Real-Time Transaction Monitoring System

Using Databricks (PySpark), AWS S3, and PostgreSQL RDS

Overview

This project implements a real-time system to monitor banking transactions and detect behavioral patterns using PySpark (Databricks), AWS S3, and PostgreSQL RDS. It consists of two main components:
- Mechanism X: Simulates real-time data feed by uploading transaction chunks to S3 every second.
- Mechanism Y: Continuously monitors the S3 bucket for new transaction chunks, detects specific patterns, and stores detection results in S3 and a PostgreSQL database.

Architecture

Data Flow:

```
[GDrive] → (Mechanism X) → [S3: transaction-chunks/]
                    ↓
           (Mechanism Y: Streaming Pattern Detection)
                    ↓
          [S3: detections/] + [PostgreSQL: pattern_detections]
```

Directory Structure

```
.
├── chunk_test.csv               # Sample test data for triggering detections
├── CustomerImportance.csv        # Customer weight metadata
├── notebooks/notebook.ipynb
│   ├── mechanism_x                          # Script to simulate upload to S3
│   └── mechanism_y_detection                # Streaming pattern detection
└── S3 bucket: transaction-monitoring-bucket/
    ├── transaction-chunks/       # Uploaded transaction chunk files (10,000 rows each)
    ├── detections/              # Streaming output (50 detection rows/file)
```

```
└── checkpoints/            # Streaming query checkpoint folder
```

Pattern Definitions

PatId1 — UPGRADE

A customer is in the top 1 percentile of transaction counts for a merchant, and their Weight (from CustomerImportance.csv) is in the bottom 1 percentile.
Detection only applies when total transactions for the merchant exceed 50,000.

PatId2 — CHILD

A customer has an average transaction amount < ₹23 and has made ≥ 80 transactions with a merchant.

PatId3 — DEI-NEEDED

A merchant where the number of female customers < number of male customers, and number of female customers > 100.

Data Format

transactions.csv

Columns: step, customer, age, gender, zipcodeOri, merchant, zipMerchant, category, amount, fraud
- 10,000 rows per CSV chunk
- Uploaded to s3://transaction-monitoring-bucket/transaction-chunks/

CustomerImportance.csv

Columns: Source (customer), Target (merchant), Weight, typeTrans, fraud
- Preloaded to /dbfs/FileStore/data/CustomerImportance.csv

PostgreSQL Table

Table: pattern_detections
SQL:

```
CREATE TABLE pattern_detections (
   ystarttime TIMESTAMP,
   detectiontime TIMESTAMP,
   patternid TEXT,
   actiontype TEXT,
   customer TEXT,
   merchant TEXT
);
```

Setup Instructions

1. Upload to S3

- Upload chunk_test.csv to:
  s3://transaction-monitoring-bucket/transaction-chunks/chunk_test_001.csv
- Upload CustomerImportance.csv to:
  /dbfs/FileStore/data/CustomerImportance.csv

2. Databricks Configuration

Configure S3 access in Databricks using:

```
spark._jsc.hadoopConfiguration().set("fs.s3a.access.key", "YOUR_ACCESS_KEY")
spark._jsc.hadoopConfiguration().set("fs.s3a.secret.key", "YOUR_SECRET_KEY")
spark._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3.amazonaws.com")
```

Install libraries:

```
%pip install psycopg2-binary boto3
```

Execution

Mechanism X

Notebook: mechanism_x.ipynb - Uploads chunks to S3 (simulated with manual upload in testing).

Mechanism Y

Notebook: mechanism_y_detection.ipynb
- Reads chunks from S3.
- Applies patterns.
- Writes results to S3 (detections/) and PostgreSQL (pattern_detections).

Debugging Tips

- Ensure files appear in checkpoints/offsets/ with the latest timestamp.
- Check logs inside foreachBatch for exceptions.
- Confirm data in S3 matches schema.
- Verify detections using:

SELECT * FROM pattern_detections ORDER BY detectiontime DESC LIMIT 10;

IAM Permissions

Ensure IAM credentials have:
- s3:ListBucket
- s3:GetObject
- s3:PutObject

And public access is disabled only for buckets not programmatically accessed.

Public Access for Testing (0.0.0.0/0):

During initial setup or testing, I might have temporarily allow full public access to your S3 bucket by permitting all IP addresses (0.0.0.0/0) in the bucket policy or IAM role condition.

Warning :
This configuration is insecure and should never be used in production. It allows anyone on the internet to access or modify your data, leading to potential data breaches, unauthorized access, or compliance violations.

Alternates :

Use Pre-Signed URLs for External Access.
Restrict by IP Range or VPC