Case Study 01 - PART 2

# Real-Time Fraud Detection in Banking Transactions



## Designing a Data Pipeline

Objective: Development of a Machine Learning Model for Real-Time Fraud Detection in Banking Transactions, as it enables prompt intervention and mitigation strategies to be deployed in response to potentially malicious or unauthorized activities.

In the rapidly evolving landscape of financial services, the prevalence of fraudulent activities poses a significant threat to the security of banking transactions. To address this challenge, there is a critical need for an advanced solution capable of identifying and mitigating fraudulent activities in real-time. This project aims to design and implement a robust machine-learning model that can accurately predict fraudulent transactions as they occur, thereby ensuring the security of the banking ecosystem.

The core objective is to develop a highly efficient and reliable machine-learning model that processes banking transactions in real time. The model will be trained on a comprehensive dataset of banking transactions, over a wide array of features that are indicative of fraudulent behavior. Upon deployment, the model will scrutinize each transaction in real time, determining the likelihood of it being fraudulent.

### Assumptions:

The banks who opt for our service would be treated as our customers and data from these customers would be accessible to our service while adhering to the Guidance for Protection of PII (Personally Identifiable Information). The banks are assumed to be based out of USA only.

# Data Pipeline:

## Training Data Pipeline:

Data Pipeline Design for One-Time Batch Data Ingestion:

To facilitate the training of machine learning models for fraud detection in banking transactions, a robust and efficient data pipeline is paramount. This pipeline is designed to manage a one-time, bulk ingestion of historical transaction data spanning from 2016 to 2023. The primary focus is on extracting, transforming, and loading this latent data efficiently from the source database to the target environment, tailored to meet the specific requirements of the model training process.

## Data Extraction and Loading:

The Raw Data Load pipeline is optimized for a single, comprehensive transfer of historical data. It leverages the capability of extracting the entire dataset from the source database and loading it directly into the target database without the need for recurrent execution. This approach ensures that the foundational data set is comprehensive and serves as a reliable basis for model training.

## Data Transformation:

Post extraction, the data undergoes a transformation process. This involves refining the dataset to include only the most pertinent features necessary for fraud detection. The targeted features include Transaction ID, User Name, Vendor, Time, Amount, Location, and Payment Type. These features are used after extensive research of existing datasets, and these are the most relevant features needed for the model. The transformation process is designed to extract these key features, thereby ensuring the data is in the most conducive format for model training.

# Technology Stack - Apache NiFi:

To orchestrate this data pipeline, Apache NiFi is the technology of choice due to its robustness and versatility in managing data flows. Apache NiFi is renowned for its capability to construct comprehensive and scalable data routing, transformation, and system mediation logic through directed graphs. Its design ethos, centered around dataflow automation and system-to-system data transfer, makes it an ideal tool for this scenario. It was specifically designed to automate the flow of data between systems. It offers a wide range of database connectors and can handle different types of databases efficiently.



# Database Interaction and Connection Management:

Within Apache NiFi, database interactions are managed by a suite of specialized processors, including but not limited to ExecuteSQL, ExecuteSQLRecord, PutSQL, and QueryDatabaseTable. These processors are adept at performing a plethora of database operations, offering extensive flexibility based on the database type and the specific operation required. The choice of processor depends on the type of database and the operation that is to be performed. For this scenario, PutSQL is ideal as it supports the execution of INSERT, UPDATE, DELETE, etc., SQL statements. A target schema is defined for the data and is stored in the **Avro Schema Registry**. **ConvertRecord** process can be used to transform the data to the desired schema. The processors use **TLS/SSL** to secure the data flow in transit.

Secure and efficient database connectivity is facilitated through the **DBCPConnectionPool** services. This shared service is instrumental in providing database connections to various processors, ensuring a streamlined and secure data flow. It employs a connection URL pattern, which typically includes vital connection parameters such as the database system name, host, port, database name, and additional parameters, fortifying the connectivity and data transfer processes. The processors use TLS/SSL to secure the data flow in transit.

The utilization of a Raw Data Pipeline is a strategic choice, especially in scenarios involving the one-time ingestion of a large historical dataset for purposes such as training a machine learning model. The reason behind employing a raw data pipeline is because of several practical and technical considerations:

**1. Volume and Nature of Data:** The dataset in question spans a significant period (2016 to 2023), encompassing a vast volume of transactions. The raw data pipeline is apt for bulk data transfers, where the data is ingested in its entirety.

**2. Efficiency in Data Transfer:** Raw data pipelines are designed to handle large batches of data efficiently. By transferring the data in a single, consolidated operation, the system minimizes the overheads associated with multiple, smaller data transfers.

**3. Cost-Effectiveness:** For a one-time data load, a raw data pipeline is often more cost-effective compared to setting up a more complex pipeline architecture. It eliminates the need for continuous monitoring and maintenance associated with real-time data pipelines, thereby reducing operational costs.

The raw data for each customer (bank) is stored in a separate database and separate machine-learning models are trained for each customer. This is done to adhere to the custom standards expected by the customers and to increase efficiency. However, the cost-effectiveness of this system must be considered as well. Another option would be to store the data in data lakes which preserve each batch of customer data separately but in a single data storage solution.

# Real-Time Data Extraction and Streaming for Fraud Detection:

In the domain of financial transactions, the urgency for immediate detection and response to fraudulent activities warrants a real-time data processing solution. To this end, a Change Data Capture (CDC) pipeline is employed, leveraging its capability to capture and relay database changes instantaneously. The objective is to minimize latency in the data extraction process, ensuring prompt and accurate fraud detection and decision-making.

## CDC Pipeline Utilization:

**1. CDC Pipeline Motivation:**

- The banks' existing database systems are robust and operational.

- Real-time changes in the database are crucial to be fetched and relayed to the consumer system without delay.

- CDC is preferred for its non-intrusive nature and minimal performance impact on the source database.

**2. Debezium for Database Monitoring:**

- Functionality: Debezium excels in converting databases into event streams, allowing applications to respond to row-level changes with minimal latency.

- Database Support: It supports a variety of databases including MySQL, PostgreSQL, and MongoDB, among others.

- Operation Methodology: Debezium operates by monitoring the database's log files (e.g., binlog in MySQL, WAL in PostgreSQL), ensuring minimal load on the source database.

- Integration with Kafka: Debezium works seamlessly with Apache Kafka, providing scalability, fault tolerance, and the ability to resume from the last point of interruption.

## Kafka Cluster Configuration:

1. Kafka Cluster Setup:

- Initiate a Kafka cluster including Kafka brokers, Zookeeper, and the creation of necessary Kafka topics.

2. Debezium Configuration:

- Connector Configuration: Configure the Debezium connector specific to the source database. This includes specifying the database credentials, hostname, port, and the tables to be monitored.

- Deployment: Deploy the connector configuration to Kafka Connect using the REST API.

## Apache Kafka as a Distributed Streaming Platform:

1. Publish-Subscribe Mechanism:

- Publish: Debezium captures and publishes each transaction as a message to a Kafka topic.

- Subscribe: Fraud detection systems and real-time analytics platforms subscribe to these topics and process the incoming messages.

2. Performance and Scalability:

- Kafka is adept at handling high data volumes, crucial for processing the vast number of transactions occurring in the banking sector.

3. Stream Processing:

- Kafka can perform real-time stream processing, including data transformation and filtering.

## Data Serialization for Efficient Transmission:

1. Necessity of Serialization:

- Systems like Apache Kafka handle data as byte arrays, necessitating the conversion of data into a serialized format for efficient transmission and storage.

2. Avro Serialization:

- Schema Inclusion: Avro serialization includes the schema along with the data, facilitating schema evolution and ensuring compatibility between producers and consumers.

- Efficiency: Avro's serialization reduces message size, optimizing bandwidth usage and transmission speed.

3. Serialization Libraries:

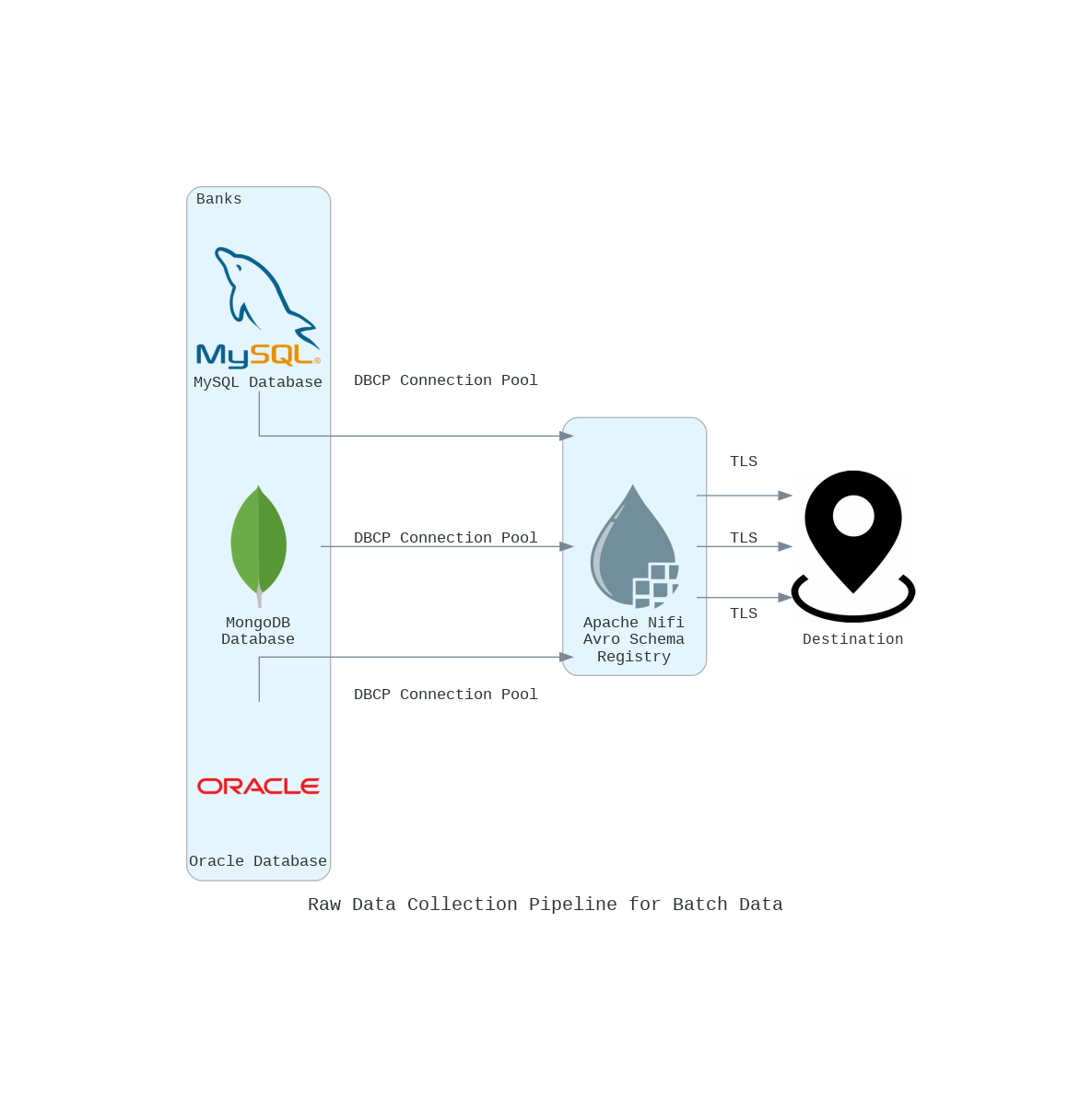
- Utilize libraries like Apache Avro for serialization, ensuring that data is efficiently packaged for streaming.

By leveraging a CDC pipeline with Debezium and Apache Kafka, coupled with efficient data serialization through Avro, the system ensures real-time monitoring and processing of banking transactions. This infrastructure is pivotal for immediate fraud detection, maintaining data integrity, and ensuring the seamless operation of real-time analytics in the banking sector. The strategic configuration and integration of these technologies underpin a robust, scalable, and efficient data processing pipeline, crucial for mitigating fraudulent transactions in the banking industry.

Different bank data streams will be a topic which the consumer can subscribe to. This data will be fed to the models in real time for deducing a score which determines if a transaction is fraudulent or not. The transaction then can be deemed complete or terminated based on this score.

# Data Pipeline Diagram

## Raw Data Collection Pipeline for Batch Data:



## CDC Pipeline for Real-Time Data Collection:

