LogicFirstSubmission

Click Stream Data from kafka

- EMR Cluster Creation on AWS

An Elastic MapReduce (EMR) cluster is created on Amazon Web Services (AWS), providing a managed Hadoop framework for processing large-scale data.From the cloud shell with the key pair we connect to the emr cluster

- Connecting to the EMR Cluster

Access to the EMR cluster is established from the AWS Cloud Shell using a specific key pair. This key pair is crucial for secure SSH access to the EMR cluster.

- Transferring the Spark-Kafka Script

The Spark code for processing Kafka data (spark_kafka_to_local) is securely copied from the local machine to the EMR cluster. This is achieved using the scp (secure copy) command, ensuring the script is available on the EMR cluster for execution.

- Execution as Hadoop Administrator

Once the script is on the EMR cluster, it is executed with Hadoop administrative privileges. This is important for permissions and access to various resources and services on the EMR cluster.

- Running the Spark Job

The script is run as a Spark job. During its execution, it fetches data from the specified Kafka topic. Writing Data to HDFS

- Writing Data to HDFS

The data retrieved from Kafka is written into the Hadoop Distributed File System (HDFS) on the EMR cluster. This allows for scalable and reliable storage of the data, which is essential for subsequent processing steps.

- Data Transformation for Further Processing

The Kafka data stored in HDFS serves as the input for further transformation processes defined in the Spark_local_flatten.py file.

Click stream data from kafka code explaination

- Initializing Spark Session:

The SparkSession is initialized with an application name ("clickstream_data_from_Kafka"). This is the entry point to interact with underlying Spark functionality.

spark.sparkContext.setLogLevel('ERROR') ensures that only error logs are shown, reducing log clutter.

```
# Initialize Spark Session
spark = SparkSession \
        .builder \
        .appName("clickstream_data_from_Kafka")\
        .getOrCreate()

spark.sparkContext.setLogLevel('ERROR')
```

- Kafka Configuration:

The Kafka server's host (18.211.252.152) and port (9092) are specified.

A Kafka topic ("de-capstone3") is defined for subscribing to the data stream.

```
# Kafka Config
host1 = "18.211.252.152"
port1 = 9092
topic1= "de-capstone3"
```

- Reading from Kafka:

The readStream method of Spark is used, indicating a streaming read operation.

Kafka-specific options are set: bootstrap servers, starting offsets, subscription to a topic, and handling data loss scenarios.

Click stream data from kafka code explaination

```
clickStream_data = spark.readStream \
                   .format('kafka') \
                   .option("kafka.bootstrap.servers",host1+":"+str(port1))\
                   .option("startingOffsets","earliest")\
                   .option("subscribe",topic1)\
                   .option("failOnDataLoss",False)\
                   .load()
```

Defining the Schema:
A JSON schema is defined using StructType and StructField. This schema corresponds to the expected format of the incoming JSON data from Kafka.

```
json_schema = StructType([StructField("customer_id",StringType()),
                          StructField("app_version",StringType()),
                         StructField("os_version",StringType()),
                         StructField("lat",StringType()),
                         StructField("page_id",StringType()),
                         StructField("button_id",StringType()),
                         StructField("is_button_click",StringType()),
                         StructField("is_page_view",StringType()),
                         StructField("is_scroll_up",StringType()),
                         StructField("is_scroll_down",StringType()),
                         StructField("timestamp",StringType())])
```

Data Processing:
The incoming data, which is in Kafka's native format, is transformed into a DataFrame.
from_json(col("value").cast("string"), json_schema) is used to parse the JSON data in the value column of the Kafka message using the defined schema.
The data is then selected into a new DataFrame clickStream.

```
clickStream =
clickStream_data.select(from_json(col("value").cast("string"),json_schema).alias("data")).select("data.*")
```

Click stream data from kafka code explanation

- Writing the Stream to Different Outputs:

Two separate streaming queries (query and query1) are set up.

query writes the stream to a CSV file in the specified path and checkpoint location, with a trigger interval of 1 minute.

query1 outputs the stream to the console for real-time viewing.

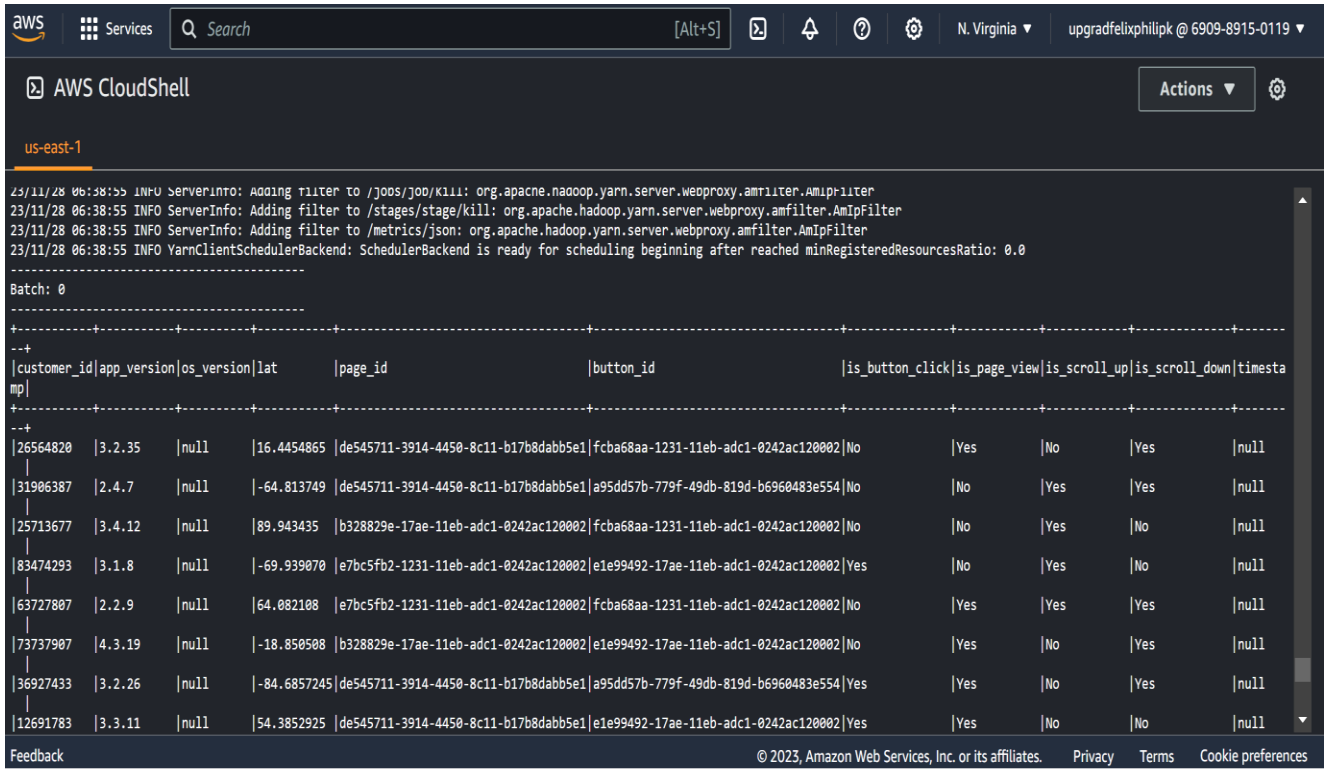
```
query = clickStream \
        .writeStream\
        .format("csv")\
        .outputMode("append")\
        .option("path","/user/ec2-user/Data/ClickStream")\
        .option("checkpointLocation","/user/ec2-user/Data/ClickStream_Check_Point")\
        .trigger(processingTime ='1 minute')\
        .start()
query1 = clickStream \
         .writeStream\
         .outputMode("append")\
         .format("console")\
         .option("truncate",False)\
         .start()
```

Awaiting Termination:

query.awaitTermination() and query1.awaitTermination() are called to keep the application running indefinitely. This is essential in streaming applications to continuously process incoming data.

```
query.awaitTermination()
query1.awaitTermination()
```

Output after Running spark submit for spark kafka to local python file



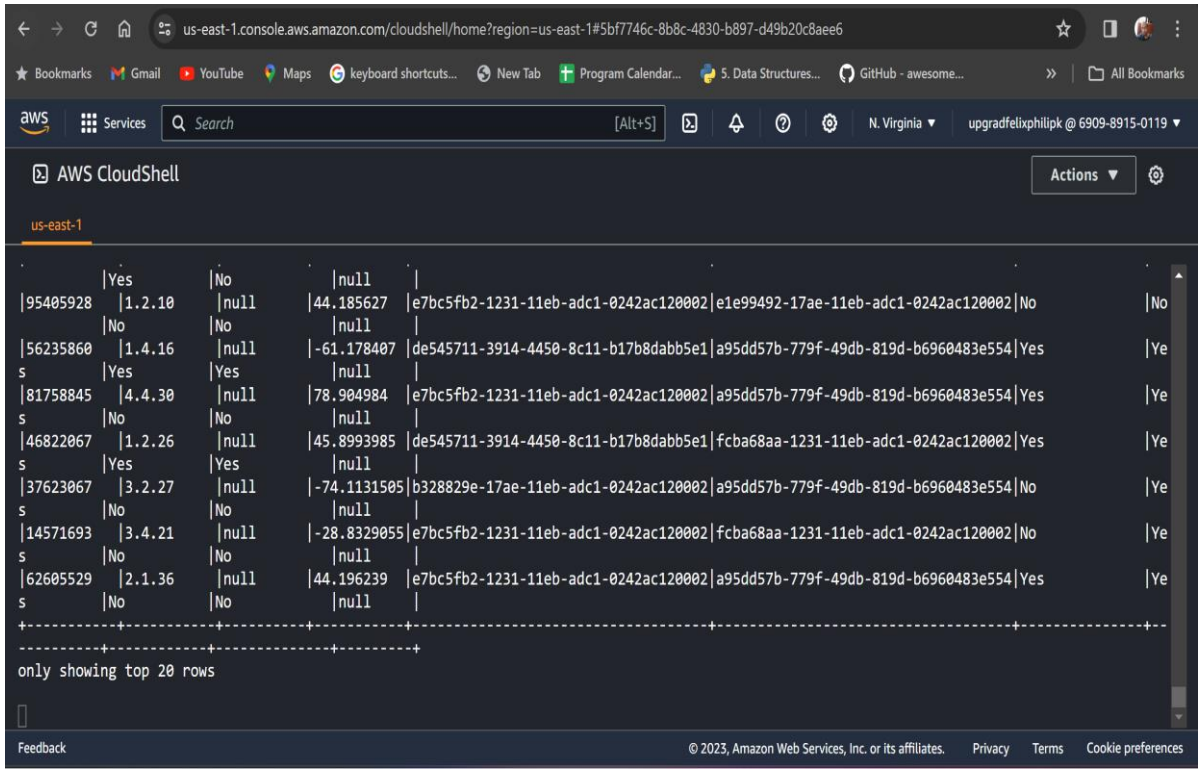
aws Services Search [Alt+S] N. Virginia upgradfelixphilipk @ 6909-8915-0119

AWS CloudShell Actions

us-east-1

```
23/11/28 06:38:55 INFO ServerInfo: Adding filter to /jobs/job/kill: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
23/11/28 06:38:55 INFO ServerInfo: Adding filter to /stages/stage/kill: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
23/11/28 06:38:55 INFO ServerInfo: Adding filter to /metrics/json: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
23/11/28 06:38:55 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
-------------------------------------------
Batch: 0
-------------------------------------------
+-----------+-----------+----------+-----------+--------------------------------+--------------------------------+--------------+------------+------------+-------------+------
--+
|customer_id|app_version|os_version|lat        |page_id                         |button_id                       |is_button_click|is_page_view|is_scroll_up|is_scroll_down|timesta
mp|
+-----------+-----------+----------+-----------+--------------------------------+--------------------------------+--------------+------------+------------+-------------+------
--+
|26564820   |3.2.35     |null      |16.4454865 |de545711-3914-4450-8c11-b17b8dabb5e1|fcba68aa-1231-11eb-adc1-0242ac120002|No            |Yes         |No          |Yes          |null
  |
|31906387   |2.4.7      |null      |-64.813749 |de545711-3914-4450-8c11-b17b8dabb5e1|a95dd57b-779f-49db-819d-b6960483e554|No            |No          |Yes         |Yes          |null
  |
|25713677   |3.4.12     |null      |89.943435  |b328829e-17ae-11eb-adc1-0242ac120002|fcba68aa-1231-11eb-adc1-0242ac120002|No            |No          |Yes         |No           |null
  |
|83474293   |3.1.8      |null      |-69.939070 |e7bc5fb2-1231-11eb-adc1-0242ac120002|e1e99492-17ae-11eb-adc1-0242ac120002|Yes           |No          |Yes         |No           |null
  |
|63727807   |2.2.9      |null      |64.082108  |e7bc5fb2-1231-11eb-adc1-0242ac120002|fcba68aa-1231-11eb-adc1-0242ac120002|No            |Yes         |Yes         |Yes          |null
  |
|73737907   |4.3.19     |null      |-18.850508 |b328829e-17ae-11eb-adc1-0242ac120002|e1e99492-17ae-11eb-adc1-0242ac120002|No            |Yes         |No          |Yes          |null
  |
|36927433   |3.2.26     |null      |-84.6857245|de545711-3914-4450-8c11-b17b8dabb5e1|a95dd57b-779f-49db-819d-b6960483e554|Yes           |Yes         |No          |Yes          |null
  |
|12691783   |3.3.11     |null      |54.3852925 |de545711-3914-4450-8c11-b17b8dabb5e1|e1e99492-17ae-11eb-adc1-0242ac120002|Yes           |Yes         |No          |No           |null
```

Feedback © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



us-east-1.console.aws.amazon.com/cloudshell/home?region=us-east-1#5bf7746c-8b8c-4830-b897-d49b20c8aee6

Bookmarks Gmail YouTube Maps keyboard shortcuts... New Tab Program Calendar... 5. Data Structures... GitHub - awesome... All Bookmarks

aws Services Search [Alt+S] N. Virginia upgradfelixphilipk @ 6909-8915-0119

AWS CloudShell Actions

us-east-1

```
          |Yes        |No         |null      |                                |                                |              |            |            |             |
|95405928   |1.2.10     |null       |44.185627 |e7bc5fb2-1231-11eb-adc1-0242ac120002|e1e99492-17ae-11eb-adc1-0242ac120002|No            |No
          |No         |No         |null      |                                |                                |              |            |            |             |
|56235860   |1.4.16     |null       |-61.178407|de545711-3914-4450-8c11-b17b8dabb5e1|a95dd57b-779f-49db-819d-b6960483e554|Yes           |Ye
s         |Yes        |Yes        |null      |                                |                                |              |            |            |             |
|81758845   |4.4.30     |null       |78.904984 |e7bc5fb2-1231-11eb-adc1-0242ac120002|a95dd57b-779f-49db-819d-b6960483e554|Yes           |Ye
s         |No         |No         |null      |                                |                                |              |            |            |             |
|46822067   |1.2.26     |null       |45.8993985|de545711-3914-4450-8c11-b17b8dabb5e1|fcba68aa-1231-11eb-adc1-0242ac120002|Yes           |Ye
s         |Yes        |Yes        |null      |                                |                                |              |            |            |             |
|37623067   |3.2.27     |null       |-74.1131505|b328829e-17ae-11eb-adc1-0242ac120002|a95dd57b-779f-49db-819d-b6960483e554|No            |Ye
s         |No         |No         |null      |                                |                                |              |            |            |             |
|14571693   |3.4.21     |null       |-28.8329055|e7bc5fb2-1231-11eb-adc1-0242ac120002|fcba68aa-1231-11eb-adc1-0242ac120002|No            |Ye
s         |No         |No         |null      |                                |                                |              |            |            |             |
|62605529   |2.1.36     |null       |44.196239 |e7bc5fb2-1231-11eb-adc1-0242ac120002|a95dd57b-779f-49db-819d-b6960483e554|Yes           |Ye
s         |No         |No         |null      |                                |                                |              |            |            |             |
+-----------+-----------+----------+-----------+--------------------------------+--------------------------------+--------------+------------+------------+-------------+--
----------+-----------+-----------+----------+
only showing top 20 rows
```

Feedback © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Kafka data transformation and spark_local_flatten python file explanation

- The transformations and processing logic are defined in the Spark local flatten python file. This python file contains the necessary code to transform the raw Kafka data into a structured and usable format.
- Initializing Spark Session for Data Processing

Initializes a Spark session, essential for any Spark application.

Sets the application name to "CleanAndStructureClicksteamData".

Configures the log level to 'ERROR' to minimize log output clutter.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
# Intialize Spark Session
spark = SparkSession \
        .builder \
        .appName("CleanAndStructureClicksteamData")\
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
```

- Defining Data Schema

A schema is defined to specify the structure of the incoming data.

This ensures that Spark correctly interprets the types of each column in the Clickstream data.

spark_local_flatten python file explanation

```
# Define schema
schema = StructType([
    StructField("customer_id",StringType()),
    StructField("app_version",StringType()),
    StructField("os_version",StringType()),
    StructField("lat",StringType()),
    StructField("lon",StringType()),
    StructField("page_id",StringType()),
    StructField("button_id",StringType()),
    StructField("is_button_click",StringType()),
    StructField("is_page_view",StringType()),
    StructField("is_scroll_up",StringType()),
    StructField("is_scroll_down",StringType()),
    StructField("timestamp",StringType()),
])
```

- Reading Data

Reads CSV data from the local file system.
Uses the defined schema and acknowledges that the first row contains headers

```
# Read data from local file system
clickStream = spark.read.csv("/user/ec2-user/Data/ClickStream",schema= schema,header=True)
```

- Data Cleaning and Transformation

The data is cleaned and transformed:
Casting some columns to appropriate data types (e.g., Integer, Double).
Converting certain string values to boolean.
Filtering out records with null customer IDs.
Removing duplicate records.
Ensures data quality and consistency for further analysis.

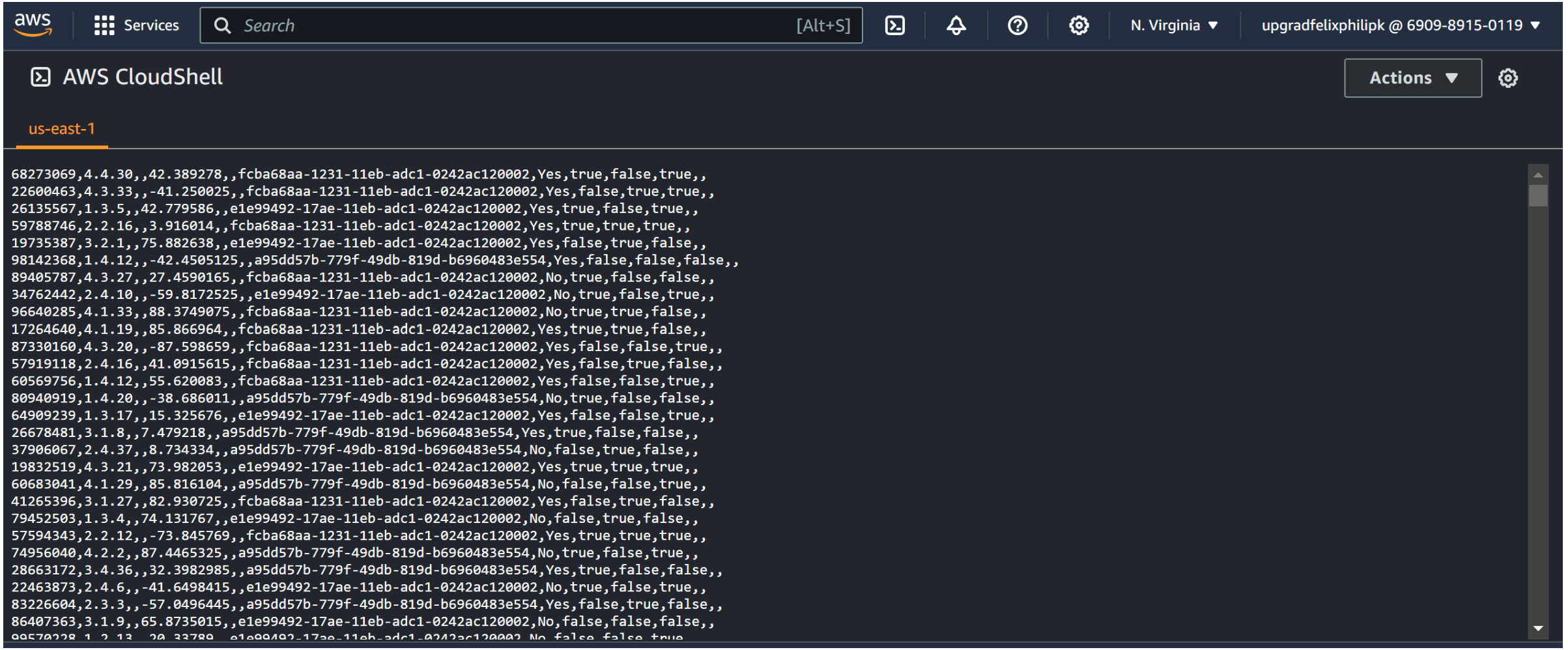
spark_local_flatten python file explaination

```
# Data Cleaning And Transformation
cleanedClickStream = clickStream \
                     .withColumn("customer_id",col("customer_id").cast("Integer"))\
                     .withColumn("lat", col("lat").cast("Double"))\
                     .withColumn("lon", col("lon").cast("Double"))\
                     .withColumn("is_button_click",col("is_button_click")==lit("Yes"))\
                     .withColumn("is_page_view",col("is_page_view")==lit("Yes"))\
                     .withColumn("is_scroll_up",col("is_scroll_up")==lit("Yes"))\
                     .withColumn("is_scroll_down",col("is_scroll_down")==lit("Yes"))\
                     .withColumn("timestamp",to_timestamp(col("timestamp")))\
                     .filter(col("customer_id").isNotNull())\
                     .dropDuplicates()
```

Writing Cleaned Data
Description:
The cleaned and transformed data is written back to the local file system in CSV format.
Overwrites any existing data in the target location.
Includes headers for better readability.
Finally, the Spark session is stopped, releasing the resource

```
# Write Cleaned Data back to local file system
cleanedClickStream.write.csv("/user/ec2-
user/Data/CleanedClickStream",mode="overwrite",header=True)
spark.stop()
```

Output cat of the csv file created after Running spark submit job for spark local flatten python file



Batch Data Ingestion

- Ingesting Data from AWS RDS Using Sqoop

Operation: Batch data is ingested from an AWS Relational Database Service (RDS) instance.

Method: Utilization of Apache Sqoop for efficient data transfer.

Destination: Data is written to the Hadoop Distributed File System (HDFS).

- Creating Hive Table for Booking Data

Data Source: The data now residing in HDFS, originally from AWS RDS.

Action: A new Hive table, named booking, is created.

Purpose: This table facilitates structured querying and analysis of booking data.

- Generating Aggregate Datewise Booking Data

Processing: Aggregation of the booking data on a date-wise basis.

Objective: To create summarized views of bookings for each date, aiding in trend analysis and decision-making.From this data we also create aggregate datewise booking

The aggregated datewise booking data is used to create a hive table

- Creating Hive Table for Aggregated Data

Data Source: The aggregated datewise booking data.

Action: Creation of a second Hive table to store this aggregated data.

Purpose: This table serves as a structured and optimized source for querying and analyzing datewise aggregated booking data.

Date wise booking aggregate spark python file explanation

- Initializing Spark Session for Datewise Booking Analysis

Initializes a Spark session with the application name "datewise_booking".

Sets the log level to "Error" to reduce log output, focusing on error messages only.

```
# Initialize Spark Session
spark = SparkSession \
        .builder \
        .appName("datewise_booking")\
        .getOrCreate()
spark.sparkContext.setLogLevel("Error")
```

- Reading Batch Data

Reads CSV data from the specified path in HDFS.
inferSchema=True allows Spark to automatically detect the data types of each column.

```
# read the batch data
batch_df = spark.read.csv('/user/ec2-user/Data/Booking_Batch_Data/part-m-00000',inferSchema=True)
```

- Renaming Columns

Renames columns from generic names (like _c0, _c1, etc.) to descriptive names.
This step is crucial for readability and ease of data manipulation

Date wise booking aggregate spark python file explanation

```
batch_df = batch_df.withColumnRenamed("_c0","booking_id")\
       .withColumnRenamed("_c1","customer_id") \
       .withColumnRenamed("_c2","driver_id") \
       .withColumnRenamed("_c3","customer_app_version")  \
       .withColumnRenamed("_c4","customer_phone_os_version") \
       .withColumnRenamed("_c5","pickup_lat") \
       .withColumnRenamed("_c6","pickup_lon") \
       .withColumnRenamed("_c7","drop_lat") \
       .withColumnRenamed("_c8","drop_lon") \
       .withColumnRenamed("_c9","pickup_timestamp")  \
       .withColumnRenamed("_c10","drop_timestamp")  \
       .withColumnRenamed("_c11","trip_fare") \
       .withColumnRenamed("_c12","tip_amount")  \
       .withColumnRenamed("_c13","currency_code") \
       .withColumnRenamed("_c14","cab_color")  \
       .withColumnRenamed("_c15","cab_registration_no") \
       .withColumnRenamed("_c16","customer_rating_by_driver")  \
       .withColumnRenamed("_c17","rating_by_customer")  \
       .withColumnRenamed("_c18","passenger_count")
```

- Adding Date Column

New column named date is added.

The date column is extracted from the pickup_timestamp, formatted as "yyyy-MM-dd".

```
# add new column date
batch_df = batch_df.withColumn("date", date_format('pickup_timestamp', "yyyy-MM-dd"))
```

- Aggregating Data

Groups the data by the date column and counts the number of bookings for each date.

This aggregation is key for analyzing datewise booking trends.

Date wise booking aggregate spark python file explanation

```
#Aggregate Data
datewise_total_booking = batch_df.groupBy("date").count()
```

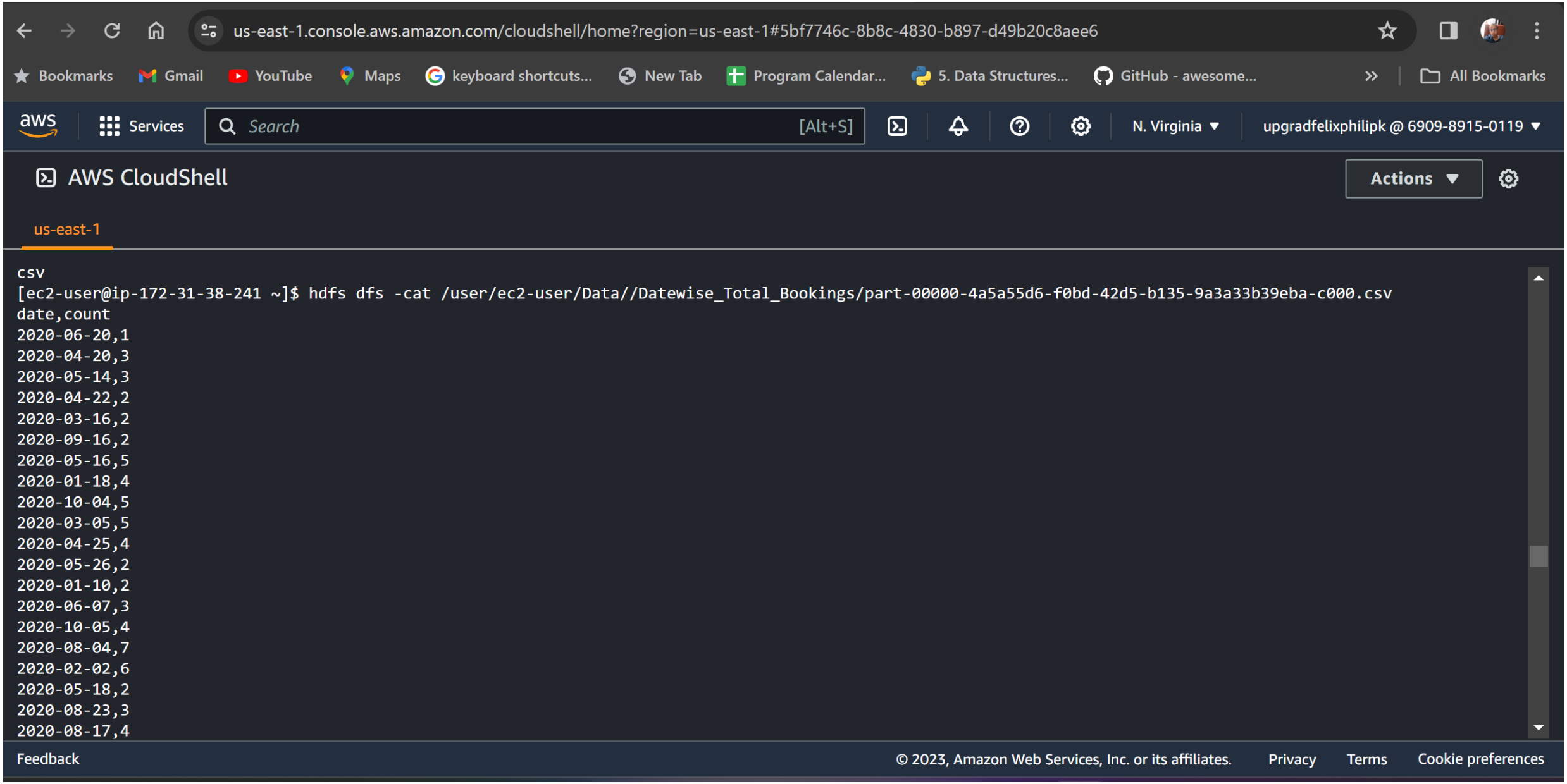
- Saving Aggregated Data and stop the spark session

The aggregated data is saved in CSV format to the specified HDFS path.
Overwrites existing data at the destination, ensuring the latest data is stored.
Includes a header row for better understanding of the data columns.
After writing the aggregated data Stops the Spark session, releasing resources.

```
# Save Aggregated Data
datewise_total_booking.write.format('com.databricks.spark.csv').mode('overwrite').save
('/user/ec2-user/Data/Datewise_Total_Bookings', header = 'true')

spark.stop()
```

Output cat for csv file created after running spark submit job for Date wise booking aggregate spark python file



Creation of hive tables and loading data from the csv files created from spark jobs

- Streaming Data Load to HDFS from Kafka

Operation: Real-time streaming data is ingested from Kafka.
Destination: Data is loaded into the Hadoop Distributed File System (HDFS).
Data Cleaning and Hive Table Loading for Clickstream Data

- Cleaning Process: The streamed data in HDFS is cleaned and structured.

Hive Integration: The cleaned data is then loaded into a Hive table specifically created for clickstream data.
Purpose: This enables structured querying and analysis of clickstream data.

- Batch Data Load from AWS RDS using Sqoop

Operation: Batch data is extracted from AWS RDS.
Method: Apache Sqoop is used for efficient data transfer.
Output: Data is written to a CSV file.

- Loading CSV Data to Hive Table for Bookings

Data Source: The CSV file generated from AWS RDS batch data.
Hive Table: Data from the CSV file is loaded into a Hive table named bookings.
Purpose: Facilitates structured access and querying of booking data.

- Aggregation of Booking Data and Hive Table Creation

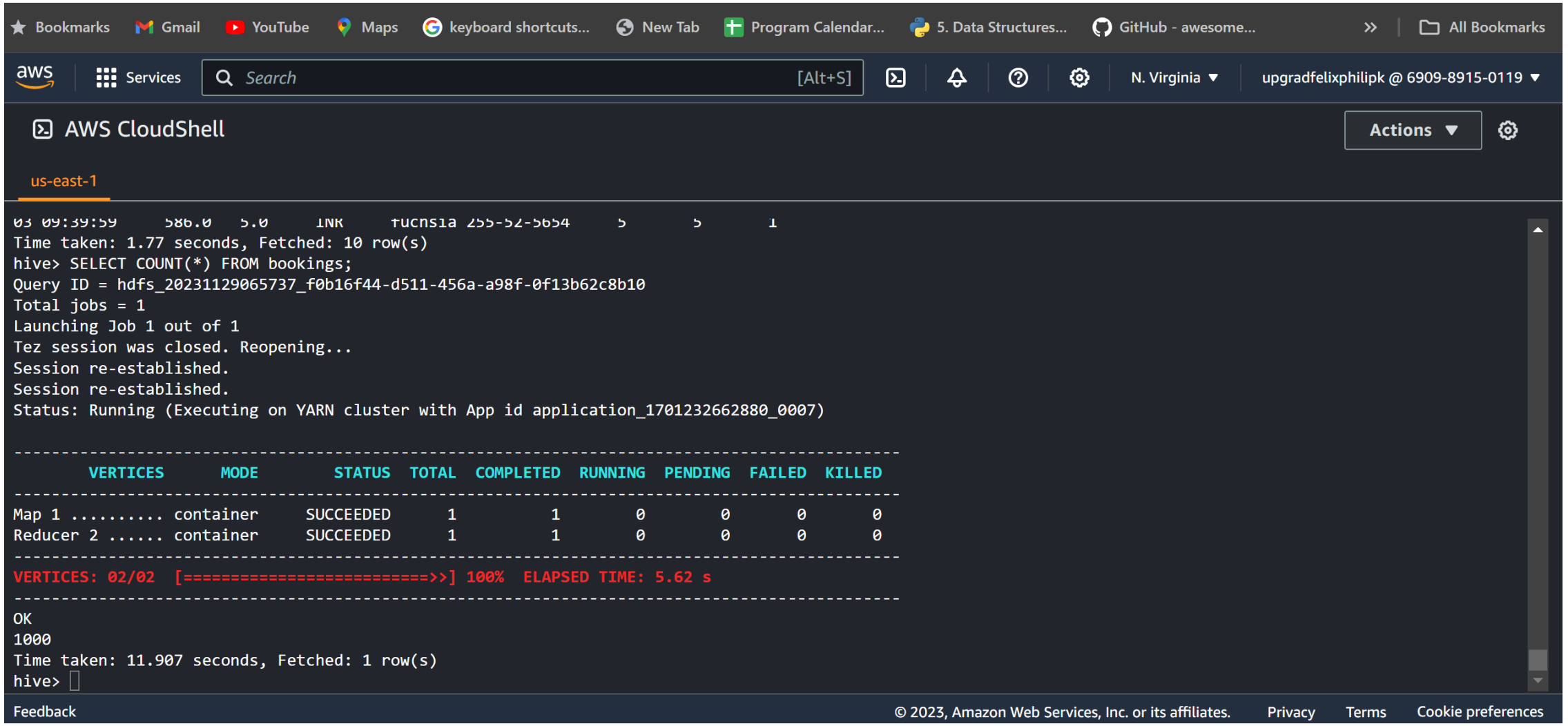
Data Processing: The batch data in the bookings table is aggregated date-wise to calculate total bookings for each date.
Hive Table for Aggregated Data: The aggregated data is then loaded into another Hive table named datewise_total_bookings.
Outcome: This provides a summarized view of bookings, useful for trend analysis and decision-making.

Hive table creation for bookings data

```
CREATE TABLE IF NOT EXISTS bookings(
      booking_id String,
      customer_id String,
      driver_id String,
      customer_app_version String,
      customer_phone_os_version String,
      pickup_lat String,
      pickup_lon String,
      drop_lat String,
      drop_lon String,
      pickup_timestamp timestamp,
      drop_timestamp timestamp,
      trip_fare float,
      tip_amount float,
      currency_code String,
      cab_color String,
      cab_registration_number String,
      customer_rating_by_driver int,
      rating_by_customer int,
      passenger_count int
 )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

Load data inpath '/user/ec2-user/Data/Booking_Batch_Data/part-m-00000' into table bookings;
```

Output after loading data to bookings table in Hive

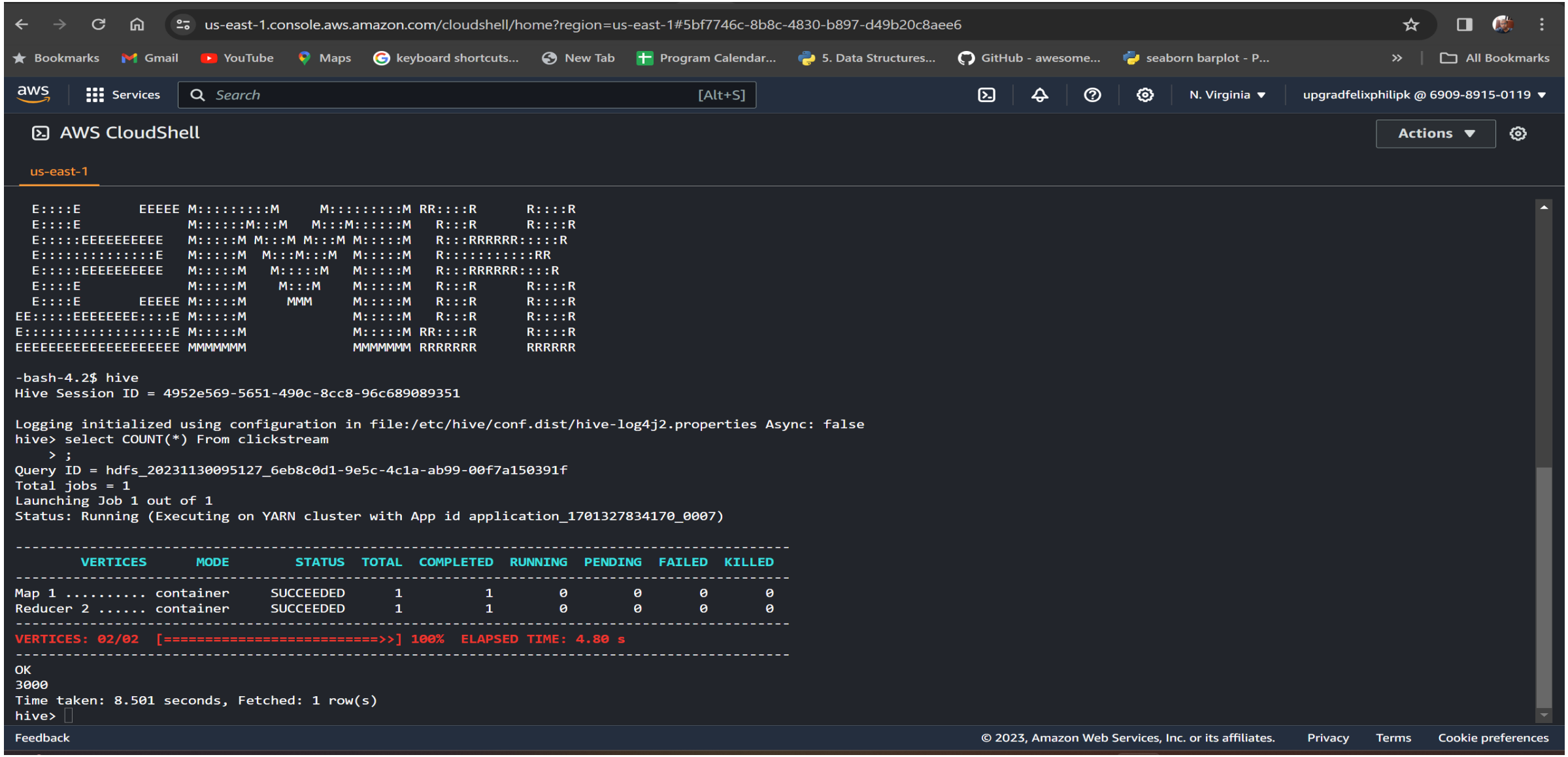


Hive table creation for clickstream data

```
CREATE TABLE IF NOT EXISTS clickstream (
    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,
    `timestamp` timestamp
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

Load data inpath '/user/ec2-user/Data/CleanedClickStream/part-00000-84fe7326-3c8d-44aa-aedc-4417b65f355b-
c000.csv' into table clickstream;
```

Output after loading data to clickstream table in Hive

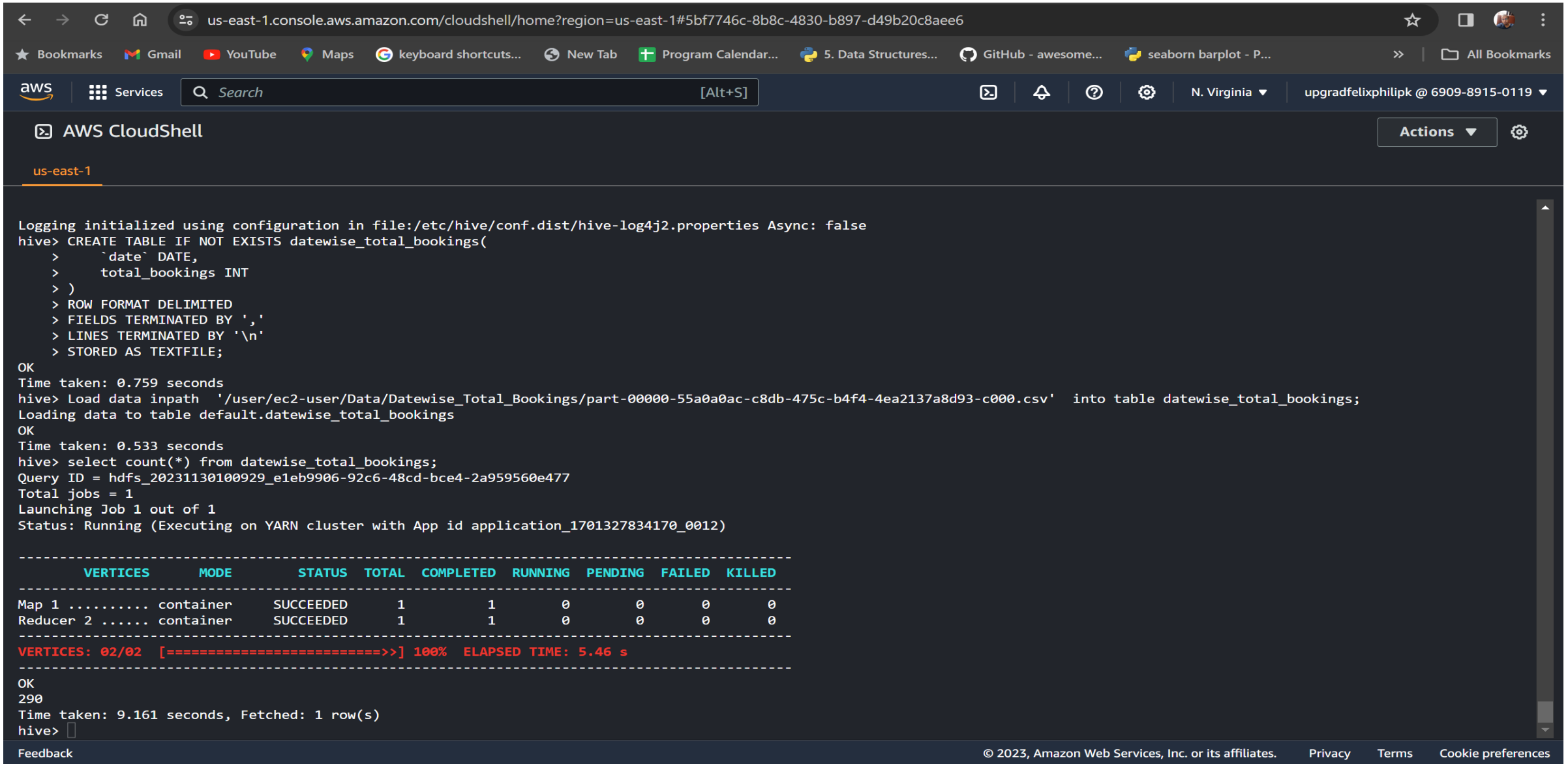


Hive table creation for date wise total bookings

```
CREATE TABLE IF NOT EXISTS datewise_total_bookings(
    `date` DATE,
    total_bookings INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
Load data inpath  '/user/ec2-user/Data/Datewise_Total_Bookings/part-00000-55a0a0ac-c8db-475c-b4f4-4ea2137a8d93-c000.csv'  into table datewise_total_bookings;
```

Output after loading data to date wise total booking table in Hive



Thank You