Logic For First Submission

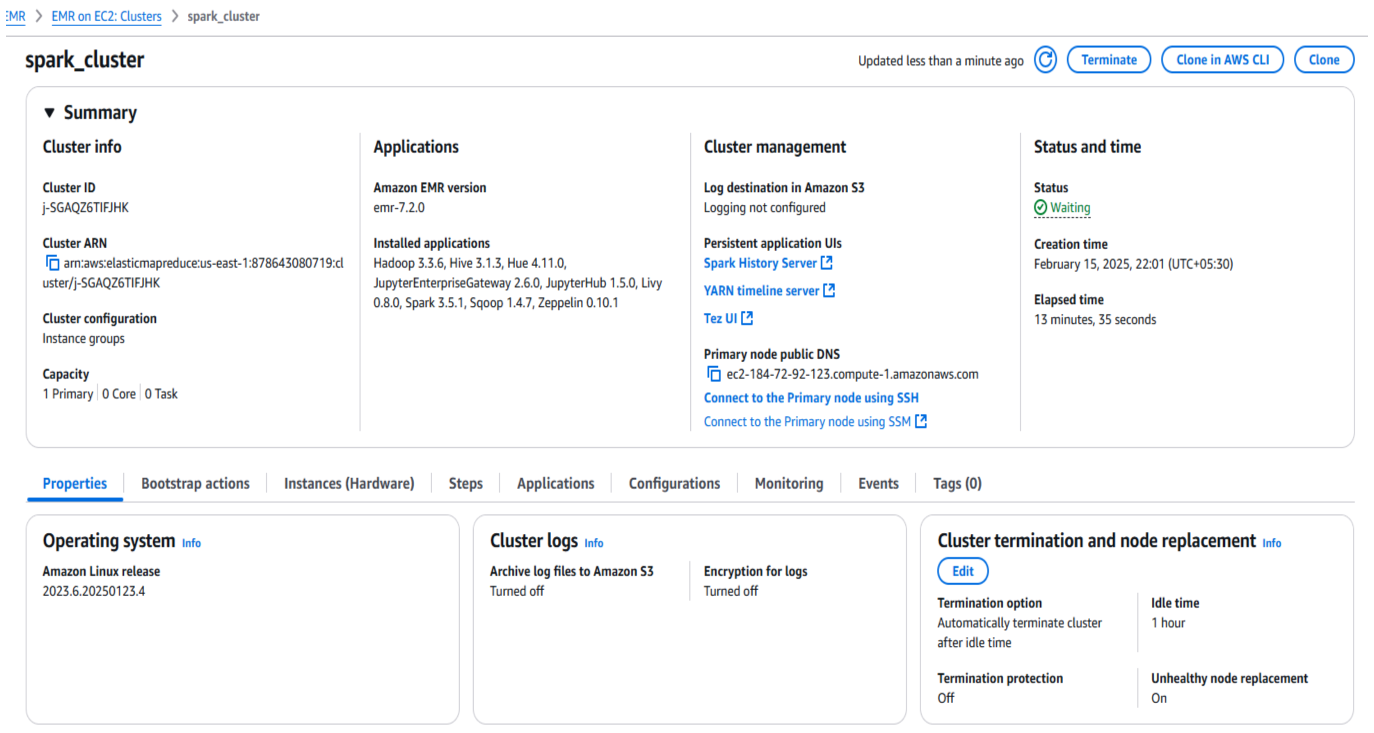
Problem Statement:

YourOwnCabs (YOC) is a leading online, on-demand cab booking service. Initially handling approximately 100 rides per day, the company has experienced exponential growth due to its strong business model and exceptional service quality. As a result, YOC continues to break its own records, with a rapidly expanding customer base and an increasing number of daily ride bookings.
To support strategic decision-making, business stakeholders require real-time, data-driven insights. However, the surge in data volume has made it increasingly challenging for management to retrieve critical business metrics efficiently. The existing MySQL backend is struggling to handle diverse and complex queries at scale, necessitating a more robust and scalable data infrastructure to accommodate the company's growing analytical needs.
.

Proposed Solution:

As a big data engineer, your task is to design and implement a scalable solution to meet the following requirements:
1. Booking Data Analytics: A robust system must be developed to store and process ride-booking data efficiently, ensuring seamless access for analytical purposes without disrupting business operations. This includes generating insights on daily, weekly, and monthly booking trends, categorizing bookings based on mobile operating systems, calculating average booking amounts, and aggregating total tip amounts.
2. Clickstream Data Analytics: Clickstream data captures user interactions within the app, such as button clicks, link clicks, and screen loads. This data helps analyze user behavior and optimize engagement strategies. Given its significantly larger volume compared to booking data, a well-structured architecture is required to store, process, and analyze it effectively for actionable insights.

Created an EMR instance with services like Hadoop, Sqoop, Hive, Hue and Spark to run the services to complete Cab Ride Capstone Project Analysis. Below is the screenshot of the EMR instance.


EMR > EMR on EC2: Clusters > spark_cluster

spark_cluster

Updated less than a minute ago | Terminate | Clone in AWS CLI | Clone

▼ Summary

Cluster info

Cluster ID
j-SGAQZ6TIFJHK

Cluster ARN
arn:aws:elasticmapreduce:us-east-1:878643080719:cluster/j-SGAQZ6TIFJHK

Cluster configuration
Instance groups

Capacity
1 Primary | 0 Core | 0 Task

Applications

Amazon EMR version
emr-7.2.0

Installed applications
Hadoop 3.3.6, Hive 3.1.3, Hue 4.11.0, JupyterEnterpriseGateway 2.6.0, JupyterHub 1.5.0, Livy 0.8.0, Spark 3.5.1, Sqoop 1.4.7, Zeppelin 0.10.1

Cluster management

Log destination in Amazon S3
Logging not configured

Persistent application UIs
Spark History Server
YARN timeline server
Tez UI

Primary node public DNS
ec2-184-72-92-123.compute-1.amazonaws.com
Connect to the Primary node using SSH
Connect to the Primary node using SSM

Status and time

Status
Waiting

Creation time
February 15, 2025, 22:01 (UTC+05:30)

Elapsed time
13 minutes, 35 seconds

Properties | Bootstrap actions | Instances (Hardware) | Steps | Applications | Configurations | Monitoring | Events | Tags (0)

Operating system Info

Amazon Linux release
2023.6.20250123.4

Cluster logs Info

Archive log files to Amazon S3
Turned off

Encryption for logs
Turned off

Cluster termination and node replacement Info

Edit

Termination option
Automatically terminate cluster after idle time

Idle time
1 hour

Termination protection
Off

Unhealthy node replacement
On

Step 1: Load data from kafka to Hadoop:

Reading clickstream data from kafka and writing the data as .json file in HDFS local directory:

To facilitate real-time data analysis, event data from the cab booking application is streamed into the Kafka topic de-capstone5 on the bootstrap server 18.211.252.152, port 9092. This data must be ingested, processed, and stored in a Hive-managed table for further analytical insights. The following steps outline the process:

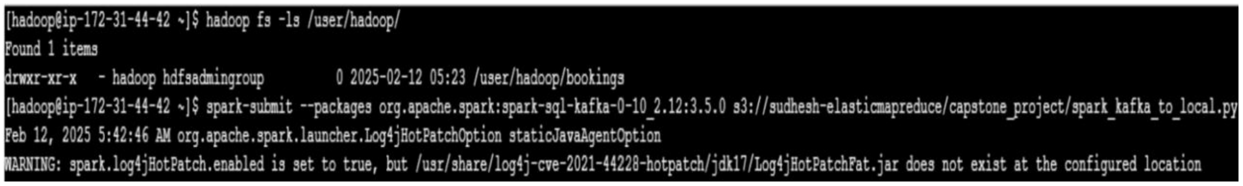
1. Reading Data from Kafka and Writing to HDFS

A PySpark script, **"spark_kafka_to_local.py"** has been developed to read streaming data from the Kafka server and write it to HDFS in JSON format. This script is securely stored in an S3 bucket for execution.

2. Executing the Script and Storing Streaming Data

The script is executed using Spark Submit, which processes and writes the data to HDFS. The following command is used to submit the job:

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 s3://sudhesh-elasticmapreduce/capstone_project/spark_kafka_to_local.py

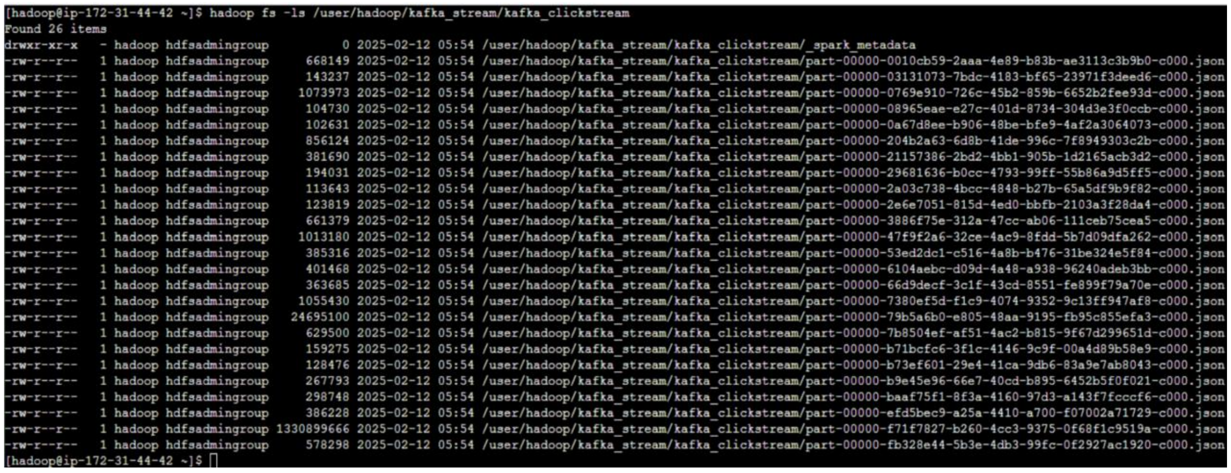


```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -ls /user/hadoop/
Found 1 items
drwxr-xr-x   - hadoop hdfsadmingroup          0 2025-02-12 05:23 /user/hadoop/bookings
[hadoop@ip-172-31-44-42 ~]$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 s3://sudhesh-elasticmapreduce/capstone_project/spark_kafka_to_local.py
Feb 12, 2025 5:42:46 AM org.apache.spark.launcher.Log4jHotPatchOption staticJavaAgentOption
WARNING: spark.log4jHotPatch.enabled is set to true, but /usr/share/log4j-cve-2021-44228-hotpatch/jdk17/Log4jHotPatchFat.jar does not exist at the configured location
```

3. Verifying Data Storage in HDFS

Once the data is written to HDFS, validation is performed to ensure successful ingestion. The presence of the data file in HDFS is checked using:

hadoop fs -ls /user/hadoop/kafka_stream/kafka_clickstream

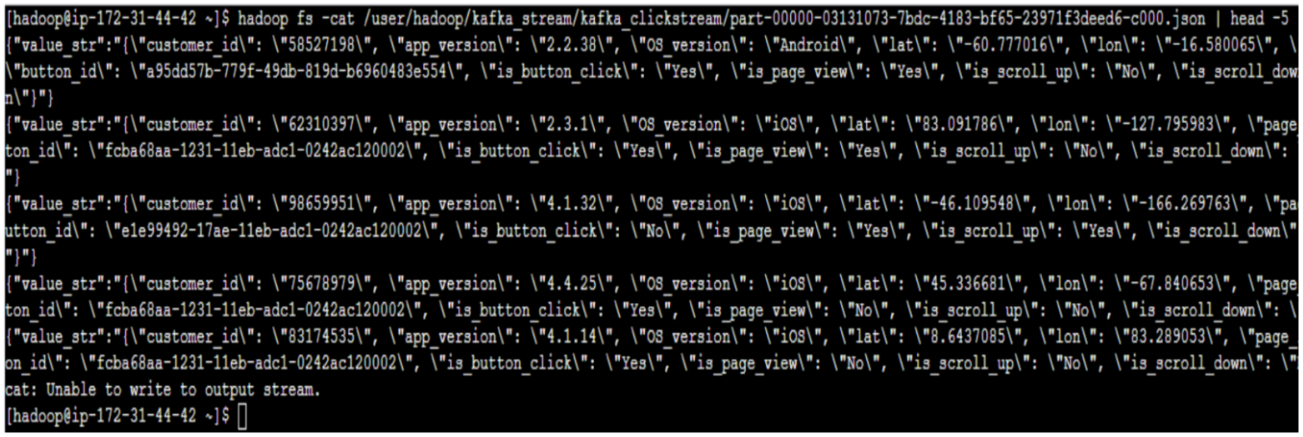


```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -ls /user/hadoop/kafka_stream/kafka_clickstream
Found 26 items
drwxr-xr-x   - hadoop hdfsadmingroup          0 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/_spark_metadata
-rw-r--r--   1 hadoop hdfsadmingroup     668149 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-0010cb59-2aaa-4e89-b83b-ae3113c3b9b0-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     143237 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-03131073-7bdc-4183-bf65-23971f3deed6-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup    1073973 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-0769e910-726c-45b2-859b-6652b2fee93d-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     104730 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-08965eae-e27c-401d-8734-304d3e3f0ccb-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     102631 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-0a67d8ee-b906-48be-bfe9-4af2a3064073-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     856124 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-204b2a63-6d8b-41de-996c-7f8949303c2b-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     381690 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-21157386-2bd2-4bb1-905b-1d2165acb3d2-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     194031 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-29681636-b0cc-4793-99ff-55b86a9d5ff5-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     113643 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-2a03c738-4bcc-4848-b27b-65a5df9b9f82-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     123819 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-2e6e7051-815d-4ed0-bbfb-2103a3f28da4-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     661379 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-3886f75e-312a-47cc-ab06-111ceb75cea5-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup    1013180 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-47f9f2a6-32ce-4ac9-8fdd-5b7d09dfa262-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     385316 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-53ed2dc1-c516-4a8b-b476-31be324e5f84-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     401468 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-6104aebc-d09d-4a48-a938-96240adeb3bb-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     363685 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-66d9decf-3c1f-43cd-8551-fe899f79a70e-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup    1055430 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-7380ef5d-f1c9-4074-9352-9c13ff947af8-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup   24695100 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-79b5a6b0-e805-48aa-9195-fb95c855efa3-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     629500 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-7b8504ef-af51-4ac2-b815-9f67d299651d-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     159275 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-b71bcfc6-3f1c-4146-9c9f-00a4d89b58e9-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     128476 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-b73ef601-29e4-41ca-9db6-83a9e7ab8043-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     267793 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-b9e45e96-66e7-40cd-b895-6452b5f0f021-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     298748 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-baaf75f1-8f3a-4160-97d3-a143f7fcccf6-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     386228 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-efd5bec9-a25a-4410-a700-f07002a71729-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup 1330899666 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-f71f7827-b260-4cc3-9375-0f68f1c9519a-c000.json
-rw-r--r--   1 hadoop hdfsadmingroup     578298 2025-02-12 05:54 /user/hadoop/kafka_stream/kafka_clickstream/part-00000-fb328e44-5b3e-4db3-99fc-0f2927ac1920-c000.json
[hadoop@ip-172-31-44-42 ~]$ 
```

4. Validating Records in JSON Format

To confirm data integrity, a sample of records from the JSON file is reviewed using the following command:

hadoop fs -cat /user/hadoop/kafka_stream/kafka_clickstream/part-00000-03131073-7bdc-4183-bf65-23971f3deed6-c000.json | head -5



```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -cat /user/hadoop/kafka_stream/kafka_clickstream/part-00000-03131073-7bdc-4183-bf65-23971f3deed6-c000.json | head -5
{"value_str":"{\"customer_id\": \"58527198\", \"app_version\": \"2.2.38\", \"OS_version\": \"Android\", \"lat\": \"-60.777016\", \"lon\": \"-16.580065\", \
\"button_id\": \"a95dd57b-779f-49db-819d-b6960483e554\", \"is_button_click\": \"Yes\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \"No\", \"is_scroll_dow
n\"}"}
{"value_str":"{\"customer_id\": \"62310397\", \"app_version\": \"2.3.1\", \"OS_version\": \"iOS\", \"lat\": \"83.091786\", \"lon\": \"-127.795983\", \"page
ton_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \"No\", \"is_scroll_down\":
"}
{"value_str":"{\"customer_id\": \"98659951\", \"app_version\": \"4.1.32\", \"OS_version\": \"iOS\", \"lat\": \"-46.109548\", \"lon\": \"-166.269763\", \"pa
utton_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \"Yes\", \"is_scroll_down\"
"}"}
{"value_str":"{\"customer_id\": \"75678979\", \"app_version\": \"4.4.25\", \"OS_version\": \"iOS\", \"lat\": \"45.336681\", \"lon\": \"-67.840653\", \"page
ton_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"No\", \"is_scroll_up\": \"No\", \"is_scroll_down\": \
{"value_str":"{\"customer_id\": \"83174535\", \"app_version\": \"4.1.14\", \"OS_version\": \"iOS\", \"lat\": \"8.6437085\", \"lon\": \"83.289053\", \"page_
on_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"No\", \"is_scroll_up\": \"No\", \"is_scroll_down\": \"
cat: Unable to write to output stream.
[hadoop@ip-172-31-44-42 ~]$ 
```

Processing and Transforming Kafka Stream Data for Hive Analysis

To enable structured data analysis, raw JSON event data from Kafka is processed and transformed into a structured CSV format before being stored in HDFS and later loaded into a Hive-managed table. The following steps outline the end-to-end data processing workflow:

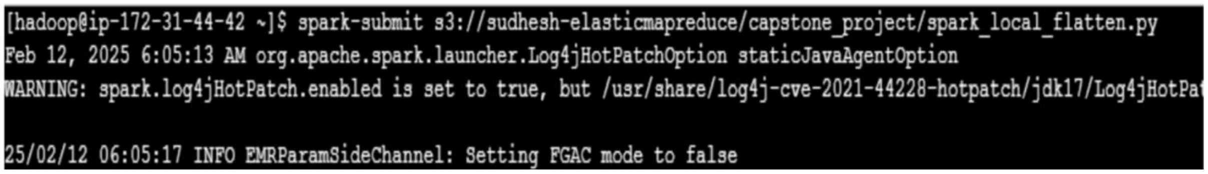
1. Reading and Transforming JSON Files

A PySpark script, **"spark_local_flatten.py"** has been developed to read raw JSON files from HDFS, filter out unnecessary data, and transform them into a structured CSV format. This script is securely stored in an S3 bucket for execution.

2. Executing the Transformation Script

The script is executed using Spark Submit to process and convert the JSON data into a structured CSV file, which is then stored in HDFS. The following command is used:

spark-submit s3://sudhesh-elasticmapreduce/capstone_project/spark_local_flatten.py

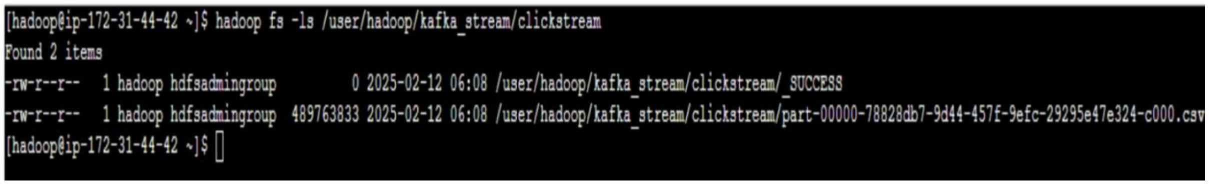


```
[hadoop@ip-172-31-44-42 ~]$ spark-submit s3://sudhesh-elasticmapreduce/capstone_project/spark_local_flatten.py
Feb 12, 2025 6:05:13 AM org.apache.spark.launcher.Log4jHotPatchOption staticJavaAgentOption
WARNING: spark.log4jHotPatch.enabled is set to true, but /usr/share/log4j-cve-2021-44228-hotpatch/jdk17/Log4jHotPat

25/02/12 06:05:17 INFO EMRParamSideChannel: Setting FGAC mode to false
```

3. Validating the Transformed CSV File

Once the transformation is complete, the presence and structure of the CSV file are verified in the designated HDFS directory using:

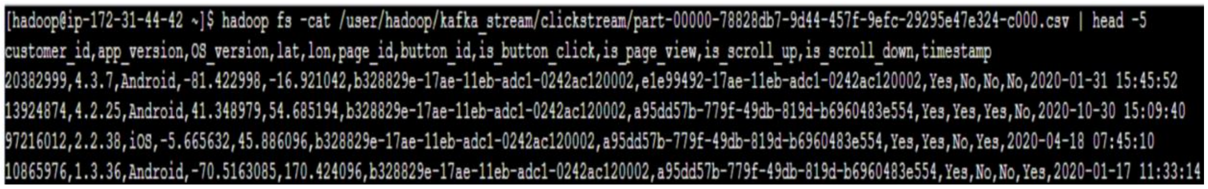
hadoop fs -ls /user/hadoop/kafka_stream/clickstream



```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -ls /user/hadoop/kafka_stream/clickstream
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup         0 2025-02-12 06:08 /user/hadoop/kafka_stream/clickstream/_SUCCESS
-rw-r--r--   1 hadoop hdfsadmingroup 489763833 2025-02-12 06:08 /user/hadoop/kafka_stream/clickstream/part-00000-78828db7-9d44-457f-9efc-29295e47e324-c000.csv
[hadoop@ip-172-31-44-42 ~]$ 
```

Additionally, sample records from the transformed file can be inspected to ensure proper formatting and data integrity.

hadoop fs -cat /user/hadoop/kafka_stream/clickstream/part-00000-78828db7-9d44-457f-9efc-29295e47e324-c000.csv | head -5



```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -cat /user/hadoop/kafka_stream/clickstream/part-00000-78828db7-9d44-457f-9efc-29295e47e324-c000.csv | head -5
customer_id,app_version,OS_version,lat,lon,page_id,button_id,is_button_click,is_page_view,is_scroll_up,is_scroll_down,timestamp
20382999,4.3.7,Android,-81.422998,-16.921042,b328829e-17ae-11eb-adc1-0242ac120002,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,No,No,No,2020-01-31 15:45:52
13924874,4.2.25,Android,41.348979,54.685194,b328829e-17ae-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,Yes,No,2020-10-30 15:09:40
97216012,2.2.38,iOS,-5.665632,45.886096,b328829e-17ae-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,No,Yes,2020-04-18 07:45:10
10865976,1.3.36,Android,-70.5163085,170.424096,b328829e-17ae-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,Yes,No,No,Yes,2020-01-17 11:33:14
```

Final Data Preparation for Hive Analysis
The structured CSV file stored in HDFS serves as the final dataset, ready to be loaded into a Hive-managed table for further analytical processing. This ensures clean, structured, and analysis-ready data, facilitating efficient querying and reporting.

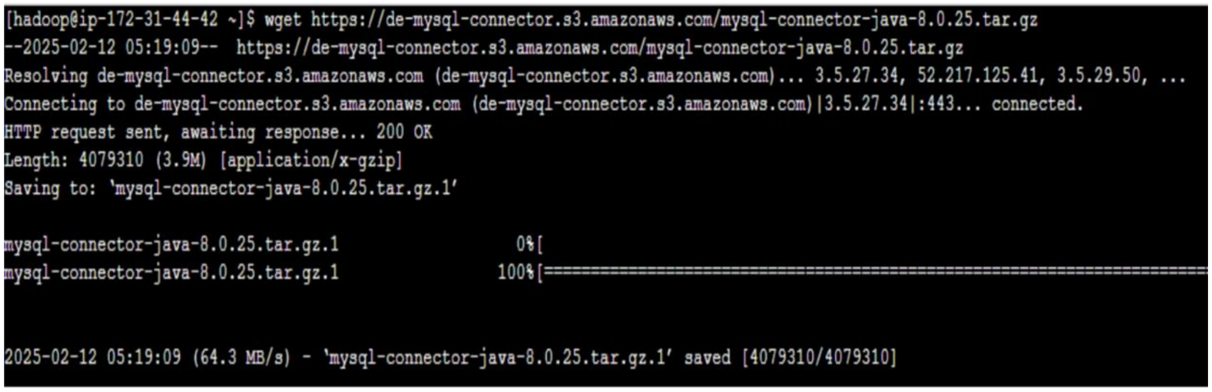
Step2: Load Data from AWS RDS to Hadoop

Data Ingestion with Sqoop

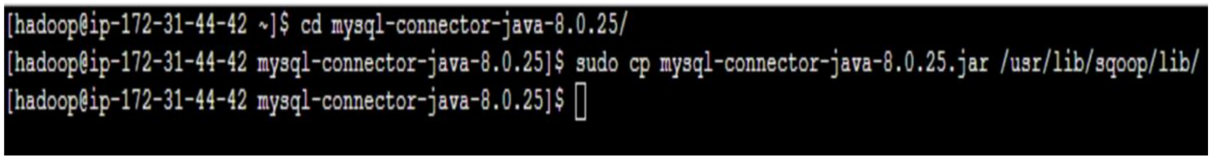
To facilitate seamless data transfer between the MySQL database and the target storage location, a MySQL connector must be installed to establish a bridge for retrieving the required dataset.

1. Installing the MySQL Connector
The following commands were executed to download, extract, and configure the MySQL connector for integration with Sqoop:

```
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/
```



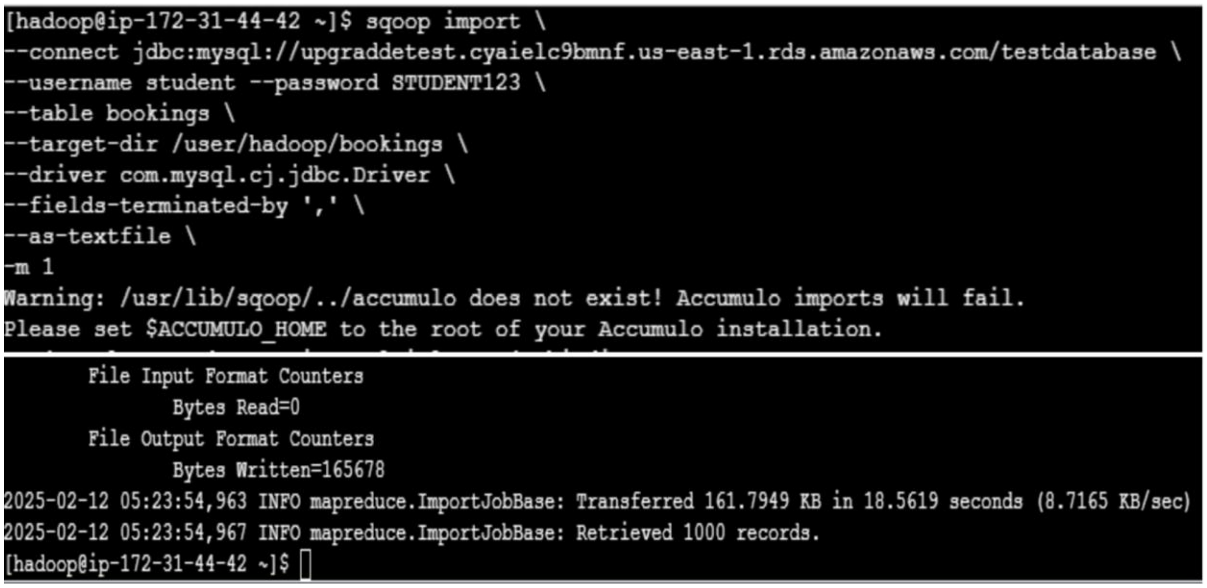
```
[hadoop@ip-172-31-44-42 ~]$ wget https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
--2025-02-12 05:19:09--  https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
Resolving de-mysql-connector.s3.amazonaws.com (de-mysql-connector.s3.amazonaws.com)... 3.5.27.34, 52.217.125.41, 3.5.29.50, ...
Connecting to de-mysql-connector.s3.amazonaws.com (de-mysql-connector.s3.amazonaws.com)|3.5.27.34|: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.1'

mysql-connector-java-8.0.25.tar.gz.1                0%[
mysql-connector-java-8.0.25.tar.gz.1              100%[==================================================================


2025-02-12 05:19:09 (64.3 MB/s) - 'mysql-connector-java-8.0.25.tar.gz.1' saved [4079310/4079310]
```



```
[hadoop@ip-172-31-44-42 ~]$ cd mysql-connector-java-8.0.25/
[hadoop@ip-172-31-44-42 mysql-connector-java-8.0.25]$ sudo cp mysql-connector-java-8.0.25.jar /usr/lib/sqoop/lib/
[hadoop@ip-172-31-44-42 mysql-connector-java-8.0.25]$ 
```

2. Executing Sqoop Import to Ingest Bookings Data
The Sqoop import command was used to extract relevant bookings data from an AWS RDS MySQL database and store it in HDFS for further processing.

```
sqoop import \
--connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-
1.rds.amazonaws.com/testdatabase \
--username student --password STUDENT123 \
--table bookings \
--target-dir /user/hadoop/bookings \
--driver com.mysql.cj.jdbc.Driver \
--fields-terminated-by ',' \
--as-textfile \
-m 1
```



```
[hadoop@ip-172-31-44-42 ~]$ sqoop import \
--connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
--username student --password STUDENT123 \
--table bookings \
--target-dir /user/hadoop/bookings \
--driver com.mysql.cj.jdbc.Driver \
--fields-terminated-by ',' \
--as-textfile \
-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.
```

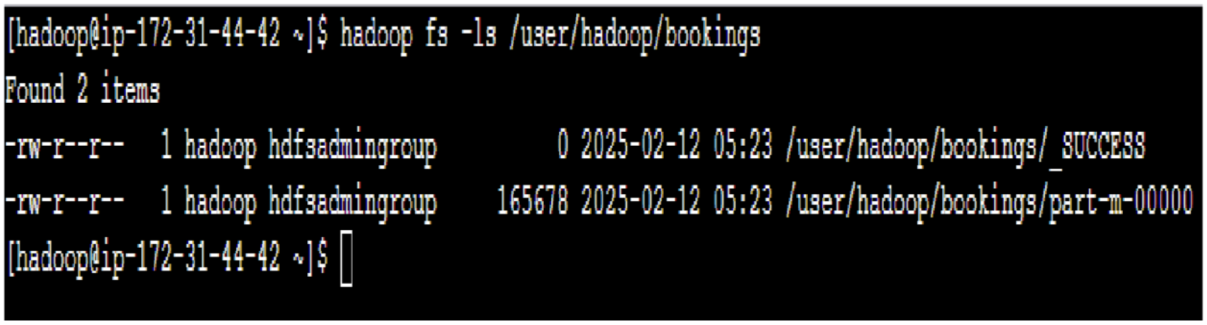
```
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=165678
2025-02-12 05:23:54,963 INFO mapreduce.ImportJobBase: Transferred 161.7949 KB in 18.5619 seconds (8.7165 KB/sec)
2025-02-12 05:23:54,967 INFO mapreduce.ImportJobBase: Retrieved 1000 records.
[hadoop@ip-172-31-44-42 ~]$
```

1000 records retrieved and imported to HDFS

3. Validating Data Ingestion in HDFS

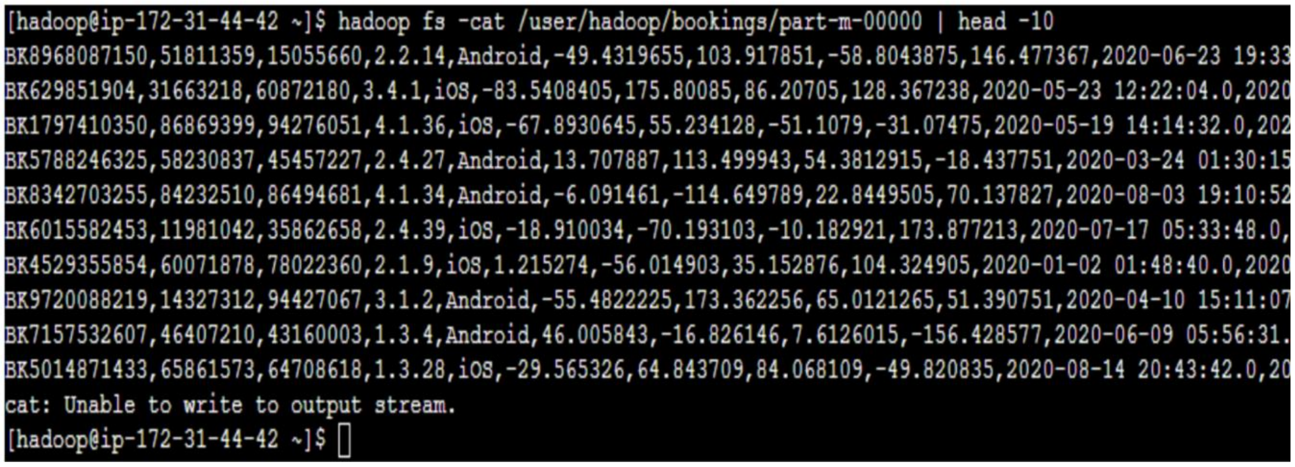
Once the data import was completed, verification was performed to ensure the bookings table was successfully ingested into the specified HDFS directory using:

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

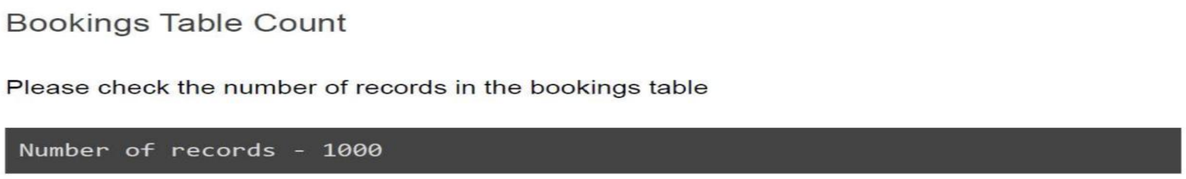


```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -ls /user/hadoop/bookings
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup          0 2025-02-12 05:23 /user/hadoop/bookings/_SUCCESS
-rw-r--r--   1 hadoop hdfsadmingroup     165678 2025-02-12 05:23 /user/hadoop/bookings/part-m-00000
[hadoop@ip-172-31-44-42 ~]$
```

Additionally, a preview of the imported data was checked by displaying the first 10 records:
hadoop fs -cat /user/hadoop/bookings/part-m-00000 | head -10



```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -cat /user/hadoop/bookings/part-m-00000 | head -10
BK8968087150,51811359,15055660,2.2.14,Android,-49.4319655,103.917851,-58.8043875,146.477367,2020-06-23 19:33
BK629851904,31663218,60872180,3.4.1,iOS,-83.5408405,175.80085,86.20705,128.367238,2020-05-23 12:22:04.0,2020
BK1797410350,86869399,94276051,4.1.36,iOS,-67.8930645,55.234128,-51.1079,-31.07475,2020-05-19 14:14:32.0,202
BK5788246325,58230837,45457227,2.4.27,Android,13.707887,113.499943,54.3812915,-18.437751,2020-03-24 01:30:15
BK8342703255,84232510,86494681,4.1.34,Android,-6.091461,-114.649789,22.8449505,70.137827,2020-08-03 19:10:52
BK6015582453,11981042,35862658,2.4.39,iOS,-18.910034,-70.193103,-10.182921,173.877213,2020-07-17 05:33:48.0,
BK4529355854,60071878,78022360,2.1.9,iOS,1.215274,-56.014903,35.152876,104.324905,2020-01-02 01:48:40.0,2020
BK9720088219,14327312,94427067,3.1.2,Android,-55.4822225,173.362256,65.0121265,51.390751,2020-04-10 15:11:07
BK7157532607,46407210,43160003,1.3.4,Android,46.005843,-16.826146,7.6126015,-156.428577,2020-06-09 05:56:31.
BK5014871433,65861573,64708618,1.3.28,iOS,-29.565326,64.843709,84.068109,-49.820835,2020-08-14 20:43:42.0,20
cat: Unable to write to output stream.
[hadoop@ip-172-31-44-42 ~]$ 
```



Bookings Table Count

Please check the number of records in the bookings table

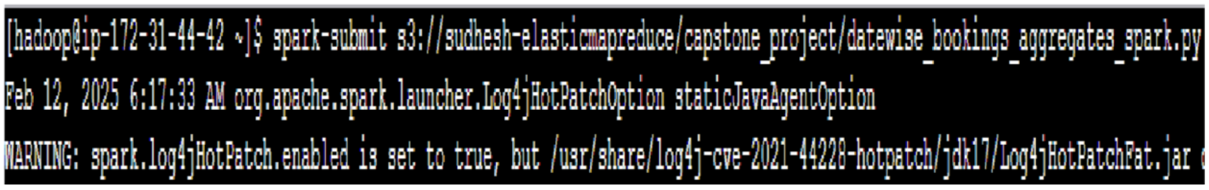
```
Number of records - 1000
```

Step 3: Load Aggregated bookings data into Hadoop

To analyze the number of bookings based on the pickup date, a PySpark script named **"datewise_bookings_aggregates_spark.py"** was developed. This script processes booking data, aggregates it by date, and stores the results in a structured format. The script is securely stored in an S3 bucket for execution.

1. Executing the Aggregation Script

The script is executed using the Spark Submit command, which processes the data and generates an aggregated output in CSV format. The execution command is:
spark-submit s3://sudhesh-elasticmapreduce/capstone_project/datewise_bookings_aggregates_spark.py

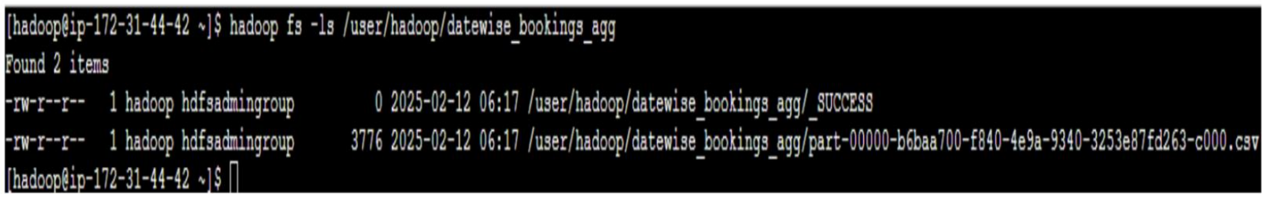


```
[hadoop@ip-172-31-44-42 ~]$ spark-submit s3://sudhesh-elasticmapreduce/capstone_project/datewise_bookings_aggregates_spark.py
Feb 12, 2025 6:17:33 AM org.apache.spark.launcher.Log4jHotPatchOption staticJavaAgentOption
WARNING: spark.log4jHotPatch.enabled is set to true, but /usr/share/log4j-cve-2021-44228-hotpatch/jdk17/Log4jHotPatchFat.jar
```

2. **Saving Aggregated Data to HDFS**

Once processed, the aggregated data is saved in HDFS as a CSV file using the following command:

agg_df.coalesce(1).write.format('csv').mode('overwrite').save('/user/hadoop/datewise_bookings_agg', header='true')

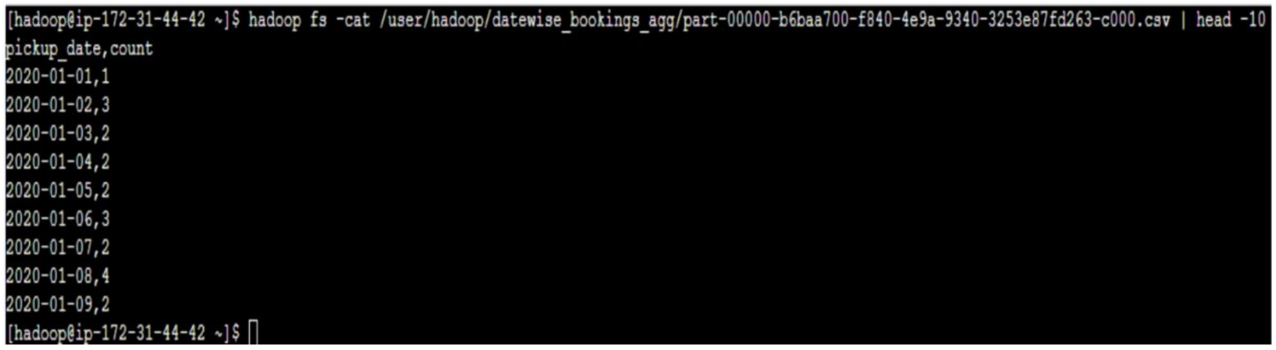


```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -ls /user/hadoop/datewise_bookings_agg
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup          0 2025-02-12 06:17 /user/hadoop/datewise_bookings_agg/_SUCCESS
-rw-r--r--   1 hadoop hdfsadmingroup       3776 2025-02-12 06:17 /user/hadoop/datewise_bookings_agg/part-00000-b6baa700-f840-4e9a-9340-3253e87fd263-c000.csv
[hadoop@ip-172-31-44-42 ~]$ 
```

3. **Validating Aggregated Data in HDFS**

After execution, the CSV file is verified in the specified HDFS directory to ensure successful data aggregation and storage.

hadoop fs -cat /user/hadoop/datewise_bookings_agg/part-00000-b6baa700-f840-4e9a-9340-3253e87fd263-c000.csv | head -10



```
[hadoop@ip-172-31-44-42 ~]$ hadoop fs -cat /user/hadoop/datewise_bookings_agg/part-00000-b6baa700-f840-4e9a-9340-3253e87fd263-c000.csv | head -10
pickup_date,count
2020-01-01,1
2020-01-02,3
2020-01-03,2
2020-01-04,2
2020-01-05,2
2020-01-06,3
2020-01-07,2
2020-01-08,4
2020-01-09,2
[hadoop@ip-172-31-44-42 ~]$ 
```

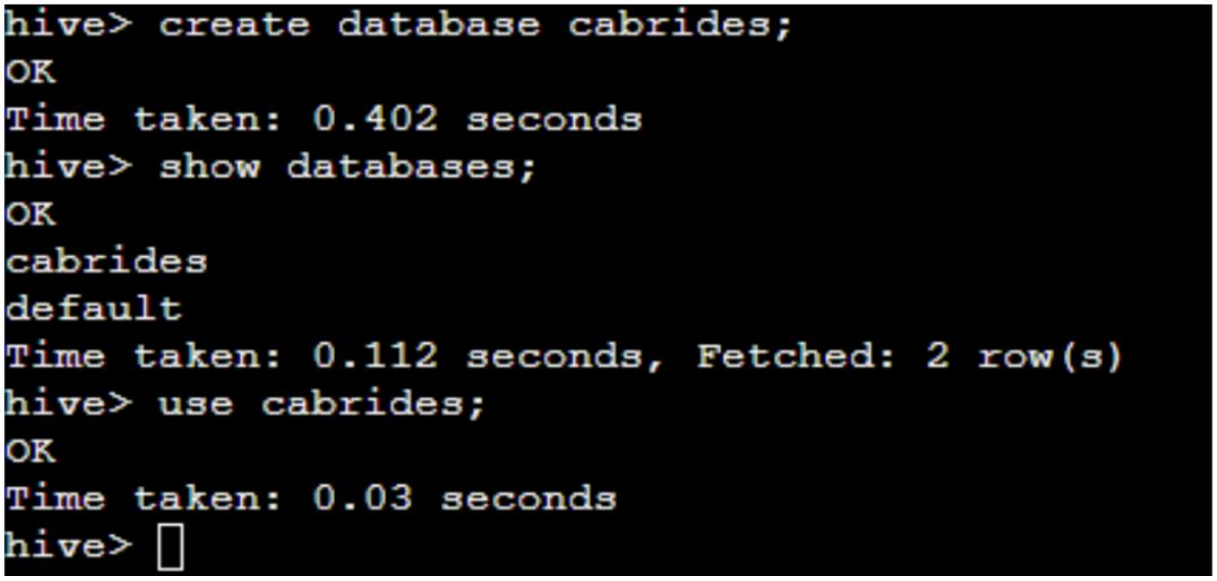
This process enables efficient reporting on date-wise booking trends, supporting data-driven decision-making for operational and business insights.

Step 4: Loading Data from AWS RDS to Hadoop and Creating Hive Managed Tables

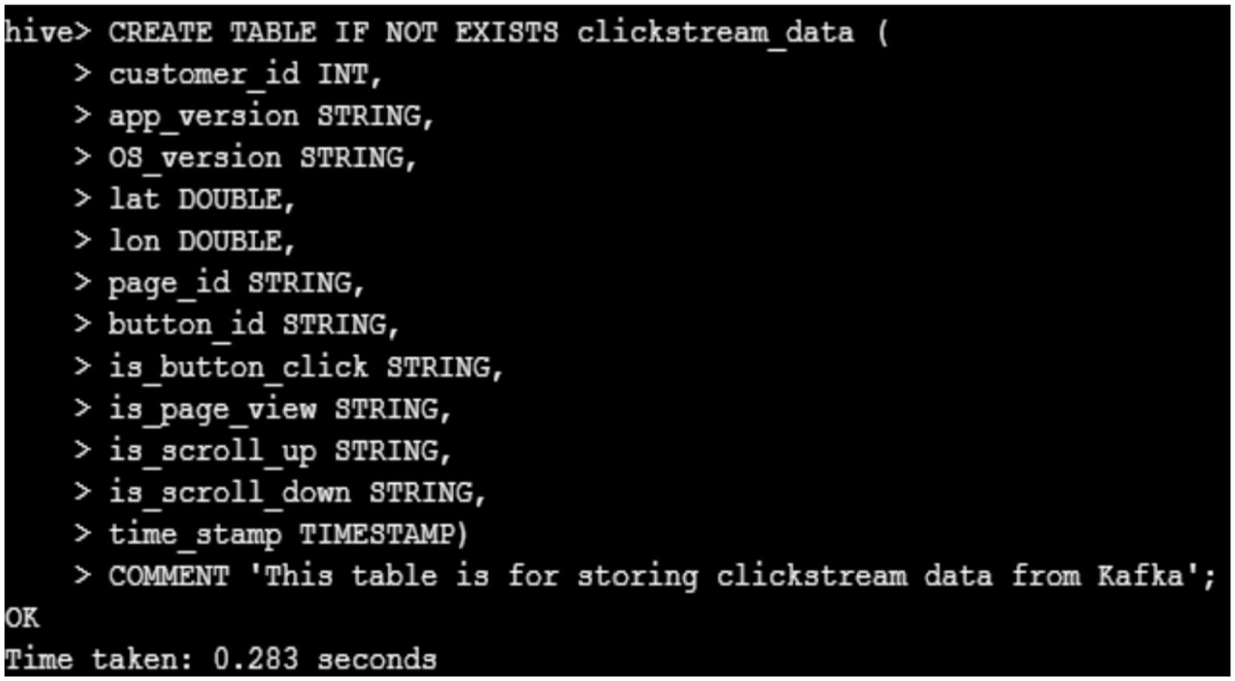
To facilitate structured data analysis, data from AWS RDS MySQL and Kafka Streaming systems is loaded into Hive-managed tables. Below are the key steps involved:

1. Creating Hive Database and Tables

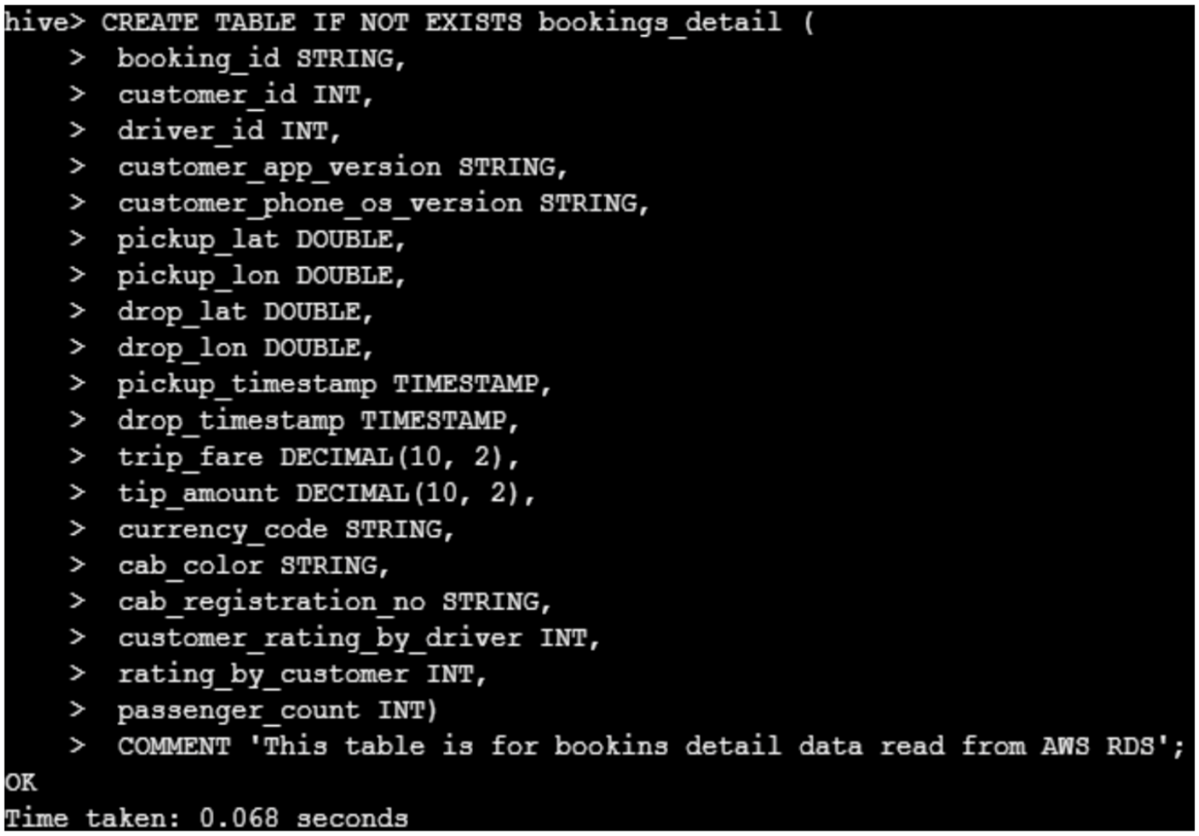
- A Hive database was created to store structured data. Three tables were defined:
    - bookings_detail – Stores ride booking details.
    - clickstream_data – Captures user interactions from the app.
    - datewise_total_bookings – Contains aggregated booking data by date.



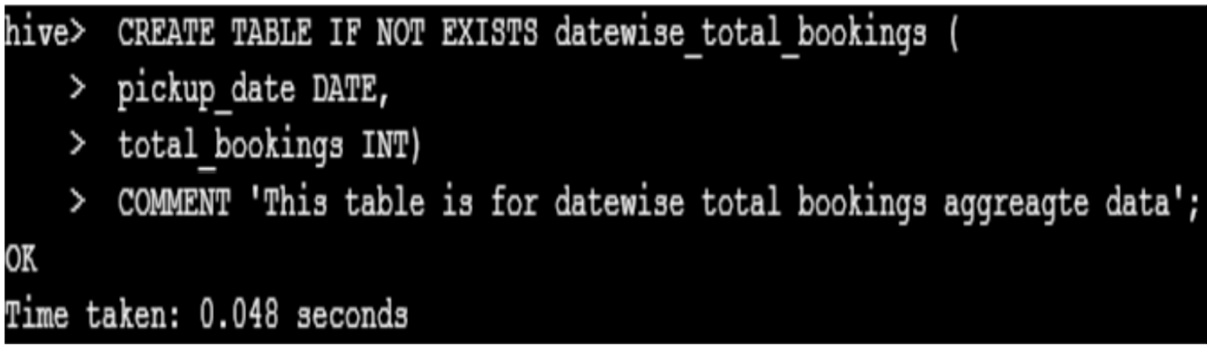
```
hive> create database cabrides;
OK
Time taken: 0.402 seconds
hive> show databases;
OK
cabrides
default
Time taken: 0.112 seconds, Fetched: 2 row(s)
hive> use cabrides;
OK
Time taken: 0.03 seconds
hive>
```



```
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 STRING,
    > is_page_view STRING,
    > is_scroll_up STRING,
    > is_scroll_down STRING,
    > time_stamp TIMESTAMP)
    > COMMENT 'This table is for storing clickstream data from Kafka';
OK
Time taken: 0.283 seconds
```



```
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 is for bookins detail data read from AWS RDS';
OK
Time taken: 0.068 seconds
```

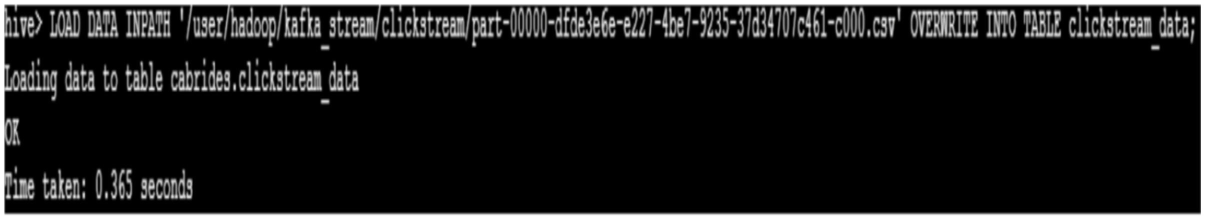


```
hive> CREATE TABLE IF NOT EXISTS datewise_total_bookings (
    > pickup_date DATE,
    > total_bookings INT)
    > COMMENT 'This table is for datewise total bookings aggreagte data';
OK
Time taken: 0.048 seconds
```

2. Loading Data into Hive Tables from HDFS

- Loading Clickstream Data into clickstream_data table.

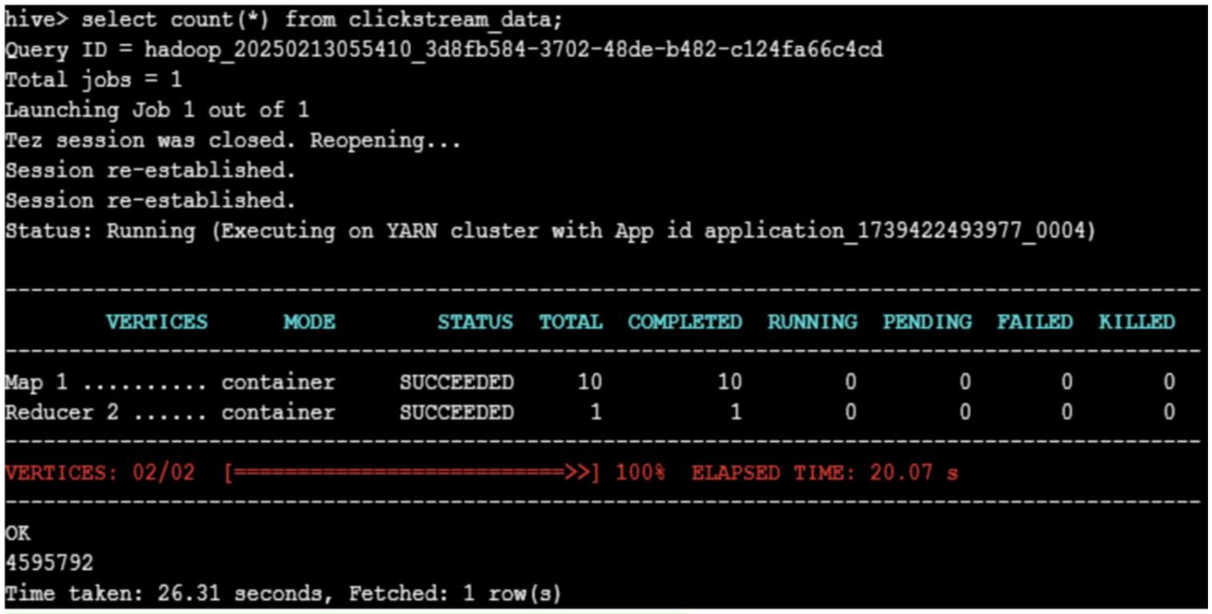
  LOAD DATA INPATH '/user/hadoop/kafka_stream/clickstream/part-00000-dfde3e6e-e227-4be7-9235-37d34707c461-c000.csv' OVERWRITE INTO TABLE clickstream_data;



```
hive> LOAD DATA INPATH '/user/hadoop/kafka_stream/clickstream/part-00000-dfde3e6e-e227-4be7-9235-37d34707c461-c000.csv' OVERWRITE INTO TABLE clickstream_data;
Loading data to table cabrides.clickstream_data
OK
Time taken: 0.365 seconds
```

- Verifying Record Count in clickstream_data table.

Select count(*) from clickstream_data;

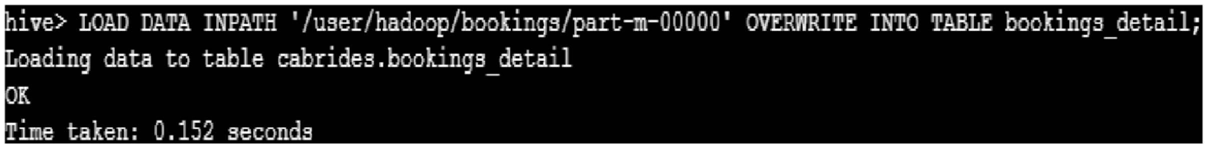


```
hive> select count(*) from clickstream_data;
Query ID = hadoop_20250213055410_3d8fb584-3702-48de-b482-c124fa66c4cd
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1739422493977_0004)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     10         10        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 20.07 s
----------------------------------------------------------------------------------------------
OK
4595792
Time taken: 26.31 seconds, Fetched: 1 row(s)
```

Total Number of records in clickstream_data table = 4595792

- Loading Bookings Data into bookings_detail table.

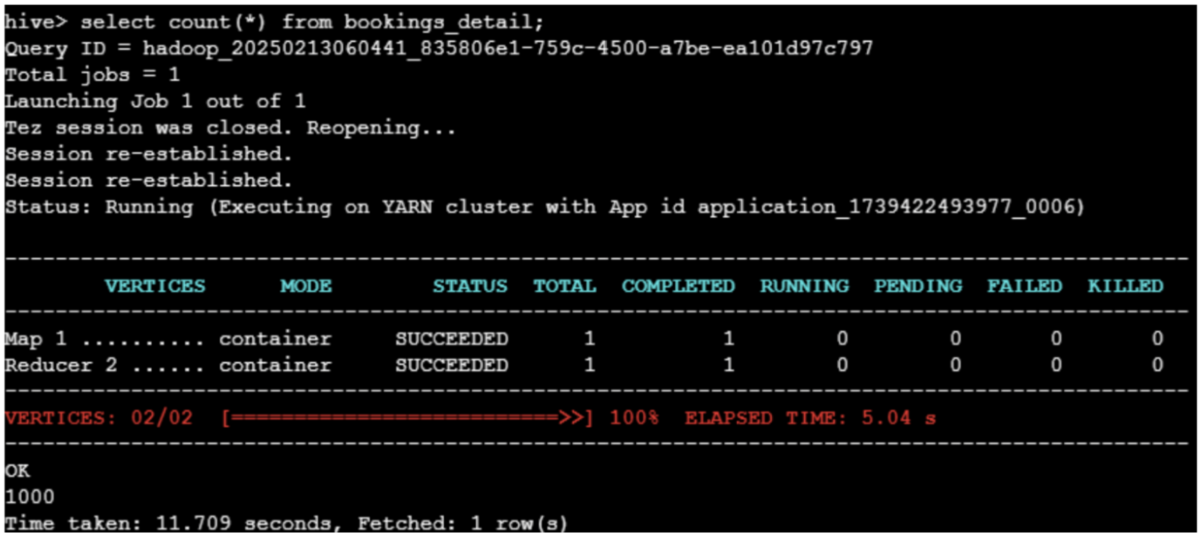
LOAD DATA INPATH '/user/hadoop/bookings/part-m-00000' OVERWRITE INTO TABLE bookings_detail;



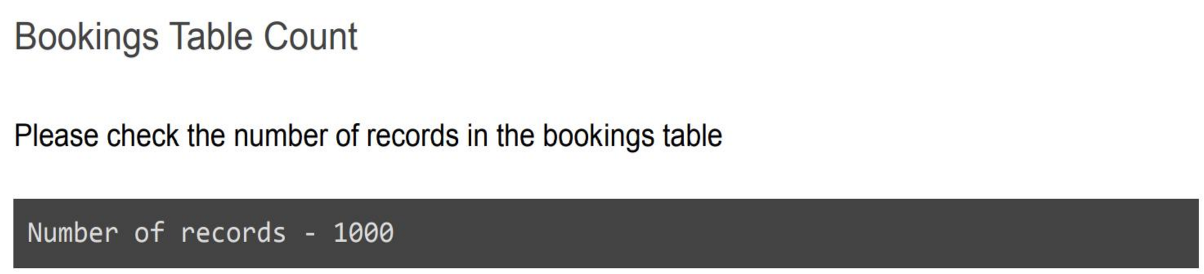
```
hive> LOAD DATA INPATH '/user/hadoop/bookings/part-m-00000' OVERWRITE INTO TABLE bookings_detail;
Loading data to table cabrides.bookings_detail
OK
Time taken: 0.152 seconds
```

- Verifying Record Count in bookings_detail table.

Select count(*) from bookings_detail;



```
hive> select count(*) from bookings_detail;
Query ID = hadoop_20250213060441_835806e1-759c-4500-a7be-ea101d97c797
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1739422493977_0006)

----------------------------------------------------------------------------------------------
        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: 5.04 s
----------------------------------------------------------------------------------------------
OK
1000
Time taken: 11.709 seconds, Fetched: 1 row(s)
```



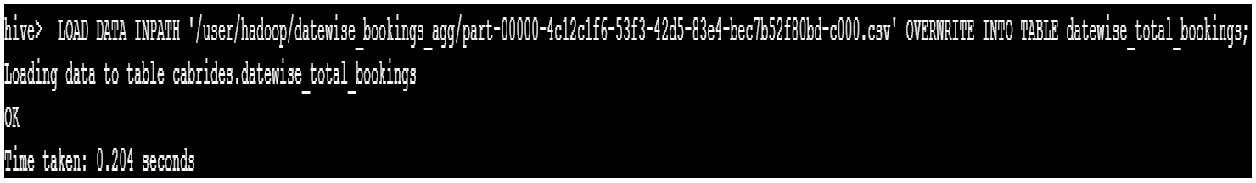
Bookings Table Count

Please check the number of records in the bookings table

```
Number of records - 1000
```

- Loading Aggregated Booking Data into datewise_total_bookings table.

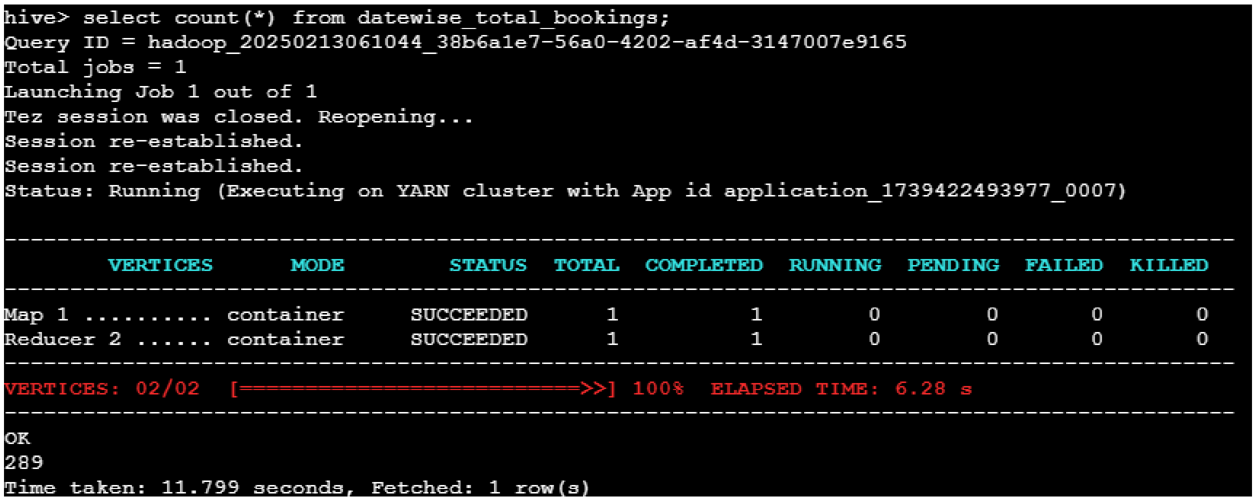
LOAD DATA INPATH '/user/hadoop/datewise_bookings_agg/part-00000-4c12c1f6-53f3-42d5-83e4-bec7b52f80bd-c000.csv' OVERWRITE INTO TABLE datewise_total_bookings



```
hive>  LOAD DATA INPATH '/user/hadoop/datewise_bookings_agg/part-00000-4c12c1f6-53f3-42d5-83e4-bec7b52f80bd-c000.csv' OVERWRITE INTO TABLE datewise_total_bookings;
Loading data to table cabrides.datewise_total_bookings
OK
Time taken: 0.204 seconds
```

- Verifying Record Count in datewise_total_bookings table.

SELECT count(*) from datewise_total_bookings;



```
hive> select count(*) from datewise_total_bookings;
Query ID = hadoop_20250213061044_38b6a1e7-56a0-4202-af4d-3147007e9165
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1739422493977_0007)

----------------------------------------------------------------------------------------------
        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: 6.28 s
----------------------------------------------------------------------------------------------
OK
289
Time taken: 11.799 seconds, Fetched: 1 row(s)
```

Total Number of records in datewise_total_bookings table: 289

Bookings Aggregates Table Count

Please check the number of records in the bookings aggregates table

```
Number of records - 289
```