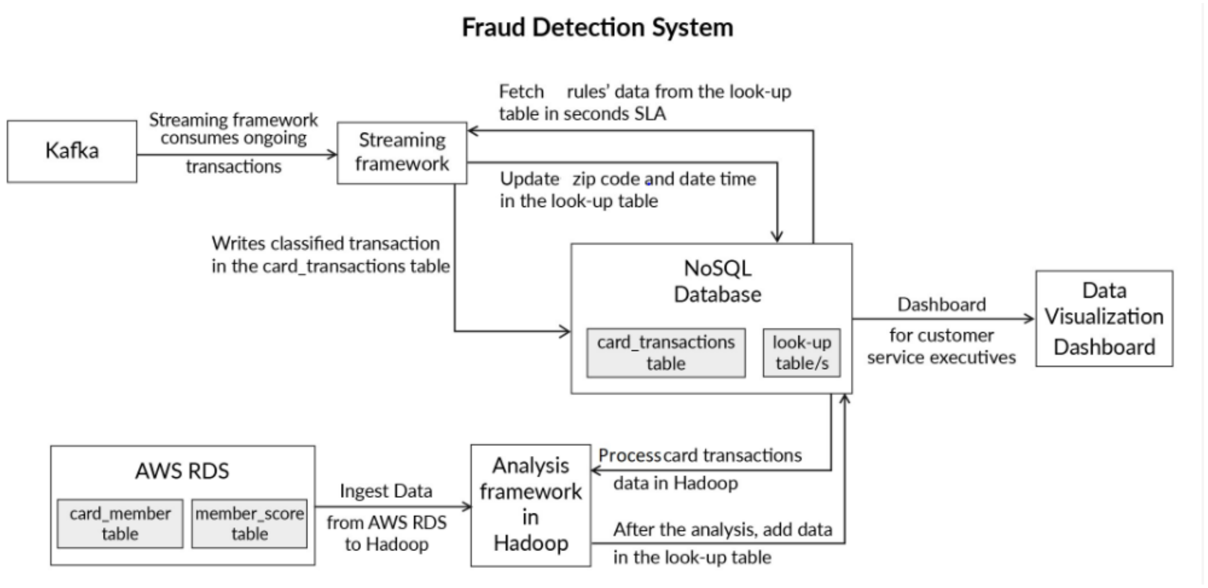
# LOGIC

## Capstone Project :

The Logic of the Entire project is explained below step by step.
The process and the screenshot for the MID-SUBMISSION was provided in the pdf in the name of Capstone Project - Doc (Mid Submission).pdf. Using the same process and explanation below.

**The Architecture of the Capstone project is shown below:**


Fraud Detection System
Kafka
Streaming framework consumes ongoing transactions
Streaming framework
Fetch rules' data from the look-up table in seconds SLA
Update zip code and date time in the look-up table
Writes classified transaction in the card_transactions table
NoSQL Database
card_transactions table
look-up table/s
Dashboard for customer service executives
Data Visualization Dashboard
AWS RDS
card_member table
member_score table
Ingest Data from AWS RDS to Hadoop
Analysis framework in Hadoop
Process card transactions data in Hadoop
After the analysis, add data in the look-up table

The Logic below is explained based on the Architecture shown above,

## Ingestion of Data from AWS RDS to HADOOP

Firstly the Dataset which was hosted on the Amazon RDS was imported to the MySQL Workbench by connecting with the RDS through Cloud.
The Database which is required for Ingestion from Relational database to Hadoop

(MySQL to HDFS) is **cred_financials_data**.

Database has the following tables – card_member and member_score
Import command for importing data from RDBMSs to the HDFS is shown below,

Table Name- **CARD_MEMBER**
Command Used: **sqoop import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table card_member**

Table Name- **MEMBER_SCORE**
Command Used: **sqoop import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table member_score**

Creating tables in HIVE to load the data

**Script:**

**1. create external table if not exists card_member_default(**

**`card_id` string,**

**`member_id` string,**

**`member_joining_dt` timestamp,**

**`card_purcharse_dt` string,**

**`country` string,**

**`city` string)**

**row format delimited fields terminated by ','**

**location 's3a://capstone31/card_member_default/';**

**2. create external table if not exists member_score_default(**

**`member_id` string,**

**`score` int)**

**row format delimited fields terminated by ','**

**location 's3a://capstone31/member_score_default/';**

**To load the MySQL table data to these two created tables the below load command is given in HIVE**
The card_member data is loaded to card_member_default and the member_score data is loaded to member_score_default respectively

**Commands:**

**1. load data inpath '/user/root/card_member/' overwrite into table card_member_default;**

**2. load data inpath '/user/root/member_score/' overwrite into table member_score_default;**

**The Data Ingestion from RDBMS to Hadoop has been successfully performed and validated in the OUTPUT.pdf**

## Load to NoSQL Database and Look-up table creation

The CSV File provided has the Historical data of each card id and this data needs to be loaded into a NoSQL database (HBASE).

Firstly the data is uploaded into HDFS and loaded into the table in HIVE.

**Scripts:**

Command to get the CSV file to the EC2

**wget https://cdn.upgrad.com/UpGrad/temp/32de2936-42f9-412f-a00c-2a0f400873cf/card_transactions.csv**

Now the data is loaded into HDFS and the we create an external table in HIVE using the below command

**create external table if not exists card_transactions_capstone (`card_id` string, `member_id` string, `Amount` Double, `postcode` INT, `pos_id` BIGINT,`transaction_dt` string, `status` string)**

**ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ("separatorChar" = ",")**
**LOCATION 'hdfs:/user/hdfs/card_details' tblproperties("skip.header.line.count"="1");**

The Data present in the HDFS path is now loaded into this external table using the command:

**LOAD DATA INPATH 'hdfs:/user/hdfs/card_transactions.csv' OVERWRITE INTO TABLE card_transactions_capstone;**

## Loading the Data to the HBASE table with HBASE-HIVE Integration:

Add the Jar in Hive for HIVE-HBASE mapping using the below command
**Add jar /opt/cloudera/parcels/CDH/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.1.0-cdh5.15.0.jar**

Create a HBASE table in ec2

**Command :**
**create 'card_transactions_hive', 'cf1', 'cf2' , 'cf3', 'cf4', 'cf5', 'cf6', 'cf7'**

Go to Hue to configure the Hbase Hive Integration for loading the data into this Hbase table

```
create external table card_transactions_hive_new(`unique_id` varchar(32), `card_id` string,
`member_id` string, `Amount` Double, `postcode` INT, `pos_id` BIGINT, `transaction_dt` string,
`status` string) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
SERDEPROPERTIES ("hbase.columns.mapping" =
":key,cf1:card_id,cf2:member_id,cf3:Amount,cf4:postcode, cf5:pos_id, cf6: transaction_dt, cf7:
status") TBLPROPERTIES ("hbase.table.name" = " card_transactions_hive'");
```

Once the table is created the data needs to be loaded for the HIVE-HBASE table

**Command:**

```
insert overwrite table card_transactions_hive_new

select concat(card_transactions.card_id+card_transactions.Amount) as unique_id,

card_transactions.card_id, card_transactions.member_id, card_transactions.Amount,
card_transactions.postcode, card_transactions.pos_id, card_transactions.transaction_dt,
card_transactions.status from card_transactions_capstone;
```

## The Data is finally Loaded in HBASE No SQL Database

Creating a LOOK-UP Table:

Create a HBASE table named lookup_hive

Command used: **create 'lookup_hive' , 'cf1' , 'cf2', 'cf3', 'cf4', 'cf5'**

**Go to HUE and create a HIVE-HBASE interface table for loading the data through batch for this HBASE look-up table**

```
create external table lookup_hbase(card_id bigint, UCL float, postcode bigint,
transaction_dt timestamp, score int)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,cf2:UCL,cf3:postcode,cf4:transaction_dt,
cf5:score")
TBLPROPERTIES ("hbase.table.name" = "lookup_hive");
```

The LOOK-UP table is successfully created.

## Script to Feed the Data for CARD_ID and UCL in the LOOK-UP table

The respective LOOK-UP table is created in the HBASE and we need to calculate the moving average and the Standard deviation of the last 10 transactions for each card_id in HIVE table - lookup_hbase and the HBASE table lookup_hive.

The below script is able to extract and feed the other relevant data for the look_up table along with card_id and UCL.

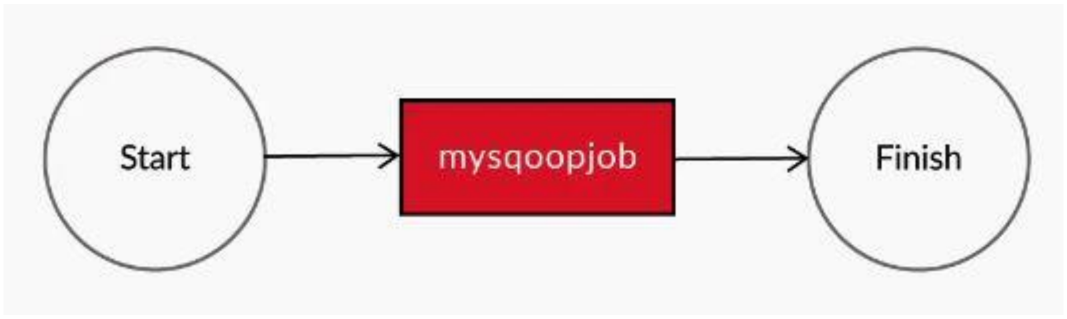
**Command for Inserting:**

```
insert overwrite table lookup_hbase
select card_id, (avg(amount) + 3*stddev(amount)) as UCL,
max( case when a=1 then postcode else 0 end ) as postcode,
max(transaction_dt) as transaction_dt,
max( case when a=1 then score else 0 end ) as score
from (select card_id, status, unix_timestamp(transaction_dt, "dd-MM-yyyy
HH:mm:ss")transaction_dt,
postcode, amount, score,
row_number() over (partition by card_id order by unix_timestamp(transaction_dt,
"dd-MM-yyyy HH:mm:ss") desc) a
from card_transactions_capstone join member on card_transactions_capstone.member_id =
member.member_id )b
where (a <= 10) and (status = "GENUINE") group by card_id;
```

Now the two Rules are processed for the analysis to be done to determine whether the transaction is GENIUNE or FRAUD.

## <u>Script to set up Job Scheduler after 4hours and feed the Look-Up table</u>

### DAG of a SQOOP ACTION



The time trigger coordinator  activates the workflow at a specific time and frequency.
The workflow then contains the set of orders to be performed in a specific order.

## <u>RULES</u>

**<u>Rule 1:</u>**

**Upper Control Limit**

The formula of UCL=(Moving Average)+3×(Standard Deviation)

Is used to derive the UCL value of each card_id.
This parameter is basically an indicator representing the transaction pattern of each customer.

**<u>The Above rule is calculated in the look_up table all we need to do is use the value of the UCL by mapping the card_id</u>**

**Rule 2:**

**Credit score of each member**

The Credit score is simply taken from the member score table

If the Score is less than 200 then their transaction is regarded as Fraudulent.

**The Above rule is can be taken from the member_score table**

**Rule 3:**

**Zip Code Distance**

The purpose to track the distance of the current transaction and the previous transaction
The previous transaction are the Historical data and the current transaction are present in KAFKA and those have to be ingested real time using the Spark streaming application

**The Above rule is yet to be calculated**

## Real Time Ingestion and Classification of transaction

### Consumption of Ongoing transactions suing the Streaming Framework

### Ingestion of Real time data from KAFKA:

The Below class stores the transaction details from KAFKA Streams
public class JsonTransaction implements Serializable {

The Spark streaming application which consumes data from Kafka and then this data which is in JSON format is processed and classifies the transaction into GENIUNE or FRAUD based on three rules.

Further Analysis has to be done by fetching the rules from the look up table.

Data has to be fetched from the Look-Up table and then store some values in the Member look-up table

**Kafka D-streams are created which will consume the process the ongoing**
**This Fetches the Card transaction data**

The HBASE Connector is used to connect the HBASE with the eclipse so that we can run the process in the EC2 instance from which data in the HBASE table will be used for the analysis.

This fetches the rules from the HBASE LOOKUP table so that we can update Zip_code, Date and Time in the Look_Up table

public static Admin HbaseConnection() {

Creating a HBASE connection

**Rule 3:**

Then the Distance is calculated by the below piece of Code

```
/*
                           * Calculating the time difference between two
transactions and getting value
                           * for distance that can be covered in this period.
                           */
                          long Difference = (date.getTime() / 1000 -
last_date.getTime() / 1000);
                          long permitted_distance = Difference / 4;

                          String code = new String(postcode, "UTF-8");

                          /*
                           * Finding out the distance between postcode of current
transaction and previous
                           * transactions. This will help us to identify whether
given transaction is
                           * fraud or not.
                           */

                          double dist = disUtil.getDistanceViaZipCode(code,
convertstock.getpostcode().toString());
```

**The Above rule is calculates the Zip code distance**

## Analysis of Rules and updating the tables in HBASE:

The Purpose of the below code is to classify whether the transaction is GENUINE or FRAUD for the card _id based on the three rules

```
if ((Double.parseDouble(convertstock.getamount()) < limit) && (score > 200) &&
(dist < permitted_distance)) {
                                 System.out.println("Limit = " + limit + " score =
" + score + " distance =" + dist);
                                 /*
                                  * If transaction is genuine we are inserting
transaction as GENUINE and
                                  * updating look up table
                                  */
```

```
                                String status = "GENUINE";
                                p.~~add~~(Bytes.toBytes("cf7"),
Bytes.toBytes("status"), Bytes.toBytes(status));
                                transactions_table.put(p);

                                u.~~add~~(Bytes.toBytes("cf8"),
Bytes.toBytes("transaction_date"),

        Bytes.toBytes(convertstock.gettransaction_dt().toString()));
                                u.~~add~~(Bytes.toBytes("cf7"),
Bytes.toBytes("postcode"),

        Bytes.toBytes(convertstock.getpostcode().toString()));
                                table.put(u);
                                return 1;
                        } else {
                                /*
                                 * If transaction is fraud then we are updating
only transaction table
                                 */
                                String status = "FRAUD";
                                p.~~add~~(Bytes.toBytes("cf7"),
Bytes.toBytes("status"), Bytes.toBytes(status));
                                transactions_table.put(p);
                                return 0;
                        }
                } else {
                        System.out.println("Null value received");
```

If All the three rules are satisfied then the transaction is said to be GENUINE else it is deemed as a FRAUD transaction.

## FINAL Card Transactions CSV export

The values are updated in the CARD_TRANSACTION_HIVE (HBASE table) and the updated values are visualized for the customer service executives are exported in the form of CSV file.

The Status of each card_id is updated in that CSV file.

******************************END*****************************************