Financial Transaction Monitoring(17.07.25)

Project Overview:

Financial fraud poses a significant threat to businesses and consumers, necessitating advanced detection mechanisms. This document details the development of a comprehensive, end-to-end financial fraud detection system designed to identify and mitigate fraudulent transactions in real-time. The core objective is to build a robust, automated, and scalable system capable of rapidly analyzing transaction data and flagging suspicious activities.

The solution uses a **cloud-native architecture on Azure**, integrating:

- **Azure Databricks** for data processing and machine learning
- **Apache Airflow** for orchestration
- **Azure Storage** for the data lake.
- **Power BI** for visualization

A **Medallion Architecture** (Bronze, Silver, Gold) within Databricks structures data processing—from raw ingestion to enriched fraud prediction—ensuring data quality and enabling efficient analysis. The automated pipeline supports ongoing monitoring and system enhancement.

Project Architecture:

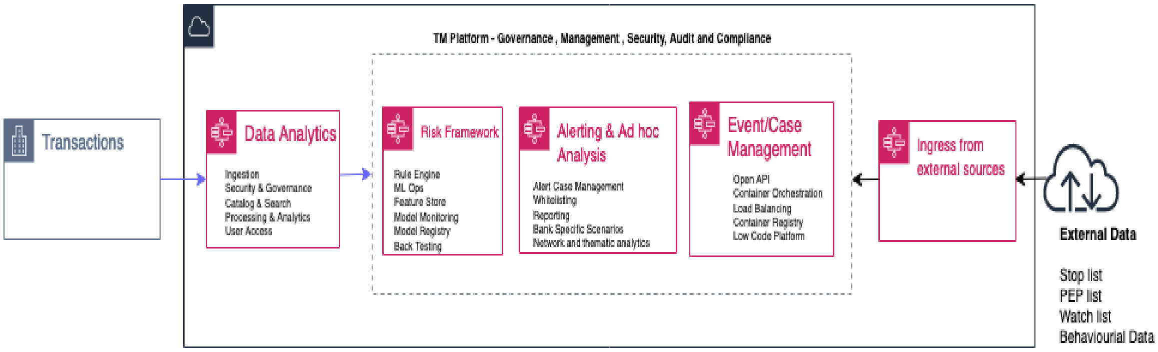
The financial fraud detection system is a **cloud-native architecture** built on **Azure services** for scalability and reliability. It begins with **simulated transactional data** from Kaggle, ingested into **Azure Blob Storage** as the raw data lake.

Using **Azure Databricks** and the **Medallion Architecture**, the pipeline processes data through:

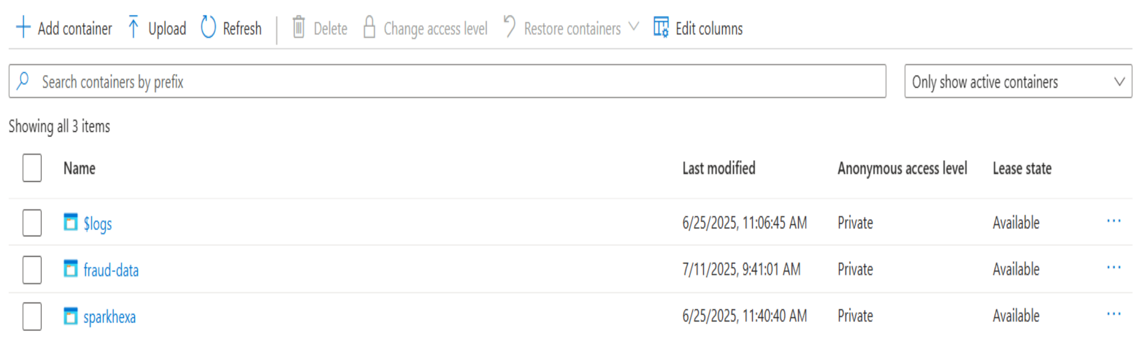
- **Bronze layer**: stores raw data
- **Silver layer**: cleans and transforms it
- **Gold layer**: enriches data, applies ML models, and flags fraud

Apache Airflow orchestrates the workflow, scheduling Databricks notebooks and retraining models. **Azure Logic Apps** monitor the Gold layer to trigger **real-time alerts** for fraudulent transactions.

Power BI connects to the Gold layer for interactive dashboards, while **Azure DevOps** handles **CI/CD**, automating infrastructure, code, and model deployments.



Creating storage account and adding a container:



Name	Last modified	Anonymous access level	Lease state
$logs	6/25/2025, 11:06:45 AM	Private	Available
fraud-data	7/11/2025, 9:41:01 AM	Private	Available
sparkhexa	6/25/2025, 11:40:40 AM	Private	Available

Kaggle:

Kaggle is a popular online platform for data science and machine learning, offering a collaborative space to access datasets, build models, and participate in competitions. It is widely used in academia and industry for learning and benchmarking.

A key feature is its extensive **dataset repository**, contributed by the community, covering diverse domains like finance, healthcare, and e-commerce—making it a valuable resource for data exploration and ML development.

Credit Card Fraud Detection Dataset:

This project uses the **"Transaction Fraud Detection"** dataset by **Ben Roshan** from Kaggle, containing **284,807 transactions**, with only **492 labeled as fraudulent**—highlighting a strong **class imbalance** (fraud = ~0.172%).

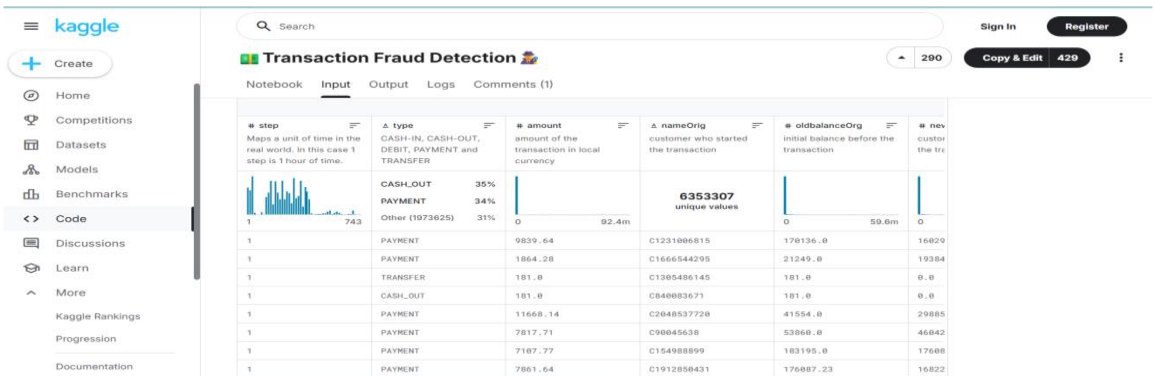
Key Features:

V1 to V28: PCA-transformed anonymized numerical features

Time: Elapsed time since the first transaction

Amount: Transaction value

Class: Target variable (0 = legitimate, 1 = fraud)



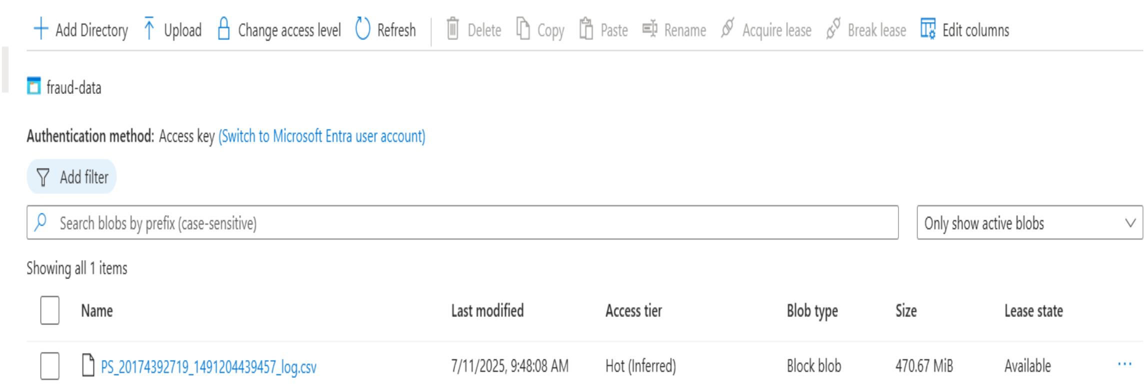
Dataset Characteristics:

Imbalanced classes pose modeling challenges

Anonymized features preserve privacy while enabling effective fraud detection modeling

For this project, the Credit Card Fraud Detection dataset from Kaggle has been selected "Transaction Fraud Detection" by Ben Roshan is a widely recognized dataset. It comprises 284,807 transactions, of which 492 are fraudulent, highlighting a significant class imbalance with frauds constituting approximately 0.172% of the data.

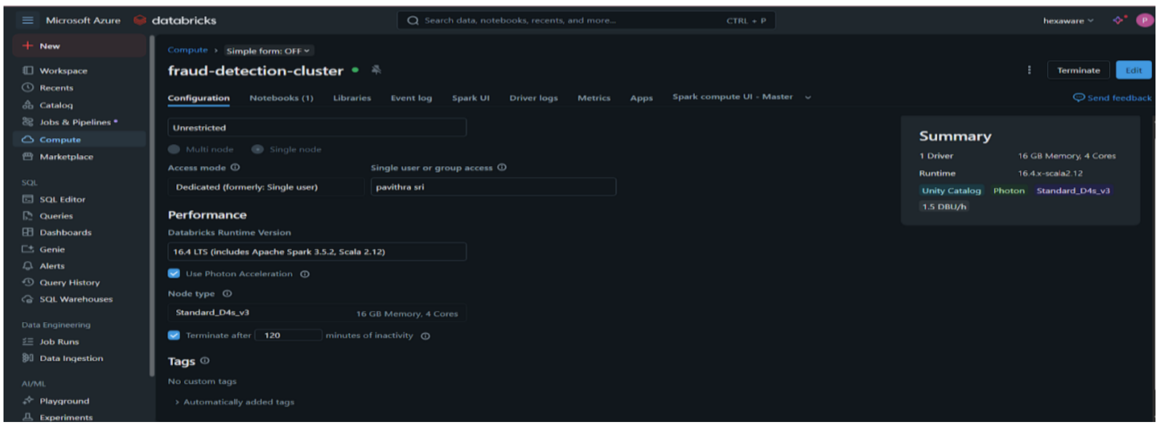
Uploading the dataset:



Initial dataframe:

```
df_raw: pyspark.sql.dataframe.DataFrame
    step: string
    type: string
    amount: string
    nameOrig: string
    oldbalanceOrg: string
    newbalanceOrig: string
    nameDest: string
    oldbalanceDest: string
    newbalanceDest: string
    isFraud: string
    isFlaggedFraud: string
    _rescued_data: string
<pyspark.sql.streaming.query.StreamingQuery at 0x7f0de0db7890>
```

Creating a cluster and connecting:



Data Ingestion: Bronze Layer

Bronze Layer (Raw):
Ingests raw, real-time transactional data from Event Hubs. Data is stored in its original form for auditing and reprocessing.

The initial stage of our fraud detection system involves ingesting raw financial transaction data into the Bronze layer. This process begins with obtaining a suitable dataset from Kaggle, which serves as a simulated source of transaction data. The selected dataset, typically in CSV format, is downloaded and subsequently uploaded to Azure Blob Storage.

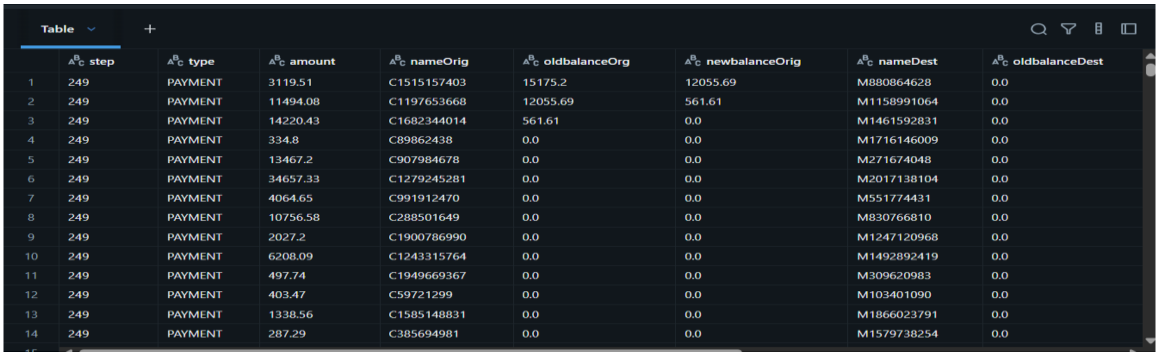
Reading and printing initial bronze table:

```
df_bronze: pyspark.sql.dataframe.DataFrame = [step: integer, type: string ... 9 more fields]
 |-- type: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: double (nullable = true)
 |-- newbalanceOrig: double (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: double (nullable = true)
 |-- newbalanceDest: double (nullable = true)
 |-- isFraud: integer (nullable = true)
 |-- isFlaggedFraud: integer (nullable = true)

+----+--------+--------+-----------+-------------+--------------+-----------+--------------+--------------+-------+--------------+
|step|    type|  amount|   nameOrig|oldbalanceOrg|newbalanceOrig|   nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFraud|
+----+--------+--------+-----------+-------------+--------------+-----------+--------------+--------------+-------+--------------+
|   1| PAYMENT| 9839.64|C1231006815|     170136.0|     160296.36|M1979787155|           0.0|           0.0|      0|             0|
|   1| PAYMENT| 1864.28|C1666544295|      21249.0|      19384.72|M2044282225|           0.0|           0.0|      0|             0|
|   1|TRANSFER|   181.0|C1305486145|        181.0|           0.0| C553264065|           0.0|           0.0|      1|             0|
|   1|CASH_OUT|   181.0| C840083671|        181.0|           0.0|  C38997010|       21182.0|           0.0|      1|             0|
|   1| PAYMENT|11668.14|C2048537720|      41554.0|      29885.86|M1230701703|           0.0|           0.0|      0|             0|
+----+--------+--------+-----------+-------------+--------------+-----------+--------------+--------------+-------+--------------+
only showing top 5 rows
```

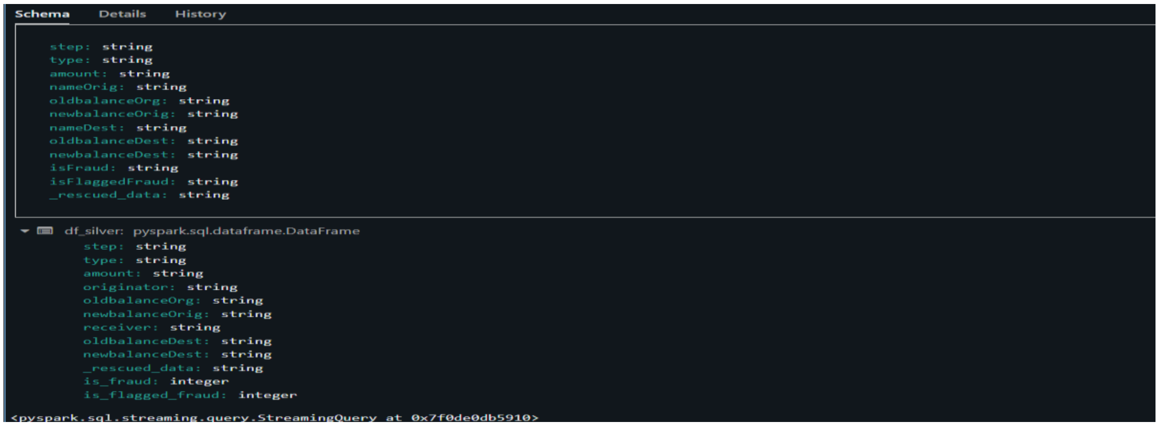
Azure Blob Storage acts as our data lake, storing the raw, untransformed data. This

Displaying bronze table:



	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
1	249	PAYMENT	3119.51	C1515157403	15175.2	12055.69	M880864628	0.0
2	249	PAYMENT	11494.08	C1197653668	12055.69	561.61	M1158991064	0.0
3	249	PAYMENT	14220.43	C1682344014	561.61	0.0	M1461592831	0.0
4	249	PAYMENT	334.8	C89862438	0.0	0.0	M1716146009	0.0
5	249	PAYMENT	13467.2	C907984678	0.0	0.0	M271674048	0.0
6	249	PAYMENT	34657.33	C1279245281	0.0	0.0	M2017138104	0.0
7	249	PAYMENT	4064.65	C991912470	0.0	0.0	M551774431	0.0
8	249	PAYMENT	10756.58	C288501649	0.0	0.0	M830766810	0.0
9	249	PAYMENT	2027.2	C1900786990	0.0	0.0	M1247120968	0.0
10	249	PAYMENT	6208.09	C1243315764	0.0	0.0	M1492892419	0.0
11	249	PAYMENT	497.74	C1949669367	0.0	0.0	M309620983	0.0
12	249	PAYMENT	403.47	C59721299	0.0	0.0	M103401090	0.0
13	249	PAYMENT	1338.56	C1585148831	0.0	0.0	M1866023791	0.0
14	249	PAYMENT	287.29	C385694981	0.0	0.0	M1579738254	0.0

Clean and transform the data and saving to the delta table:



```
Schema   Details   History

step: string
type: string
amount: string
nameOrig: string
oldbalanceOrg: string
newbalanceOrig: string
nameDest: string
oldbalanceDest: string
newbalanceDest: string
isFraud: string
isFlaggedFraud: string
_rescued_data: string

df_silver: pyspark.sql.dataframe.DataFrame
    step: string
    type: string
    amount: string
    originator: string
    oldbalanceOrg: string
    newbalanceOrig: string
    receiver: string
    oldbalanceDest: string
    newbalanceDest: string
    _rescued_data: string
    is_fraud: integer
    is_flagged_fraud: integer
<pyspark.sql.streaming.query.StreamingQuery at 0x7f0de0db5910>
```

Data Transformation: Silver Layer

The **Silver layer** in the Medallion Architecture refines raw data from the Bronze layer by **cleaning, standardizing, and transforming** it into a consistent, queryable format for analytics and ML.

Key activities include:

Handling **missing values** (e.g., dropping or imputing)

Correcting **data types** (e.g., timestamps, numerics)

Performing **basic feature engineering** (e.g., extracting hour from timestamps, categorizing transaction amounts)

The result is a clean, structured **Delta table** optimized for use in the Gold layer.

Displaying silver transformed table:

	amount	originator	oldbalanceOrg	newbalanceOrig	receiver	oldbalanceDest	newbalanceDest
1	3119.51	C1515157403	15175.2	12055.69	M880864628	0.0	0.0
2	11494.08	C1197653668	12055.69	561.61	M1158991064	0.0	0.0
3	14220.43	C1682344014	561.61	0.0	M1461592831	0.0	0.0
4	334.8	C89862438	0.0	0.0	M1716146009	0.0	0.0
5	13467.2	C907984678	0.0	0.0	M271674048	0.0	0.0
6	34657.33	C1279245281	0.0	0.0	M2017138104	0.0	0.0
7	4064.65	C991912470	0.0	0.0	M551774431	0.0	0.0
8	10756.58	C288501649	0.0	0.0	M830766810	0.0	0.0
9	2027.2	C1900786990	0.0	0.0	M1247120968	0.0	0.0
10	6208.09	C1243315764	0.0	0.0	M1492892419	0.0	0.0
11	497.74	C1949669367	0.0	0.0	M309620983	0.0	0.0
12	403.47	C59721299	0.0	0.0	M103401090	0.0	0.0
13	1338.56	C1585148831	0.0	0.0	M1866023791	0.0	0.0
14	287.29	C385694981	0.0	0.0	M1579738254	0.0	0.0

Feature Engineering & ML Preparation: Gold Layer

The **Gold layer** is the final data prep stage, enriching data from the Silver layer with **advanced features** for **ML models** and **analytics**. It is optimized for dashboards, reports, and model input.

Key activities include:

Adding derived features like is_suspicious (based on business rules) and fraud_risk_score (from preliminary ML models).

Performing **aggregations**, such as:

Average transaction amount per user

Number of transactions in a time window

Ratio of international to domestic transactions

It stores this enhanced, analytics-ready data in **Delta tables**, ensuring usability for fraud detection and decision-making.

Feature engineering example aggregations and saving it to the gold delta table:

```
df_gold: pyspark.sql.dataframe.DataFrame
    originator: string
    total_transactions: long
    total_amount_sent: double
    average_amount_sent: double
    fraud_count: long
df_silver: pyspark.sql.dataframe.DataFrame
Schema  Details  History

    step: string
    type: string
    amount: string
    originator: string
    oldbalanceOrg: string
    newbalanceOrig: string
    receiver: string
    oldbalanceDest: string
    newbalanceDest: string
    _rescued_data: string
    is_fraud: integer
    is_flagged_fraud: integer
```

Gold table sample, schema, and count of records:

	originator	total_transactions	total_amount_sent	average_amount_sent	fraud_count
1	C876714021	1	443010.87	443010.87	0
2	C224938823	1	2959.54	2959.54	0
3	C1676650606	1	34739.91	34739.91	0
4	C289153199	1	371.23	371.23	0
5	C1181048539	1	17411.44	17411.44	0

5 rows

```
root
 |-- originator: string (nullable = true)
 |-- total_transactions: long (nullable = true)
 |-- total_amount_sent: double (nullable = true)
 |-- average_amount_sent: double (nullable = true)
 |-- fraud_count: long (nullable = true)

Gold Table Record Count: 6347542
```

Checking Missing Values in Gold Table:

	originator	total_transactions	total_amount_sent	average_amount_sent	fraud_count
1	0	0	0	0	0

Displaying gold table:

	originator	total_transactions	total_amount_sent	average_amount_sent	fraud_count
1	C876714021	1	443010.87	443010.87	0
2	C224938823	1	2959.54	2959.54	0
3	C1676650606	1	34739.91	34739.91	0
4	C289153199	1	371.23	371.23	0
5	C1181048539	1	17411.44	17411.44	0
6	C1503438438	1	196626.77	196626.77	0
7	C765877097	1	63191.19	63191.19	0
8	C477278966	1	287868.3	287868.3	0
9	C326407304	1	250341.94	250341.94	0
10	C1285801989	1	90349.08	90349.08	0
11	C1424608077	1	13656.17	13656.17	0
12	C714358180	1	401340.14	401340.14	0
13	C909652139	1	29606.84	29606.84	0
14	C2124981192	1	313025.88	313025.88	0
15	C17536176	1	203336.28	203336.28	0

Convert to Pandas for sklearn:

```
df_pd: pandas.core.frame.DataFrame
    total_transactions: int64
    total_amount_sent: float64
    average_amount_sent: float64
    fraud_count: int64
```

Machine Learning Model Development & Deployment:

At the core of our fraud detection system is an **Isolation Forest** machine learning model, chosen for its efficiency with high-dimensional data and its ability to detect outliers without needing labeled fraud data. It works by isolating anomalies using fewer partitions compared to normal transactions.

Train model, 1 = normal, -1 = anomaly -> Convert to 0/1, Show sample

	total_transactions	total_amount_sent	average_amount_sent	fraud_count	anomaly_flag
0	1	5014.17	5014.17	0	0
1	1	15188.56	15188.56	0	0
2	1	9349.98	9349.98	0	0
3	1	35423.27	35423.27	0	0
4	1	71906.86	71906.86	0	0
5	1	80557.71	80557.71	0	0
6	1	7762.46	7762.46	0	0
7	1	8759.81	8759.81	0	0
8	1	405.30	405.30	0	0
9	1	8600.77	8600.77	0	0

The model is trained using **preprocessed data from the Silver layer**, following a feature selection process to identify the most relevant variables. The dataset is split into training and testing sets, and model parameters like the number of estimators and contamination rate are tuned to maximize precision and recall.

Save final output with anomaly flag to Delta Lake:

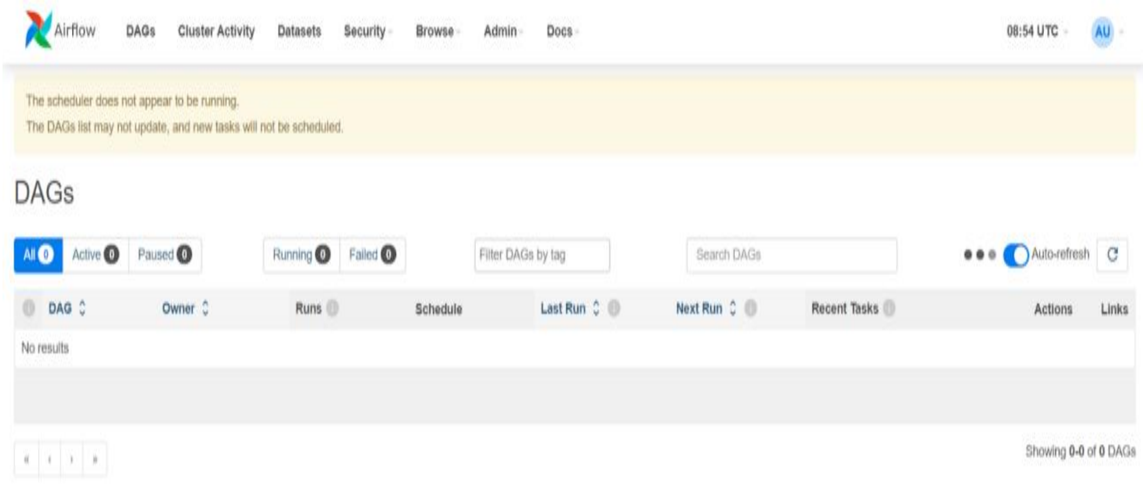
```
anomaly_sdf: pyspark.sql.dataframe.DataFrame = [total_transactions: long, total_amount_sent: double ... 3 more fields]

Anomaly detection results saved to Delta successfully.
```

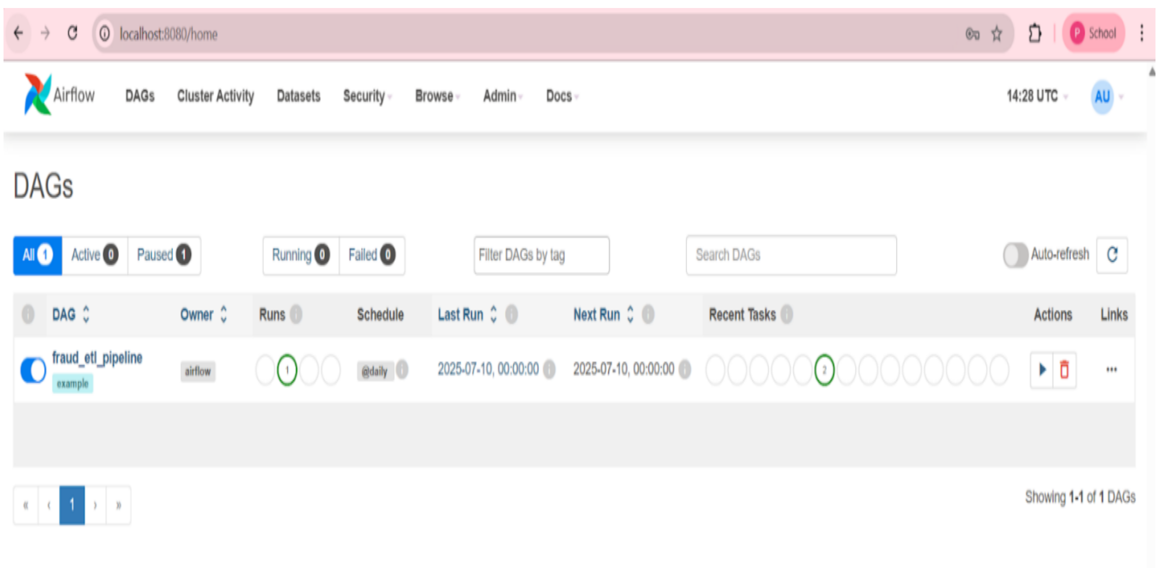
Future improvements include **automated retraining using Apache Airflow**, triggered periodically to keep the model updated with evolving transaction patterns. Additionally, **continuous performance monitoring** will be implemented to detect model drift and initiate retraining when necessary.

Workflow Automation with Apache Airflow:

Apache Airflow is essential for automating and orchestrating the financial fraud detection pipeline. It uses **DAGs (Directed Acyclic Graphs)** to define task sequences like data ingestion (Bronze), transformation (Silver), feature generation (Gold), ML model execution, and fraud alerting—ensuring correct task order and dependencies.



We use **DatabricksSubmitRunOperator** to trigger specific Databricks notebooks for each pipeline stage, enabling seamless integration between Airflow and databricks.

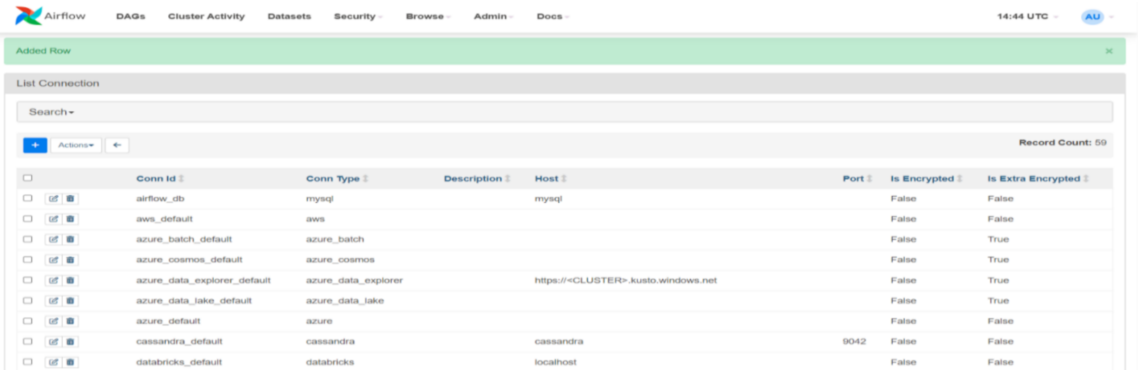


Key benefits of Airflow include:

Reliable execution with automatic retries for failed tasks.

Scheduled runs at predefined intervals.

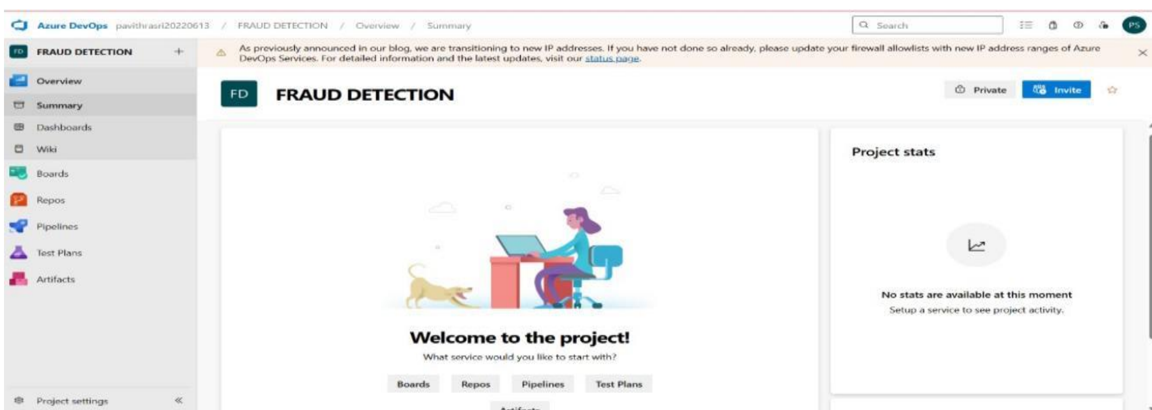
Monitoring tools for tracking and managing pipeline health.

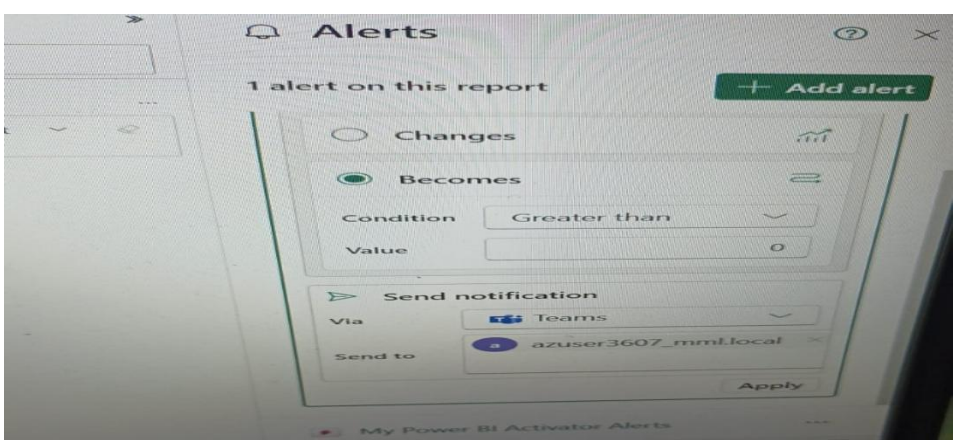


Real-time Alerting with Azure Logic Apps:

Azure Logic Apps act as the real-time alert system for fraud detection. They are triggered either by an **HTTP POST from Databricks** after detecting fraud or by **polling the Gold Delta Table** for new is_fraud = True records.

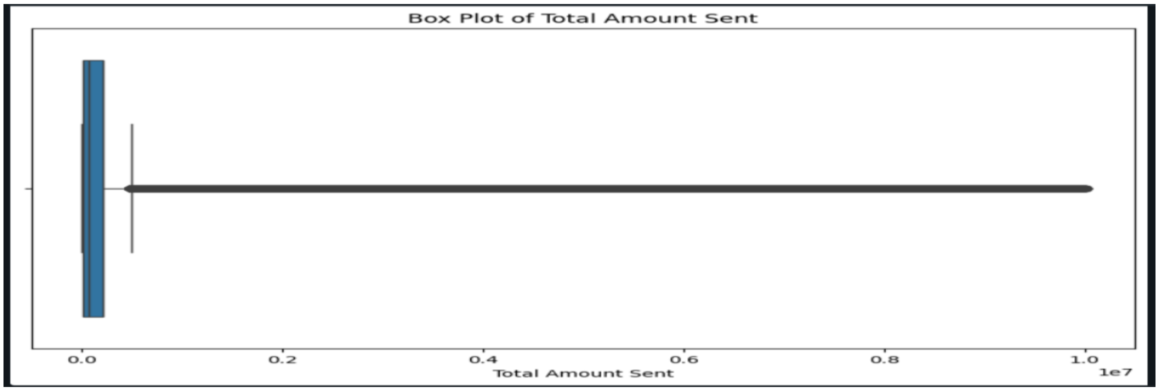
Once triggered, the Logic App runs a workflow that sends alerts via **email**, **Microsoft Teams**, or **webhooks**, ensuring prompt notifications to analysts or systems. This enables quick response to fraudulent activity, helping reduce potential losses.





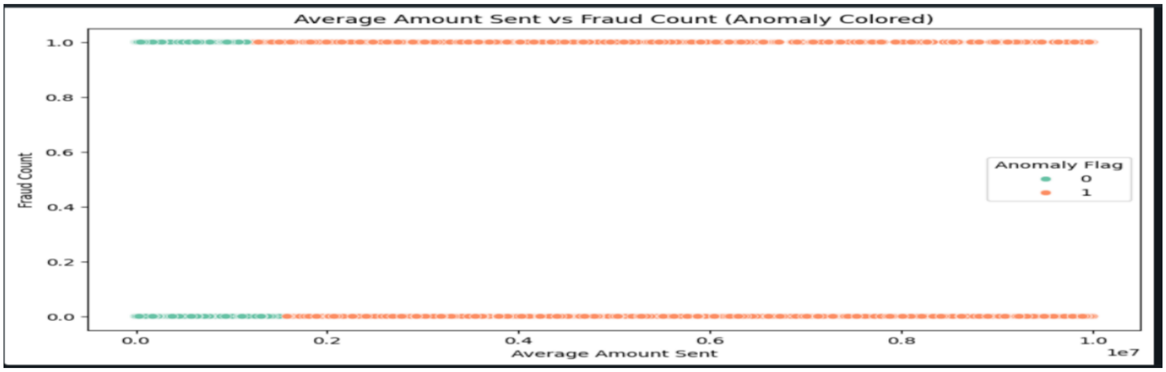
Data Visualization & Monitoring with Power BI:

A **box plot** (or **box-and-whisker plot**) is a statistical visualization used in Python to show the **distribution, central tendency, and variability** of a dataset. It highlights the **median**, **quartiles**, and **potential outliers**.

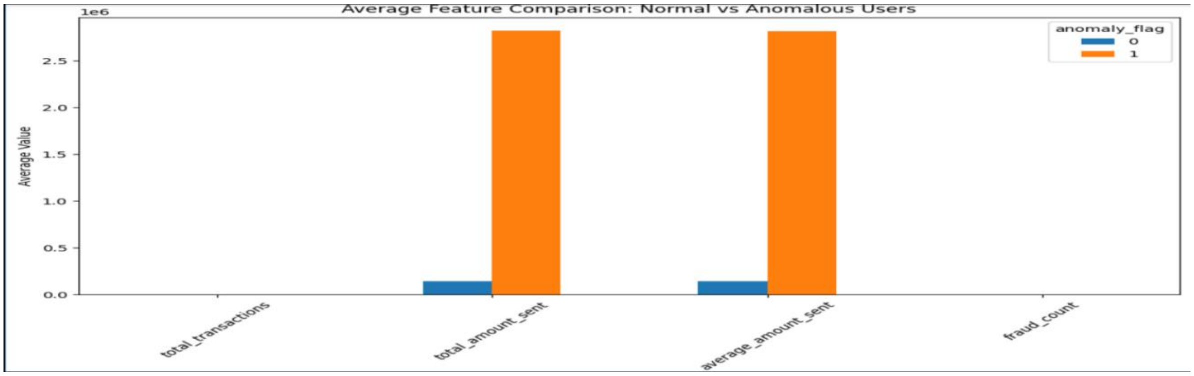


Key visualizations and reports include:

- Fraud trends over time, displayed through line charts to identify patterns and anomalies.



- Distribution of suspicious amounts using histograms to highlight common fraud values.



- Risk breakdowns by merchant, region, or transaction type via geographical maps and bar charts to pinpoint high-risk areas.
- Overall system performance metrics, such as processing time and model accuracy, shown with KPI cards.
- A Box Plot (also called a Box-and-Whisker plot) is a graphical representation of the distribution of a dataset.It helps you quickly identify outliers, spread, and central tendency of numerical data.

Power BI is used to visualize key fraud detection metrics and support real-time monitoring. It connects to the **Gold Delta Table** via **DirectQuery** for real-time insights or uses CSV exports for near-real-time analysis.

Key visualizations include:

Line charts to track fraud trends over time

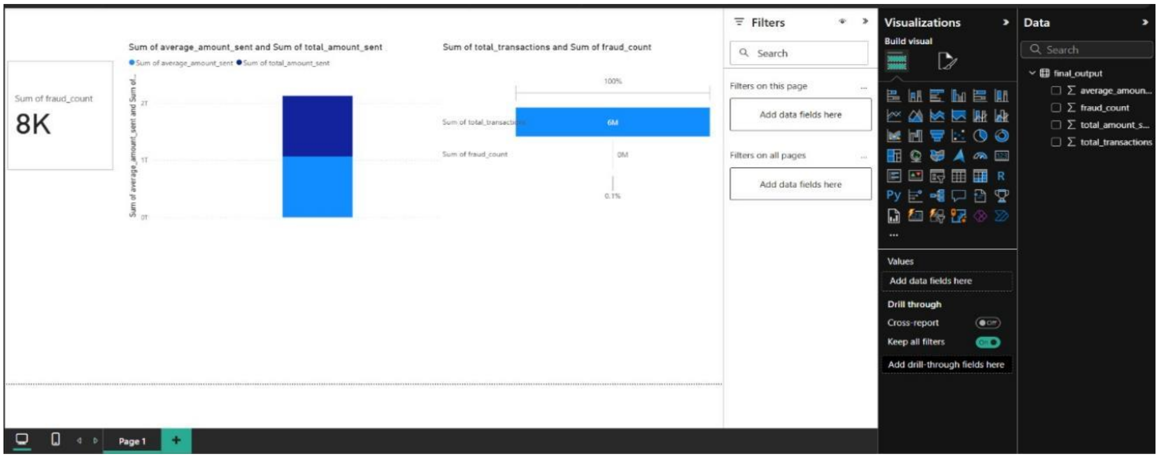
Histograms to show distribution of suspicious transaction amounts

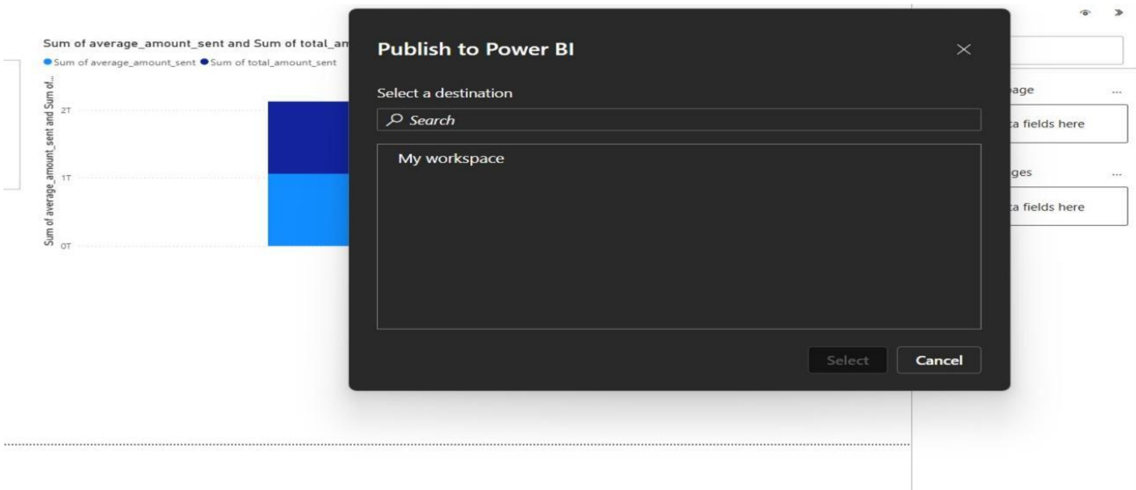
Maps and bar charts to analyze fraud by merchant, region, or transaction type

KPI cards for system performance metrics like processing time and model accuracy

Box plots to detect outliers and understand data distribution

These visuals help analysts identify and act on fraud patterns effectively.





STEP 1:

Create a Power BI Streaming Dataset

Go to Power BI Service

On the left pane, click on My workspace (or your workspace).

Click + Create > Streaming dataset

Select API as source, then click Next

Define your dataset fields, e.g.:

Turn on Historic data analysis → Click Create

Copy the Push URL. We’ll use this in Logic Apps.

STEP 2:

Create Azure Logic App to Forward Data

Go to Azure Portal → Search "Logic Apps"

Click Create → Choose Logic App (Consumption)

Fill in:

Resource group,Name,Region

Leave Log Analytics optional

After deployment, open your Logic App → Add a Trigger:

Search HTTP Request

Select: When an HTTP request is received

Paste the sample JSON body:

Next Step: Add Action → Search Power BI → Choose:

Add rows to a dataset

Sign in to your Power BI account

Select your workspace, dataset, table

Click Save. Logic App gives you a URL endpoint.

STEP 3:

Call Logic App from Databricks (Inside Notebook)

Use Python in your Databricks notebook to POST to the Logic App:

```
import requests

import json
```

This is your Logic App URL

logic_app_url = "<your_logic_app_trigger_url>"

Sample data row

Trigger the Logic App

response = requests.post(logic_app_url, json=data)

print(f"Status Code: {response.status_code}")

print(response.text)

You can do this inside your ML output block, wherever you detect anomaly_flag == 1.

STEP 4:

Create Power BI Dashboard

Go back to Power BI Service+ Create → Report

Use the Streaming Dataset you created

Add charts like:

Total frauds over time

Originators with most fraud

Real-time card showing latest anomalies

Key Technologies Utilized:

This project leverages a suite of Azure and open-source technologies for building a comprehensive fraud detection system. The tools are strategically selected to cover all aspects of the pipeline, from data storage to real-time alerting and visualization.

Purpose	**Tool**	**Role**
Storage	Azure Blob Storage	Data lake for raw transaction data.
Data Processing	Azure Databricks	ETL and ML model training.
Workflow Orchestration	Apache Airflow	Automated pipeline scheduling.
Machine Learning	Scikit-learn (Isolation Forest)	Anomaly detection model.
Alerting	Azure Logic Apps	Real-time fraud notifications.
Visualization	Power BI	Interactive fraud dashboards.
CI/CD	Azure DevOps	Automated deployments.
ML Tracking	MLflow	Model registry and experiment tracking.

Conclusion & Future Enhancements:

This project delivers a complete, real-time **financial fraud detection system** using **Medallion Architecture on Azure Databricks**, orchestrated by **Apache Airflow**, with **Azure Logic Apps** for alerting and **Power BI** for visualization.

Future improvements include:

Integrating **real-time streaming sources** like Azure Event Hubs or Kafka

Using **advanced ML models** (e.g., deep learning, ensembles)

Implementing **A/B testing** for continuous model evaluation

Expanding alerts via **SMS or security systems**

Enhancing reporting with **predictive risk scoring** for deeper fraud insights.