'Description' field is optional and empty. At the bottom, there are 'Review + assign', 'Previous', and 'Next' buttons, along with 'Select' and 'Close' buttons.

- 5) Copy the silver data folder url and then use it in SQL script:

The screenshot shows the 'Containers' page for the 'silver' storage account. A specific blob named 'AdventureWorks_Products/part-00000-tid-2010985...' is selected. The 'Properties' section shows the URL as 'https://awstorageaccount2.blob.core.windows.net/silver/AdventureWorks_Products/part-00000-tid-2010985...'. A tooltip indicates the URL has been copied.

And then replace blob in the filename with dfs as we are using data lake and not blob storage to query the parquet data and get tabular data

The screenshot shows the 'Synapse Analytics' workspace for 'awproject-sa'. In the 'Data' section, 'AWsScript' is selected. The query editor contains the following T-SQL script:

```

1 SELECT
2 * FROM
3 OPENROWSET(
4
5 BULK 'https://awstorageaccount2.dfs.core.windows.net/silver/AdventureWorks_Products',
6 FORMAT = 'PARQUET'
7 ) as Products

```

The URL 'https://awstorageaccount2.dfs.core.windows.net/silver/AdventureWorks_Products' is highlighted in red. The results pane shows a table with columns: ProductKey, ProductSubcat..., ProductSKU, ProductName, ModelName, ProductDescri..., ProductColor, Pro...

6) Create Gold Schema so that we can create views

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. In the center, there is a query editor window titled 'AWsaScript' containing the SQL command: '1 CREATE Schema gold;'. To the right of the editor, a 'Properties' panel is open, showing the 'General' tab with the name 'Create Schema'.

7) Create views from parquet data using below script



The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. In the center, there is a query editor window titled 'Gold_views' containing the following SQL script:

```
1 --CREATE VIEW CALENDAR
2
3 CREATE VIEW gold.calendar
4 AS
5
6 SELECT
7 *
8 FROM
9 OPENROWSET(
10
11      BULK 'https://awstorageaccount2.dfs.core.windows.net/silver/Calendar/',
12      FORMAT = 'PARQUET'
13 ) as Calendar
14
```

8) Now next step is to create external tables from the views by creating external sources

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. In the center, there is a query editor window titled 'Gold_views' containing the SQL command: '1 Create MASTER KEY ENCRYPTION BY PASSWORD = 'Susmita\$123''. To the right, a 'Properties' panel is open for 'SQL script 1', showing the 'General' tab with the name 'SQL script 1'.

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. In the center, there is a query editor window titled 'Gold_views' containing the following SQL script:

```
1 Create DATABASE SCOPED CREDENTIAL cred_sukanya
2 WITH
3     IDENTITY='Managed Identity'
```

```

5 CREATE EXTERNAL DATA SOURCE source_silver
6 WITH
7 (
8     LOCATION= 'https://awstorageaccount2.dfs.core.windows.net/silver',
9     CREDENTIAL= cred_sukanya
10 )
11
12 CREATE EXTERNAL DATA SOURCE source_gold
13 WITH
14 (
15     LOCATION= 'https://awstorageaccount2.dfs.core.windows.net/gold',
16     CREDENTIAL= cred_sukanya
17 )
18
19 CREATE EXTERNAL FILE FORMAT format_parquet
20 WITH
21 (
22     FORMAT_TYPE = PARQUET,
23     DATA_COMPRESSION= 'org.apache.hadoop.io.compress.SnappyCodec'
24 )

```

No results to show
Your query yielded no displayable results

00:00:01 Query executed successfully.

9) CREATE EXTERNAL TABLE AS SELECT (CTAS)

Transfer the views created on silver layer into gold layer as external table using ctas



```

26 ---CREATE EXTERNAL TABLE EXTSALES
27
28
29
30     CREATE EXTERNAL TABLE gold.extsales
31     WITH
32     (
33         LOCATION='exsales',
34         DATA_SOURCE=source_gold,
35         FILE_FORMAT=format_parquet
36     )
37     AS
38     SELECT * from gold.sales
39
40     select * from gold.extsales;
41
42

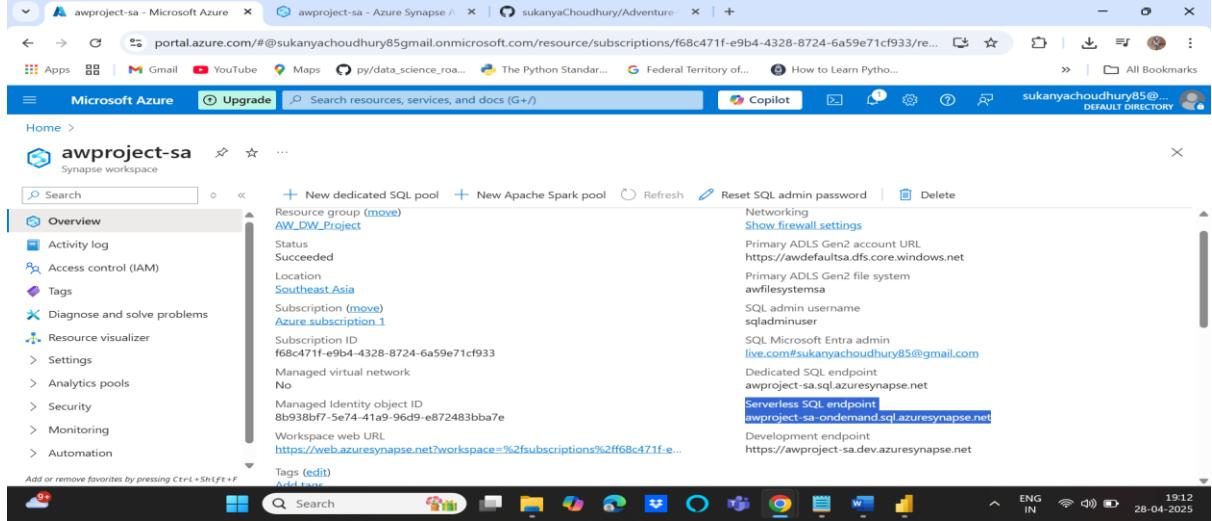
```

Results Messages

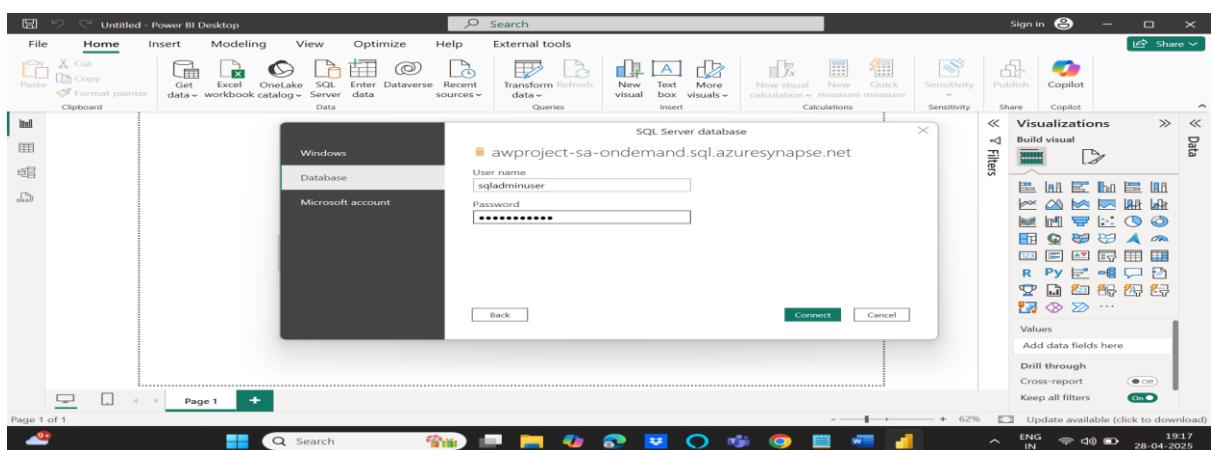
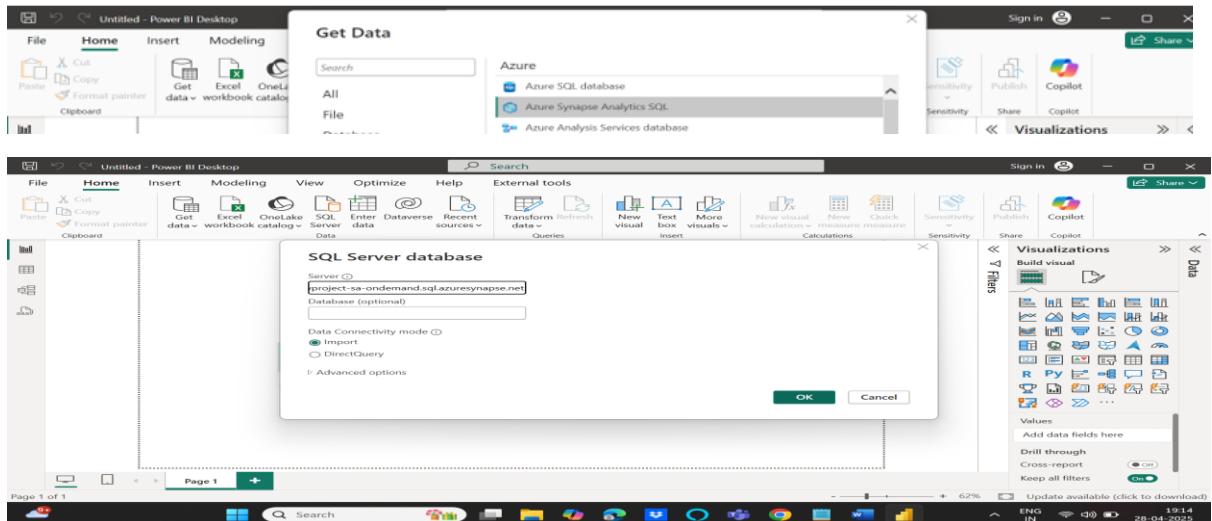
00:00:03 Query executed successfully.

Phase 4: Data Reporting Using Power BI

Connect to power bi using Serverless SQL endpoint in our case, go to synapse workspace ,you can see the SQL endpoint details on the R.H.S:



Then go to power bi → get data→Azure Synapse Analytics SQL



An interactive Dashboard is created to showcase the sales figures:

