Data Engineering project

ZoomCamp - 2024
Maria Fisher

Introduction

The project is built on Azure with Terraform, using the Medallion architecture for data transformation with three layers: bronze, silver, and gold. Key Azure services like Azure Data Factory, Azure Databricks, and Azure Data Lake Storage Gen2.

Synthetic datasets were used to simulate food storage, environmental data scenarios and evaluate fungus and consequently mycotoxin contamination risks. A pipeline was created to transfer data from blob storage to an SQL database, and another pipeline was established to collect data from the database for processing in the medallion layers.

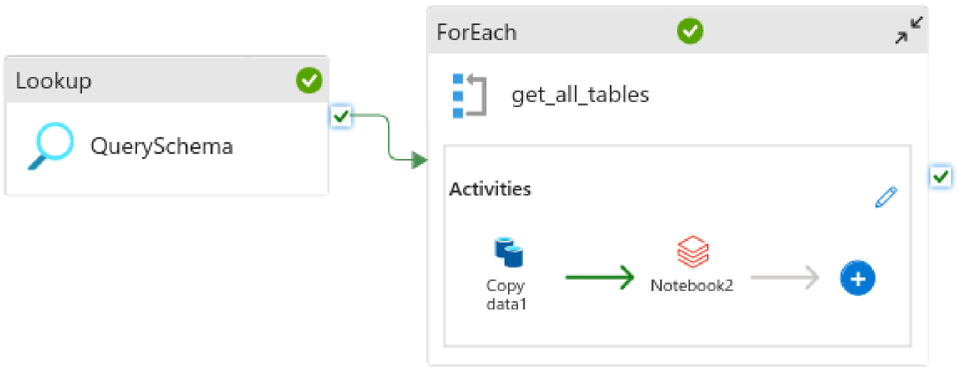
In Databricks a workspace was created within Azure, and a secret key was set up using Azure Key Vault for secure access.

The pipeline was structured to adhere to the Medallion architecture's Bronze-Silver-Gold layers, ensuring accurate data copying and processing.

For data transformation dbt-Databricks was used and SQL files for snapshots of tables were created. The transformed data was saved in the silver layers, and the final output in the gold layer.

Dashboard was created with Databricks and connected with Looker.

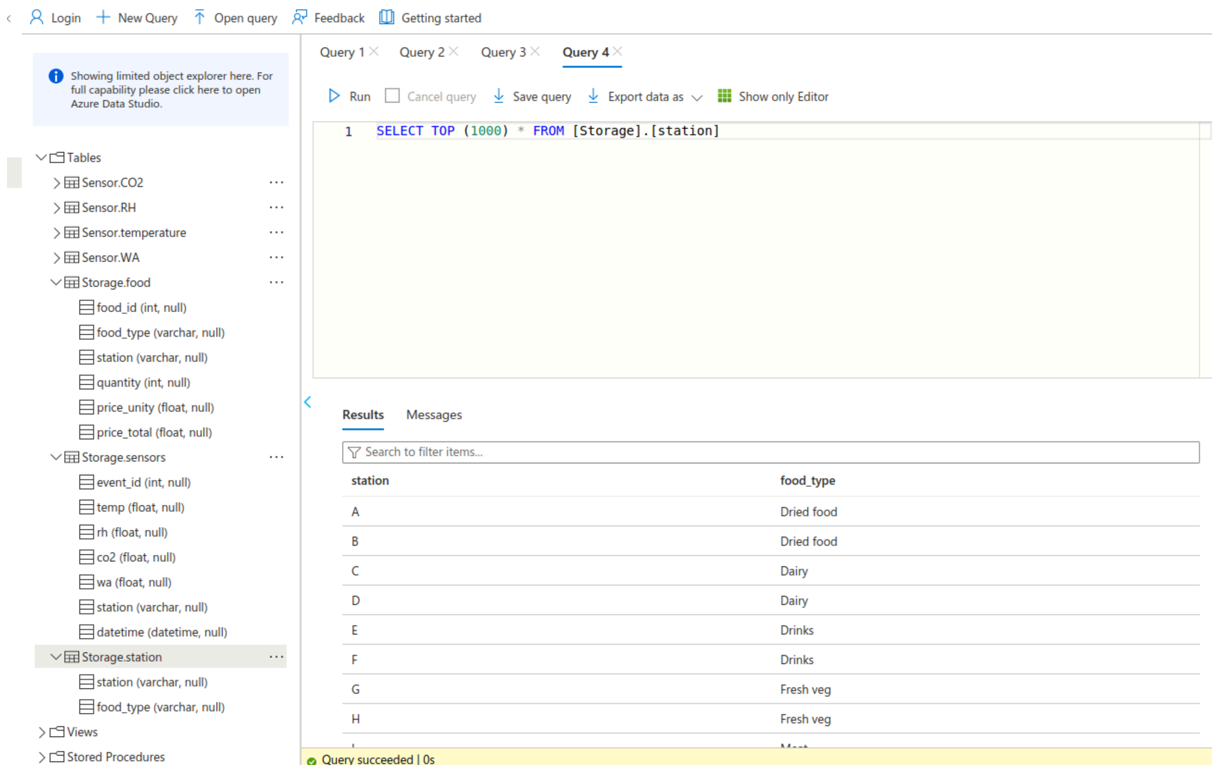
Project pipeline:



This pipeline extracts data from an SQL database and loads it into a Databricks notebook for analysis and storage in a DataLake for further use.

Steps:

Firstly, ensure that your SQL database contains the necessary data. You have two options: populate your database from the Azure marketplace or use your own data. If you prefer the latter, I have prepared a detailed document outlining the steps to create your database using your own data. You can find the instructions in the file named ''Blob storage to Azure SQL database''.



```
Tables
  Sensor.CO2
  Sensor.RH
  Sensor.temperature
  Sensor.WA
  Storage.food
    food_id (int, null)
    food_type (varchar, null)
    station (varchar, null)
    quantity (int, null)
    price_unity (float, null)
    price_total (float, null)
  Storage.sensors
    event_id (int, null)
    temp (float, null)
    rh (float, null)
    co2 (float, null)
    wa (float, null)
    station (varchar, null)
    datetime (datetime, null)
  Storage.station
    station (varchar, null)
    food_type (varchar, null)
Views
Stored Procedures
```

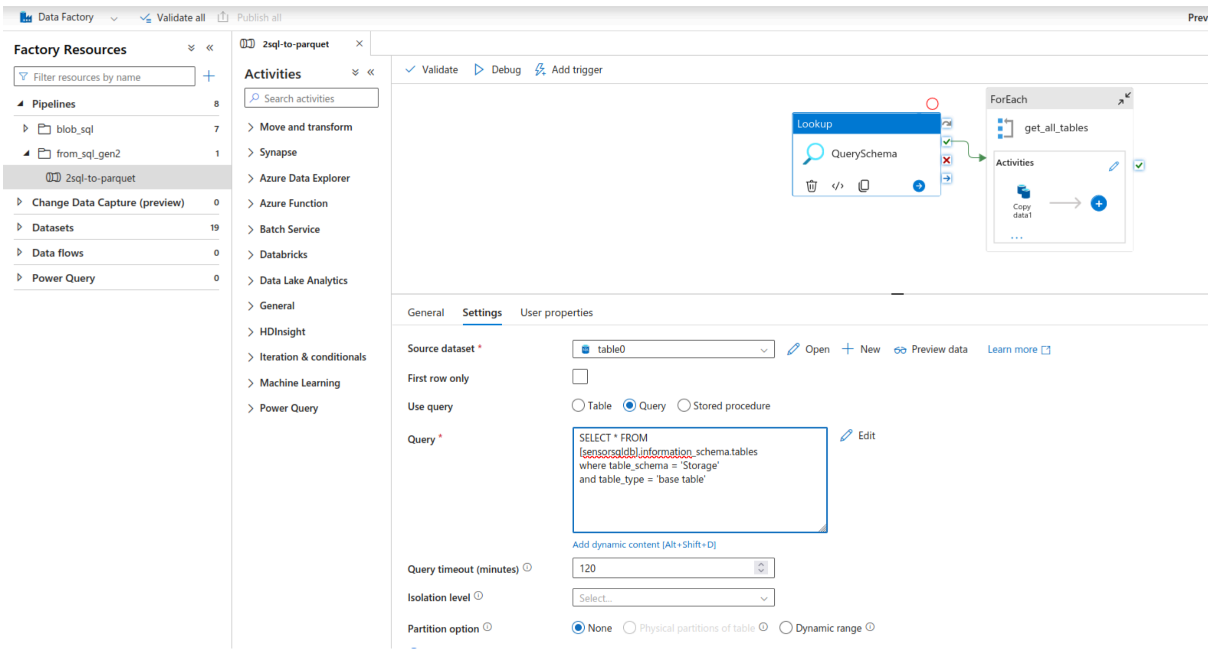
```sql
SELECT TOP (1000) * FROM [Storage].[station]
```

station	food_type
A	Dried food
B	Dried food
C	Dairy
D	Dairy
E	Drinks
F	Drinks
G	Fresh veg
H	Fresh veg

1. Create pipeline to collect data from database to be processed in the medallion layers

- The system requires getting data from a SQL database and storing it in a data lake (that was created previously).
- A query is required to retrieve all tables needed from the database schema.

Pipeline to Copy data from SQL database and save as parquet file into storage Gen2



Source dataset * table0

First row only ☐

Use query: Table [✓ Query] Stored procedure

Query *

```
SELECT * FROM
[sensorsqldb].information_schema.tables
where table_schema = 'Storage'
and table_type = 'base table'
```

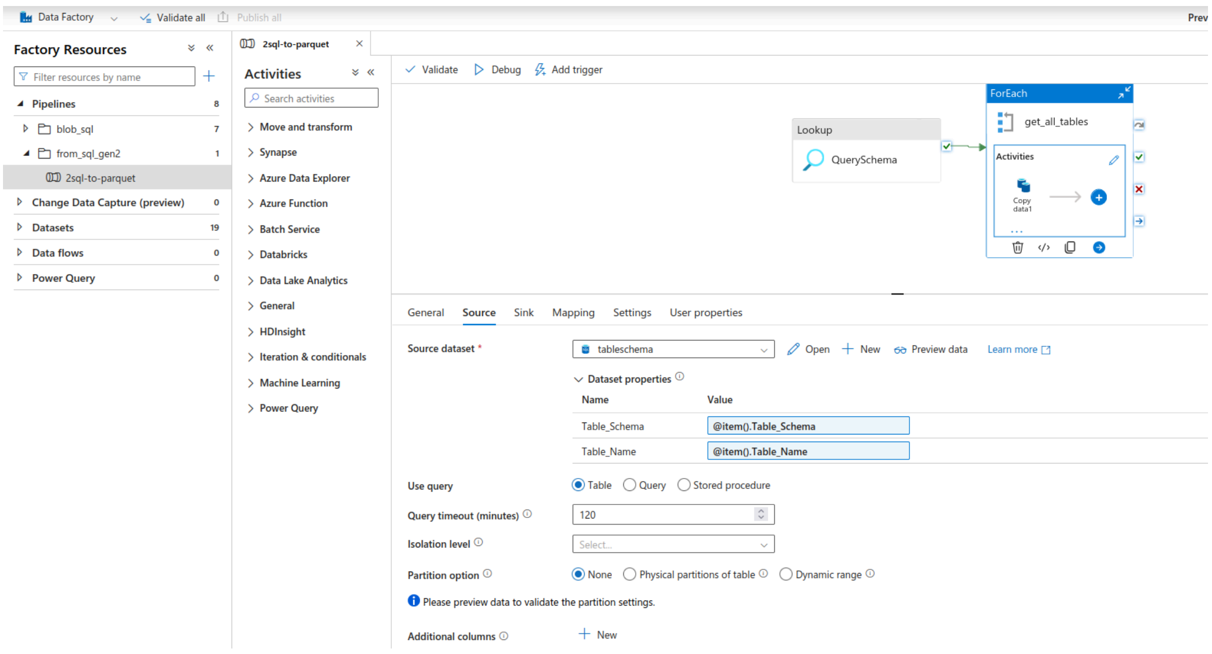
Query timeout (minutes) 120

Isolation level Select...

Partition option [✓ None] Physical partitions of table Dynamic range

- In the pipeline a ForEach activity is established to iterate through each table retrieved from the SQL query.
- Within the ForEach loop, a Copy Data activity is configured to **copy the data** from each table into the bronze layer (**source**) to the data lake (**sink**). The data will be saved as **.parquet**
- Parameters for schema name and table name are dynamically passed to the datasets and activities. However, if the schema does not match you can change the Json file manually. I have added instructions on how to do that in the doc **From_blob_storage_to_Azure_SQL_db.**
- A table is created for the destination in the Azure Data Lake Storage Gen2, with parameters for folder name and file name.
- Folder and file names are dynamically constructed using schema name, table name.

Souce (SQL database):



Source dataset * tableschema Open New Preview data Learn more

Dataset properties

Name	Value
Table_Schema	@item().Table_Schema
Table_Name	@item().Table_Name

Use query: [✓ Table] Query Stored procedure

Query timeout (minutes): 120

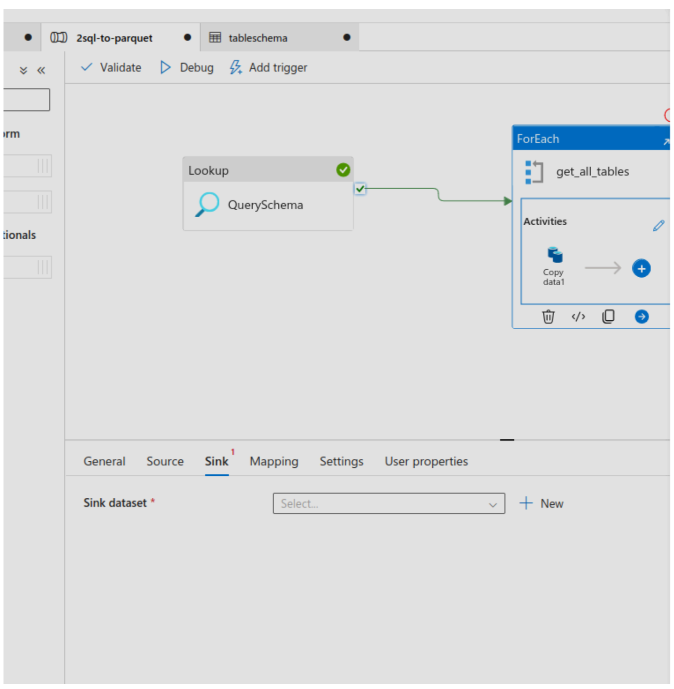
Isolation level: Select...

Partition option: [✓ None] Physical partitions of table Dynamic range

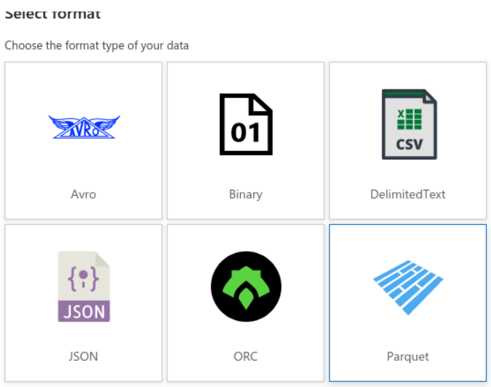
Please preview data to validate the partition settings.

Additional columns: New

Sink (Storage Gen2):



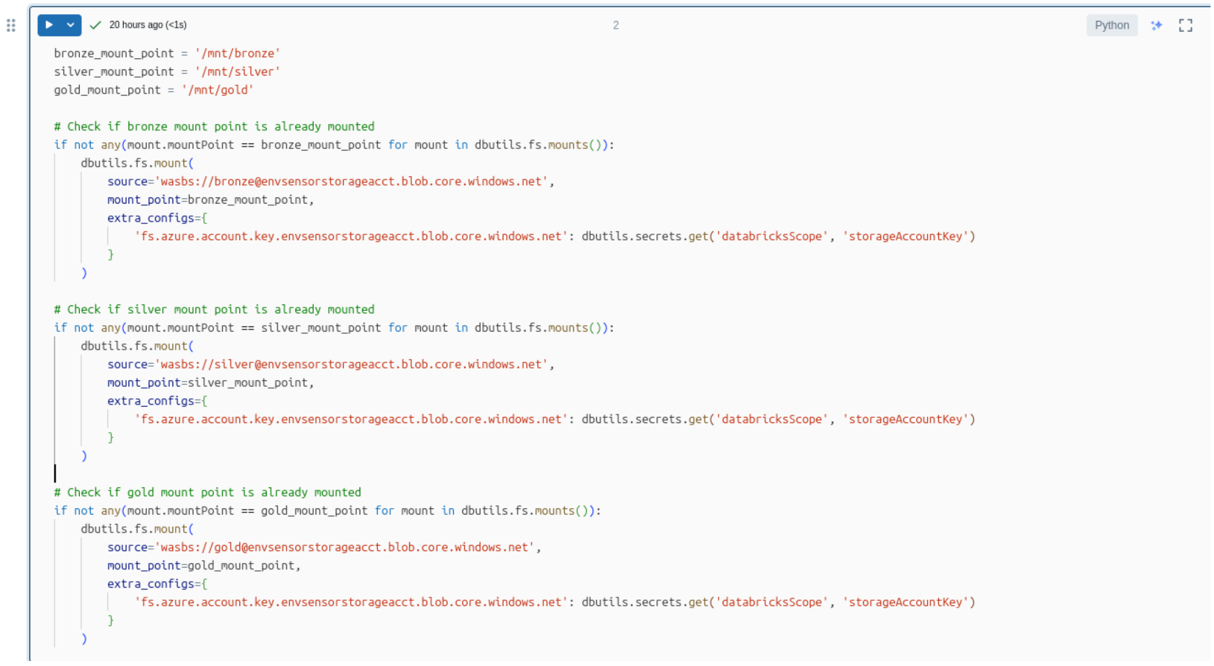
Sink dataset * Select... New



2. Databricks data processing

- Next steps involve populating the data into a Databricks notebook for further processing.
- A new Databricks workspace is created within Azure.

- A secret key is set up in Databricks using Azure Key Vault to securely store access key, enabling secure access to Azure resources from Databricks notebooks.
- Next, in Databricks create a **‘Compute’** for data processing, then mount the silver and gold storage accounts and confirm their accessibility.
- Create the Compute and specify the configuration needed [for this example a single node, 12.2 LTS (includes Apache Spark 3.3.2, Scala 2.12) and Standard_DS3_v2 were selected for processing].
- Create a notebook and mount to the storage to execute scripts.



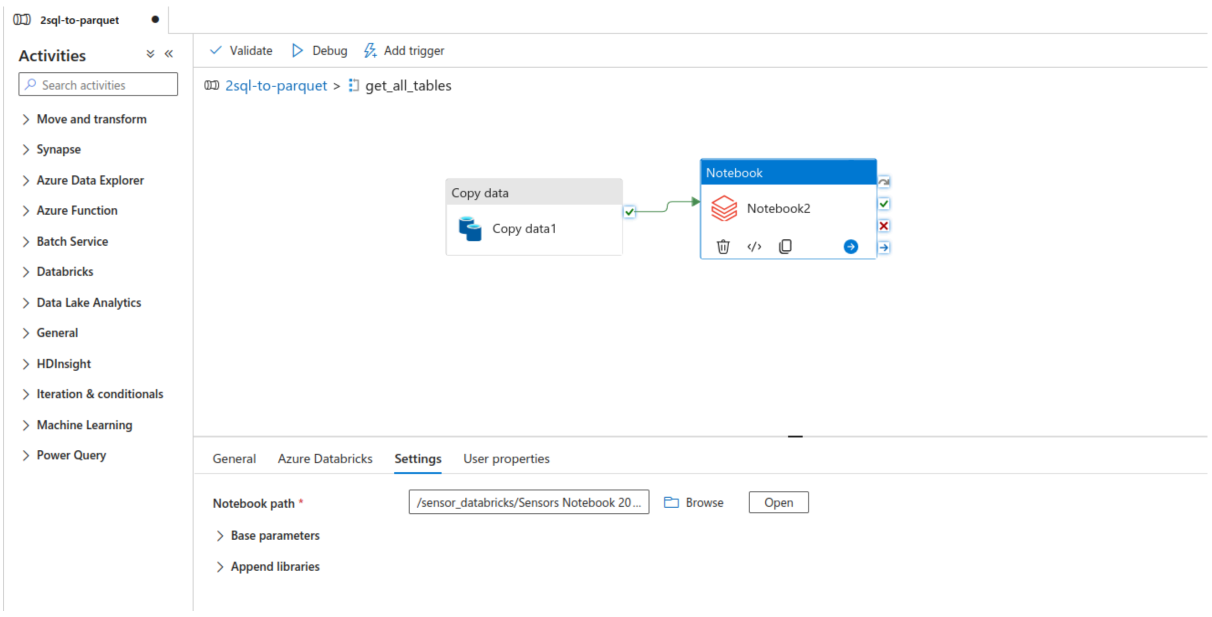
```python
bronze_mount_point = '/mnt/bronze'
silver_mount_point = '/mnt/silver'
gold_mount_point = '/mnt/gold'

# Check if bronze mount point is already mounted
if not any(mount.mountPoint == bronze_mount_point for mount in dbutils.fs.mounts()):
    dbutils.fs.mount(
        source='wasbs://bronze@envsensorstorageacct.blob.core.windows.net',
        mount_point=bronze_mount_point,
        extra_configs={
            'fs.azure.account.key.envsensorstorageacct.blob.core.windows.net': dbutils.secrets.get('databricksScope', 'storageAccountKey')
        }
    )

# Check if silver mount point is already mounted
if not any(mount.mountPoint == silver_mount_point for mount in dbutils.fs.mounts()):
    dbutils.fs.mount(
        source='wasbs://silver@envsensorstorageacct.blob.core.windows.net',
        mount_point=silver_mount_point,
        extra_configs={
            'fs.azure.account.key.envsensorstorageacct.blob.core.windows.net': dbutils.secrets.get('databricksScope', 'storageAccountKey')
        }
    )

# Check if gold mount point is already mounted
if not any(mount.mountPoint == gold_mount_point for mount in dbutils.fs.mounts()):
    dbutils.fs.mount(
        source='wasbs://gold@envsensorstorageacct.blob.core.windows.net',
        mount_point=gold_mount_point,
        extra_configs={
            'fs.azure.account.key.envsensorstorageacct.blob.core.windows.net': dbutils.secrets.get('databricksScope', 'storageAccountKey')
        }
    )
```

Establish connections between Data Factory and Databricks, the notebook can be added inside the pipeline.

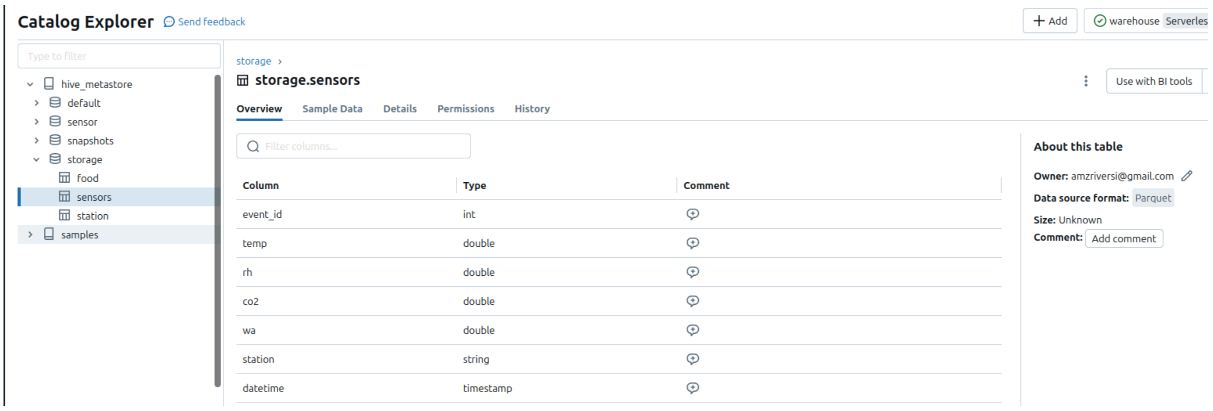


Notebook path * /sensor_databricks/Sensors Notebook 20 ...

Steps:

- A pipeline named "get-all-tables" is created, consisting of activities to get table names and loop through each table for further processing.
- A Lookup activity retrieves the table names using the dataset created earlier, and a ForEach activity is added to iterate through each table.
- Within the ForEach loop, a Copy Data activity is added for each table to copy data from the SQL database into the Bronze layer in Azure Data Lake Storage Gen2.
- Configure the settings such as source and sink tables to Copy Data and ensure the data is copied accurately.
- The pipeline is structured to adhere to the Medallion architecture's Bronze-Silver-Gold layers, with data flowing from the Bronze layer (SQL database) to the Silver layer (Azure Data Lake Storage Gen2) for further processing.

Once the pipeline execution is successful, verify if the data is correctly copied into the bronze layer and if the databases and tables are created as expected in the Azure Databricks environment.



storage.sensors

Column	Type	Comment
event_id	int	
temp	double	
rh	double	
co2	double	
wa	double	
station	string	
datetime	timestamp	

About this table

Owner: amzriversi@gmail.com

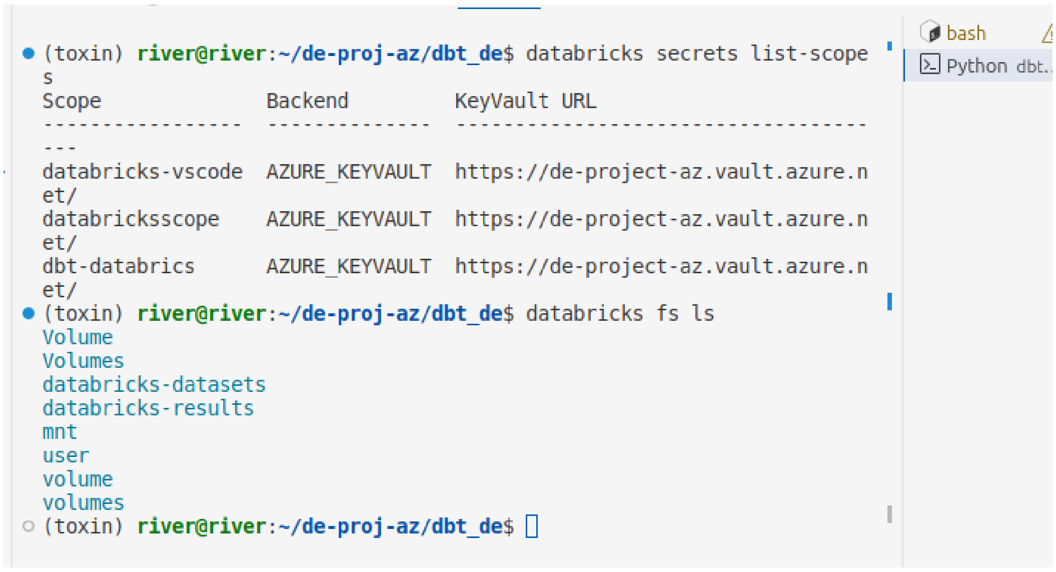
Data source format: Parquet

Size: Unknown

Comment: Add comment

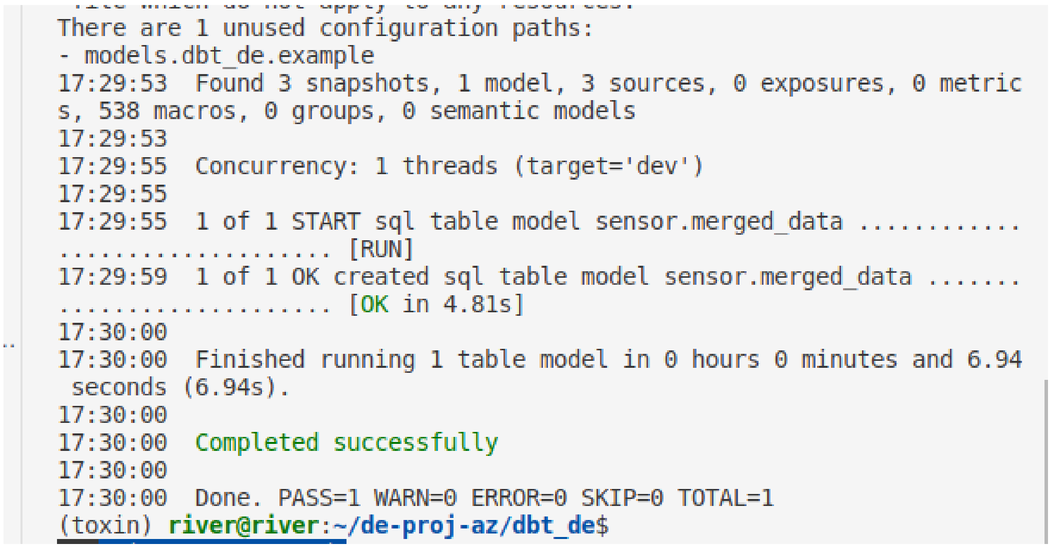
3. Processing the data in dbt-Databricks

- Transitioning to the **dbt** (Data Build Tool) for data transformation, a new project is created.
- The project name is set as "de-proj-az" with Python 3.9.
- Dependencies including Databricks and dbt are installed with "pip install" commands.
- Configuration of Databricks CLI is done to establish a connection with Databricks on the cloud.
- Testing the Databricks connection and file system listing ensures successful connection.



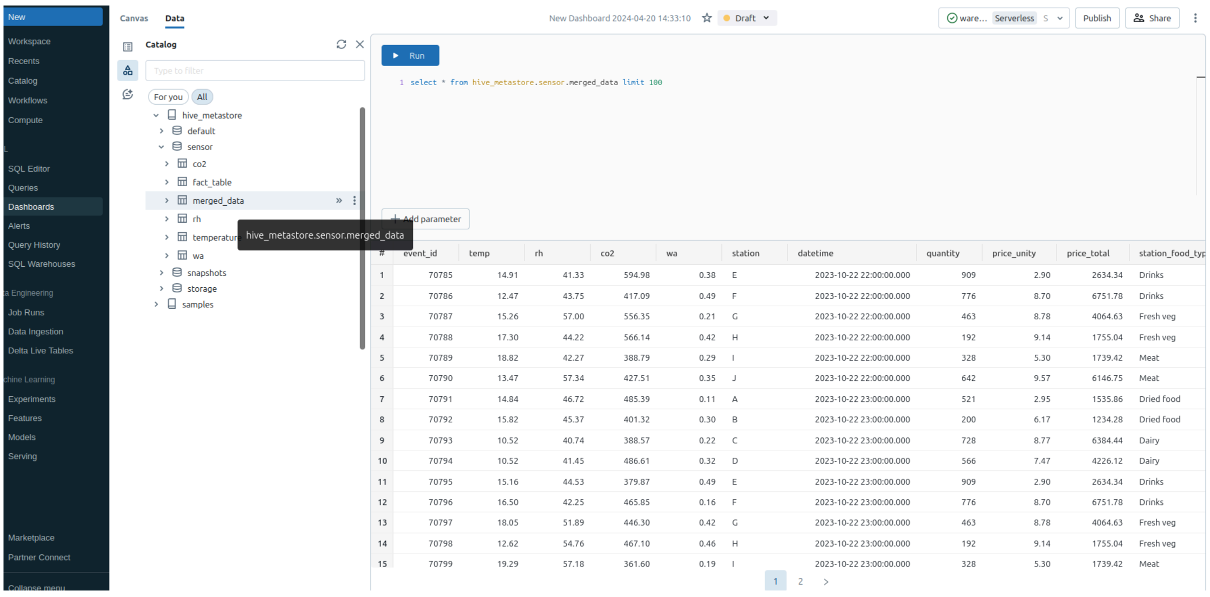
```
● (toxin) river@river:~/de-proj-az/dbt_de$ databricks secrets list-scope
  s
  Scope              Backend         KeyVault URL
  -----------------  --------------  -----------------------------------
  ---
  databricks-vscode  AZURE_KEYVAULT  https://de-project-az.vault.azure.n
  et/
  databricksscope    AZURE_KEYVAULT  https://de-project-az.vault.azure.n
  et/
  dbt-databrics      AZURE_KEYVAULT  https://de-project-az.vault.azure.n
  et/
● (toxin) river@river:~/de-proj-az/dbt_de$ databricks fs ls
  Volume
  Volumes
  databricks-datasets
  databricks-results
  mnt
  user
  volume
  volumes
○ (toxin) river@river:~/de-proj-az/dbt_de$ 
```

- dbt initialization is performed with "dbt init" command, setting project name, database, Databricks host, access token, and default schema.
- Created SQL files for snapshots of various tables: data schema [ ]
- Added YAML configuration files for recognizing source tables and schemas from Azure Databricks.
- Executed the dbt snapshot command after resolving dependencies, generating snapshots for all tables.
- The dbt system keeps track of historical changes in data, including the last valid date before changes.
- Data snapshots are stored in the silver section of the storage account, providing a means to replay data if necessary.
- Transformation of data occurs in the bronze and silver sections, while the final output will be stored in the gold layer .
- SQL scripts are used for transformations, selecting data from snapshots and performing necessary operations.
- The dbt documentation displays the structure of the transformations, including source tables, code, and dependencies, aiding in understanding the data transformation pipeline.



```
There are 1 unused configuration paths:
- models.dbt_de.example
17:29:53  Found 3 snapshots, 1 model, 3 sources, 0 exposures, 0 metric
s, 538 macros, 0 groups, 0 semantic models
17:29:53
17:29:55  Concurrency: 1 threads (target='dev')
17:29:55
17:29:55  1 of 1 START sql table model sensor.merged_data ............
.................... [RUN]
17:29:59  1 of 1 OK created sql table model sensor.merged_data .......
.................... [OK in 4.81s]
17:30:00
17:30:00  Finished running 1 table model in 0 hours 0 minutes and 6.94
 seconds (6.94s).
17:30:00
17:30:00  Completed successfully
17:30:00
17:30:00  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(toxin) river@river:~/de-proj-az/dbt_de$
```

- The structure of the data on the database level mirrors what is seen in the dbt project, with tables like "merged_data" and "snapshot" will be added to Azure database containers (bronze, silver and gold layers) as well as into Databricks storage for further use.

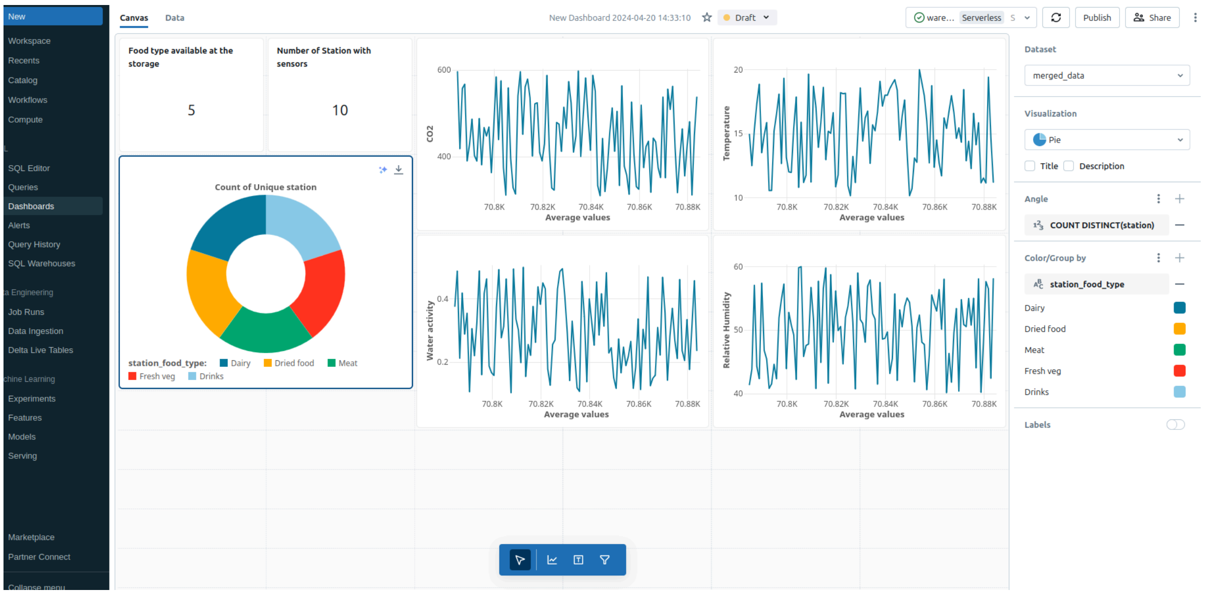


```
select * from hive_metastore.sensor.merged_data limit 100
```

#	event_id	temp	rh	co2	wa	station	datetime	quantity	price_unity	price_total	station_food_typ
1	70785	14.91	41.33	594.98	0.38	E	2023-10-22 22:00:00.000	909	2.90	2634.34	Drinks
2	70786	12.47	43.75	417.09	0.49	F	2023-10-22 22:00:00.000	776	8.70	6751.78	Drinks
3	70787	15.26	57.00	556.35	0.21	G	2023-10-22 22:00:00.000	463	8.78	4064.63	Fresh veg
4	70788	17.30	44.22	566.14	0.42	H	2023-10-22 22:00:00.000	192	9.14	1755.04	Fresh veg
5	70789	18.82	42.27	388.79	0.29	I	2023-10-22 22:00:00.000	328	5.30	1739.42	Meat
6	70790	13.47	57.34	427.51	0.35	J	2023-10-22 22:00:00.000	642	9.57	6146.75	Meat
7	70791	14.84	46.72	485.39	0.11	A	2023-10-22 23:00:00.000	521	2.95	1535.86	Dried food
8	70792	15.82	45.37	401.32	0.30	B	2023-10-22 23:00:00.000	200	6.17	1234.28	Dried food
9	70793	10.52	40.74	388.57	0.22	C	2023-10-22 23:00:00.000	728	8.77	6384.44	Dairy
10	70794	10.52	41.45	486.61	0.32	D	2023-10-22 23:00:00.000	566	7.47	4226.12	Dairy
11	70795	15.16	44.53	379.87	0.49	E	2023-10-22 23:00:00.000	909	2.90	2634.34	Drinks
12	70796	16.50	42.25	465.85	0.16	F	2023-10-22 23:00:00.000	776	8.70	6751.78	Drinks
13	70797	18.05	51.89	446.30	0.42	G	2023-10-22 23:00:00.000	463	8.78	4064.63	Fresh veg
14	70798	12.62	54.76	467.10	0.46	H	2023-10-22 23:00:00.000	192	9.14	1755.04	Fresh veg
15	70799	19.29	57.18	361.60	0.19	I	2023-10-22 23:00:00.000	328	5.30	1739.42	Meat

4. Visualisation dashboard

Dashboards can be created in Databricks or in connection with Looker



Dataset: merged_data

Visualization: Pie

Angle: COUNT DISTINCT(station)

Color/Group by: station_food_type

Dairy	🟦
Dried food	🟧
Meat	🟩
Fresh veg	🟥
Drinks	🟦

Summary

This data engineering project was successfully completed as part of the ZoomCamp course. I developed this project on Azure implementing the Medallion architecture. The infrastructure was created with Terraform, and I leveraged Azure Data Factory, Azure Databricks, and Azure Data Lake Storage Gen2 to transform the data efficiently.

The main objective of this project was to streamline data processing and analysis in a cloud environment, enabling efficient decision-making processes. By using Azure services and implementing the Medallion architecture, scalability and flexibility can be ensured. The use of synthetic datasets aided in risk assessment, while pipelines facilitated smooth data transfer.

I use dbt-Databricks for data transformation and analysis, integrating SQL files to capture table snapshots and gain valuable insights. The transformed data was then stored in the appropriate layer (bronze, silver, or gold) based on the processing stage.
Furthermore, a dashboard was created using Databricks and Looker for data visualisation.

Overall, this project exemplified effective data engineering practices within the Azure environment, showcasing the power of cloud-based solutions for complex data projects.