Final-Submission – Logic Explanation

Explanation of the solution to the streaming layer problem

*****Please zoom to 180% or 200% to see screenshots with better clarity**

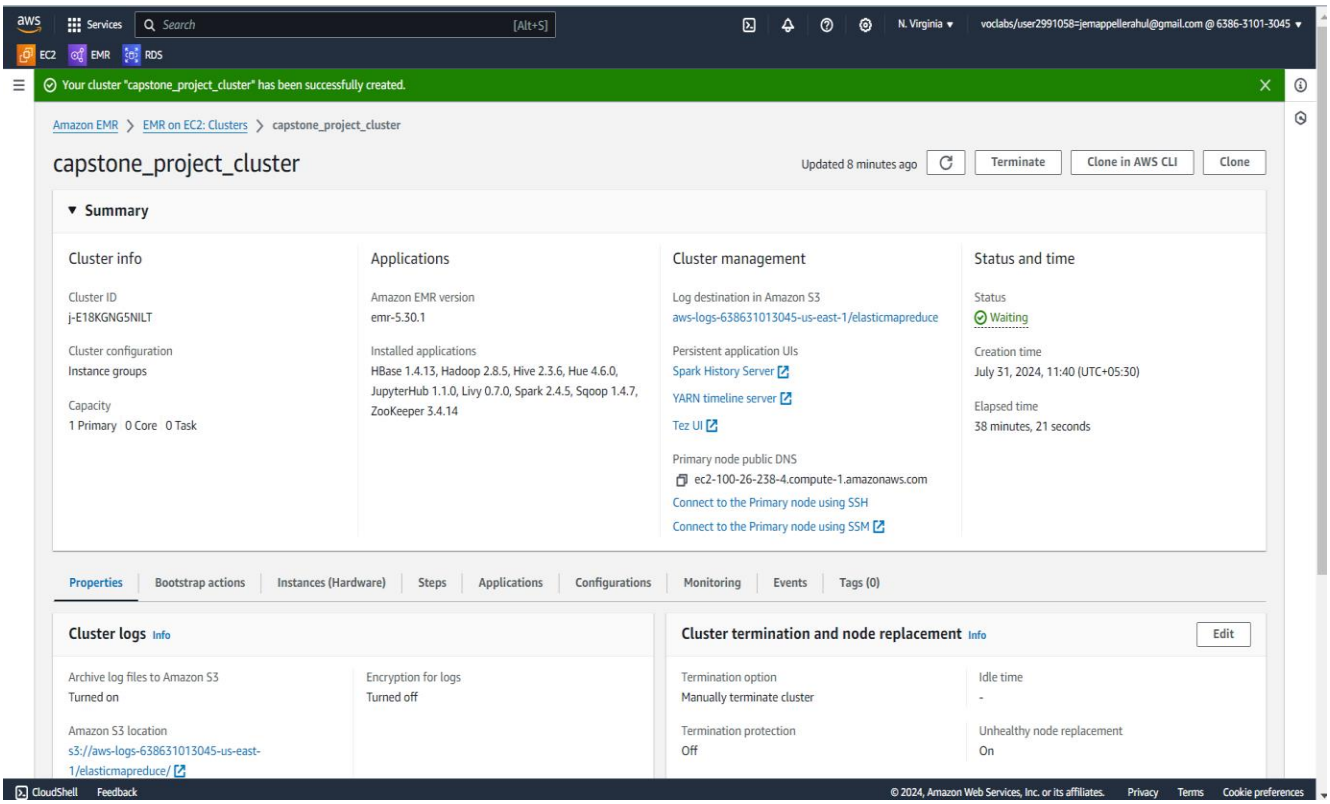
1. In order to complete below tasks, I have created EMR cluster with **HBase , Hadoop , Hive , Hue , jupyterhub , Livy , spark , Sqoop and Zookeeper** , Root device EBS volume size as 20 GB

• **Task 5**: Create a streaming data processing framework that ingests real-time POS transaction data from
Kafka. The transaction data is then validated based on the three rules' parameters (stored in the NoSQL database) discussed previously.

• **Task 6**: Update the transactions data along with the status (fraud/genuine) in the card_transactions table.

• **Task 7**: Store the 'postcode' and 'transaction_dt' of the current transaction in the look-up table in the NoSQL database if the transaction was classified as genuine.

EMR Cluster Configuration:



2. Logged into EMR instance as “hadoop”:

```
login as: hadoop
Authenticating with public key "rahulskey"

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R:::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R::::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR:::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E::::::EEEEEEEEEE  M::::::M M:::M M:::M M::::::M   R:::RRRRRR:::::R
  E:::::::::::::::E  M::::::M  M:::M:::M  M::::::M   R:::::::::::RR
  E::::::EEEEEEEEEE  M::::::M   M:::::M   M::::::M   R:::RRRRRR::::R
  E::::E             M::::::M    M:::M    M::::::M   R:::R      R::::R
  E::::E       EEEEE M::::::M     MMM     M::::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M::::::M             M::::::M   R:::R      R::::R
E::::::::::::::::::E M::::::M             M::::::M RR:::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR
```

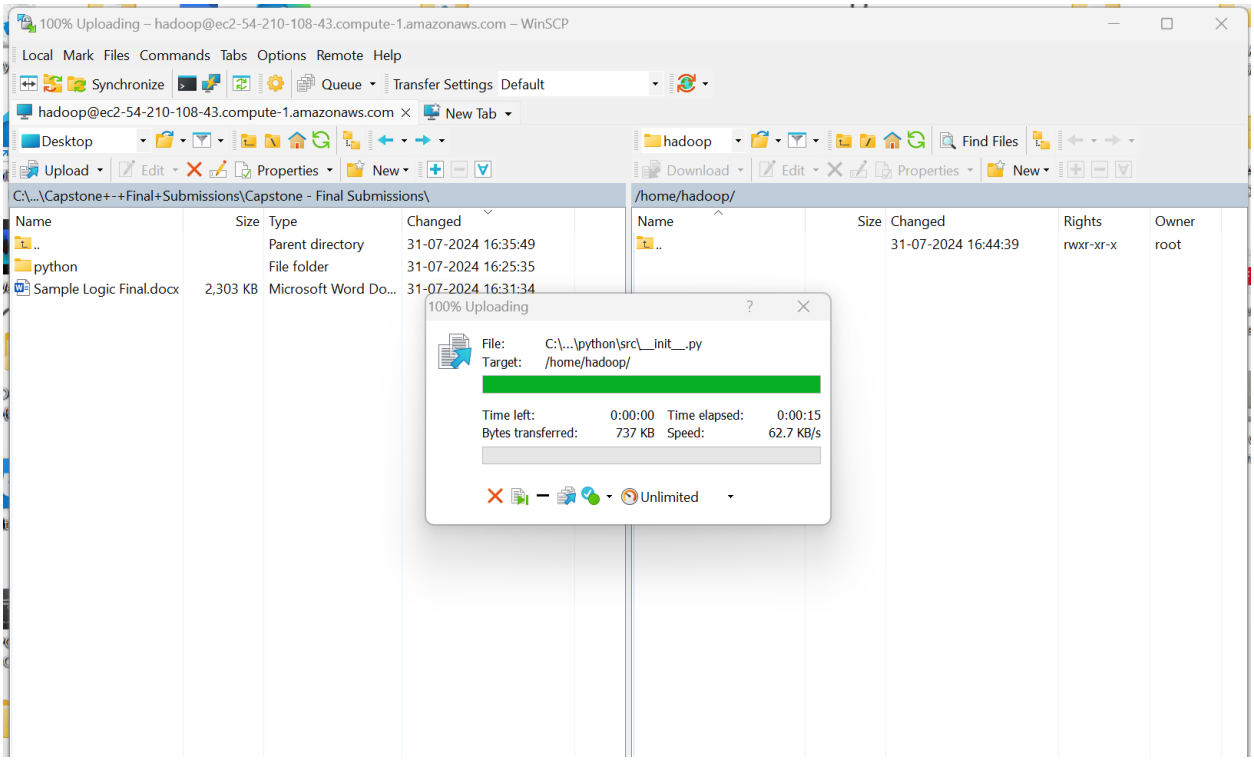
3. Switch to root user and run pip install kafka-python and then again use “sudo -i –u hadoop” to be a hadoop user

```
[root@ip-172-31-49-181 hadoop]#  pip install kafka-python
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip3 install --user` instead.
Collecting kafka-python
  Downloading https://files.pythonhosted.org/packages/75/68/dcb0db055309f680ab2931a3eeb22d865604b638acf8c914bedf4c1a0c8c/kafka_python-2.0.2-py2.py3-none-any.
whl (246kB)
    100% |████████████████████████████████| 256kB 3.8MB/s
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
[root@ip-172-31-49-181 hadoop]#
```

4. Run the following commands in order to Install Happy base and start thrift server

- **sudo yum update**
- **sudo yum install python3-devel**
- **pip install happybase**
- **/usr/lib/hbase/bin/hbase-daemon.sh start thrift -p 9090**

5. Downloaded db-> **dao.py , geomap.py ,rules-> rules.py ,driver.py ,unzipsv.csv** from the resource section of the capstone project from the learning platform and transfer it to hadoop instance via WinSCP.



6. Updated the Public IP of your EC2 Instance “**100.26.238.4**”(self.host) in **dao.py** file

```python
import happybase

class HBaseDao:
    """
    Dao class for operation on HBase
    """
    __instance = None

    @staticmethod
    def get_instance():
        """ Static access method. """
        if HBaseDao.__instance == None:
            HBaseDao()
        return HBaseDao.__instance

    def __init__(self):
        if HBaseDao.__instance != None:
            raise Exception("This class is a singleton!")
        else:
            HBaseDao.__instance = self
            self.host = '100.26.238.4'
            #self.host = 'localhost'
            for i in range(2):
```

7. Updated **rules.py with following parameters:**
lookup_table = 'lookup_data_hbase'
master_table = 'card_transactions_hbase'

```
# List all the functions to check for the rules
from db.dao import HBaseDao
from db.geo_map import GEO_Map
from datetime import datetime
import uuid

# Create UDF functions
lookup_table = 'lookup_data_hbase'
master_table = 'card_transactions_hbase'
```

8. **Created Python functions, containing the logic for the UDFs (rules.py)**
verify_ucl_data : Function to verify the UCL rule Transaction amount should be less than Upper control limit (UCL)

```
def verify_ucl_data(card_id, amount):
    try:
        hbasedao = HBaseDao.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        card_ucl = (card_row[b'card_data:ucl']).decode("utf-8")

        if amount < float(card_ucl):
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

verify_credit_score_data: Function to verify the credit score rule .Credit score of each member should be greater than 200

```
def verify_credit_score_data(card_id):

    try:
        hbasedao = HBaseDao.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        card_score = (card_row[b'card_data:score']).decode("utf-8")

        if int(card_score) > 200:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

verify_postcode_data: Function to verify the following zipcode rules.ZIP code distance

```
def verify_postcode_data(card_id, postcode, transaction_dt):

    try:
        hbasedao = HBaseDao.get_instance()
        geo_map = GEO_Map.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        last_postcode = (card_row[b'card_data:postcode']).decode("utf-8")
        last_transaction_dt = (card_row[b'card_data:transaction_dt']).decode("utf-8")

        current_lat = geo_map.get_lat(str(postcode))
        current_lon = geo_map.get_long(str(postcode))
        previous_lat = geo_map.get_lat(last_postcode)
        previous_lon = geo_map.get_long(last_postcode)

        dist = geo_map.distance(lat1=current_lat, long1=current_lon, lat2=previous_lat, long2=previous_lon)

        speed = calculate_speed(dist, transaction_dt, last_transaction_dt)

        if speed < speed_threshold:
            return True
        else:
            return False

    except Exception as e:
        raise Exception(e)
```

calculate_speed : A function to calculate the speed from distance and transaction timestamp differentials

```
def calculate_speed(dist, transaction_dt1, transaction_dt2):

    transaction_dt1 = datetime.strptime(transaction_dt1, '%d-%m-%Y %H:%M:%S')
    transaction_dt2 = datetime.strptime(transaction_dt2, '%d-%m-%Y %H:%M:%S')

    elapsed_time = transaction_dt1 - transaction_dt2
    elapsed_time = elapsed_time.total_seconds()

    try:
        return dist / elapsed_time
    except ZeroDivisionError:
        return 299792.458
# (Speed of light)
```

verify_rules_status: A function to verify all the three rules - ucl, credit score and speed

```
def verify_rules_status(card_id, member_id, amount, pos_id, postcode, transaction_dt):


    hbasedao = HBaseDao.get_instance()

    # Check if the POS transaction passes all rules.
    # If yes, update the lookup table and insert data in master table as genuine.
    # Else insert the transaction in master table as Fraud.

    rule1 = verify_ucl_data(card_id, amount)
    rule2 = verify_credit_score_data(card_id)
    rule3 = verify_postcode_data(card_id, postcode, transaction_dt)

    if all([rule1, rule2, rule3]):
        status = 'GENUINE'
        hbasedao.write_data(key=str(card_id),
                            row={'card_data:postcode': str(postcode), 'card_data:transaction_dt': str(transaction_dt)},
                            table=lookup_table)
    else:
        status = 'FRAUD'

    new_id = str(uuid.uuid4()).replace('-', '')
    hbasedao.write_data(key=new_id,
                        row={'cardDetail:card_id': str(card_id), 'cardDetail:member_id': str(member_id),
                             'transactionDetail:amount': str(amount), 'transactionDetail:pos_id': str(pos_id),
                             'transactionDetail:postcode': str(postcode), 'transactionDetail:status': str(status),
                             'transactionDetail:transaction_dt': str(transaction_dt)},
                        table=master_table)
    return status
```

9. Next, I updated the **‘driver.py’** file with the following code Setting up the system dependencies and importing necessary libraries and modules

```
import os
import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from rules.rules import *
```

10. Initializing the Spark session and reading input data from Kafka mentioning the details of the Kafka broker, such as bootstrap server, port and topic name
1. Connect to kafka topic using
Bootstrap-server: 18.211.252.152
Port Number: 9092
Topic: transactions-topic-verified

```
# Initialize Spark session
spark = SparkSession.builder.appName("CreditCardFraud").getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Read stream from Kafka
credit_data = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("startingOffsets", "earliest") \
    .option("failOnDataLoss", "false") \
    .option("subscribe", "transactions-topic-verified") \
    .load()
```

11. Define JSON schema of each transactions

```
# Define schema for transaction
dataSchema = StructType([
    StructField("card_id", LongType(), True),
    StructField("member_id", LongType(), True),
    StructField("amount", DoubleType(), True),
    StructField("pos_id", LongType(), True),
    StructField("postcode", IntegerType(), True),
    StructField("transaction_dt", StringType(), True)
])
```

12. Read the raw JSON data from Kafka as 'credit_data_stream' and Define UDF's to verify rules

```
# Extract parsed fields and filter nulls
credit_data_stream = parsed.select("credit_data.*") \
    .filter(
        col("card_id").isNotNull() &
        col("member_id").isNotNull() &
        col("amount").isNotNull() &
        col("postcode").isNotNull() &
        col("pos_id").isNotNull() &
        col("transaction_dt").isNotNull()
    )

# Define UDF which verifies all the rules for each transaction and updates the lookup and master tables
verify_all_rules = udf(verify_rules_status, StringType())

Final_data = credit_data_stream \
    .withColumn('status', verify_all_rules(credit_data_stream['card_id'],
                                           credit_data_stream['member_id'],
                                           credit_data_stream['amount'],
                                           credit_data_stream['pos_id'],
                                           credit_data_stream['postcode'],
                                           credit_data_stream['transaction_dt']))
```

13. Code to display output in console

```
# Write output to console as well
output_data = Final_data \
    .select("card_id", "member_id", "amount", "pos_id", "postcode", "transaction_dt") \
    .writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .start()
```

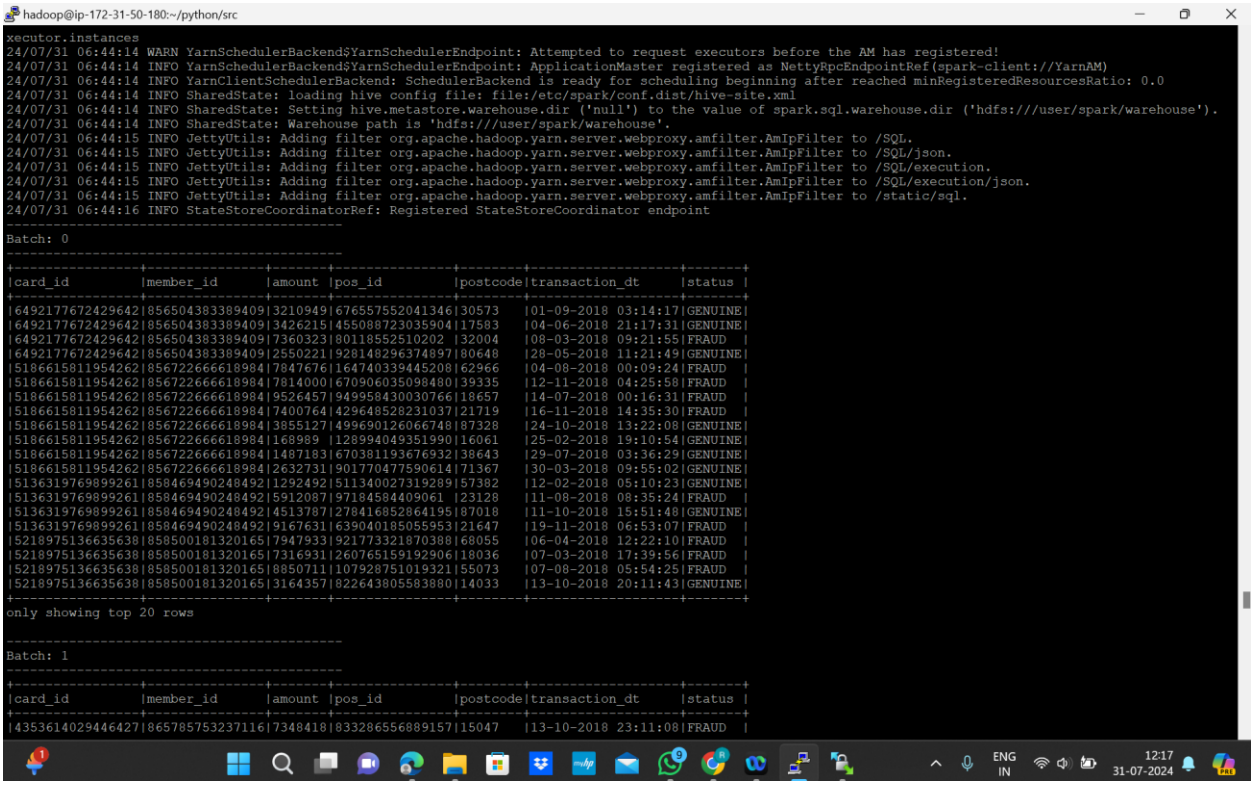
14. Define spark termination

```
# Indicating Spark to await termination
output_data.awaitTermination()
```

15. Set the Kafka Version using the following command
export SPARK_KAFKA_VERSION=0.10

16. Run the spark-submit command, specifying the Spark-SQL-Kafka package and python file and Check Output in console :

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 driver.py



17 . Count Data in Hbase: count 'lookup_data_hive'