

Data Engineering project

ZoomCamp - 2024

Maria Fisher

This project is built on Azure with Terraform , the data will be processed with Spark and dbt, using the Medallion architecture that has three layers of data transformation: bronze, silver, and gold. In the bronze layer, data remains unchanged, while the silver layer involves some transformation and potential data removal. The gold layer houses data ready for business intelligence consumption. Key Azure services like Azure Data Factory, Azure Databricks, and Azure Data Lake Storage Gen2 will be used.

The process involves creating resources with Terraform, setting up users with specific permissions, and configuring Azure Data Factory to access Gen2 Lake Service and a SQL database. The SQL database is created with a specific server and authentication settings.

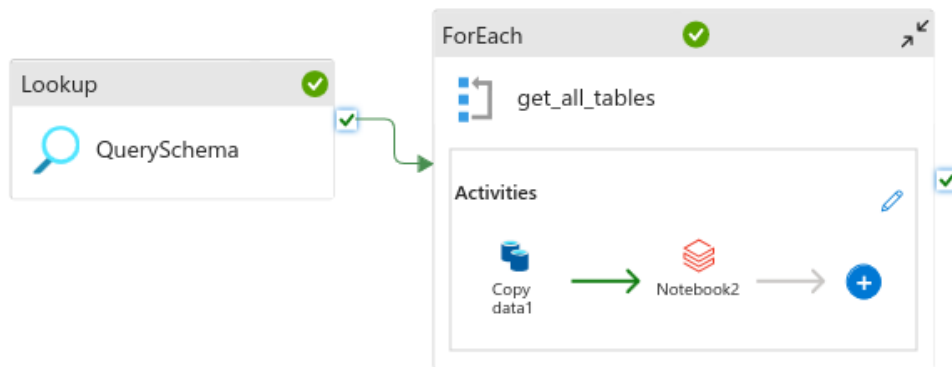
I used a synthetic dataset to simulate a food storage scenario and evaluate the risk of mycotoxin contamination. This data will be used to populate the SQL database using Python code.

Pipeline to transfer data sets from blob storage to sql database (instruction are in a separated file “Blob storage to Azure SQL database.pdf”)

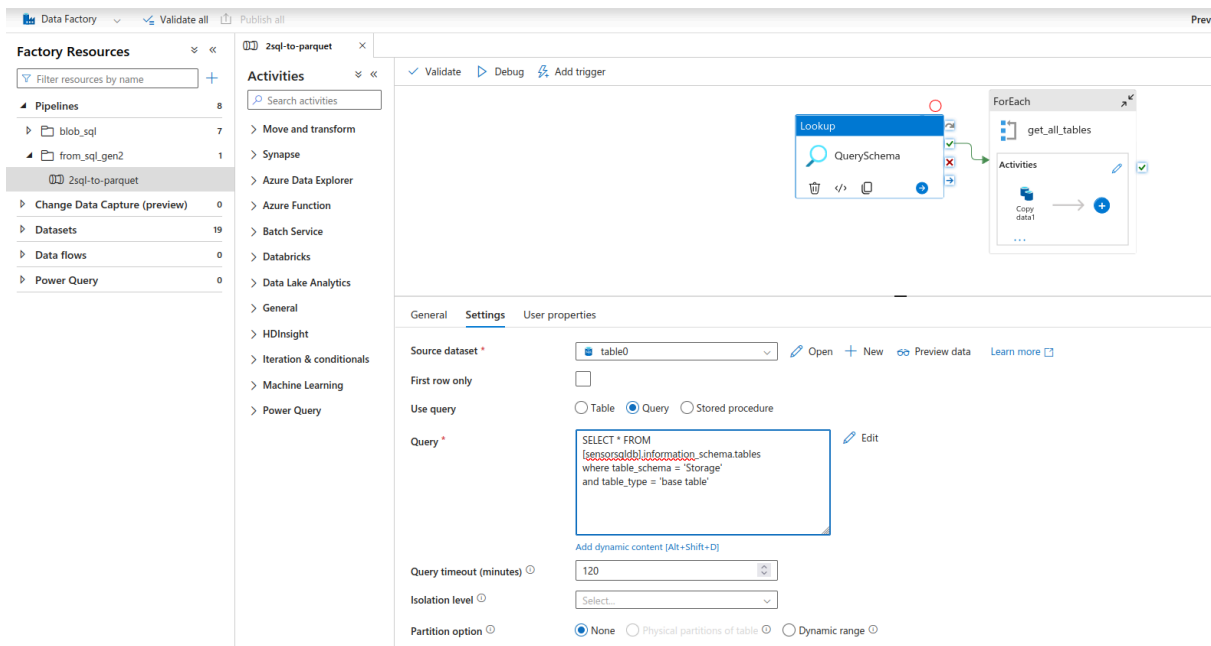
The screenshot displays the Azure Data Studio interface. On the left, the 'Object Explorer' pane shows a tree view of the database schema. Under 'Tables', there are several tables including 'Sensor.CO2', 'Sensor.RH', 'Sensor.temperature', 'Sensor.WA', 'Storage.food', 'Storage.sensors', and 'Storage.station'. The 'Storage.station' table is currently selected. The main editor pane shows a SQL query: `SELECT TOP (1000) * FROM [Storage].[station]`. Below the query editor, the 'Results' pane displays the query output as a table with two columns: 'station' and 'food_type'. The results show 8 rows of data, with stations A through H and their corresponding food types. A status bar at the bottom indicates 'Query succeeded | 0s'.

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 fetching 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.



- 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 data from each table into the bronze layer (**source**) of 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.
- A dataset 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.

Source:

Factory Resources

- Pipelines: 8
 - blob_sql: 7
 - from_sql_gen2: 1
 - 2sql-to-parquet**
 - Change Data Capture (preview): 0
 - Datasets: 19
 - Data flows: 0
 - Power Query: 0

Activities

- Move and transform
- Synapse
- Azure Data Explorer
- Azure Function
- Batch Service
- Databricks
- Data Lake Analytics
- General
- HDInsight
- Iteration & conditionals
- Machine Learning
- Power Query

Source

Source dataset: **tableschema**

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:

Select format

Choose the format type of your data

- Avro
- Binary
- DelimitedText
- JSON
- ORC
- Parquet**


Sink

Sink dataset: Select...

4. Databricks data processing

- Next steps involve populating the data into a Databricks notebook for further processing.
- A new Databricks workspace is created within Azure, and a notebook named "base notebook" is created within a workspace folder.

- Secrets are set up in Databricks using Azure Key Vault to securely store access keys for Azure resources like storage accounts.
- The URI and resource ID of the Key Vault are linked to Databricks for access, enabling secure access to Azure resources from Databricks notebooks.
- If the secret has been created successfully, confirm and check the base notebook and execute a mount command for our storage accounts.



```

20 hours ago (<1s) 2 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')
        }
    )

```

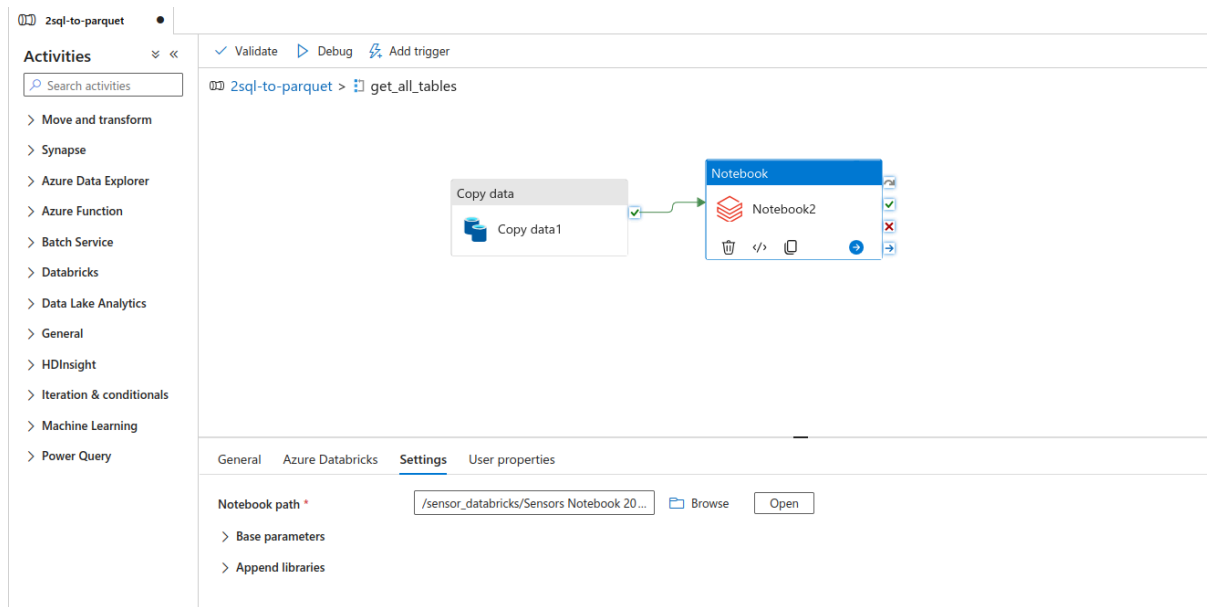
Next, in Databricks create a **‘Compute’** for data processing, then mount the silver and gold storage accounts and confirm their accessibility.

After testing database creation in our Databricks catalogue, we prepare to automate table creation in our database.

This involves establishing connections between Databricks and Data Factory, configuring notebooks, and setting up linked services for Data Factory to access Databricks.

steps:

- Add the secret Code has been created and confirm.
- Mounting storage accounts into the notebook is essential for proper functionality.
- 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].
- A notebook was created to mount storage and execute scripts.
- Establishing connections between Data Factory and Databricks is necessary. Inside the pipeline add Notebook (Databricks).



Steps:

- A pipeline named "get-all-tables" is created, consisting of activities to fetch 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.
- Configuration settings such as source and sink datasets are specified in the Copy Data activity to 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.

Catalog Explorer [Send feedback](#) + Add warehouse Serverless

Type to filter

- hive_metastore
 - default
 - sensor
 - sensors
 - station
 - samples

storage.sensors Use with BI tools

Overview Sample Data Details Permissions History

Filter columns...

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

About this table

Owner: amzrivers@gmail.com [Edit](#)

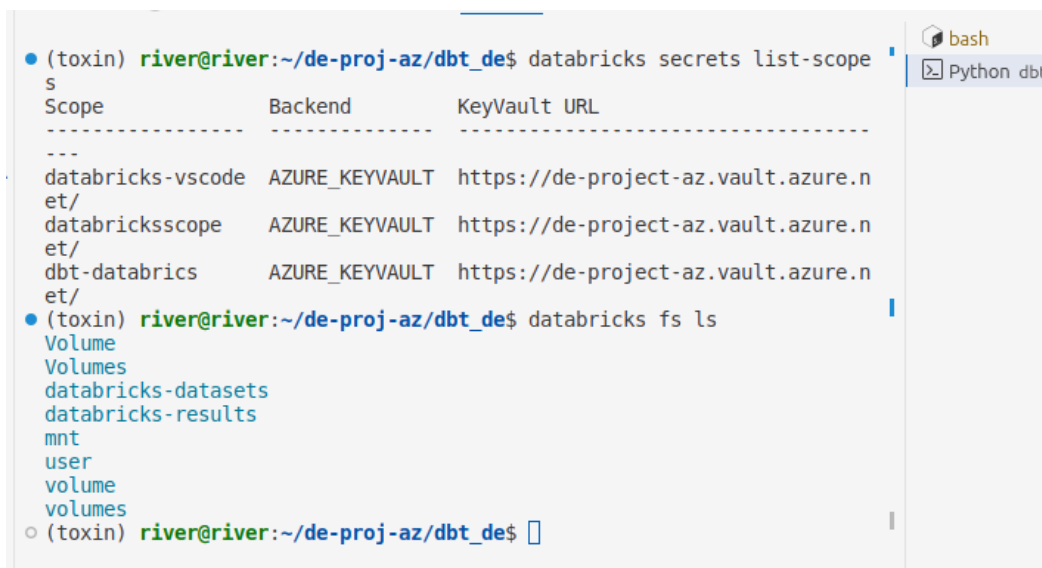
Data source format: [Parquet](#)

Size: Unknown

Comment: [Add comment](#)

5. Processing the data in dbt-Databricks

- Transitioning to the dbt (Data Build Tool) section 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
Scope          Backend      KeyVault URL
-----
databricks-vscode AZURE_KEYVAULT https://de-project-az.vault.azure.net/
databricksscope  AZURE_KEYVAULT https://de-project-az.vault.azure.net/
dbt-databricks   AZURE_KEYVAULT https://de-project-az.vault.azure.net/
(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 transformation for gold occurs in the mat section.
- SQL scripts are used for transformations, selecting data from snapshots and performing necessary operations.
- YAML configuration files define the sources and schemas recognized by dbt for transformation.

Generated documentation provides insights into the data transformation processes, including source tables, dependencies, and SQL code.

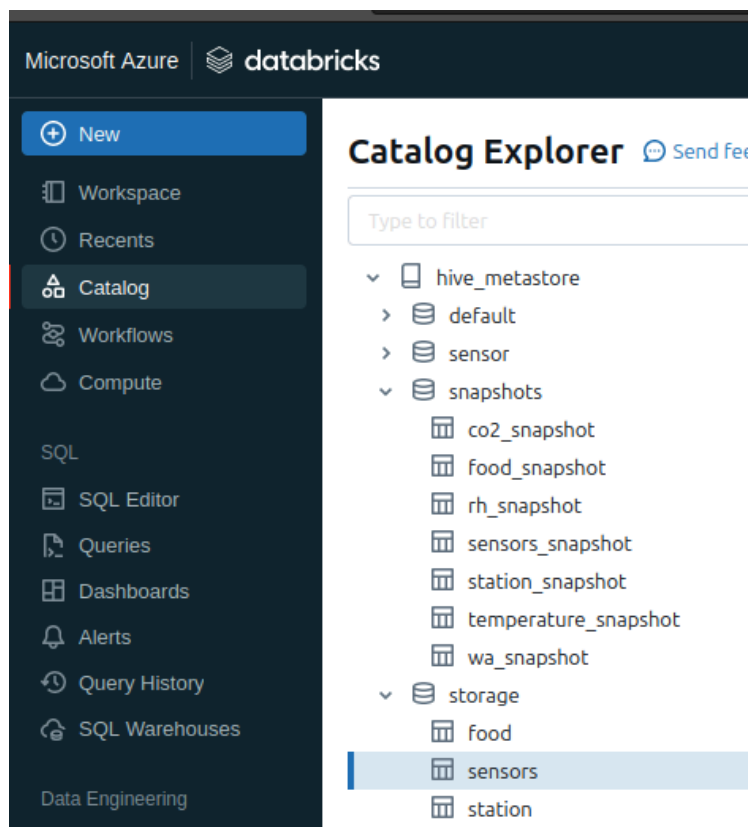
- The dbt documentation displays the structure of the transformations, including source tables, code, and dependencies, aiding in understanding the data transformation pipeline.

```

There are 1 unused configuration paths:
- models.dbt_de.example
17:29:53 Found 3 snapshots, 1 model, 3 sources, 0 exposures, 0 metrics, 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.



6. Visualisation dashboard

Dashboards can be created in Databricks or in connection with Looker

