# Kalyan Big Data Projects – Project 10
# How To Stream REGEX Data Into Hive Using Apache Flume

**Pre-Requisites of Flume + Hive Project:**

hadoop-2.6.0 , flume-1.6.0 , hive-1.2.1 , java-1.7

**NOTE:** Make sure that install all the above components

**Flume + Hive Project Download Links:**

`hadoop-2.6.0.tar.gz` ==> link
(https://archive.apache.org/dist/hadoop/core/hadoop-2.6.0/hadoop-2.6.0.tar.gz)

`apache-flume-1.6.0-bin.tar.gz` ==> link
(https://archive.apache.org/dist/flume/1.6.0/apache-flume-1.6.0-bin.tar.gz)

`apache-hive-1.2.1-bin.tar.gz` ==> link
(http://mirror.fibergrid.in/apache/hive/hive-1.2.1/apache-hive-1.2.1-bin.tar.gz)

`kalyan-bigdata-examples.jar` ==> link
(https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/kalyan/kalyan-bigdata-examples.jar)

`kalyan-flume-hive-sink-1.6.0.jar` ==> link
(https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/kalyan/kalyan-flume-hive-sink-1.6.0.jar)

`kalyan-regex-hive-agent.conf` ==> link
(https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/flume/project10-hive-regex/kalyan-regex-hive-agent.conf)

---

**Learnings of this Project:**

---

- We will learn Flume Configurations and Commands
- Flume Agent
  1. Source (Exec Source)
  2. Channel (Memory Channel)
  3. Sink (Hive Sink)
- Major project in Real Time `Product Log Analysis`
  1. We are extracting the data from server logs
  2. This data will be useful to do analysis on product views
  3. Complex Data is the output format then REGEX is best solution
- We can use Hive to analyze this data

---

1. create "**kalyan-regex-hive-agent.conf**" file with below content

```
agent.sources = EXEC
agent.sinks = HIVE
agent.channels = MemChannel

agent.sources.EXEC.type = exec
agent.sources.EXEC.command = tail -F /tmp/users.csv
agent.sources.EXEC.channels = MemChannel

agent.sinks.HIVE.type = hive
agent.sinks.HIVE.hive.metastore = thrift://localhost:9083
agent.sinks.HIVE.hive.database = kalyan
agent.sinks.HIVE.hive.table = users3
agent.sinks.HIVE.serializer = REGEX
agent.sinks.HIVE.serializer.regex = ^([^,]*),([^,]*),([^,]*),([^,]*),([^,]*),([^,]*),([^,]*),([^,]*)$
agent.sinks.HIVE.channel = MemChannel

agent.channels.MemChannel.type = memory
agent.channels.MemChannel.capacity = 1000
agent.channels.MemChannel.transactionCapacity = 100
```

2. Copy "**kalyan-regex-hive-agent.conf**" file into "**$FUME_HOME/conf**" folder

3. Copy "**kalyan-bigdata-examples.jar and kalyan-flume-hive-sink-1.6.0.jar**" files into "**$FLUME_HOME/lib**" folder

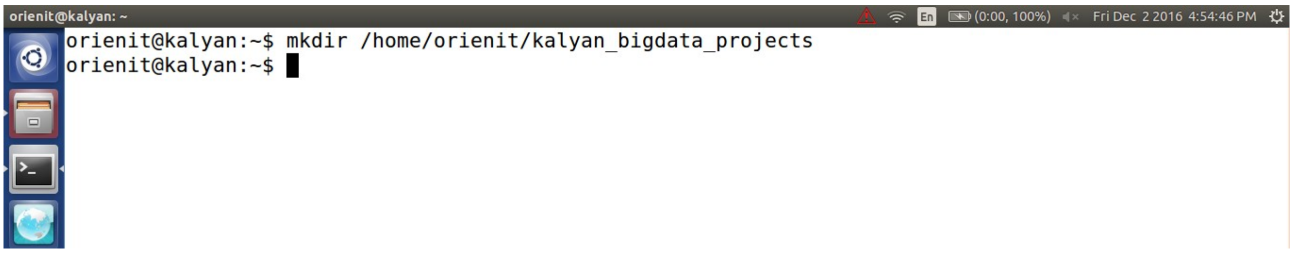
4. Generate Large Amount of Sample CSV data follow this article.

(http://kalyanbigdatatraining.blogspot.com/2016/12/how-to-generate-large-amount-of-sample.html)

5. Follow below steps...

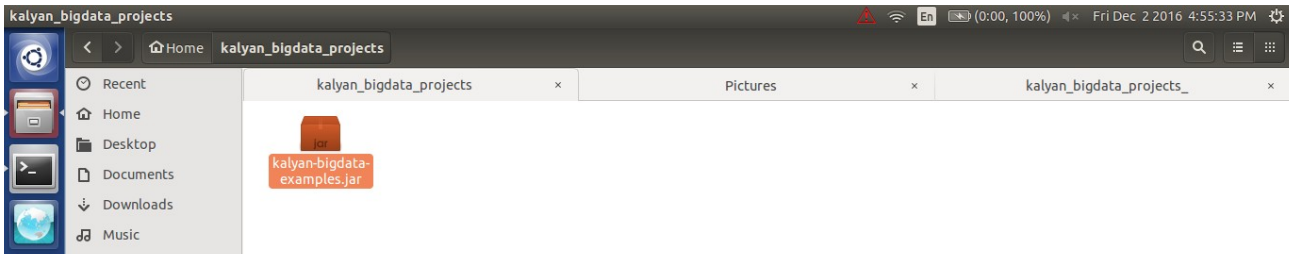
i) Create '**kalyan_bigdata_projects**' folder in **user home** (i.e **/home/orienit**)

**Command:** *mkdir /home/orienit/kalyan_bigdata_projects*



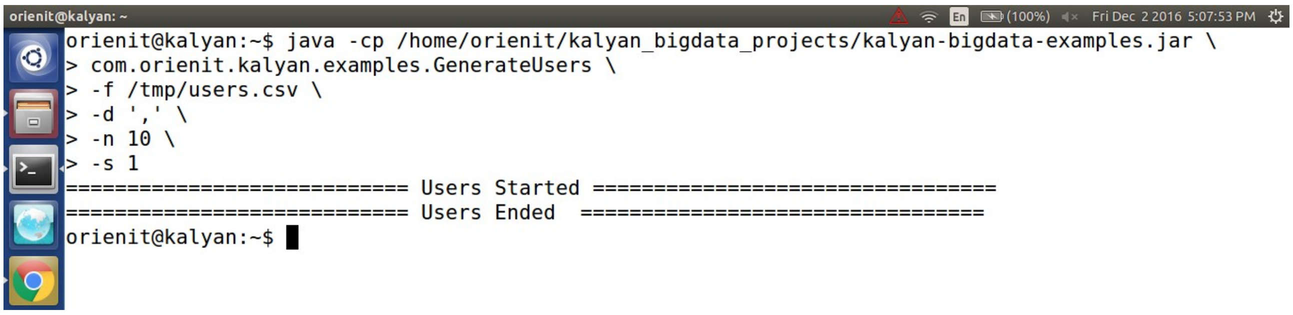
```
orienit@kalyan:~$ mkdir /home/orienit/kalyan_bigdata_projects
orienit@kalyan:~$ 
```

ii) Copy '**kalyan-bigdata-examples.jar**' jar file into '**/home/orienit/kalyan_bigdata_projects**' folder



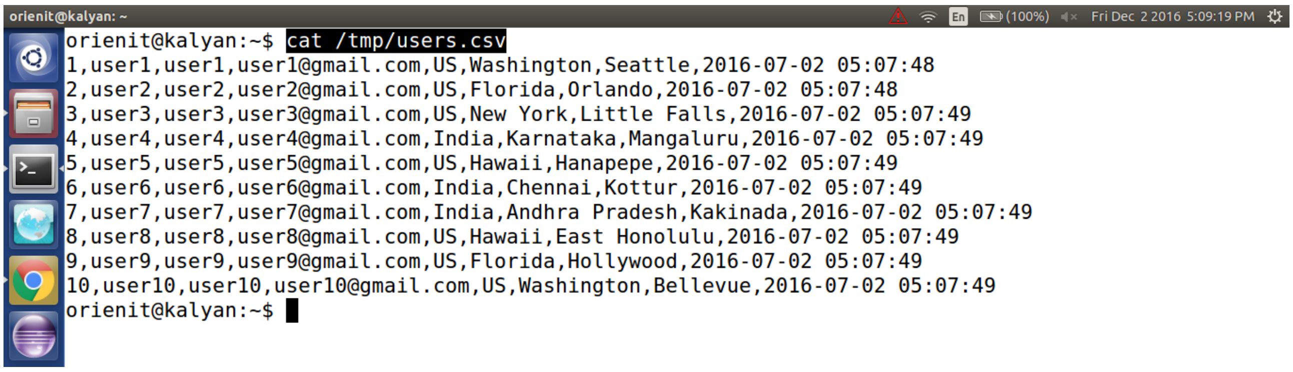
iii) Execute Below Command to Generate Sample CSV data with 100 lines. Increase this number to get more data ...

```
java -cp /home/orienit/kalyan_bigdata_projects/kalyan-bigdata-examples.jar \
com.orienit.kalyan.examples.GenerateUsers \
-f /tmp/users.csv \
-d ',' \
-n 100 \
-s 1
```


```
orienit@kalyan:~$ java -cp /home/orienit/kalyan_bigdata_projects/kalyan-bigdata-examples.jar \
> com.orienit.kalyan.examples.GenerateUsers \
> -f /tmp/users.csv \
> -d ',' \
> -n 10 \
> -s 1
============================ Users Started =================================
============================ Users Ended  =================================
orienit@kalyan:~$
```

6. Verify the Sample CSV data in Console, using below command

```
cat /tmp/users.csv
```


```
orienit@kalyan:~$ cat /tmp/users.csv
1,user1,user1,user1@gmail.com,US,Washington,Seattle,2016-07-02 05:07:48
2,user2,user2,user2@gmail.com,US,Florida,Orlando,2016-07-02 05:07:48
3,user3,user3,user3@gmail.com,US,New York,Little Falls,2016-07-02 05:07:49
4,user4,user4,user4@gmail.com,India,Karnataka,Mangaluru,2016-07-02 05:07:49
5,user5,user5,user5@gmail.com,US,Hawaii,Hanapepe,2016-07-02 05:07:49
6,user6,user6,user6@gmail.com,India,Chennai,Kottur,2016-07-02 05:07:49
7,user7,user7,user7@gmail.com,India,Andhra Pradesh,Kakinada,2016-07-02 05:07:49
8,user8,user8,user8@gmail.com,US,Hawaii,East Honolulu,2016-07-02 05:07:49
9,user9,user9,user9@gmail.com,US,Florida,Hollywood,2016-07-02 05:07:49
10,user10,user10,user10@gmail.com,US,Washington,Bellevue,2016-07-02 05:07:49
orienit@kalyan:~$
```

7. To work with **Flume + Hive Integration**, Follow the below steps

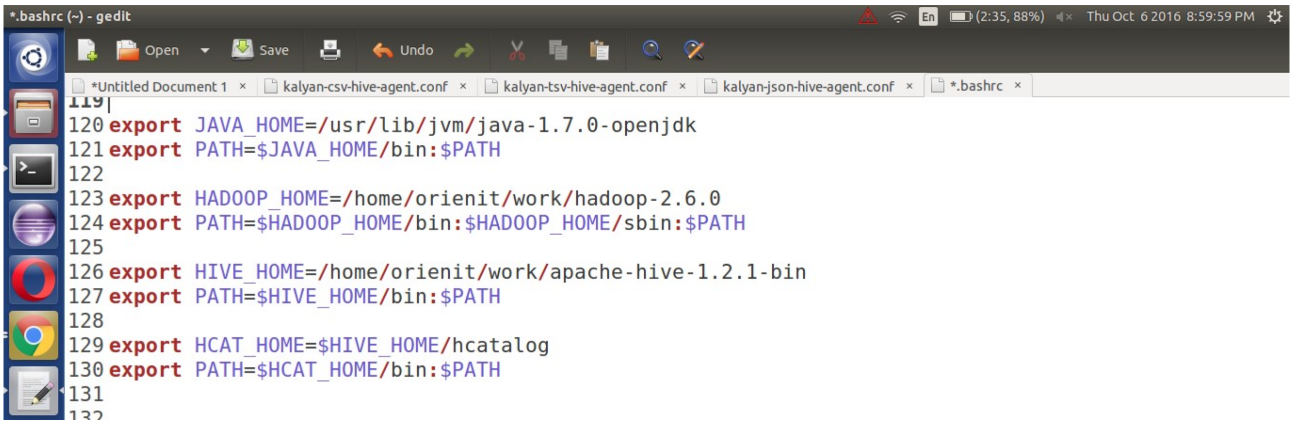
Follow this aritcle to work with below procedure.

Refer: http://kalyanbigdatatraining.blogspot.in/2016/10/how-to-work-with-acid-functionality-in.html

i) update '**~/.bashrc**' file with below changes

export HIVE_HOME=/home/orienit/work/apache-hive-1.2.1-bin
export PATH=$HIVE_HOME/bin:$PATH

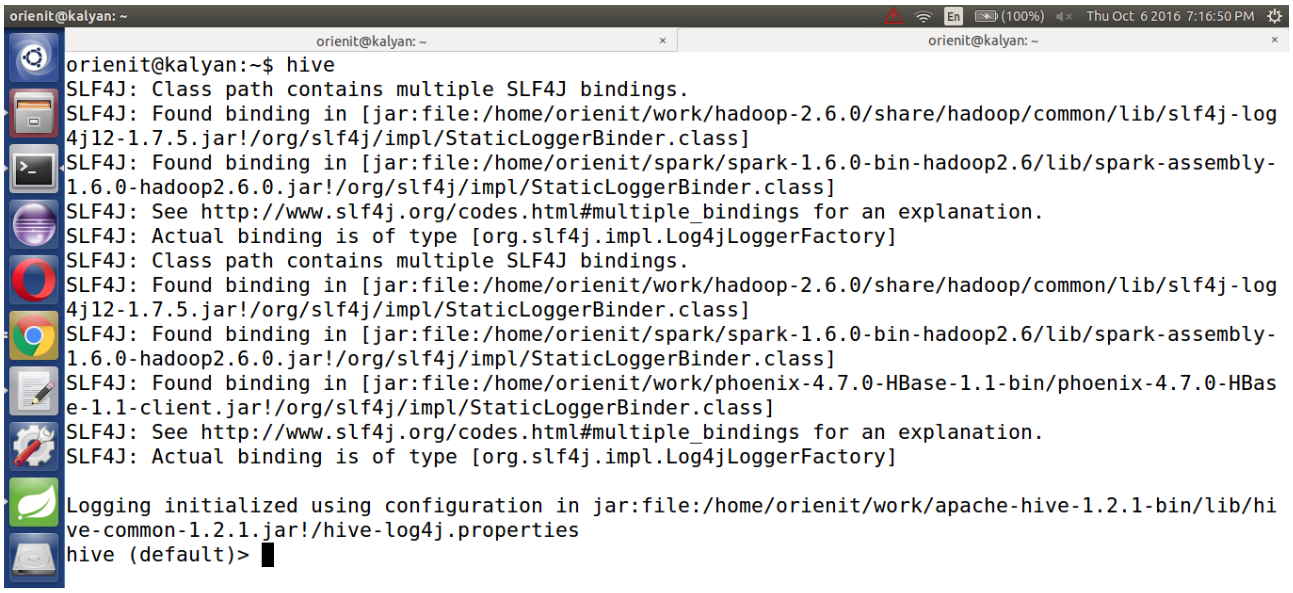
export HCAT_HOME=$HIVE_HOME/hcatalog
export PATH=$HCAT_HOME/bin:$PATH



```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk
export PATH=$JAVA_HOME/bin:$PATH

export HADOOP_HOME=/home/orienit/work/hadoop-2.6.0
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH

export HIVE_HOME=/home/orienit/work/apache-hive-1.2.1-bin
export PATH=$HIVE_HOME/bin:$PATH

export HCAT_HOME=$HIVE_HOME/hcatalog
export PATH=$HCAT_HOME/bin:$PATH
```

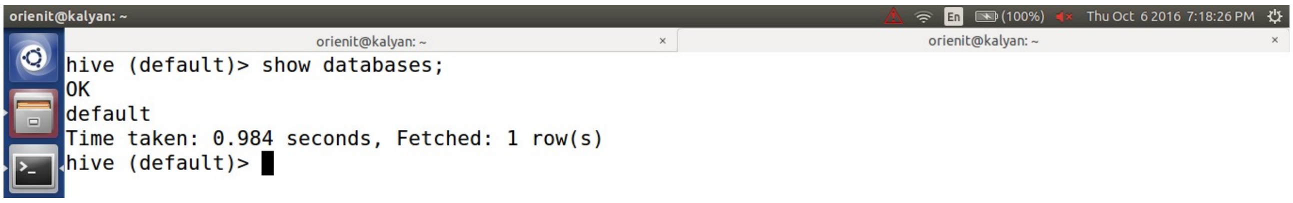
ii. reopen the Terminal

iii. start the hive using '**hive**' command.



```
orienit@kalyan:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/spark/spark-1.6.0-bin-hadoop2.6/lib/spark-assembly-
1.6.0-hadoop2.6.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/spark/spark-1.6.0-bin-hadoop2.6/lib/spark-assembly-
1.6.0-hadoop2.6.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/work/phoenix-4.7.0-HBase-1.1-bin/phoenix-4.7.0-HBas
e-1.1-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/home/orienit/work/apache-hive-1.2.1-bin/lib/hi
ve-common-1.2.1.jar!/hive-log4j.properties
hive (default)>
```

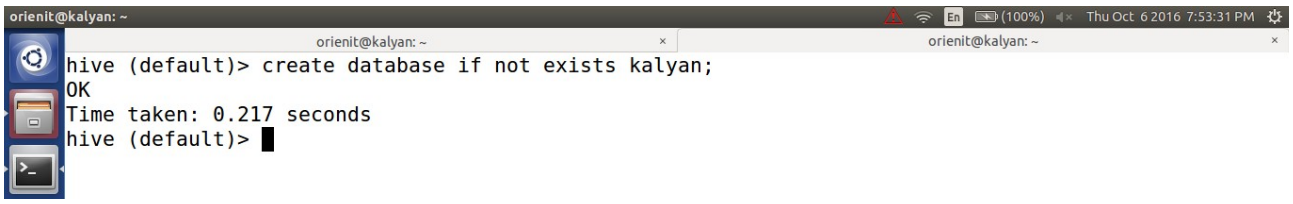
iv. list out all the databases in hive using '**show databases;**' command



```
hive (default)> show databases;
OK
default
Time taken: 0.984 seconds, Fetched: 1 row(s)
hive (default)>
```

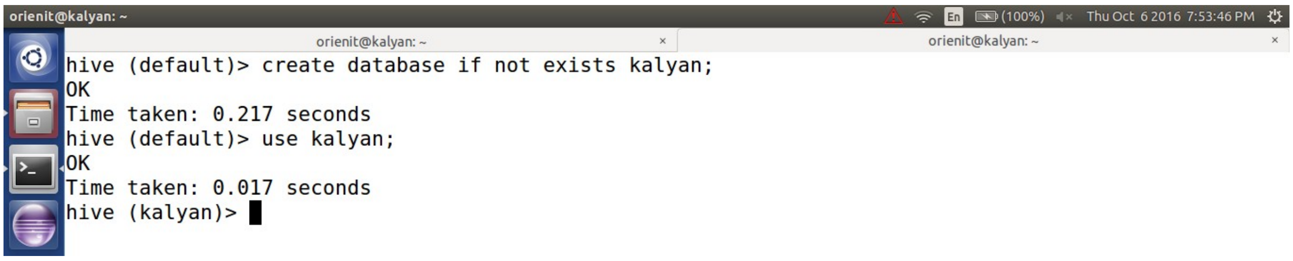
v. create a new database (**kalyan**) in hive using below command.

create database if not exists kalyan;



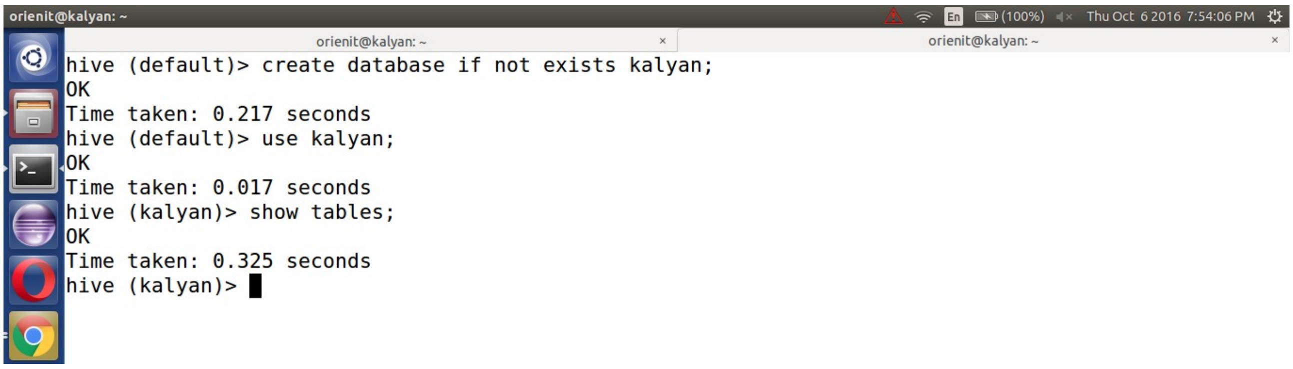
```
hive (default)> create database if not exists kalyan;
OK
Time taken: 0.217 seconds
hive (default)>
```

vi. use kalyan database using '**use kalyan;**' command



```
hive (default)> create database if not exists kalyan;
OK
Time taken: 0.217 seconds
hive (default)> use kalyan;
OK
Time taken: 0.017 seconds
hive (kalyan)>
```

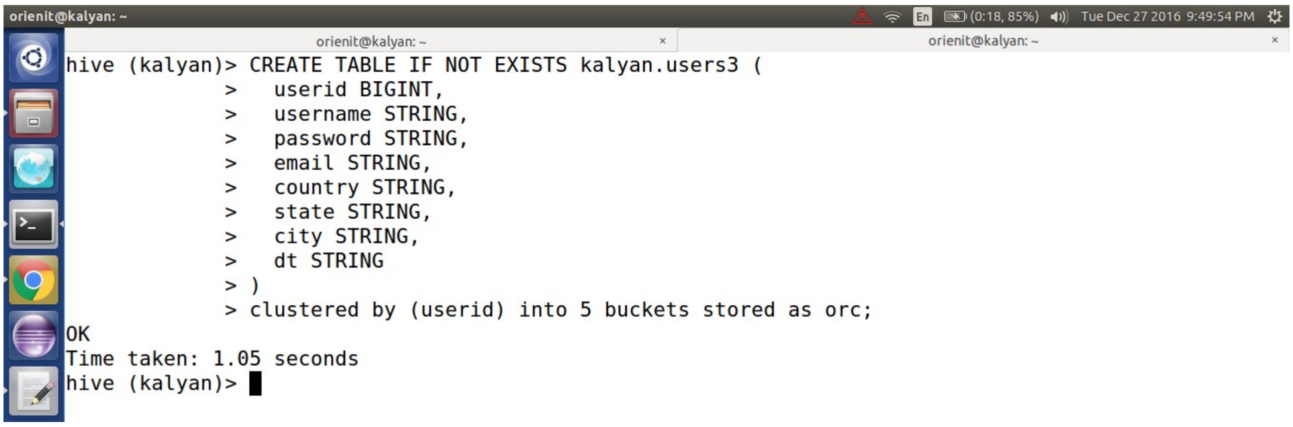
vii. list out all the tables in kalyan database using '**show tables;**' command.



```
hive (default)> create database if not exists kalyan;
OK
Time taken: 0.217 seconds
hive (default)> use kalyan;
OK
Time taken: 0.017 seconds
hive (kalyan)> show tables;
OK
Time taken: 0.325 seconds
hive (kalyan)>
```

viii. create '**users3**' table in kalyan database using below command.

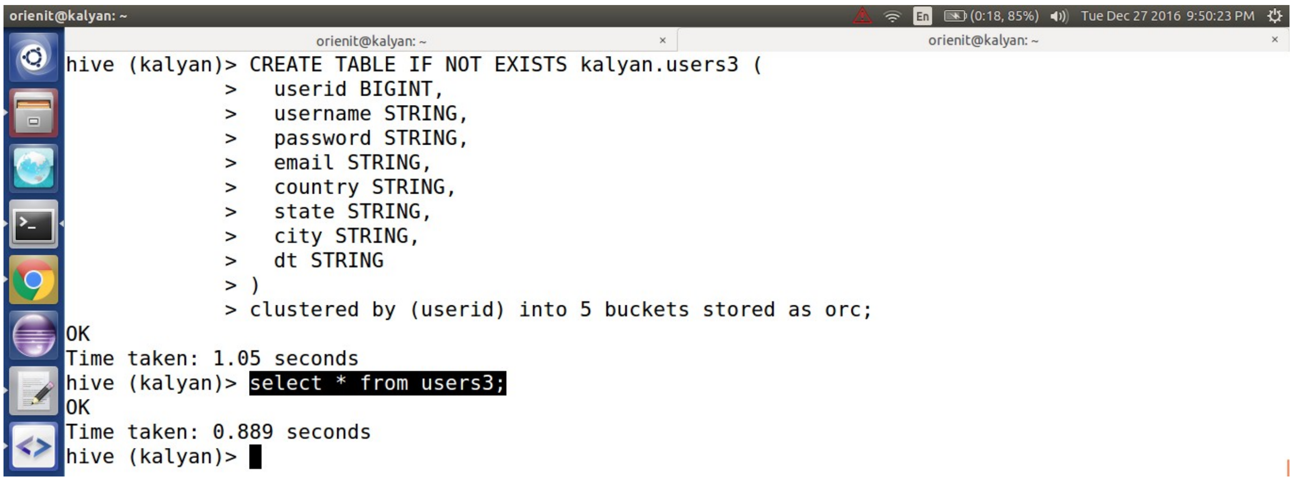
```
CREATE TABLE IF NOT EXISTS kalyan.users3 (
  userid BIGINT,
  username STRING,
  password STRING,
  email STRING,
  country STRING,
  state STRING,
  city STRING,
  dt STRING
)
clustered by (userid) into 5 buckets stored as orc;
```



```
hive (kalyan)> CREATE TABLE IF NOT EXISTS kalyan.users3 (
             >   userid BIGINT,
             >   username STRING,
             >   password STRING,
             >   email STRING,
             >   country STRING,
             >   state STRING,
             >   city STRING,
             >   dt STRING
             > )
             > clustered by (userid) into 5 buckets stored as orc;
OK
Time taken: 1.05 seconds
hive (kalyan)>
```

ix. Display the data from '**users3**' table using below command

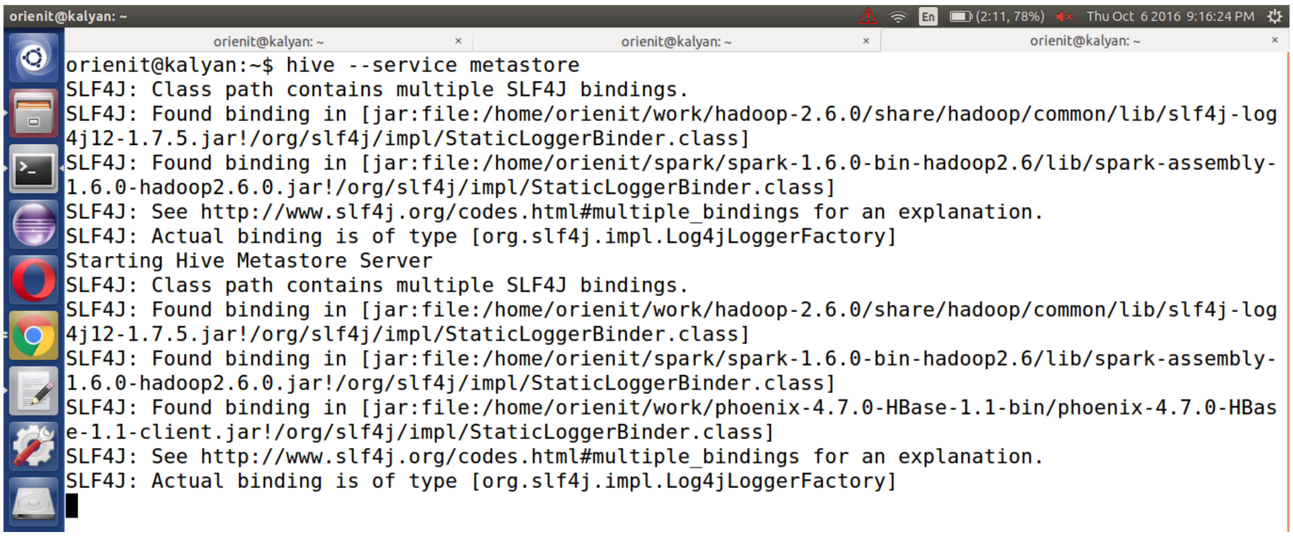
select * from users3;



```
hive (kalyan)> CREATE TABLE IF NOT EXISTS kalyan.users3 (
             >   userid BIGINT,
             >   username STRING,
             >   password STRING,
             >   email STRING,
             >   country STRING,
             >   state STRING,
             >   city STRING,
             >   dt STRING
             > )
             > clustered by (userid) into 5 buckets stored as orc;
OK
Time taken: 1.05 seconds
hive (kalyan)> select * from users3;
OK
Time taken: 0.889 seconds
hive (kalyan)>
```

x. start the hive in external metastore db mode using below command

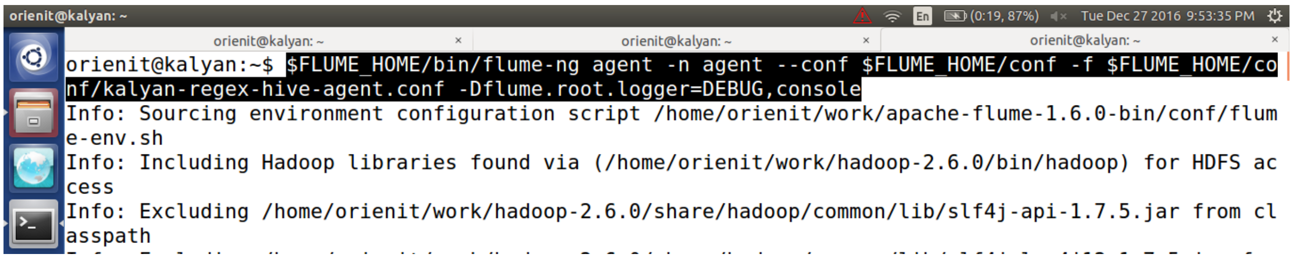
hive --service metastore



```
orienit@kalyan:~$ hive --service metastore
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/spark/spark-1.6.0-bin-hadoop2.6/lib/spark-assembly-
1.6.0-hadoop2.6.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Starting Hive Metastore Server
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/spark/spark-1.6.0-bin-hadoop2.6/lib/spark-assembly-
1.6.0-hadoop2.6.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/work/phoenix-4.7.0-HBase-1.1-bin/phoenix-4.7.0-HBas
e-1.1-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

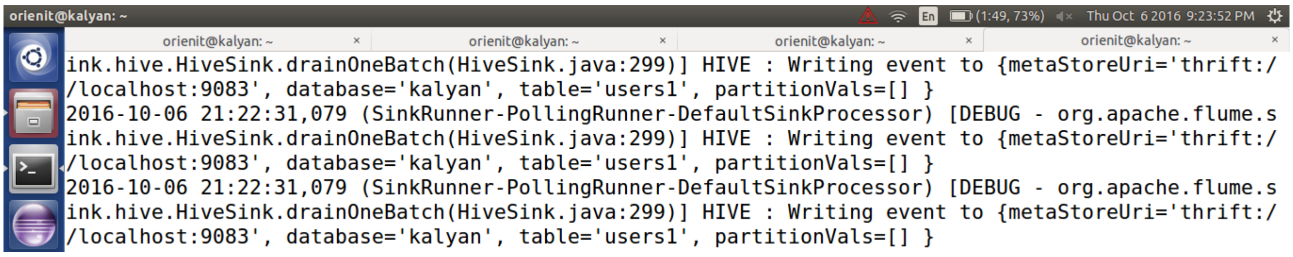
8. Execute the below command to `**Extract data from REGEX data into Hive using Flume**`

$FLUME_HOME/bin/flume-ng agent -n agent --conf $FLUME_HOME/conf -f $FLUME_HOME/conf/kalyan-regex-hive-agent.conf -Dflume.root.logger=DEBUG,console



```
orienit@kalyan:~$ $FLUME_HOME/bin/flume-ng agent -n agent --conf $FLUME_HOME/conf -f $FLUME_HOME/co
nf/kalyan-regex-hive-agent.conf -Dflume.root.logger=DEBUG,console
Info: Sourcing environment configuration script /home/orienit/work/apache-flume-1.6.0-bin/conf/flum
e-env.sh
Info: Including Hadoop libraries found via (/home/orienit/work/hadoop-2.6.0/bin/hadoop) for HDFS ac
cess
Info: Excluding /home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-api-1.7.5.jar from cl
asspath
```

9. Verify the data in console

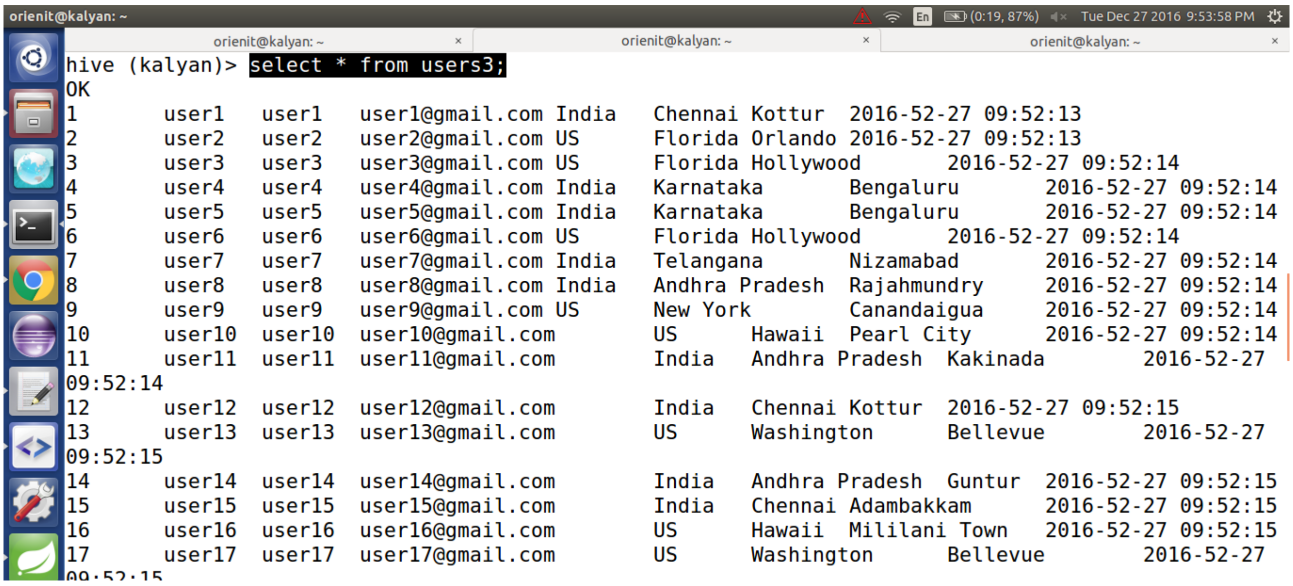


```
ink.hive.HiveSink.drainOneBatch(HiveSink.java:299)] HIVE : Writing event to {metaStoreUri='thrift:/
/localhost:9083', database='kalyan', table='users1', partitionVals=[] }
2016-10-06 21:22:31,079 (SinkRunner-PollingRunner-DefaultSinkProcessor) [DEBUG - org.apache.flume.s
ink.hive.HiveSink.drainOneBatch(HiveSink.java:299)] HIVE : Writing event to {metaStoreUri='thrift:/
/localhost:9083', database='kalyan', table='users1', partitionVals=[] }
2016-10-06 21:22:31,079 (SinkRunner-PollingRunner-DefaultSinkProcessor) [DEBUG - org.apache.flume.s
ink.hive.HiveSink.drainOneBatch(HiveSink.java:299)] HIVE : Writing event to {metaStoreUri='thrift:/
/localhost:9083', database='kalyan', table='users1', partitionVals=[] }
```

10. Verify the data in Hive

Execute below command to get the data from hive table '**users3**'

select * from users3;



```
hive (kalyan)> select * from users3;
OK
1       user1   user1   user1@gmail.com India   Chennai Kottur  2016-52-27 09:52:13
2       user2   user2   user2@gmail.com US      Florida Orlando 2016-52-27 09:52:13
3       user3   user3   user3@gmail.com US      Florida Hollywood       2016-52-27 09:52:14
4       user4   user4   user4@gmail.com India   Karnataka       Bengaluru       2016-52-27 09:52:14
5       user5   user5   user5@gmail.com India   Karnataka       Bengaluru       2016-52-27 09:52:14
6       user6   user6   user6@gmail.com US      Florida Hollywood       2016-52-27 09:52:14
7       user7   user7   user7@gmail.com India   Telangana       Nizamabad       2016-52-27 09:52:14
8       user8   user8   user8@gmail.com India   Andhra Pradesh  Rajahmundry     2016-52-27 09:52:14
9       user9   user9   user9@gmail.com US      New York        Canandaigua     2016-52-27 09:52:14
10      user10  user10  user10@gmail.com        US      Hawaii  Pearl City      2016-52-27 09:52:14
11      user11  user11  user11@gmail.com        India   Andhra Pradesh  Kakinada        2016-52-27
09:52:14
12      user12  user12  user12@gmail.com        India   Chennai Kottur  2016-52-27 09:52:15
13      user13  user13  user13@gmail.com        US      Washington      Bellevue        2016-52-27
09:52:15
14      user14  user14  user14@gmail.com        India   Andhra Pradesh  Guntur  2016-52-27 09:52:15
15      user15  user15  user15@gmail.com        India   Chennai Adambakkam      2016-52-27 09:52:15
16      user16  user16  user16@gmail.com        US      Hawaii  Mililani Town   2016-52-27 09:52:15
17      user17  user17  user17@gmail.com        US      Washington      Bellevue        2016-52-27
```