معهد تكنولوجيا المعلومات
Information Technology Institute (ITI)
فرع قناة السويس

Design and Implementation of a Cloud-Native, Real-Time Financial Fraud Detection System

Submitted to: Information Technology Institute (ITI)

Submitted by:

- Seif El-Deen Gaber
- Omar Adel
- Yasmine Samir
- Abdelrahman Wael
- Ahmed Srour

Date: August 2025

Project Supervisor: Ibrahim Mohamed

Acknowledgements

First and foremost, we express our deepest gratitude to the **Information Technology Institute (ITI)** for granting us this opportunity to undertake such a comprehensive and impactful capstone project.

The facilities, training, and support provided throughout our journey have been instrumental in shaping both our technical capabilities and our mindset as future engineers.

We are especially thankful to our instructors, advisors, and the ITI technical staff whose insights and constructive feedback guided us through every phase of the project, from early ideation to final implementation.

We also wish to thank our families and friends, whose unwavering patience, motivation, and moral support enabled us to meet deadlines, push boundaries, and stay focused on our goals.

To the members of our team—Seif El-Deen Gaber, Omar Adel, Yasmine Samir, Abdelrahman Wael, and Ahmed Sorour—this project has been a testament to the power of collaboration, trust, and mutual dedication. Each of us brought unique strengths to the table, and together, we transformed a vision into a working solution.

Finally, we recognize the broader tech community and open-source contributors whose tools, platforms, and shared knowledge have allowed us to learn and build more effectively.

Executive Summary

Financial systems have evolved rapidly, embracing speed, scale, and convenience. But with these advances comes a darker side: fraud. Traditional fraud detection mechanisms, which rely heavily on static rules and batch processing, are increasingly unable to keep up with the sophistication and velocity of modern financial threats.

This capstone project addresses that gap by designing and implementing a real-time fraud detection system, leveraging the full power of cloud-native AWS services and machine learning.

The project simulates a real-world financial environment using synthetic data and integrates technologies such as Apache Kafka, Amazon Kinesis, AWS Glue, Amazon Redshift, AWS Lambda, SNS, and Amazon QuickSight to build a pipeline that can ingest, process, score, and visualize transactional fraud in near real-time.

At the heart of this system lies a supervised machine learning model—XGBoost—trained on carefully engineered features that mimic real-world fraud patterns. Fraud detection results are integrated into an alerting workflow that notifies stakeholders within seconds of suspicious activity, enabling faster incident response.

The entire architecture is serverless, modular, and scalable, designed to meet the demands of real-world financial applications. It also includes a detailed fraud analytics dashboard, allowing business users to visualize trends, performance metrics, and threat insights in real time.

This project demonstrates not only technical proficiency in cloud engineering, data pipelines, and machine learning, but also highlights the critical importance of security, observability, automation, and cost-efficiency.

It is our hope that this work can serve as a practical blueprint for financial institutions, researchers, and technology practitioners seeking to modernize fraud detection in today’s fast-moving digital economy.

Abbreviations and Acronyms

Acronym	Full Form	Description
AWS	Amazon Web Services	A cloud computing platform offering a wide range of services including compute, storage, analytics, and AI.
EC2	Elastic Compute Cloud	A scalable virtual server service in AWS used to host applications.
S3	Simple Storage Service	AWS service for object-based cloud storage.
VPC	Virtual Private Cloud	A logically isolated section of the AWS cloud where resources are launched.
IAM	Identity and Access Management	AWS service to manage user permissions and roles.
SNS	Simple Notification Service	A fully managed messaging service for application and user notifications.
Glue	AWS Glue	Serverless data integration service used for ETL processes.
ETL	Extract, Transform, Load	A process of collecting, cleaning, and storing data from various sources.
ML	Machine Learning	A field of AI that enables systems to learn patterns from data.
XGBoost	Extreme Gradient Boosting	A powerful machine learning algorithm based on decision trees.
ROC-AUC	Receiver Operating Characteristic - Area Under Curve	A performance metric used in classification problems.
F1-Score	F1 Score	Harmonic mean of precision and recall; measures model accuracy.
KPI	Key Performance Indicator	Metrics used to evaluate the success of a system or model.
SLA	Service Level Agreement	A contract that defines the expected performance and uptime of a system.
JSON	JavaScript Object Notation	A lightweight format for storing and transporting data.
CSV	Comma-Separated Values	A simple file format used to store tabular data.
OSS	Open Source Software	Software with source code that anyone can inspect, modify, and enhance.
SDK	Software Development Kit	A collection of software tools and libraries for building applications.
CLI	Command-Line Interface	A text-based interface for interacting with software or the operating system.

Glossary of Terms

Term	Definition
Fraud Persona	A simulated behavioral profile of a fraudulent user created to test the system's ability to detect specific fraud patterns (e.g., account takeover, velocity attack).
Apache Kafka	A high-throughput, distributed messaging system used to collect and forward streaming data.
Amazon Kinesis	AWS service used to collect, process, and analyze real-time data streams.
Amazon Redshift	A fully managed, petabyte-scale cloud data warehouse service by AWS.
AWS Lambda	A serverless compute service that automatically runs code in response to events.
Amazon QuickSight	A cloud-powered business intelligence service used to visualize data and build dashboards.
ETL Pipeline	A sequence of data processing steps—Extract, Transform, Load—used to prepare raw data for analytics or machine learning.
Data Lake	A centralized storage repository (like S3) that holds raw data in its native format until needed for analysis.
Data Warehouse	A structured storage system optimized for fast queries on large datasets (e.g., Redshift).
Serverless Architecture	A design approach where cloud providers manage server provisioning, allowing developers to focus only on writing application logic.
Real-Time Processing	The capability of a system to process and respond to data events immediately (or within seconds) as they occur.
Machine Learning Inference	The process of using a trained ML model to make predictions on new, unseen data.
Anomaly Detection	A technique used in fraud detection to identify unusual or suspicious patterns that deviate from normal behavior.
Event-Driven Architecture	A software design where system behavior is triggered by events such as data arrival, function execution, or user actions.
Step Functions	AWS service that coordinates multiple AWS services into serverless workflows.
CloudWatch	AWS service for monitoring resources and applications, collecting logs and triggering alarms.
Model Drift	The degradation of a machine learning model's performance over time due to changes in data patterns.
Concept Drift	When the statistical properties of the target variable change over time, making the model's assumptions invalid.
SCD Type 2	Slowly Changing Dimension technique used in data warehousing to keep a history of changes in dimensional data.

Table of Contents

1. Abstract

The accelerated digital transformation of the global financial ecosystem has triggered a parallel surge in the frequency, complexity, and scale of financial fraud.

Sophisticated attack vectors such as synthetic identity fraud, account takeover, and real-time payment scams now exploit weaknesses in conventional fraud detection systems—most of which are designed around static rule engines, batch processing, and post-event forensic analysis.

These approaches fail to meet the low-latency, high-throughput, and adaptive decision-making needed in modern financial systems.

In response, this project presents the design and implementation of a fully cloud-native, real-time fraud detection pipeline built exclusively on Amazon Web Services (AWS), capable of ingesting, processing, detecting, alerting, and visualizing fraudulent activity as it happens.

The solution architecture integrates a custom synthetic data generator that simulates high-volume, high-velocity financial transactions, blending legitimate behaviors with complex fraud scenarios. This component allows the system to be tested and benchmarked under varied, realistic operational conditions without relying on sensitive real-world data.

The generated data stream is published into a self-managed Apache Kafka cluster deployed on EC2, offering high availability and fine-grained control over message partitioning, replication, and throughput tuning. Kafka acts as the backbone of the streaming layer, which interfaces directly with Amazon Kinesis to leverage AWS-native scalability and reduce operational overhead in stream ingestion.

Once ingested, the raw transactional data undergoes transformation and enrichment via AWS Glue, which applies schema normalization, timestamp parsing, categorical encoding, and other preprocessing steps.

These transformations enable the downstream machine learning pipeline to consume high-quality, feature-rich data. Processed records are written to Amazon Redshift—a columnar data warehouse optimized for complex analytical queries—which serves both historical modeling and real-time scoring use cases.

At the core of the fraud detection engine is a supervised machine learning model trained via the XGBoost algorithm, known for its robustness to imbalanced datasets and ability to handle non-linear relationships.

The model is trained on engineered features such as transaction velocity, geolocation deviation, merchant trust score, and user behavioral profiling, all derived from the enriched dataset.

Model inference is deployed as a serverless AWS Lambda function, capable of evaluating incoming events in real-time. Fraud probabilities exceeding a predefined threshold trigger an event-driven notification workflow orchestrated via AWS Step Functions.

The alerting system utilizes Amazon SNS to send immediate alerts via email, SMS, or webhook to fraud analysts, security teams, or downstream automated systems. This event-based architecture ensures minimal detection-to-response latency.

System observability is achieved through Amazon CloudWatch, which collects and visualizes metrics such as model inference time, throughput per partition, consumer lag in Kafka, ETL job duration, and Lambda error rates. This instrumentation is essential for maintaining SLA compliance and diagnosing performance bottlenecks.

To facilitate operational analytics and investigation, our AWS-based solution includes a dynamic reporting and visualization layer using AWS QuickSight.

Analysts are equipped with an interactive dashboard presenting real-time fraud trends, model confidence distributions, alert volumes over time, and performance KPIs. These insights support root cause analysis, fraud typology classification, and continuous improvement of detection logic.

This document offers a complete walkthrough of the solution—from architecture diagrams, infrastructure as code (IaC) scripts, and model training workflows, to performance evaluation using metrics that include precision, recall, ROC-AUC, and F1-score.

Furthermore, it discusses deployment trade-offs, cost optimization strategies, and security considerations such as IAM role boundaries, encryption at rest and in transit, and audit logging.

The result is a scalable, low-latency, and modular fraud detection pipeline that demonstrates how modern cloud-native technologies can be orchestrated to meet the urgent need for proactive fraud mitigation in real-time financial systems.

---

2. Introduction

2.1. Background and Motivation

In today's interconnected digital economy, the scale and frequency of financial transactions have increased significantly due to the widespread adoption of online banking, digital wallets, contactless payments, and real-time financial services. This shift has improved accessibility and user convenience across industries, but it has also introduced new operational and security challenges. Among the most pressing is the rise in financial fraud, which has grown in both volume and complexity.

Fraudsters now employ a range of sophisticated techniques that target weaknesses in digital payment infrastructures. Common methods include account takeovers, where unauthorized users gain access to legitimate accounts; synthetic identity fraud, where fictitious identities are created using real and fabricated data; and velocity attacks, where multiple rapid transactions are executed to avoid detection thresholds. These techniques are increasingly automated and coordinated, allowing attackers to operate at a scale that is difficult to manage with legacy systems.

The financial consequences are well-documented. According to the U.S. Federal Trade Commission (FTC), consumer-reported losses to fraud totaled $10 billion in 2023, marking a significant year-over-year increase. Globally, the 2024 Global Financial Crime Report by Verafin estimated that banks and financial institutions experienced over $485 billion in fraud-related losses during the previous year. These figures reflect not only the direct costs of fraud but also the broader economic impact on consumers, businesses, and regulatory systems.

Despite the scale of the problem, many financial institutions still rely on traditional fraud detection methods that are limited in speed, scope, and adaptability. These systems often operate on batch-processing pipelines, where data is collected, processed, and analyzed at fixed intervals, such as hourly or daily. Detection rules are typically static and predefined, meaning they can only identify known fraud patterns and struggle to detect novel or evolving schemes. As a result, fraudulent activities may go unnoticed for extended periods, during which transactions are completed and funds irreversibly transferred.

This latency introduces a critical vulnerability: the inability to act on suspicious activity in real-time. By the time fraud is detected through retrospective analysis or manual investigation, the opportunity to prevent financial loss has already passed. The operational cost of managing false positives and conducting post-incident reviews also places a heavy burden on compliance and risk teams.

This project is motivated by the need for a shift in approach—from delayed, reactive detection to proactive, real-time fraud prevention. The proposed solution leverages cloud-native infrastructure, real-time data streaming, and machine learning algorithms to build a detection pipeline capable of analyzing and responding to transaction events as they occur.

2.2. Project Scope

The scope of this project is to design, build, and document an end-to-end, real-time fraud detection pipeline on the AWS cloud platform. The project encompasses the entire data lifecycle, from generation to actionable insight. The key stages are:

- **Data Generation:** Development of a synthetic transaction data simulator in Python to create a large-scale, realistic, and labeled dataset. This dataset includes a variety of predefined fraud scenarios, providing the "ground truth" necessary for supervised machine learning.
- **Data Ingestion:** Implementation of a robust ingestion layer using Apache Kafka and Amazon Kinesis to

capture high-velocity streaming data.

- **Data Processing and Storage:** Creation of ETL pipelines with AWS Glue to process and transform the data, and establishment of a centralized data warehouse in Amazon Redshift and a data lake in Amazon S3.
- **Machine Learning:** Development of a predictive model using the XGBoost algorithm to classify transactions as fraudulent or legitimate. This includes feature engineering, model training, evaluation, and deployment.
- **Automation and Orchestration:** Use of AWS Step Functions to automate the entire data pipeline, from ingestion to processing and model execution, creating a cohesive and manageable workflow.
- **Real-Time Alerting:** Implementation of a serverless alerting system using AWS Lambda and Amazon SNS to send instant notifications when fraud is detected.
- **Monitoring and Logging:** Integration of Amazon CloudWatch to centralize logs from all system components for comprehensive monitoring, debugging, and operational oversight.
- **Visualization and Reporting:** Creation of an interactive dashboard in Amazon QuickSight to visualize key fraud metrics, transaction trends, and model performance, providing actionable insights for fraud analysts.

This project aims to serve as a comprehensive blueprint for building a modern, scalable, and effective fraud detection system capable of meeting the challenges of the digital age.

---

3. Problem Statement

3.1. Challenges in Traditional Fraud Detection

Financial institutions have long grappled with the challenge of detecting and preventing fraud. However, their traditional systems face several fundamental limitations that render them less effective against modern, agile fraud schemes:

- **Latency:** The most significant drawback is the reliance on **batch processing**. Data is often collected over a period (e.g., several hours or a full day) and then processed in a single batch. This inherent delay means that fraudulent transactions are often discovered long after they have been completed, making it impossible to prevent the initial loss.
- **Static, Rule-Based Systems:** Many legacy systems depend on a set of hard-coded rules to flag suspicious activity (e.g., "flag any transaction over $1,000 from a new location"). While useful, these rules are brittle and easy for fraudsters to circumvent. As fraudsters adapt their tactics, these rules quickly become outdated and require constant manual updates, which is a slow and labor-intensive process.
- **Scalability Issues:** Traditional on-premises infrastructure often struggles to handle the massive volume and velocity of modern transaction data. As transaction volumes peak, these systems can become overwhelmed, leading to processing delays and an increased risk of missing fraudulent activities.
- **Data Silos:** In many organizations, data is fragmented across different systems and departments. This lack of a unified view makes it difficult to spot complex fraud patterns that span multiple channels or products. For example, a fraudster might use information stolen from one system to perpetrate fraud in another.
- **Inability to Detect Novel Fraud Patterns:** Rule-based systems can only detect known fraud patterns. They are ineffective against new, emerging attack vectors that have not been seen before. Sophisticated schemes, like synthetic identity fraud, can go undetected for long periods.

3.2. The Need for a Real-Time Solution

To overcome these challenges, a modern fraud detection system must be built on a foundation of real-time data processing and machine learning. The core requirements for such a system are:

- **Low-Latency Processing:** The system must be able to ingest, process, and analyze transaction data within seconds, or even milliseconds, of its creation. This is essential for blocking fraudulent transactions *before* they are finalized.
- **Adaptive and Intelligent Detection:** The system must move beyond static rules and employ machine learning models that can learn from historical data to identify complex, subtle, and non-linear patterns indicative of fraud. These models can adapt to new fraud tactics without requiring manual intervention.
- **Scalability and Elasticity:** The underlying infrastructure must be able to scale automatically to handle fluctuating transaction volumes. Cloud-native services are ideal for this, as they provide the ability to provision and de-provision resources on demand, ensuring performance while optimizing costs.
- **Unified Data Platform:** The system requires a centralized data lake and data warehouse to consolidate data from all sources. This provides a holistic view of customer behavior and transaction

activity, enabling the detection of more sophisticated, cross-channel fraud patterns.

- **Automation and Orchestration:** To manage the complexity of a multi-stage data pipeline, a robust workflow orchestration tool is needed. This ensures that all processes run in the correct sequence, with proper error handling and retry logic, minimizing the need for manual oversight.

---

4. Objectives

The goal of this project is to develop a fully functional, end-to-end real-time fraud detection system on the AWS cloud. Here's a list of the objectives required:

1. **Develop a High-Fidelity Synthetic Data Source:** Create a Python-based data generator capable of producing a large-scale and realistic dataset. The dataset must be context-specific (e.g., Egyptian-centric), include a wide range of transaction attributes, and embed a variety of well-defined, labeled fraud patterns to serve as a reliable ground truth for model training.
2. **Implement a Scalable Real-Time Ingestion Pipeline:** Architect and build a data ingestion layer that can handle high-velocity streaming data. This involves setting up a resilient Apache Kafka cluster on EC2 for initial buffering and integrating it with Amazon Kinesis Data Streams and Kinesis Firehose for managed, scalable ingestion into the AWS ecosystem.
3. **Build a Predictive Machine Learning Model:** Train, evaluate, and deploy a high-performance machine learning model using the XGBoost algorithm. The model must be capable of accurately classifying financial transactions as either fraudulent or legitimate, with a strong focus on achieving a high F1-score to balance precision and recall.
4. **Establish an Automated End-to-End Workflow:** Utilize AWS Step Functions to orchestrate the entire data pipeline. This includes automating the sequence of AWS Glue jobs for ETL, the machine learning inference process, and the final alerting mechanism, creating a hands-off, event-driven system.
5. **Create an Instantaneous Fraud Alerting System:** Implement a serverless alerting mechanism using AWS Lambda and Amazon SNS. The system must trigger an immediate email notification to designated stakeholders the moment a transaction is classified as fraudulent by the machine learning model.
6. **Construct a Centralized Data Warehouse:** Design and populate a data warehouse in Amazon Redshift using a modified star schema. This warehouse will store cleaned, structured, and enriched data, optimized for complex analytical queries and business intelligence.
7. **Develop a Comprehensive Monitoring and Logging Solution:** Integrate Amazon CloudWatch to capture and centralize logs, metrics, and events from all AWS services used in the pipeline. This will provide a unified view for troubleshooting, performance monitoring, and operational health checks.
8. **Build an Interactive Insights Dashboard:** Create a dynamic and user-friendly dashboard in Amazon QuickSight. The dashboard will visualize key performance indicators (KPIs) such as fraud rates, transaction volumes, model accuracy, and false positive rates, empowering fraud analysts with actionable insights.
9. **Ensure Secure and Compliant Infrastructure:** Deploy all components within a secure Virtual Private Cloud (VPC), adhering to the principle of least privilege through meticulously configured IAM roles and security groups.

---

5. Prerequisites

To fully understand, replicate, and extend this project, a certain level of technical proficiency and access to specific tools and services are required.

5.1. Technical Skills

- **Python Programming:** Intermediate to advanced proficiency in Python is essential. This includes experience with data manipulation libraries (Pandas, NumPy), machine learning frameworks (scikit-learn, XGBoost), and AWS SDKs (Boto3).
- **SQL:** Strong knowledge of SQL for querying and data manipulation, particularly with experience in a data warehousing context like Amazon Redshift.
- **AWS Cloud Services:** Foundational knowledge of core AWS services is crucial. This includes hands-on experience with:
  - **Compute:** EC2
  - **Storage:** S3
  - **Networking:** VPC, Subnets, Security Groups, IAM
  - **Databases:** Redshift
  - **Analytics & Streaming:** Kinesis, Glue, QuickSight
  - **Serverless:** Lambda, Step Functions, SNS
- **Big Data Concepts:** Familiarity with concepts of data streaming, ETL/ELT processes, data lakes, and data warehousing.
- **Docker and Docker Compose:** Basic understanding of containerization for deploying the Apache Kafka cluster on EC2.
- **Command-Line Interface (CLI):** Comfort working with a Linux/Unix shell for managing the EC2 instance and running scripts.

5.2. Software and Tools

- **AWS Account:** An active AWS account with sufficient permissions to create and manage the resources listed above.
- **Python 3.x:** A local or server-based Python environment.
- **AWS CLI:** The AWS Command Line Interface configured with appropriate credentials.
- **Docker and Docker Compose:** Installed on the EC2 instance or a local machine for testing.
- **An IDE or Code Editor:** Such as Visual Studio Code, PyCharm, or Jupyter Notebook for developing and testing scripts.
- **Git:** For version control and managing code.

5.3. AWS Services

The following AWS services are integral to the project's architecture:

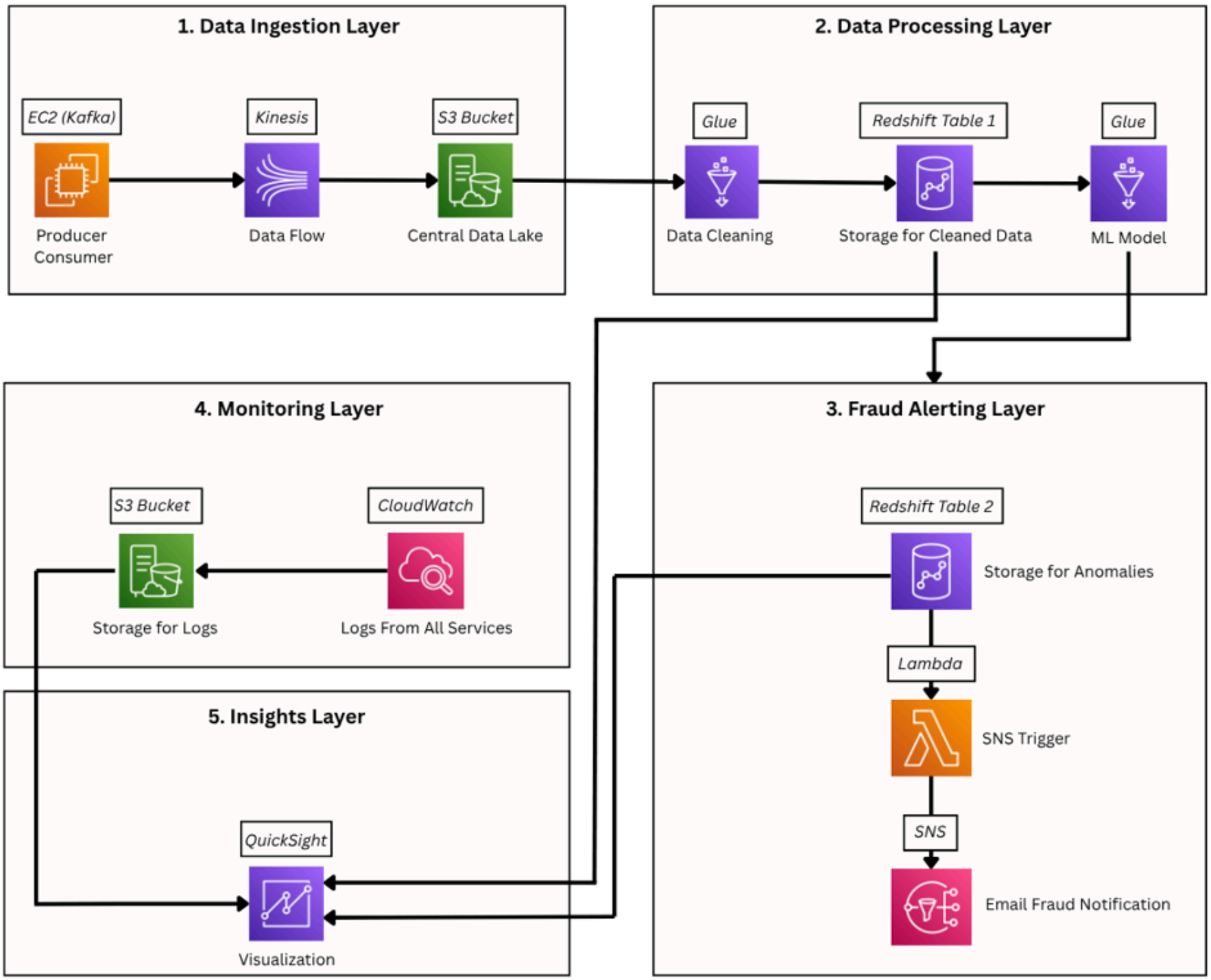
- **Amazon EC2 (Elastic Compute Cloud):** To host the Apache Kafka cluster and run the data producer/consumer scripts.
- **Amazon S3 (Simple Storage Service):** To serve as the central data lake for raw and processed data, and for storing the trained machine learning model.
- **Amazon Kinesis:** To ingest real-time streaming data at scale.
- **Kinesis Data Firehose:** To reliably deliver streaming data to S3.
- **AWS Glue:** For serverless ETL jobs to perform data cleaning, transformation, feature engineering, and model inference.
- **Amazon Redshift:** As the cloud data warehouse for storing structured data and serving analytics queries for the QuickSight dashboard.
- **AWS Lambda:** To run serverless functions that trigger alerts.
- **Amazon SNS (Simple Notification Service):** To send email notifications for fraud alerts.
- **AWS Step Functions:** To orchestrate the entire multi-step data pipeline.
- **Amazon CloudWatch:** For centralized logging, monitoring, and setting alarms.
- **Amazon QuickSight:** For data visualization and business intelligence.
- **AWS IAM (Identity and Access Management):** To manage secure access to resources through roles and policies.

---

6. System Architecture

6.1. Architectural Overview

The Real-Time Fraud Detection System is designed as a multi-layered, cloud-native architecture on AWS. Each layer performs a specific function in the data lifecycle, from initial ingestion to the final delivery of insights and alerts. This layered approach ensures modularity, scalability, and maintainability.



The architecture is composed of five distinct layers:

1. **Data Ingestion Layer:** Captures raw transaction data from its source.
2. **Data Processing Layer:** Cleans, transforms, and enriches the data, and runs the machine learning model.
3. **Fraud Alerting Layer:** Triggers real-time notifications for detected fraud.
4. **Monitoring Layer:** Collects logs and metrics from all services for operational oversight.
5. **Insights Layer:** Visualizes data and results for human analysis.

The following sections provide a detailed breakdown of each layer.

6.2. Layer 1: Data Ingestion

Objective: To capture high-velocity, real-time transaction data reliably and scalably.

Components:

- **EC2 (Kafka Producer/Consumer):** An Amazon EC2 instance serves as the starting point of the pipeline.
  - **Producer:** A Python script (producer.py) runs on this instance. It reads from the synthetically generated CSV file and publishes each transaction as a message to a local Apache Kafka topic. This simulates a real-world application generating transaction events.
  - **Consumer:** A second Python script (consumer.py) acts as a forwarder. It consumes messages from the local Kafka topic and pushes them into the AWS managed streaming service, Amazon Kinesis.
- **Apache Kafka (on EC2):** A 3-broker Kafka cluster, deployed using Docker Compose on the EC2 instance, acts as a resilient and high-throughput buffer. This decouples the data source from the cloud pipeline, preventing data loss if the downstream services are temporarily unavailable.
- **Amazon Kinesis Data Streams:** This is the primary entry point for streaming data into the AWS ecosystem. It is a fully managed service that can handle massive volumes of data from hundreds of thousands of sources. The consumer.py script sends records to this stream.
- **Amazon Kinesis Data Firehose:** This service is configured to automatically read data from the Kinesis Data Stream. It batches the records into larger files and reliably delivers them to a specified destination.
- **Amazon S3 (Central Data Lake):** Kinesis Firehose delivers the raw, batched transaction data to an S3 bucket. This bucket acts as the project's **data lake**, providing durable, long-term storage for the raw data. The data is partitioned by arrival time (e.g., YYYY/MM/DD/HH/), which optimizes it for subsequent processing and querying.

Flow: EC2 (Producer) -> Kafka Cluster -> EC2 (Consumer) -> Kinesis Stream -> Kinesis Firehose -> S3 Lake

6.3. Layer 2: Data Processing

Objective: To transform raw data into a clean, structured format suitable for analytics and to apply the machine learning model for fraud detection.

Components:

- **AWS Glue:** This is a serverless ETL service that performs two critical jobs:
  1. **Data Cleaning ETL:** The first Glue job reads the raw data from the S3 data lake, performs cleaning and transformation operations (e.g., data type casting, handling nulls), and loads the structured data into the fact_transaction and dimension tables in the Amazon Redshift data warehouse.
  2. **ML ETL (Inference Job):** The second Glue job is triggered after the data is cleaned. It reads the transaction data from Redshift, applies the same feature engineering logic used during model training, loads the trained XGBoost model from S3, predicts whether each transaction is fraudulent, and writes the predictions back to a results table in Redshift.
- **Amazon Redshift (Cleaned Data):** The data warehouse stores the cleaned and structured transaction data in a star schema. This table is the source of truth for both ML and QuickSight.

Flow: S3 Lake -> Glue (Data Cleaning ETL) -> Redshift (Cleaned Data) -> Glue (ML ETL) -> Redshift (Predictions)

6.4. Layer 3: Fraud Alerting

Objective: To provide immediate, automated notifications when a fraudulent transaction is identified.

Components:

- **Storage for Anomalies in Redshift:** The ML ETL job writes its predictions to our data warehouse in the predicted_fraud table. A separate process or a scheduled query identifies newly added fraudulent transactions (anomalies).
- **AWS Lambda:** A Lambda function is configured to be triggered when a new fraudulent transaction is detected. The trigger could be an event from Redshift or a scheduled check. The function's role is to read the details of the fraudulent transaction (e.g., transaction ID, amount, customer).
- **Amazon SNS (Simple Notification Service):** The Lambda function, upon retrieving the transaction details, publishes a message to an SNS topic. The message is formatted into a human-readable alert.
- **Email Fraud Notification:** An email endpoint is subscribed to the SNS topic. When SNS receives the message from Lambda, it automatically forwards it as an email to the subscribed addresses (e.g., a fraud investigation team's distribution list).

Flow: Redshift (New Anomaly Detected) -> Lambda (Trigger) -> SNS Topic (Publish Message) -> Email Notification

6.5. Layer 4: Monitoring

Objective: To maintain operational health and visibility across all components of the pipeline.

Components:

- **Amazon CloudWatch:** This service acts as the central hub for monitoring.
  - **Logs:** CloudWatch Logs collects and stores logs from all services, including the EC2 instance (via the CloudWatch Agent), Kinesis, Glue jobs, and Lambda functions. This provides a centralized place for debugging and auditing.
  - **Metrics:** CloudWatch collects performance metrics from all AWS services automatically (e.g., Kinesis PutRecord latency, Glue job execution time, Lambda invocations).
  - **Alarms:** Alarms can be configured to trigger notifications (e.g., via SNS) if certain thresholds are breached (e.g., a spike in Lambda errors, a Glue job failure).
- **S3:** Certain application-level or audit logs could optionally be stored in a dedicated S3.

Flow: All AWS Services -> Amazon CloudWatch (Logs & Metrics)

6.6. Layer 5: Insights

Objective: To provide a user-friendly interface for business users and fraud analysts to explore the data, monitor trends, and investigate fraud.

Component:

- **Amazon QuickSight:** This is a serverless business intelligence service used to create interactive dashboards. It connects directly to the Amazon Redshift data warehouse. The dashboard visualizes key metrics such as:
  - Total transaction volume and value.
  - Real-time fraud rate.
  - Breakdown of fraud by merchant category, card type, etc.

    - Performance of the machine learning model (e.g., precision, false positive rate).

Flow: Redshift (Cleaned Data & Predictions) -> Amazon QuickSight (Visualization)

---

7. Key Components Deep Dive

This section provides a more detailed look at the core services and technologies that power the fraud detection system.

7.1. Apache Kafka on EC2

- **Role:** Acts as the initial ingestion point and a durable buffer for incoming transaction data.
- **Implementation:** A 3-broker Kafka cluster is deployed on a t3.medium EC2 instance using Docker and Docker Compose. This multi-broker setup provides high availability and fault tolerance; if one broker fails, the others can continue to operate.
- **Why Kafka?** Kafka is designed for high-throughput, low-latency streaming and provides strong ordering guarantees. Using a self-managed Kafka cluster offers flexibility and control, and it simulates a common real-world scenario where data originates from an on-premises or non-AWS environment. The buffering capability is critical for decoupling the data producers from the cloud consumers, preventing data loss during downstream maintenance or failures.

7.2. Amazon Kinesis Data Streams & Firehose

- **Role:** Managed, scalable, and serverless data streaming and delivery.
- **Kinesis Data Streams:** This service acts as the front door to the AWS cloud for our streaming data. It is designed to ingest massive volumes of data in real-time. The stream is provisioned with a specific number of shards, which determines its throughput capacity.
- **Kinesis Data Firehose:** This service simplifies the process of loading streaming data into data stores. It is configured to subscribe to the Kinesis Data Stream, automatically collecting, batching, and compressing the data before delivering it to our S3 data lake. This serverless approach eliminates the need to write and manage custom data delivery applications.
- **Why Kinesis?** It provides a fully managed, pay-as-you-go solution that integrates seamlessly with other AWS services like S3, Glue, and Lambda. This removes the operational overhead of managing and scaling a streaming platform.

7.3. AWS Glue

- **Role:** Serverless ETL (Extract, Transform, Load) and machine learning inference.
- **Implementation:** The project uses two distinct Glue jobs written in PySpark:
    1. **Data Cleaning Job:** This job connects to the S3 data lake, reads the raw JSON data, applies transformations (data type validation, schema enforcement), and loads the cleaned data into the Redshift data warehouse, populating the star schema tables.
    2. **ML Inference Job:** This job reads the newly loaded transaction data from Redshift, performs the necessary feature engineering, downloads the trained XGBoost model (.pkl file) from S3, applies the model to predict fraud, and writes the results back to a prediction table in Redshift.
- **Why Glue?** Glue is a serverless ETL service, meaning there are no servers to provision or manage. It

automatically scales resources based on the workload. Its integration with the Glue Data Catalog provides a persistent metadata store, and its ability to run Python/PySpark scripts makes it highly flexible for complex transformations and machine learning tasks.

7.4. Amazon Redshift

- **Role:** Cloud data warehouse for structured data storage and analytics.
- **Implementation:** A Redshift cluster is provisioned to store the transaction data in a modified star schema, consisting of one large fact table (fact_transaction) and several dimension tables (dim_customer, dim_merchant, etc.). This schema is optimized for fast analytical queries. Redshift's columnar storage and massively parallel processing (MPP) architecture allow it to execute complex queries over large datasets much faster than traditional row-based databases.
- **Why Redshift?** It is a powerful, fully managed data warehouse that can scale from gigabytes to petabytes of data. It integrates seamlessly with AWS Glue for ETL and Amazon QuickSight for visualization, making it the ideal analytical engine for this project.

7.5. AWS Lambda

- **Role:** Serverless, event-driven compute for the alerting system.
- **Implementation:** A Python-based Lambda function is created to handle fraud alerts. It is configured to be triggered whenever a new fraudulent transaction is detected. The function's code is simple: it extracts key details about the fraudulent transaction and constructs a message.
- **Why Lambda?** Lambda allows you to run code without provisioning or managing servers. You pay only for the compute time you consume. It is perfect for short-lived, event-driven tasks like sending an alert, as it can scale automatically and has built-in fault tolerance.

7.6. Amazon SNS (Simple Notification Service)

- **Role:** A fully managed pub/sub messaging service for sending notifications.
- **Implementation:** An SNS topic is created. The Lambda function publishes its alert message to this topic. An email address (or multiple addresses) is subscribed to this topic.
- **Why SNS?** SNS decouples the message producer (Lambda) from the message consumers (email, SMS, etc.). This is a highly flexible and reliable way to distribute notifications. You can easily add more subscribers (e.g., a Slack channel, an SMS number) to the same topic without changing the Lambda function's code.

7.7. Amazon QuickSight

- **Role:** Serverless business intelligence (BI) and data visualization.
- **Implementation:** QuickSight is connected directly to the Amazon Redshift data warehouse. A dataset is created by querying the transaction and prediction tables. An interactive dashboard is then built on top of this dataset, featuring various visualizations (KPIs, bar charts, line charts) to track fraud trends and model performance.
- **Why QuickSight?** It is a powerful, easy-to-use BI tool that integrates natively with AWS data sources. Its serverless architecture and pay-per-session pricing model make it a cost-effective solution for data visualization.

7.8. AWS Step Functions

- **Role:** Serverless workflow orchestration.
- **Implementation:** A Step Functions state machine is designed to define and automate the entire end-to-end pipeline. The state machine is a visual workflow that dictates the sequence of tasks. For example:
  1. **Start:** The workflow is triggered (e.g., on a schedule or by an event).
  2. **Run Glue Cleaning Job:** The first state invokes the data cleaning Glue job.
  3. **Wait for Completion:** The state machine waits for the job to succeed.
  4. **Run ML Inference Job:** Upon success, the next state invokes the ML inference Glue job.
  5. **Check for Fraud:** After the inference job, a state checks if any fraud was detected.
  6. **Trigger Alert:** If fraud is found, it invokes the alerting Lambda function.
  7. **End:** The workflow completes.
- **Why Step Functions?** It provides a reliable way to coordinate multiple AWS services into a single, automated workflow. It handles sequencing, error handling (with built-in retry logic), parallel execution, and state management, making complex pipelines easier to build, manage, and debug.

7.9. Amazon CloudWatch

- **Role:** Centralized monitoring, logging, and alarming.
- **Implementation:** CloudWatch is used passively and actively throughout the project.
  - **Passively:** It automatically collects metrics and logs from Kinesis, Glue, Lambda, Step Functions, and Redshift.
  - **Actively:** The EC2 instance is configured with the CloudWatch Agent to push system-level logs (e.g., from the Kafka producer/consumer scripts) to CloudWatch Logs. Alarms are set up to notify administrators via SNS if a critical component like a Glue job or a Lambda function fails.
- **Why CloudWatch?** It provides a unified, cross-service view of the entire system's health and performance. This is indispensable for maintaining operational excellence, troubleshooting issues quickly, and ensuring the reliability of the fraud detection pipeline.

---

8. Data Collection: The Synthetic Transaction Simulator

Since real-world financial fraud data is sensitive, private, and scarce, this project utilizes a custom-built synthetic transaction simulator.

8.1. Purpose and Design

The simulator was developed in Python with the following key objectives:

- **Realism:** To generate data that closely mimics the patterns and complexities of real-world financial transactions, with a specific focus on the Egyptian market.
- **Scale:** To produce a large volume of data (500,000 transactions) sufficient for training robust machine learning models.
- **Labeled Ground Truth:** To explicitly embed a variety of distinct, well-defined fraud patterns directly into the dataset and label each transaction with an is_fraud flag.
- **Temporal Consistency:** To move beyond generating random, disconnected rows of data by using a persona-based model. The simulator creates individual "personas" (customers) and simulates their transaction behavior over time, resulting in more realistic and temporally consistent data patterns.

8.2. Fraud Persona Topics

Around 11% of the simulated personas are designated as fraudulent, each representing a distinct vector.

Fraud Persona Class	Description of Simulated Attack Pattern
FraudPersona_AccountTakeover	Simulates a classic ATO. A single, very high-value transaction from a new device/IP, often from a geographically distant location, draining the account.
FraudPersona_CardTesting	A rapid burst of very small-value transactions from the same IP/device in quick succession to check if a stolen card is valid. Often results in multiple declines.
FraudPersona_VelocityAttack	A series of large-value transactions occurring at different merchants in a very short time frame, far exceeding normal spending velocity.
FraudPersona_ImpossibleTravel	Generates two transactions: one legitimate, followed shortly by a fraudulent one from a location that is physically impossible to travel to in the elapsed time.
FraudPersona_MerchantBustOut	A new customer colluding with a fraudulent merchant. They make several large transactions exclusively at this high-risk merchant to cash out.
FraudPersona_SpendingAnomaly	A customer who normally shops in specific categories (e.g., Grocery, Fuel) suddenly makes a very large purchase in a completely unrelated, high-risk category (e.g., Electronics).

FraudPersona_SyntheticIdentity	A persona with a very short tenure and a high-risk email domain. After a few small "warm-up" transactions to appear normal, they make a massive "cash-out" transaction and disappear.
FraudPersona_AnomalousHour	A transaction that occurs during unusual, high-risk hours (e.g., 2 AM - 5 AM), which is abnormal for the customer's typical behavior.
FraudPersona_AddressChange	Simulates a fraudster making a large online purchase immediately after the customer's address was changed (within 0-1 days).
FraudPersona_ChannelPivot	A customer who almost exclusively uses one channel (e.g., in-person POS) suddenly makes a large online transaction from a new device.

8.3. Data Schema and Dictionary

The generated CSV file contains 38 columns, providing a rich set of features for model training.

Category	Field Name	Data Type	Description	Example Value(s)
Transaction Details	transaction_id	UUID	Unique identifier for each transaction.	5959989e-30b8-4168-afbe-7276564a7331
	transaction_timestamp	TIMESTAMP_TZ	The exact date and time (UTC) when the transaction occurred.	2025-07-01T14:31:35Z
	transaction_amount	DECIMAL(10, 2)	The monetary value of the transaction.	1550.75
	currency	VARCHAR(3)	The three-letter ISO currency code.	EGP
	transaction_type	VARCHAR(10)	The method of transaction.	POS, Online, ATM, Fawry
	transaction_status	VARCHAR(10)	The outcome of the transaction attempt.	approved, declined
	transaction_hour_of_day	INTEGER	The local hour of the day (0-23) for the transaction.	16
Customer Details	customer_id	UUID	Unique identifier for the customer.	2ed0c485-437d-4cad-9617-9cb71869d2ef

	customer_name	VARCHAR(255)	[PII] Full name of the customer.	Ahmed Mahmoud Hassan
	customer_tenure_days	INTEGER	How long the person has been a customer, in days.	1068
	customer_email_domain	VARCHAR(100)	The domain of the customer's registered email.	gmail.com, yahoo.com
	customer_address_change_days	INTEGER	Days since the customer last changed their billing/shipping address.	950
Card & Auth Details	card_number_hash	VARCHAR(64)	A SHA-256 hash of the credit card number for security.	44d5d6...
	card_type	VARCHAR(20)	The brand of the card.	Visa, MasterCard, Meeza
	issuing_bank_name	VARCHAR(100)	The name of the bank that issued the card.	CIB, NBE, Banque Misr
	card_entry_method	VARCHAR(20)	How the card details were captured.	chip, swiped, contactless
	cvv_match_result	VARCHAR(15)	Result of the CVV/CVC check.	match, no_match
Merchant Details	merchant_id	UUID	Unique identifier for the merchant.	bb08a011-402f-48ff-a17a-4ed95b90ab34
	merchant_name	VARCHAR(255)	The legal name of the merchant.	Spinneys Supermarket - Maadi
	merchant_category	VARCHAR(50)	The category of the merchant's business.	Retail, Restaurants, Electronics
	merchant_latitude	DECIMAL(9, 6)	The geographical latitude of the merchant.	29.96251

	merchant_longitude	DECIMAL(9, 6)	The geographical longitude of the merchant.	31.27689
	merchant_risk_score	FLOAT	A pre-calculated risk score (0-1) for the merchant.	0.15
Contextual & Behavioral Details	ip_address	VARCHAR(45)	The IP address used for the transaction.	156.204.117.82
	ip_proxy_type	VARCHAR(10)	Type of proxy detected for the IP address.	none, vpn
	device_id	UUID	A unique identifier for the device used.	00a281fa-972e-45fd-b462-8f5e73c4c651
	device_os	VARCHAR(20)	The operating system of the device.	Android, iOS, Windows
	user_agent	TEXT	The full user-agent string from the browser/device.	Mozilla/5.0 (Linux; Android 13; ...)
	is_international	BOOLEAN	True if the card country differs from the merchant country.	FALSE
	distance_from_home_km	FLOAT	The distance in kilometers from the customer's primary address.	8.5
	distance_from_last_txn_km	FLOAT	The distance in kilometers from the location of the previous transaction.	2.1
	time_since_last_txn_sec	INTEGER	The time in seconds that has passed since the customer's last transaction.	7200
Behavioral Feature	is_first_time_customer_merchant	BOOLEAN	True if this is the customer's first transaction with this merchant.	FALSE
Target Label	is_fraud	BOOLEAN	The ground truth label. `true` if fraudulent, `false` otherwise.	FALSE

9. Data Preprocessing and Warehousing

Once the raw data is ingested into the S3 data lake, it must be processed, cleaned, and structured before it can be used for machine learning and analytics. This is accomplished through AWS Glue and Amazon Redshift.

9.1. ETL with AWS Glue

The project utilizes a serverless ETL pipeline powered by AWS Glue. The core logic is implemented in a PySpark script that runs as a Glue job.

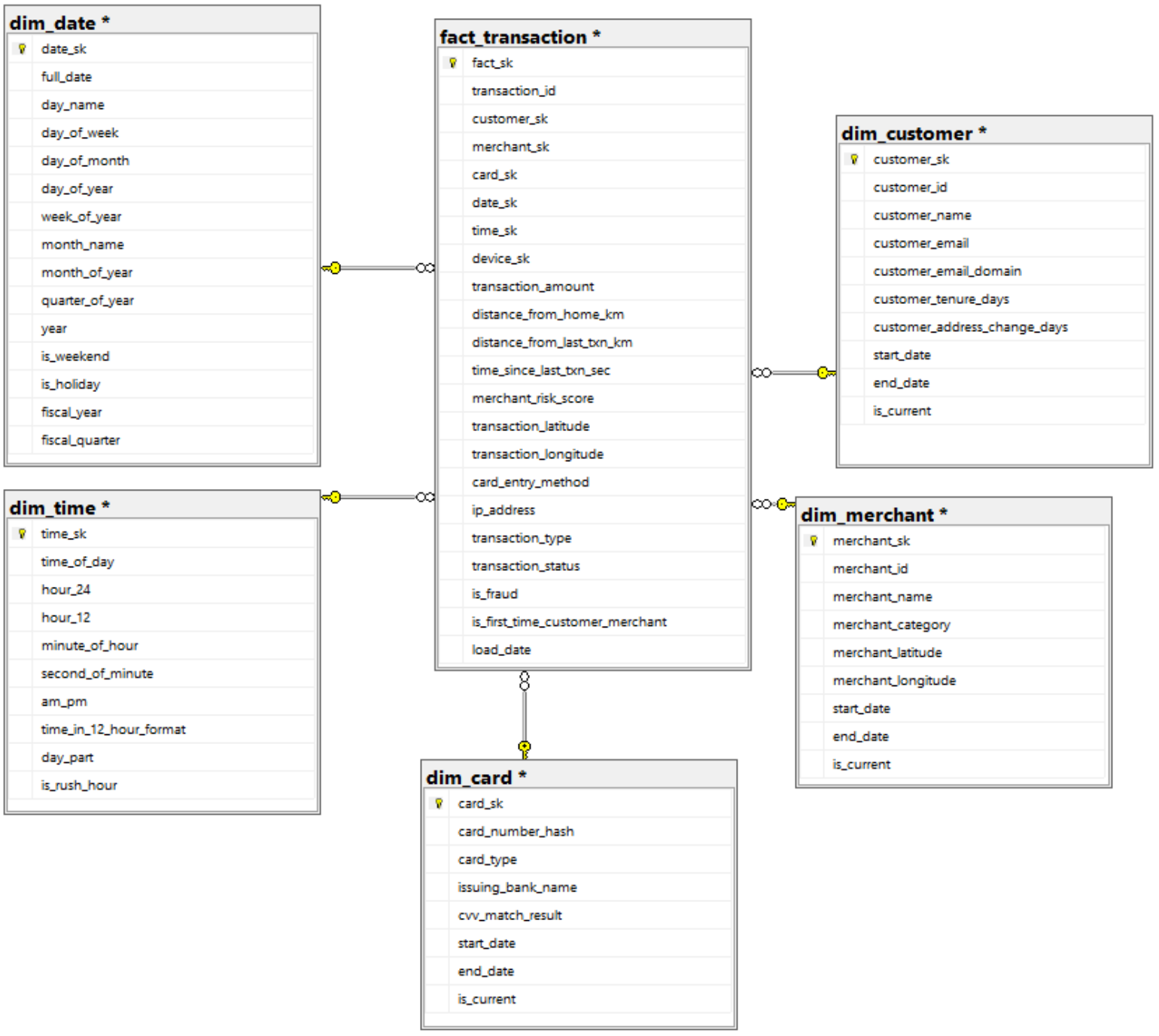
ETL Process Architecture:

1. **Extract:** The Glue job starts by reading the raw, time-partitioned JSON data from the S3 data lake. It uses the Glue Data Catalog to understand the schema of the incoming data.
2. **Transform:** This is the most critical phase, where several transformations are applied to the raw data:
    - **Data Cleaning:** Handling missing or null values, correcting data types (e.g., casting string amounts to decimals), and ensuring data consistency.
    - **Dimensional Modeling:** The script processes the flat JSON structure and prepares it for loading into the star schema of the data warehouse. This involves separating dimensional attributes (like customer, merchant, and device details) from the transactional facts.
    - **Slowly Changing Dimensions (SCD):** The pipeline implements Type 2 SCD logic for the dimension tables. This means that when an attribute of a dimension changes (e.g., a customer changes their email address), the old record is expired (by setting an end_date and is_current flag) and a new record is inserted with the updated information. This preserves a full historical record of all changes, which is crucial for analyzing trends over time.
    - **Surrogate Key Generation:** Unique, numeric surrogate keys (e.g., customer_sk, merchant_sk) are generated for each dimension record. These keys are then used as foreign keys in the fact table, which is more efficient for joins than using natural keys (like customer_id).
3. **Load:** After transformation, the Glue job loads the processed data into the Amazon Redshift data warehouse.
    - Dimension tables (dim_customer, dim_merchant, etc.) are updated using a MERGE operation to handle the SCD Type 2 logic efficiently.
    - The fact_transaction table is populated with the transactional data and the corresponding surrogate keys that link to the dimension tables.

This single-path streaming architecture, as described in the glue_doc.docx, avoids the complexity of a lambda architecture by using a unified pipeline to serve both real-time and analytical needs.

9.2. Data Warehouse Schema (Amazon Redshift)

The Amazon Redshift data warehouse is designed using a **modified star schema**, which is optimized for fast, real-time fraud detection queries. The design philosophy is to minimize the number of JOINs required for the most critical queries.



Schema Components:

- **One Fact Table:** fact_transaction
- **Six Dimension Tables:** dim_customer, dim_card, dim_device, dim_merchant, dim_date, dim_time

Fact Table: fact_transaction

This is the central table in the schema, containing the quantitative measures (facts) of each transaction.

- **Distribution Strategy:** ALL distribution is used to ensure an even distribution of data across all nodes in the Redshift cluster, which is suitable for balanced query performance.
- **Sort Key:** A compound sort key of (date_sk, is_fraud) is used. This optimizes for time-based range queries, which are common in fraud analysis (e.g., "show me all fraudulent transactions from last week").

Key Fields in fact_transaction:

Field	Data Type	Constraints	Business Purpose
fact_sk	bigint	PK, NN	Fact table surrogate key
transaction_id	character varying(65)	NN	Business transaction identifier
customer_sk	bigint	FK, NN	Customer dimension reference
merchant_sk	bigint	FK, NN	Merchant dimension reference
card_sk	bigint	FK, NN	Card dimension reference
date_sk	integer	FK, NN	Date dimension reference
time_sk	integer	FK, NN	Time dimension reference
device_sk	bigint	FK, NN	Device dimension reference
transaction_amount	double precision	NN	Transaction monetary value
distance_from_home_km	double precision	NULL	Geographic fraud indicator
distance_from_last_txn_km	double precision	NULL	Velocity fraud indicator
time_since_last_txn_sec	bigint	NULL	Temporal fraud indicator
merchant_risk_score	double precision	NULL	Merchant-based risk assessment
transaction_latitude	double precision	NULL	Geographic coordinates
transaction_longitude	double precision	NULL	Geographic coordinates
card_entry_method	character varying(20)	NULL	Payment method (embedded junk)
ip_address	character varying(15)	NULL	Device IP (embedded junk)
transaction_type	character varying(10)	NN	Transaction category (embedded junk)
transaction_status	character varying(20)	NN	Processing status (embedded junk)
is_fraud	Boolean	NN	Fraud classification flag
is_first_time_customer_merchant	Boolean	NN	New relationship indicator
load_date	timestamp	NULL	ETL processing timestamp

Dimension Tables

Dimension tables store the descriptive attributes of the business entities.

- **Distribution Strategy:** KEY distribution is used on the surrogate key (e.g., customer_sk). This co-locates related data, ensuring that rows from the fact and dimension tables that are frequently joined together are stored on the same node, which dramatically improves JOIN performance.
- **SCD Type 2 Implementation:** Fields like start_date, end_date, and is_current are used to track the history of changes to dimensional attributes.

dim_customer Fields:

Field	Data Type	Constraints	Business Purpose
customer_sk	bigint	PK, NN	Surrogate key
customer_id	character varying(50)	NN	Business key
customer_name	character varying(100)	NULL	Customer identification
customer_email	character varying(100)	NULL	Contact information
customer_email_domain	character varying(50)	NULL	Email domain analysis
customer_tenure_days	integer	NULL	Account age calculation
customer_address_change_days	integer	NULL	Address stability indicator
start_date	timestamp	NULL	Effective date
end_date	timestamp	NULL	Expiration date
is_current	boolean	NULL	Current record indicator

dim_card Fields:

Field	Data Type	Constraints	Business Purpose
card_sk	bigint	PK, NN	Surrogate key
card_number_hash	character varying(100)	NN	Tokenized card identifier
card_type	character varying(20)	NULL	Card category classification
issuing_bank_name	character varying(100)	NULL	Issuer identification
start_date	timestamp	NULL	Effective date
end_date	timestamp	NULL	Expiration date
is_current	boolean	NULL	Current record indicator

dim_device Fields:

Field	Data Type	Constraints	Business Purpose
device_sk	bigint	PK, NN	Surrogate key
device_id	character varying(50)	NN	Device fingerprint identifier
device_os	character varying(20)	NULL	Operating system detection
user_agent	character varying(500)	NULL	Browser/app identification
start_date	timestamp	NULL	Effective date
end_date	timestamp	NULL	Expiration date
is_current	boolean	NULL	Current record indicator

dim_merchant Fields:

Field	Data Type	Constraints	Business Purpose
merchant_sk	bigint	PK, NN	Surrogate key
merchant_id	character varying(65)	NN	Business identifier
merchant_name	character varying(200)	NULL	Merchant identification
merchant_category	character varying(100)	NULL	Industry classification
merchant_latitude	double precision	NULL	Geographic location
merchant_longitude	double precision	NULL	Geographic location
start_date	timestamp	NULL	SCD effective date
end_date	timestamp	NULL	SCD expiration date
is_current	boolean	NULL	Current record indicator

This robust data warehouse architecture provides a foundation for both the machine learning pipeline and the analytical queries from the QuickSight dashboard.

---

10. Machine Learning Model: XGBoost for Fraud Detection

The core intelligence of the fraud detection system lies in its machine learning model. This section details the development, training, and deployment of the model used to identify fraudulent transactions.

10.1. Model Selection and Rationale

A supervised classification approach was adopted, and the XGBoost (Extreme Gradient Boosting) algorithm was selected for this task.

During the development phase, multiple machine learning algorithms were evaluated, including **Random Forest** and **Gradient Boosting**. However, **XGBoost consistently outperformed the others** in terms of F1 score and generalization performance, especially on the imbalanced dataset.

XGBoost is a powerful and popular tree-based ensemble algorithm, and it was chosen for several key reasons:

- **High Performance**: It consistently achieves state-of-the-art results on tabular data, which is the format of our transaction dataset.
- **Scalability**: XGBoost is designed for efficiency and can be parallelized, making it suitable for training on large datasets.
- **Handling of Imbalanced Data**: Financial fraud datasets are inherently imbalanced (fraudulent transactions are rare). XGBoost provides a parameter (`scale_pos_weight`) specifically designed to handle this, which gives more weight to the minority class during training.
- **Regularization**: It has built-in L1 and L2 regularization to prevent overfitting, which leads to better generalization on unseen data.
- **Feature Importance**: XGBoost can provide a ranked list of feature importances, which helps in understanding which factors are most predictive of fraud.

10.2. Feature Engineering

To enhance the model's predictive power, several new features were engineered from the raw data. These features are designed to capture more complex behavioral, transactional, and spatial patterns.

- **txn_per_hour:** Calculated from time_since_last_txn_sec, this feature estimates the transaction frequency. A sudden spike in this value can be a strong indicator of a velocity attack.
- **is_online:** A binary feature derived from card_entry_method to flag whether a transaction was conducted online. Online transactions are generally higher risk.
- **cvv_match_result (Binary):** The categorical cvv_match_result was converted into a binary flag (1 for match, 0 for mismatch). A CVV mismatch is a very strong fraud signal.
- **amount_binned:** The transaction amount was binned into several categories. This helps the model capture non-linear relationships between the transaction amount and the likelihood of fraud.
- **distance_to_merchant:** The Haversine distance was calculated between the customer's home location and the merchant's location. A large distance for a physical transaction can be suspicious.
- **is_far_txn:** A binary flag that is set to 1 if the distance_from_last_txn_km exceeds a certain threshold

(e.g., 300 km), indicating potentially impossible travel.

All engineered features were log-transformed where appropriate to reduce skew and were standardized using StandardScaler to normalize their ranges.

10.3. Model Training and Evaluation

The model was trained and evaluated using a rigorous process:

1. **Data Splitting:** The dataset was split into a training set and a holdout test set.
2. **Handling Class Imbalance:** The scale_pos_weight hyperparameter in the XGBoost classifier was set to a value calculated from the ratio of negative to positive class instances. This ensures the model pays more attention to the rare fraud cases.
3. **Cross-Validation:** Stratified K-Fold Cross-Validation (with k=5) was used on the training data to tune hyperparameters and get a robust estimate of the model's performance. The F1-score was used as the primary evaluation metric because it provides a balance between precision and recall, which is critical for imbalanced classification problems.
4. **Final Evaluation:** After training on the full training set, the model was evaluated on the unseen test set.

Evaluation Results:

- **Cross-Validation Average F1 Score:** 0.9417
- **Test Set Performance:**
  - **Accuracy:** 97.91%
  - **Precision:** 84.12% (Of all transactions flagged as fraud, 84.12% were actually fraud).
  - **Recall:** 97.50% (The model successfully identified 97.50% of all actual fraud).
  - **F1 Score:** 90.32%
  - **AUC (Area Under ROC Curve):** 1.00

These results indicate an extremely high-performing model. The high recall is particularly important for a fraud detection system, as the primary goal is to catch as much fraud as possible. The strong precision indicates that the model does not generate an excessive number of false positives, which is important for minimizing customer friction.

10.4. Model Deployment and Inference

The trained XGBoost model was saved as a .pkl file and stored in an S3 bucket. The model is deployed for batch inference within a dedicated **AWS Glue job**.

Inference Pipeline Workflow:

1. **Load Model:** The Glue job starts by downloading the xgb_fraud_model.pkl file from S3 into its local environment.
2. **Fetch New Data:** It connects to the Redshift data warehouse and fetches the latest batch of cleaned transaction data.
3. **Apply Transformations:** The script applies the *exact same* feature engineering, encoding, and scaling steps that were used during the training phase. This is crucial to ensure consistency and

prevent training-serving skew.

4. **Predict:** The preprocessed features are fed into the loaded XGBoost model, which predicts a fraud label (is_fraud_pred) for each transaction.
5. **Store Predictions:** The transaction IDs and their corresponding predictions are written to a temporary CSV file in S3.
6. **Load to Redshift:** The Redshift COPY command is used to efficiently load the predictions from the S3 CSV file into a dedicated prediction table (predicted_fraud) in the data warehouse.

This serverless deployment architecture allows the model to be applied to large volumes of data in a scalable and cost-effective manner, without the need to manage dedicated inference servers.

---

11. Alerting and Automation

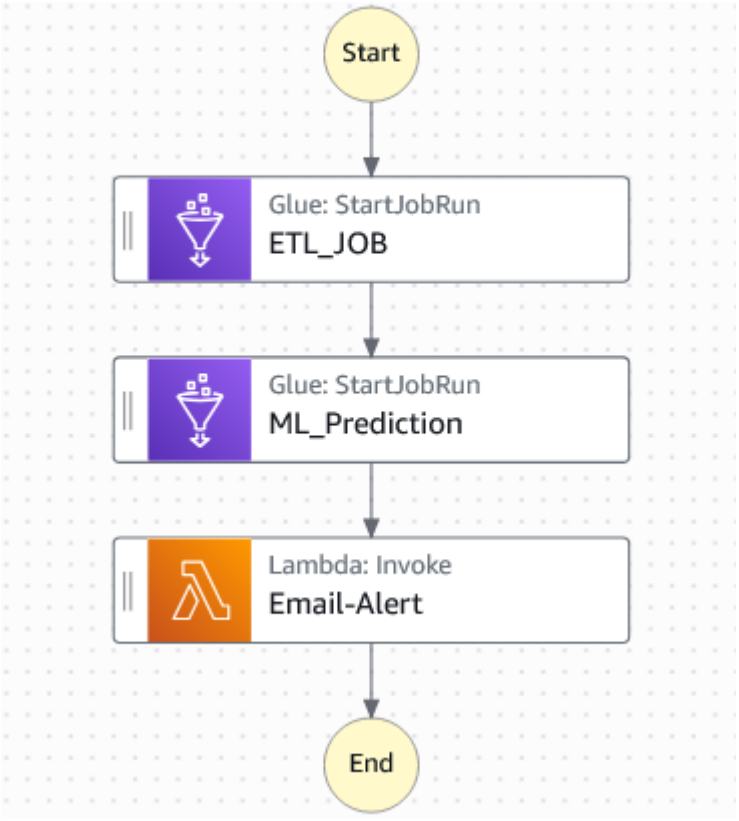
The system automatically processes transaction data, applies machine learning models for fraud scoring, and delivers immediate notifications for high-risk transactions.

The system follows a sequential processing pipeline triggered by S3 object creation events:

1. **Event Trigger**: New transaction data uploaded to S3 triggers EventBridge
2. **Data Processing**: ETL pipeline transforms and loads data into Redshift
3. **ML Scoring**: Pre-trained fraud detection model scores transactions
4. **Alert Generation**: High-risk transactions trigger immediate email notifications

The core of the architecture is a high-frequency polling loop, managed by AWS Step Functions, that triggers an AWS Lambda function. This function is responsible for identifying new fraudulent transactions, dispatching formatted HTML email alerts via Amazon SES, and updating the database to prevent duplicate notifications.

The design prioritizes security, reliability, and scalability by leveraging managed AWS services and adhering to the principle of least privilege. The system is composed of several key AWS services working in concert. Each component has a specific role in the overall workflow.



11.2. AWS Lambda

The sendFraudAlertEmail Lambda function contains all the business logic for the system. It is a serverless, event-driven compute function that runs for a maximum of 30 seconds per invocation.

It is configured with environment variables that provide it with the necessary operational context, such as the Redshift cluster details, the database name, the sender email address, and the location of the database credentials in AWS Secrets Manager.

11.3. Amazon Redshift

The system interacts with a single table named predicted_fraud within the dev database of the Redshift data warehouse. This table serves as the source of truth for fraudulent activity. The Lambda function interacts with the following key columns:

- **transaction_id:** The unique identifier for each transaction.
- **is_fraud_pred:** A flag indicating if the transaction is suspected fraud (a value of 1).
- **email:** The customer's email address, used as the destination for the alert.
- **email_sent:** A boolean flag that tracks whether an alert has been dispatched. It defaults to false and is the key to preventing duplicate alerts.

11.4. Amazon SES (Simple Email Service)

To ensure high deliverability and professional formatting, the system uses Amazon SES to send email alerts. The Lambda function constructs and sends HTML-formatted emails to customers identified in the fraudulent transaction records.

Critical Note: The system's ability to send emails is initially restricted by the SES "Sandbox" environment, which requires all recipient email addresses to be pre-verified. To operate in a production environment and send emails to any customer, a request for Production Access must be approved for the SES account.

11.5. AWS Secrets Manager

To maintain a high standard of security, database credentials are never hardcoded into the application code. Instead, they are stored securely in Secrets Manager under the name dev/redshift/awsuser_creds.

The Lambda function is granted specific permission to retrieve these credentials at runtime, ensuring they are never exposed.

11.6. AWS IAM

Security and access control are managed through AWS Identity and Access Management (IAM), following the principle of least privilege. Two distinct roles are used:

- **Lambda Execution Role (FraudAlertLambdaRole):** This role is assigned to the Lambda function. It grants the function precise permissions to perform its duties: writing logs to CloudWatch, accessing the Redshift Data API, sending emails via SES, and retrieving the secret from Secrets Manager.
- **Step Functions Execution Role:** This role is assigned to the state machine. It has a very narrow scope of permissions, granting it only the ability to invoke the target sendFraudAlertEmail Lambda function.

11.7. Core Processing Workflow

The Lambda function executes the following sequence of steps each time it is triggered by the Step Functions loop:

1. **Initiation:** The TenSecondFraudChecker state machine invokes the Lambda function.
2. **Query for New Fraud:** The function connects to the Redshift cluster using the Redshift Data API. It

executes a query to find up to 10 records in the predicted_fraud table that meet two criteria: the is_fraud_pred flag is set to 1 AND the email_sent flag is false. The batch size is limited to 10 to control the workload of a single invocation.

3. **Check for Results:**
    - If the query returns no records, it means there are no new fraudulent transactions to process. The function logs this and exits successfully.
    - If records are found, the function proceeds to the next step.
4. **Process and Alert:** The function iterates through each record retrieved from the database. For each transaction:
    - It extracts the transaction_id and the customer's email.
    - It composes a standardized, HTML-formatted security alert email, embedding the unique transaction ID.
    - It uses Amazon SES to send this email to the customer's address.
5. **Prevent Duplicate Alerts:** After iterating through all records and sending the corresponding emails, the function executes a final UPDATE query against the Redshift database. This query sets the email_sent flag to true for all the transaction IDs that were just processed, ensuring they will not be picked up in the next polling cycle.
6. **Error Handling:** If any step in this process fails (e.g., the database is unavailable or an email fails to send), the function will raise an error. This error is caught by the parent Step Functions state machine, which immediately halts the entire 10-second loop, preventing further executions until the issue is investigated and resolved.

11.8. System Activation and Deactivation

- **Activation:** The system is started by manually beginning a new execution of the TenSecondFraudChecker state machine from the AWS Step Functions console. Once started, the loop will run indefinitely.
- **Deactivation:** The primary method for stopping the system and all associated polling activity is to manually **Stop** the running execution from the Step Functions console. This is the official procedure for halting operations and controlling costs.

11.9. Monitoring and Logging

- **Execution Logs:** Detailed logs from every Lambda invocation, including status messages, query IDs, and any errors, are available in the function's dedicated Amazon CloudWatch Log Group.
- **Execution Health:** The Step Functions console provides a visual graph of the execution history, making it easy to see if the loop is running successfully or if it has entered a failed state.

11.10. Recommended Production Alerting

For a production environment, it is highly recommended to configure Amazon CloudWatch Alarms to automatically notify administrators of system failures. Alarms should be set on the following metrics:

- **Lambda Function Errors:** To be alerted if the core processing logic fails.
- **Step Functions ExecutionsFailed:** To be alerted if the entire orchestration loop halts due to an error.

11.11. Key System Characteristics

- **Security:** The principle of least privilege is applied. The Lambda role has specific permissions, and

database credentials are never exposed in code, relying instead on AWS Secrets Manager.

- **Scalability & Performance:** The serverless nature of Lambda and Step Functions allows the system to scale automatically. The use of `LIMIT 10` in the SQL query acts as a simple batching mechanism to control the workload per invocation.
- **Reliability:** The Step Function's built-in `Catch` block provides robust error handling for the entire workflow. If the Lambda fails for any reason, the loop will stop and enter a `Fail` state, preventing runaway executions.
- **Critical Dependency (SES Sandbox):** The system's ability to send email is constrained by the SES Sandbox limits. For production use, requesting Production Access for SES is mandatory to increase sending quotas and improve email deliverability.

---

12. AWS Services Implementation Details

This section provides more granular detail on the configuration of the key AWS services that form the infrastructure of the project.

12.1. Networking: VPC, Subnets, and Security Groups

Security is paramount. The entire infrastructure is deployed within a custom **Virtual Private Cloud (VPC) to pro**vide logical isolation.

- **VPC:** A VPC with a CIDR block of 10.0.0.0/16 is created, named Fraud-Detection-Project.
- **Subnets:**
  - **Public Subnet:** A public subnet (10.0.1.0/24) is created to host the EC2 instance. It is "public" because it has a route to an Internet Gateway, allowing it to be accessed from the internet (e.g., via SSH) and to access other AWS services.
  - **Private Subnets:** The Amazon Redshift cluster is deployed across multiple private subnets. These subnets do not have a direct route to the Internet Gateway, meaning the database is not exposed to the public internet, which is a critical security best practice.
- **Security Groups:** These act as stateful firewalls for the resources.
  - **EC2 Security Group:** Allows inbound SSH traffic (port 22) from a trusted IP address for management and allows all outbound traffic.
  - **Redshift Security Group:** This is more restrictive. It only allows inbound traffic on the Redshift port (5439) from specific sources: the security group of the Glue service and the security group used by QuickSight. All other inbound traffic is denied.
  - **QuickSight Security Group:** To allow QuickSight (a public service) to connect to the private Redshift cluster, a dedicated security group is created for the QuickSight network interface. An outbound rule allows traffic to the Redshift security group on port 5439, and a corresponding inbound rule is added to the Redshift security group to allow this traffic.

12.2. Compute: EC2 Configuration

- **Instance Type:** A t3.medium instance is chosen to provide a balance of CPU and memory sufficient for running the 3-broker Kafka cluster and the Python scripts without performance issues.

- **AMI:** A standard Amazon Linux 2 AMI is used.
- **Elastic IP:** An Elastic IP address is attached to the instance. This provides a static, public IP address that does not change if the instance is stopped and restarted, which is important for stable DNS resolution and access.
- **IAM Role:** An IAM role (kinesis-access) is attached to the EC2 instance. This role grants the instance permissions to perform actions on other AWS services, specifically kinesis:PutRecord. This is a secure way to grant permissions without hard-coding AWS credentials into the consumer script.

12.3. Storage: S3 Bucket Strategy

- **Bucket Name:** streaming-data-central
- **Purpose:** This bucket serves as the central data lake.
- **Prefix Structure:** Kinesis Firehose is configured to partition the incoming data by arrival time using the prefix format YYYY/MM/DD/HH/. This is a highly efficient structure for time-series data, as it allows query engines like AWS Glue and Amazon Athena to perform "partition pruning," scanning only the relevant folders for a given time range, which dramatically reduces query time and cost.
- **Model Storage:** A separate folder within the same bucket (or a different bucket entirely, e.g., fraud-model) is used to store the trained xgb_fraud_model.pkl file.

12.4. Streaming: Kinesis Setup

- **Kinesis Data Stream:** Named kinesis_data_stream, configured with a single shard initially. The number of shards can be easily scaled up or down as the data volume changes.
- **Kinesis Data Firehose:** Named kinesis_to_s3, this delivery stream is configured with the Kinesis Data Stream as its source and the S3 bucket as its destination. It is also configured with buffering hints (e.g., deliver every 5 minutes or when 5MB of data is collected) to create optimally sized files in S3.

12.5. Data Warehouse: Redshift Cluster Configuration

- **Node Type:** A small, cost-effective node type (e.g., dc2.large) is used for this project.
- **Cluster Size:** A 2-node cluster is sufficient for the project's data volume.
- **VPC Deployment:** The cluster is deployed into the private subnets of the VPC for security.
- **IAM Role for S3 Access:** An IAM role (Redshift-S3-Access-Role) is created and attached to the Redshift cluster. This role grants Redshift permissions to execute the COPY command from the S3 bucket where the Glue job places the prediction files.

12.6. Serverless: Lambda, Step Functions, and IAM Roles

- **IAM Roles:** The principle of least privilege is strictly followed. Each serverless component has its own dedicated IAM role with only the permissions it needs to function.
  - **Lambda Execution Role:** Grants permissions to read from Redshift and publish to the specific SNS topic.
  - **Step Functions Execution Role:** Grants permissions to invoke Glue jobs and Lambda functions.
  - **Glue Service Role:** Grants permissions to read/write from the S3 data lake and the Redshift cluster.
- **Lambda Configuration:** The function is configured with a reasonable memory allocation (e.g., 256MB) and a timeout (e.g., 30 seconds). Environment variables are used to store configuration details

like the Redshift cluster endpoint and the SNS topic ARN.

- **Step Functions Definition:** The state machine is defined using the Amazon States Language (a JSON-based structured language) to specify each state, its properties, and the transitions between them.

---

13. Results and Evaluation

The success of the fraud detection system is measured by the performance of its machine learning model and the overall effectiveness and usability of the pipeline and its outputs.

13.1. Model Performance Metrics

As detailed in Section 10, the XGBoost model demonstrated exceptional performance on the holdout test set.

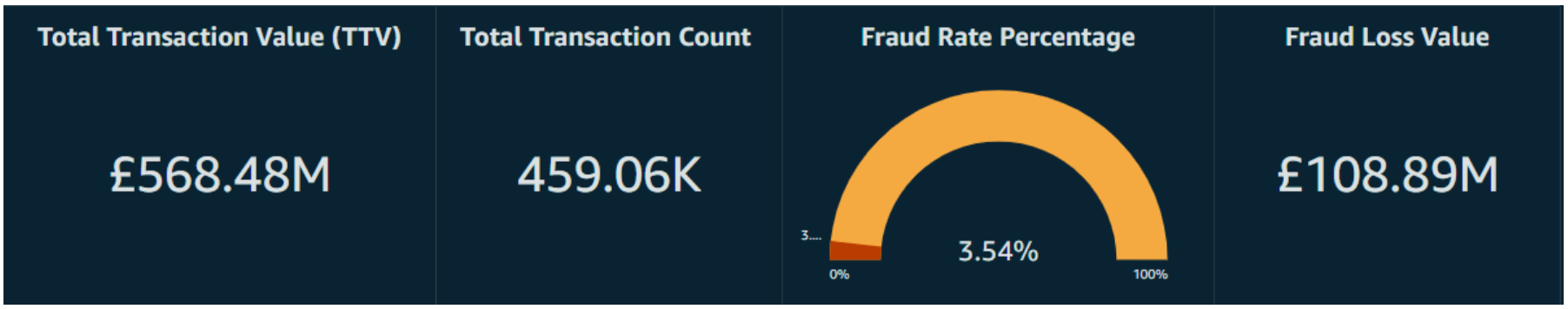
- **F1 Score (90.32%):** This high F1 score indicates a strong balance between identifying fraud and avoiding false alarms, which is the ideal outcome for a fraud detection model.
- **Recall (97.50%):** The model is extremely effective at its primary job: catching fraudulent transactions. It successfully identified 97.5% of all fraud cases in the test data, minimizing potential financial losses.
- **Precision (84.12%):** While the recall is very high, the precision is also strong. This means that when the model flags a transaction, it is correct over 84% of the time. This is crucial for maintaining a positive customer experience, as it minimizes the number of legitimate transactions that are incorrectly blocked or challenged (false positives).
- **AUC (1.00):** An Area Under the ROC Curve of 1.00 indicates perfect separation between the positive and negative classes in the test set, highlighting the model's outstanding discriminatory power.

13.2. Fraud Monitoring Dashboard (QuickSight)

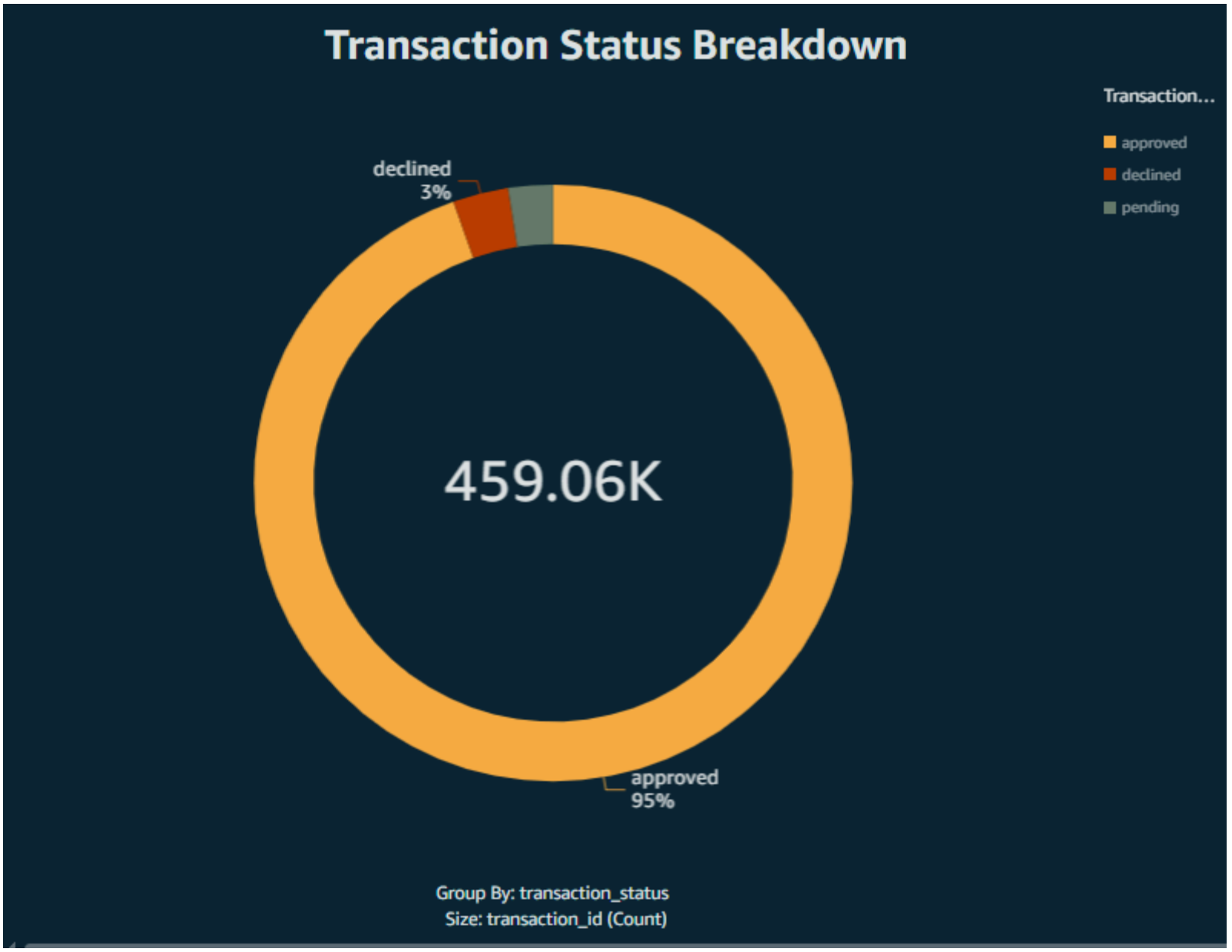
The Amazon QuickSight dashboard serves as the primary interface for human analysts to interact with the system's results. It provides a comprehensive, at-a-glance view of transaction health and fraud patterns.

Key Visualizations and Their Interpretation:

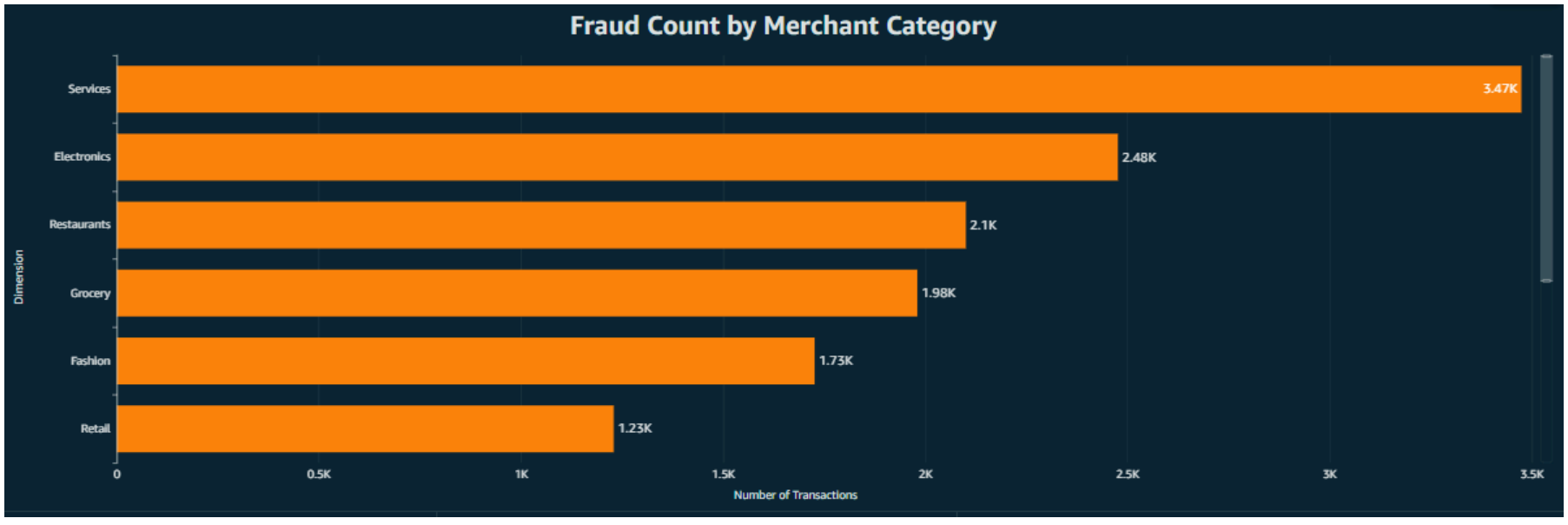
- **Headline KPIs:** The top banner displays the most critical metrics for the selected time period:
  - **Total Transaction Value:** Measures overall business volume.
  - **Total Transaction Count:** Measures platform activity.
  - **Fraud Rate Percentage:** The key risk indicator. A sudden spike here would be an immediate cause for investigation.
  - **Fraud Loss Value:** Quantifies the direct financial impact of fraud.



- **Transaction Status Breakdown (Donut Chart):** This visualizes the approval rate. A high decline rate could indicate overly strict rules or issues with a particular payment processor, leading to lost revenue.

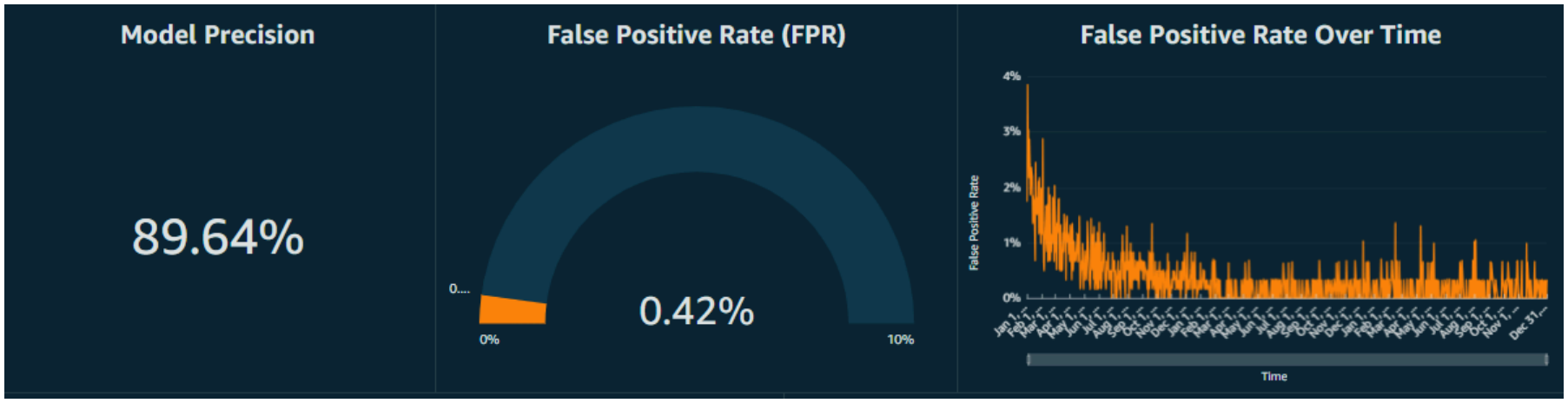


- **Fraud Count by Dimension (Pivotable Bar Chart):** This is a powerful investigative tool. An analyst can dynamically pivot the view to see fraud counts by **Merchant Category**, **Card Type**, **Device OS**, or **Issuing Bank**. For example, if they see a sudden spike in fraud coming from the "Electronics" merchant category, they can drill down to investigate specific merchants.

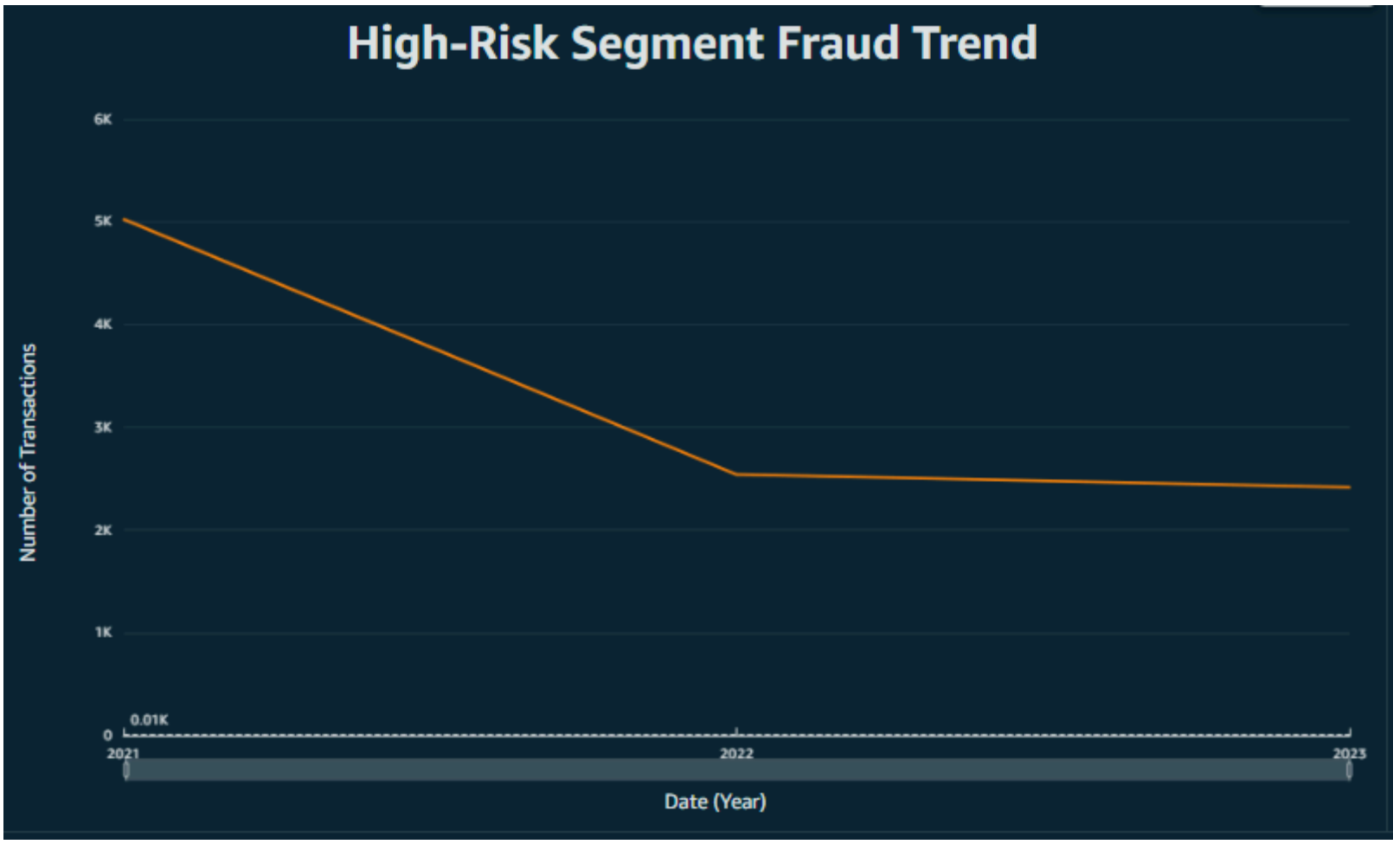


- **Model Performance Visuals:**
  - **Model Precision:** Tracks how accurate the model's fraud predictions are. A declining trend in precision would signal that the model may need to be recalibrated.
  - **False Positive Rate (FPR) Over Time:** A rising FPR means more legitimate customers are being

inconvenienced, and the fraud rules or model may be too aggressive.



- **Trend Analysis (Line Chart):** This chart tracks the volume of fraud over time within specific high-risk segments (e.g., transactions from a VPN, first-time interactions). This helps analysts understand if fraudsters are shifting their tactics to target different areas.



13.3. System Performance

- **Latency:** The end-to-end latency of the pipeline, from a transaction being produced by the Kafka producer to it being scored and appearing in the QuickSight dashboard, is designed to be in the range of a few minutes. This is primarily determined by the batching intervals set in Kinesis Firehose and the schedule of the Glue jobs. While not instantaneous (sub-second), this near real-time performance is a massive improvement over traditional batch systems and is sufficient to enable rapid investigation and response.
- **Scalability:** The use of managed AWS services like Kinesis, Glue, and Lambda ensures that the system can scale automatically to handle significant increases in transaction volume without manual intervention.
- **Reliability:** The orchestration with AWS Step Functions, which includes built-in error handling and retry logic, makes the pipeline highly resilient to transient failures in individual components.

14. Challenges and Limitations

While the project successfully implements a robust and modern fraud detection system, it is important to acknowledge its challenges and limitations.

14.1. Dataset Limitations

- **Synthetic Data:** The most significant limitation is the reliance on a **synthetically generated dataset**. While the simulator was designed to be highly realistic, it cannot perfectly replicate the infinite complexity, noise, and nuance of real-world transaction data. Real data often contains unexpected formats, missing values, and subtle patterns that are difficult to simulate. Therefore, the model's stellar performance on the synthetic test set may not fully translate to a production environment with real data.
- **Potential for Feature Leakage:** The extremely high performance scores (especially the AUC of 1.00) could suggest a risk of minor feature leakage in the synthetic data generation process. This means some features might inadvertently contain information about the target label that would not be available in a real-world, real-time scenario. For example, if a certain type of fraudulent persona was accidentally always generated with a specific IP address range, the model might learn that simple rule instead of a more complex pattern.
- **Class Imbalance:** Although the fraud rate was adjusted, the dataset is still imbalanced. This can make it challenging for the model to learn the patterns of very rare fraud types that may only have a few examples in the training data.

14.2. Real-Time Processing Constraints

- **Batch-Oriented Components:** The current architecture, while "near real-time," is not truly "streaming" in the sub-second sense. The use of AWS Glue jobs, which have a start-up overhead, and Kinesis Firehose, which batches data, introduces latency measured in minutes. For use cases requiring instantaneous blocking of transactions, a different architecture (e.g., using AWS Lambda or Kinesis Data Analytics for stream processing) would be required.
- **Static Model:** The deployed model is static. It does not automatically retrain or adapt to new fraud patterns over time. In a real-world scenario, fraudsters constantly change their tactics, which can lead to **concept drift**, where the model's performance degrades because the patterns in the live data no longer match the patterns it was trained on.

14.3. Model Generalization

- **Geographic and Contextual Focus:** The model was trained on data specifically designed to be "Egypt-centric." Its performance may not generalize well to transaction data from other countries or different economic contexts without retraining on relevant data.
- **Lack of Explainability:** The current implementation does not include model explainability techniques (e.g., SHAP values). While the XGBoost model can provide feature importances, it doesn't explain *why* a single, specific transaction was flagged as fraudulent. This can be a challenge for fraud analysts who need to justify their decisions and for meeting regulatory requirements.

---

15. Conclusion and Future Work

15.1. Project Summary

This project has successfully designed, implemented, and documented an end-to-end, near real-time fraud detection system on the AWS cloud.

Using a modern data stack that includes Apache Kafka, Amazon Kinesis, AWS Glue, Amazon Redshift, and XGBoost, the system provides a scalable, resilient, and intelligent solution to a critical business problem.

Key achievements of this project include:

- The creation of a sophisticated synthetic data generator that provides a solid foundation for model development.
- The implementation of a robust, multi-layered data pipeline that handles the entire data lifecycle from ingestion to insight.
- The development of a high-performance machine learning model capable of detecting a wide variety of fraud patterns with exceptional accuracy.
- The automation of the entire workflow using AWS Step Functions, creating a hands-off, manageable system.
- The integration of a serverless, real-time alerting mechanism with AWS Lambda and SNS.
- The delivery of actionable insights through an interactive Amazon QuickSight dashboard.

This project serves as a powerful demonstration of how cloud-native services and machine learning can be combined to build advanced analytical systems that are far superior to traditional, on-premises solutions.

15.2. Future Enhancements

The current system provides a strong foundation that can be extended and improved in several key areas to create an even more powerful and production-ready solution.

- **Integrate with a True Real-Time Data Source:** The next logical step is to replace the batch file producer with a direct connection to a real transactional stream, such as a production Kafka topic or a Kinesis stream fed by point-of-sale systems. This would move the system from "near real-time" to "true real-time."
- **Implement a Streaming Inference Architecture:** To achieve sub-second fraud detection for transaction blocking, the batch-based Glue inference job could be replaced or augmented with a streaming-first solution. This could involve using **Kinesis Data Analytics** or **AWS Lambda** to process and score transactions one by one as they arrive in the Kinesis stream.
- **Automated Model Retraining and MLOps:** To combat concept drift, a full MLOps pipeline should be implemented. This would involve:
    - **Model Monitoring:** Using a tool like **Amazon SageMaker Model Monitor** to automatically detect drift in data quality and model performance.
    - **Auto-Retraining:** Creating a scheduled pipeline (using Step Functions or Airflow) that automatically retrains the model on fresh data, evaluates its performance, and, if the new model is better, deploys it to production without manual intervention.

- **Explore Unsupervised Anomaly Detection:** To detect novel, unknown fraud patterns, unsupervised models could be explored. Algorithms like **Isolation Forest** or **Autoencoders** could be trained to identify transactions that deviate significantly from normal behavior, flagging them for investigation even if they don't match any previously seen fraud type.
- **Integrate Model Explainability (XAI):** To provide more transparency and assist fraud analysts, model explainability libraries like **SHAP (SHapley Additive exPlanations)** should be integrated. This would allow the system to provide a reason for each fraud prediction (e.g., "this transaction was flagged due to a combination of impossible travel and a high-risk merchant"), which is critical for audit and investigation.
- **A/B Testing Framework for Models:** Implement a framework to A/B test new models against the current champion model in a production environment. This would allow for the safe and data-driven rollout of model improvements.

Pursuing these future workstreams can help the current system evolve into a adaptive and self-learning fraud detection platform.

16. References

- Amazon Web Services Documentation. (2023). *AWS Documentation*. Retrieved from https://docs.aws.amazon.com/
- Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
- Docker Inc. (2023). *Docker Documentation*. Retrieved from https://docs.docker.com/
- Scikit-learn Development Team. (2023). *scikit-learn: Machine Learning in Python*. Retrieved from https://scikit-learn.org/
- The Apache Software Foundation. (2023). *Kafka Documentation*. Retrieved from https://kafka.apache.org/documentation/
- U.S. Federal Trade Commission. (2024). *Consumer Sentinel Network Data Book 2023*. Retrieved from https://www.ftc.gov/reports/consumer-sentinel-network-data-book-2023
- Verafin. (2024). *2024 Global Financial Crime Report*. Retrieved from https://static.poder360.com.br/2024/03/relatorio-crimes-financeiros-nasdaq-2024.pdf