Scripts Execution

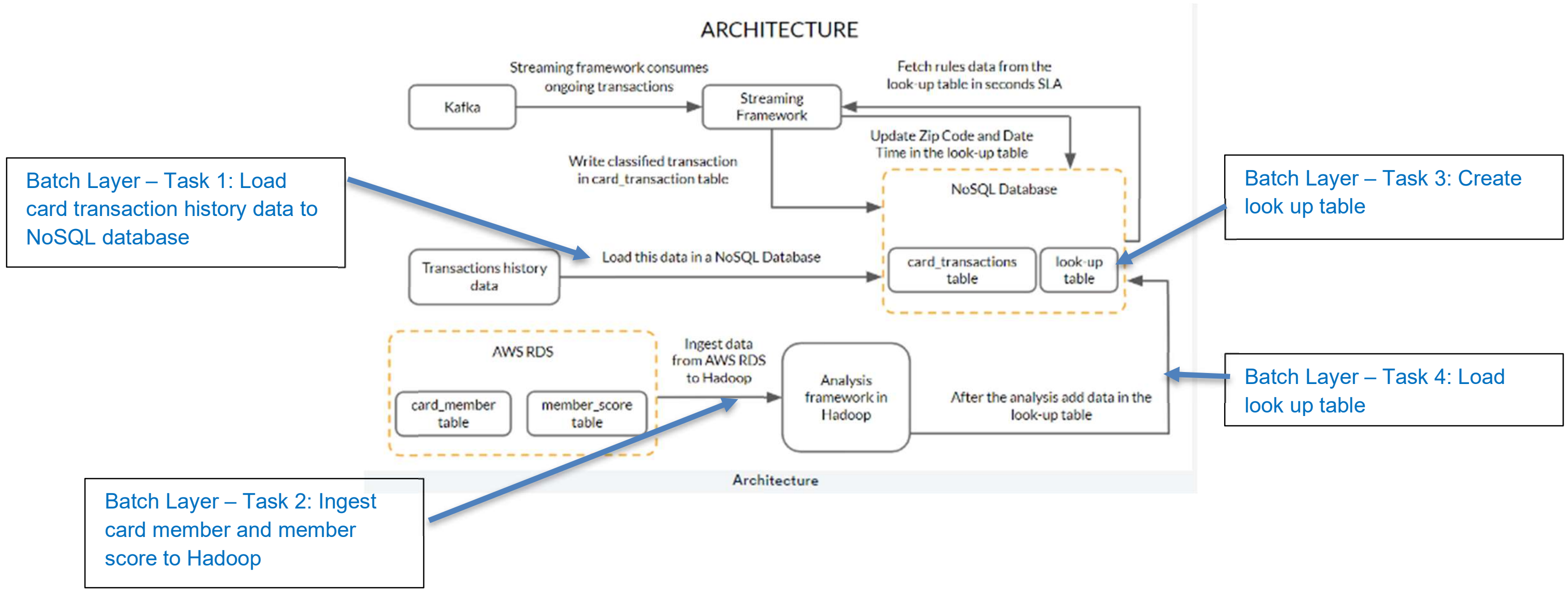
Explanation of the solution to the batch layer problem

Purpose of document:
This pdf document is intent to provide explanation of the solution to the batch layer problem in detail should be provided properly in a document.

Problem Statement:
Credit card fraud is defined as a form of identity theft in which an individual uses someone else's credit card information to make purchases or to withdraw funds from the account. The incidence of such fraudulent transactions has skyrocketed as the world has moved towards a digital era.
With the rising number of fraud cases, the company's major focus is to provide its customers with a delightful experience while ensuring that security is not compromised.
As a big data engineers, we need to provide an architecture and build a solution to cater to the following requirements:

- Fraud detection solution is to detect fraudulent transactions, wherein once a cardmember swipes their card for payment, the transaction is classified as fraudulent or authentic based on a set of predefined rules.
- If fraud is detected, then the transaction must be declined.
- Please note that incorrectly classifying a transaction as fraudulent will incur huge losses to the company and also provoke negative consumer sentiment.
- Customer information: Our solution to batch layer problem should address to continuously update the relevant information about the customers on a platform from where the customer support team can retrieve relevant information in real-time to resolve customer complaints and queries.

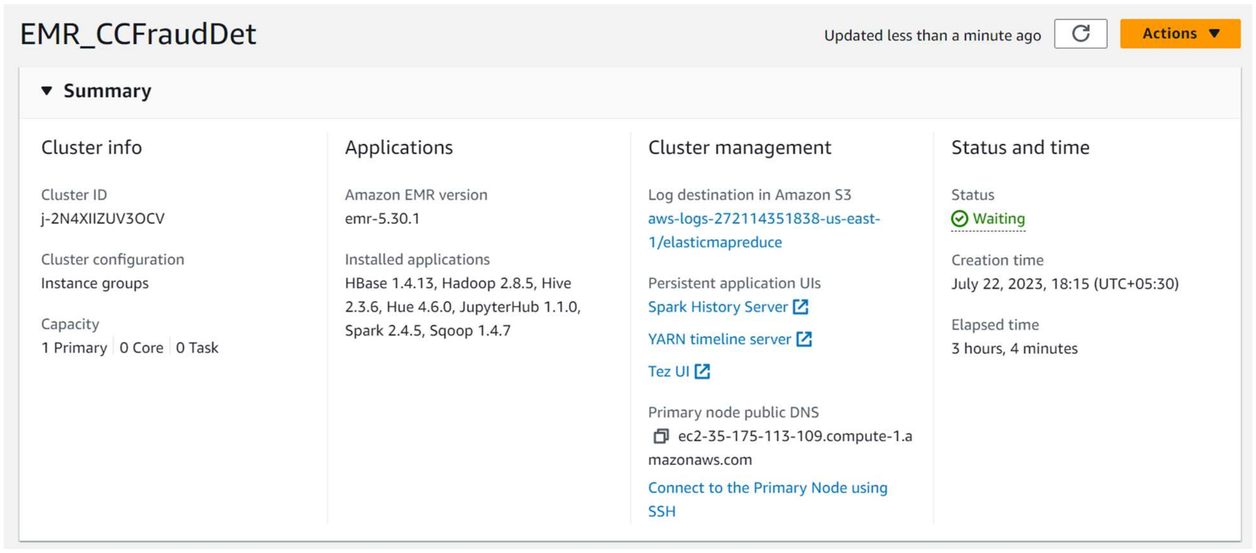
Solution Architecture:


ARCHITECTURE
Streaming framework consumes ongoing transactions
Fetch rules data from the look-up table in seconds SLA
Kafka
Streaming Framework
Update Zip Code and Date Time in the look-up table
Write classified transaction in card_transaction table
NoSQL Database
Transactions history data
Load this data in a NoSQL Database
card_transactions table
look-up table
AWS RDS
card_member table
member_score table
Ingest data from AWS RDS to Hadoop
Analysis framework in Hadoop
After the analysis add data in the look-up table
Architecture
Batch Layer – Task 1: Load card transaction history data to NoSQL database
Batch Layer – Task 2: Ingest card member and member score to Hadoop
Batch Layer – Task 3: Create look up table
Batch Layer – Task 4: Load look up table

Properly explain the step by step process followed for the completion of tasks till **task 4.** The steps should be properly documented here.

Step by step process followed for this (till Task 4):

Step 1 - Creation of EMR Cluster:


EMR_CCFraudDet

Updated less than a minute ago

Summary

Cluster info

Cluster ID
j-2N4XIIZUV3OCV

Cluster configuration
Instance groups

Capacity
1 Primary | 0 Core | 0 Task

Applications

Amazon EMR version
emr-5.30.1

Installed applications
HBase 1.4.13, Hadoop 2.8.5, Hive 2.3.6, Hue 4.6.0, JupyterHub 1.1.0, Spark 2.4.5, Sqoop 1.4.7

Cluster management

Log destination in Amazon S3
aws-logs-272114351838-us-east-1/elasticmapreduce

Persistent application UIs
Spark History Server
YARN timeline server
Tez UI

Primary node public DNS
ec2-35-175-113-109.compute-1.amazonaws.com

Connect to the Primary Node using SSH

Status and time

Status
Waiting

Creation time
July 22, 2023, 18:15 (UTC+05:30)

Elapsed time
3 hours, 4 minutes

Create EMR Cluster with necessary applications like Hbase, Hadoop, Hive, Hue, Spark and Sqoop

- 1 primary with EBS storage chosen as 20 GB
- EMR Version: 5.30.1
- Inbound rules verified for SSH connection.

Step 2: Creation of card transactions table:

- Card_transaction table with 'TD' as column family
- Creation of hive integrated card_transactions hbase table.

Step 3: Loading data into card transactions

(Task 1: Load the transactions history data (card_transactions.csv) in a NoSQL database.)

- Creation of staging table for card transactions.
- Load the raw data from csv file card transaction staging table.
- Load final card transaction table into hive integrated Hbase card_transactions table

Step 4: Populate Card_member and member_score table from RDS

(Task 2: Ingest the relevant data from AWS RDS to Hadoop.)

- Sqoop command for loading card_member and member_score data from RDS to HDFS
- Create card_member and member_score table in hive and load the data from HDFS

Step 5: Creation of look up table:

(Task 3: Create a look-up table with columns specified earlier in the problem statement.)

- Create in hbase - look_up_table with column families ‘card_details’, 'Member_details', 'Location', 'Rule_params'
- Creation of hive integrated look_up_table hbase table.

Step 6: Populate Lookup table :

(Task 4: After creating the table, you need to load the relevant data in the lookup table.)

- Create a view which has ranking based on transaction date and amount in descending order for each card for genuine transactions.
- Join the latest transactions for each card with card_member and member_score table to get following:
    - **UCL (Upper Client Limit)** – derive this value for each card. Average + 3 * standard deviation for average amount taken from latest 10 genuine transactions for each card holder. For those cards which has less then 10 transactions average is calculated from < 10 transaction which ever available.
    - **Postal code** is derived from latest transaction
    - **Member_score** is derived from member_score table
- These information will help us to classify the transactions whether Genuine or Fraud.

Capstone project – Credit Card Fraud Detection (mid submission)

by

Susil Patro, Krishna Mohan & Ashmeet Singh Deol