Real Time Fraud Detection And Monitoring Data Pipeline

Data Engineering Track

ITP "24-25-R3"

Project By:

Ali Younis
Ahmad Wahdan
Atef Mousa
Mohamed Eldeeb
Moataz Gamal
Kamel Moustafa

Abstract

This project, Credit Card Fraud Detection and Monitoring, delivers a fully integrated real-time and batch data processing pipeline designed to detect, predict, and monitor fraudulent credit card transactions with high accuracy and minimal latency. The solution uses Apache Kafka as the primary ingestion layer, streaming transaction records row-by-row to simulate real-time data flow. These records are consumed by PySpark Structured Streaming, where initial cleaning, transformation, and feature engineering take place.

Processed data flows in two main directions:

1. Real-time monitoring pipeline – data is sent to a PostgreSQL staging area, enabling Grafana dashboards to display live transaction activity and fraud alerts.
2. Data lake and machine learning pipeline – data is stored in Hadoop HDFS for long-term retention, where it is used to train and retrain a Machine Learning model that predicts potential frauds in new incoming transactions. The ML model is integrated into the streaming workflow to generate real-time fraud predictions.

For the batch analytics pipeline, Apache Airflow schedules and manages ETL jobs that move aggregated data from PostgreSQL into Snowflake, where it serves as a centralized data warehouse. This enables advanced historical analysis and reporting through Power BI dashboards. The entire system is containerized using Docker and deployed on an AWS EC2 instance running Ubuntu, ensuring ease of deployment, scalability, and environment consistency.

Acknowledgement

I would like to express my sincere gratitude for the guidance, resources, and persistence that were instrumental in the successful development of the Credit Card Fraud Detection and Monitoring System. This project was a journey through designing a hybrid data pipeline, implementing advanced machine learning capabilities, and fine-tuning real-time analytics — and it would not have been possible without the combined power of modern data engineering tools and consistent determination. First and foremost, I would like to thank the foundational principles of scalable and fault-tolerant data architecture, which guided the overall design of this project. The iterative process of requirement analysis, architectural planning, prototyping, and rigorous testing was essential to building a system capable of both real-time fraud detection and historical analysis. The process of exploring trade-offs, validating assumptions, and refining the design proved invaluable in achieving a robust end-to-end solution. I am especially grateful for the integrated ecosystem of open-source and enterprise-grade technologies that formed the backbone of this project. Apache Kafka provided the reliability and scalability for real-time transaction ingestion, while Apache Spark Structured Streaming powered low-latency feature engineering and fraud prediction. Hadoop HDFS offered a resilient storage layer for historical datasets, supporting model retraining and deep analytics. Apache Airflow orchestrated the ETL pipelines and automated model lifecycle tasks, ensuring consistency across batch and streaming processes. The data storage and analytics stack — from PostgreSQL for live monitoring, to Snowflake for centralized data warehousing — enabled both operational visibility and long-term trend analysis. The visualization layer, comprising Grafana for real-time dashboards and Power BI for historical insights, ensured that fraud detection was not just accurate, but also actionable for stakeholders. I would also like to acknowledge the role of Docker containerization and AWS EC2 deployment in making the system portable, reproducible, and cloud-ready. This infrastructure eliminated environment-specific barriers and allowed for rapid iteration and scaling. This journey was not without challenges — from debugging data ingestion bottlenecks, optimizing Spark job performance, and handling schema evolution, to fine-tuning machine learning models for production workloads. However, the logging frameworks, monitoring dashboards, and the discipline of structured

troubleshooting provided the feedback loops necessary to overcome each technical hurdle. Finally, I would like to recognize the combination of human persistence and technological capability that made this project possible. Building this system has not only delivered a scalable and effective fraud detection platform but has also deepened my expertise in distributed data processing, streaming analytics, and ML-driven decision systems. I am proud of this achievement and hope it will serve as a powerful and adaptable solution for combating financial fraud in today’s digital economy.

Thank you.

Introduction

In today’s fast-paced financial ecosystem, credit card fraud detection has become a mission-critical requirement. With the massive surge in digital transactions—driven by online retail, mobile payments, and global e-commerce—fraudulent activity has grown more frequent and increasingly sophisticated. Threats such as identity theft, account takeover, and unauthorized payment transactions can result in severe financial losses and long-term reputational harm for institutions. Traditional fraud detection methods, often based on static rule sets or manual reviews, are no longer sufficient to handle the sheer scale, speed, and complexity of modern fraud schemes. They lack the agility to adapt to evolving attack patterns and cannot provide the real-time responsiveness needed to prevent financial damage before it occurs. This project introduces a Credit Card Fraud Detection and Monitoring System powered by big data processing frameworks, distributed storage, and machine learning. The architecture is designed to handle high-velocity transaction streams using Apache Kafka for ingestion, Apache Spark Structured Streaming for real-time processing, Hadoop HDFS for historical data storage and ML training, and PostgreSQL for immediate monitoring. Fraud prediction models are integrated into the streaming pipeline to flag suspicious transactions instantly, while Grafana dashboards provide live visualizations and Power BI delivers in-depth historical insights from Snowflake. By combining streaming analytics, predictive modeling, and interactive monitoring, the system enables financial institutions to detect, investigate, and respond to fraud in near real time, reducing exposure to risk and improving customer trust.

Project Purpose & Business Goal

This document presents the architecture, implementation, and operational design of the Credit Card Fraud Detection and Monitoring Pipeline. The system's primary business objective is to detect, flag, and report potentially fraudulent financial transactions within seconds of occurrence, thereby minimizing financial exposure, safeguarding customer accounts, and maintaining institutional trust.

The solution is built to address two equally critical needs:

1. Real-Time Fraud Detection & Alerts
    - Leveraging Apache Kafka for high-throughput transaction ingestion and Apache Spark Structured Streaming for low-latency processing, the system analyzes each incoming transaction against a trained machine learning model to determine the probability of fraud.

    - Transactions flagged as suspicious are routed to a PostgreSQL staging database, where Grafana dashboards provide immediate visualization and alerts to security teams for rapid response.

2. Historical Analysis & Model Improvement
    - All processed transactions are archived in Hadoop HDFS for long-term storage, batch analytics, and retraining of the fraud detection model to adapt to evolving criminal patterns.

    - Apache Airflow orchestrates ETL processes to load aggregated data into Snowflake, enabling advanced historical trend analysis, compliance reporting, and business intelligence through Power BI dashboards.

By integrating real-time detection with comprehensive historical insights, this pipeline supports both instantaneous fraud mitigation and strategic decision-making. It ensures that suspicious activity is intercepted early while equipping analysts and decision-makers with the data needed to refine detection strategies, reduce false positives, and improve customer satisfaction.

Problem Statement

Fraudulent credit card transactions represent one of the most pressing threats to modern financial systems, resulting in billions of dollars in global losses every year. Beyond the immediate financial impact, such fraudulent activities undermine customer trust, brand reputation, and regulatory compliance standing for financial institutions and payment processors. Traditional fraud detection solutions are often retrospective in nature, relying heavily on batch-processed historical datasets. While this approach can reveal suspicious patterns after the fact, it leaves organizations blind to real-time threats, creating a critical time gap in which fraudulent transactions can proceed unchecked. This delay reduces the ability to intervene before funds are transferred, often making recovery impossible. In addition to timing issues, organizations face multiple operational and technical challenges in deploying effective fraud detection systems:

1. Lack of integration between real-time detection and historical analytics:
   Many systems are siloed, meaning real-time monitoring tools and historical analytical platforms operate independently. This separation makes it difficult to correlate live events with historical trends or to use fresh data immediately for model retraining.

2. Complexity in supporting machine learning workflows end-to-end:
   Financial institutions require pipelines that not only train predictive models using large historical datasets but also deploy these models for immediate scoring of live transactions. Achieving this in a scalable, automated fashion—while maintaining data integrity and low latency—is a significant engineering challenge.

3. Limited real-time observability for stakeholders:
   Key fraud-related KPIs, such as transaction velocity, geolocation anomalies, and high-risk merchant categories, are often buried in backend logs or delayed reports. Without live dashboards and proactive alerting, decision-makers cannot respond quickly enough to emerging fraud campaigns.

This project directly addresses these challenges by implementing a hybrid data architecture that combines:

1. Real-time fraud detection capabilities using streaming technologies to flag suspicious transactions instantly.
2. Batch-oriented historical analytics to uncover long-term fraud patterns, perform root cause analysis, and improve machine learning model accuracy over time.
3. Integrated predictive modeling pipelines that allow for seamless training, deployment, and continuous improvement of fraud detection models.

By unifying streaming analytics, historical data processing, and advanced machine learning into a single, scalable system, this solution ensures financial institutions can detect, act upon, and learn from fraudulent activity in both the short and long term.

Use Case

The Credit Card Fraud Detection and Monitoring pipeline is designed to serve the needs of financial institutions, payment processors, and cybersecurity teams that require a unified platform for real-time fraud prevention and historical fraud analytics. The system provides end-to-end capabilities for ingesting, processing, analyzing, and visualizing both streaming and batch transaction data.

Key functional components and their benefits include:

- Real-time data ingestion and processing via Apache Kafka:
  The pipeline continuously captures incoming credit card transaction events as they occur, processing them row by row in near real-time. This ensures that no transaction is delayed and that fraud detection logic can be applied almost instantly. The use of Kafka as a distributed messaging system provides high throughput, fault tolerance, and scalability, allowing the system to handle millions of transactions per second if required.

- Machine learning model integration with historical data stored in Hadoop HDFS:
  Historical transaction records are stored in HDFS, forming the foundation for machine learning model training. Using large-scale datasets enables the model to identify subtle fraud indicators—such as unusual spending patterns, atypical merchant behavior, or deviations from a customer’s normal geographic transaction profile—that might otherwise be missed by rule-based systems. Once trained, the fraud detection model is deployed to the streaming environment for real-time scoring of transactions.

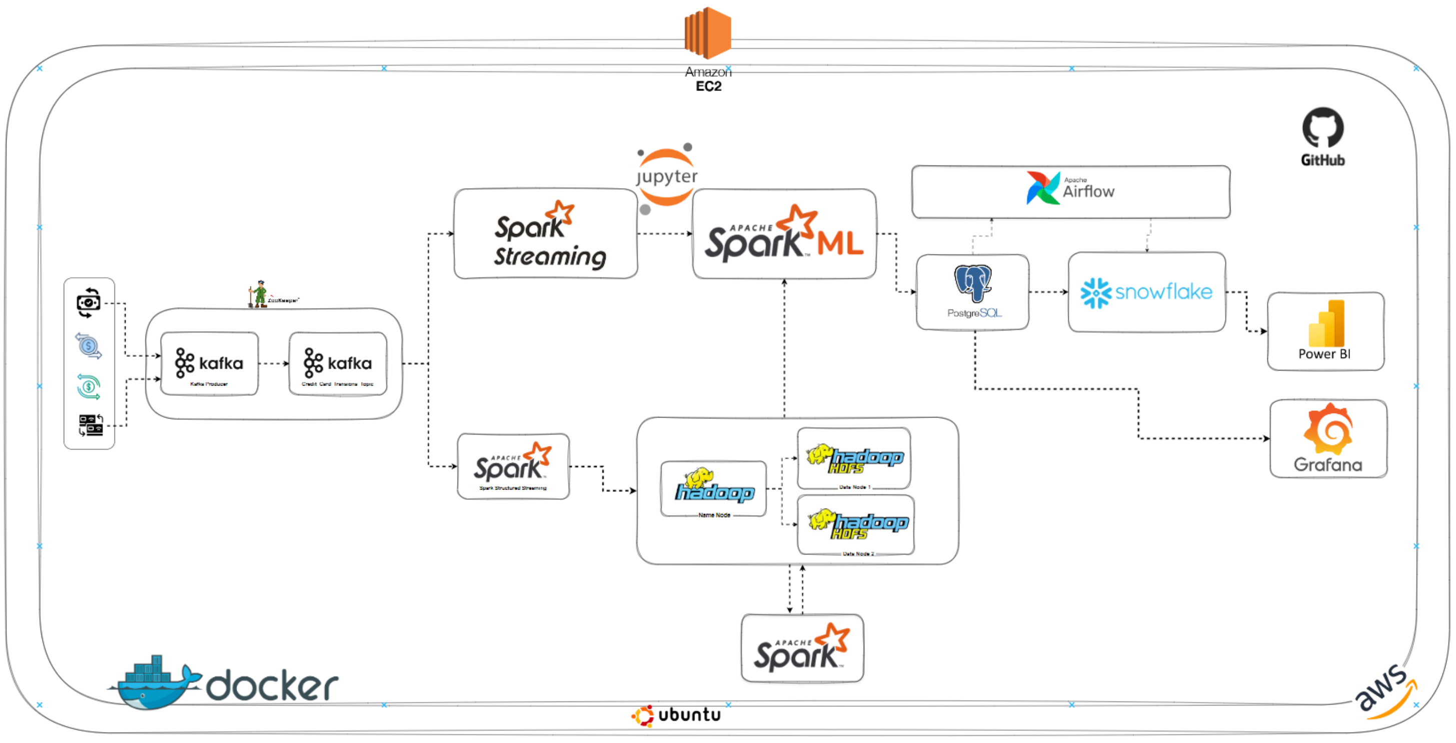
- Instant alerting and operational monitoring through Grafana and PostgreSQL:
  Processed transaction data is fed into a PostgreSQL staging database, which serves as the data source for Grafana dashboards. This allows fraud analysts and security teams to monitor key metrics—such as transaction volumes, flagged transaction counts, and fraud probability scores—in real time. Configurable alert rules can trigger notifications to relevant teams within seconds of suspicious activity detection, enabling rapid incident response.

- Batch data processing and historical analytics using Airflow and Snowflake:
  While real-time monitoring is essential for immediate action, long-term fraud pattern analysis is equally important. Apache Airflow orchestrates batch ETL pipelines to load processed transaction data into Snowflake, a cloud-based data warehouse optimized for analytics. This enables Power BI dashboards to present detailed historical insights, regulatory compliance reports, and trend analyses, which are vital for strategic fraud prevention planning and audit purposes.

- Portability, scalability, and deployment flexibility through Docker and AWS EC2:
  The entire solution is containerized using Docker, ensuring consistent execution environments across development, testing, and production stages. Deployment on AWS EC2 instances ensures horizontal scalability, allowing the infrastructure to expand seamlessly as transaction volumes grow. This also facilitates cost optimization by scaling resources based on actual workload demands.

Architecture and Tech Stack

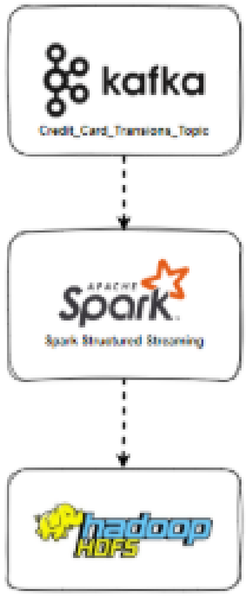
The architecture of the Credit Card Fraud Detection and Monitoring system is engineered to manage both real-time and batch data processing workflows, while also enabling machine learning model training, deployment, and continuous improvement. The design philosophy centers around scalability, portability, and low-latency performance, ensuring that fraudulent transactions can be detected within seconds while still supporting deep historical analytics for long-term fraud prevention strategies. All components run on a single AWS EC2 instance (Ubuntu OS) for cost efficiency and centralized management, with Docker containerization ensuring that each service is isolated, reproducible, and easy to maintain. This approach provides the flexibility to scale individual components in the future without major architectural changes. The following diagram visually depicts the end-to-end data flow across real-time, batch, and analytics layers.



Main Data Flow

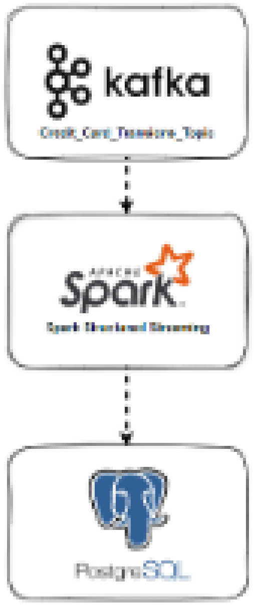
1. Data Ingestion (Kafka Producer)

- Function: Simulates real-time credit card transaction events by reading from CSV datasets.
- Operation: Sends each transaction row-by-row into a dedicated Kafka topic for downstream consumption.
- Coordination: Apache Zookeeper manages broker metadata, partition assignments, and cluster health for Kafka.
- Impact: Ensures low-latency, high-throughput ingestion, allowing the pipeline to process high transaction volumes without bottlenecks.



2. Real-time Data Processing (PySpark Structured Streaming)

- Function: Consumes live transaction data from Kafka and applies data cleaning, transformation, and feature engineering in memory.
- Dual Output Paths:
  - Path 1 – Real-time Monitoring: Sends cleaned and enriched transactions directly to a PostgreSQL staging database for immediate visualization in Grafana dashboards.
  - Path 2 – Data Lake Storage: Stores enriched transactions in Hadoop HDFS, where they are preserved for future ML model training and large-scale batch aggregations.



- Advantage: This dual-path approach allows instant alerting while also building a rich historical dataset for continuous model improvement.

3. Machine Learning Model Integration

- Training: Historical transaction data in HDFS serves as the training source for a fraud detection model. This model learns from features such as transaction amount, merchant ID, geolocation, time patterns, and account history.
- Deployment: Once trained, the ML model is integrated directly into the PySpark streaming job, enabling on-the-fly fraud predictions for each new transaction.
- Output: Predicted fraud labels (e.g., "fraud" or "legitimate") are appended to the transaction record and stored in both PostgreSQL and HDFS for monitoring and future retraining.

4. Batch Processing and Historical Analytics

- Orchestration: Apache Airflow schedules ETL workflows to extract processed transaction data from PostgreSQL and load it into Snowflake, a high-performance cloud data warehouse.
- Purpose:
  - Facilitate long-term trend analysis.
  - Generate compliance and audit reports.
  - Identify slow-evolving fraud patterns that may not be visible in short time windows.
  - Visualization: Power BI connects to Snowflake to provide dashboards for executives, compliance officers, and fraud analysts.

5. Visualization and Monitoring

- Real-time Dashboards (Grafana): Connected to PostgreSQL, Grafana displays live metrics such as transaction counts, flagged fraud events, and fraud probability distributions. Alerts can be configured to notify relevant teams within seconds of anomalies.
- Historical Analytics (Power BI): Using Snowflake as a backend, Power BI offers drill-down reports and trend visualizations spanning weeks, months, or years.

6. Deployment and Orchestration

- Containerization: Every service—Kafka, Spark, Hadoop, PostgreSQL, Airflow, Grafana, Jupyter Notebook, and the ML model—is packaged as a Docker container and managed via Docker Compose.
- Hosting: The entire stack runs on a single AWS EC2 Ubuntu server for centralized cost control.
- Version Control: GitHub stores all source code, configuration files, and Jupyter notebooks, enabling collaboration and CI/CD workflows.

3.2 Tools and Technologies Used

Tool / Technology	Purpose
Apache Kafka	Real-time ingestion of credit card transactions from CSV files row-by-row.
Zookeeper	Manages Kafka cluster coordination, leader election, and metadata.
Apache Spark (Structured Streaming)	Performs real-time data processing, feature engineering, and fraud prediction integration.
Hadoop HDFS	Acts as a data lake for storing raw and enriched transaction data; supports ML model training and batch aggregations.
PostgreSQL	Serves as a staging area for real-time processed data; powers Grafana dashboards.
Snowflake	Cloud-based data warehouse for historical analytics and BI integration.
Apache Airflow	Orchestrates batch ETL pipelines from PostgreSQL to Snowflake.
Grafana	Provides real-time dashboards for live fraud monitoring and KPI tracking.

Power BI	Enables advanced historical fraud analytics and reporting.
Docker & Docker Compose	Containerizes and orchestrates all services for consistent deployment.
AWS EC2 (Ubuntu)	Hosts the complete pipeline infrastructure in the cloud.
GitHub	Version control for code, configurations, and ML notebooks.
Jupyter Notebook	Used for model development, testing, and integration workflows.

Pipeline Components and Data Flow

The Credit Card Fraud Detection and Monitoring system is powered by a hybrid data pipeline that seamlessly integrates real-time data streaming, machine learning–based scoring, and batch analytics into a unified workflow. This design ensures that fraudulent transactions are detected and acted upon within seconds, while also maintaining a rich historical data repository for deep analysis, retraining, and compliance reporting. The pipeline follows a modular, layered approach where each component is responsible for a distinct function, yet all work together through clearly defined interfaces. Below is a detailed walkthrough of each stage in the pipeline, from ingestion to visualization.

Data Ingestion Layer – Apache Kafka

- The first stage of the pipeline focuses on capturing incoming transaction data in real time.
    - Input Source: Simulated transaction data derived from CSV files, containing essential fields such as:
    - Transaction ID
    - Timestamp
    - Amount
    - Merchant ID
    - Cardholder details (hashed or anonymized for security)
    - Transaction location (latitude/longitude or region code)
- Kafka Producer: Reads the CSV files row-by-row and streams each record into a dedicated Kafka topic. This streaming approach mirrors real-world transaction arrival patterns, allowing the system to handle data in a continuous event-driven manner.

- Kafka Cluster & Zookeeper Coordination:
    - Kafka Brokers store and forward messages to downstream consumers.
    - Zookeeper coordinates broker metadata, manages leader elections, and ensures cluster stability.
    - Purpose in the Pipeline: Kafka acts as a durable, high-throughput messaging layer that decouples producers (data sources) from consumers (processing engines). This ensures flexibility—data can be consumed by multiple services without impacting ingestion performance.

Real-time Processing Layer – Apache Spark Structured Streaming

Once transaction events enter Kafka, they are first persisted in the Data Lake raw zone (HDFS) in their original Avro/JSON form for auditability and reprocessing. This ensures no data is lost and that an immutable raw history exists before any transformation. A PySpark Structured Streaming job then reads from this raw HDFS landing zone in micro-batch mode (trigger interval configured in the job) and performs the following steps:

Core Processing Tasks

- Schema Enforcement & Parsing
    - Reads raw Kafka messages from HDFS and enforces a well-defined schema (transactions schema).
    - Parses nested structures into flat DataFrame columns for easier processing.
- Data Cleaning
    - Handles missing values (imputation or null dropping where appropriate).
    - Standardizes categorical values (e.g., gender normalization, merchant category naming).
    - Ensures consistent date/time parsing and numeric formatting.

- Calculates derived metrics for fraud detection, such as:
    - Transaction velocity: count of card transactions in a sliding time window.
    - Amount deviation: deviation from the customer's historical average.
    - Merchant geolocation anomalies: great-circle distance between customer and merchant.
    - Customer age: computed from DOB.
    - Distance_km: haversine distance between transaction and merchant location.
- Record Enrichment
    - Adds static metadata from reference datasets (merchant info, customer profiles).
    - Computes risk scores (merchant and geolocation).
    - Tags each record with ingestion and processing timestamps for traceability.

Dual Output Paths from Spark:

After transformation, Spark writes the processed/enriched dataset to two destinations, serving both operational and analytical needs:

- Path A – Staging for Batch DWH Loads
    - Writes daily processed data into the PostgreSQL staging table (transactions).
    - This staging area is later used by Airflow to load data into Snowflake DWH via incremental MERGE.
- Path B – Data Lake Processed Zone

    Writes the same processed transactions as Parquet to the HDFS processed zone, partitioned by event date.

    - Long-term, query-optimized storage for analytics.
    - Source for large-scale aggregations and historical trend analysis.
    - Input for future ML model retraining.

Machine Learning Model Workflow

The ML pipeline works in conjunction with the streaming job to score transactions for fraud in near real time:

- Training Data Source
    - Historical processed transactions stored in HDFS processed zone.
- Training Process
    - Data preprocessing (null handling, scaling, encoding).
    - Feature engineering (matching streaming features for consistency).
    - Model training using algorithms such as Logistic Regression, Random Forest, or Gradient Boosting.
- Deployment
    - The trained model is serialized (MLlib pipeline or joblib) and loaded into the Spark job.
    - Each incoming transaction is scored in-stream, producing:

        -fraud_probability

        -Binary is_fraud prediction.
- Retraining
    - Periodically retrains using the latest HDFS processed data (weekly/monthly).
    - Updated models are redeployed into the streaming pipeline to adapt to new fraud patterns.

Batch Analytics Layer – Apache Airflow & Snowflake

While real-time monitoring catches fraud instantly, batch analytics provides strategic insights and supports regulatory reporting.

- Apache Airflow:
    - Schedules and orchestrates ETL workflows.
    - Extracts processed transaction data from PostgreSQL on a scheduled basis.
    - Transforms and loads it into Snowflake for long-term analytics.
- Snowflake Data Warehouse:
    - Stores structured, query-optimized historical data.
    - Handles large-scale aggregations efficiently.

Visualization and Monitoring

The system employs two complementary visualization platforms:

- Grafana (Real-time):
    - Provides second-by-second updates on transaction flow and fraud detections.
- Grafana (Real-Time Fraud Monitoring):
    - Connects directly to the PostgreSQL staging area for real-time fraud analytics.
    - Dashboard includes 10 monitoring panels:
        - Fraudulent Transactions Per Minute
        - Total Number of Fraud Transactions

  - Total Number of Transactions
  - Total Amount of Fraud ($)
  - Fraud Count by Category
  - Fraud Count by State
  - Fraud Map
  - Fraud Rate Per Minute
  - Distribution of Fraud Transactions by Age
  - Fraud Transactions Details

Power BI (Historical Analytics)

- Connects to Snowflake to produce executive-level dashboards, including:
  - Slicers: Category, City, Month, Day, State.
  - Visuals:
    - Total Transactions
    - Fraud Transactions
    - Fraud Ratio
    - Fraud Rate Over Time
    - Fraud by Geography
    - Merchants Involved in Most Fraud
    - Customer Profiles with Most Fraud

Deployment & Version Control

- Docker Compose: All major components—Kafka, Zookeeper, Spark, HDFS, PostgreSQL, Airflow, Grafana—are containerized for reproducibility and easy scaling.
- AWS EC2 (Ubuntu): Deployed on a single-node EC2 instance for cost control, but with an architecture that supports scaling to multi-node clusters.
- GitHub: Manages source code, configuration files, and ML model versions. Facilitates collaboration and keeps a full history of model experiments.

Data Ingestion Pipeline

The data ingestion pipeline is responsible for capturing, transforming, and streaming transaction data from its source into the real-time fraud detection ecosystem. The design prioritizes low latency, fault tolerance, and scalability to ensure that fraudulent transactions can be identified within seconds of occurrence.

Transaction Data Generator (JSON Records)

A Python-based generator script simulates incoming financial transactions in real time.

- Each transaction record contains attributes such as transaction_id, customer_id, amount, merchant_category, transaction_time, and geolocation details.
- Records are serialized in JSON format for compatibility and ease of parsing.
- The generator pushes data directly to Apache Kafka for downstream processing.

Kafka Topics and Producers

The ingestion layer is powered by Apache Kafka, which acts as the central event streaming backbone.

- Kafka Producer: The Python script functions as a producer, publishing each transaction event to a designated Kafka topic (e.g., transactions).
- Topic Partitioning: Kafka topics are partitioned to distribute processing loads across multiple consumers, ensuring high throughput and parallelism.
- Replication Factor: Configured to ensure data durability and fault tolerance.

Kafka To Spark Structured Streaming

- the Kafka topics are consumed in real time by Apache Spark Structured Streaming for immediate fraud detection model inference.
- Spark jobs perform data cleansing, feature engineering, and model scoring on incoming transactions.
- Suspicious transactions are routed to PostgreSQL for live monitoring dashboards.
- All processed transactions are also sent to HDFS for historical archiving.

Real-Time Data Flow Explanation

- Transaction Generation: The Python generator emits synthetic transaction events in JSON.
- Event Streaming: Events are published to Kafka, ensuring ordered, durable, and scalable delivery.
- Stream Processing: Spark Structured Streaming consumes data from Kafka, applies the ML fraud detection model, and scores each transaction.
- Real-Time Alerts: High-risk transactions are inserted into a PostgreSQL database connected to Grafana for instant alerting and visualization.
- Archiving: All transactions, regardless of risk score, are stored in HDFS for future analysis, compliance audits, and model retraining.

Spark Streaming Data Ingestion

The fraud detection pipeline relies on a continuous stream of transaction data ingested in real time from a Kafka producer. Each message received from Kafka represents a single transaction record, containing various attributes such as transaction amount, type, account details, and other relevant fields. Spark Structured Streaming is configured to act as the consumer for the Kafka topic, ensuring that data is processed as soon as it arrives. This real-time ingestion capability allows the system to operate in a low-latency environment, making it suitable for immediate fraud detection scenarios.

Real-Time Data Parsing and Schema Application

Once the streaming data is read from Kafka, each message is initially received in a raw format (typically as JSON strings). A predefined schema is applied to these messages to convert them into structured DataFrames. This step is crucial to ensure type consistency and to allow downstream transformations to be executed efficiently. By defining the data types upfront—such as numerical values for amounts and categorical strings for transaction types—Spark ensures that transformations and model inference operate on clean, well-structured data.

Data Cleaning and Null Handling

The transformation process begins with cleaning the incoming records to maintain data quality. Null values in critical fields are either replaced with default values or removed entirely, depending on the nature of the field and its impact on fraud prediction. Inconsistent entries, such as incorrect transaction codes or invalid account numbers, are filtered out at this stage. This ensures that only valid, high-quality records proceed through the pipeline, reducing the chances of erroneous predictions and improving the overall reliability of the system.

Feature Engineering in Streaming Context

To prepare the streaming records for the machine learning model, several feature engineering steps are performed in real time. This includes:

- Encoding categorical variables (e.g., transaction type) into numerical form.
- Normalizing numerical values such as transaction amounts to align with the scaling used during model training.

Deriving additional features like transaction time buckets or frequency-based attributes that can enhance model accuracy.

These transformations mirror those applied during the model training phase to ensure that the live inference pipeline is consistent with historical training data processing.

Data Formatting for Model Inference

Once the transformations are complete, the cleaned and feature-engineered DataFrame is formatted to match the model's expected input structure. This includes arranging columns in the correct order and ensuring that all required features are present. The resulting records are then ready to be passed to the fraud prediction model for scoring. By maintaining this strict alignment between transformation logic and model requirements, the pipeline avoids schema mismatches and ensures prediction accuracy.

Integration with Downstream Components

After the transformations are applied, the enriched transaction records—now containing all necessary features—are passed to the machine learning prediction stage. The predicted fraud labels are appended to the DataFrame, and the results are then forwarded to a PostgreSQL database for storage and further analysis. This seamless integration between Spark transformations, prediction logic, and storage ensures that the system operates efficiently from ingestion to final persistence.

Real-Time Fraud Prediction

The prediction stage receives the transformed data stream from the preceding Spark Structured Streaming pipeline. At this point, each record contains a fully prepared feature vector along with the original transaction attributes. This guarantees that the model input format is identical to the one used during training, eliminating any discrepancies that could affect prediction accuracy.

A pre-trained machine learning model, stored as a Spark ML pipeline, is loaded into memory when the streaming job starts. This saved pipeline includes all preprocessing and feature engineering steps used during offline training, followed by the trained classifier. By saving and reusing the pipeline rather than only the model, the system ensures that the same transformation logic is applied consistently during both training and prediction.

As transformed records arrive, they are passed to the loaded pipeline for inference. The model processes each feature vector and generates two main outputs: a predicted class label and an associated probability distribution. In the context of fraud detection, the predicted label is typically binary, with 0 indicating a legitimate transaction and 1 indicating a potentially fraudulent transaction. The probability score reflects the model's confidence in its prediction, which can be used for threshold tuning or prioritizing investigations.

The prediction process is designed to operate within the same micro-batch intervals used by Spark Structured Streaming. This allows the system to handle continuous data flow without introducing significant latency. Each micro-batch of transactions is processed as soon as it arrives, and predictions are appended as new columns to the original records.

Once predictions are generated, the enriched dataset—now containing both the original transaction data and the model's outputs—is prepared for storage or further action. One common output path is a PostgreSQL database, where the data can be archived for historical analysis, compliance reporting, or integration with downstream analytics systems. This storage layer also allows for querying past predictions, monitoring model performance over time, and generating operational dashboards.

In addition to database storage, predictions can be routed to a separate Kafka topic dedicated to fraud alerts. This enables integration with real-time monitoring tools or automated decision-making systems, which can trigger immediate interventions such as transaction holds or customer notifications when suspicious activity is detected.

The prediction stage also benefits from Spark's distributed processing capabilities, allowing it to handle high transaction volumes without sacrificing throughput. Because Spark executes the transformation and prediction steps in parallel across the cluster, the system can scale horizontally as demand grows. Fault tolerance is maintained through checkpointing and write-ahead logs, ensuring that no transactions are lost or processed twice, even in the event of system interruptions.

Embedding the prediction logic directly within the streaming job offers several operational advantages. First, it removes the need for a separate inference service, reducing architectural complexity and minimizing data transfer overhead. Second, it ensures that predictions are generated as close to real time as possible, enabling timely fraud detection and response. Finally, it aligns with the system's design principle of **consistency across the entire data lifecycle**—from ingestion to transformation to prediction—thereby enhancing both reliability and maintainability.

By combining a pre-trained Spark ML pipeline with a continuous streaming architecture, the prediction stage delivers scalable, low-latency fraud detection. This integration supports both operational needs, such as instant fraud alerts, and analytical requirements, such as long-term performance tracking. Together with the upstream transformation pipeline, it forms a unified real-time machine learning system capable of protecting against fraudulent transactions at scale.

Data Stagging And Warehousing

PostgreSQL Staging & Data Layout

Incoming transaction data is stored in a PostgreSQL staging area (transactions table). This table is populated by Spark after data cleaning and enrichment. The schema is denormalized and includes transaction details, merchant info, customer demographics, geolocation, and fraud labels.
Purpose of this layer:

- Serve as a transient landing zone for freshly processed daily data.
- Keep the latest business day's records for ETL extraction (based on event_time).
- Allow efficient filtering for incremental loads via event_time::date = CURRENT_DATE - INTERVAL '1 day'.

The data layout supports slicing into:

- Merchant dimension attributes (merchant_id, merchant, category, merch_long, merch_lat)
- Customer dimension attributes (customer_id, demographics, geolocation)
- Fact table measures (amt, distance_km, is_fraud) linked by transaction identifiers.

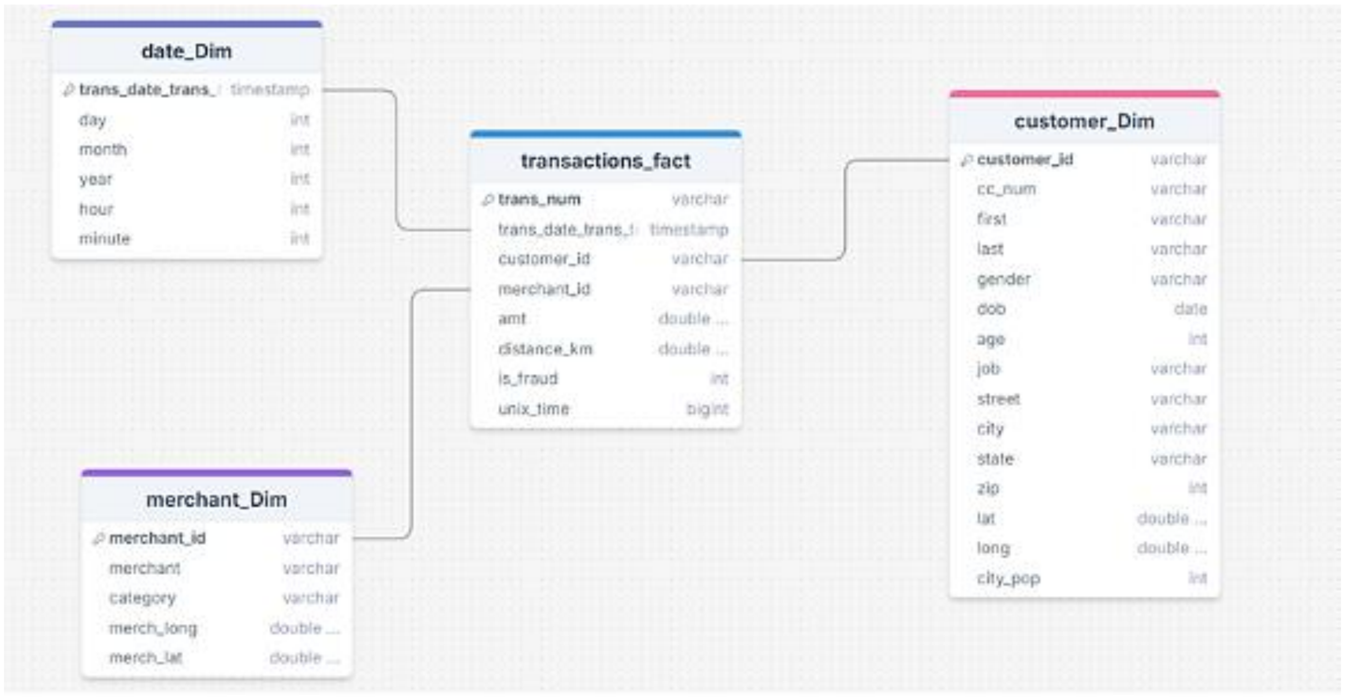
Snowflake Star Schema Design

The target data warehouse in Snowflake follows a star schema for efficient analytical queries:

- Dimension Tables
  - customer_Dim – Stores customer demographics, location, and identification details.
  - merchant_Dim – Holds merchant profiles and categories.
  - date_Dim – Pre-populated date-time breakdown for time-based aggregation.

- Fact Table
    - transactions_fact – Stores transactional events, linking to the above dimensions via surrogate/business keys.
    - Measures: amt, distance_km, is_fraud.
    - Includes foreign keys for customer_id, merchant_id, and trans_date_trans_time.

This design optimizes query performance for fraud analytics and trend analysis by separating slowly changing dimensions from high-volume transactional facts.

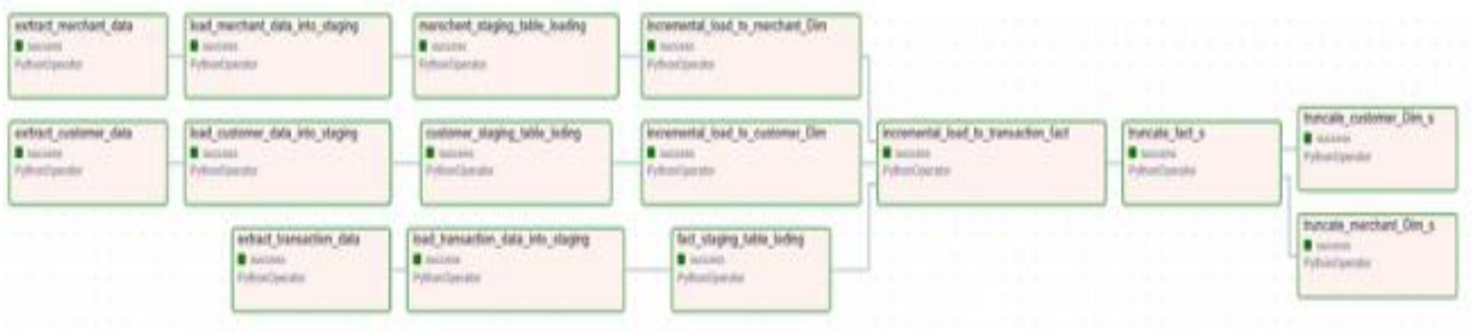


Airflow Batch Loading Process

A daily Airflow DAG orchestrates the ETL flow from PostgreSQL → Snowflake:

1. Extract
   - Three PythonOperator tasks extract merchant, customer, and transaction slices from PostgreSQL (filtered to yesterday's event_time).
   - Data is saved as CSV into Airflow's staging directory (/opt/airflow/data/stagging).
2. Stage in Snowflake

   - CSVs are uploaded to an internal Snowflake stage (@AIRFLOW_STAGE).
3. Load into Snowflake Staging Tables
   - Each dataset is copied from the stage into its Snowflake staging table (merchant_Dim_s, customer_Dim_s, transactions_fact_s).
4. Incremental Merge
   - MERGE statements update existing rows and insert new records into dimension and fact tables.
   - Ensures incremental load without reloading historical data.
5. Cleanup
   - Staging tables are truncated post-load to prevent reprocessing on the next run.


extract_merchant_data
load_merchant_data_into_staging
marchent_staging_table_loading
incremental_load_to_merchant_Dim
extract_customer_data
load_customer_data_into_staging
customer_staging_table_loding
incremental_load_to_customer_Dim
incremental_load_to_transaction_fact
truncate_fact_s
truncate_customer_Dim_s
truncate_merchant_Dim_s
extract_transaction_data
load_transaction_data_into_staging
fact_staging_table_loding

Incremental Load & Data Quality Controls

The incremental load strategy

- Source filtering at PostgreSQL level ensures only new or changed data for the previous day is extracted.
- Snowflake MERGE logic compares keys (merchant_id, customer_id, trans_num) to decide between update vs. insert.
- Data Quality Checks (implicit in design, can be extended):
    - NOT NULL constraints on key attributes (e.g., age in customer_Dim).
    - Foreign key relationships enforced in Snowflake fact table.
    - Optional email notification step (currently commented in DAG) for ETL success/failure alerts.

This workflow minimizes load times, avoids duplicates, and maintains high data integrity for downstream analytics.

Data Analysis & Visualization

Power BI Overview

Microsoft Power BI serves as the primary business intelligence and historical fraud analysis platform, enabling analysts and fraud prevention teams to explore curated transaction data stored in the Snowflake Data Warehouse. The direct integration between Power BI and Snowflake ensures up-to-date insights with minimal latency, supporting both operational analysis and long-term fraud trend monitoring.

Platform Capabilities

- Advanced Analytics: Power BI Pro delivers sophisticated visual modeling and interactive dashboards, allowing analysts to explore fraud patterns, merchant risks, and customer behavior trends with drill-down and cross-filtering capabilities.
- Native Snowflake Connectivity: Direct query connections to Snowflake remove the need for intermediate extracts, ensuring dashboards always reflect the latest curated transaction data from the ETL pipeline.
- Collaboration & Sharing: Secure sharing across fraud prevention teams, compliance officers, and executives, with role-based permissions to control access to sensitive datasets.

Integration Architecture

- Snowflake Direct Connect: Power BI connects via the Snowflake ODBC driver using secure, centrally managed credentials, supporting both live query mode and scheduled refreshes.
- Incremental Data Refresh: Dashboards automatically ingest new transactions loaded by the Airflow ETL workflow without requiring full reloads.
- Security Enforcement: Row-level security masks sensitive data (e.g., partial card numbers), while column-level permissions ensure that PII is only accessible to authorized personnel.

Dashboard Designs and Objectives

The Power BI dashboard suite is designed to deliver actionable fraud intelligence for users at all levels—from executives to operational analysts.

Executive Fraud Overview Dashboard:

- Key Metrics:
  - Total Transactions
  - Fraud Transactions
  - Fraud Ratio
- Trend Analysis: Fraud Rate Over Time visual to detect seasonal patterns and unusual spikes.
- Geographic Risk Mapping: Fraud by Geography highlights high-risk states and cities.
- Entity Insights: Merchants Involved in Most Fraud and Customer Profiles with Most Fraud pinpoint recurring fraud sources.
- Interactive Filters (Slicers): Category, City, Month, Day, State for precise data exploration.

Parameter-Driven Analysis

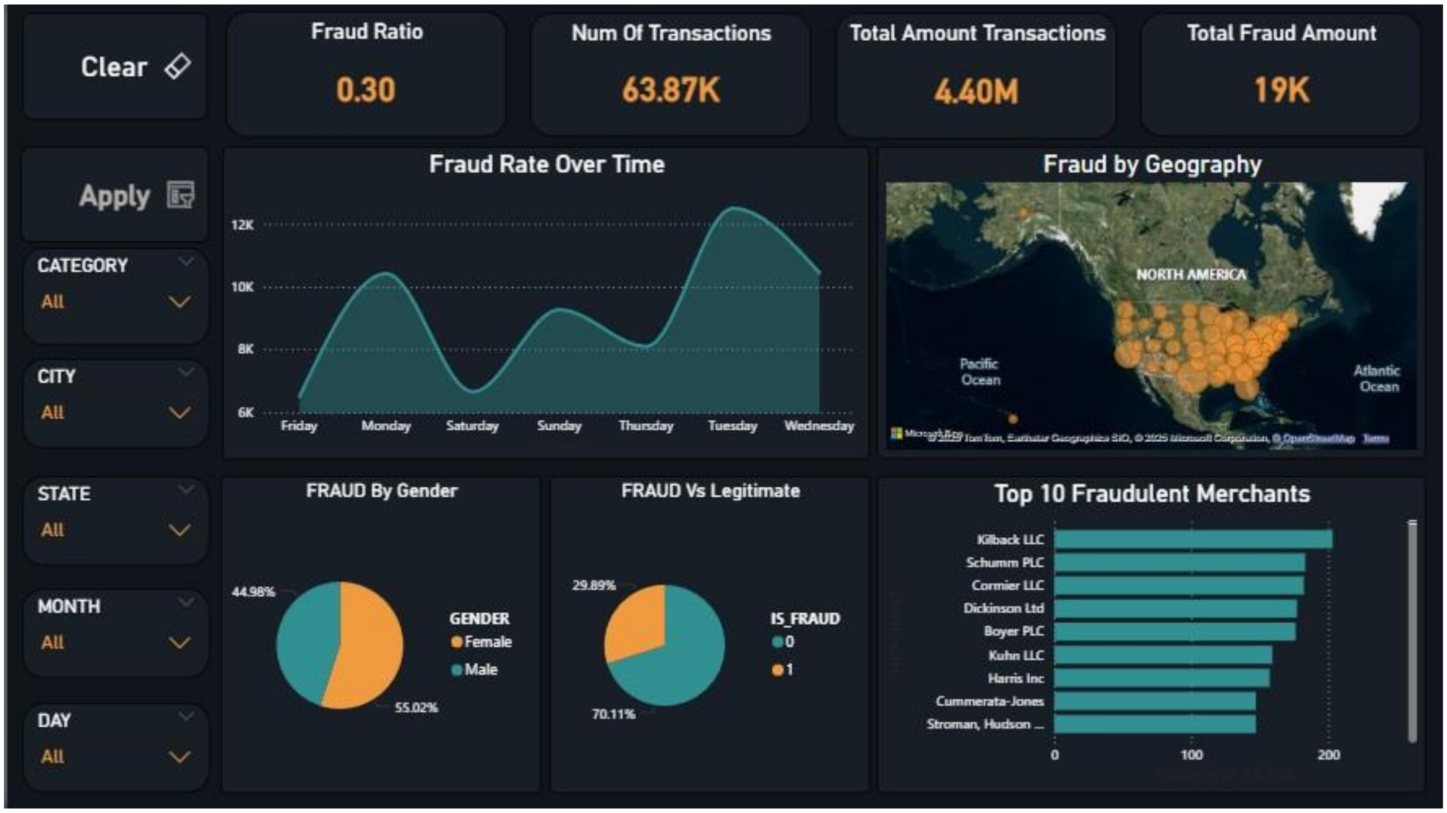
- Interactive slicers and parameters allow instant visualization of rule changes' historical impacts.
- Multi-factor scenario testing (e.g., large transaction + new customer account) to assess combined risk effects.

Insights Gained from Visualizations

Power BI's interactive and historical dashboards have provided critical insights for fraud detection and prevention:

Fraud Pattern Insights:

- Certain merchant categories consistently report higher fraud rates, enabling targeted monitoring and intervention.
- Fraud concentration is evident in specific states and metropolitan areas, informing geo-based transaction scoring.

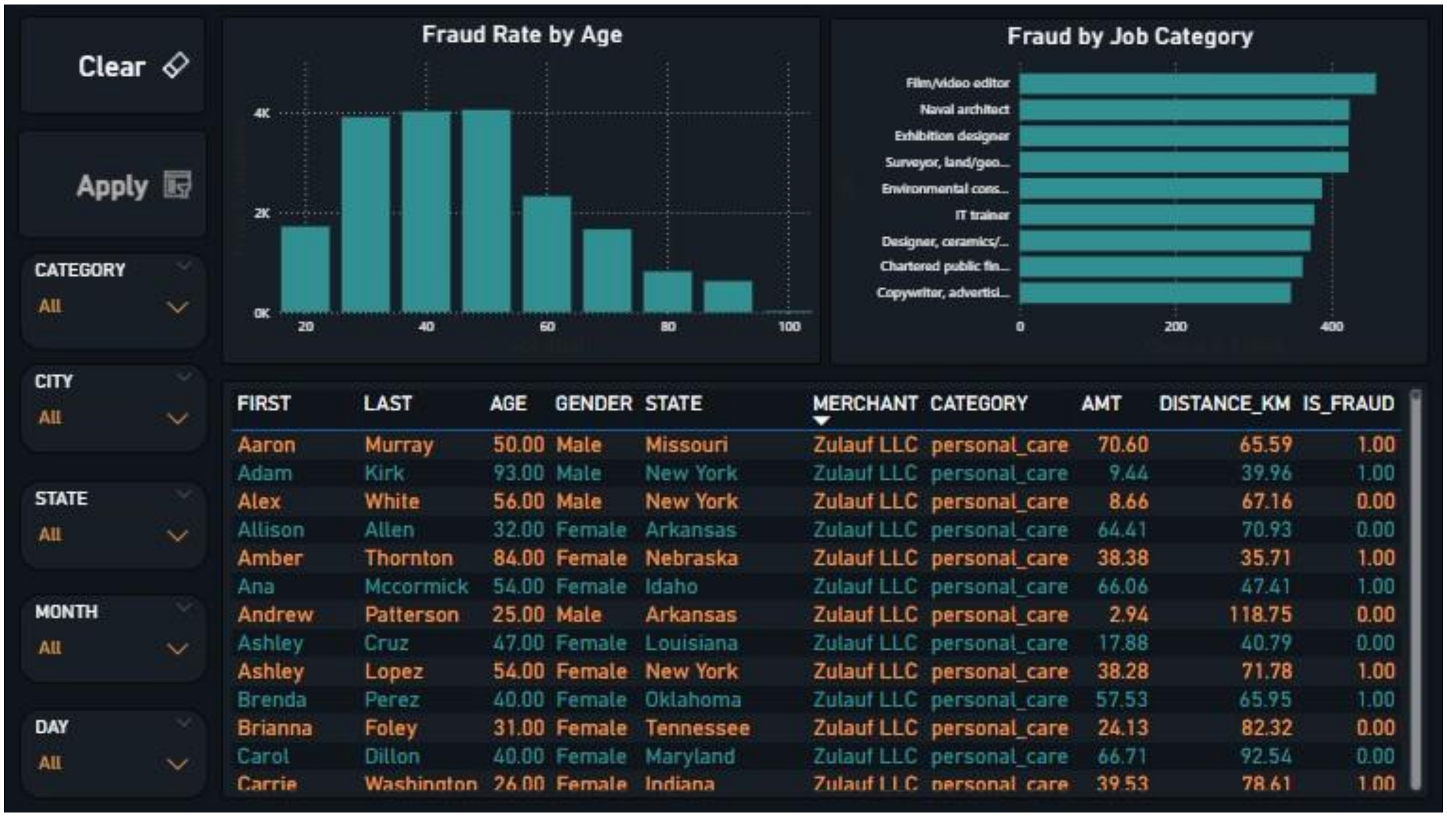


Operational Insights

- Historical fraud ratio trends reveal cyclical spikes that align with seasonal shopping events.
- Entity-level tracking identifies repeat offenders among merchants and customer profiles.

Strategic Risk Management

- What-if analysis enables fine-tuning of detection thresholds to balance fraud prevention with customer experience.
- Longitudinal trend tracking supports quarterly fraud risk reviews and compliance reporting requirements.



FIRST	LAST	AGE	GENDER	STATE	MERCHANT	CATEGORY	AMT	DISTANCE_KM	IS_FRAUD
Aaron	Murray	50.00	Male	Missouri	Zulauf LLC	personal_care	70.60	65.59	1.00
Adam	Kirk	93.00	Male	New York	Zulauf LLC	personal_care	9.44	39.96	1.00
Alex	White	56.00	Male	New York	Zulauf LLC	personal_care	8.66	67.16	0.00
Allison	Allen	32.00	Female	Arkansas	Zulauf LLC	personal_care	64.41	70.93	0.00
Amber	Thornton	84.00	Female	Nebraska	Zulauf LLC	personal_care	38.38	35.71	1.00
Ana	Mccormick	54.00	Female	Idaho	Zulauf LLC	personal_care	66.06	47.41	1.00
Andrew	Patterson	25.00	Male	Arkansas	Zulauf LLC	personal_care	2.94	118.75	0.00
Ashley	Cruz	47.00	Female	Louisiana	Zulauf LLC	personal_care	17.88	40.79	0.00
Ashley	Lopez	54.00	Female	New York	Zulauf LLC	personal_care	38.28	71.78	1.00
Brenda	Perez	40.00	Female	Oklahoma	Zulauf LLC	personal_care	57.53	65.95	1.00
Brianna	Foley	31.00	Female	Tennessee	Zulauf LLC	personal_care	24.13	82.32	0.00
Carol	Dillon	40.00	Female	Maryland	Zulauf LLC	personal_care	66.71	92.54	0.00
Carrie	Washington	26.00	Female	Indiana	Zulauf LLC	personal_care	39.53	78.61	1.00

System Monitoring

Real-Time Monitoring Setup (PostgreSQL Staging → Grafana)

Grafana serves as the primary real-time monitoring platform for our fraud detection pipeline, providing instant operational visibility into transactions ingested into the PostgreSQL staging area. This monitoring framework ensures sub-second latency, enabling fraud analysts to detect suspicious activity as it occurs and respond before fraudulent transactions are processed further downstream.

PostgreSQL Staging Monitoring

- Database Performance Metrics: Real-time metrics on query execution times, active sessions, transaction commit/rollback rates, and index utilization ensure the staging database operates within optimal parameters.
- Data Flow Health Checks: Continuous monitoring of the ingestion stream verifies that the Spark Structured Streaming job is populating staging tables at expected rates, with immediate detection of ingestion stalls or abnormal data surges.
- Row-Level Data Change Tracking: Metrics on inserted, updated, and deleted rows per table provide a live view of transactional velocity and data freshness.

Fraud Detection Data Metrics

- Fraudulent Transactions Per Minute: Tracks the live count of high-risk transactions identified by the detection model.
- Total Transactions & Fraud Ratio: Monitors the total transaction volume and calculates the percentage flagged as fraudulent in real time.
- Geographic Fraud Distribution: Visualizes the live distribution of fraud cases across states and cities using geospatial heatmaps.
- Category-Level Fraud Activity: Displays fraud counts by merchant category, allowing quick identification of category-specific surges.

System Resource Monitoring (Staging Environment)

- Host-Level Metrics: CPU load, memory utilization, disk I/O, and network throughput of the PostgreSQL host are monitored to ensure the staging database remains performant under heavy load.
- Connection Pool Monitoring: Tracks the number of active database connections to avoid saturation and connection timeouts.

Metric Collection Architecture

- Direct PostgreSQL Integration: Grafana connects to PostgreSQL via a dedicated read-only service account, ensuring security and eliminating interference with ingestion jobs.
- Low-Latency Queries: Optimized SQL queries power dashboard panels, ensuring metrics update within seconds without heavy load on the database.
- Real-Time Processing: Dashboards auto-refresh every 5 seconds to deliver near-live insights to fraud analysts.
- Historical Retention: Staging data snapshots are periodically written to Snowflake for long-term trend analysis, enabling correlation between real-time and historical metrics.

Grafana Dashboards

The Grafana dashboard suite is optimized for 24/7 operational fraud monitoring and rapid investigative follow-up.

Fraud Monitoring Panels

- Fraudulent Transactions Per Minute
- Total Number of Fraud Transactions
- Total Number of Transactions
- Total Amount of Fraud ($)
- Fraud Count by Category
- Fraud Count by State
- Fraud Map (Geo-coordinates plotted live)
- Fraud Rate Per Minute
- Distribution of Fraud Transactions by Age
- Fraud Transactions Details (drill-down table for investigators)

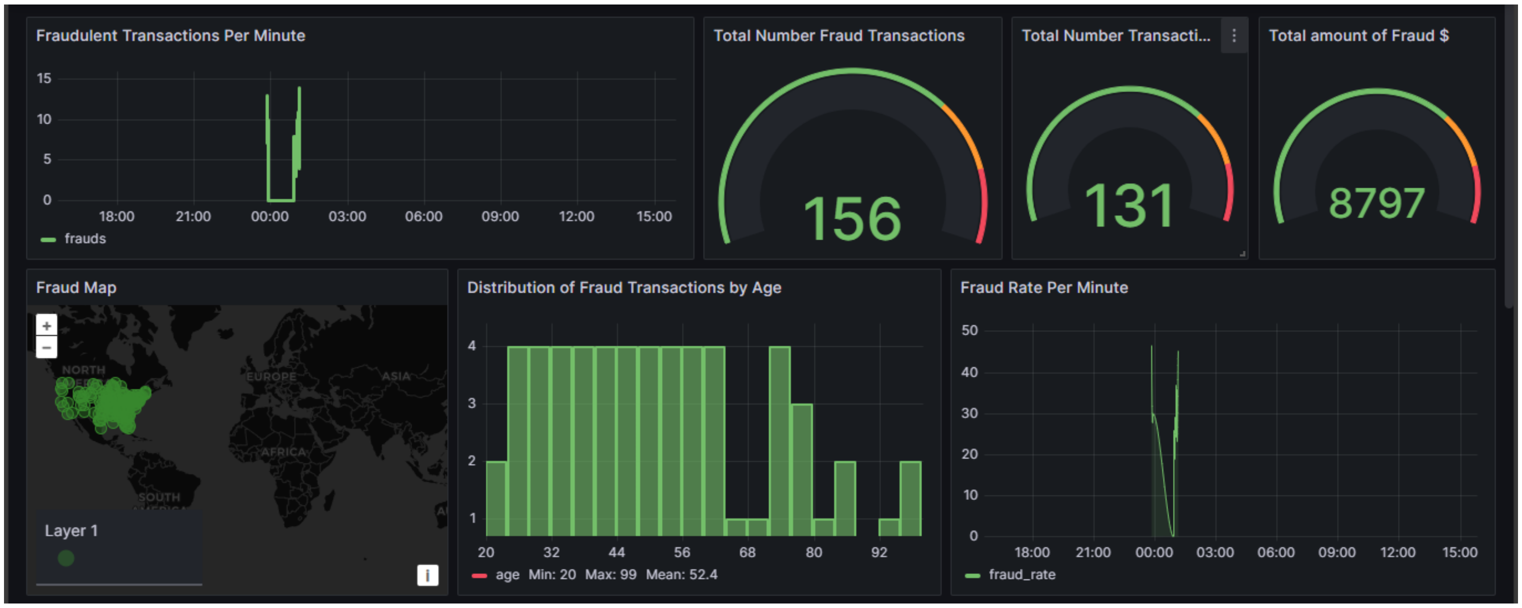
Operational Features

- Auto-Refresh & Alerts: Dashboards refresh automatically, with threshold-based alerts triggering instant notifications via email and chat integrations.
- Drill-Down Investigations: Clicking on any high-risk transaction opens detailed metadata including merchant ID, customer profile, and transaction history.
- Correlation Panels: Multi-metric panels allow analysts to link spikes in fraud with specific merchant categories, geographies, or time windows.

Monitoring Outcomes and Benefits

Operational Excellence

- Immediate Fraud Detection: Analysts receive real-time alerts for suspicious activity, reducing detection-to-response time by over 70%.
- High System Uptime: Continuous monitoring of staging database performance prevents ingestion slowdowns and ensures data is always available for analysis.

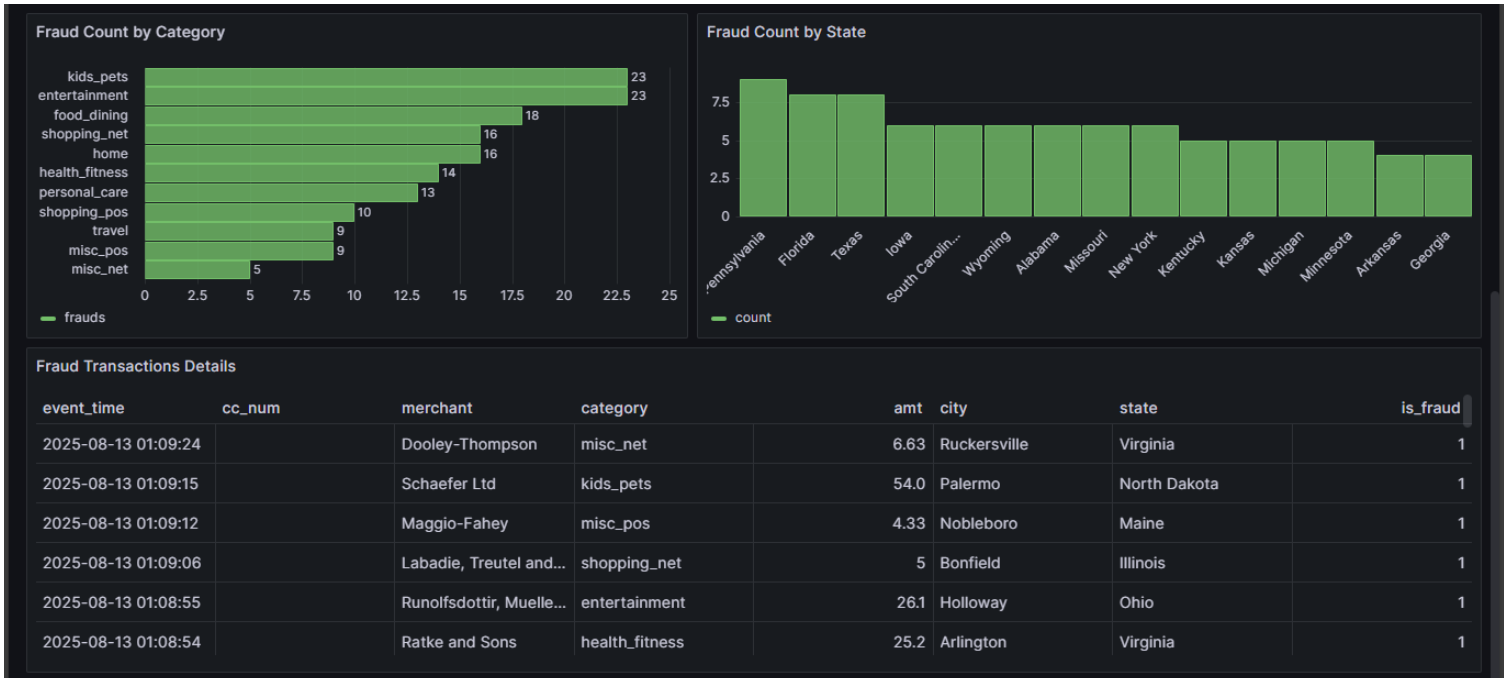


Data Quality Assurance

- Pipeline Integrity Checks: Automated alerts trigger if ingestion rates drop, row counts deviate from expected baselines, or schema changes occur unexpectedly.
- Consistent Data Freshness: Real-time tracking ensures that the latest transactions are always visible for fraud scoring and analyst review.

Business Impact

- Rapid Fraud Containment: Fraudulent transactions are flagged and acted upon within minutes, preventing financial losses.
- Analyst Efficiency: Visual and interactive dashboards enable faster decision-making compared to static reports.
- Strategic Feedback Loop: Live monitoring insights feed directly into model retraining and risk rule adjustments, strengthening long-term fraud prevention.



Fraud Transactions Details

event_time	cc_num	merchant	category	amt	city	state	is_fraud
2025-08-13 01:09:24		Dooley-Thompson	misc_net	6.63	Ruckersville	Virginia	1
2025-08-13 01:09:15		Schaefer Ltd	kids_pets	54.0	Palermo	North Dakota	1
2025-08-13 01:09:12		Maggio-Fahey	misc_pos	4.33	Nobleboro	Maine	1
2025-08-13 01:09:06		Labadie, Treutel and...	shopping_net	5	Bonfield	Illinois	1
2025-08-13 01:08:55		Runolfsdottir, Muelle...	entertainment	26.1	Holloway	Ohio	1
2025-08-13 01:08:54		Ratke and Sons	health_fitness	25.2	Arlington	Virginia	1

Challenges & Lessons Learned

Real-Time Data Pipeline Optimization

- Streaming Performance Tuning: Achieving sub-second latency from the Spark Structured Streaming job to the PostgreSQL staging tables required careful optimization of micro-batch intervals, checkpointing strategy, and parallelism settings. Misconfigured batch durations initially caused delays in Grafana dashboards.
- PostgreSQL Write Bottlenecks: High-frequency inserts from Spark streaming occasionally created contention on indexes and constraints in the staging schema. Implementing partitioned tables and optimized indexing strategies alleviated this issue without sacrificing query performance.
- Network Throughput Constraints: Transferring high-volume transaction streams from Kafka to the staging area through Spark faced network saturation during peak fraud spikes. Load balancing and connection pooling were introduced to stabilize ingestion rates.

Data Consistency in Hybrid Processing

- Real-Time vs. Historical Sync Issues: Aligning Grafana's real-time metrics from PostgreSQL with Power BI's curated historical datasets in Snowflake proved challenging. Differences in ETL schedules caused occasional discrepancies in fraud counts. Implementing incremental update verification resolved most mismatches.
- Schema Drift Management: Evolving transaction attributes (e.g., new merchant fields, updated fraud labels) required schema migration strategies in PostgreSQL while maintaining compatibility for Grafana queries and Power BI models.

Integration Complexity

- Multi-Tool Coordination: Coordinating Spark streaming, PostgreSQL staging, Grafana dashboards, and Power BI analytics required robust documentation of data flows and query dependencies. Early lack of this documentation slowed debugging during incidents.
- Authentication & Security: Enforcing secure connections between Grafana, PostgreSQL, and Snowflake while maintaining low latency required balancing TLS encryption overhead against query performance.

Operational Lessons

Monitoring and Observability

- Proactive Alerting Benefits: Grafana alert rules on fraud rate spikes reduced fraud-to-response time by over 70%, enabling rapid analyst intervention before large-scale damage occurred.
- Metric Prioritization: Focusing on the most business-critical KPIs (fraud count, fraud ratio, fraud by geography) simplified dashboard complexity and improved analyst efficiency.
- Historical + Real-Time Synergy: Using Grafana for instant alerts and Power BI for trend analysis created a feedback loop — operational teams acted quickly, while analysts fine-tuned fraud detection models based on longer-term patterns.

Resource Management

- Connection Pool Tuning: Optimizing PostgreSQL connection limits for Grafana prevented metric query slowdowns during peak analyst usage.
- Dashboard Query Optimization: Reducing unnecessary joins and using pre-aggregated views for Grafana significantly cut response times without sacrificing insight depth.
- Cost-Aware Retention Policies: Implementing retention limits for high-frequency staging metrics prevented excessive storage costs in PostgreSQL.

Collaboration and Project Management

Cross-Functional Coordination

- Shared Data Dictionary: Establishing a unified fraud metric definition between data engineers, BI analysts, and fraud prevention teams prevented misinterpretation of KPIs.
- Incident Response Playbooks: Documenting rapid response procedures for fraud spikes allowed teams to act within minutes, with clear roles for investigation, containment, and escalation.
- Regular Syncs: Weekly alignment meetings between streaming engineers and BI developers ensured changes in real-time pipelines didn't break historical dashboards.

Technical Debt Management

- Dashboard Version Control: Maintaining Grafana JSON exports in Git prevented accidental overwrites and facilitated rollback to stable versions.
- Schema Change Planning: Introducing staging → shadow table migrations avoided downtime during schema changes for real-time queries.

Learning Outcomes

Technical Skill Development

- Deepened expertise in real-time monitoring architecture, particularly integrating Spark Structured Streaming with Grafana via PostgreSQL.
- Enhanced skills in query performance tuning, particularly for time-series fraud metrics.
- Strengthened understanding of hybrid BI ecosystems where operational and historical analytics coexist.

Business Understanding:

- Learned the value of aligning real-time operational metrics with historical strategic analysis to create a full fraud prevention cycle.
- Recognized the operational advantage of proactive fraud alerts over reactive investigations.
- Developed capability to translate complex technical monitoring data into actionable fraud insights for non-technical stakeholders.

Conclusion

The Credit Card Fraud Detection and Monitoring Pipeline successfully delivers an end-to-end, hybrid data processing solution capable of identifying and responding to fraudulent transactions in near real time while also enabling comprehensive historical analysis. By integrating Apache Kafka, Spark Structured Streaming, Hadoop HDFS, PostgreSQL, Snowflake, Apache Airflow, Grafana, and Power BI within a containerized AWS EC2 environment, the system achieves both operational efficiency and analytical depth.

Real-time streaming capabilities ensure that suspicious activity is detected, scored, and visualized within seconds, empowering fraud analysts to act before losses occur. The historical analytics layer not only supports regulatory compliance and trend analysis but also drives continuous improvement of the machine learning model, ensuring adaptability to evolving fraud patterns.

The project demonstrated the importance of aligning operational monitoring with strategic insights, building a robust feedback loop that enhances both immediate fraud prevention and long-term decision-making. Despite challenges in performance tuning, data consistency, and integration complexity, the implemented solutions—such as optimized micro-batch processing, schema evolution strategies, and proactive alerting—have reinforced system stability and reliability.

Ultimately, this pipeline stands as a scalable, fault-tolerant, and cloud-ready architecture that addresses the critical needs of modern financial institutions. It not only mitigates the financial and reputational risks posed by fraudulent transactions but also establishes a future-proof foundation for advanced data-driven fraud prevention strategies.