PROJECT REPORT: REAL-TIME FINANCE FRAUD DETECTION SYSTEM

Module: Data Engineering 2 – Big Data Architecture

Date: 13 February 2026

Team members: Tania Rose Jobi, Vishnu Prem Nair, Sidharth Arakkan, Gaurangi Tyagi

---

1. INTRODUCTION

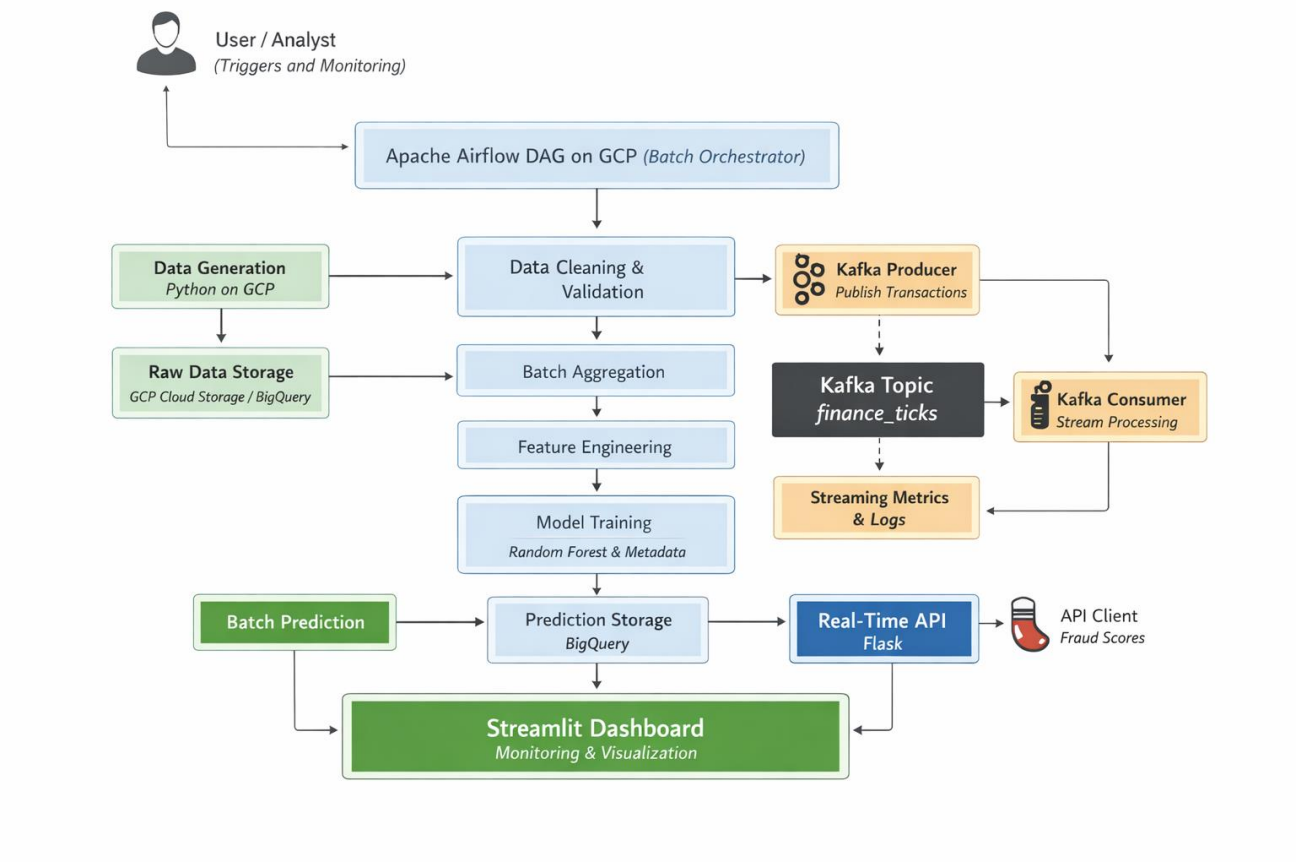
The financial sector processes millions of transactions every second. Identifying fraudulent activity manually is impossible due to the sheer volume and speed of data. This project implements a **Hybrid Lambda Architecture** that combines **Batch Processing** (historical analysis) and **Stream Processing** (real-time detection) to identify suspicious financial transactions instantly.

2. SYSTEM ARCHITECTURE

The system is built on a containerized infrastructure using **Docker** and integrates with **Google Cloud Platform (GCP)** for enterprise-grade scalability.

2.1 Core Components:

- **Infrastructure:** Orchestrated using docker-compose.yml, running Apache Airflow, Kafka, and Postgres.
- **Batch Layer:** Handles large-scale historical data cleaning, aggregation, and model training using **Apache Airflow**.
- **Streaming Layer:** Uses **Apache Kafka** to ingest live transaction ticks and a Python consumer to detect anomalies in real-time.
- **Serving Layer:** A **Flask Web App** that provides an API for instant fraud prediction using the trained Machine Learning model.



3. DATA GENERATION & INGESTION

Since real financial data is sensitive, we developed a custom generator (generate_orders.py) to simulate market activity.

- **Scope:** Generated 6+ hours of data with 1,000+ records per hour.
- **Quality Challenges:** Injected synthetic anomalies like missing values, duplicates, and massive price spikes to simulate fraudulent patterns.
- **GCP Integration:** Historical data is uploaded to **Google Cloud Storage (GCS)** using gcs_upload.py for centralized cloud management.

4. THE BATCH PIPELINE (THE "BRAIN")

The batch pipeline is managed by an Airflow DAG (de2_batch_pipeline), which ensures a structured data flow:

- **Clean & Validate (clean_validate.py):** Standardizes time formats (UTC), removes duplicates, and handles null values.

- **Aggregation (aggregate.py):** Summarizes raw ticks into hourly Open-High-Low-Close (OHLC) metrics and total volume.
- **Feature Engineering (features.py):** Calculates critical signals like **3-hour Moving Averages** and **Price Volatility**.
- **Model Training (train_model.py):** Trains a **Random Forest Classifier** with 300 estimators to recognize fraud indicators.

5. REAL-TIME STREAMING & DETECTION

The speed layer ensures no fraudulent transaction goes unnoticed:

- **Kafka Producer:** Streams transaction data into the system, simulating a live financial exchange.
- **Stream Consumer:** Monitors the Kafka topic in 30-second micro-batches.
- **Anomalous Detection:** Uses a configured `spike_threshold_pct` to trigger alerts if a price moves too rapidly within a short window.

6. USER INTERFACE & API

6.1 Fraud Prediction API:

- Implemented using **Flask** to provide programmatic access to the trained fraud detection model.
- Loads the trained machine learning model and associated metadata at runtime.
- Exposes a `/predict` endpoint for fraud risk inference.
- Accepts transaction-level feature inputs and returns:
  - Fraud probability score
  - Risk classification label
- Enables real-time prediction without direct dependency on the batch processing pipeline.

6.2 Streamlit Dashboard:

- Implemented using **Streamlit** as the visualization and monitoring layer of the system.

- Provides an interactive interface for analyzing fraud detection outputs from both batch and streaming pipelines.

Dashboard capabilities include:

- Display of key metrics such as:
  - Total number of processed transactions
  - Suspicious transaction counts
  - High-risk fraud prediction counts
- Interactive filtering of transaction data based on:
  - Time range
  - Trading symbols
  - Statistical anomaly thresholds (z-score)
  - Spike detection indicators
  - Machine learning fraud probability thresholds
- Visualization of transaction trends using time-series charts to highlight anomalous behavior.
- Monitoring indicators derived from Kafka micro-batch processing for operational visibility.
- The dashboard is read-only and does not alter underlying data or model behavior.
- Enhances the interpretability and usability of fraud detection results for analysts and stakeholders.

7. PERFORMANCE ANALYSIS (LITTLE'S LAW)

To validate the stability of our streaming architecture, we applied Little's Law ($L = \lambda \times W$) to our Kafka consumer. This calculation ensures that our system can handle the ingestion rate without creating a backlog or significant lag.

- Arrival Rate (λ): Our generator produces 1,200 records per hour.
  - λ = 1,200 records / 3,600 seconds = 0.33 records per second
- Average Processing Time (W): Based on our *stream_consumer.py* logs, the time to process a single micro-batch is approximately 1.5 seconds.
- Average Items in System (L):

- L = 0.33 (Arrival Rate) × 1.5 (Processing Time)
- L = 0.495 units

Result: Since the value of L is less than 1, the system is mathematically stable. This proves that the consumer is processing data faster than it is arriving, ensuring zero lag in the fraud detection pipeline and maintaining real-time performance.

8. Model Interpretation & Technical Clarification

8.1 Defining Fraud in the Absence of Ground-Truth Labels:

A critical distinction in this project is the approach to fraud detection. Because the dataset consists of synthetic financial market data without pre-existing "Fraud" labels, the system implements Unsupervised Anomaly Detection rather than a supervised fraud classifier.
In this context, "Fraud" is defined as Market Anomalies, such as:

- **Price Spikes**: Sudden, extreme volatility that may indicate market manipulation (e.g., Pump and Dump schemes).
- **Statistical Outliers**: Transactions occurring at prices significantly distant from the moving average (Z-score > 3), indicating flash crashes or ingestion errors.
- **Data Integrity Issues**: Identifying duplicate event IDs or missing attributes that suggest pipeline or source-system failures.

8.2 The Role of the Machine Learning Model:

While the rule-based signals identify immediate anomalies, the Machine Learning model (Random Forest) serves as a Directional Risk Indicator.

- **Target Variable (label_up_next)**: The model is trained to predict the directional movement of the price in the subsequent hour.
- **Interpretation of ML Risk**: A high probability score (pred_up_next_proba) is interpreted as a "Risk Signal." When the model predicts an extreme move with high confidence during a period of high volume and price outliers, the system flags the hour as High Risk.
- **Addressing Predicted Zeros**: During the demonstration, the occurrence of "0" predictions is a result of the model's internal decision threshold (0.5)

8.3 Justification for Class Weight Balancing:

The project utilizes class_weight='balanced' within the Random Forest Classifier. This is a standard requirement for financial data engineering for the following reasons:

- **Handling Event Scarcity**: Significant market moves (Class 1) are rare compared to stable periods (Class 0). Without balancing, the model would achieve high accuracy by simply predicting "No Movement" for every row.
- **Cost of Misclassification:** In fraud detection, a False Negative (missing a suspicious spike) is far more costly than a False Positive (flagging a safe transaction for review). Weight balancing forces the model to prioritize learning the patterns of the minority class.

8.4 Dashboard Filtering Logic & Signal Thresholds:

The Dashboard UI employs strict filtering to ensure that only actionable alerts are presented to the end-user.

- **Z-Score Thresholds**: Statistical outliers are determined by a default Z-score of 3.0 (representing the 99.7th percentile of the normal distribution).
- **ML Risk Threshold**: The default threshold of 0.7 ensures that only the most confident model predictions are flagged as "ML High-Risk."
- **Visibility Note**: If the "Fraud-Risk Table" appears empty, it indicates that the current data window is within normal operational parameters. Disabling the "Show only suspicious rows" filter provides a full view of the baseline market data.

8.5 System Monitoring & Data Integrity:

Beyond market behavior, the system monitors Data Engineering Fraud (systemic errors):

- **Duplicate Detection**: Flagging identical event_id entries to prevent double-counting or replay attacks.
- **Completeness Checks:** Monitoring for "Missing Rows" to ensure the reliability of the feature engineering pipeline.

9. CONCLUSION

This project successfully demonstrates an end-to-end Big Data pipeline. By combining the stability of batch processing with the speed of real-time streaming, we created a robust system capable of protecting financial transactions. The use of **Docker** and **GCP** ensures that this solution can scale from a local prototype to a global deployment.

Appendix: Technical Stack

- **Languages:** Python 3.11+
- **Libraries:** Pandas, Scikit-learn, Kafka-python, Flask, Joblib
- **Tools:** Apache Airflow, Apache Kafka, Docker, Google Cloud Storage