CREDIT CARD FRAUD DETECTION – REAL TIME STREAMING

PROBLEM APPROACH

1. Read the source card transactions data from Kafka topic (**transactions-topic-verified**) and define the proper data structure / schema on top of the streaming dataframe.
2. Derive the column **"status"** with two distinct values as **"GENUINE"** and **"FRAUD"** based on the following three rules:
    a. **Credit Score: -** Get the credit score from card lookup HBase table based on card id fetched from the source transactions data and validate it. If the score is less than 200 then label it as "FRAUD" and move to next transaction. Otherwise proceed validating the second rule.
    b. **Check Upper Control Limit (UCL): -** Get the amount from the transaction data and compare it with UCL value present in card lookup HBase table. If the amount is greater than UCL then label it as "FRAUD" and move to next transaction. Otherwise proceed validating the third and final rule.
    c. **Zip code Distance: -** Here we calculate the distance travelled between two locations of the recent two transactions and its time difference and measure speed in terms of Kilometers/Second. We set the threshold value to **0.25 Kilometers/Second (900 Kilometers/Hour – average speed of commercial flights).** If the speed measured is greater than threshold value, then label it as "FRAUD". Otherwise, it can be treated as "GENUINE".
3. Once the transaction status is derived, we can insert the card transactions source data along with its status in card transactions HBase table.
4. Update the postcode and transaction date in card lookup HBase table for the transactions happened only if the status is labelled as "GENUINE".
5. After processing all the transactions within the same micro batch, display the card transactions information with its status in the console for live analysis.

CODE IMPLEMENTATION

1. Import the necessary libraries related to PySpark, Spark Streaming, classes and functions definitions provided as boilerplate code (**dao.py, geo_map.py and rules.py**).

```
# Import necessary libraries
from datetime import datetime
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, to_timestamp, udf
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, LongType, TimestampType
from db.dao import HBaseDao
from db.geo_map import GEO_Map
from rules.rules import is_score_rejected, is_ucl_exceeded, is_zipcode_invalid
```

2. Define the required utility functions and UDF on top of it.
   a. **get_distance(): -** This function is to calculate the distance between current and previous locations where transactions happened based on its latitude and longitude. It instantiates and calls GEO_Map class (geo_map.py) methods to get the latitude and longitude attributes value and calculate the distance in Kilometers.

geo_map.py

```
import math
import pandas as pd


class GEO_Map():
    """
    It hold the  map for zip code and its latitute and longitute
    """
    __instance = None

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

    def __init__(self):
        """ Virtually private constructor. """
        if GEO_Map.__instance != None:
            raise Exception("This class is a singleton!")
        else:
            GEO_Map.__instance = self
            self.map = pd.read_csv("uszipsv.csv", header=None, names=['A',"B",'C','D','E'])
            self.map['A'] =  self.map['A'].astype(str)

    def get_lat(self, pos_id):
        return self.map[self.map.A == pos_id ].B

    def get_long(self, pos_id):
        return self.map[self.map.A == pos_id ].C

    def distance(self, lat1, long1, lat2, long2):
        theta = long1 - long2
        dist = math.sin(self.deg2rad(lat1)) * math.sin(self.deg2rad(lat2)) + math.cos(self.deg2rad(lat1)) * math.cos(self.deg2rad(lat2)) * math.cos(self.deg2rad(thet
a))
        dist = math.acos(dist)
        dist = self.rad2deg(dist)
        dist = dist * 60 * 1.1515 * 1.609344
        return dist

    def rad2deg(self, rad):
        return rad * 180.0 / math.pi

"geo_map.py" 54L, 1205B                                                                1,1           Top
```

driver.py

```
# Utility function to calculate distance between last two locations where transactions happened
def get_distance(trans_postcode, lkp_postcode):
    '''
        Calculate the distance between current and previous locations where transactions happened
        Input Arguments: trans_postcode (postcode from card_transactions real time data - Kafka topic in JSON format)
                         lkp_postcode (postcode from card_lookup table (Hbase) of the same card_id from card_transactions real time data).
    '''
    # Instantiate GEO_Map class object
    geo_obj = GEO_Map.get_instance()
    lat1 = float(geo_obj.get_lat(trans_postcode).values[0])
    long1 = float(geo_obj.get_long(trans_postcode).values[0])
    lat2 = float(geo_obj.get_lat(lkp_postcode).values[0])
    long2 = float(geo_obj.get_long(lkp_postcode).values[0])
    return geo_obj.distance(lat1, long1, lat2, long2)
```

   b. **get_time_diff_secs(): -** This function calculates the time difference between current and previous transactions timestamp.

driver.py

```
# Utility function to calculate time difference between last two transactions happened
def get_time_diff_secs(trans_transactiondt, lkp_transactiondt):
    '''
        Calculate the time difference between current and previous transactions timestamp.
        Input Arguments: trans_transactiondt (transaction_dt from card_transactions real time data - Kafka topic in JSON format)
                         lkp_transactiondt (transaction_dt from card_lookup table (Hbase) of the same card_id from card_transactions real time data).
    '''
    return (trans_transactiondt - lkp_transactiondt).total_seconds()
```

c. **insert_card_transactions_hbase(): -** This function inserts card transactions real time data (Kafka topic in JSON format) into card_transactions HBase table. It first instantiates HbaseDao class object to connect to HBase database and then call required methods to insert the data in card transactions table.

dao.py

```
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 = 'localhost'
            self.connect()


    def connect(self):
        for i in range(3):
            try:
                self.pool = happybase.ConnectionPool(size=3, host=self.host, port=9090)
                break
            except:
                print("Exception in connecting HBase")



    def get_data(self, key, table):
        for i in range(2):
            try:
                with self.pool.connection() as connection:
                    t = connection.table(table)
                    row = t.row(key)
                    return row
            except:
                self.reconnect()

"dao.py" 60L, 1168B
```

```
    def write_data(self, key, row, table):
        error = None
        for i in range(2):
            try:
                with self.pool.connection() as connection:
                    t = connection.table(table)
                    return t.put(key, row)
            except Exception as e:
                error = e
                self.reconnect()
        raise Exception(key + str(e))

    def reconnect(self):
        self.connect()
```

driver.py

```
# Write card transactions final data to HBase table
def insert_card_transactions_hbase(transaction_data):
    '''
        Insert card transactions real time data (Kafka topic in JSON format) into card_transactions hbase table.
        Input Arguments: transaction_data (list object of card transactions real time data - Kafka topic in JSON format).
    '''
    # Instantiate HBaseDao class object to connect to HBase
    hbase_obj = HBaseDao.get_instance()
    trans_key = '|'.join([str(v) for v in transaction_data])
    data = {b'transaction_data:card_id':bytes(str(transaction_data[0]), encoding='utf8'),
            b'transaction_data:member_id':bytes(str(transaction_data[1]), encoding='utf8'),
            b'transaction_data:amount':bytes(str(transaction_data[2]), encoding='utf8'),
            b'transaction_data:pos_id':bytes(str(transaction_data[3]), encoding='utf8'),
            b'transaction_data:postcode':bytes(str(transaction_data[4]), encoding='utf8'),
            b'transaction_data:transaction_dt':bytes(str(transaction_data[5]), encoding='utf8'),
            b'transaction_data:status':bytes(str(transaction_data[6]), encoding='utf8')}
    hbase_obj.write_data(bytes(trans_key, encoding='utf8'), data, 'card_transactions')
```

d. **update_card_lookup_hbase(): -** This function updates postcode and transaction_dt attributes in card lookup table HBase for which transactions are treated as GENUINE. It first instantiates HbaseDao class object to connect to HBase database and then call required methods to update the card lookup data.

driver.py

```
# Update latest postcode and transaction date in card lookup HBase table
def update_card_lookup_hbase(trans_card_id, status_flag, lkp_data):
    '''
        Update postcode and transaction_dt attributes in card lookup table HBase for which transactions are treated as GENUINE.
        Input Arguments: trans_card_id
                         status_flag
                         lkp_data (List object of postcode and transaction_dt card transactions real time data - Kafka topic in JSON format).
    '''
    # Instantiate HBaseDao class object to connect to HBase
    hbase_obj = HBaseDao.get_instance()
    if status_flag == 'GENUINE':
        key = bytes(str(trans_card_id), encoding='utf8')
        data = {b'lkp_info:postcode':bytes(str(lkp_data[0]), encoding='utf8'),
                b'lkp_info:transaction_dt':bytes(str(lkp_data[1]), encoding='utf8')}
        hbase_obj.write_data(key, data, 'card_lookup')
```

e. **check_status(): -** This function is to detect the fradulent of the live transactions based on the rules defined **(mentioned in the problem approach)** and lookup data prepared as part of batch processing.

rules.py

```
def is_score_rejected(score):
    '''
        This function is to get the credit score of the particular card which transaction happened and see whether it has greater than or equal to 200. If not then i
t is labelled as Fraud.
        Input Arguments: score (credit_score from card_transactions real time data - Kafka topic in JSON format)
    '''
    if score < 200:
        return True
    return False


def is_ucl_exceeded(amount, ucl):
    '''
        This function is to check whether the amount exceeds the upper control limit of the transaction to be done. If it is exceeded then it is labelled as Fraud.
        Input Arguments: amount (card_transactions real time data - Kafka topic in JSON format)
                         ucl (UCL attribute value from card_lookup table - HBase)
    '''
    if amount > ucl:
        return True
    return False


def is_zipcode_invalid(distance, time_diff_secs):
    '''
        This function is to measure the speed in Kilometers/second travelled by the customer between two locations where customer made the recent two transactions (c
urrent and previous).
        Usually commercial airplanes travel with an average speed of around 800-950 Kilometers/hour. We considered the speed threshold value as 900 Kilometers/hour a
nd converted it to seconds which is (900/3600) = 0.25 Kilometers/second. If the measure speed between two locations is greater than 0.25 Kilometers/second then it is
 labelled as Fraud.
        Input Arguments: distance (distance convered between two locations of recent two transactions)
                         time_diff_secs (Time difference between last two transactions happened)
    '''
    if time_diff_secs == 0:
        if distance != 0:
            return True
    else:
        speed_kmps = distance / time_diff_secs
        if speed_kmps > 0.25:
            return True
    return False
```

driver.py

```
def check_status(trans_card_id, trans_member_id, trans_amount, trans_pos_id, trans_postcode, trans_transactiondt):
    '''
        This function is to detect the fradulent of the live transactions based on the rules defined and lookup data prepared as part of batch processing.
        The following three rules to be validated:
            1. Check credit score
            2. Check UCL threshold with amount
            3. Measure the speed travelled in Kilometers/second between the locations of last two transactions.
        If any of the above rule fails then the transaction is labelled as FRAUD. Otherwise it is labelled as GENUINE.
        Input Arguments: trans_card_id (card id attribute from card_transactions real time data - Kafka topic in JSON format)
                         trans_member_id (member id attribute from card_transactions real time data - Kafka topic in JSON format)
                         trans_amount (transaction amount from card_transactions real time data - Kafka topic in JSON format)
                         trans_pos_id (pos_id from card_transactions real time data - Kafka topic in JSON format)
                         trans_postcode (postcode from card_transactions real time data - Kafka topic in JSON format)
                         trans_transactiondt (transaction_dt from card_transactions real time data - Kafka topic in JSON format)
    '''
    status_flag = 'GENUINE'
    # Instantiate HBaseDao class object to connect to HBase
    hbase_obj = HBaseDao.get_instance()
    lkp_row = hbase_obj.get_data(str(trans_card_id), 'card_lookup')
    if lkp_row == {}:
        pass
    else:
        lkp_score, lkp_postcode, lkp_transactiondt, lkp_ucl = [val.decode('utf8') for val in list(lkp_row.values())]
        lkp_score = int(lkp_score)
        lkp_postcode = int(lkp_postcode)
        lkp_transactiondt = datetime.strptime(lkp_transactiondt, '%Y-%m-%d %H:%M:%S')
        lkp_ucl = float(lkp_ucl)

        # Check transactions rules to determine credit score, UCL and zipcode distance.
        if is_score_rejected(lkp_score):
            status_flag = 'FRAUD'
        elif is_ucl_exceeded(trans_amount, lkp_ucl):
            status_flag = 'FRAUD'
        elif is_zipcode_invalid(get_distance(str(trans_postcode), str(lkp_postcode)), get_time_diff_secs(trans_transactiondt, lkp_transactiondt)):
            status_flag = 'FRAUD'
        else:
            pass

    # Call insert_card_transactions_hbase() function to insert the card transaction data with status derived
    insert_card_transactions_hbase([trans_card_id, trans_member_id, trans_amount, trans_pos_id, trans_postcode, trans_transactiondt, status_flag])
    # Call update_card_lookup_hbase() to insert/update postcode and transaction_dt attributes in card_lookup HBase table
    update_card_lookup_hbase(trans_card_id, status_flag, [trans_postcode, trans_transactiondt])

    return status_flag
```

f. Define UDF (fraud_detection) on top of check_status() utility function that to be used in spark streaming dataframe for deriving “status” attribute during real time. As part of this UDF, it insert the records into card transactions HBase table and update the postcode and transaction date in card lookup HBase table (as shown in check_status() driver.py screenshot above).

driver.py

```
# Create fraud detection UDF on top of status utility function
fraud_detection = udf(check_status, StringType())
```

3. Create a Spark Session which is an entry point to spark application and set the log level to ‘ERROR’.

driver.py

```
# Create a spark session
spark = SparkSession \
        .builder \
        .appName("Fraud Detection - Card Transactions") \
        .getOrCreate()

# Set log level to ERROR
spark.sparkContext.setLogLevel('ERROR')
```

4. Read the card transactions real time data from Kafka source topic (transactions-topic-verified).

driver.py

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

5. Define the proper data structure or schema and assign it to the source data read from Kafka topic.

driver.py

```
# Defining the Data Structure for source card transactions (json format).
data_structure = StructType([
                            StructField("card_id", LongType()),
                            StructField("member_id", LongType()),
                            StructField("amount", IntegerType()),
                            StructField("pos_id", LongType()),
                            StructField("postcode", IntegerType()),
                            StructField("transaction_dt", StringType())
                        ])

# Converting source data from json format to dataframe (table).
card_trans_formatted =  card_trans_source \
                .select(from_json(col("value").cast("string"), data_structure).alias("card_transactions")) \
                .select("card_transactions.*") \
                .withColumn("transaction_dt", to_timestamp(col('transaction_dt'),'dd-MM-yyyy HH:mm:ss')) \
                .select("card_id",
                        "member_id",
                        "amount",
                        "pos_id",
                        "postcode",
                        "transaction_dt"
                        )
```

6. Add a new column “status” by calling the fraud_detection UDF defined above. This UDF will also take of inserting and updating the records in card transactions and card lookup HBase tables.

driver.py

```
# Derive status attribute: GENUINE or FRAUD based on rules defined
card_trans_transformed = card_trans_formatted \
                        .withColumn("status", fraud_detection(card_trans_formatted.card_id, card_trans_formatted.member_id, card_trans_formatted.amount, card_trans_
formatted.pos_id, card_trans_formatted.postcode, card_trans_formatted.transaction_dt)) \
                        .select("card_id",
                        "member_id",
                        "amount",
                        "pos_id",
                        "postcode",
                        "transaction_dt",
                        "status"
                        )
```

7. Write the final transformed card transactions data with its status in the console for live analysis. It runs as a micro-batch with processing time of every **“1 Minute”** until it is interrupted/terminated.

driver.py

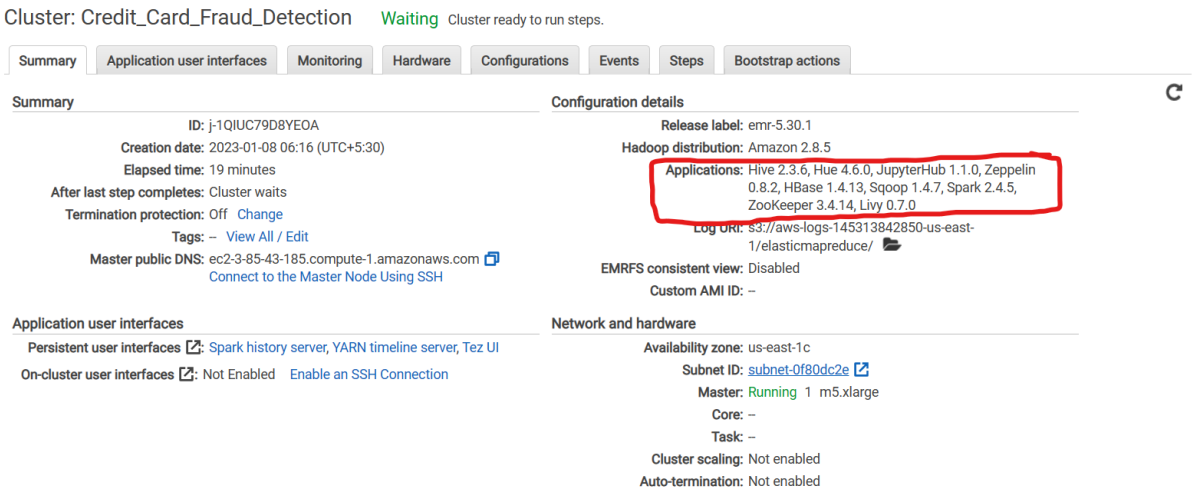
```
# Write the final transformed data to sink
card_trans_sink = card_trans_transformed \
                  .writeStream \
                  .outputMode("append") \
                  .format("console") \
                  .option("truncate", "false") \
                  .trigger(processingTime="1 minute") \
                  .start()

# Await termination
card_trans_sink.awaitTermination()
```

DEPLOYMENT PLAN

PRE-REQUISITES:

1. Create an EMR cluster with **Hive, HBase, Sqoop, Spark, Zookeeper** applications installed in it as shown below:



Applications such as Hue, JupyterHub, Zeppelin, Livy are optional and depends based on your preference.

2. Download and install MySQL connector in AWS EMR for performing Sqoop data ingestion from AWS RDS to HDFS.
   cd /home/hadoop
   wget https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
   tar -xvf mysql-connector-java-8.0.25.tar.gz

cd mysql-connector-java-8.0.25/
sudo cp mysql-connector-java-8.0.25.jar /usr/lib/sqoop/lib/

3. Download and install Hive-HBase handler jar file which will be used for card lookup data preparation for Hive and Hbase tables using PySpark. and place it in spark libraries folder. Also place some other required HBase jar files from HBase libraries to spark libraries folder.
cd /home/hadoop
wget https://repo1.maven.org/maven2/org/apache/hive/hive-hbase-handler/3.1.3/hive-hbase-handler-3.1.3.jar
sudo cp hive-hbase-handler-3.1.3.jar /usr/lib/spark/jars/
sudo cp /usr/lib/hbase/*.jar /usr/lib/spark/jars/
sudo cp /usr/lib/hbase/lib/htrace-core-3.1.0-incubating.jar /usr/lib/spark/jars/
sudo cp /usr/lib/hbase/lib/guava-12.0.1.jar /usr/lib/spark/jars/
sudo cp /usr/lib/hbase/lib/metrics-core-2.2.0.jar /usr/lib/spark/jars/

4. Create all the required tables in Hive and HBase such as **card_member, member_score, card_transactions (historical data) and card_lookup** tables and load the pre-processed data in it **as part of batch processing (mid submission).**

HBase:

```
hbase(main):001:0> list
TABLE
card_lookup
card_transactions
2 row(s) in 0.1870 seconds

=> ["card_lookup", "card_transactions"]
hbase(main):002:0>
```

Hive:

```
[hadoop@ip-172-31-84-212 ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> show databases;
OK
cred_financials_data
default
Time taken: 0.499 seconds, Fetched: 2 row(s)
hive> use cred_financials_data;
OK
Time taken: 0.019 seconds
hive> show tables;
OK
card_lookup
card_member
card_transactions
member_score
Time taken: 0.024 seconds, Fetched: 4 row(s)
hive>
```

5. Switch to root user in EMR cluster and install pandas and happybase python packages.
   sudo su root
   yum install gcc (Press 'Y' and ENTER after it prompts to download)
   yum install python3-devel (Press 'Y' and ENTER after it prompts to download)
   pip install cython
   pip install numpy
   pip install pandas
   pip install happybase
   exit

DEPLOYMENT STEPS:

1. Login to AWS EMR and create the python directory under home directory and place all the executable python files inside it (either using scp command or WinSCP tool) and change its permissions.
   cd /home/hadoop
   mkdir -p /home/hadoop/python/src

   Place the files using scp command from local to EMR

```
C:\Users\Ganesh Jalakam>scp -r -P 22 -i "C:\Users\Ganesh Jalakam\Documents\AWS\Instance_Keys\RHEL.pem" "D:\IIITB_Data_Sc
ience\PG_DS_Course\Capstone_Project\Final_Submission\python\*" hadoop@ec2-3-85-43-185.compute-1.amazonaws.com:/home/hado
op/python/
dao.py                                                                    100% 1168     4.9KB/s   00:00
geo_map.py                                                                100% 1205     4.3KB/s   00:00
__init__.py                                                               100%    0     0.0KB/s   00:00
dao.cpython-37.pyc                                                        100% 1732     6.7KB/s   00:00
__init__.cpython-37.pyc                                                   100%  125     0.4KB/s   00:00
driver.py                                                                 100%   10KB  44.5KB/s   00:00
rules.py                                                                  100% 2760    11.9KB/s   00:00
__init__.py                                                               100%    0     0.0KB/s   00:00
uszipsv.csv                                                               100%  735KB 474.0KB/s   00:01
__init__.py                                                               100%    0     0.0KB/s   00:00

C:\Users\Ganesh Jalakam>
```

   Change the executable files permissions:
   cd /home/hadoop/python/src
   chmod 755 *.py
   chmod 755 db/*.py
   chmod 755 rules/*.py

   driver.py - /home/hadoop/python/src

```
[hadoop@ip-172-31-84-212 src]$ pwd
/home/hadoop/python/src
[hadoop@ip-172-31-84-212 src]$ ls -ltr
total 748
drwxrwxrwx 3 hadoop hadoop     76 Jan  8 01:35 db
-rwxr-xr-x 1 hadoop hadoop  10319 Jan  8 01:36 driver.py
drwxrwxrwx 2 hadoop hadoop     41 Jan  8 01:36 rules
-rw-rw-r-- 1 hadoop hadoop 752688 Jan  8 01:36 uszipsv.csv
-rwxr-xr-x 1 hadoop hadoop      0 Jan  8 01:36 __init__.py
[hadoop@ip-172-31-84-212 src]$
```

dao.py and geo_map.py – /home/hadoop/python/src/db

```
/home/hadoop/python/src/db
[hadoop@ip-172-31-84-212 db]$ ls -ltr
total 8
-rwxr-xr-x 1 hadoop hadoop 1168 Jan  8 01:35 dao.py
-rwxr-xr-x 1 hadoop hadoop 1205 Jan  8 01:35 geo_map.py
-rwxr-xr-x 1 hadoop hadoop    0 Jan  8 01:35 __init__.py
drwxrwxrwx 2 hadoop hadoop   63 Jan  8 01:35 __pycache__
[hadoop@ip-172-31-84-212 db]$
```

rules.py - /home/hadoop/python/src/rules

```
/home/hadoop/python/src/rules
[hadoop@ip-172-31-84-212 rules]$ ls -ltr
total 4
-rwxr-xr-x 1 hadoop hadoop 2760 Jan  8 01:36 rules.py
-rwxr-xr-x 1 hadoop hadoop    0 Jan  8 01:36 __init__.py
[hadoop@ip-172-31-84-212 rules]$
```

2. Zip all the python dependencies into one file with name src.zip (containing dao.py, geo_map.py and rules.py in it).

cd /home/hadoop/python/src
zip src.zip __init__.py rules/* db/*

```
[hadoop@ip-172-31-84-212 src]$ zip src.zip __init__.py rules/* db/*
  adding: __init__.py (stored 0%)
  adding: rules/__init__.py (stored 0%)
  adding: rules/rules.py (deflated 68%)
  adding: db/dao.py (deflated 61%)
  adding: db/geo_map.py (deflated 56%)
  adding: db/__init__.py (stored 0%)
  adding: db/__pycache__/ (stored 0%)
[hadoop@ip-172-31-84-212 src]$ ls -ltr
total 752
drwxrwxrwx 3 hadoop hadoop     76 Jan  8 01:35 db
-rwxr-xr-x 1 hadoop hadoop  10319 Jan  8 01:36 driver.py
drwxrwxrwx 2 hadoop hadoop     41 Jan  8 01:36 rules
-rw-rw-r-- 1 hadoop hadoop 752688 Jan  8 01:36 uszipsv.csv
-rwxr-xr-x 1 hadoop hadoop      0 Jan  8 01:36 __init__.py
-rw-rw-r-- 1 hadoop hadoop   2968 Jan  8 01:53 src.zip
```

3. Run the following command to enable Kafka Integration with Apache Spark.
export SPARK_KAFKA_VERSION=0.10

4. Run driver.py script to start spark streaming job which consumes real time transactions data and detects the status (FRAUD or GENUINE) and display the final output of each micro batches in the console.

spark-submit --py-files src.zip --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 --files uszipsv.csv driver.py

Console Output:

```
23/01/08 01:57:57 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /static/sql.
23/01/08 01:57:58 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
-------------------------------------------
Batch: 0
-------------------------------------------
+----------------+-------------+-------+---------------+--------+-------------------+-------+
|card_id         |member_id    |amount |pos_id         |postcode|transaction_dt     |status |
+----------------+-------------+-------+---------------+--------+-------------------+-------+
|348702330256514 |37495066290  |4380912|248063406800722|96774   |2018-03-01 08:24:29|GENUINE|
|348702330256514 |37495066290  |6703385|786562777140812|84758   |2018-06-02 04:15:03|GENUINE|
|348702330256514 |37495066290  |7454328|4669525713935O8|93645   |2018-02-12 09:56:42|GENUINE|
|348702330256514 |37495066290  |4013428|45845320330319 |15868   |2018-06-13 05:38:54|GENUINE|
|348702330256514 |37495066290  |5495353|545499621965697|79033   |2018-06-16 21:51:54|GENUINE|
|348702330256514 |37495066290  |3966214|369266342272501|22832   |2018-10-21 03:52:51|GENUINE|
|348702330256514 |37495066290  |1753644|947502929267l  |17923   |2018-08-23 00:11:30|GENUINE|
|348702330256514 |37495066290  |1692115|27647525195860 |55708   |2018-11-23 17:02:39|GENUINE|
|5189563368503974|117826301530 |9222134|525701337355194|64002   |2018-03-01 20:22:10|GENUINE|
|5189563368503974|117826301530 |4133848|182031383443115|26346   |2018-09-09 01:52:32|GENUINE|
|5189563368503974|117826301530 |8938921|799748246411019|76934   |2018-12-09 05:20:53|GENUINE|
|5189563368503974|117826301530 |1786366|131276818071265|63431   |2018-08-12 14:29:38|GENUINE|
|5189563368503974|117826301530 |9142237|5642402596789O3|50635   |2018-06-16 19:37:19|GENUINE|
|5407073344486464|1147922084344|6885448|887913906711117|59031   |2018-05-05 07:53:53|GENUINE|
|5407073344486464|1147922084344|4028209|116266051118182|80118   |2018-08-11 01:06:50|GENUINE|
|5407073344486464|1147922084344|3858369|896105817613325|53820   |2018-07-12 17:37:26|GENUINE|
|5407073344486464|1147922084344|9307733|729374116016479|14898   |2018-07-13 04:50:16|GENUINE|
|5407073344486464|1147922084344|4011296|543373367319647|44028   |2018-10-17 13:09:34|GENUINE|
|5407073344486464|1147922084344|9492531|211980095659371|49453   |2018-04-21 14:12:26|GENUINE|
|5407073344486464|1147922084344|7550074|345533088112099|15030   |2018-09-29 02:34:52|GENUINE|
+----------------+-------------+-------+---------------+--------+-------------------+-------+
only showing top 20 rows
```

Job runs continuously as micro batches with 1 Minute time interval until it is terminated/interrupted.

VALIDATION

1. Check the card transactions count before and after streaming the data.
   HBase: - count 'card_transactions'
   BEFORE STREAMING – 53292 records

```
Current count: 13000, row: 378118026791597|9660204312957O4|6203511|16127|9220957723853S4|2017-08-28 03:42:28|GENUINE
Current count: 14000, row: 4009218272111551|4235179192116O3|3454802|17968|9497993300941Z8|2016-04-12 02:27:30|GENUINE
Current count: 15000, row: 4067184730430984|6217160904962451|2799558|34142|9897258011858371|2017-03-11 00:11:10|GENUINE
Current count: 16000, row: 413238112204142G|6583276704439Z|4640125|22430|2226941745402971|2016-06-06 18:28:07|GENUINE
Current count: 17000, row: 4264553579186587|4700219005367OO|9146367|97492|1197386298728l0|2017-07-26 19:44:00|GENUINE
Current count: 18000, row: 4373464339970856|353188612655228|7070719|53820|8961058176133Z5|2018-01-05 17:27:03|GENUINE
Current count: 19000, row: 4439501833420177|6400039683427O1|7241995|58212|4306275069690Z9|2017-06-06 18:54:15|GENUINE
Current count: 20000, row: 4500499651579063|9821713214770331|5918038|4925A|340254570257538|2017-11-19 18:21:24|GENUINE
Current count: 21000, row: 4566853351752365|2083093364762641|5007068|42712|4072041597002851|2016-03-31 06:03:05|GENUINE
Current count: 22000, row: 4600996294769125|6416109555267391|2700042|62028|3259767048028591|2017-10-07 23:19:53|GENUINE
Current count: 23000, row: 4689314809377828|2602620565358l2|9627742|24554|6174772079114881|2017-08-21 03:10:13|GENUINE
Current count: 24000, row: 4766789897935106|6899627423644351|2625284|48617|9993659017698401|2017-08-20 06:06:12|GENUINE
Current count: 25000, row: 4863127030291206|7883348231400961|4043137|23177|7587274239910591|2016-11-24 20:48:46|GENUINE
Current count: 26000, row: 4907253800863053|7231326592006371|8283600|49874|5230305394771151|2017-03-15 13:01:59|GENUINE
Current count: 27000, row: 4995478705000641|7336628987603341|353412|24557|6969038557385001|2017-04-10 21:02:55|GENUINE
Current count: 28000, row: 5134479292018417|2094329909406811|1693879|46713|9948652024951621|2017-11-01 09:42:05|GENUINE
Current count: 29000, row: 5162808285745682|9905713849232601|9357712|82932|2725105899591301|2016-08-16 13:53:15|GENUINE
Current count: 30000, row: 5186615811954262|8567226666189841|1162144|24554|6174772079114881|2018-01-19 22:02:13|GENUINE
Current count: 31000, row: 5221229593682054|7622457493025981|2225338|46076|7109634354537491|2017-12-19 09:22:27|GENUINE
Current count: 32000, row: 5258095619226135|4799166510764771|4662643|78931|3188678414995|2017-12-20 06:26:28|GENUINE
Current count: 33000, row: 5294592751808411|8095181237919251|6415146|22847|1458098198921391|2017-04-20 09:16:35|GENUINE
Current count: 34000, row: 5342400571435088|8732267588672|6413895|25860|9882604681349361|2016-03-30 16:00:23|GENUINE
Current count: 35000, row: 5380978184175608|2916078628032021|6916930|78933|4320509139752021|2017-02-17 19:03:46|GENUINE
Current count: 36000, row: 5414439899219272|9485997903290371|2975624|17353|9940508995365341|2017-09-28 12:51:34|GENUINE
Current count: 37000, row: 5481808794715436|6898278072589041|5088559|79331|3624551491342661|2016-12-20 09:31:00|GENUINE
Current count: 38000, row: 5534323829711423|5276098774733441|733037|62313|8082271427030951|2018-01-14 20:54:02|GENUINE
Current count: 39000, row: 5584977018799504|7658997649055551|9068245|46510|1891520624283601|2016-09-13 01:59:34|GENUINE
Current count: 40000, row: 6011139413319542|5822886284800571|9799683|65604|4705651804342921|2017-03-19 18:51:29|GENUINE
Current count: 41000, row: 6011525010455848|5385552135011981|9400038|28719|1461984749453281|2017-11-17 22:28:00|GENUINE
Current count: 42000, row: 6011782857327719|9699662222267711|1982856|17061|8411716948486801|2017-09-26 11:42:09|GENUINE
Current count: 43000, row: 6221796595498984|6173139528791291|1551821|35674|8340755789646611|2017-01-10 18:09:06|GENUINE
Current count: 44000, row: 6224271253849917|8151877372537021|140177|83849|7131165095637771|2018-01-25 18:29:05|GENUINE
Current count: 45000, row: 6225606551069826|2846434194526031|7500507|22901|6062024484448931|2017-12-05 03:01:57|GENUINE
Current count: 46000, row: 6228733641419063|6103306395946121|3051758|39169|3095272908275921|2016-07-14 03:34:45|GENUINE
Current count: 47000, row: 6447877814927926|9079729499987451|923252|56685|9927479682107441|2018-01-11 00:00:00|GENUINE
Current count: 48000, row: 6461356425954109|7393259968607561|8515416|10801|5318203282043831|2017-12-25 04:03:59|GENUINE
Current count: 49000, row: 6480152634975473|4390839985268211|9304601|71047|1817117984213061|2018-01-08 19:11:10|GENUINE
Current count: 50000, row: 6505080237250161|6157545673071501|8801408|27341|4869821678528201|2017-12-05 17:04:37|GENUINE
Current count: 51000, row: 6544876671165176|2690986107602551|4604408|14510|5364978824670981|2018-01-21 07:23:36|GENUINE
Current count: 52000, row: 6574255180086418|8917022430607471|5991679|33097|8915869718489581|2016-04-22 01:13:11|GENUINE
Current count: 53000, row: 6595814135833988|2368644264088371|3243199|56162|8343078852601851|2016-10-08 23:28:28|GENUINE
53292 row(s) in 2.8260 seconds

=> 53292
hbase(main):003:0>
```

AFTER STREAMING – 59367 records

```
Current count: 19000, row: 4264553579186587|4700219005367000|4272858|47578|552792990977827|2017-02-13 04:27:36|GENUINE
Current count: 20000, row: 4367125470626536|1799978374740451|5811475|32535|945113329039084|2017-09-28 11:06:14|GENUINE
Current count: 21000, row: 4411296946890133|2290734922852000|5784600|46986|931311742932204|2018-01-18 16:11:30|GENUINE
Current count: 22000, row: 4492848090158931|6138385241281301|3001807|71043|366296280375812|2017-05-04 01:34:23|GENUINE
Current count: 23000, row: 4539744177029050|6425606486414301|1999812|16921|198567387970489|2016-10-14 16:24:41|GENUINE
Current count: 24000, row: 4585507001684561|4479174676558711|1078211|99136|888926523445515|2016-06-15 16:39:06|GENUINE
Current count: 25000, row: 4625371899772166|3060364661764961|4422540|61064|639432351795127|2017-10-07 12:47:53|GENUINE
Current count: 26000, row: 4719418574936017|5000004043659347|1912413|54859|765832549139249|2016-10-31 19:13:16|GENUINE
Current count: 27000, row: 4782879464621468|2571348992932541|3321180|30738|599036413189891|2017-06-22 22:53:28|GENUINE
Current count: 28000, row: 4865556006187980|4541023588263141|3037500|88337|400124434036810|2017-10-04 00:31:46|GENUINE
Current count: 29000, row: 4907253800863053|7231326592000637|3515852|83436|482389848742790|2017-04-11 00:33:35|GENUINE
Current count: 30000, row: 4981548424137027|9964116352892701|5381248|98377|332218533399343|2017-01-12 08:46:29|GENUINE
Current count: 31000, row: 5127318999406559|3916030082950071|1100805|27503|452641550490931|2017-04-12 22:12:19|GENUINE
Current count: 32000, row: 5155306014315751|4038088712282321|5720247|38847|559005303391321|2016-04-21 00:38:29|GENUINE
Current count: 33000, row: 5175735819607028|2276860597957991|7781043|52054|215945504043719|2018-02-11 00:00:00|GENUINE
Current count: 34000, row: 5204123273729700|1330861957601041|9026991|13319|215972630915917|2017-11-01 07:56:45|GENUINE
Current count: 35000, row: 5231456036333304|7795696335538551|3973522|73569|681145934981754|2016-11-04 23:17:24|GENUINE
Current count: 36000, row: 5263774491002052|8396574761810151|7642957|955389144524492|61050|2018-05-11 18:39:03|GENUINE
Current count: 37000, row: 5300868809771757|3746294492452681|9985475|117986020341542|39421|2018-08-23 09:54:37|GENUINE
Current count: 38000, row: 5343422029247948|9102947645937391|9686191|45850|753629127321217|2016-03-18 03:47:58|GENUINE
Current count: 39000, row: 5380978184175608|2916078628032021|5072606|96744|996436980380873|2017-08-13 07:19:17|GENUINE
Current count: 40000, row: 5411856842735919|5582894027589171|521850|65270|6502847843078561|2016-02-16 17:28:13|GENUINE
Current count: 41000, row: 5462457589398710|1981690890872461|8455206|23085|783487405458889|2017-12-21 14:02:30|GENUINE
Current count: 42000, row: 5517057937158439|8257231347594411|1828956|50261|313053798939914|2017-06-30 08:12:24|GENUINE
Current count: 43000, row: 5572427538311236|9767403978945981|2801376|62297|3602072327463441|2017-04-15 04:37:44|GENUINE
Current count: 44000, row: 5595273277507573|2628249032761971|9786992|16240|623747780727948|2017-12-11 00:00:00|GENUINE
Current count: 45000, row: 6011289432864968|7260158897254171|6860143|46301|6021792296834661|2016-09-18 04:53:05|GENUINE
Current count: 46000, row: 6011590374869018|4301327836205231|8045248|27814|3584669184607421|2017-07-08 00:32:41|GENUINE
Current count: 47000, row: 6011798606614876|5575090219840371|4722088|24317|8250496086376541|2018-01-30 21:22:36|GENUINE
Current count: 48000, row: 6221801089095872|4574817526644801|9172465|51263912098343331|42223|2018-04-08 01:44:58|GENUINE
Current count: 49000, row: 6223938739934916|8219364714228851|6186915|42056|847187293322830|2016-05-18 02:12:46|GENUINE
Current count: 50000, row: 6225472681819758|7189587359657811|2297129|48005|811847495441590|2017-02-22 14:55:55|GENUINE
Current count: 51000, row: 6227925299479230|5657908394322791|9384231|513431311148007|22511|2018-06-18 14:14:06|GENUINE
Current count: 52000, row: 6443783023069597|5233259378729911|2070595|13154|606469468197708|2017-07-16 17:27:38|GENUINE
Current count: 53000, row: 6457507104644777|6833074049616911|434978|46392|234052426105135|2017-09-11 00:00:00|GENUINE
Current count: 54000, row: 6474282068752847|1828020112039061|1511781|35646|191149080747748|2017-06-21 06:30:45|GENUINE
Current count: 55000, row: 6490380638769885|4711935015544211|12117|39851|150112068926146|2017-10-28 03:52:37|GENUINE
Current count: 56000, row: 6515567258324915|2032593822492551|3888023|74450|409035176270258|2017-02-12 02:07:33|GENUINE
Current count: 57000, row: 6545952322379984|6007668538287001|4283702|15956|325720495071260|2017-12-06 00:05:58|GENUINE
Current count: 58000, row: 6574656060675651|5551270394774421|3607958|78368|946610588625308|2016-07-26 14:01:52|GENUINE
Current count: 59000, row: 6595638658736751|3779182136146391|7015530|24464|4524889047980812017-11-19 07:49:23|GENUINE
59367 row(s) in 1.6260 seconds

=> 59367
hbase(main):005:0>
```

Hive: - select status, count(*) from card_transactions group by status;

BEFORE STREAMING

Status	_c1
FRAUD	82
GENUINE	53210

```
hive> select status, count(*) from card_transactions group by status;
Query ID = hadoop_20230108012501_db1b7cd1-2a03-468e-820a-1d2ba880cd36
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1673139411187_0014)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.27 s
--------------------------------------------------------------------------------
OK
status  _c1
FRAUD   82
GENUINE 53210
Time taken: 12.494 seconds, Fetched: 2 row(s)
hive>
```

AFTER STREAMING

status	_c1
FRAUD	212
GENUINE	59155

```
hive> select status, count(*) from card_transactions group by status;
Query ID = hadoop_20230108020621_9e72e43d-2d80-49a8-ab62-0304aab4204e
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1673139411187_0016)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.30 s
--------------------------------------------------------------------------------
OK
status  _c1
FRAUD   212
GENUINE 59155
Time taken: 19.15 seconds, Fetched: 2 row(s)
hive>
```

2. Check the latest transaction date available in card_transactions table.

Hive: - select max(transaction_dt) from card_transactions;

BEFORE STREAMING - 2018-02-11 00:00:09

```
hive> select max(transaction_dt) from card_transactions;
Query ID = hadoop_20230108012636_d113f307-2408-4f8f-be2c-fb70bc3170a4
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1673139411187_0014)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.17 s
--------------------------------------------------------------------------------
OK
_c0
2018-02-11 00:00:09
Time taken: 9.303 seconds, Fetched: 1 row(s)
hive>
```

AFTER STREAMING - 2018-12-10 09:00:34

```
hive> select max(transaction_dt) from card_transactions;
Query ID = hadoop_20230108020942_ed66881c-69e6-482c-9197-3eba6068c19b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1673139411187_0016)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.36 s
--------------------------------------------------------------------------------
OK
_c0
2018-12-10 09:00:34
Time taken: 8.877 seconds, Fetched: 1 row(s)
hive>
```