

Capstone Project Submission Report

1. Project Title

- **Title of the Project:** Scalable ETL and Real-Time Analytics for Financial Data

2. Student Information

- **Name:** Parul Shah
 - **Student ID:** 127202
 - **Course/Program:** Data Engineering Graduate Trainee
 - **Instructor Name:** Piyush Raj Katyayan
-

3. Problem Definition and Objectives

(Rubric: Problem Definition and Objectives)

Problem Definition

Financial institutions process vast amounts of transactional data daily, requiring a scalable, real-time solution for fraud detection, risk assessment, and regulatory compliance. The challenge is to efficiently ingest, process, and analyze data from multiple sources while ensuring security, scalability, and cost-effectiveness. This project aims to build an end-to-end Azure-based ETL and analytics pipeline to enable real-time fraud detection and data-driven decision-making.

Objectives:

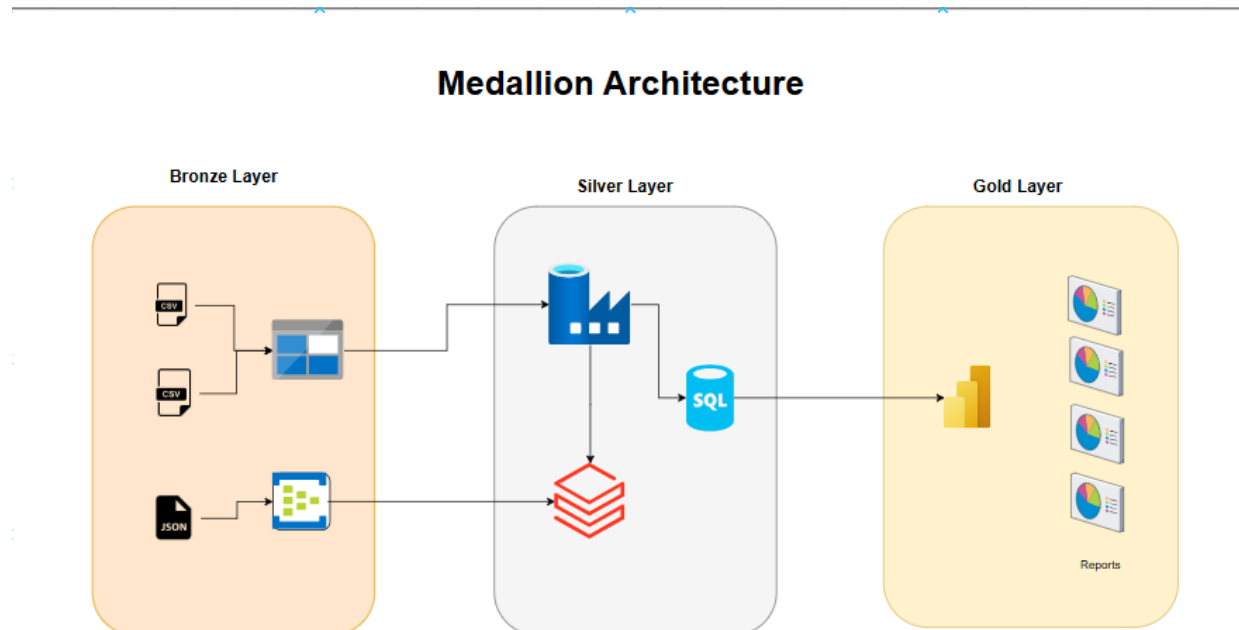
The objectives include:

- **ETL Pipeline for BFSI** – Build an ETL pipeline to ingest, transform, and store financial transaction data.
- **Batch & Streaming Analytics** – Enable both historical batch processing and real-time transaction analysis.
- **Risk Assessment & Compliance** – Analyze merchant and transaction risk to ensure regulatory compliance.

- **Scalability & Cost Optimization** – Design an optimized, high-availability data pipeline using Azure Databricks, Event Hub, and ML models

Diagram: High-Level Overview of Problem Context

The given diagram shows how the approach would start from bronze layer where it mainly concerns with the raw data, wherein silver layer mainly focusses on how to clean the raw data and gold layer focuses on insights.



4. Data Collection, Exploratory Data Analysis (EDA), and Preprocessing

(Rubric: Data Collection & Preprocessing)

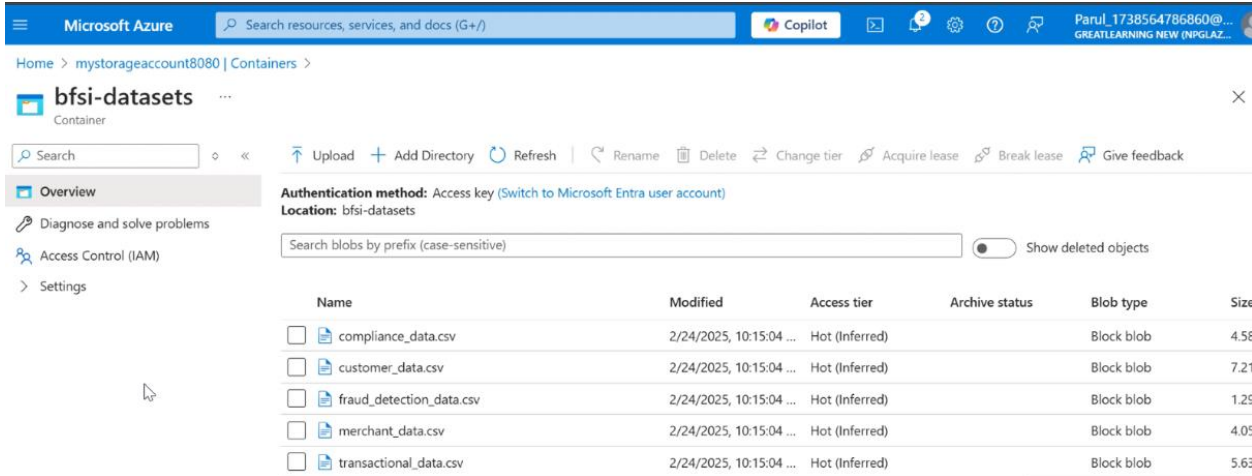
Data Collection:

Type of Data Collected

- **Transactional & Real-time Data** – Financial transactions for fraud detection.
- **Customer Data** – User details for segmentation.
- **Merchant Data** – Merchant risk profiling.
- **Fraud Detection Data** – Fraud scores and detection insights.
- **Regulatory Compliance Data** – Compliance checks and statuses.

Challenges Faced

- *Data Integration – Merging batch and real-time data.*
- *Data Quality Issues – Missing values, inconsistencies, duplicates.*
- *Scalability – Handling high-volume transactions efficiently.*



Microsoft Azure | Search resources, services, and docs (G+/I) | Copilot | Parul_1738564786860@... GREATLEARNING NEW (NPGLAZ...)

Home > mystorageaccount8080 | Containers >

bfsi-datasets Container

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

Overview | Diagnose and solve problems | Access Control (IAM) | Settings

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bfsi-datasets

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

Name	Modified	Access tier	Archive status	Blob type	Size
compliance_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	4.58
customer_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	7.21
fraud_detection_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	1.29
merchant_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	4.05
transactional_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	5.63

Table: Data Description

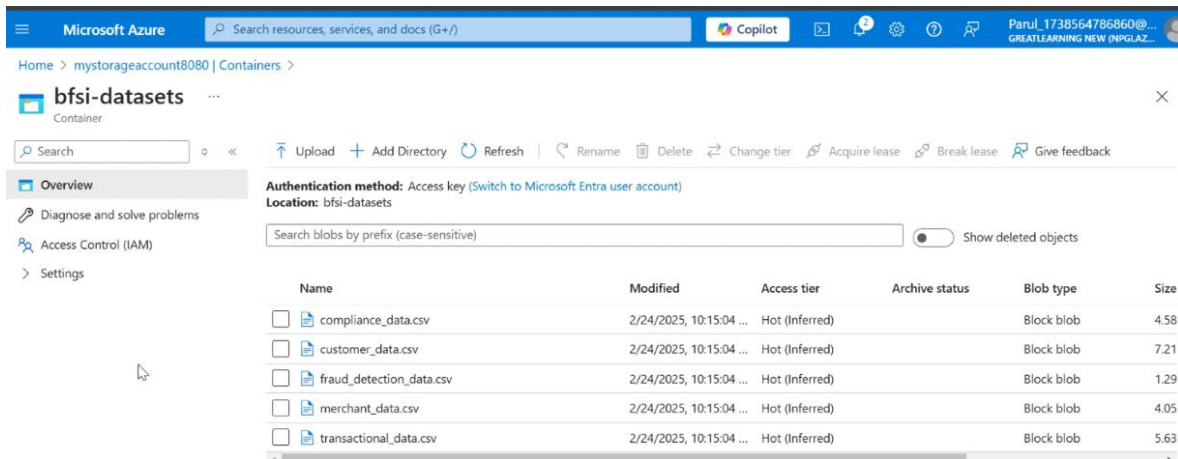
Attribute Name	Data Type	Description	Missing Values (%)
transaction_id	Integer	Unique ID for transactions	0%
account_id	Integer	Unique ID for accounts	2%
transaction_date	Date	Date of transaction	0%
transaction_amount	Float	Amount of transaction	1%
transaction type	String	Type of transaction	0%
merchant_id	Integer	Unique ID for merchants	3%
first_name	String	Customer's first name	5%
last_name	String	Customer's last name	5%
email	String	Customer's email address	0%
signup_date	Date	Date of account signup	0%
account_type	String	Type of account	0%
merchant_name	String	Name of the merchant	0%
category	String	Merchant category	0%
risk_rating	Float	Merchant risk rating	4%
fraud_id	Integer	Unique ID for fraud cases	0%
fraud_score	Float	Fraud risk score	0%
detection_date	Date	Date of fraud detection	0%
compliance_id	Integer	Unique ID for compliance	0%
Compliance_check	String	Compliance status	0%
status	String	Status of compliance check	0%

5. Data Storage and Optimization

(Rubric: Storage & Optimization)

Data Storage Solution:

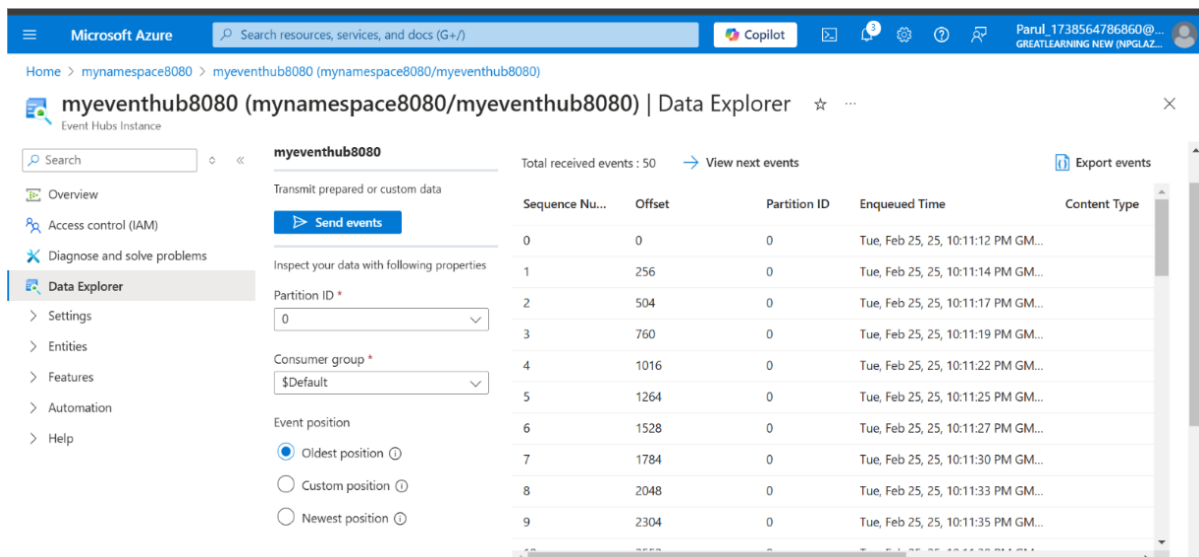
- **Azure Blob Storage (Namespace Hierarchy)** – Used for storing CSV batch data due to its scalability, cost-efficiency, and easy integration with Azure Data Factory for ETL processing.



Microsoft Azure portal showing the 'bfsi-datasets' container. The interface includes a search bar, navigation menu, and a table of blobs.

Name	Modified	Access tier	Archive status	Blob type	Size
compliance_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	4.58
customer_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	7.21
fraud_detection_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	1.29
merchant_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	4.05
transactional_data.csv	2/24/2025, 10:15:04 ...	Hot (Inferred)		Block blob	5.63

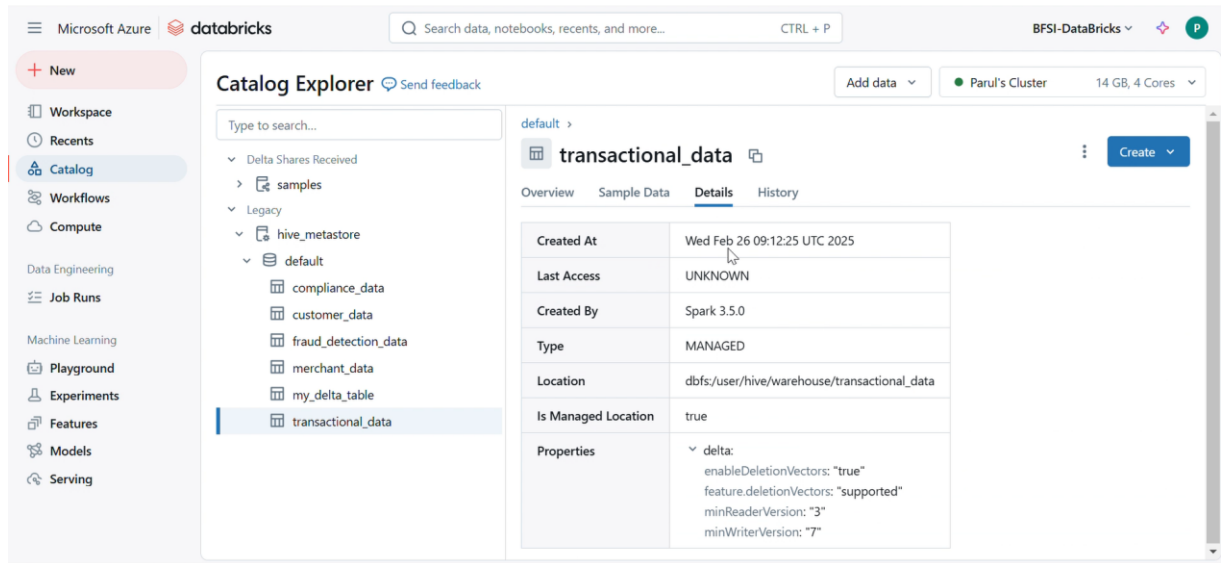
- **Azure Event Hub** – Facilitates real-time data ingestion, ensuring seamless data streaming into Databricks for transformations.



Microsoft Azure portal showing the 'myeventhub8080' Data Explorer. The interface includes a search bar, navigation menu, and a table of events.

Sequence Nu...	Offset	Partition ID	Enqueued Time	Content Type
0	0	0	Tue, Feb 25, 25, 10:11:12 PM GM...	
1	256	0	Tue, Feb 25, 25, 10:11:14 PM GM...	
2	504	0	Tue, Feb 25, 25, 10:11:17 PM GM...	
3	760	0	Tue, Feb 25, 25, 10:11:19 PM GM...	
4	1016	0	Tue, Feb 25, 25, 10:11:22 PM GM...	
5	1264	0	Tue, Feb 25, 25, 10:11:25 PM GM...	
6	1528	0	Tue, Feb 25, 25, 10:11:27 PM GM...	
7	1784	0	Tue, Feb 25, 25, 10:11:30 PM GM...	
8	2048	0	Tue, Feb 25, 25, 10:11:33 PM GM...	
9	2304	0	Tue, Feb 25, 25, 10:11:35 PM GM...	

- **Azure Databricks** – Serves as the processing layer, performing batch and real-time transformations before loading data into the final storage.



- **Azure SQL Database** – Stores the transformed and structured data for analytics, reporting, and querying

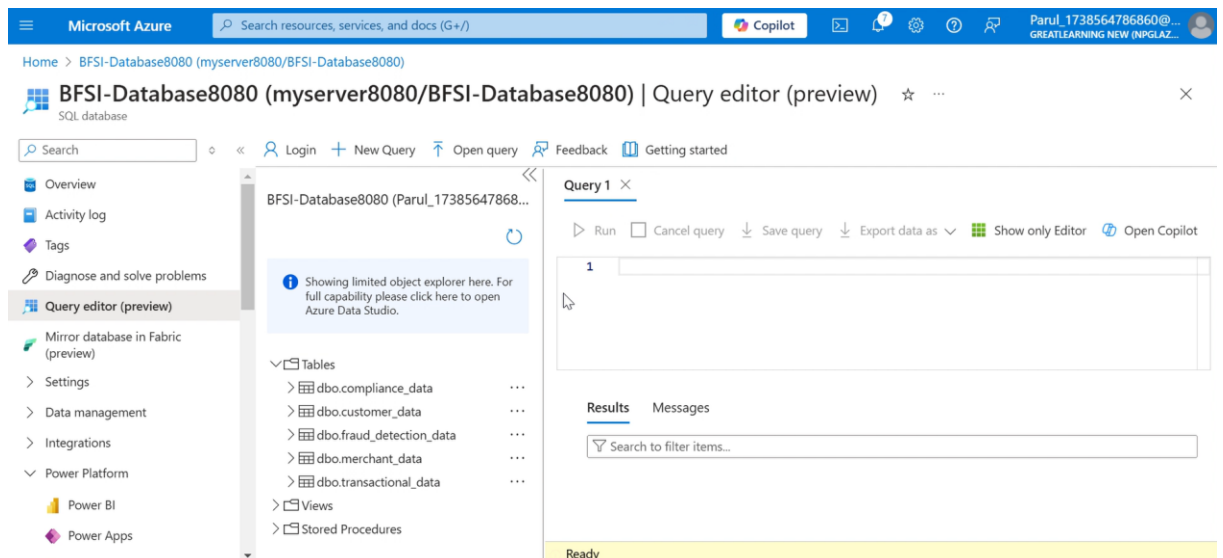
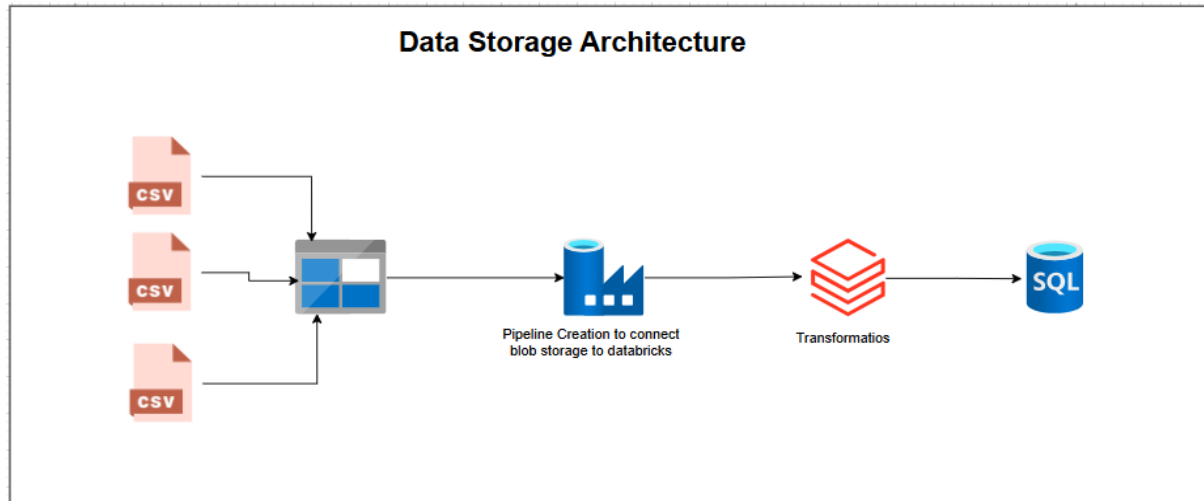


Diagram: Data Storage Architecture



6. Real-Time Data Processing and Streaming

(Rubric: Real-Time Data Processing)

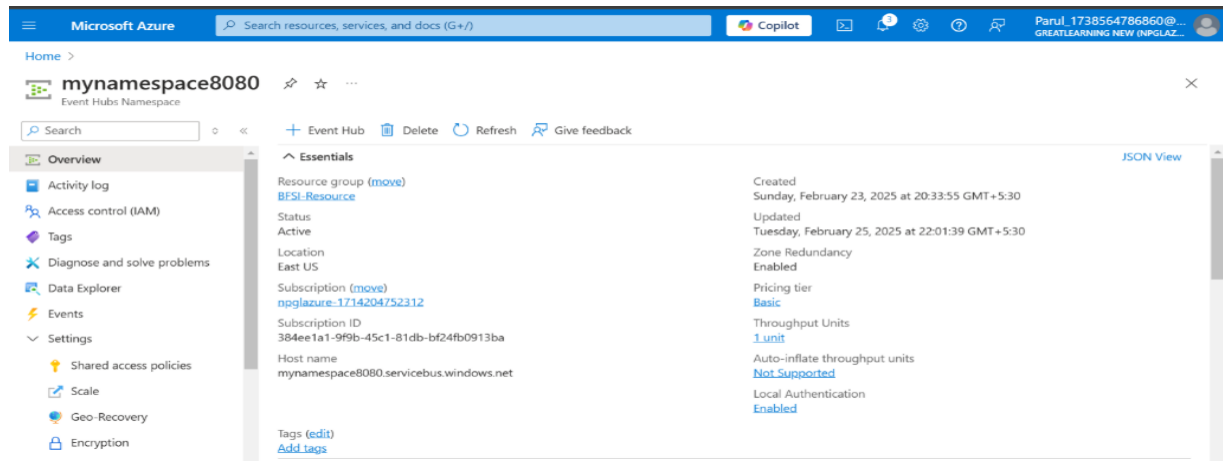
Real-Time Data Processing:

Real-time data processing involves continuously ingesting, processing, and analyzing data as it arrives, enabling immediate insights and actions. This ensures that businesses can detect patterns, anomalies, or trends as they happen, which is crucial for fraud detection, recommendation systems, and other time-sensitive applications.

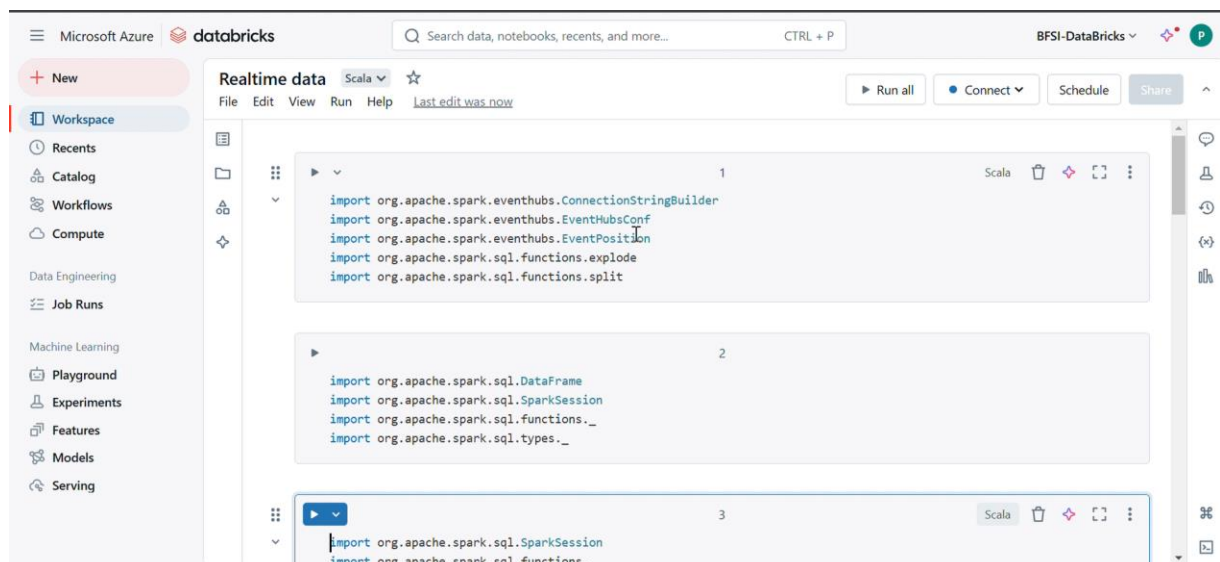
In this implementation, the following technologies are used:

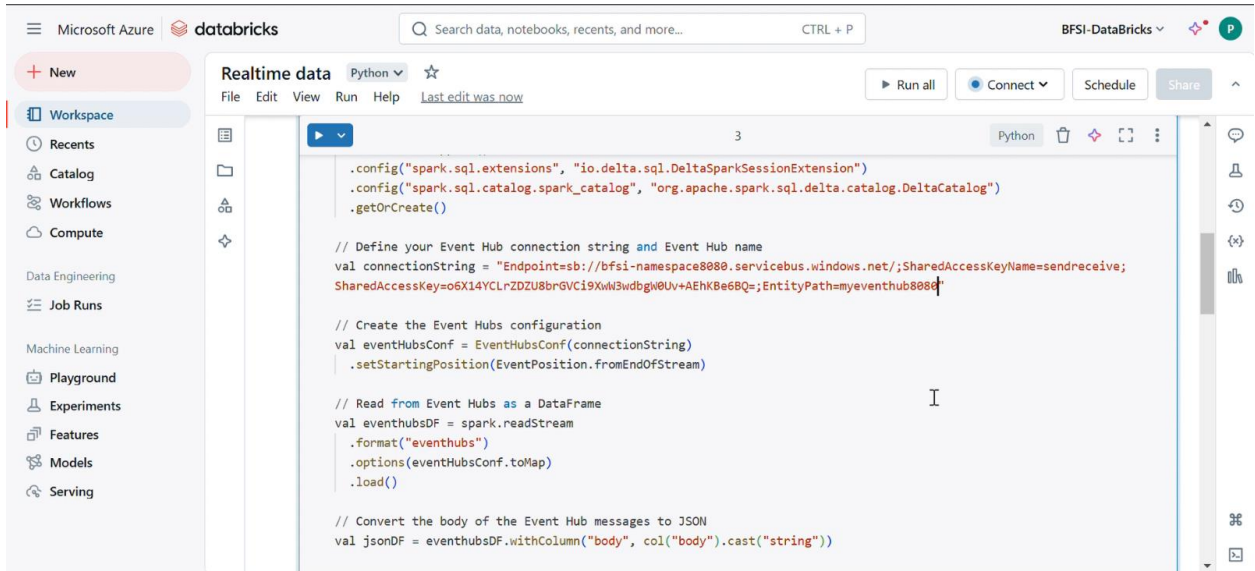
Technologies Used

- Azure Event Hub → Captures and ingests streaming data.



- *Databricks (Apache Spark Streaming - Scala & Python) → Processes the streaming data in real time.*





```

Microsoft Azure databricks
Search data, notebooks, recents, and more... CTRL + P
BFSI-DataBricks
+ New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving

Realtime data Python
File Edit View Run Help Last edit was now
Run all Connect Schedule Share

.config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
.config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")
.getOrCreate()

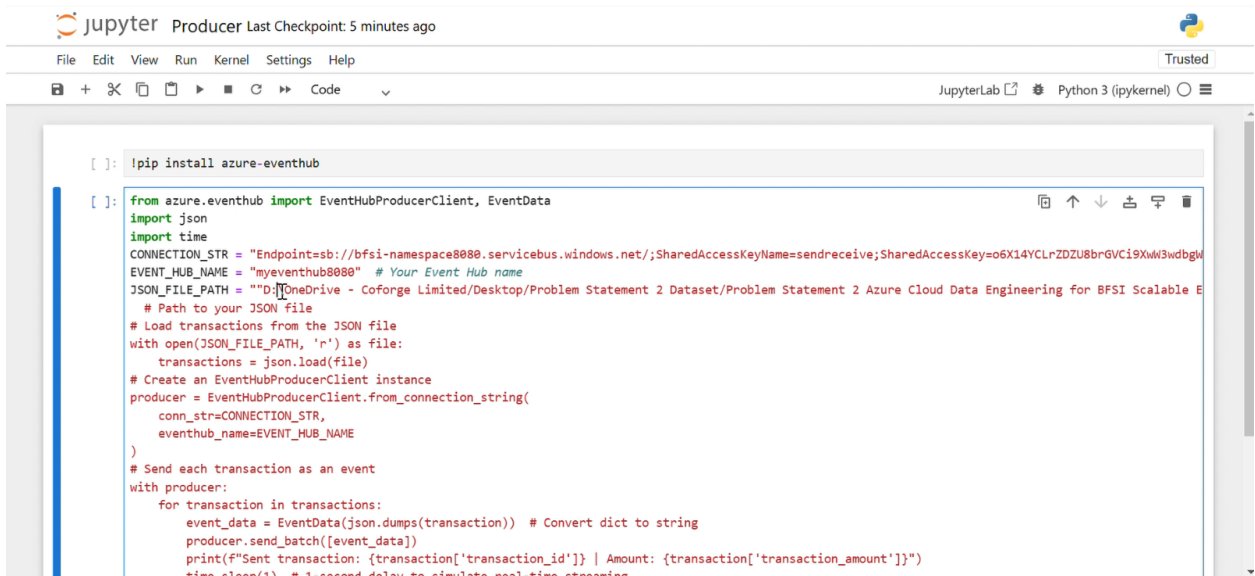
// Define your Event Hub connection string and Event Hub name
val connectionString = "Endpoint=sb://bfsi-namespace8080.servicebus.windows.net;/SharedAccessKeyName=sendreceive;
SharedAccessKey=o6X14YCLrZDZU8brGVCi9XwM3wdbgw8Uv+AEhKBe6BQ=;EntityPath=myeventhub8080"

// Create the Event Hubs configuration
val eventHubsConf = EventHubsConf(connectionString)
.setStartingPosition(EventPosition.FromEndOfStream)

// Read from Event Hubs as a DataFrame
val eventHubsDF = spark.readStream
    .format("eventhubs")
    .options(eventHubsConf.toMap())
    .load()

// Convert the body of the Event Hub messages to JSON
val jsonDF = eventHubsDF.withColumn("body", col("body").cast("string"))
  
```

- Jupyter Notebook (Producer Code) → Sends JSON transaction data to Event Hub.



```

jupyter Producer Last Checkpoint: 5 minutes ago
File Edit View Run Kernel Settings Help
Trusted
JupyterLab Python 3 (ipykernel)

[ ]: !pip install azure-eventhub

[ ]: from azure.eventhub import EventHubProducerClient, EventData
import json
import time
CONNECTION_STR = "Endpoint=sb://bfsi-namespace8080.servicebus.windows.net;/SharedAccessKeyName=sendreceive;SharedAccessKey=o6X14YCLrZDZU8brGVCi9XwM3wdbgw8Uv+AEhKBe6BQ=;EntityPath=myeventhub8080"
EVENT_HUB_NAME = "myeventhub8080" # Your Event Hub name
JSON_FILE_PATH = "D:\\OneDrive - Coforge Limited\\Desktop\\Problem Statement 2 Dataset\\Problem Statement 2 Azure Cloud Data Engineering for BFSI Scalable E
# Path to your JSON file
# Load transactions from the JSON file
with open(JSON_FILE_PATH, 'r') as file:
    transactions = json.load(file)
# Create an EventHubProducerClient instance
producer = EventHubProducerClient.from_connection_string(
    conn_str=CONNECTION_STR,
    eventhub_name=EVENT_HUB_NAME
)
# Send each transaction as an event
with producer:
    for transaction in transactions:
        event_data = EventData(json.dumps(transaction)) # Convert dict to string
        producer.send_batch([event_data])
        print(f"Sent transaction: {transaction['transaction_id']} | Amount: {transaction['transaction_amount']}")
        time.sleep(1) # 1-second delay to simulate real-time streaming
  
```


JupyterLab Producer Last Checkpoint: 16 minutes ago

File Edit View Run Kernel Settings Help

JupyterLab Python 3 (pykernel)

```

Sent transaction: RT_TXN20080 | Amount: 150000
Sent transaction: RT_TXN20081 | Amount: 5000
Sent transaction: RT_TXN20082 | Amount: 20000
Sent transaction: RT_TXN20083 | Amount: 1000
Sent transaction: RT_TXN20084 | Amount: 200000
Sent transaction: RT_TXN20085 | Amount: 10000
Sent transaction: RT_TXN20086 | Amount: 5000
Sent transaction: RT_TXN20087 | Amount: 2000
Sent transaction: RT_TXN20088 | Amount: 15000
Sent transaction: RT_TXN20089 | Amount: 20000
Sent transaction: RT_TXN20090 | Amount: 10000
Sent transaction: RT_TXN20091 | Amount: 50000
Sent transaction: RT_TXN20092 | Amount: 150000
Sent transaction: RT_TXN20093 | Amount: 5000
Sent transaction: RT_TXN20094 | Amount: 20000
Sent transaction: RT_TXN20095 | Amount: 20000
Sent transaction: RT_TXN20096 | Amount: 500
Sent transaction: RT_TXN20097 | Amount: 1000
Sent transaction: RT_TXN20098 | Amount: 200
Sent transaction: RT_TXN20099 | Amount: 20000
All transactions sent successfully!

```

- *SQL Database (Azure SQL) → Stores the processed data for analytics.*

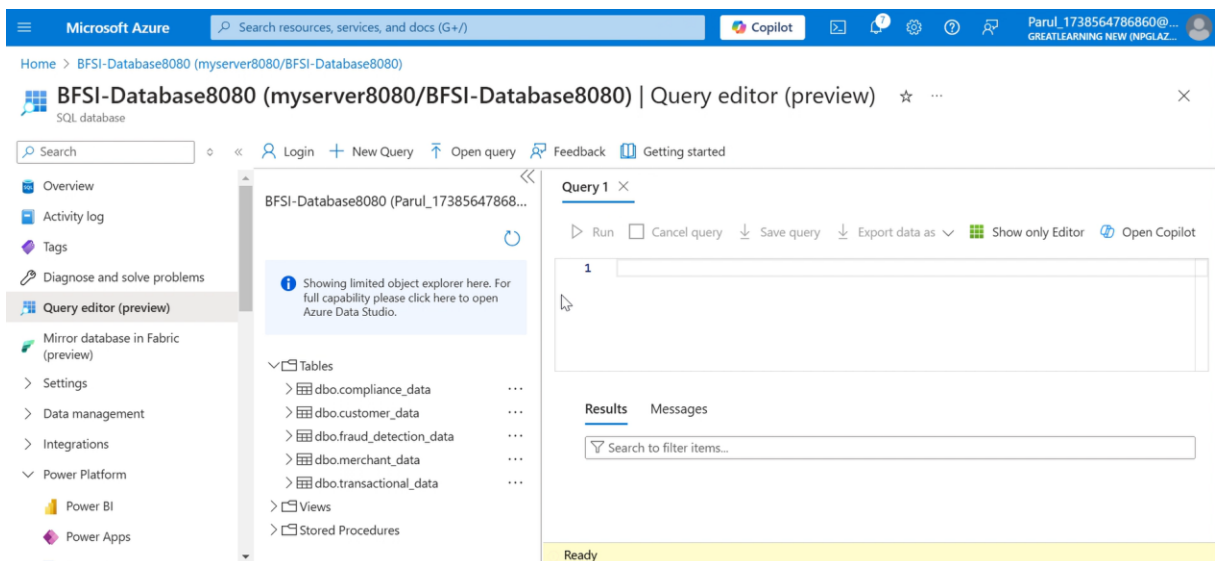
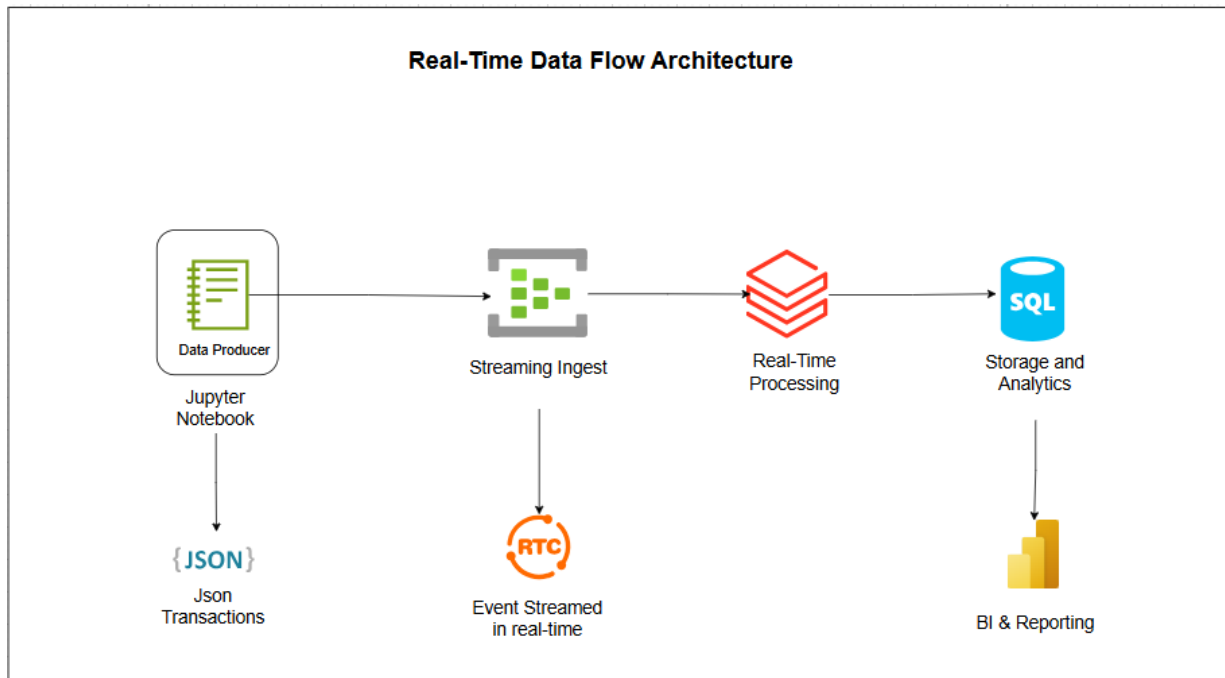


Diagram: Real-Time Data Flow



7. Solution Design & Integration

(Rubric: Solution Design & Integration)

Complete ETL Process with Orchestration:

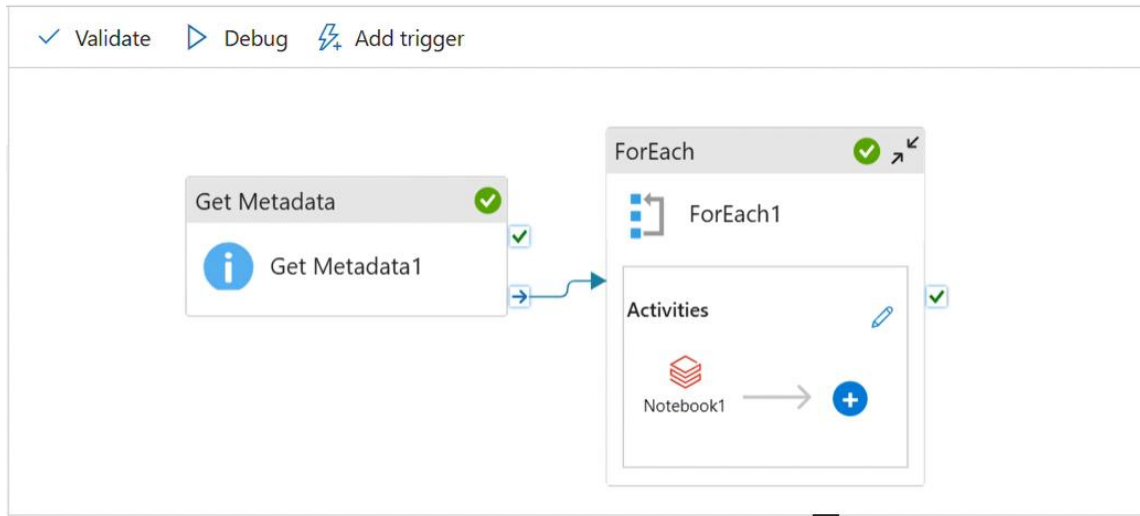
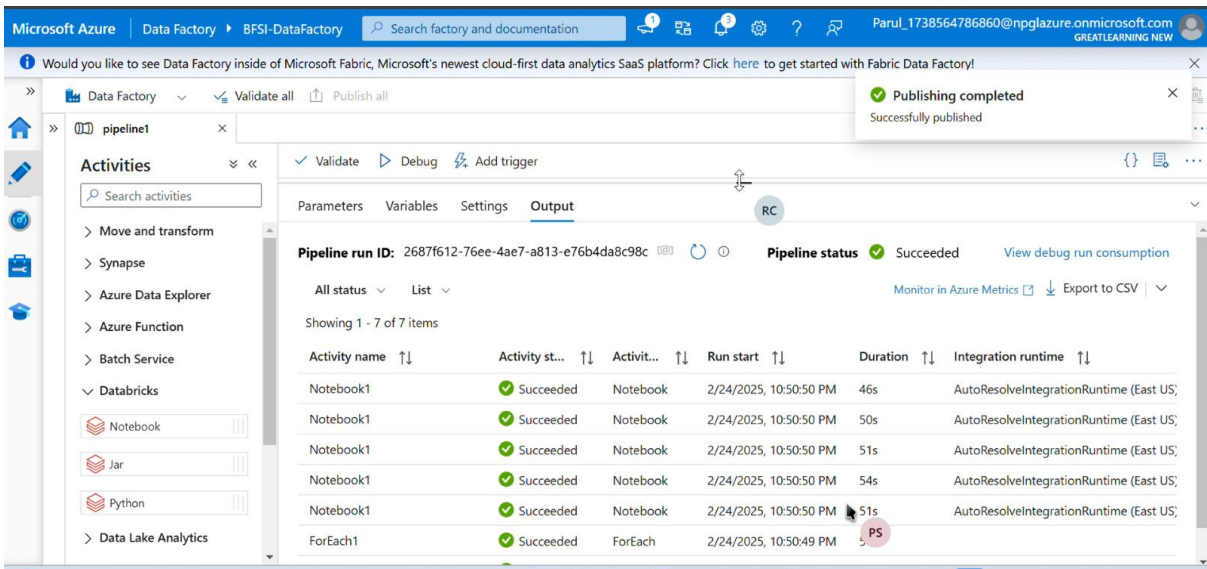
ETL Pipeline Overview:

*The ETL pipeline follows a structured approach to ingest, transform, and store data efficiently using **Azure Data Factory, Databricks, and Azure SQL Database**.*

ETL Steps:

Data Extraction:

1. *Metadata-driven pipeline in Azure Data Factory (ADF) retrieves data source details.*
2. *ForEach activity loops through different datasets and triggers Databricks notebooks.*

Microsoft Azure | Data Factory | BFSI-DataFactory | Search factory and documentation | Parul_1738564786860@npglazure.onmicrosoft.com | GREATLEARNING NEW

Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click [here](#) to get started with Fabric Data Factory!

Publishing completed
Successfully published

Activities | Search activities | Validate | Debug | Add trigger

Parameters | Variables | Settings | Output

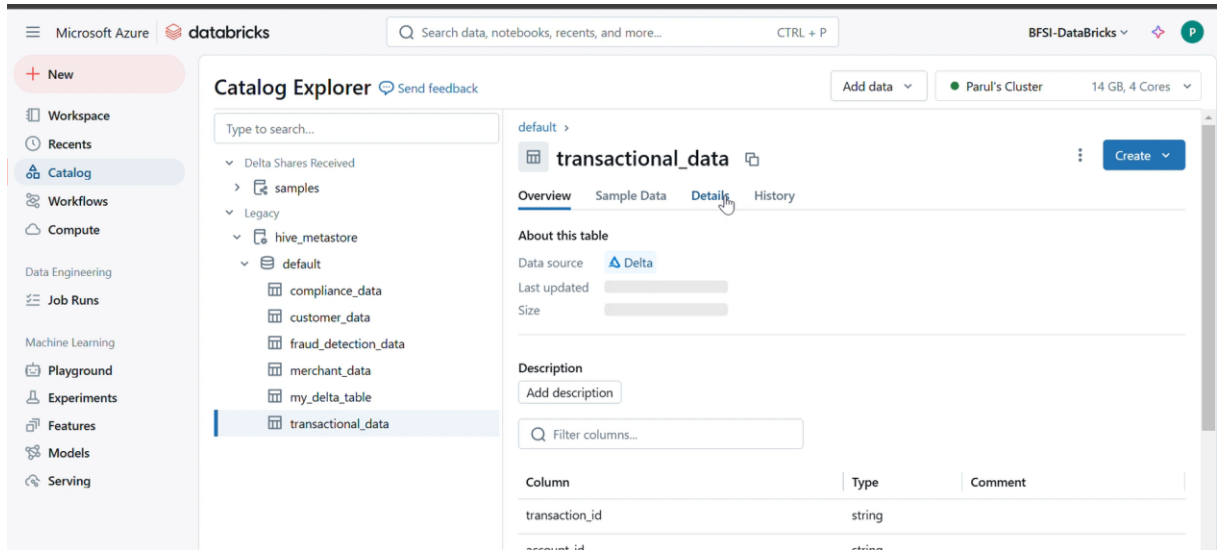
Pipeline run ID: 2687f612-76ee-4ae7-a813-e76b4da8c98c | Pipeline status: Succeeded | View debug run consumption

All status | List | Showing 1 - 7 of 7 items

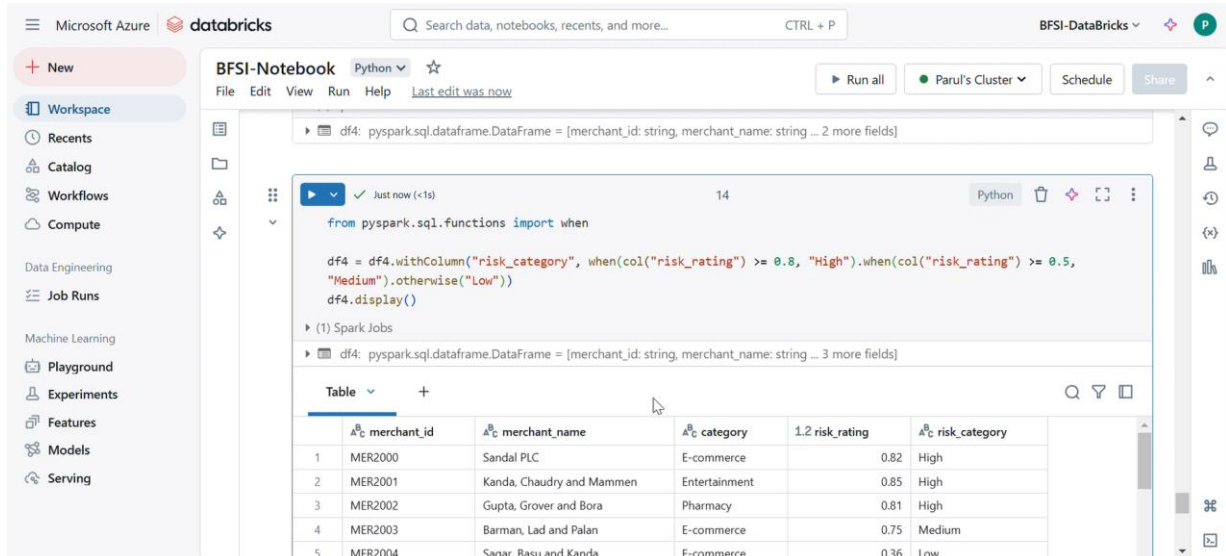
Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime
Notebook1	Succeeded	Notebook	2/24/2025, 10:50:50 PM	46s	AutoResolveIntegrationRuntime (East US)
Notebook1	Succeeded	Notebook	2/24/2025, 10:50:50 PM	50s	AutoResolveIntegrationRuntime (East US)
Notebook1	Succeeded	Notebook	2/24/2025, 10:50:50 PM	51s	AutoResolveIntegrationRuntime (East US)
Notebook1	Succeeded	Notebook	2/24/2025, 10:50:50 PM	54s	AutoResolveIntegrationRuntime (East US)
Notebook1	Succeeded	Notebook	2/24/2025, 10:50:50 PM	51s	AutoResolveIntegrationRuntime (East US)
ForEach1	Succeeded	ForEach	2/24/2025, 10:50:49 PM		

Data Transformation:

1. Databricks (PySpark) processes the data (cleaning, aggregations, joins, etc.).
2. The transformed data is structured into DataFrames.



The screenshot shows the Databricks Catalog Explorer interface. On the left, a sidebar lists navigation options like Workspace, Recents, Catalog, Workflows, Compute, Data Engineering, Job Runs, Machine Learning, Playground, Experiments, Features, Models, and Serving. The main area displays the 'transactional_data' table under the 'default' schema. The 'Details' tab is active, showing 'About this table' information: Data source is Delta, Last updated, and Size. Below this, a 'Description' section has an 'Add description' button. A table lists columns: 'transaction_id' (string) and 'account_id' (string).

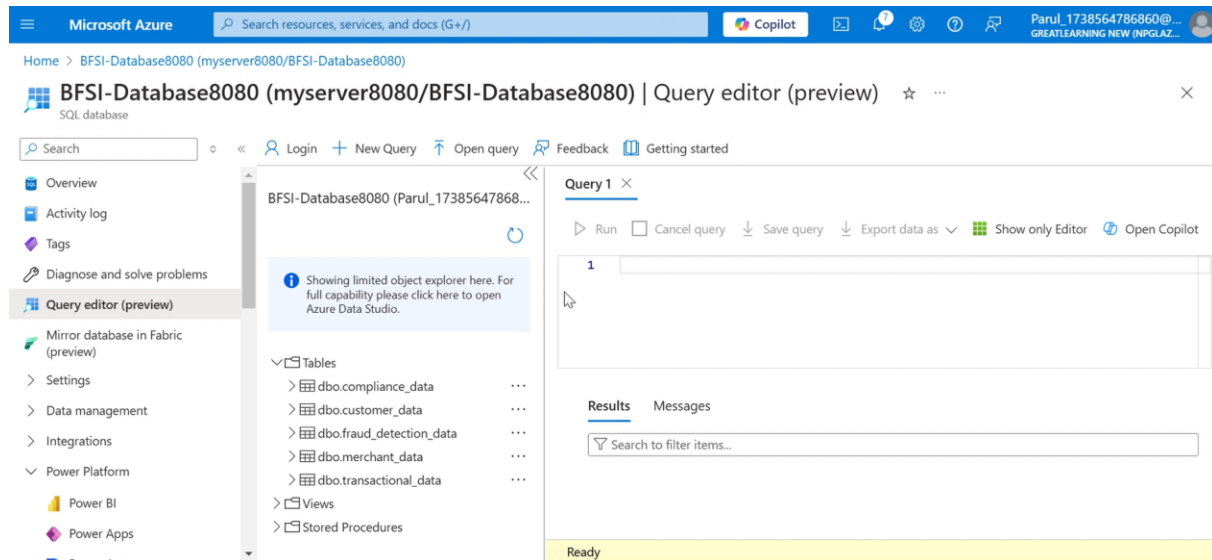


The screenshot shows the Databricks BFSI-Notebook interface. The top bar includes 'Run all', 'Parul's Cluster', 'Schedule', and 'Share' buttons. The notebook code defines a DataFrame 'df4' and applies a filter based on 'risk_rating'. The output is displayed as a table with 5 rows and 5 columns: 'merchant_id', 'merchant_name', 'category', 'risk_rating', and 'risk_category'.

	merchant_id	merchant_name	category	risk_rating	risk_category
1	MER2000	Sandal PLC	E-commerce	0.82	High
2	MER2001	Kanda, Chaudry and Mammen	Entertainment	0.85	High
3	MER2002	Gupta, Grover and Bora	Pharmacy	0.81	High
4	MER2003	Barman, Lad and Palan	E-commerce	0.75	Medium
5	MER2004	Sagar, Basu and Kanda	E-commerce	0.36	Low

Data Loading:

1. Processed data is stored in Azure SQL Database for further reporting and analytics.



Automated Workflows:

1. *ADF orchestrates the ETL flow, scheduling and managing dependencies.*
2. *Databricks notebooks handle real-time data transformations.*
3. *SQL Database stores final data, making it available for BI tools.*

Automated Workflows:

Step 1: Automated Data Ingestion (Real-time & Batch)

1. Real-time data ingestion:

- *Azure Event Hub continuously captures incoming streaming data.*
- *Event Hub triggers Databricks Notebook for immediate processing.*

2. Batch data ingestion:

- *Azure Data Factory (ADF) Pipeline runs on a schedule to pick up batch files (CSV) from Azure Data Lake Storage.*
- *Uses Metadata-driven ingestion (dynamic pipeline using metadata tables).*

Step 2: Automated ETL Pipeline Execution

1. **ADF Pipeline:**

- Uses a *ForEach* activity to process multiple files dynamically.
- Calls *Databricks Notebooks* via a *Notebook* activity for transformation.

2. **Databricks (Apache Spark):**

- Automatically reads data from *Event Hub & Data Lake*.
- Cleans, transforms, and stores structured data in *Azure SQL Database*.
- Incremental processing ensures only new data is processed.

Step 3: Automated Data Storage & Optimization

1. **Processed data stored in:** *Azure SQL Database (for structured analytics).*

Step 4: Automated Reporting & Monitoring

1. **Power BI Dashboards:**

- Connects directly to *Azure SQL Database*.
- Refreshes reports automatically to provide real-time analytics.

Table: Component Breakdown

Component	Responsibility	Technology Used
Data Ingestion	Collect raw data	Azure blob Storage
ETL Processing	Transform and load data	Azure Data Factory
Data Storage	Store processed data	Azure SQL Database

8. Implementation and Results

(Rubric: Implementation & Results)

Workflow/Process Flow:

1. **Data Collection**

- **Source Identification:** Identify data sources such as banking APIs, structured databases, and event-driven streams.
- **Data Ingestion:** Use Azure Data Factory to ingest data from various sources. This includes:
 - a. **Batch Processing:** For structured databases.
 - b. **Real-Time Streaming:** For event-driven streams using Azure Event Hubs.

2. Data Storage

- **Raw Data Storage:** Store ingested raw data in Azure Blob Storage for initial storage and backup.

3. Data Preprocessing

- **Data Cleaning:** Use Azure Data Factory to clean data by removing duplicates, handling missing values, and correcting errors.
- **Data Transformation:** Use Azure Databricks to transform data into a suitable format. This includes:
 - **Normalization:** Standardizing data formats.
 - **Aggregation:** Summarizing data for analysis.
- **Data Integration:** Combine data from different sources to create a unified dataset in Azure Databricks.

9. Conclusion and Future Work

(Rubric: Conclusion & Future Work)

Key Findings :

1. **Automated ETL Workflow** – The system enables seamless data ingestion, transformation, and storage with zero manual intervention.
2. **Live Reporting** – Power BI dashboards refresh automatically to provide real-time insights.
3. **Resilient & Fault-Tolerant System** – Using ADF retries, Databricks error handling, and Azure Monitor alerts, failures are detected and resolved quickly.

Future Enhancements:

1. Machine Learning Integration for Predictive Fraud Detection:

Train a fraud detection model in Azure Machine Learning to improve anomaly detection. Deploy the model in Databricks and Azure ML Endpoint for real-time inference.

2. Multi-Cloud Support & Cross-Region Replication:

Extending data pipeline support to AWS Kinesis or Google Pub/Sub for multi-cloud ingestion. Implement geo-redundancy to replicate data across multiple Azure regions for high availability.

3. Advanced Security & Compliance

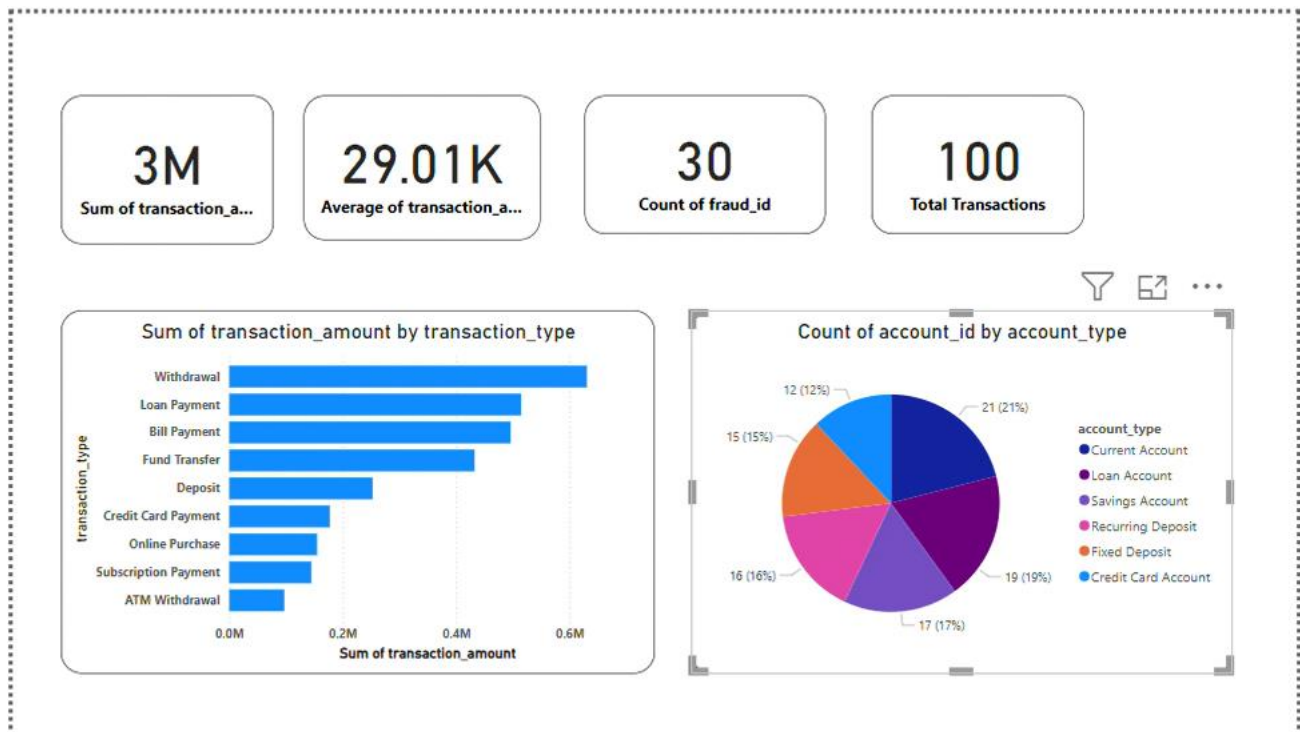
Implement row-level security in Azure SQL for data access control.

Introducing Data Masking and Encryption for sensitive transaction data.

10. Final Presentation

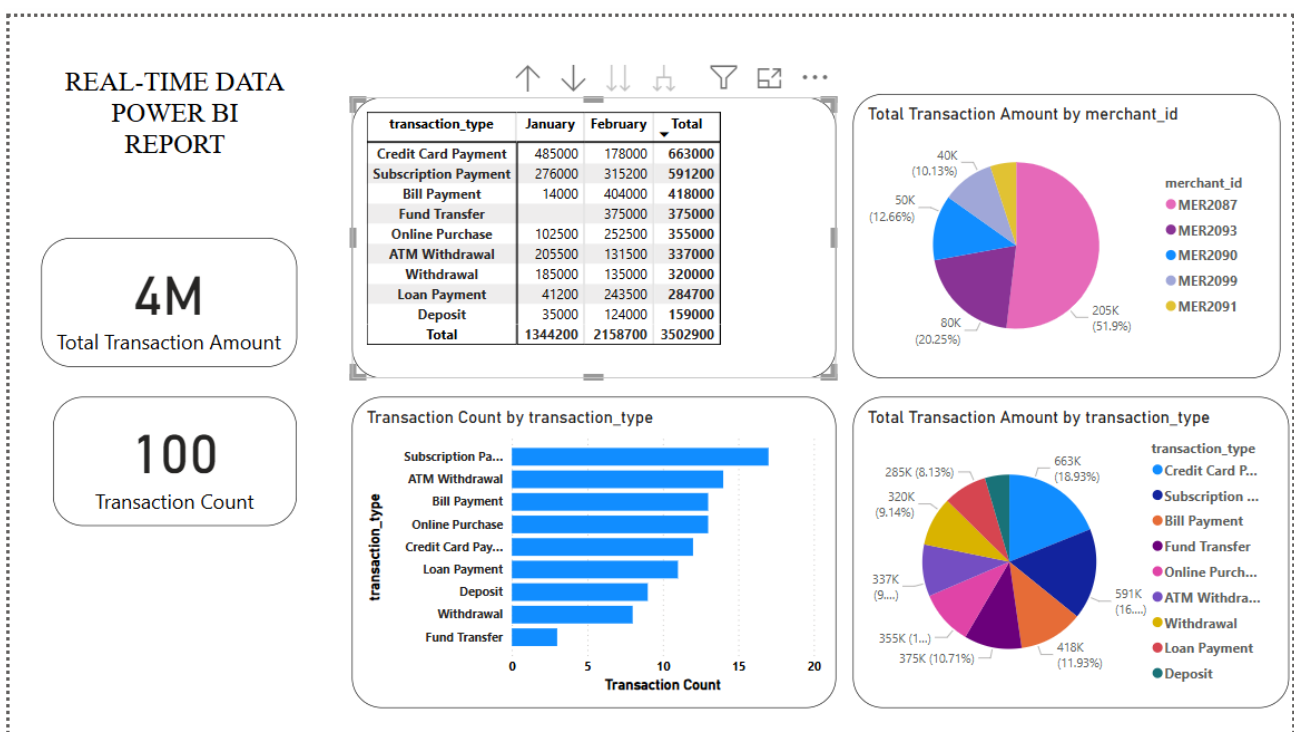
(Rubric: Presentation & Documentation)

Dashboard Screenshots:





Real-time Dashboard



11. Appendices

• Appendix A: Code Snippets

```

FinalTransformationNeww.ipynb  FinalTransformation.ipynb
D: > OneDrive - Coforge Limited > Desktop > FinalTransformation.ipynb
+ Code + Markdown ...
Select Kernel

spark.conf.set("fs.azure.account.key.mystorageaccount8080.blob.core.windows.net", "KxmYfoE31l+PEHyJ+8B11jhXCSq+r89VMz68DV9svkI

filename = dbutils.widgets.get("filename")

print(filename)

df = spark.read.csv(f"wasbs://bfsi-datasets@mystorageaccount8080.blob.core.windows.net/{filename}", header=True, inferSchema=1

df.write.csv(f"/dbfs/tmp/{filename.split('/')[-1]}")

Python

df = spark.read.csv("/dbfs/tmp/compliance_data.csv", header=False, inferSchema=True)
df.display()

Python

...

from pyspark.sql.functions import lit
# Rename columns
df = df.withColumnRenamed("_c0", "compliance_id") \
        .withColumnRenamed("_c1", "transaction_id") \
        .withColumnRenamed("_c2", "transaction_amount")

```

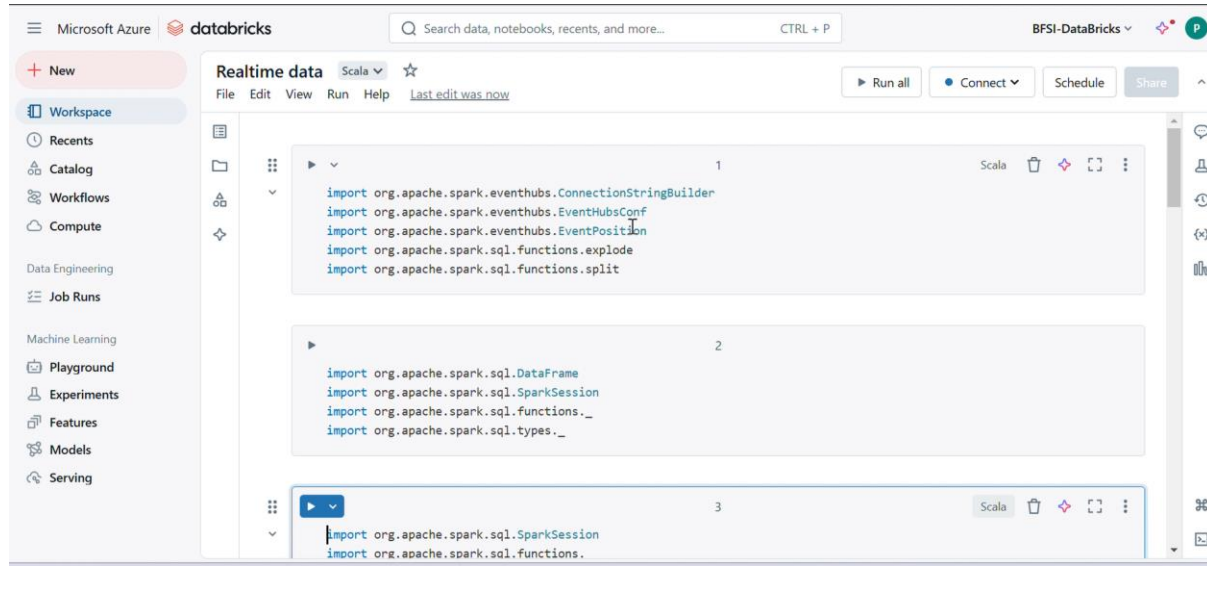
```

Jupyter Producer Last Checkpoint: 5 minutes ago
File Edit View Run Kernel Settings Help
Trusted
JupyterLab Python 3 (ipykernel)

[*]: !pip install azure-eventhub

[ ]: from azure.eventhub import EventHubProducerClient, EventData
import json
import time
CONNECTION_STR = "Endpoint=sb://bfsi-namespace8080.servicebus.windows.net/;SharedAccessKeyName=sendreceive;SharedAccessKey=o6X14YCLrZDZU8brGVCi9XwM3wdbgw
EVENT_HUB_NAME = "myeventhub8080" # Your Event Hub name
JSON_FILE_PATH = "D:/OneDrive - Coforge Limited/Desktop/Problem Statement 2 Dataset/Problem Statement 2 Azure Cloud Data Engineering for BFSI Scalable ET
# Path to your JSON file
# Load transactions from the JSON file
with open(JSON_FILE_PATH, 'r') as file:
    transactions = json.load(file)
# Create an EventHubProducerClient instance
producer = EventHubProducerClient.from_connection_string(
    conn_str=CONNECTION_STR,
    eventhub_name=EVENT_HUB_NAME
)
# Send each transaction as an event
with producer:
    for transaction in transactions:
        event_data = EventData(json.dumps(transaction)) # Convert dict to string
        producer.send_batch([event_data])
        print(f"Sent transaction: {transaction['transaction_id']} | Amount: {transaction['transaction_amount']}")
        time.sleep(1) # 1-second delay to simulate real-time streaming

```



12. Total Score Summary

- **Mid Report:** /50
- **Final Report:** /50
- **Overall Score:** /100