Case Study 2 SF

Scenario: Build a real-time log analytics pipeline using Azure Databricks, Snowpark, and Snowflake.

Task: Ingest JSON logs from ADLS, transform them in Databricks using Snowpark, and load them into Snowflake for analysis.

Step 1: Understand the Data & Scenario

- Scenario: You have logs in JSON format stored in Azure Data Lake Storage (ADLS). These logs contain information like:

  event: login/logout

  timestamp: 2025-10-16T09:23:00Z

  user: pooja

- Goal: Read these logs in Databricks, clean/transform them using Snowpark, and load them into a Snowflake table for analytics.

Think of it as a stream of logs coming from your application, which you want to analyze in near real-time.

Step 2: Set Up Your Environment

A. Azure Databricks

- Create a Databricks workspace in Azure.
- Create a cluster (make sure it's running).
- Install the Snowflake connector for Spark and Snowpark library on the cluster.

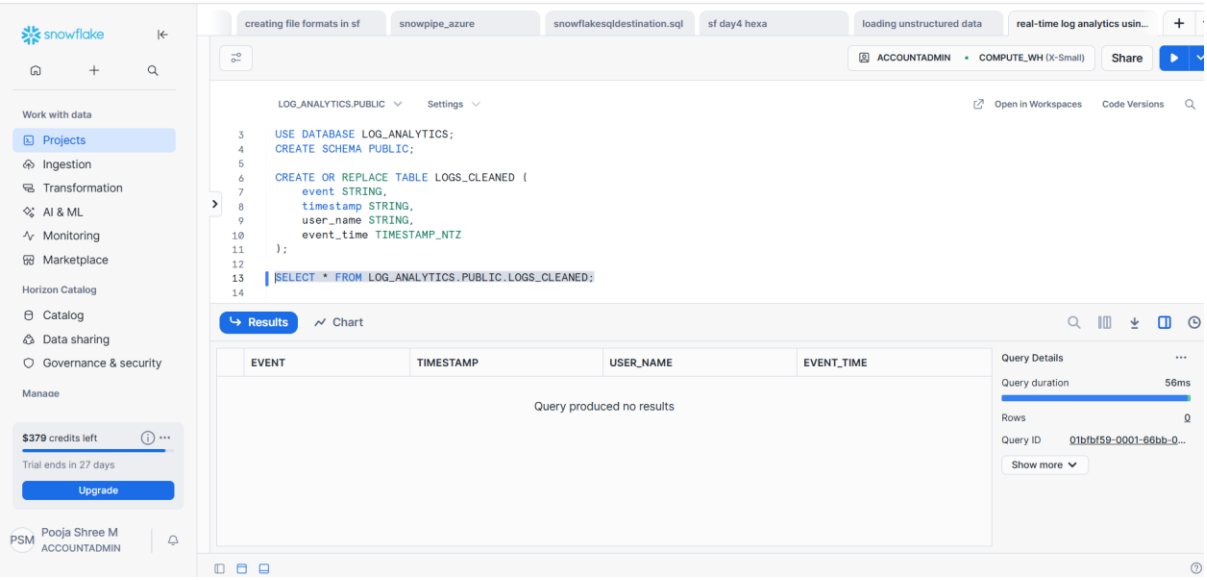
B. Snowflake

- Create a database and schema for logs.
- Create a table for cleaned logs:

```
CREATE OR REPLACE DATABASE LOG_ANALYTICS;
USE DATABASE LOG_ANALYTICS;
CREATE OR REPLACE SCHEMA PUBLIC;
CREATE OR REPLACE TABLE LOGS_CLEANED (
   event STRING,
   timestamp STRING,
   user_name STRING,
   event_time TIMESTAMP_NTZ
);
```

Notes:

- user_name instead of user because USER is reserved in Snowflake.
- event_time will store proper timestamp values.

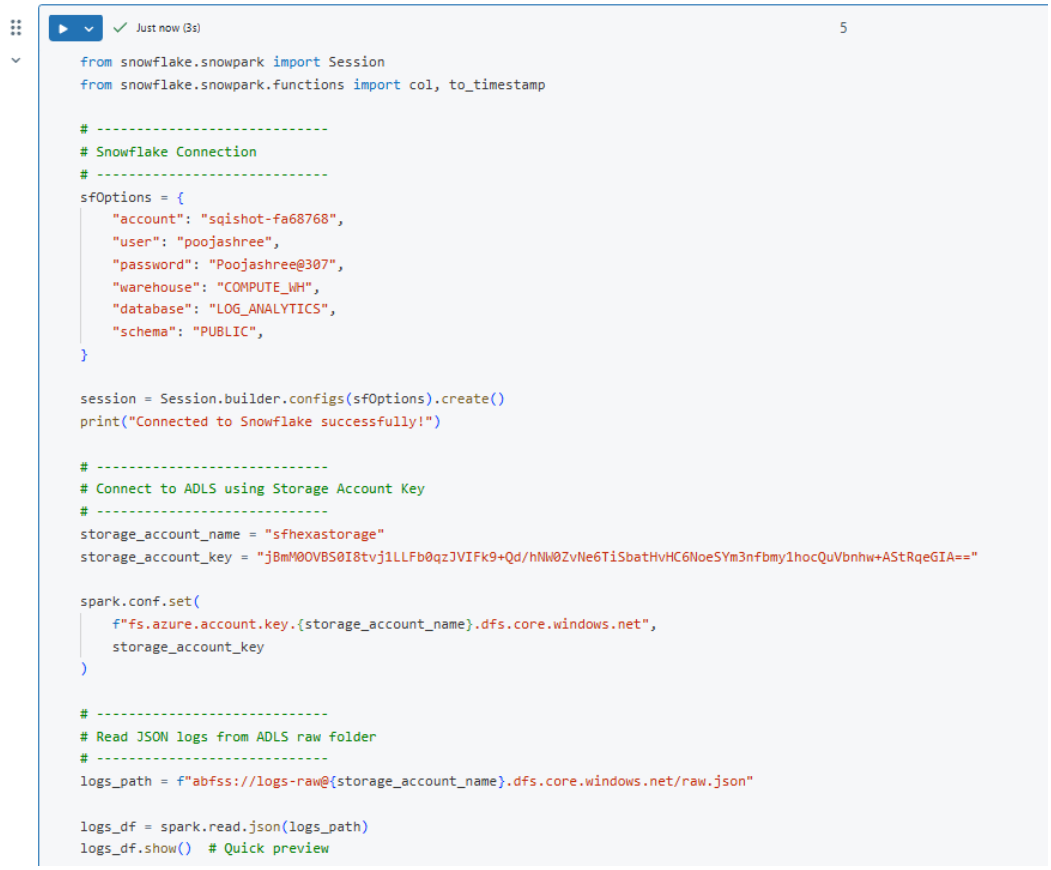


Step 3: Connect Databricks to ADLS

Since you don't have secrets management or role assignment authority, the simplest method is to use Storage Account Key:

```
storage_account_name = "sfhexastorage"

storage_account_key = "<your-storage-account-key>"

spark.conf.set(

  f"fs.azure.account.key.{storage_account_name}.dfs.core.windows.net",

  storage_account_key

)

# Container + path

logs_path = f"abfss://logs-raw@{storage_account_name}.dfs.core.windows.net/raw.json"


# Read JSON

logs_df = spark.read.json(logs_path)

display(logs_df)
```

After this step, your logs are loaded into a Spark DataFrame in Databricks.



```
Just now (3s)                                                    5

from snowflake.snowpark import Session
from snowflake.snowpark.functions import col, to_timestamp

# ----------------------------
# Snowflake Connection
# ----------------------------
sfOptions = {
    "account": "sqishot-fa68768",
    "user": "poojashree",
    "password": "Poojashree@307",
    "warehouse": "COMPUTE_WH",
    "database": "LOG_ANALYTICS",
    "schema": "PUBLIC",
}

session = Session.builder.configs(sfOptions).create()
print("Connected to Snowflake successfully!")

# ----------------------------
# Connect to ADLS using Storage Account Key
# ----------------------------
storage_account_name = "sfhexastorage"
storage_account_key = "jBmM0OVBS0I8tvj1LLFb0qzJVIFk9+Qd/hNW0ZvNe6TiSbatHvHC6NoeSYm3nfbmy1hocQuVbnhw+AStRqeGIA=="

spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.dfs.core.windows.net",
    storage_account_key
)

# ----------------------------
# Read JSON logs from ADLS raw folder
# ----------------------------
logs_path = f"abfss://logs-raw@{storage_account_name}.dfs.core.windows.net/raw.json"

logs_df = spark.read.json(logs_path)
logs_df.show()  # Quick preview
```

Step 4: Transform Data Using Snowpark

- Snowpark is like pandas but for Snowflake, integrated into Databricks via Spark.
- You can do transformations, e.g., converting timestamp strings to proper TIMESTAMP_NTZ, renaming columns to match Snowflake table:

```
from snowflake.snowpark import Session

from snowflake.snowpark.functions import col, to_timestamp

# Transform the DataFrame

sp_df_cleaned = logs_df.select(

    col("event").alias("event"),

    col("timestamp").alias("timestamp"),

    col("user").alias("user_name"),

    to_timestamp(col("timestamp")).alias("event_time")

)
```

At this point, the DataFrame matches the Snowflake table schema.

Step 5: Connect to Snowflake from Databricks

- Use Snowflake connector / Snowpark session:

```
from snowflake.snowpark import Session
connection_parameters = {
    "account": "<your_account>",
    "user": "<your_user>",
    "password": "<your_password>",
    "warehouse": "<your_warehouse>",
    "database": "LOG_ANALYTICS",
    "schema": "PUBLIC"
}
session = Session.builder.configs(connection_parameters).create()
```

```
# Convert Spark DataFrame to Pandas (for small batch)
logs_pd = logs_df.toPandas()

# ------------------------------
# Create Snowpark DataFrame from Pandas
# ------------------------------
sp_df = session.create_dataframe(logs_pd)

# ------------------------------
# Rename + Format Columns
# ------------------------------
sp_df_cleaned = (
    sp_df
    .with_column_renamed("user", "USER_NAME")
    .with_column_renamed("event", "EVENT")
    .with_column("EVENT_TIME", to_timestamp(col('"timestamp"')))
)

sp_df_cleaned.show()
'''
# ------------------------------
# Create database/schema/table if not exists
# ------------------------------
session.sql("CREATE DATABASE IF NOT EXISTS LOG_ANALYTICS").collect()
session.sql("USE DATABASE LOG_ANALYTICS").collect()
session.sql("CREATE SCHEMA IF NOT EXISTS PUBLIC").collect()
session.sql("""
    CREATE OR REPLACE TABLE LOGS_CLEANED (
        EVENT STRING,
        USER_NAME STRING,
        TIMESTAMP STRING,
        EVENT_TIME TIMESTAMP_NTZ
    )
""").collect()
'''
```

Step 6: Write the Data to Snowflake

```
sp_df_cleaned.write.mode("append").save_as_table("LOGS_CLEANED")
print("Data written to Snowflake successfully!")
```

Now your cleaned logs are in Snowflake and ready for analysis.

Step 7: Analysis in Snowflake

- You can now query the logs, e.g.:

```
SELECT user_name, COUNT(*) AS logins
FROM LOGS_CLEANED
WHERE event = 'login'
GROUP BY user_name;
```

- Or track activity trends over time:

```
SELECT DATE_TRUNC('hour', event_time) AS hour, COUNT(*) AS events
FROM LOGS_CLEANED
GROUP BY hour
ORDER BY hour;
```

Step 8: Optional - Automate for Real-Time

- For near real-time ingestion, you can use:
    1. Databricks Structured Streaming to watch ADLS folder.
    2. Transform each batch using Snowpark.
    3. Append to Snowflake continuously.

This avoids manually re-running jobs every time logs are generated.

```
    # ----------------------------
    # Write cleaned data to Snowflake table
    # ----------------------------
    # Automatic insert, append mode
    sp_df_cleaned.write.mode("append").option("column_quote", '"').save_as_table("LOGS_CLEANED")
    print("Data written to Snowflake table LOGS_CLEANED successfully!")
```

▸ (3) Spark Jobs

▸ logs_df: pyspark.sql.dataframe.DataFrame = [event: string, timestamp: string ... 1 more field]

▸ logs_pd: pandas.core.frame.DataFrame = [event: object, timestamp: object ... 1 more field]

```
Connected to Snowflake successfully!
+------+--------------------+-----+
| event|           timestamp| user|
+------+--------------------+-----+
| login|2025-10-16T09:23:00Z|pooja|
|logout|2025-10-16T09:30:00Z| ravi|
+------+--------------------+-----+

----------------------------------------------------------------------
|"event"  |"timestamp"           |"user"  |"EVENT_TIME"         |
----------------------------------------------------------------------
|login    |2025-10-16T09:23:00Z  |pooja   |2025-10-16 09:23:00  |
|logout   |2025-10-16T09:30:00Z  |ravi    |2025-10-16 09:30:00  |
----------------------------------------------------------------------

Data written to Snowflake table LOGS_CLEANED successfully!
```