

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 explanation

- 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 explanation

```
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

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

-----
--t
|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|
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--t
|26564820   |3.2.35        |null              |[6.4454865     |de545711-3914-4450-8c11-b17b8dabb5e1|fcb6a8aa-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|fcb6a8aa-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|fcb6a8aa-1231-11eb-adc1-0242ac120002|No            |Yes         |Yes         |Yes          |null
|73737907   |4.3.19        |null              |[-18.80508     |b328829e-17ae-11eb-adc1-0242ac120002|e1e99492-17ae-11eb-adc1-0242ac120002|No            |Yes         |Yes         |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

The screenshot displays the AWS CloudShell environment. At the top, there's a navigation bar with the AWS logo, a search bar, and a list of services. Below this, the CloudShell interface shows a terminal window with a table of data. The table has 10 columns and 10 rows of data. The first column contains IDs, the second contains dates, and the others contain various alphanumeric strings. The interface includes a top navigation bar with the AWS logo, a search bar, and a list of services. The bottom of the screen shows a feedback button and copyright information.

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

only showing top 20 rows

# 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 "CleanAndStructureClickstreamData".

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("CleanAndStructureClickstreamData")\
    .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 explanation

```
# 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


 Services

[Alt+S]











N. Virginia ▼

upgradfelixphilipk @ 6909-8915-0119 ▼

 AWS CloudShell

Actions ▼ 

us-east-1

```
68273069,4.4.30,,42.389278,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,true,false,true,,
22600463,4.3.33,,41.250025,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,false,true,true,,
26135567,1.3.5,,42.779586,,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,true,false,true,,
59788746,2.2.16,,3.916014,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,true,true,true,,
19735387,3.2.1,,75.882638,,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,false,true,false,,
98142368,1.4.12,,42.4505125,,a95dd57b-779f-49db-819d-b6960483e554,Yes,false,false,false,,
89405787,4.3.27,,27.4590165,,fcba68aa-1231-11eb-adc1-0242ac120002,No,true,false,false,,
34762442,2.4.10,,59.8172525,,e1e99492-17ae-11eb-adc1-0242ac120002,No,true,false,true,,
96640285,4.1.33,,88.3749075,,fcba68aa-1231-11eb-adc1-0242ac120002,No,true,true,false,,
17264640,4.1.19,,85.866964,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,true,true,false,,
87330160,4.3.20,,87.598659,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,false,false,true,,
57919118,2.4.16,,41.0915615,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,false,true,false,,
60569756,1.4.12,,55.620083,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,false,false,true,,
80940919,1.4.20,,38.686011,,a95dd57b-779f-49db-819d-b6960483e554,No,true,false,false,,
64909239,1.3.17,,15.325676,,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,false,false,true,,
26678481,3.1.8,,7.479218,,a95dd57b-779f-49db-819d-b6960483e554,Yes,true,false,false,,
37906067,2.4.37,,8.734334,,a95dd57b-779f-49db-819d-b6960483e554,No,false,true,false,,
19832519,4.3.21,,73.982053,,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,true,true,true,,
60683041,4.1.29,,85.816104,,a95dd57b-779f-49db-819d-b6960483e554,No,false,false,true,,
41265396,3.1.27,,82.930725,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,false,true,false,,
79452503,1.3.4,,74.131767,,e1e99492-17ae-11eb-adc1-0242ac120002,No,false,true,false,,
57594343,2.2.12,,73.845769,,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,true,true,true,,
74956040,4.2.2,,87.4465325,,a95dd57b-779f-49db-819d-b6960483e554,No,true,false,true,,
28663172,3.4.36,,32.3982985,,a95dd57b-779f-49db-819d-b6960483e554,Yes,true,false,false,,
22463873,2.4.6,,41.6498415,,e1e99492-17ae-11eb-adc1-0242ac120002,No,true,false,true,,
83226604,2.3.3,,57.0496445,,a95dd57b-779f-49db-819d-b6960483e554,Yes,false,true,false,,
86407363,3.1.9,,65.8735015,,e1e99492-17ae-11eb-adc1-0242ac120002,No,false,false,false,,
89570228,1.2.13,,20.33780,,e1e99492-17ae-11eb-adc1-0242ac120002,No,false,false,true,,
```

# 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

←

→

↺

🏠

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

Services

🔍

Search

[Alt+S]

N. Virginia ▼

upgradfelixphilipk @ 6909-8915-0119 ▼

AWS CloudShell

Actions ▼

us-east-1

csv

[ec2-user@ip-172-31-38-241 ~]\$ hdfs dfs -cat /user/ec2-user/Data//Datewise\_Total\_Bookings/part-00000-4a5a55d6-f0bd-42d5-b135-9a3a33b39eba-c000.csv

date,count

2020-06-20,1

2020-04-20,3

2020-05-14,3

2020-04-22,2

2020-03-16,2

2020-09-16,2

2020-05-16,5

2020-01-18,4

2020-10-04,5

2020-03-05,5

2020-04-25,4

2020-05-26,2

2020-01-10,2

2020-06-07,3

2020-10-05,4

2020-08-04,7

2020-02-02,6

2020-05-18,2

2020-08-23,3

2020-08-17,4

Feedback

© 2023, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

# 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

☆ Bookmarks

Gmail

YouTube

Maps

keyboard shortcuts...

New Tab

Program Calendar...

5. Data Structures...

GitHub - awesome...

>> | All Bookmarks

aws

Services

[Alt+S]

N. Virginia ▾

upgradfelixphilipk @ 6909-8915-0119 ▾

AWS CloudShell

Actions ▾

us-east-1

03 09:39:59 586.0 5.0 INK tUCNS1a 255-52-5654 5 5 1

Time taken: 1.77 seconds, Fetched: 10 row(s)

hive> SELECT COUNT(\*) FROM bookings;

Query ID = hdfs\_20231129065737\_f0b16f44-d511-456a-a98f-0f13b62c8b10

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\_1701232662880\_0007)

	VERTICES	MODE	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1 .....	container	SUCCEEDED	1	1	0	0	0	0	0
Reducer 2 .....	container	SUCCEEDED	1	1	0	0	0	0	0

VERTICES: 02/02 [=====>>] 100% ELAPSED TIME: 5.62 s

OK

1000

Time taken: 11.907 seconds, Fetched: 1 row(s)

hive>

Feedback

© 2023, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

## 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

←→↺🏠

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... seaborn barplot - P...

» All Bookmarks

aws

Services

Q Search

[Alt+S]

📺🔔🔍⚙️

N. Virginia ▼

upgradfelixphilipk @ 6909-8915-0119 ▼

AWS CloudShell

Actions ▼ ⚙️

us-east-1

E:::E EEEEE M:::M M:::M RR:::R R:::R

E:::E M:::M M:::M M:::M R:::R R:::R

E:::EEEEEEEEEE M:::M M:::M M:::M R:::RRRRRR:::R

E:::M:::M M:::M M:::M R:::RRRRRR:::R

E:::EEEEEEEEEE M:::M M:::M M:::M R:::RRRRRR:::R

E:::E M:::M M:::M M:::M R:::R R:::R

E:::E EEEEE M:::M MMM M:::M R:::R R:::R

EE:::EEEEEEEE:::E M:::M M:::M R:::R R:::R

E:::M:::M M:::M RR:::R R:::R

EEEEEEEEEEEEEEEE MMMMMM MMMMMM RRRRRR RRRRRR

-bash-4.2\$ hive

Hive Session ID = 4952e569-5651-490c-8cc8-96c689089351

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false

hive> select COUNT(\*) From clickstream

> ;

Query ID = hdfs\_20231130095127\_6eb8c0d1-9e5c-4c1a-ab99-00f7a150391f

Total jobs = 1

Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application\_1701327834170\_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: 4.80 s

OK

3000

Time taken: 8.501 seconds, Fetched: 1 row(s)

hive>

Feedback

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

## 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

←→↺🏠

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... 📊 seaborn barplot - P...

📌 All Bookmarks

aws

Services

🔍 Search

[Alt+S]

📧 🔔 ⓘ ⚙️

N. Virginia ▼

upgradfelixphilipk @ 6909-8915-0119 ▼

📌 AWS CloudShell

Actions ▼ ⚙️

us-east-1

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> 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;
OK
Time taken: 0.759 seconds
hive> Load data inpath  '/user/ec2-user/Data/Datewise_Total_Bookings/part-00000-55a0a0ac-c8db-475c-b4f4-4ea2137a8d93-c000.csv'  into table datewise_total_bookings;
Loading data to table default.datewise_total_bookings
OK
Time taken: 0.533 seconds
hive> select count(*) from datewise_total_bookings;
Query ID = hdfs_20231130100929_e1eb9906-92c6-48cd-bce4-2a959560e477
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701327834170_0012)

-----
      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: 5.46 s
-----
OK
290
Time taken: 9.161 seconds, Fetched: 1 row(s)
hive> 
```

Feedback

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

Thank You