Logic For First Submission

1. We started by creating python file spark_kafka_to_local.py with the following command

   vi spark_kafka_to_local.py

2. Here is the code we wrote to load kafka data

```
# Importing the modules
import sys
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.functions import from_json

# Creating spark session
spark = SparkSession.builder \
    .master("local") \
    .appName("Kafka To HDFS") \
    .getOrCreate()

spark.sparkContext.setLogLevel('ERROR')

# Creating Dataframe from Kafka Data
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("subscribe", "de-capstone3") \
    .option("startingOffsets", "earliest") \
    .load()

# Transforming dataframe by changing value columns' data type and dropping
some columns
df = df \
    .withColumn('value_str', df['value'].cast('string').alias('key_str')).drop('value') \
    .drop('key','topic','partition','offset','timestamp','timestampType')

# Writing the dataframe to local file directory and keep it running until termination
df.writeStream \
    .outputMode("append") \
    .format("json") \
    .option("truncate","false") \
    .option("path", "clickstream_data") \
    .option("checkpointLocation","clickstream_checkpoint") \
    .start() \
    .awaitTermination()
```

3. Thereafter, we run the spark submit command

spark2-submit --jars "spark-sql-kafka-0-10_2.11-2.3.0.jar" spark_kafka_to_local.py

4. Then we created another python file to clean the kafka data and then run the spark submit command

vi spark_local_flatten.py

```
# Importing modules
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Creating spark session
spark = SparkSession.builder \
    .master("local") \
    .appName("Kafka To HDFS") \
    .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Reading json data into dataframe
df = spark.read.json('clickstream_data/part-00000-09f8130f-4a14-890f-d28b-
fec98641896b-c000.json')

# Extracting columns from json values and creating new dataframe with new
columns
df = df.select( \
            get_json_object(df["value_str"],"$.customer_id").alias("customer_id"),\
            get_json_object(df["value_str"],"$.app_version").alias("app_version"),\
            get_json_object(df["value_str"],"$.OS_version").alias("OS_version"),\
            get_json_object(df["value_str"],"$.lat").alias("lat"),\
            get_json_object(df["value_str"],"$.lon").alias("lon"),\
            get_json_object(df["value_str"],"$.page_id").alias("page_id"),\
            get_json_object(df["value_str"],"$.button_id").alias("button_id"),\

get_json_object(df["value_str"],"$.is_button_click").alias("is_button_click"),\
            get_json_object(df["value_str"],"$.is_page_view").alias("is_page_view"),\
            get_json_object(df["value_str"],"$.is_scroll_up").alias("is_scroll_up"),\

get_json_object(df["value_str"],"$.is_scroll_down").alias("is_scroll_down"),\
            get_json_object(df["value_str"],"$.timestamp").alias("timestamp"),\
           )

# Saving the dataframe as a csv file in local directory
df.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save('use
r/root/clickstream_data_flatten', header = 'true')
```

spark2-submit --jars "spark-sql-kafka-0-10_2.11-2.3.0.jar" spark_kafka_to_local.py

5. Then we imported data from AWS RDS to Hadoop via sqoop

```
sqoop import \
Connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-
1.rds.amazonaws.com/testdatabase \
--table bookings \
--username student \
--password STUDENT123 \
--target-dir /user/root/bookings_data \
-m 1
```

6. Then we created aggregation table and then run the spark submit command

vi datewise_bookings_aggregates_spark.py

```
# Importing the modules
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json
from pyspark.sql.types import TimestampType, IntegerType, FloatType,
ArrayType,LongType
from pyspark.sql import functions as func

# Initializing Spark Session
spark = SparkSession  \
    .builder  \
    .appName("StructuredSocketRead")  \
    .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Defining the schema
schema_agg = StructType([
        StructField("booking_id", StringType()),
        StructField("customer_id",  LongType()),
        StructField("driver_id",  LongType()),
        StructField("customer_app_version", StringType()),
        StructField("customer_phone_os_version", StringType()),
        StructField("pickup_lat", DoubleType()),
        StructField("pickup_lon", DoubleType()),
        StructField("drop_lat", DoubleType()),
        StructField("drop_lon", DoubleType()),
        StructField("pickup_timestamp", TimestampType()),
        StructField("drop_timestamp", TimestampType()),
        StructField("trip_fare", IntegerType()),
        StructField("tip_amount", IntegerType()),
        StructField("currency_code", StringType()),
        StructField("cab_color", StringType()),
```

```
        StructField("cab_registration_no", StringType()),
        StructField("customer_rating_by_driver", IntegerType()),
        StructField("rating_by_customer", IntegerType()),
        StructField("passenger_count", IntegerType())
        ])

# Reading the file "part-m-00000" in bookings_data folder in hadoop
df=spark.read.csv("bookings_data/part-m-00000", schema=schema_agg)

# Creating data from column "pickup_date" and "pickup_timestamp"
df = df.withColumn("pickup_date", func.to_date(func.col("pickup_timestamp")))

# Group the data by "pickup_date"
date = df.groupBy('pickup_date').count()

# Saving the datewise total booking data in .csv format
date.coalesce(1).write.format('csv').save("date_aggregated_data/")

# Saving the booking data in .csv format
df.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save('boo
king_data_csv', header = 'true')
```

spark2-submit --jars "spark-sql-kafka-0-10_2.11-2.3.0.jar"
datewise_bookings_aggregates_spark.py

7. Then we created the Hive managed tables to retrieve required data

a. CREATING DATABASE

```
create database if not exists cab_booking;
use cab_booking;
```

b. CREATING HIVE TABLE FOR CLICKSTREAM DATA

```
create table if not exists clickstream_data (
customer_id int,
app_version string,
os_version string,
lat string,
lon string,
page_id string,
button_id string,
is_button_click varchar(3),
is_page_view varchar(3),
is_scroll_up varchar(3),
is_scroll_down varchar(3)
)
```

row format delimited fields terminated by “,”;

c. CREATING HIVE TABLE FOR BOOKINGS DATA

```
create table if not exists booking_data(
booking_id varchar(255),
customer_id int,
driver_id int,
customer_app_version varchar(255),
customer_phone_os_version string,
pickup_lat double,
pickup_lon double,
drop_lat double,
drop_lon double,
pickup_timestamp timestamp,
drop_timestamp timestamp,
trip_fare int,
tip_amount int,
currency_code string,
cab_color string,
cab_registration_no varchar(255),
customer_rating_by_driver int,
rating_by_customer int, passenger_count int
)
row format delimited fields terminated by “,”;
```

d. CREATING HIVE TABLE FOR AGGREGATED DATA

```
create table if not exists datewise_data (
date string,
count int
)
row format delimited fields terminated by “,”;
```

8. Then, we loaded the data int =o these hive tables

a. LOADING DATA INTO ‘clickstream_data’ HIVE TABLE

```
load data inpath 'clickstream_data_flatten/ part-00000-c0685d2e-b1ef-400f-df04-
901aafe4d6771c5c-c000.csv ' into table clickstream_data;
```

b. LOADING DATA INTO ‘booking_data’ HIVE TABLE

```
load data inpath 'bookings_data / part-00000-26e64217-57ec-e6b6-5a2a-
```

4f1ee2a582e5bd2d-c000.csv' into table booking_data;

c. LOADING DATA INTO ‘datewise_data’ HIVE TABLE

load data inpath 'date_aggregated_data/ part-00000-24b5db76-44d4-249b-efb45054190b9483-c000.csv' into table datewise_data;