Hive Case Study

By
- Amber Kaushal (amber.kaushal10@gmail.com)

Problem Statement:-

With online sales gaining popularity, tech companies are exploring ways to improve their sales by analysing customer behaviour and gaining insights about product trends. Furthermore, the websites make it easier for customers to find the products they require without much scavenging. The clickstream data contains all the logs as to how you navigated through the website. From this, machine learning engineers or business analysts use this data to derive valuable insights.

Objective:-

For this Case Study, We have to work with a public clickstream dataset of a cosmetics store. Using this dataset, our job is to extract valuable insights which generally data engineers come up within an e-retail company. And required to provide answers to the questions given below:

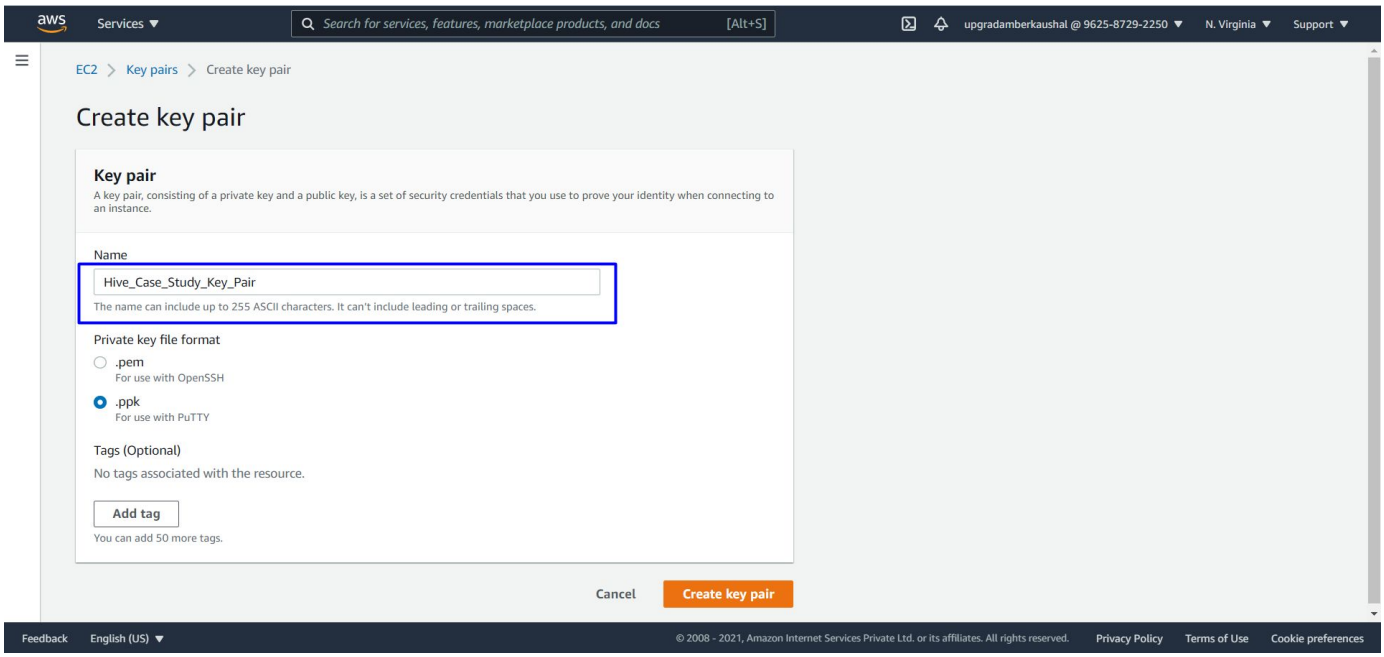
- Find the total revenue generated due to purchases made in October.
- Write a query to yield the total sum of purchases per month in a single output.
- Write a query to find the change in revenue generated due to purchases from October to November.
- Find distinct categories of products. Categories with null category code can be ignored.
- Find the total number of products available under each category.
- Which brand had the maximum sales in October and November combined?
- Which brands increased their sales from October to November?
- Your company wants to reward the top 10 users of its website with a Golden Customer plan. Write a query to generate a list of top 10 users who spend the most.

Index

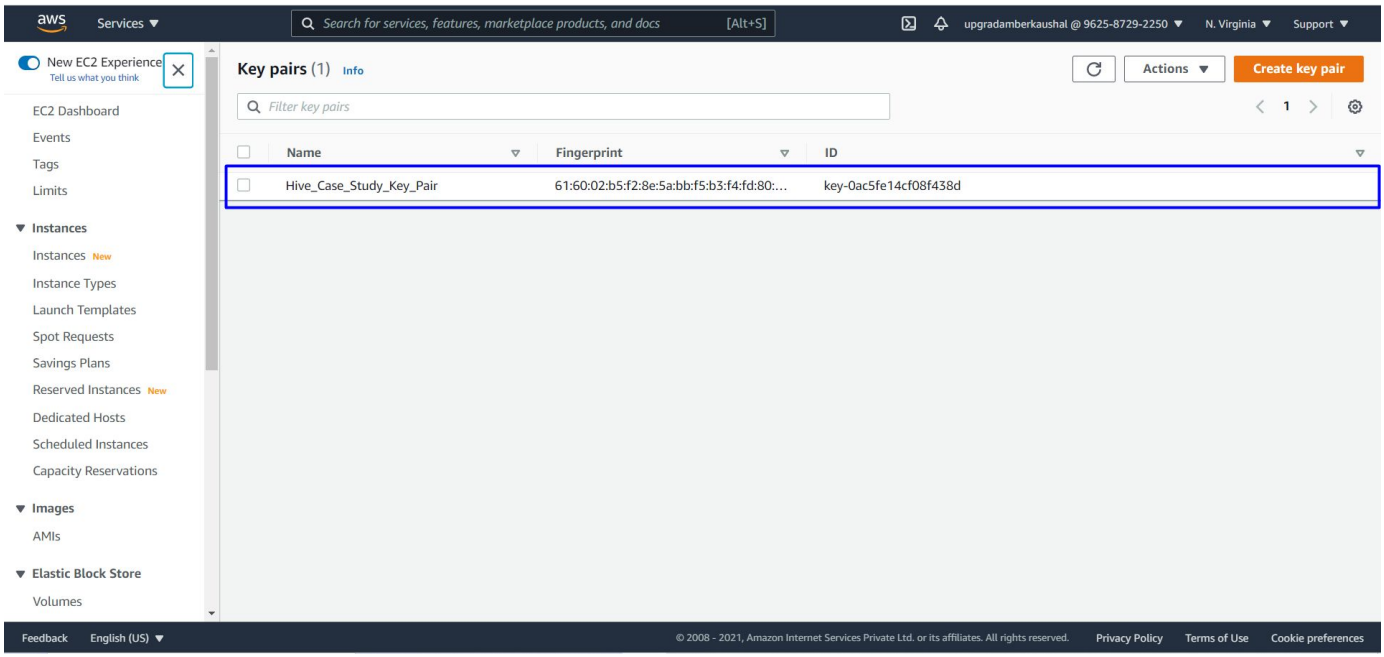
1. EMR Cluster Creation

a. Creating Key-Pair

First, we have to create Key-Pair with ppk format.

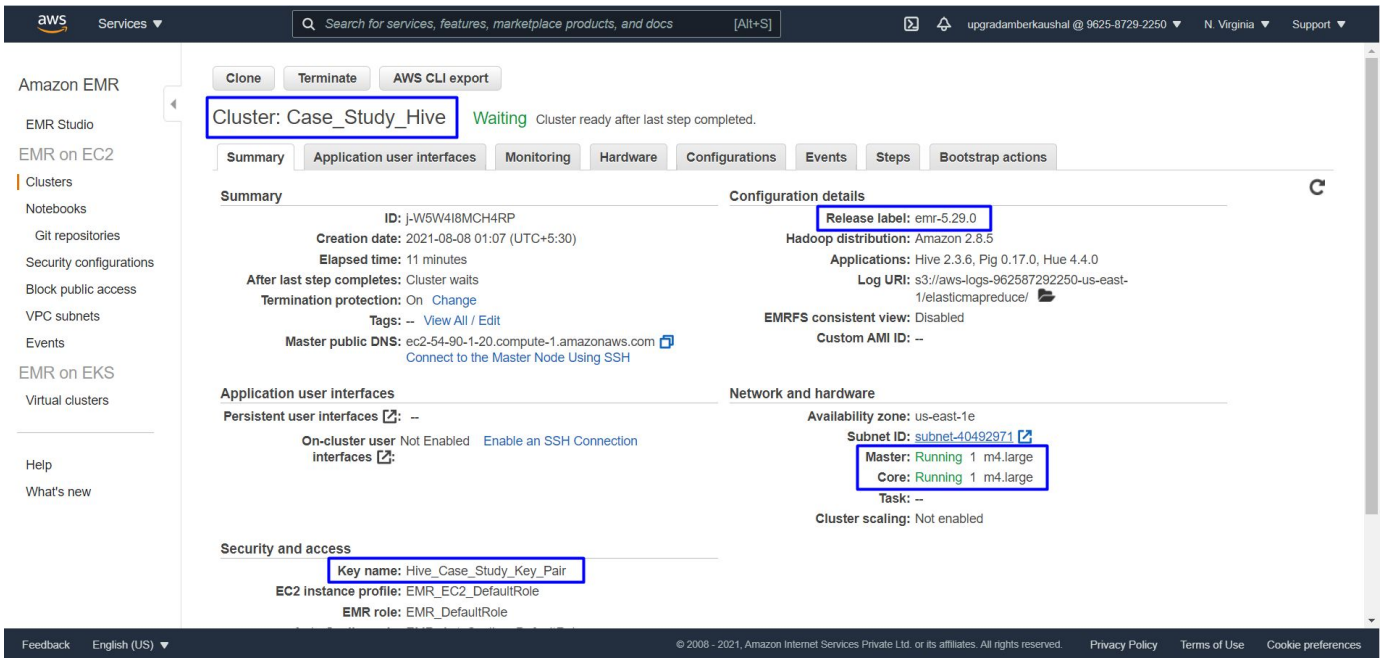


We have successfully created and downloaded the key-pair "**Hive_Case_Study_Key_Pair**". Please refer to below screenshot.

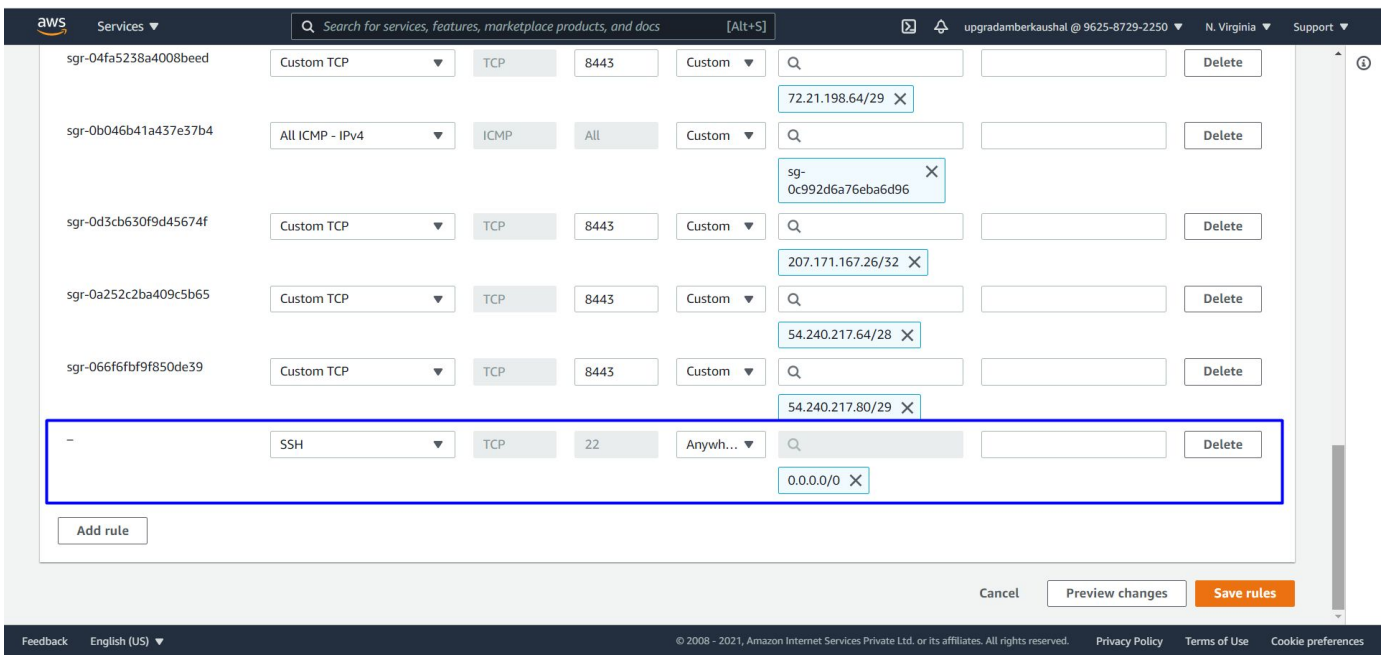


b. Cluster Creation

We have created the cluster “Case_Study_Hive” with **emr-5.29.0** release and used a **2-node** EMR cluster with both the master and core nodes as **M4.large**.

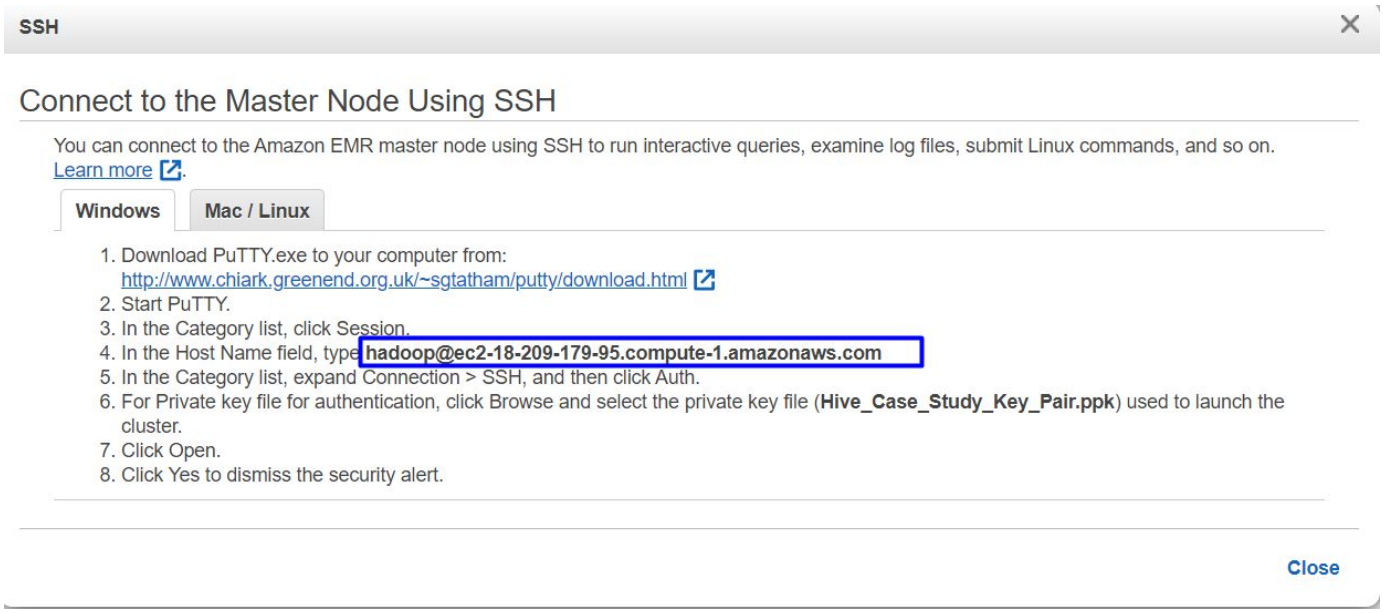


Now set the SSH rule to inbound rule for the Master node, if not created yet.

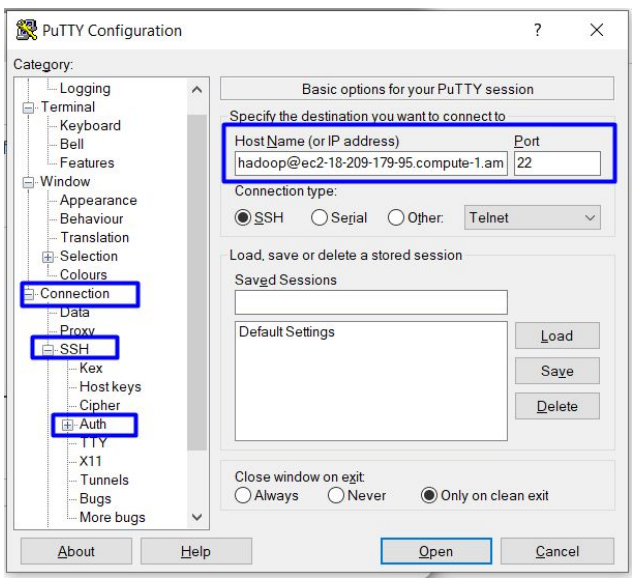


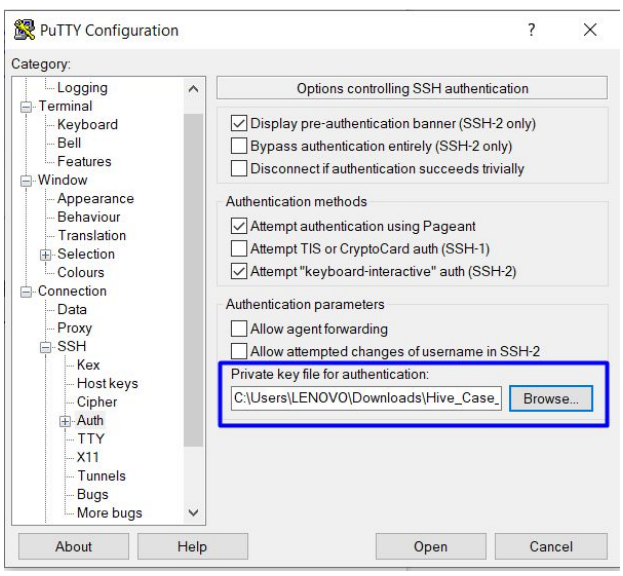
c. Connect The Master Node Through PUTTY

After the cluster reddy, we have to click on Master public DNS and copy the hostname **hadoop@ec2-18-209-179-95.compute-1.amazonaws.com.**



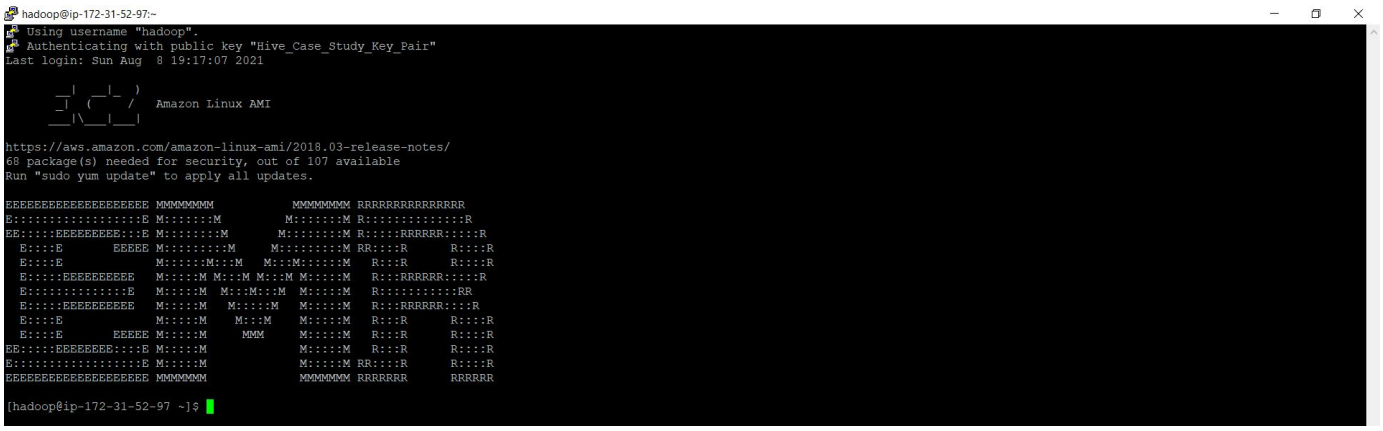
After the cluster reddy, we have to click on Master public DNS and copy the hostname **hadoop@ec2-18-209-179-95.compute-1.amazonaws.com.**





After the cluster reddy, we have to click on Master public DNS and copy the hostname **hadoop@ec2-18-209-179-95.compute-1.amazonaws.com.**

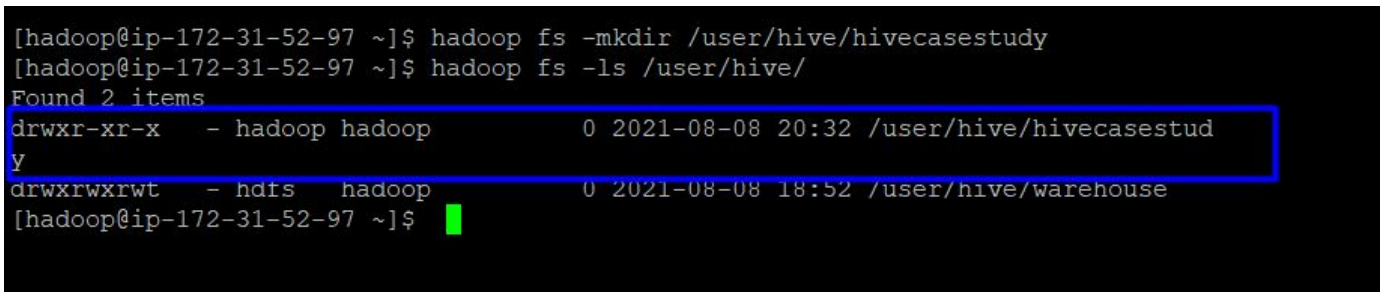
EMR CLI launched Successfully. Please Refer the Screenshot below:



2. Create New Directory

- We have to create a new directory "**hivecasestudy**" to load the data into the HDFS from s3.
- **Command:** hadoop fs -mkdir /user/hive/hivecasestudy

- And verifying whether new directory created or not.
- **Command:** hadoop fs -ls /user/hive/



The directory created successfully

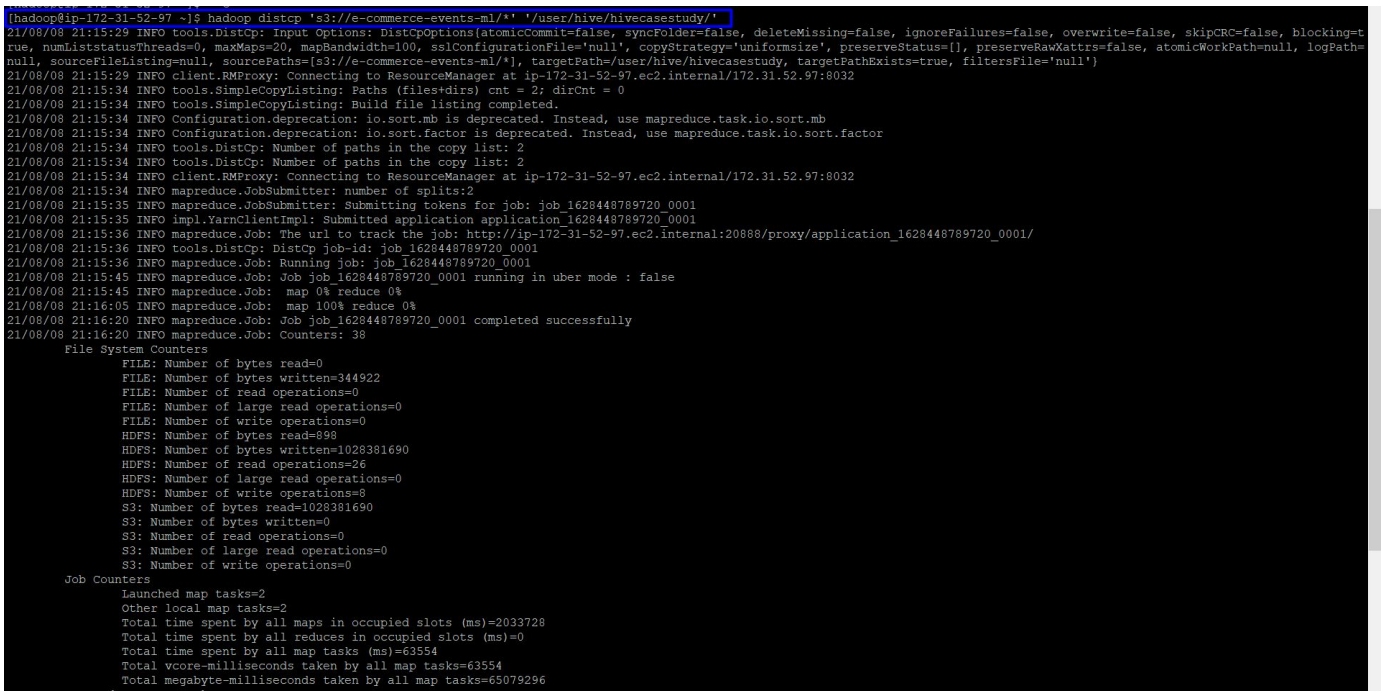
To load the datasets into HDFS from S3, for that purpose we are going to use distributed copy command.

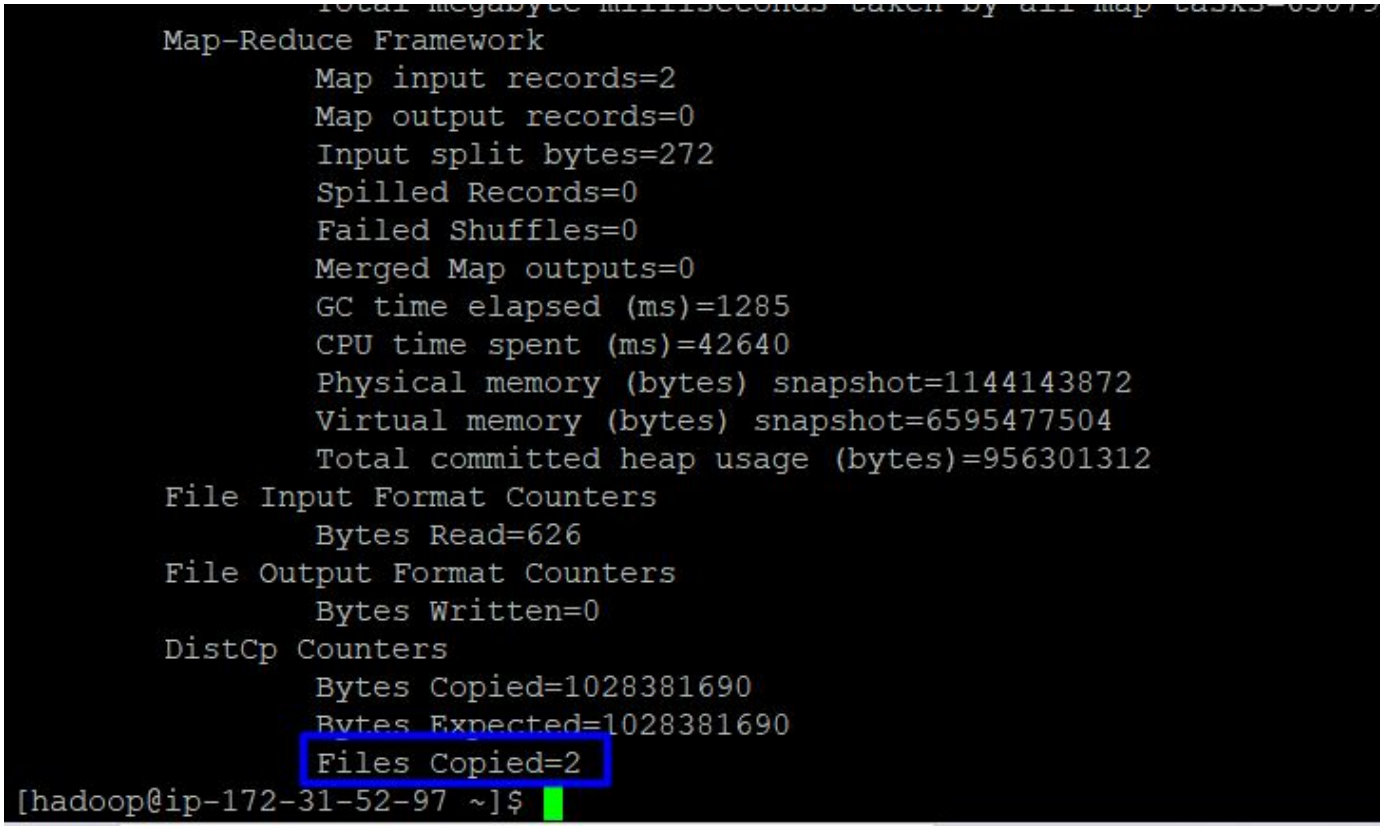
3. Load the Datasets into HDFS from S3

To load the datasets into HDFS from S3, for that purpose we are going to use distributed copy command.

Command:

hadoop distcp 's3://e-commerce-events-ml/*' '/user/hive/hivecasestudy/'





To Check the data sets in directory

Command: hadoop fs -ls /user/hive/hivecasestudy/

```
[hadoop@ip-172-31-52-97 ~]$ hadoop fs -ls /user/hive/hivecasestudy/
Found 2 items
-rw-r--r--   1 hadoop hadoop  545839412 2021-08-08 21:16 /user/hive/hivecasestudy/2019-Nov.csv
-rw-r--r--   1 hadoop hadoop  482542278 2021-08-08 21:16 /user/hive/hivecasestudy/2019-Oct.csv
[hadoop@ip-172-31-52-97 ~]$
```

Files copied successfully into HDFS from S3 buckets

Showing the data in HDFS

Command: hadoop fs -cat /user/hive/hivecasestudy/2019-Nov.csv | head

hadoop fs -cat /user/hive/hivecasestudy/2019-Oct.csv | head

```
[hadoop@ip-172-31-52-97 ~]$ hadoop fs -cat /user/hive/hivecasestudy/2019-Nov.csv | head
event_time,event_type,product_id,category_id,category_code,brand,price,user_id,user_session
2019-11-01 00:00:02 UTC,view,5802432,1487580009286598681,,,0.32,562076640,09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:09 UTC,cart,5844397,1487580006317032337,,,2.38,553329724,2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:10 UTC,view,5837166,1783999064103190764,,pnb,22.22,556138645,57ed222e-a54a-4907-9944-5a875c2d7f4f
2019-11-01 00:00:11 UTC,cart,5876812,1487580010100293687,,jessnail,3.16,564506666,186c1951-8052-4b37-adce-dd9644b1d5f7
2019-11-01 00:00:24 UTC,remove_from_cart,5826182,1487580007483048900,,,3.33,553329724,2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:24 UTC,remove_from_cart,5826182,1487580007483048900,,,3.33,553329724,2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:25 UTC,view,5856189,1487580009026551821,,runail,15.71,562076640,09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:32 UTC,view,5837835,1933472286753424063,,,3.49,514649199,432a4e95-375c-4b40-bd36-0fc039e77580
2019-11-01 00:00:34 UTC,remove_from_cart,5870838,1487580007675986893,,milv,0.79,429913900,2f0bff3c-252f-4fe6-afcd-5d8a6a92839a
cat: Unable to write to output stream.
[hadoop@ip-172-31-52-97 ~]$ hadoop fs -cat /user/hive/hivecasestudy/2019-Oct.csv | head
event_time,event_type,product_id,category_id,category_code,brand,price,user_id,user_session
2019-10-01 00:00:00 UTC,cart,5773203,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-4778-8d2c-92e149dab885
2019-10-01 00:00:03 UTC,cart,5773353,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-4778-8d2c-92e149dab885
2019-10-01 00:00:07 UTC,cart,5881589,2151191071051219817,,lovely,13.48,429681830,49e8d843-adf3-428b-a2c3-fe8bc6a307c9
2019-10-01 00:00:07 UTC,cart,5723490,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-4778-8d2c-92e149dab885
2019-10-01 00:00:15 UTC,cart,5881449,1487580013522845895,,lovely,0.56,429681830,49e8d843-adf3-428b-a2c3-fe8bc6a307c9
2019-10-01 00:00:16 UTC,cart,5857269,1487580005134238553,,runail,2.62,430174032,73dea1e7-664e-43f4-8b30-d32b9d5af04f
2019-10-01 00:00:19 UTC,cart,5739055,1487580008246412266,,kapous,4.75,377667011,81326ac6-daa4-4f0a-b488-fd0956a78733
2019-10-01 00:00:24 UTC,cart,5825598,1487580009445982239,,,0.56,467916806,2f5b5546-b8cb-9ee7-7ecd-84276f8ef486
2019-10-01 00:00:25 UTC,cart,5698989,1487580006317032337,,,1.27,385985999,d30965e8-1101-44ab-b45d-cc1bb9fae694
cat: Unable to write to output stream.
[hadoop@ip-172-31-52-97 ~]$
```

4. Launch Hive

We have added the data into the directory. Now we are going to launch hive

Command: hive

```
[hadoop@ip-172-31-51-218 ~]$ hive

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

Let's view the databases

Command: show databases ;

```
hive> show databases ;
OK
default
Time taken: 0.655 seconds, Fetched: 1 row(s)
hive>
```

We can see that only the default database is present. We have to create the database first.

5. Create The Database - clickstream

Command: CREATE DATABASE IF NOT EXISTS clickstream ;
SHOW DATABASES ;
DESCRIBE DATABASE clickstream ;
USE clickstream ;

```
hive> CREATE DATABASE IF NOT EXISTS clickstream ;
OK
Time taken: 0.762 seconds
```

```
hive> SHOW DATABASES ;
OK
clickstream
default
Time taken: 0.687 seconds, Fetched: 2 row(s)
hive> DESCRIBE DATABASE clickstream ;
OK
clickstream             hdfs://ip-172-31-51-218.ec2.internal:8020/user/hive/warehouse/clickstream.db    hadoop  USER
Time taken: 0.073 seconds, Fetched: 1 row(s)
hive> USE clickstream ;
OK
Time taken: 0.047 seconds
hive>
```

6. Creating The External Table - retaildata

Command: CREATE EXTERNAL TABLE IF NOT EXISTS retaildata (event_time timestamp, event_type string, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE LOCATION '/user/hive/hivecasestudy' TBLPROPERTIES ("skip.header.line.count"="1");

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS retaildata (event_time timestamp, even
t_type string, product_id string, category_id string, category_code string, bran
d string, price float, user_id bigint, user_session string) ROW FORMAT SERDE 'or
g.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE LOCATION '/user/hiv
e/hivecasestudy' TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.801 seconds
hive>
```

Describe the table
Command: DESCRIBE retaildata ;

```
hive> DESCRIBE retaildata ;
OK
col_name        data_type       comment
event_time              string                  from deserializer
event_type              string                  from deserializer
product_id              string                  from deserializer
category_id             string                  from deserializer
category_code           string                  from deserializer
brand                   string                  from deserializer
price                   string                  from deserializer
user_id                 string                  from deserializer
user_session            string                  from deserializer
Time taken: 0.065 seconds, Fetched: 9 row(s)
hive>
```

To display the header of the table

Command: SET hive.cli.print.header = true;

Viewing the first five rows of the table

Command: SELECT * FROM retaildata LIMIT 5 ;

```
hive> SELECT * FROM retaildata LIMIT 5 ;
OK
retaildata.event_time   retaildata.event_type   retaildata.product_id   retailda
ta.category_id  retaildata.category_code        retaildata.brand        retailda
ta.price        retaildata.user_id      retaildata.user_session
2019-11-01 00:00:02 UTC view    5802432 1487580009286598681                     0
.32     562076640       09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:09 UTC cart    5844397 1487580006317032337                     2
.38     553329724       2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:10 UTC view    5837166 1783999064103190764             pnb     2
2.22    556138645       57ed222e-a54a-4907-9944-5a875c2d7f4f
2019-11-01 00:00:11 UTC cart    5876812 1487580010100293687             jessnail
3.16    564506666       186c1951-8052-4b37-adce-dd9644b1d5f7
2019-11-01 00:00:24 UTC remove_from_cart        5826182 14875800074830489000    3
.33     553329724       2067216c-31b5-455d-a1cc-af0575a34ffb
Time taken: 2.178 seconds, Fetched: 5 row(s)
hive>
```

Our base table is ready (retaildata), now we can run a query, we are taking Question one for testing purposes i.e. Find the total revenue generated due to purchases made in October.

Command: SELECT SUM(price) AS total_revenue FROM retaildata WHERE MONTH(event_time)=’10’ AND event_type='purchase';

```
hive> SELECT SUM(price) AS total_revenue FROM retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';
Query ID = hadoop_20210809212550_1957088b-ac77-44b8-94a2-38a341cc5bc5
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628530574915_0004)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      2          2        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 64.83 s
----------------------------------------------------------------------------------------------
OK
total_revenue
1211538.4299997438
Time taken: 65.517 seconds, Fetched: 1 row(s)
hive>
```

Time Taken to execute the above query is 65.51 sec.

7. Partitioning & Bucketing

Time taken to compute that query on base table is too high it's time to optimize the table by partitioning and bucketing, we will try different combinations of partitioning & bucketing and execute the same query until we get an optimized table.

To enable the dynamic partitioning and bucketing we are going to use below commands

Command: SET hive.exec.dynamic.partition.mode = nonstrict;
SET hive.exec.dynamic.partition = true;
SET hive.enforce.bucketing = true ;

```
hive> SET hive.exec.dynamic.partition.mode = nonstrict;
hive> SET hive.exec.dynamic.partition = true;
hive> SET hive.enforce.bucketing = true ;
hive>
```

a. Creating Partition Table 1 - part_1_retaildata

We are going to partition on "event_type" and clustered by "price".

Command: CREATE TABLE IF NOT EXISTS part_1_retaildata (event_time timestamp, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) PARTITIONED BY (event_type string) CLUSTERED BY (price) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE ;

```
hive> CREATE TABLE IF NOT EXISTS part_1_retaildata (event_time timestamp, produc
t_id string, category_id string, category_code string, brand string, price float
, user_id bigint, user_session string) PARTITIONED BY (event_type string) CLUSTE
RED BY (price) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.O
penCSVSerde' STORED AS TEXTFILE  ;
OK
Time taken: 0.069 seconds
```

Describe the newly created table

Command: DESCRIBE part_1_retaildata ;

```
hive> DESCRIBE part_1_retaildata ;
OK
col_name        data_type       comment
event_time              string                  from deserializer
product_id              string                  from deserializer
category_id             string                  from deserializer
category_code           string                  from deserializer
brand                   string                  from deserializer
price                   string                  from deserializer
user_id                 string                  from deserializer
user_session            string                  from deserializer
event_type              string

# Partition Information
# col_name              data_type               comment

event_type              string
Time taken: 0.13 seconds, Fetched: 14 row(s)
hive>
```

Now Inserting the data into the new optimized table (part_1_retaildata) from base table (retaildata)

Command: INSERT INTO TABLE part_1_retaildata PARTITION (event_type) SELECT event_time, product_id, category_id, category_code, brand, price, user_id, user_session, event_type FROM retaildata ;

```
hive> INSERT INTO TABLE part_1_retaildata PARTITION (event_type) SELEC
T event_time, product_id, category_id, category_code, brand, price, us
er_id, user_session, event_type FROM retaildata ;
Query ID = hadoop_20210810212944_c9c7f369-328a-49d6-b690-e3325cf3daef
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_162
8624052120_0012)

Map 1: 0/2      Reducer 2: 0/5
Map 1: 0/2      Reducer 2: 0/5
Map 1: 0(+1)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
```

```
Loading data to table clickstream.part_1_retaildata partition (event_t
ype=null)


         Time taken to load dynamic partitions: 0.344 seconds
         Time taken for adding to write entity : 0.004 seconds
OK
event_time      product_id      category_id     category_code   brandp
rice    user_id user_session    event_type
Time taken: 180.394 seconds
hive>
```

Now running the same query on the optimized table and note the time.

Command: SELECT SUM(price) AS total_revenue FROM part_1_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';

```
hive> SELECT SUM(price) AS total_revenue FROM part_1_retaildata WHERE
MONTH(event_time)='10' AND event_type='purchase';
Query ID = hadoop_20210810213731_7a7aafeb-6fce-4cda-a883-3461afde6404
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_162
8624052120_0012)

Map 1: 0/3      Reducer 2: 0/1
Map 1: 0/3      Reducer 2: 0/1
Map 1: 0/3      Reducer 2: 0/1
Map 1: 0(+1)/3  Reducer 2: 0/1
Map 1: 0(+2)/3  Reducer 2: 0/1
Map 1: 0(+3)/3  Reducer 2: 0/1
Map 1: 0(+3)/3  Reducer 2: 0/1
Map 1: 0(+3)/3  Reducer 2: 0/1
Map 1: 0(+3)/3  Reducer 2: 0/1
Map 1: 0(+3)/3  Reducer 2: 0/1
Map 1: 1(+2)/3  Reducer 2: 0(+1)/1
Map 1: 1(+2)/3  Reducer 2: 0(+1)/1
Map 1: 2(+1)/3  Reducer 2: 0(+1)/1
Map 1: 3/3      Reducer 2: 0(+1)/1
Map 1: 3/3      Reducer 2: 1/1
OK
total_revenue
1211538.430000279
Time taken: 25.211 seconds, Fetched: 1 row(s)
hive>
```

Time Taken to execute the above query is 25.211 sec.

b. Creating Partition Table 2 - part_2_retaildata

We are going to partition on "month" and clustered by "brand".
Command: CREATE TABLE IF NOT EXISTS part_2_retaildata (event_time timestamp, event_type string, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) PARTITIONED BY (month int) CLUSTERED BY (brand) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE ;

```
hive> CREATE TABLE IF NOT EXISTS part_2_retaildata (event_time timestamp, event_
type string, product_id string, category_id string, category_code string, brand
string, price float, user_id bigint, user_session string) PARTITIONED BY (month
int) CLUSTERED BY (brand) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hi
ve.serde2.OpenCSVSerde' STORED AS TEXTFILE ;
OK
Time taken: 0.104 seconds
hive>
```

Now Inserting the data into the new optimized table (part_2_retaildata) from base table (retaildata)
Command: INSERT INTO TABLE part_2_retaildata PARTITION (month) SELECT event_time, event_type, product_id, category_id, category_code, brand, price, user_id, user_session, MONTH(CAST(REPLACE(event_time,'UTC',' ') AS timestamp)) FROM retaildata ;

```
hive> INSERT INTO TABLE part_2_retaildata PARTITION (month) SELECT event_time, e
vent_type, product_id, category_id, category_code, brand, price, user_id, user_s
ession, MONTH(CAST(REPLACE(event_time,'UTC','') AS timestamp)) FROM retaildata ;

Query ID = hadoop_20210811174356_36cc4fc7-21f9-45fb-a6be-33ca36ac89e2
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628701152532
_0003)

Map 1: 0/2      Reducer 2: 0/5
Map 1: 0/2      Reducer 2: 0/5
Map 1: 0(+1)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
Map 1: 0(+2)/2  Reducer 2: 0/5
```

```
Map 1: 2/2      Reducer 2: 3(+2)/5
Map 1: 2/2      Reducer 2: 4(+1)/5
Map 1: 2/2      Reducer 2: 4(+1)/5
Map 1: 2/2      Reducer 2: 5/5
Loading data to table clickstream.part_2_retaildata partition (month=null)


         Time taken to load dynamic partitions: 0.152 seconds
         Time taken for adding to write entity : 0.0 seconds
OK
event_time      event_type      product_id      category_id     category_code  b
rand    price   user_id user_session    _c9
Time taken: 177.59 seconds
hive>
```

Now running the same query on the optimized table 2 and note the time.
Command: SELECT SUM(price) AS total_revenue FROM part_2_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';

```
hive> SELECT SUM(price) AS total_revenue FROM part_2_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';
Query ID = hadoop_20210811180052_97aef317-c96a-487d-94b0-d54950fd9d8f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628701152532_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      7          7        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 82.44 s
--------------------------------------------------------------------------------
OK
total_revenue
1211538.4300000907
Time taken: 83.094 seconds, Fetched: 1 row(s)
```

Time Taken to execute the above query is 83.094 sec.

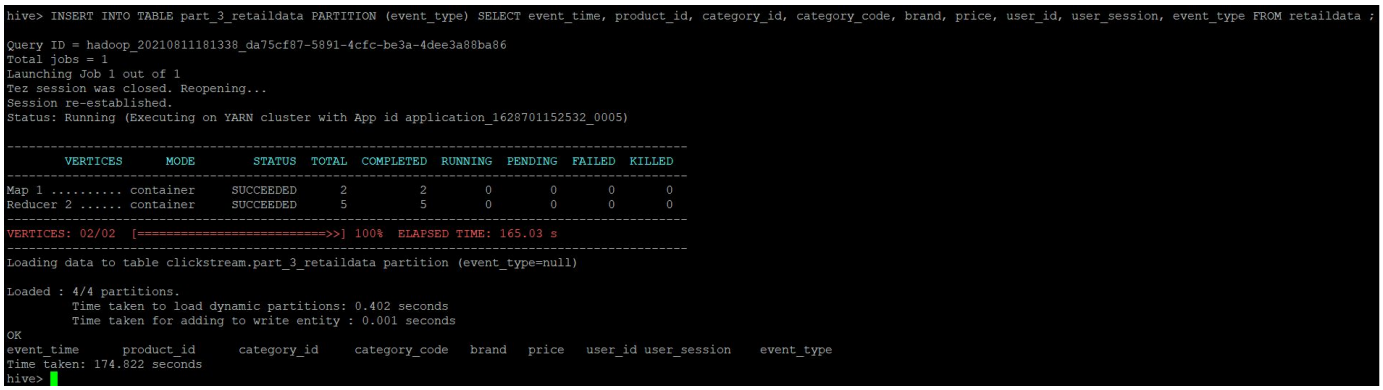
c. Creating Partition Table 3 - part_3_retaildata

We are going to partition on "event_type" and clustered by "user_id".
Command: CREATE TABLE IF NOT EXISTS part_3_retaildata (event_time timestamp, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) PARTITIONED BY (event_type string) CLUSTERED BY (user_id) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE ;

```
hive> CREATE TABLE IF NOT EXISTS part_3_retaildata (event_time timestamp, produc
t_id string, category_id string, category_code string, brand string, price float
, user_id bigint, user_session string) PARTITIONED BY (event_type string) CLUSTE
RED BY (user_id) INTO 10 BUCKETS ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2
.OpenCSVSerde' STORED AS TEXTFILE ;
OK
Time taken: 0.057 seconds
hive>
```

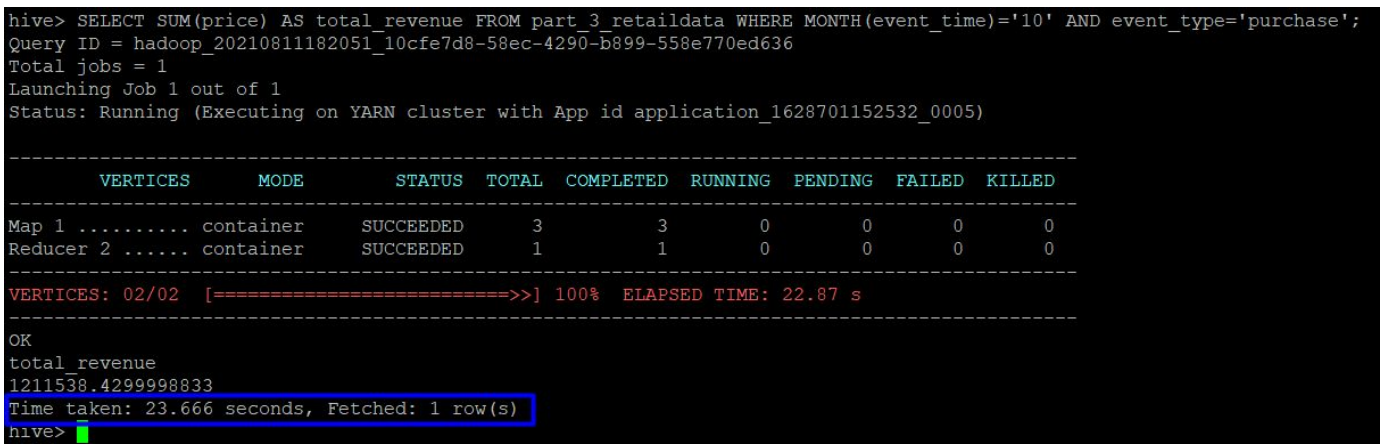
Now Inserting the data into the new optimized table (part_3_retaildata) from base table (retaildata)

Command: INSERT INTO TABLE part_3_retaildata PARTITION (event_type) SELECT event_time, product_id, category_id, category_code, brand, price, user_id, user_session, event_type FROM retaildata ;



Now running the same query on the optimized table 3 and note the time.

Command: SELECT SUM(price) AS total_revenue FROM part_3_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';



Time Taken to execute the above query is 23.666 sec.

Optimized Table	Excution Time
part_1_retaildata	25.211 sec
part_2_retaildata	83.094 sec
part_3_retaildata	23.666 sec

From the above table, we can see that the execution time (23.666 sec) of the optimized table "**part_3_retaildata**" is less as compared to others.

"part_3_retaildata" table is Partitioning on "**event_type**" and clustering by "**user_id**" so we are going to use that table for all the following analysis.

8. Answering The Questions

Q1. Find the total revenue generated due to purchases made in October.

Query: SELECT SUM(price) AS total_revenue FROM part_3_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';

```
hive> SELECT SUM(price) AS total_revenue FROM part_3_retaildata WHERE MONTH(event_time)='10' AND event_type='purchase';
Query ID = hadoop_20210811182051_10cfe7d8-58ec-4290-b899-558e770ed636
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628701152532_0005)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 22.87 s
--------------------------------------------------------------------------------
OK
total_revenue
1211538.4299998833
Time taken: 23.666 seconds, Fetched: 1 row(s)
```

Insights:

- Time Taken to execute the above query is 23.66 sec.
- The total revenue generated due to purchases made in October is 1,211,538.43 /-

Q2. Write a query to yield the total sum of purchases per month in a single output.

Query: SELECT MONTH(event_time) AS month, COUNT(event_type) AS sum_purchases FROM part_3_retaildata WHERE event_type = 'purchase' GROUP BY MONTH(event_time);

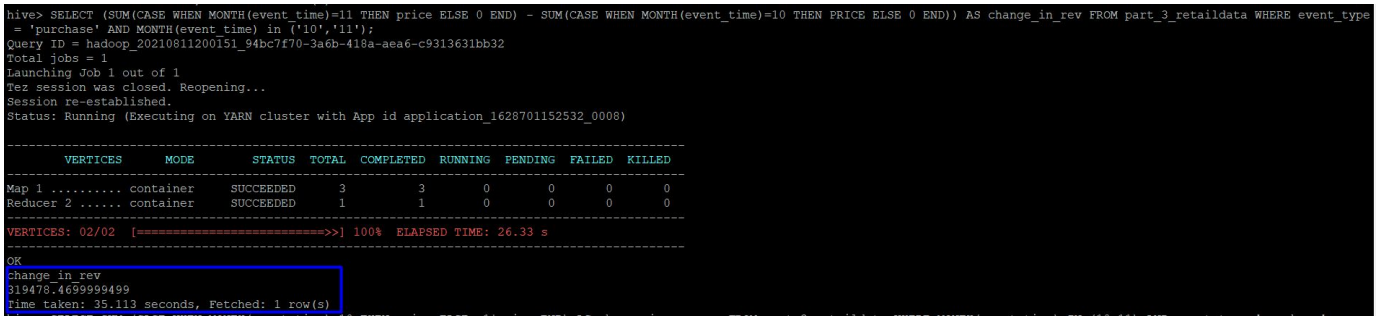
```
hive> SELECT MONTH(event_time) AS month, COUNT(event_type) AS sum_purchases FROM part_3_retaildata WHERE event_type = 'purchase' GROUP BY MONTH(event_time);
Query ID = hadoop_20210811193120_b62b5ccc-7fa9-4fe5-88aa-3e57c4178cad
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628701152532_0007)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 23.11 s
--------------------------------------------------------------------------------
OK
month   sum_purchases
10      245624
11      322417
Time taken: 23.826 seconds, Fetched: 2 row(s)
```

Insights:

- Time Taken to execute the above query is 23.826 sec.
- The Sum of purchases in November month is 322417 which is higher than October month purchases 245624. We can conclude that the number of purchases increased in November month.

Q3. Write a query to find the change in revenue generated due to purchases from October to November.

Query: SELECT (SUM(CASE WHEN MONTH(event_time)=11 THEN price ELSE 0 END) - SUM(CASE WHEN MONTH(event_time)=10 THEN PRICE ELSE 0 END)) AS change_in_rev FROM part_3_retaildata WHERE event_type = 'purchase' AND MONTH(event_time) IN ('10','11');

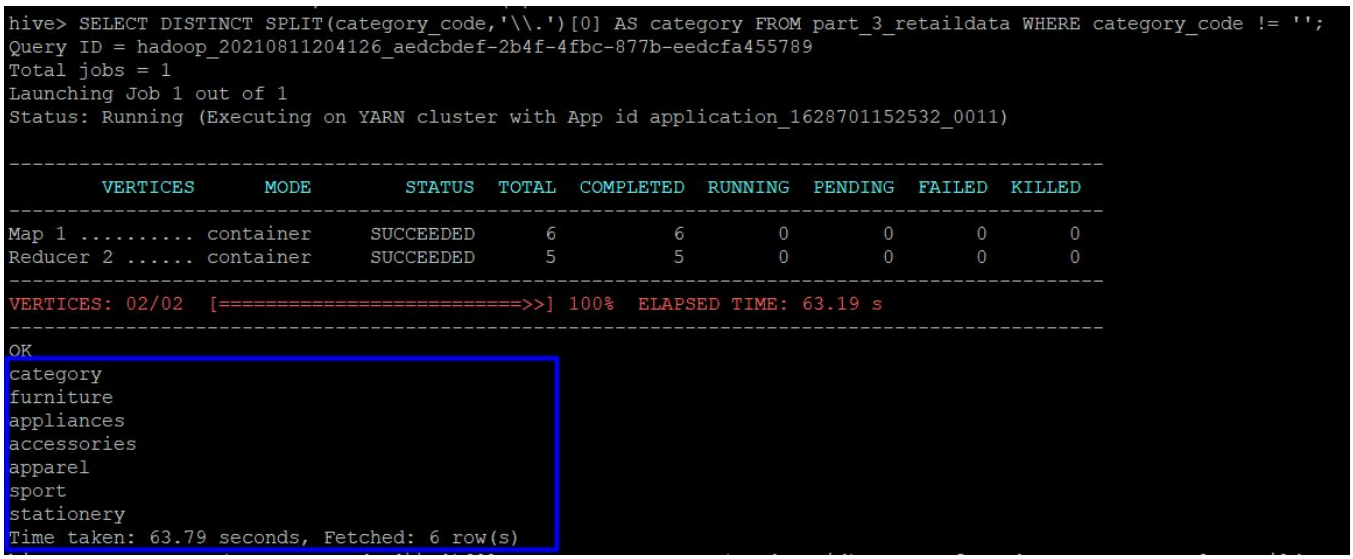


Insights:

- Time Taken to execute the above query is 35.113 sec.
- The change in revenue generated from October to November is 319478.47/-

Q4. Find distinct categories of products. Categories with null category code can be ignored.

Query: SELECT DISTINCT SPLIT(category_code,'\\.')[0] AS category FROM part_3_retaildata WHERE category_code != '';

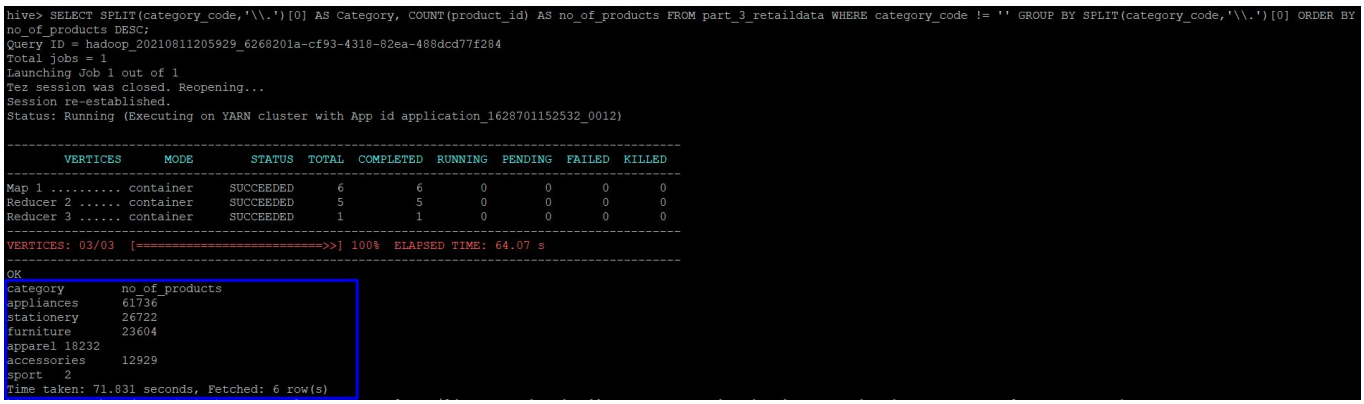


Insights:

- Time Taken to execute the above query is 63.79 sec.
- We have found a total 6 number of distinct product categories: furniture, appliances, accessories, apparel, sport, stationary

Q5. Find the total number of products available under each category.

Query: SELECT SPLIT(category_code,'\\.')[0] AS Category, COUNT(product_id) AS no_of_products FROM part_3_retaildata WHERE category_code != '' GROUP BY SPLIT(category_code,'\\.')[0] ORDER BY no_of_products DESC;



Insights:

- Time Taken to execute the above query is 71.831 sec.
- “appliances” has the highest number of products,
- “Sport” category has the least number of products,
- The number of products in the range of 23000 to 27000 is available in the "furniture" & "stationery" category

Q6. Which brand had the maximum sales in October and November combined?

Query: SELECT brand, SUM(price) AS max_sales FROM part_3_retaildata WHERE brand <>'' AND event_type='purchase' GROUP BY brand ORDER BY max_sales DESC LIMIT 1 ;

Insights:

- Time Taken to execute the above query is 22.446 sec.
- Brand “runail” has the maximum sales i.e. 148297.94 for both months combined
- Brand "runail" is quite popular among users.

Q7. Which brands increased their sales from October to November?

Query: WITH monthly_sales AS (SELECT brand, SUM(CASE WHEN date_format(event_time, 'MM')=10 THEN price ELSE 0 END) AS Oct_sales, SUM(CASE WHEN date_format(event_time, 'MM')=11 THEN price ELSE 0 END) AS Nov_sales FROM part_3_retaildata WHERE event_type='purchase' AND date_format(event_time, 'MM') IN ('10', '11') GROUP BY brand) SELECT brand, Oct_sales, Nov_sales, Nov_sales-Oct_sales AS Sales_diff FROM monthly_sales WHERE (Nov_sales - Oct_sales)>0 ORDER BY Sales_diff;

```
hive> WITH monthly_sales AS ( SELECT brand, SUM(CASE WHEN date_format(event_time, 'MM')=10 THEN price ELSE 0 END) AS Oct_sales, SUM(CASE WHEN date_format(event_time, 'MM')=11 THEN price ELS
E 0 END) AS Nov_sales FROM part_3_retaildata WHERE event_type='purchase' AND date_format(event_time, 'MM') IN ('10', '11') GROUP BY brand ) SELECT brand, Oct_sales, Nov_sales, Nov_sales-Oct
_sales AS Sales_diff FROM monthly_sales WHERE (Nov_sales - Oct_sales)>0 ORDER BY Sales_diff;
Query ID = hadoop_20210812175117_af6c3bec-9cf9-4b13-927c-6e44f943861b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1628783284537_0005)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 39.35 s
--------------------------------------------------------------------------------
OK
brand   oct_sales       nov_sales       sales_diff
ovale   2.54    3.1     0.56
cosima  20.229999999999997      20.929999999999996      0.6999999999999993
grace   100.91999999999997      102.60999999999999      1.690000000000012
helloganic      0.0     3.1     3.1
skinity 8.88    12.440000000000001      3.5600000000000005
bodyton 1376.3400000000004      1380.6400000000003      4.2999999999999545
moyou   5.71    10.280000000000001      4.570000000000001
neoleor 43.41   51.7    8.290000000000006
soleo   204.19999999999965      212.5299999999998       8.330000000000155
jaguar  1102.1100000000004      1110.6500000000003      8.539999999999964
tertio  236.16000000000003      245.79999999999998      9.639999999999958
fly     17.14   27.17   10.030000000000001
rasyan  18.799999999999997      28.939999999999998      10.14
deoproce        316.84000000000003      329.16999999999996      12.329999999999927
barbie  0.0     12.39   12.39
supertan        50.37   66.51   16.140000000000008
treaclemoon     163.37  181.49  18.120000000000005
kamill  63.00999999999999       81.49000000000001       18.480000000000018
juno    0.0     21.08   21.08
veraclara       50.11   71.21000000000001       21.10000000000001
glysolid        69.72999999999999       91.58999999999999       21.86
godefroy        401.22  425.12  23.899999999999977
binacil 0.0     24.259999999999998      24.259999999999998
blixz   38.95   63.39999999999999       24.44999999999999
profepil        93.36   118.02000000000001      24.66000000000001
estelare        444.8100000000003       471.87000000000006      27.059999999999775
orly    902.38  931.09  28.710000000000036
biore   60.650000000000006      90.31   29.659999999999997
beautyblender   78.74000000000001       109.41  30.669999999999987
```

```
bpw.style       11572.150000000194      14837.440000000333      3265.290000000139
staleks 8519.730000000018       11875.610000000019      3355.880000000001
freedecor       3421.7799999999975      7671.799999999962       4250.019999999964
runail  71539.27999999985       76758.65999999993       5219.380000000077
polarus 6013.720000000001       11371.930000000006      5358.210000000005
cosmoprofi      8322.809999999989       14536.99000000007       6214.180000000082
jessnail        26287.84000000018       33345.230000000156      7057.389999999978
strong  29196.630000000012      38671.27000000001       9474.64
ingarden        23161.39000000004       33566.21000000022       10404.820000000178
lianail 5892.839999999985       16394.239999999998      10501.400000000012
uno     35302.030000000086      51039.75000000007       15737.719999999987
grattol 35445.540000000045      71472.7100000033        36027.17000000325
        474679.06000001146      619509.2400000115       144830.18000000005
Time taken: 44.135 seconds, Fetched: 161 row(s)
hive>
```

Insights:

- Time Taken to execute the above query is 44.135 sec.
- Total 161 brands increased their sales from October to November

Q8. Your company wants to reward the top 10 users of its website with a Golden Customer plan. Write a query to generate a list of top 10 users who spend the most.

Query: SELECT user_id, SUM(price) AS total_spend FROM part_3_retaildata WHERE event_type = 'purchase' GROUP BY user_id ORDER BY total_spend DESC LIMIT 10;

```
hive> SELECT user_id, SUM(price) AS total_spend FROM part_3_retaildata WHERE event_type = 'purchase' GROUP BY user_id ORDER BY total_spend DESC LIMIT 10;
Query ID = hadoop_20210812182241_2b56e1ba-fa9e-4075-ad56-24d229f8df02
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_1628783284537_0007)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 25.35 s
--------------------------------------------------------------------------------
OK
user_id total_spend
557790271       2715.8699999999976
150318419       1645.9700000000003
562167663       1352.8500000000001
531900924       1329.449999999999
557850743       1295.4800000000002
522130011       1185.39
561592095       1109.7000000000007
431950134       1097.589999999999
566576008       1056.3600000000004
521347209       1040.91
Time taken: 33.589 seconds, Fetched: 10 row(s)
hive>
```

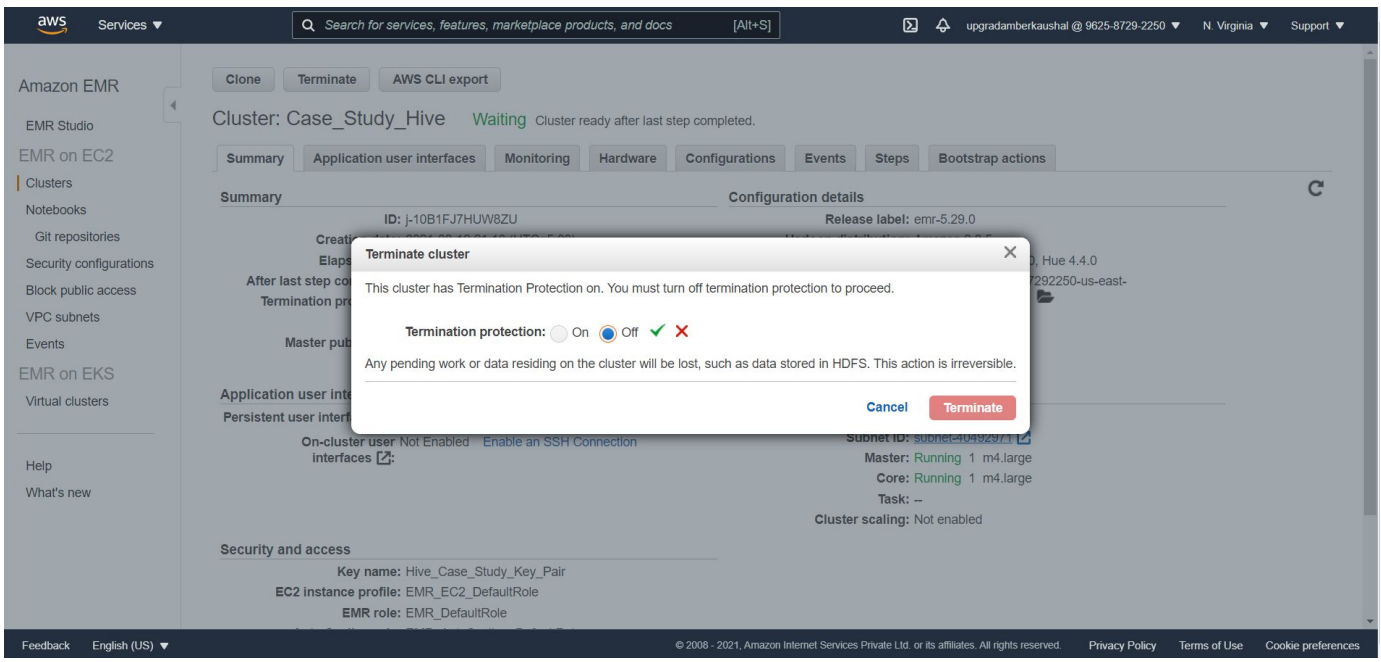
Insights:

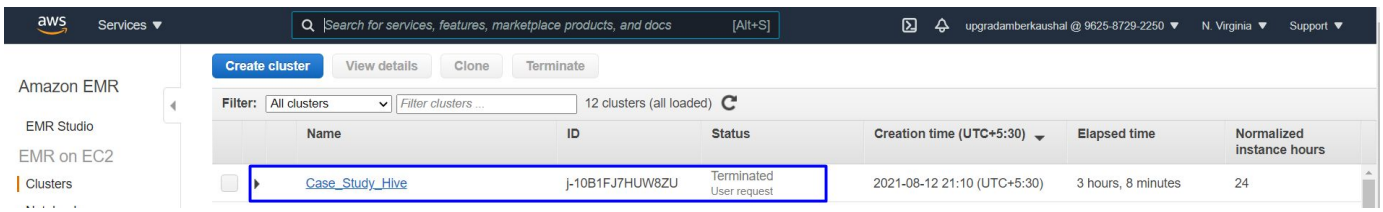
- Time Taken to execute the above query is 33.589 sec.
- We have got the list of the top ten users who spends the most. We can reward them with Golden Customer plan. It could attract new users.

9. Cleaning up

- We have done the analysis, Now deleting the database & terminating the cluster.

```
hive> drop database clickstream;
OK
Time taken: 0.155 seconds
hive> show databases;
OK
database_name
default
Time taken: 0.008 seconds, Fetched: 1 row(s)
```





Cluster Terminated
