Data Pipeline for Customer Account Analysis

Data Ingestion (Backend Storage to Raw(Bronze) Container)

Step 1: Created Resource group

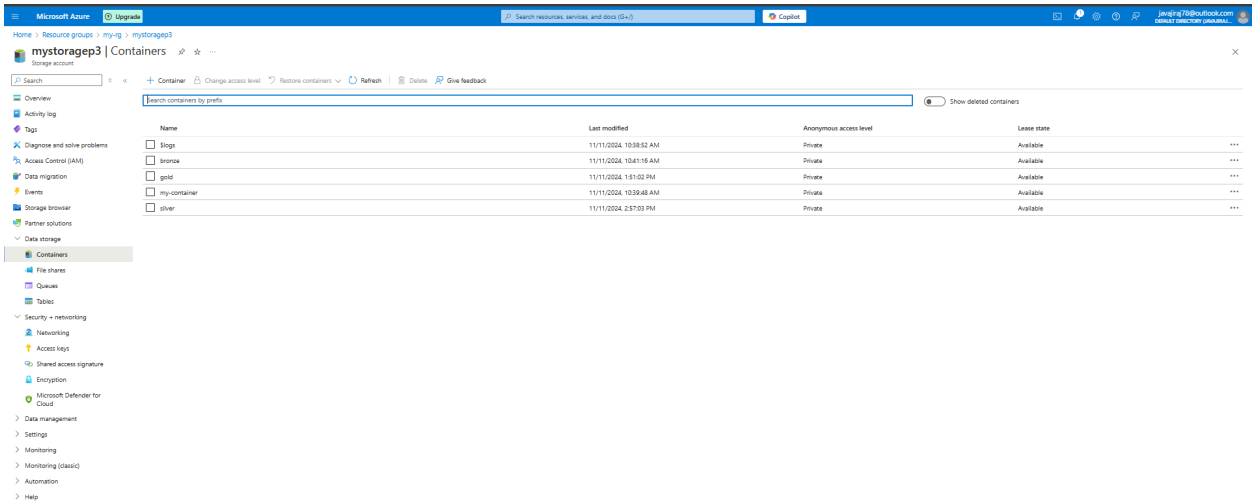
- Resource group name: “my-rg”

Step 2: Created Storage account

- Storage account name: “mystoragep3”

Step 3: Created 4 containers

- 1st Container name: “my-container”
- Uploaded accounts, customers, loan_payments, loans, transactions files in the container “my-container”.
- 2nd Container name: “bronze”
- Keeping the container empty.
- 3rd Container name: “silver”
- Keeping the container empty.
- 4rd Container name: “gold”
- Keeping the container empty.



Step 4: Created Azure Data Factory

- Azure Data Factory name: “A-datafactory1”
- Launch studio
- Select - Author
- Select - New Pipeline

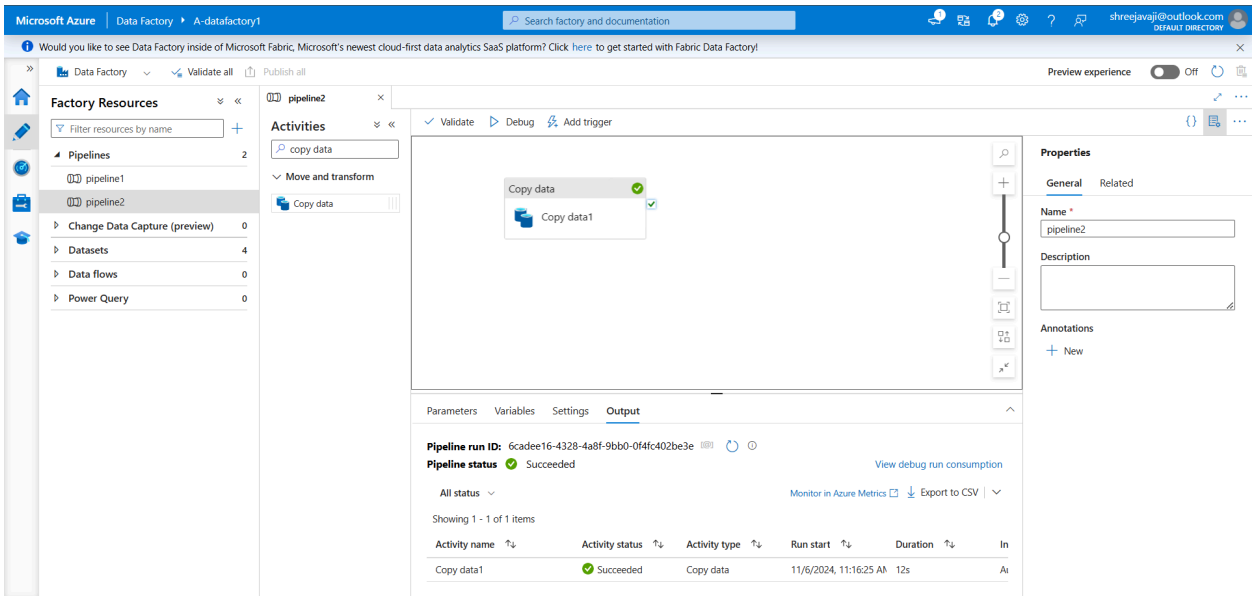
- Copy Data - Drag and drop

Linked Services: “Azure Data lake Gen2”

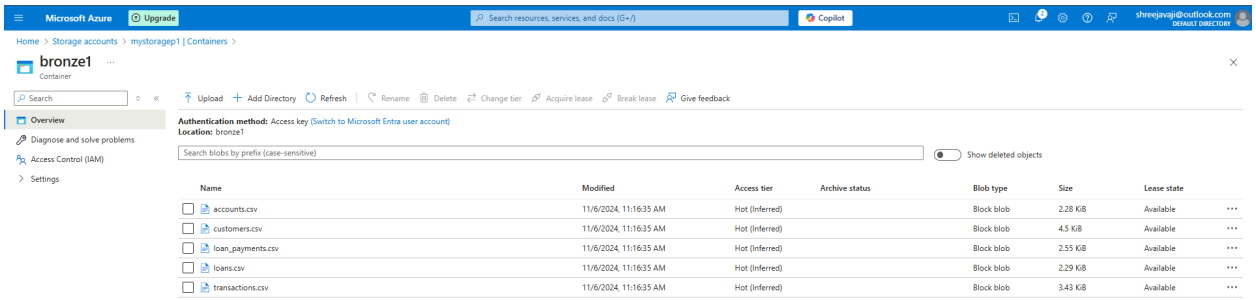
- Select - Source
- Click on - New
- Select - “Azure Data lake Gen2”
- Select file format - “CSV”
- Name - “1dlstoragegen2”
- Linked service - New
- Name - “lservice1”
- Select - Storage account : “mystoragep3”
- Click on - Test Connection
- Select - Create
- Select - File Path - “my-container” - Ok - Ok
- In Source: file path type : select - wild card file path (All the files in the selected container in source will be selected together)

Linked Services: “Azure Data lake Gen2”

- Select - Sink
- Click on - New
- Select - “Azure Data lake Gen2”
- Select file format - “CSV”
- Name - “2dlstoragegen2”
- Linked service - select existing 1st linked service “lservice1”
- File path - Container - “bronze” - ok
- Validate
- Publish all
- Debug



- Cross check in the storage account “mystoragep1” in the container “bronze1”
- All the files in the “my-container” have been moved to the “Bronze1” container.



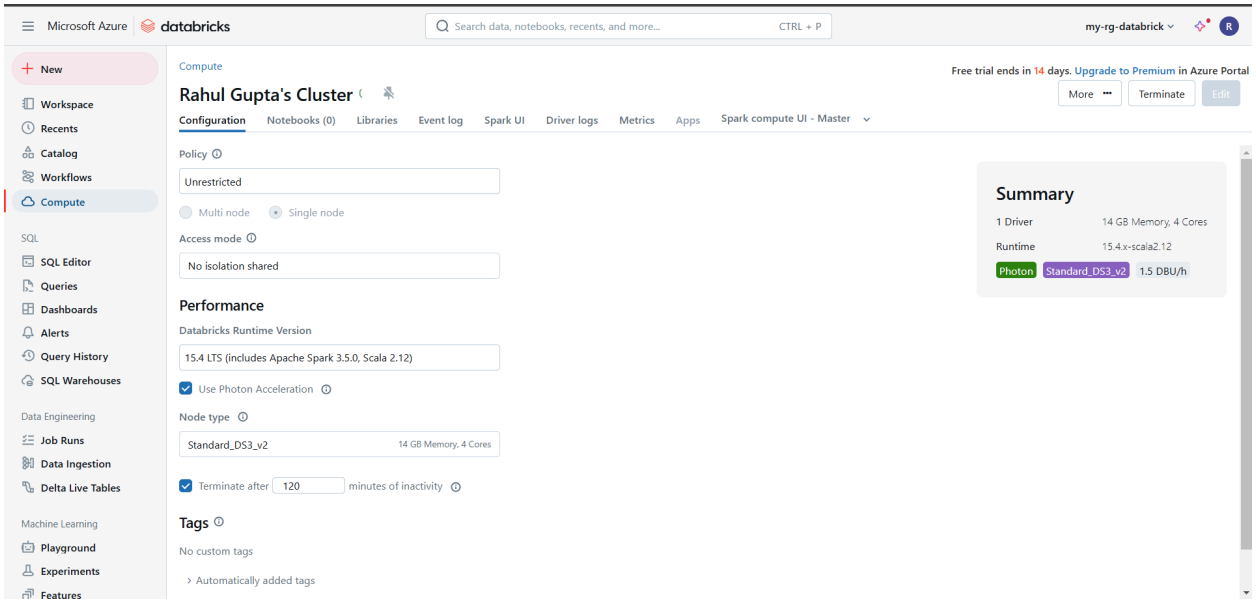
Databricks Activity (Incremental/Delta Processing)

Step 1: Created Azure DataBricks

- Azure DataBricks name: “my-rg-databrick”
- Launch workspace

Step 2: Creating a cluster

- Select compute
- Select - create compute



Step 3: Create a notebook:

- Select workspace - Create - folder
- Folder name “my-rg-dbfolder”
- In this folder creating notebook

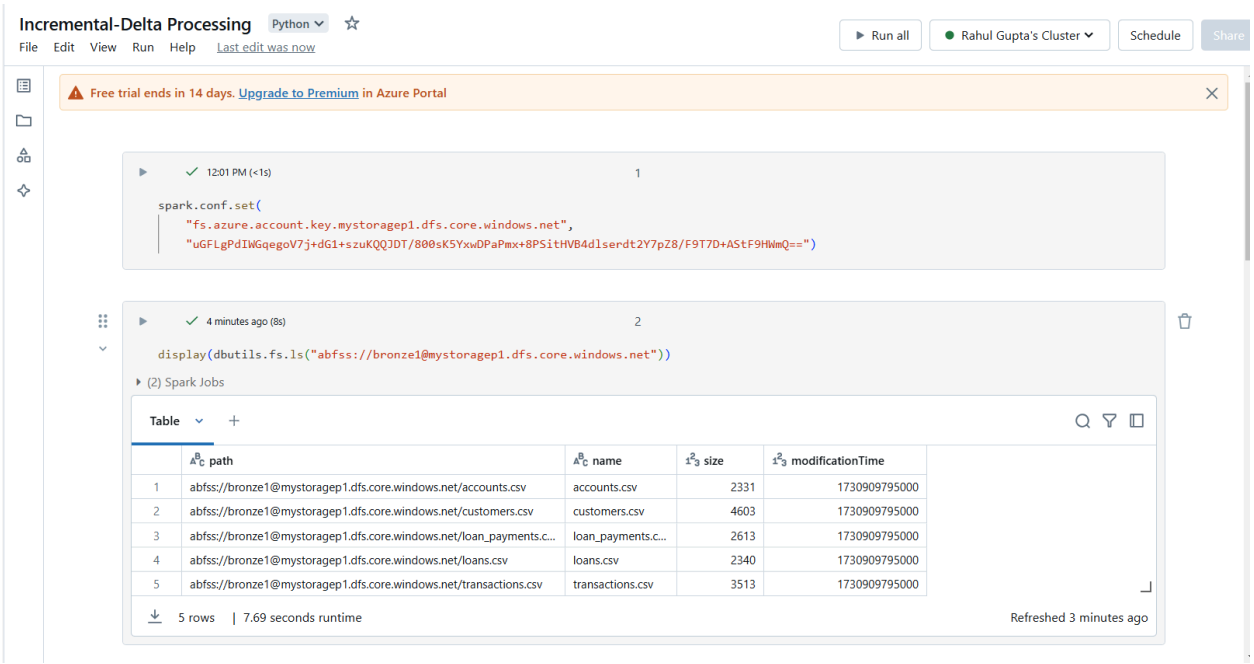
- Select - Create - Notebook
- Rename the notebook as “Incremental-Delta Processing”

Code 1:

```
spark.conf.set(
    "fs.azure.account.key.mystoragep3.dfs.core.windows.net",

"uGFLgPdIWGqegoV7j+dG1+szuKQQJDT/800sK5YxwDPaPmx+8PSitHVB4dlserdt2Y7pZ8/F9T7D+A
StF9HWmQ==")
```

Code 2:

```
display(dbutils.fs.ls("abfss://bronze@mystoragep3.dfs.core.windows.net"))
```

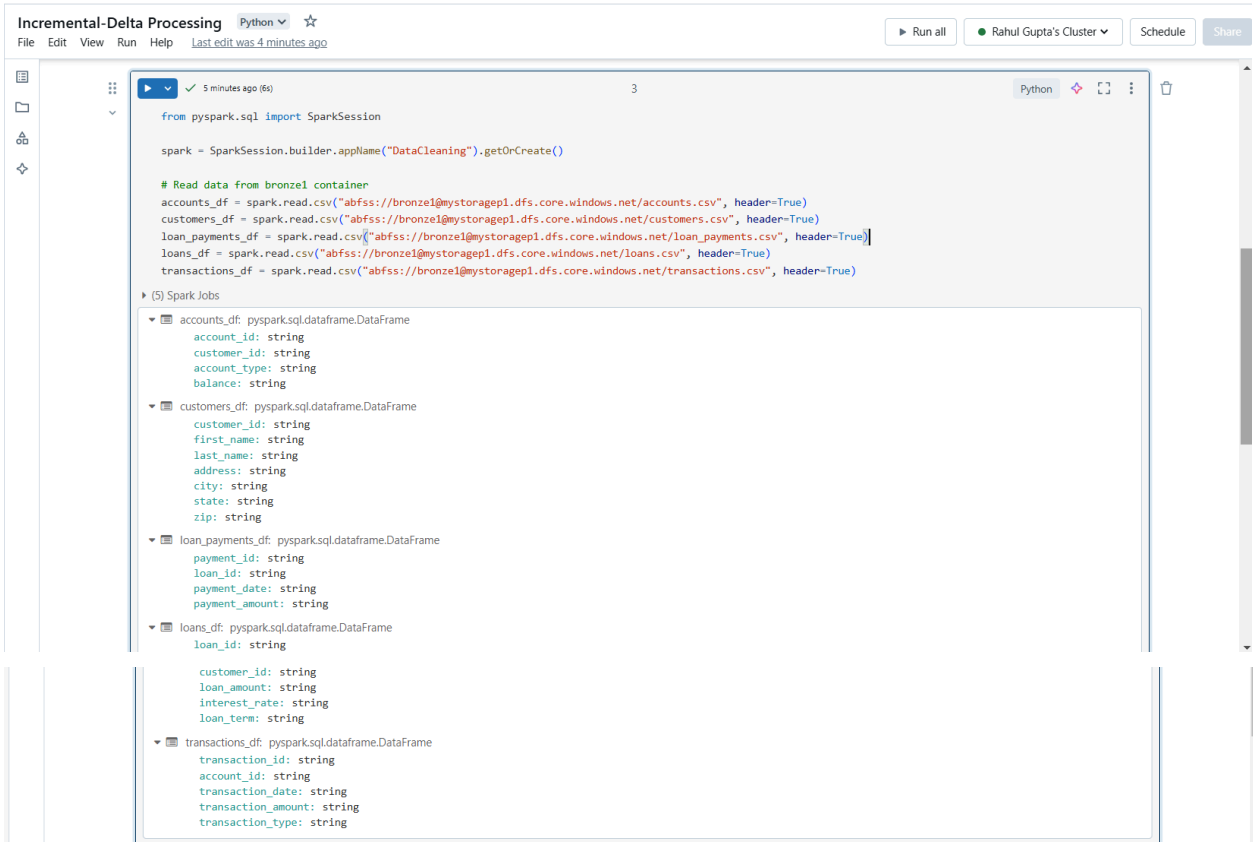


Code 3:

Read Data from Raw (Bronze) Container

```
from pyspark.sql import SparkSession



spark = SparkSession.builder.appName("DataCleaning").getOrCreate()



# Read data from bronze container
```

```
accounts_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/accounts.csv",
header=True)
customers_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/customers.csv",
header=True)
loan_payments_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/loan_payments.c
sv", header=True)
loans_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/loans.csv",
header=True)
transactions_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/transactions.cs
v", header=True)

accounts_df.show(5)
customers_df.show(5)
loan_payments_df.show(5)
loans_df.show(5)
transactions_df.show(5)
```



Data Cleaning and Data Transformation:

Implement logic to handle null values, remove duplicates, and clean data.

- Remove null values

```
accounts_df = accounts_df.dropna()

customers_df = customers_df.dropna()

loan_payments_df = loan_payments_df.dropna()

loans_df = loans_df.dropna()

transactions_df = transactions_df.dropna()
```

- Remove Duplicates

```
accounts_df = accounts_df.dropDuplicates()
```

```
customers_df = customers_df.dropDuplicates()
loan_payments_df = loan_payments_df.dropDuplicates()
loans_df = loans_df.dropDuplicates()
transactions_df = transactions_df.dropDuplicates()
```

- Renaming the column name from zip to postal_code from customer file.

```
customers_df = customers_df.withColumnRenamed("zip", "postal_code")
```

- Change data types for **customer** table Converting 'customer_id' and 'postal_code' to Integer

```
from pyspark.sql import functions as F

# Change data types for customer table Converting 'customer_id' and
'postal_code' to Integer
customers_df = (customers_df
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("zip", F.col("zip").cast("int"))
)

# Display the updated schema to verify changes
customers_df.printSchema()

# Display the updated schema to verify changes
customers_df.show(5)
```

- Change data types for **accounts** table Converting 'customer_id', 'account_id' to integer and 'balance' to double

```
accounts_df = (accounts_df
    .withColumn("account_id", F.col("account_id").cast("int"))
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("balance", F.col("balance").cast("double"))  # Change to
"float" if desired
)
```

```
# Display the updated schema to verify changes
accounts_df.printSchema()

# Display the updated schema to verify changes
accounts_df.show(5)
```

- Change data types for **loan_payments** table converting ‘payment_id’, ‘loan_id’ to Int and ‘payment_date’ to date and ‘payment_amount’ to double.

```
loan_payments_df = (loan_payments_df
    .withColumn("payment_id", F.col("payment_id").cast("int"))
    .withColumn("loan_id", F.col("loan_id").cast("int"))
    .withColumn("payment_date", F.to_date(F.col("payment_date"), "yyyy-MM-dd"))
# Adjust date format as needed
    .withColumn("payment_amount", F.col("payment_amount").cast("double"))  #
Use "float" if desired
)

# Display the updated schema to verify changes
loan_payments_df.printSchema()

# Display the updated schema to verify changes
loan_payments_df.show(5)
```

- Change data types for **loans** table converting ‘loan_id’, ‘customer_id’ and ‘loan_term’ to Int and ‘loan_amount’, ‘interest_rate’ to double

```
loans_df = (loans_df
    .withColumn("loan_id", F.col("loan_id").cast("int"))
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("loan_amount", F.col("loan_amount").cast("double"))
    .withColumn("interest_rate", F.col("interest_rate").cast("double"))
    .withColumn("loan_term", F.col("loan_term").cast("int"))
)

# Display the updated schema to verify changes
```

```
loans_df.printSchema()

# Display the updated schema to verify changes
loans_df.show(5)
```

- Change data types for **transactions** table converting ‘transaction_id’, ‘account_id’ and ‘transaction_date’ to Date and ‘trannsaction_amount’’ to double

```
transactions_df = (transactions_df
    .withColumn("transaction_id", F.col("transaction_id").cast("int"))
    .withColumn("account_id", F.col("account_id").cast("int"))
    .withColumn("transaction_date", F.to_date(F.col("transaction_date"),
"yyyy-MM-dd"))
    .withColumn("transaction_amount",
F.col("transaction_amount").cast("double"))
)

# Display the updated schema to verify changes
transactions_df.printSchema()

# Display the updated schema to verify changes
transactions_df.show(5)
```

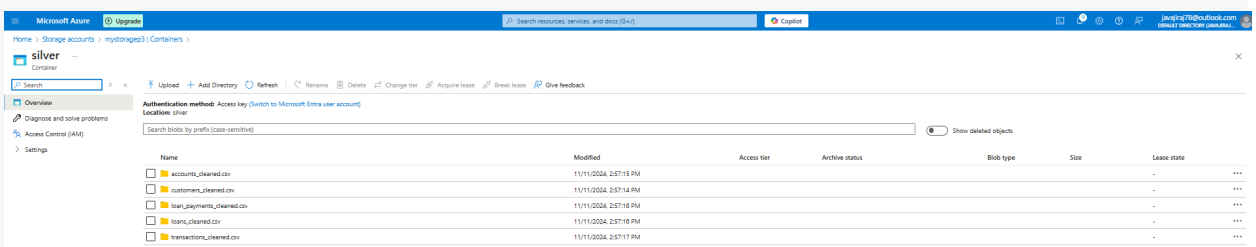
- Writing the path to the silver container:

```
# Define the paths to the Silver container

silver_accounts =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/accounts_delta"

silver_customers =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/customers_delta"

silver_loan_payments =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/loan_payments_delta"
```

```
silver_loans =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/loans_delta"

silver_transactions =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/transactions_delta"
```

- Writing the cleaning data back to the silver container:

```
# Write the cleaned data back to the Silver container

accounts_df.write.format("delta").mode("overwrite").save(silver_accounts)

customers_df.write.format("delta").mode("overwrite").save(silver_customers)

loan_payments_df.write.format("delta").mode("overwrite").save(silver_loan_paym
ents)

loans_df.write.format("delta").mode("overwrite").save(silver_loans)

transactions_df.write.format("delta").mode("overwrite").save(silver_transactio
ns)
```



Databricks Activity (ETL Processing)

Step 1: Create a notebook:

- In the same Folder name “my-rg-dbfolder”
- In this folder creating new notebook
- Select - Create - Notebook
- Rename the notebook as “ETL Processing”

Step 2: **Read Data from Silver Container:**

```
spark.conf.set(
    "fs.azure.account.key.mystoragep3.dfs.core.windows.net",

"/mskNq6Zye/Yg1ZV10L9BIG0KNhXMrg59eX49h0Q+ekAw9BuggYKDX8CuhM8o+qMsK1BzNyA83mP+A
StbAh9Vw==")

# Read Data from Silver Container:
accounts_df =
spark.read.format("delta").load("abfss://silver@mystoragep3.dfs.core.windows.ne
t/delta/accounts_delta")
customers_df =
spark.read.format("delta").load("abfss://silver@mystoragep3.dfs.core.windows.ne
t/delta/customers_delta")
loan_payments_df =
spark.read.format("delta").load("abfss://silver@mystoragep3.dfs.core.windows.ne
t/delta/loan_payments_delta")
loans_df =
spark.read.format("delta").load("abfss://silver@mystoragep3.dfs.core.windows.ne
t/delta/loans_delta")
transactions_df =
spark.read.format("delta").load("abfss://silver@mystoragep3.dfs.core.windows.ne
t/delta/transactions_delta")
```

Step 3: Apply Business Logic for Transformation:

Write a query to calculate the total balance across all accounts for each customer, ensuring that all columns from the accounts and customers tables are selected and included

```
from pyspark.sql import functions as F



# Cast balance to Double to ensure correct aggregation

accounts_df = accounts_df.withColumn("balance",
F.col("balance").cast("double"))



# Join the accounts and customers DataFrames on customer_id

customer_accounts_df = accounts_df.join(customers_df, on="customer_id",
how="inner")



# Calculate the total balance for each customer, include all customer columns,
and keep individual account columns

result_df = (customer_accounts_df

    .groupBy("customer_id", "first_name", "last_name", "address", "city",
"state", "zip")

    .agg(

        F.sum("balance").alias("total_balance")

    )

    .join(accounts_df.select("customer_id", "account_id", "account_type",
"balance"), on="customer_id", how="left")

)
```

```
# Show the result

result_df.show(truncate=False)
```

Step 4: Data Loading: Store the transformed data in the Refined(gold) container of your data lake.

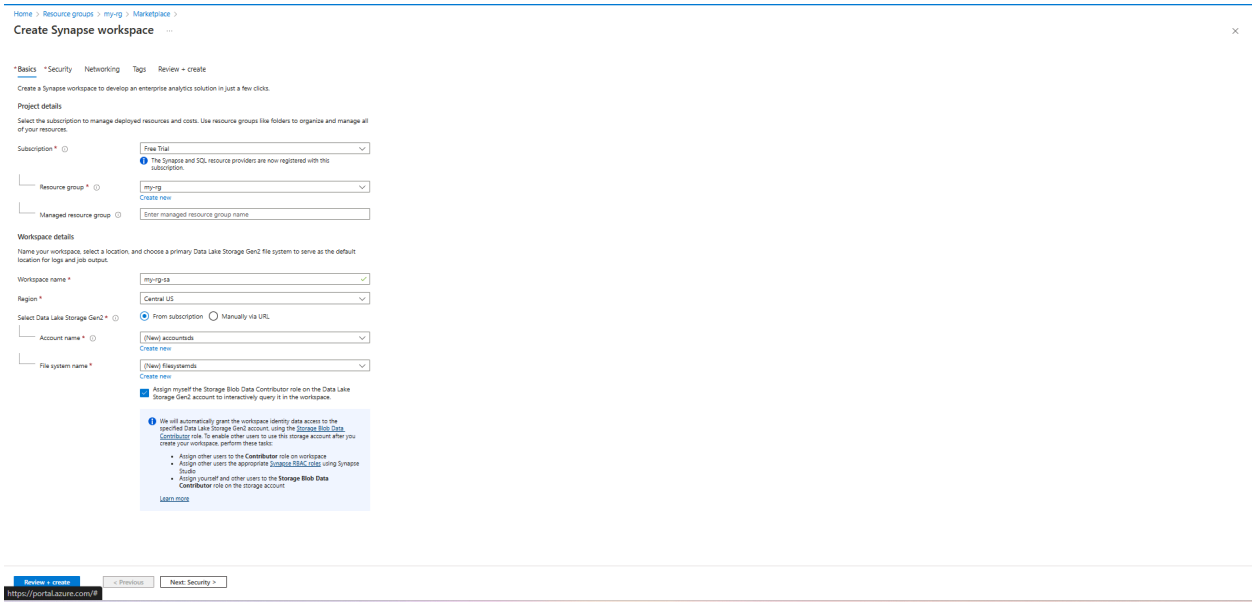
```
 # Define the path for gold container

gold_delta = "abfss://gold@mystoragep3.dfs.core.windows.net/delta/gold_delta"



# Save the DataFrame in Delta format, overwriting if it exists

result_df.write.format("delta").mode("overwrite").save(gold_delta)
```

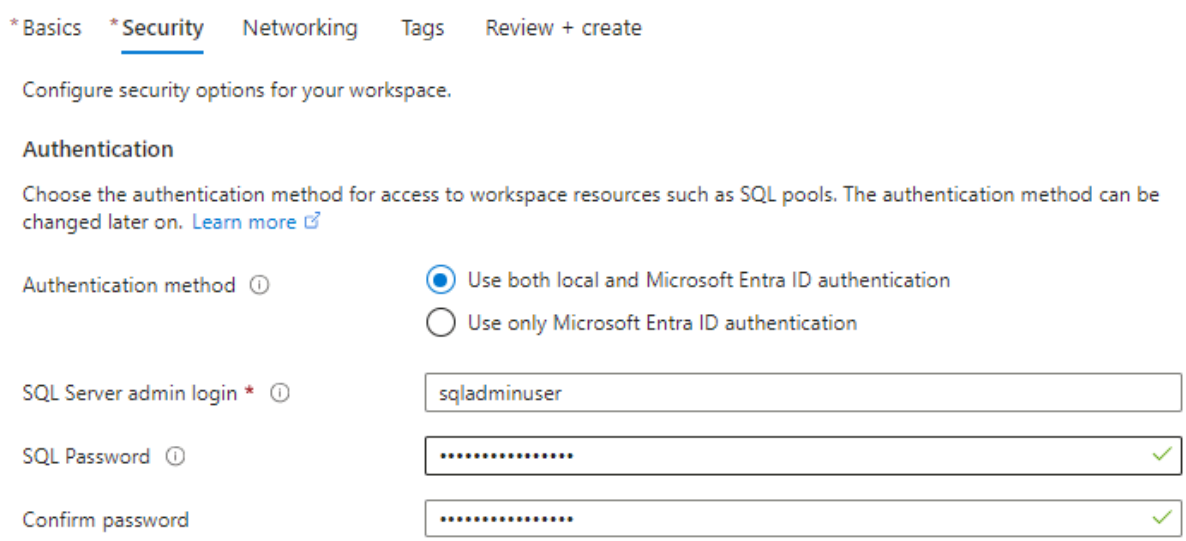
Azure Synapse Analytics

Step 1: Create azure synapse analytics

Basics

- Resource group - Create New - “synapseresourse”
- Workspace name: “sa-my-rg”
- Region: “Central US”
- Select Data Lake Storage Gen2: “From Subscription”
- Account name - create new - “mystoragep3”
- File system name - create new - “silver”
- Next Security.





- Next Networking (no changes - keep it default)
- Next Tags (no changes - keep it default)
- Next Review + create

Step 2: Launch Synapse studio

- Develop - SQL script - copy paste the script -

```
-- Example for creating an external data source for the Silver container
CREATE EXTERNAL DATA SOURCE SilverDataSource
WITH (
   LOCATION = 'abfss://silver@strgacc2831.dfs.core.windows.net'
);
```

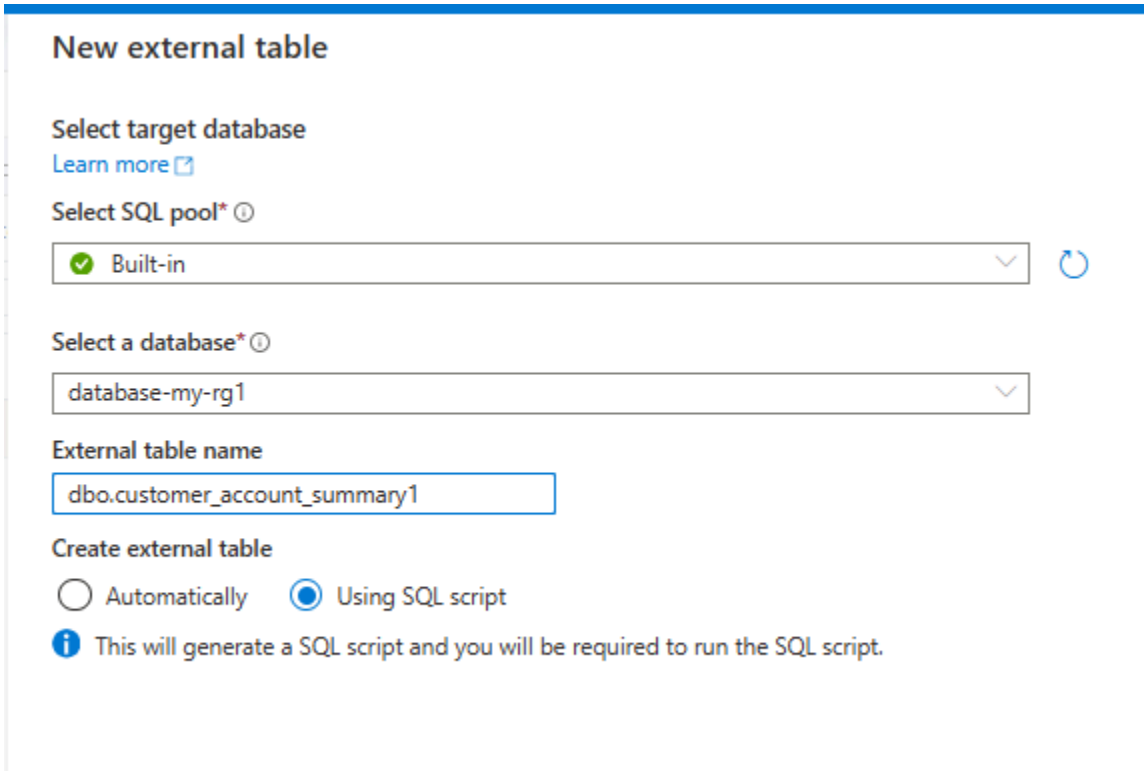
- Data + sql database - serverless - name: database-my-rg1
- For the above script - run the script.
- Run the 2nd script:

```
-- Example for creating an external data source for the Gold container
CREATE EXTERNAL DATA SOURCE GoldDataSource
WITH (
   LOCATION = 'abfss://silver@strgacc2831.dfs.core.windows.net'
);
```

- Run the 3rd Script:

```
-- Example for creating a CSV file format
CREATE EXTERNAL FILE FORMAT CsvFileFormat
WITH (
   FORMAT_TYPE = DELIMITEDTEXT,
   FORMAT_OPTIONS (FIELD_TERMINATOR = ',', STRING_DELIMITER = '"', FIRST_ROW = 2)
);
```

-
- Select - Linked - Select : “sa-my-rg” - You will be able to see the lists of containers which is already created.
- Select - gold - Right click on select : “customer_account_summary“ select : new sql script - create external table
- Continue - External table name: dbo.customer_account_summary1



- Open script - Run the entire script.