Explanation of the solution to the streaming layer problem

I will first explain the solution approach in step by step:

Logics implemented in the Driver.py file

1. Read the Kafka hosted data using readstream method

```
lines = spark  \
        .readStream  \
        .format("kafka")  \
        .option("kafka.bootstrap.servers","18.211.252.152:9092")  \
        .option("subscribe","transactions-topic-verified")  \
        .option("startingOffsets", "earliest") \
        .load()
```

2. Defined schema for the incoming data

```
schema = StructType([
    StructField("card_id", StringType()),
    StructField("member_id", StringType()),
    StructField("amount", IntegerType()),
    StructField("pos_id", StringType()),
    StructField("postcode", StringType()),
    StructField("transaction_dt", StringType()),
])
```

3. Parsed the incoming Json data using the schema details

```
kafkaDF = lines.select(from_json(col("value").cast("string"), schema).alias("data")).select("data.*")
```

4. Created user defined functions to retrive the lookup transactions

© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved

```
def getDetails_Score(card_id):
    d = dao.HBaseDao.get_instance()
    data = d.get_data(card_id, table='look_up_table')
    return data['info:score']

def getDetails_UCL(card_id):
    d = dao.HBaseDao.get_instance()
    data = d.get_data(card_id, table='look_up_table',)
    return data['info:ucl']

def getDetails_Postcode(card_id):
    d = dao.HBaseDao.get_instance()
    data = d.get_data(card_id, table='look_up_table')
    return data['info:postcode']

def getDetails_transaction_dt(card_id):
    d = dao.HBaseDao.get_instance()
    data = d.get_data(card_id, table='look_up_table')
    return data['info:transaction_dt']
```

5. Registering the User defined functions

```
# Registering the User defined functions to retrieve the previous transactions data
details_score_udf = udf(getDetails_Score, StringType())
details_ucl_udf = udf(getDetails_UCL, StringType())
details_ucl_postcode = udf(getDetails_Postcode, StringType())
details_ucl_transactiondt = udf(getDetails_transaction_dt, StringType())

# Registering the User defined functions to write transactions data

write_ucl = udf(dao.write_data_ucl, StringType())
write_trans_hist = udf(dao.write_data_transaction, StringType())


# Registering the User defined functions to calculate the distance between the two post codes
details_long_udf = udf(geo_map.get_lat, DoubleType())
details_lat_udf = udf(geo_map.get_long, DoubleType())
details_dist_udf = udf(get_long.distance, DoubleType())

# Registering the User defined functions to perform the credit card rules
rules_score_udf = udf(checkUCL.get_lat, DoubleType())
rules_ucl_udf = udf(checkScore.get_long, DoubleType())
rules_dist_udf = udf(rules.distance, DoubleType())
rules_dist_udf = udf(rules.checkstatus, DoubleType())
```

6. Created new fields to hold the details from previous transactions

© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved

```
# Creating new columns using the UDF functions to hold the previous transaction from Hbase tables
kafkaDF= kafkaDF.withColumn('transaction_date', from_unixtime(unix_timestamp(kafkaDF.transaction_dt,'DD-MM-YYYY HH:MM:SS')).cast(TimestampType()))
kafkaDF = kafkaDF.withColumn('score', details_score_udf(kafkaDF.card_id))
kafkaDF = kafkaDF.withColumn('ucl', details_ucl_udf(kafkaDF.card_id))
kafkaDF = kafkaDF.withColumn('Previous_Post_Code', details_ucl_postcode(kafkaDF.card_id))
kafkaDF = kafkaDF.withColumn('Previous_trans_dt', details_ucl_transactiondt(kafkaDF.card_id))
kafkaDF = kafkaDF.withColumn('Previous_trans_dt_format', details_ucl_transactiondt(kafkaDF.card_id).cast(TimestampType()))
```

7. Created new fields to hold the details to derive distance between previous and current transactions

```
# Creating new columns using the UDF functions to hold the current and Previous post code distance
kafkaDF = kafkaDF.withColumn('Time_difference', abs(datediff(minute,Previous_trans_dt_format,transaction_date)))
kafkaDF = kafkaDF.withColumn('Current_Post_Code_Lattitude', details_lat_udf(kafkaDF.postcode))
kafkaDF = kafkaDF.withColumn('Previous_Post_Code_Lattitude', details_lat_udf(kafkaDF.Previous_Post_Code))
kafkaDF = kafkaDF.withColumn('Current_Post_Code_Longitude', details_lat_udf(kafkaDF.postcode))
kafkaDF = kafkaDF.withColumn('Previous_Post_Code_Longitude', details_lat_udf(kafkaDF.Previous_Post_Code))
kafkaDF = kafkaDF.withColumn('Distance', details_lat_udf(Current_Post_Code_Lattitude,Current_Post_Code_Longitude,Previous_Post_Code_Lattitude,Previous_Post_Code_Longitude))
```

8. Created new fields to hold the output of rules check flag either 1 or 0

```
##Creating new columns using the UDF functions to hold the rules check flag either 1 or 0
kafkaDF = kafkaDF.withColumn('Score_Chk_flag', details_ucl_udf(kafkaDF.score))
kafkaDF = kafkaDF.withColumn('UCL_Chk_flag', details_ucl_udf(kafkaDF.amount,kafkaDF.ucl,))
kafkaDF = kafkaDF.withColumn('Travel_distance_Chk_flag', details_ucl_udf(kafkaDF.Distance,kafkaDF.Time_difference))
```

9. Created new fields to hold status of rules check and also created two new field to have the row data to insert into Hbase UCL and Transaction History table

```
## Deriving Row for writing into HBASE tables
kafkaDF = kafkaDF.withColumn('Status', details_ucl_udf(kafkaDF.Score_Chk_flag,kafkaDF.UCL_Chk_flag,kafkaDF.Travel_distance_Chk_flag))
kafkaDF = kafkaDF.withColumn('Row_Hist', concat(kafkaDF.card_id,',',kafkaDF.member_id,',',kafkaDF.amount,',',kafkaDF.postcode,',',kafkaDF.pos_id,',',kafkaDF.transaction_dt,',',kafka
kafkaDF = kafkaDF.withColumn('Row_UCL', concat(kafkaDF.card_id,',',kafkaDF.UCL,',',kafkaDF.postcode,',',kafkaDF.transaction_dt,',',kafkaDF.score))
```

10. Finally created column to hold status of Hbase script and to write to console

```
## Calling the write function to load into HBASE

kafkaDF = kafkaDF.withColumn('Write_UCL_status', write_ucl(kafkaDF.Row_UCL))
kafkaDF = kafkaDF.withColumn('Write_Trans_Hist_status', write_trans_hist(kafkaDF.Row_Hist))

query = kafkaDF  \
        .writeStream  \
        .outputMode("append")  \
        .format("console")  \
        .option("truncate", "False")  \
        .start()

query.awaitTermination()
```

© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved

Logics implemented in the Rules.py file

1. **Implemented logic to detect card with score less than 200**

```
def checkUCL(amount, UCL):
    if(amount < UCL):
        return 1
    else:
        return 0
```

2. **Impelemented function to detect source amount is greater than UCL**

```
def uclcheck(ucl,amount):
    if (amount>ucl):
        return 1
    else:
        return 0
```

3. **Impelmented function to detect the distance travelled activity**

```
def checkdistancetravelled(distance,timediff):
    acceptable_distance_per_minute= 2
    distance_travelled_acceptable_range= timediff  * acceptable_distance_per_minute

    if distance > distance_travelled_acceptable_range:
        return 1
    else:
        return 0
```

4. **Implemented function to detect the overall status of the transactions**

```
def checkstatus(uclcheck,scorechk,travelchk):
    if (uclcheck+scorechk+travelchk) > 0:
        return 'Fraud'
    else:
        return 'Geniune'
```

© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved

Logics implemented in the Dao.py file

Along with existing function to retrive the UCL data, I have craeted two new function to write the data into Hbase for UCL and Transaction history data

1. Write_data_ucl to write into UCL table only when the row is Geninue

```
def write_data_ucl(self,status, row, table):
    for i in range(2):
        try:
            with self.pool.connection() as connection:
                t = connection.table('lookup_ucl_govind')
                vals = row.split(",")
                if status != 'Fraud':
                    t.put(bytes(vals[0] + vals[1] + vals[2] + vals[3], encoding='utf8'),
                              {b'cf:card_id': bytes(vals[0], encoding='utf8'),
                               b'cf:UCL': bytes(vals[1], encoding='utf8'),
                               b'cf:postcode': bytes(vals[2], encoding='utf8'),
                               b'cf:transaction_dt': bytes(vals[3], encoding='utf8'),
                               b'cf:score': bytes(vals[4], encoding='utf8')})
                    return 'Success'
        except:
            self.reconnect()
```

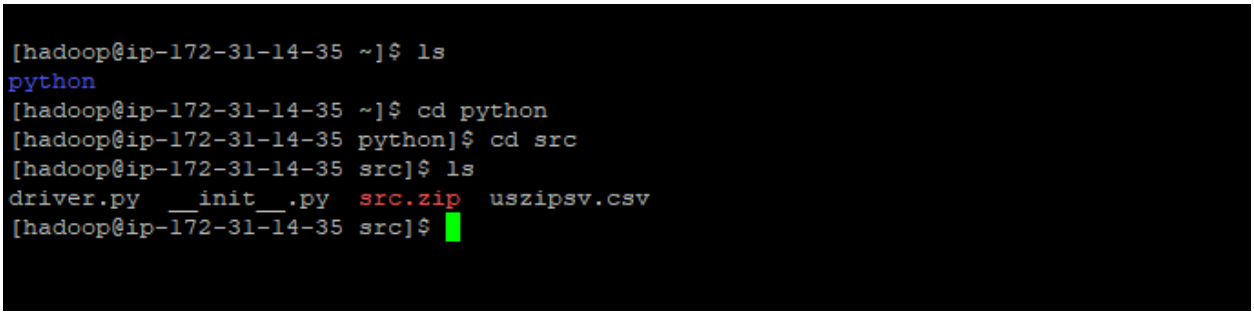
2. write_data_transaction to load transaction history data

```
def write_data_transaction(self, row, table):
    for i in range(2):
        try:
            with self.pool.connection() as connection:
                t = connection.table('card_transaction_history_govind')
                vals = line.split(",")
                if vals[0] != 'card_id':
                    t.put(bytes(vals[0] + vals[2] + vals[4] + vals[5], encoding='utf8'),
                              {b'cf:card_id': bytes(vals[0], encoding='utf8'),
                               b'cf:member_id': bytes(vals[1], encoding='utf8'),
                               b'cf:amount': bytes(vals[2], encoding='utf8'),
                               b'cf:postcode': bytes(vals[3], encoding='utf8'),
                               b'cf:pos_id': bytes(vals[4], encoding='utf8'),
                               b'cf:transaction_dt': bytes(vals[5], encoding='utf8'),
                               b'cf:status': bytes(vals[6], encoding='utf8')})
                    return 'Success'
        except:
            self.reconnect()
    def reconnect(self):
        self.pool = happybase.ConnectionPool(size=3, host=self.host)
```

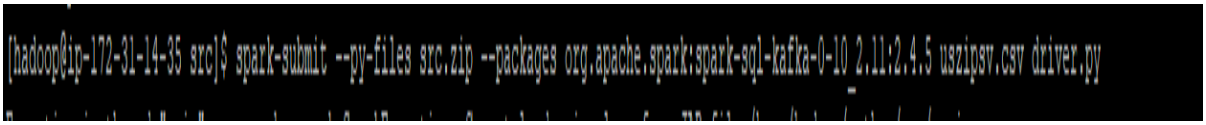
© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved

Scripts executed screen shot:

1. **Copied the python and zip files in to EMR cluster**



2. **Executed the python program**



© Copyright 2020. upGrad Education Pvt. Ltd. All rights reserved