

Logic For First Submission

Problem Statement

YourOwnCabs (YOC), a thriving on-demand taxi service, initially managed around 100 rides daily. The refined business strategy and stellar customer service, the company is experiencing swift growth, with ride numbers consistently surpassing previous records. There is a daily increase in both YOC's clientele and ride frequency.

For YOC's decision-makers, it's critical to quickly access real-time data for informed strategic planning. However, the rapid expansion has led to challenges in data retrieval, as the current backend MySQL database struggles with the query load from the burgeoning data volumes.

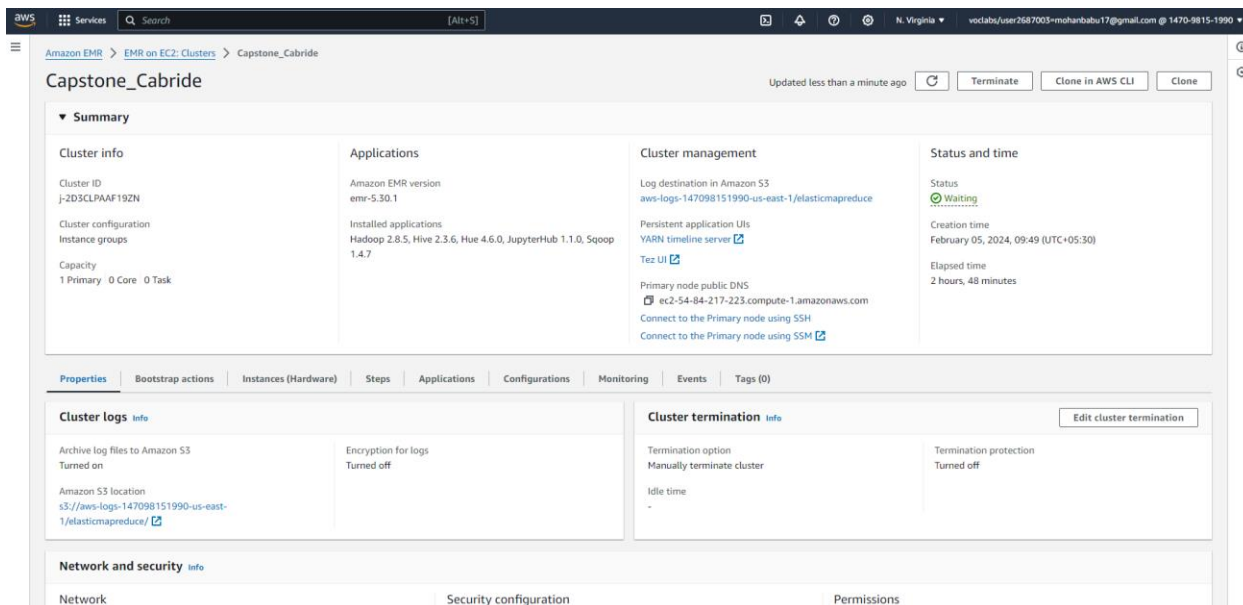
Strategic Solutions Proposed:

As a Big Data specialist, it's our responsibility to design and deploy a data-driven analytical system that can dissect the extensive booking and usage data, providing actionable insights for strategic business decisions.

Enhanced Booking Data Analysis: We propose a system where ride-booking information is stored efficiently, enabling comprehensive analytics without disrupting operational workflows. This system will offer insights through various metrics, including daily, weekly, and monthly ride bookings, categorizations by mobile OS, average fare, total gratuities received, and more.

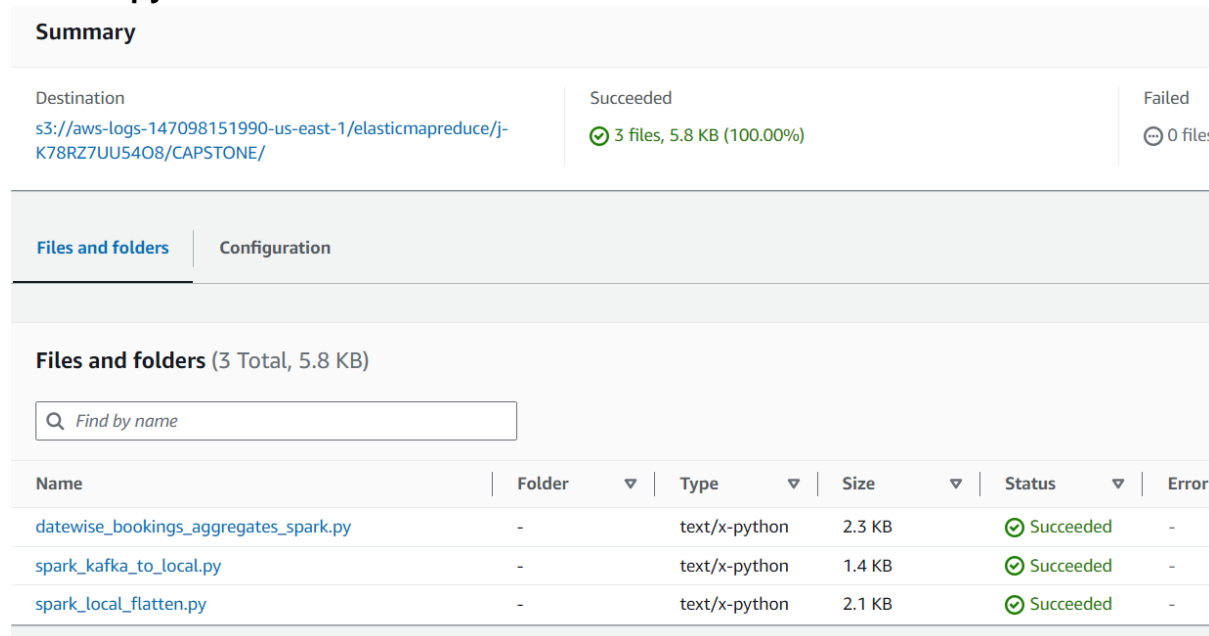
Advanced Click-Stream Data Analysis: The proposed solution will also analyze click-stream data, which is the digital footprint users leave when navigating the app. This encompasses actions such as link and button clicks, as well as screen views. Analyzing this data will reveal user intent and interaction patterns, allowing us to refine user engagement strategies. Given the substantial volume of click-stream data, it necessitates a robust architecture and meticulous storage practices to facilitate effective analysis.

An EMR instance was established, integrating technologies such as Hadoop, Sqoop, Hive, Hue, and Spark, to facilitate the execution of the Cab Ride Capstone Project



The screenshot shows the AWS Management Console interface for an Amazon EMR cluster. The cluster is named 'Capstone_Cabride' and is in the 'Waiting' status. The console displays various tabs for cluster management, including Summary, Properties, Bootstrap actions, Instances (Hardware), Steps, Applications, Configurations, Monitoring, Events, and Tags (0). The Summary tab is active, showing cluster information, applications, cluster management, and status and time. The cluster is located in the us-east-1 region and is using the emr-5.30.1 version. It has 1 Primary instance and 0 Core instances. The cluster is configured with Hadoop 2.8.5, Hive 2.3.6, Hue 4.6.0, JupyterHub 1.1.0, and Sqoop 1.4.7. The cluster is managed by the YARN timeline server and Tez UI. The status and time section shows that the cluster was created on February 05, 2024, at 09:49 (UTC+05:30) and has elapsed 2 hours and 48 minutes.

Load all python file in S3 bucket:



The screenshot shows the AWS Management Console interface for an Amazon EMR cluster. The cluster is named 'Capstone_Cabride' and is in the 'Waiting' status. The console displays various tabs for cluster management, including Summary, Properties, Bootstrap actions, Instances (Hardware), Steps, Applications, Configurations, Monitoring, Events, and Tags (0). The Summary tab is active, showing cluster information, applications, cluster management, and status and time. The cluster is located in the us-east-1 region and is using the emr-5.30.1 version. It has 1 Primary instance and 0 Core instances. The cluster is configured with Hadoop 2.8.5, Hive 2.3.6, Hue 4.6.0, JupyterHub 1.1.0, and Sqoop 1.4.7. The cluster is managed by the YARN timeline server and Tez UI. The status and time section shows that the cluster was created on February 05, 2024, at 09:49 (UTC+05:30) and has elapsed 2 hours and 48 minutes.

Step 1: Ingesting Streaming Data from Kafka into HDFS

Ingesting and Persisting Kafka Streams as JSON Files in the Local HDFS Directory

Event data from the cab booking application are transmitted to the de-capstone3 Topic on a Bootstrap server located at 18.211.252.152, port 9092. Our objective is to process these streaming data and transfer them into a Hive-managed table for subsequent analytical purposes. Load all files in S3 bucket called “cabride”. The following are the detailed procedures to retrieve data from the Kafka topic and deposit it into the Hive table:

Retrieval of Data from kafka.bootstrap.servers:

A PySpark script named “spark_kafka_to_local.py” has been developed to fetch the streaming data from the Kafka server.

```
#Importing required libraries
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_161/jre"
os.environ["SPARK_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initializing Spark session/application
spark = SparkSession \
    .builder \
    .appName("Kafka-to-local") \
    .getOrCreate()

#Reading Streaming data from de-capstone3 kafka topic
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("startingOffsets", "earliest") \
    .option("subscribe", "de-capstone3") \
    .load()

df= df \
    .withColumn('value_str',df['value'].cast('string').alias('key_str')).drop('value') \
    .drop('key','topic','partition','offset','timestamp','timestampType')

#Writing data from kakfa to local file
df.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("path", "/user/root/clickstream_data_dump") \
    .option("checkpointLocation", "/user/root/clickstream_data_dump_cp") \
    .start() \
    .awaitTermination()
```

Running the Script to Relocate Streaming Data to the Local HDFS Directory:

A series of commands were executed to offload the data into the local directory, confirm the creation of the file, and check the integrity of the records in the JSON file.

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 s3://aws-logs-147098151990-us-east-1/elasticmapreduce/j-3P5AA3ER77VJN/cabride/spark_kafka_to_local.py
```

Kafka stream reading

[illegible]

check the json files dump in belwo dir:

```
hadoop fs -ls /user/root/clickstream_data_dump
```

```
hadoop@ip-172-31-76-175 ~ (2.361s)
hadoop fs -ls /user/root/clickstream_data_dump

Found 2 items
drwxr-xr-x    - hadoop hadoop          0 2024-02-03 18:30 /user/root/clickstream_data_dump/_spark_metadata
-rw-r--r--    1 hadoop hadoop    1267706 2024-02-03 18:30 /user/root/clickstream_data_dump/part-00000-cc409037-2aeb-4a29-9b54-f3aacbce09c3-c000.json

hadoop@ip-172-31-76-175 ~
```

Check top 5 json file data:

```
hadoop fs -cat /user/root/clickstream_data_dump/part-00000-e2d31120-6de1-428d-8b62-b1808ff4a5f0-c000.json | head -n 5
```

```
hadoop@ip-172-31-76-175 ~ (3.534s)
hadoop fs -cat /user/root/clickstream_data_dump/part-00000-cc409037-2aeb-4a29-9b54-f3aacbce09c3-c000.json | head -n 5
{"value_str":{"customer_id":"26564820","app_version":"3.2.35","os_version":"Android","lat":"16.4454865","lon":"99.902065","page_id":"de545711-3914-4450-8c11-b17b8dabb5e1","button_id":"fcb68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"Yes","is_scroll_up":"No","is_scroll_down":"Yes","timestamp":"2020-09-14 09:59:07\n\n"}}
{"value_str":{"customer_id":"31906387","app_version":"2.4.7","os_version":"iOS","lat":"-64.813749","lon":"-133.527040","page_id":"de545711-3914-4450-8c11-b17b8dabb5e1","button_id":"a95dd57b-779f-49db-819d-b6960483e554","is_button_click":"No","is_page_view":"No","is_scroll_up":"Yes","is_scroll_down":"Yes","timestamp":"2020-05-16 16:30:21\n\n"}}
{"value_str":{"customer_id":"25713677","app_version":"3.4.12","os_version":"Android","lat":"89.943435","lon":"127.313415","page_id":"b328829e-17ae-11eb-adc1-0242ac120002","button_id":"fcb68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"No","is_scroll_up":"Yes","is_scroll_down":"No","timestamp":"2020-02-09 00:52:13\n\n"}}
{"value_str":{"customer_id":"83474293","app_version":"3.1.8","os_version":"Android","lat":"-69.939070","lon":"-36.451670","page_id":"e7bc5fb2-1231-11eb-adc1-0242ac120002","button_id":"e1e99492-17ae-11eb-adc1-0242ac120002","is_button_click":"Yes","is_page_view":"No","is_scroll_up":"Yes","is_scroll_down":"No","timestamp":"2020-06-17 10:42:50\n\n"}}
{"value_str":{"customer_id":"63727807","app_version":"2.2.9","os_version":"iOS","lat":"64.082108","lon":"-81.822078","page_id":"e7bc5fb2-1231-11eb-adc1-0242ac120002","button_id":"fcb68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"Yes","is_scroll_up":"Yes","is_scroll_down":"Yes","timestamp":"2020-07-06 02:51:53\n\n"}}
cat: Unable to write to output stream.
```

Processing the .json File from HDFS for Conversion into a Structured .csv Format

The .json dump file generated previously contains extraneous data which necessitates cleaning prior to its integration into a Hive managed table for in-depth analysis. For this purpose, the PySpark script “spark_local_flatten.py” has been developed to refine the data into a structured format suitable for loading into a .csv file within the HDFS directory, subsequently facilitating its import into the Hive table.

Enhancing JSON to Formatted CSV Conversion:

The script “spark_local_flatten.py” is designed to process the raw streaming data from the .json file, structuring it into a more organized .csv format before committing it to the HDFS directory.

```
# Importing required libraries
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_232-cloudera/jre"
os.environ["SPARK_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession
from pyspark.sql.functions import *

#Initializing Spark session/application
spark=SparkSession.builder.appName("Kafka-to-HDFS").master("local").getOrCreate()
spark

#Reading data from hdfs
df=spark.read.json("/user/root/clickstream_data_dump/part-00000-3865fbc8-b81c-4ba6-ba1c-2175a0868351-c000.json")

df.show(10,truncate=False)
```

```
#Selecting the columns from the clickstream data set and assigning alias
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")
            )

# Filter out rows where customer_id is not null or empty
df = df.filter((col("customer_id").isNotNull() & (col("customer_id") != "")))

#validating schema
df.printSchema()

#Validating records in top 10 records
df.show(10)

#Writing the format final dataset to hdfs
df.coalesce(1).write.format('csv').mode('overwrite').save('/user/root/clickstream_flattened',header='true')
```

Script Execution for Data Structuring and HDFS Uploading:

A sequence of specified commands has been employed to transmute the .json file into a more orderly .csv format, ready for storage in the local HDFS directory.

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 s3://aws-logs-147098151990-us-east-1/elasticmapreduce/j-3EXAGRFFL6LY6/cabride/spark_local_flatten.py
```

```
hadoop@ip-172-31-76-175 ~ (15.532s)
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark_local_flatten.py

Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-sql-kafka-0-10_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-49929c89-bb68-479a-a4b0-636b875e47f5;1.0
  confs: [default]
    found org.apache.spark#spark-sql-kafka-0-10_2.11:2.4.5 in central
    found org.apache.kafka#kafka-clients;2.0.0 in central
    found org.lz4#lz4-java;1.4.0 in central
    found org.xerial.snappy#snappy-java;1.1.7.3 in central
    found org.slf4j#slf4j-api;1.7.16 in central
    found org.spark-project.spark#unused;1.0.0 in central
:: resolution report :: resolve 411ms :: artifacts dl 11ms
:: modules in use:
  org.apache.kafka#kafka-clients;2.0.0 from central in [default]
  org.apache.spark#spark-sql-kafka-0-10_2.11:2.4.5 from central in [default]
  org.lz4#lz4-java;1.4.0 from central in [default]
  org.slf4j#slf4j-api;1.7.16 from central in [default]
  org.spark-project.spark#unused;1.0.0 from central in [default]
  org.xerial.snappy#snappy-java;1.1.7.3 from central in [default]
-----
|               | modules | artifacts |
| conf | number | search | dwnlded | evicted | number | dwnlded |
|-----|-----|-----|-----|-----|-----|-----|
| default | 6 | 0 | 0 | 0 | 6 | 0 |
-----
:: retrieving :: org.apache.spark#spark-submit-parent-49929c89-bb68-479a-a4b0-636b875e47f5
  confs: [default]
  0 artifacts copied, 6 already retrieved (0KB/10ms)
24/02/03 18:48:01 INFO SparkContext: Running Spark version 2.4.5-amzn-0
24/02/03 18:48:01 INFO SparkContext: Submitted application: Kafka-to-HDFS
24/02/03 18:48:01 INFO SecurityManager: Changing view acls to: hadoop
24/02/03 18:48:01 INFO SecurityManager: Changing modify acls to: hadoop
24/02/03 18:48:01 INFO SecurityManager: Changing view acls groups to:
24/02/03 18:48:01 INFO SecurityManager: Changing modify acls groups to:
24/02/03 18:48:01 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop); groups with view permissions: Set(); users with modify permissions: Set(hadoop); groups with modify permissions: Set()
24/02/03 18:48:01 INFO Utils: Successfully started service 'sparkDriver' on port 44669.
24/02/03 18:48:01 INFO SparkEnv: Registering MapOutputTracker
24/02/03 18:48:01 INFO SparkEnv: Registering BlockManagerMaster
24/02/03 18:48:01 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
24/02/03 18:48:01 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
24/02/03 18:48:01 INFO DiskBlockManager: Created local directory at /mnt/tmp/blockmgr-23e43487-0b5c-450d-b25a-bbf012683145
```


Check file exist:

```
hadoop fs -ls /user/root/clickstream_flattened
```

```
hadoop@ip-172-31-76-175 ~ (2.579s)
hadoop fs -ls /user/root/clickstream_flattened
Found 2 items
-rw-r--r-- 1 hadoop hadoop 0 2024-02-03 18:48 /user/root/clickstream_flattened/_SUCCESS
-rw-r--r-- 1 hadoop hadoop 403841 2024-02-03 18:48 /user/root/clickstream_flattened/part-00000-6a454a06-57eb-4f41-8c69-bd7017f10398-c000.csv
```

Validate file data:

```
hadoop fs -cat clickstream_flattened/part-00000-65875598-48b5-4f6b-b2df-e8e32020a98a-c000.csv | head -n 5
```

```
hadoop@ip-172-31-76-175 ~ (3.534s)
hadoop fs -cat clickstream_data_dump/part-00000-cc409037-2aeb-4a29-9b54-f3aacbce09c3-c000.json | head -n 5
{"value_str":{"customer_id":"26564820","app_version":"3.2.35","OS_version":"Android","lat":"16.4454865","lon":"99.902065","page_id":"de545711-3914-4450-8c11-b17b8dabb5e1","button_id":"fcba68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"Yes","is_scroll_down":"No","is_scroll_down":"Yes","timestamp":"2020-09-14 09:59:07"},"value_str":{"customer_id":"31906387","app_version":"2.4.7","OS_version":"iOS","lat":"-64.813749","lon":"-133.527040","page_id":"de545711-3914-4450-8c11-b17b8dabb5e1","button_id":"a95dd57b-779f-49db-819d-b6960483e554","is_button_click":"No","is_page_view":"No","is_scroll_down":"Yes","is_scroll_down":"Yes","timestamp":"2020-05-16 16:30:21"},"value_str":{"customer_id":"25713677","app_version":"3.4.12","OS_version":"Android","lat":"89.943435","lon":"127.313415","page_id":"b328829e-17ae-11eb-adc1-0242ac120002","button_id":"fcba68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"No","is_scroll_down":"No","is_scroll_down":"No","timestamp":"2020-02-09 00:52:13"},"value_str":{"customer_id":"83474293","app_version":"3.1.8","OS_version":"Android","lat":"-69.939070","lon":"-36.451670","page_id":"e7bc5fb2-1231-11eb-adc1-0242ac120002","button_id":"e1e99492-17ae-11eb-adc1-0242ac120002","is_button_click":"Yes","is_page_view":"No","is_scroll_down":"No","is_scroll_down":"No","timestamp":"2020-06-17 10:42:50"},"value_str":{"customer_id":"63727807","app_version":"2.2.9","OS_version":"iOS","lat":"64.082108","lon":"-81.822078","page_id":"e7bc5fb2-1231-11eb-adc1-0242ac120002","button_id":"fcba68aa-1231-11eb-adc1-0242ac120002","is_button_click":"No","is_page_view":"Yes","is_scroll_down":"Yes","is_scroll_down":"Yes","timestamp":"2020-07-06 02:51:53"},"cat: Unable to write to output stream.
```

Step – 2: Load data from AWS RDS to Hadoop

Data Ingestion with Sqoop: We need MySQL connector installed to make a bridge in between MySQL database and targeted location to retrieve required dataset from MySQL.

Commands executed to install MySQL connector:

```
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/
```

```
hadoop@ip-172-31-76-175 ~ (0.276s)
wget https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
--2024-02-03 19:00:18-- https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
Resolving de-mysql-connector.s3.amazonaws.com (de-mysql-connector.s3.amazonaws.com)... 52.216.54.9, 52.216.63.73, 52.216.138.219, ...
Connecting to de-mysql-connector.s3.amazonaws.com (de-mysql-connector.s3.amazonaws.com)|52.216.54.9|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4079310 (3.9M) [application/x-gzip]
Saving to: 'mysql-connector-java-8.0.25.tar.gz'

100%[=====] 4,079,310 --.-K/s in 0.1s

2024-02-03 19:00:19 (35.9 MB/s) - 'mysql-connector-java-8.0.25.tar.gz' saved [4079310/4079310]

hadoop@ip-172-31-76-175 ~ (0.182s)
tar -xvf mysql-connector-java-8.0.25.tar.gz
mysql-connector-java-8.0.25/
mysql-connector-java-8.0.25/src/
mysql-connector-java-8.0.25/src/build/
mysql-connector-java-8.0.25/src/build/java/
mysql-connector-java-8.0.25/src/build/java/documentation/
mysql-connector-java-8.0.25/src/build/java/instrumentation/
mysql-connector-java-8.0.25/src/build/misc/
mysql-connector-java-8.0.25/src/build/misc/debian.in/
```

Executed sqoop import command to ingest the bookings table data from AWS RDS to HDFS location:

Sqoop Import executed Code Screenshot:-

```
sqoop import \
--connect jdbc:mysql://upgradtest.cyaieic9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
--table bookings \
--username student --password STUDENT123 \
--target-dir /user/root/bookings \
-m 1
```

```
hadoop@ip-172-31-76-175 ~/mysql-connector-java-8.0.25 (27.509s)
sqoop import \
> --connect jdbc:mysql://upgradtest.cyaieic9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
> --table bookings \
> --username student --password STUDENT123 \
> --target-dir /user/root/bookings \
> -m 1

Warning: /usr/lib/sqoop/./accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
24/02/03 19:12:35 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/redshift/jdbc/redshift-jdbc42-1.2.37.1061.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
24/02/03 19:12:35 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
24/02/03 19:12:35 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
24/02/03 19:12:35 INFO tool.CodeGenTool: Beginning code generation
Loading class 'com.mysql.jdbc.Driver'. This is deprecated. The new driver class is 'com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and m
```

Check ingested file:

```
hadoop fs -ls /user/root/bookings
```

```
hadoop@ip-172-31-76-175 ~/mysql-connector-java-8.0.25 (2.295s)
hadoop fs -ls /user/root/bookings

Found 2 items
-rw-r--r-- 1 hadoop hadoop 0 2024-02-03 19:12 /user/root/bookings/_SUCCESS
-rw-r--r-- 1 hadoop hadoop 165678 2024-02-03 19:12 /user/root/bookings/part-m-000000
```

Check top 5 data:

```
hadoop fs -cat /user/root/bookings/part-m-000000 | head -n 5
```



```
hadoop@lg-172-31-76-175 ~/mysql-connector-java-8.0.25 (2.628s)
hadoop fs -cat /user/root/bookings/part-m-00000 | head -n 5
BK8968087150,51811359,15055660,2.2.14,Android,-49.4319655,103.917851,-58.8043875,146.477367,2020-06-23 19:33:10.0,2020-06-06 09:02:10.0,534,83,INR,black,054-38-4479,4,3,3
BK629851904,31663218,60872180,3.4.1,iOS,-83.5408405,175.80085,86.20705,128.367238,2020-05-23 12:22:04.0,2020-08-09 19:02:56.0,126,67,INR,lime,796-39-6801,3,2,4
BK1797410350,86869399,94276051,4.1.36,iOS,-67.8930645,55.234128,-51.1079,-31.07475,2020-05-19 14:14:32.0,2020-08-23 18:38:39.0,297,63,INR,olive,748-73-1579,1,3,3
BK5788246325,58230837,45457227,2.4.27,Android,13.707887,113.499943,54.3812915,-18.437751,2020-03-24 01:30:15.0,2020-05-19 11:16:45.0,932,32,INR,white,558-80-6346,3,2,2
BK0342703255,84232510,86494681,4.1.34,Android,-6.091461,-114.649789,22.8449505,70.137827,2020-08-03 19:10:52.0,2020-03-24 08:25:40.0,260,7,INR,blue,068-72-1637,3,3,3
cat: Unable to write to output stream.
```

Step – 3: Load Aggregated bookings data into Hadoop

Pyspark file “datewise_bookings_aggregates_spark.py” created to aggregate total number of bookings by pickup date and save aggregated in .csv format and save in HDFS location.

```
# Importing required libraries
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_232-cloudera/jre"
os.environ["SPARK_HOME"]="/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initiating Spark application/session
spark=SparkSession.builder.appName("datewise_bookings_aggregates_spark").master("local").getOrCreate()
spark

#Reading bookings data stored into HDFS directory
df=spark.read.csv("/user/root/bookings/part-m-00000")

#Verifying that all columns are imported
df.printSchema()

# Verify imported data with first 10 records
df.show(10)

#Verifying that entire file is imported and no of records are equal to no of records in HDFS file
df.count()

#Renaming the default columns names
new_col = ["booking_id","customer_id","driver_id","customer_app_version","customer_phone_os_version","pickup_lat","pickup_lon","drop_lat",
           "drop_lon","pickup_timestamp","drop_timestamp","trip_fare","tip_amount","currency_code","cab_color","cab_registration_no","customer_rating_by_driver",
           "rating_by_customer","passenger_count"]

#Assigning column names
new_df = df.toDF(*new_col)

#Converting pickup_timestamp to date by extracting date from pickup_timestamp for aggregation
new_df=new_df.select("booking_id","customer_id","driver_id","customer_app_version","customer_phone_os_version","pickup_lat","pickup_lon","drop_lat",
                    "drop_lon",to_date(col('pickup_timestamp')).alias('pickup_date').cast("date"),"drop_timestamp","trip_fare","tip_amount","currency_code","cab_color",
                    "cab_registration_no","customer_rating_by_driver",
                    "rating_by_customer","passenger_count")

#Verifying column names
new_df.show(5)

#Aggregation on pickup_date
agg_df=new_df.groupBy("pickup_date").count().orderBy("pickup_date")

#Verifying Aggregated dataframe
agg_df.show()

# Saving aggregated data into .csv format back to HDFS location
agg_df.coalesce(1).write.format('csv').mode('overwrite').save('/user/root/datewise_bookings_agg',header='true')
```

Executed this file to save aggregated file as .csv format into HDFS location

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 s3://aws-logs-147098151990-us-east-1/elasticmapreduce/j-3EXAGRFFL6LY6/cabride/datewise_bookings_aggregates_spark.py
```

```
hadoop@ip-172-31-76-175 ~ (17.505s)
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 datewise_bookings_aggregates_spark.py

Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-sql-kafka-0-10_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-e9bbef5a-9b87-4003-8dc5-0760bd796a2f;1.0
  confs: [default]
    found org.apache.spark#spark-sql-kafka-0-10_2.11:2.4.5 in central
    found org.apache.kafka#kafka-clients;2.0.0 in central
    found org.lz4#lz4-java;1.4.0 in central
    found org.xerial.snappy#snappy-java;1.1.7.3 in central
    found org.slf4j#slf4j-api;1.7.16 in central
    found org.spark-project.spark#unused;1.0.0 in central
  :: resolution report :: resolve 394ms :: artifacts dl 13ms
  :: modules in use:
    org.apache.kafka#kafka-clients;2.0.0 from central in [default]
    org.apache.spark#spark-sql-kafka-0-10_2.11:2.4.5 from central in [default]
    org.lz4#lz4-java;1.4.0 from central in [default]
    org.slf4j#slf4j-api;1.7.16 from central in [default]
    org.spark-project.spark#unused;1.0.0 from central in [default]
    org.xerial.snappy#snappy-java;1.1.7.3 from central in [default]
  -----
  |   conf   | | modules | | artifacts |
  |-----| |-----| |-----|
  | default | | 6       | | 0         |
  |-----| |-----| |-----|
```

Check CSV file created:

```
hadoop fs -ls /user/root/datewise_bookings_agg/
```

```
[hadoop@ip-172-31-29-137 ~]$ hadoop fs -ls /user/root/datewise_bookings_agg/
Found 2 items
-rw-r--r-- 1 hadoop hadoop 0 2024-02-05 05:32 /user/root/datewise_bookings_agg/ SUCCESS
-rw-r--r-- 1 hadoop hadoop 3776 2024-02-05 08:03 /user/root/datewise_bookings_agg/part-00000-e993f8e3-a13f-4c91-8e55-4f1520f4eaf8-c000.csv
```

Check aggregation data sample for each day:

```
hadoop fs -cat /user/root/datewise_bookings_agg/part-00000-e993f8e3-a13f-4c91-8e55-4f1520f4eaf8-c000.csv | head -n 5
```

```
[hadoop@ip-172-31-29-137 ~]$ hadoop fs -cat /user/root/datewise_bookings_agg/part-00000-e993f8e3-a13f-4c91-8e55-4f1520f4eaf8-c000.csv | head -n 5
pickup_date,count
2020-01-01,1
2020-01-02,3
2020-01-03,2
2020-01-04,2
```

Step – 4: Map HDFS files to Hive managed tables

Creating database and tables and Loading the data into these Hive tables from HDFS files

Once these tables are created in Hive DW, next step is to load data from files saved in HDFS directory from MySQL Bookings and Kafka Streaming systems.

Created database and bookings, clickstream and datewise_total_bookings (aggregated) in it:

1. Connect to hive instance

hive

```
hadoop@ip-172-31-74-21 ~/mysql-connector-java-8.0.25
hive
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive>
```

2. Create database and aggregated tables in it

create database cabrides;
use cabrides;

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> create database cabrides;
OK
Time taken: 0.957 seconds
hive> use cabrides;
OK
Time taken: 0.033 seconds
hive>
```

3. Create clickstream_data table with skip header , add terminate and delimiter since source from CSV

```
CREATE TABLE IF NOT EXISTS clickstream_data (
customer_id INT,
app_version STRING,
os_version STRING,
lat DOUBLE,
lon DOUBLE,
page_id STRING,
button_id STRING,
is_button_click BOOLEAN,
is_page_view BOOLEAN,
is_scroll_up BOOLEAN,
is_scroll_down BOOLEAN,
time_stamp TIMESTAMP)
COMMENT 'This table will store click streaming data red from kafka'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
hive> CREATE TABLE IF NOT EXISTS clickstream_data (
  > customer_id INT,
  > app_version STRING,
  > os_version STRING,
  > lat DOUBLE,
  > lon DOUBLE,
  > page_id STRING,
  > button_id STRING,
  > is_button_click BOOLEAN,
  > is_page_view BOOLEAN,
  > is_scroll_up BOOLEAN,
  > is_scroll_down BOOLEAN,
  > time_stamp TIMESTAMP)
  > COMMENT 'This table will store click streaming data red from kafka'
  > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  > LINES TERMINATED BY '\n'
  > STORED AS TEXTFILE
  > TBLPROPERTIES ("skip.header.line.count"="1");

OK
Time taken: 0.078 seconds
hive>
```

4. Create bookings_detail table

```
CREATE TABLE IF NOT EXISTS bookings_detail (
  booking_id STRING,
  customer_id INT,
  driver_id INT,
  customer_app_version STRING,
  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 DECIMAL(10, 2),
  tip_amount DECIMAL(10, 2),
  currency_code STRING,
  cab_color STRING,
  cab_registration_no STRING,
  customer_rating_by_driver INT,
  rating_by_customer INT,
  passenger_count INT
)
COMMENT 'This table will store Bookings'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
hive> CREATE TABLE IF NOT EXISTS bookings_detail (
  > booking_id STRING,
  > customer_id INT,
  > driver_id INT,
  > customer_app_version STRING,
  > 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 DECIMAL(10, 2),
  > tip_amount DECIMAL(10, 2),
  > currency_code STRING,
  > cab_color STRING,
  > cab_registration_no STRING,
  > customer_rating_by_driver INT,
  > rating_by_customer INT,
  > passenger_count INT)
  > COMMENT 'This table will store Bookings data red from MySQL RDS'
  > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  > LINES TERMINATED BY '\n'
  > STORED AS TEXTFILE;

OK
Time taken: 0.152 seconds
hive> 
```

5. create datewise_total_bookings table with skip header , add terminate and delimiter since source from CSV

```
CREATE TABLE IF NOT EXISTS datewise_total_bookings (

pickup_date DATE,

total_bookings INT)

COMMENT 'This table will store aggregated count of booking by pickup
Date'

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE
```

```
TBLPROPERTIES ("skip.header.line.count"="1");
hive> CREATE TABLE IF NOT EXISTS datewise_total_bookings (
  > pickup_date DATE,
  > total_bookings INT)
  > COMMENT 'This table will store aggregated count of booking by pickup date'
  > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  > LINES TERMINATED BY '\n'
  > STORED AS TEXTFILE
  > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.088 seconds
hive> █
```

Load data in hive tables

1. Load click stream data into clickstream_data table;

```
LOAD DATA INPATH '/user/root/clickstream_flattened/part-00000-88d3ea85-3f3e-437b-913c-f884392ec174-c000.csv' OVERWRITE INTO TABLE
clickstream_data
```

```
hive> LOAD DATA INPATH '/user/root/clickstream_flattened/part-00000-88d3ea85-3f3e-437b-913c-f884392ec174-c000.csv' OVERWRITE INTO TABLE clickstream_data;
Loading data to table cabrides.clickstream_data
OK
Time taken: 1.0 seconds
```

2. verify records count in clickstream_data table

```
select count(*) from clickstream_data;
```

```
hive> select count(*) from cabrides.clickstream_data;
Query ID = hadoop_20240205033546_cf19e35b-a7a1-4350-8516-95df3a75b4f2
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_1707097198836_0004)

-----
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: 5.95 s
-----
OK
3000
Time taken: 16.768 seconds, Fetched: 1 row(s)
```

Around 3000 records are present, 3 records removed because of customer id is empty in spark_local_flatten.py code.

3. Load booking file into bookings_detail table

```
LOAD DATA INPATH '/user/root/bookings/part-m-00000' OVERWRITE INTO TABLE
bookings_detail;
```

```
Time taken: 17.172 seconds, Fetched: 1 row(s)
hive> LOAD DATA INPATH '/user/root/bookings/part-m-00000' OVERWRITE INTO TABLE bookings_detail;
Loading data to table cabrides.bookings_detail
OK
Time taken: 0.478 seconds
```

4. verify records in bookings_detail table

```
select count(*) from bookings_detail;
```

```
hive> select count(*) from bookings_detail;
Query ID = hadoop_20240205055131_f92977f9-497a-4320-a69c-69b208a9e982
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1707107456620_0004)

-----
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: 5.34 s
-----
OK
1000
Time taken: 5.968 seconds, Fetched: 1 row(s)
```

Around 1000 records are present

5. Loading data into datewise_totoal_bookings and verifying records count

```
LOAD DATA INPATH '/user/root/datewise_bookings_agg/part-00000-d43d1e9b-987b-42d4-b140-1b9f318026dd-c000.csv' OVERWRITE INTO TABLE datewise_total_bookings;
```

```
hive> LOAD DATA INPATH '/user/root/datewise_bookings_agg/part-00000-b4d31edb-98b7-42d4-b140-1b9f318026dd-c000.csv' OVERWRITE INTO TABLE datewise_total_bookings;
Loading data to table cabrides.datewise_total_bookings
OK
Time taken: 0.437 seconds
hive>
```

```
select count(*) from datewise_total_bookings;
```

```
hive> select count(*) from datewise_total_bookings;
Query ID = hadoop_20240205055236_4a3ae733-f8b7-4b5a-abfb-1fb816bbe117
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1707107456620_0004)

-----
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: 5.74 s
-----
OK
289
Time taken: 6.355 seconds, Fetched: 1 row(s)
hive>
```


Around 289 records are present