**APACHE FLUME**

A service for streaming logs into Hadoop.

Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming data into the Hadoop Distributed File System (HDFS). It has a simple and flexible architecture based on streaming data flows; and is robust and fault tolerant with tunable reliability mechanisms for failover and recovery.

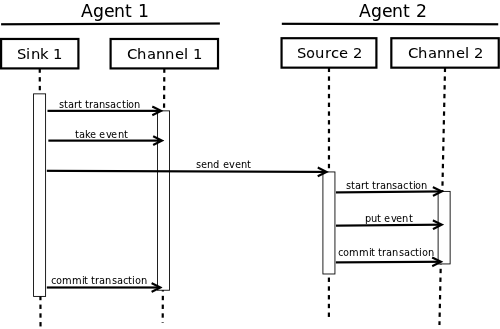
YARN coordinates data ingest from Apache Flume and other services that deliver raw data into an Enterprise Hadoop cluster.

**WHAT FLUME DOES**

Flume lets Hadoop users ingest high-volume streaming data into HDFS for storage. Specifically, Flume allows users to:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Stream data** | Ingest streaming data from multiple sources into Hadoop for storage and analysis |
| **Insulate systems** | Buffer storage platform from transient spikes, when the rate of incoming data exceeds the rate at which data can be written to the destination |
| **Guarantee data delivery** | Flume NG uses channel-based transactions to guarantee reliable message delivery. When a message moves from one agent to another, two transactions are started, one on the agent that delivers the event and the other on the agent that receives the event. This ensures guaranteed delivery semantics |
| **Scale horizontally** | To ingest new data streams and additional volume as needed |

Enterprises use Flume’s powerful streaming capabilities to land data from high-throughput streams in the [Hadoop Distributed File System (HDFS)](http://hortonworks.com/hadoop/hdfs/). Typical sources of these streams are application logs, sensor and machine data, geo-location data and social media. These different types of data can be landed in Hadoop for future analysis using interactive queries in Apache Hive. Or they can feed business dashboards served ongoing data by Apache HBase.



In one specific example, Flume is used to log manufacturing operations. When one run of product comes off the line, it generates a log file about that run. Even if this occurs hundreds or thousands of times per day, the large volume log file data can stream through Flume into a tool for same-day analysis with Apache Storm or months or years of production runs can be stored in HDFS and analyzed by a quality assurance engineer using Apache Hive.

**HOW FLUME WORKS**

Flume’s high-level architecture is built on a streamlined codebase that is easy to use and extend. The project is highly reliable, without the risk of data loss. Flume also supports dynamic reconfiguration without the need for a restart, which reduces downtime for its agents.

The following components make up Apache Flume:

|  |  |
| --- | --- |
| **Component** | **Definition** |
| **Event** | A singular unit of data that is transported by Flume (typically a single log entry) |
| **Source** | The entity through which data enters into Flume. Sources either actively poll for data or passively wait for data to be delivered to them. A variety of sources allow data to be collected, such as log4j logs and syslogs. |
| **Sink** | The entity that delivers the data to the destination. A variety of sinks allow data to be streamed to a range of destinations. One example is the HDFS sink that writes events to HDFS. |
| **Channel** | The conduit between the Source and the Sink. Sources ingest events into the channel and the sinks drain the channel. |
| **Agent** | Any physical Java virtual machine running Flume. It is a collection of sources, sinks and channels. |
| **Client** | The entity that produces and transmits the Event to the Source operating within the Agent. |

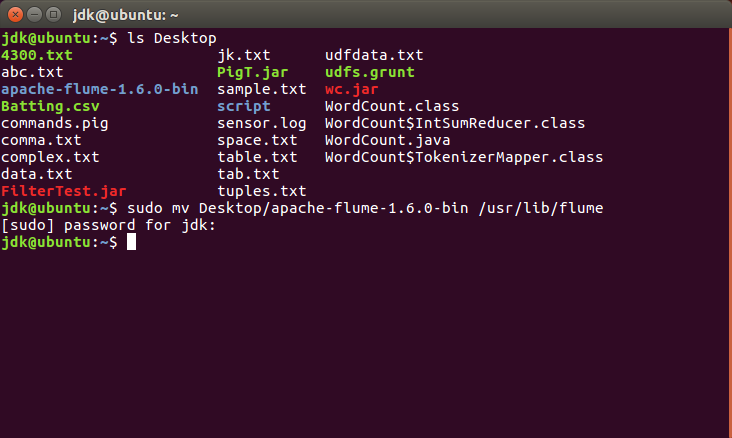
Flume components interact in the following way:

1. A flow in Flume starts from the **Client**.
2. The **Client** transmits the **Event** to a **Source** operating within the **Agent**.
3. The **Source** receiving this **Event** then delivers it to one or more **Channels**.
4. One or more **Sinks** operating within the same **Agent** drains these **Channels**.
5. **Channels** decouple the ingestion rate from drain rate using the familiar producer-consumer model of data exchange.
6. When spikes in client side activity cause data to be generated faster than can be handled by the provisioned destination capacity can handle, the **Channel** size increases. This allows sources to continue normal operation for the duration of the spike.
7. The **Sink** of one **Agent** can be chained to the **Source** of another **Agent**. This chaining enables the creation of complex data flow topologies.

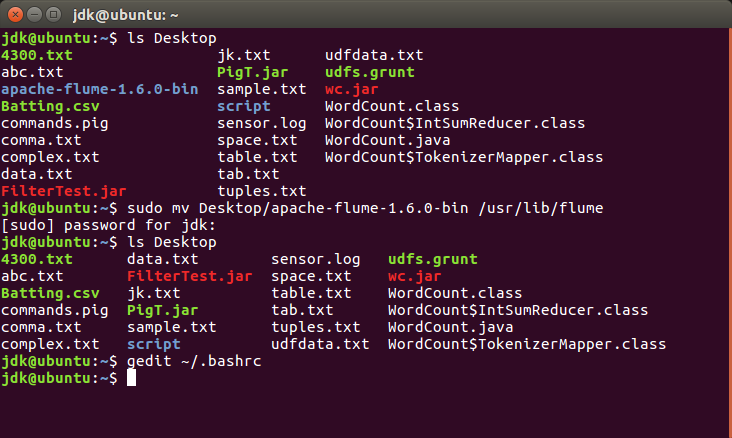
Because Flume’s distributed architecture requires no central coordination point. Each agent runs independently of others with no inherent single point of failure, and Flume can easily scale horizontally.

We start with the Live streaming of data as follows:

* First we will install flume in the system. Copy the tar file of the apache flume and then extract it on the desktop. Now we move this extracted file from the desktop to the destination **/usr/lib/flume** using the **sudo mv** command.



* To check if the file has been moved, we use the command **ls.** Now open the bashrc file as shown.



* We add the following lines in this file for configuration purposes:

**export FLUME\_HOME=/usr/lib/flume**

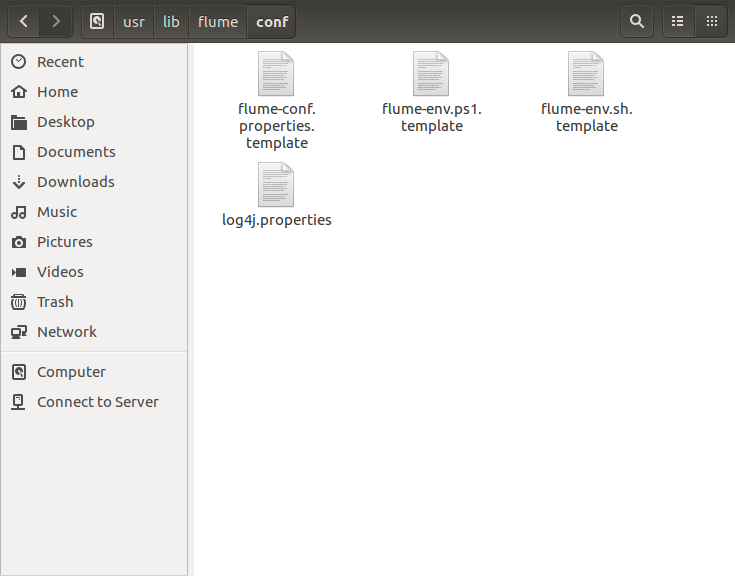
**export FLUME\_CONF\_DIR=$FLUME\_HOME/conf**

**export FLUME\_CLASSPATH=$FLUME\_CONF\_DIR**

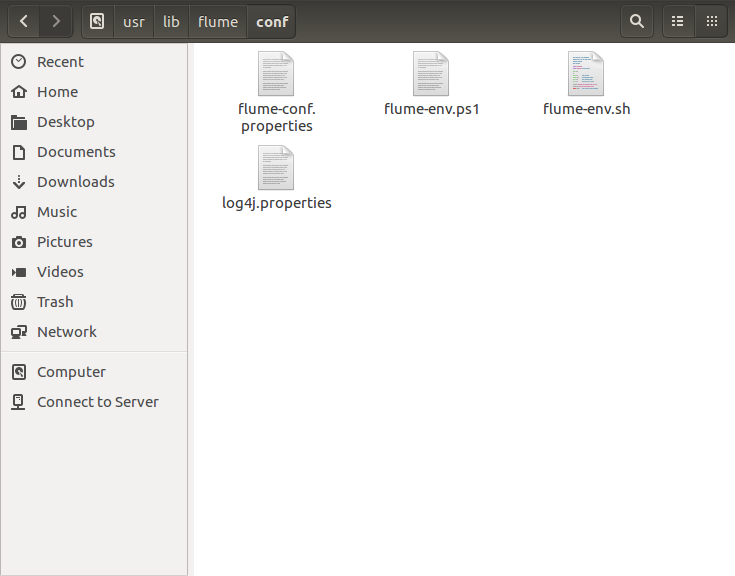
**export PATH=$PATH:$FLUME\_HOME/bin**



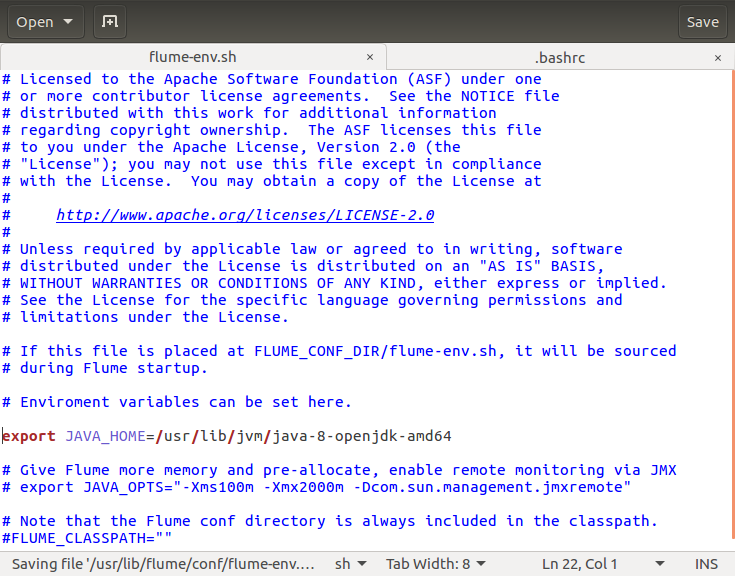
* Now, we change the names of 3 files **flume-conf.properties.template, flume-env.ps1.template** and **flume-env.sh.template** renaming them and removing the word template from their name.



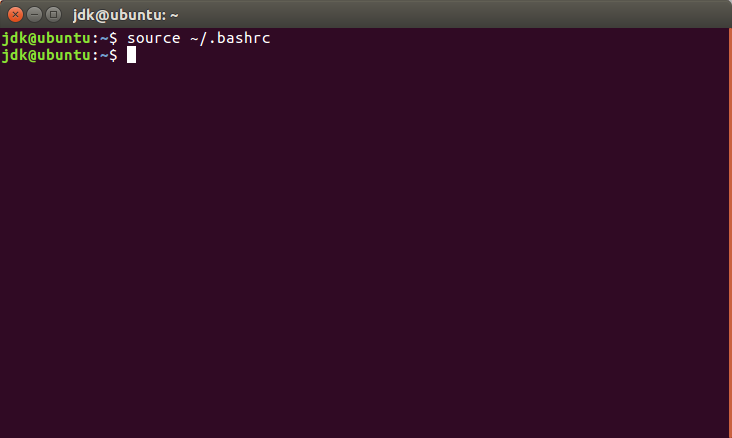
* This is the output of the same.



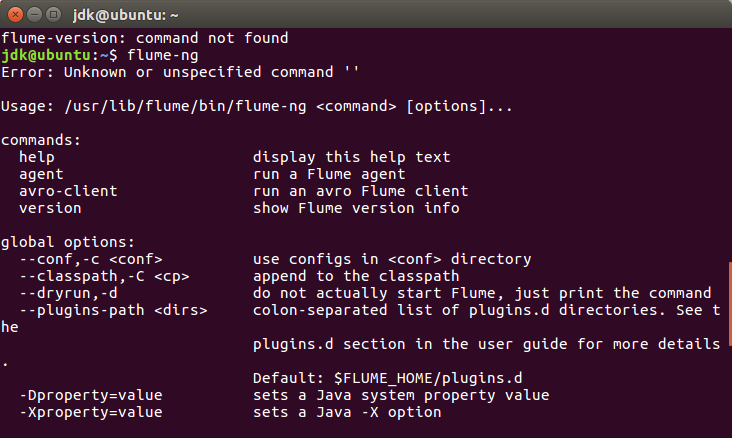
* Now, we open the **flume-env.sh** file and add the JAVA\_HOME PATH in it.



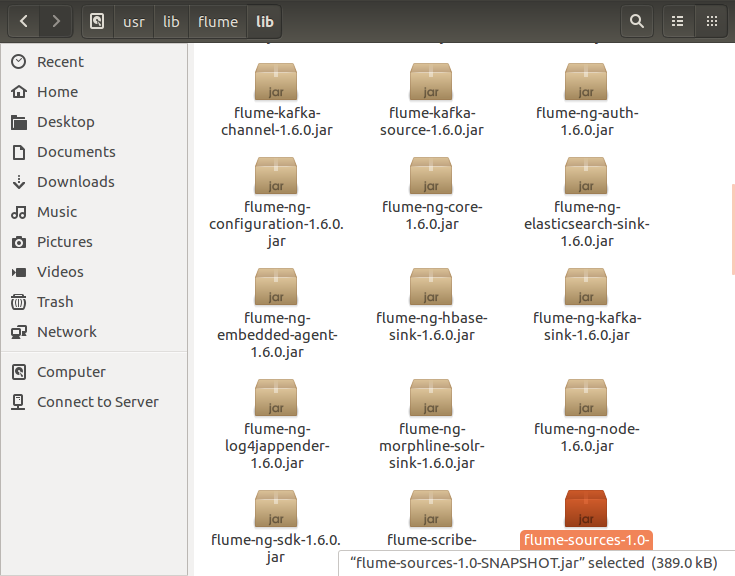
* Now we permanently save the bashrc file.



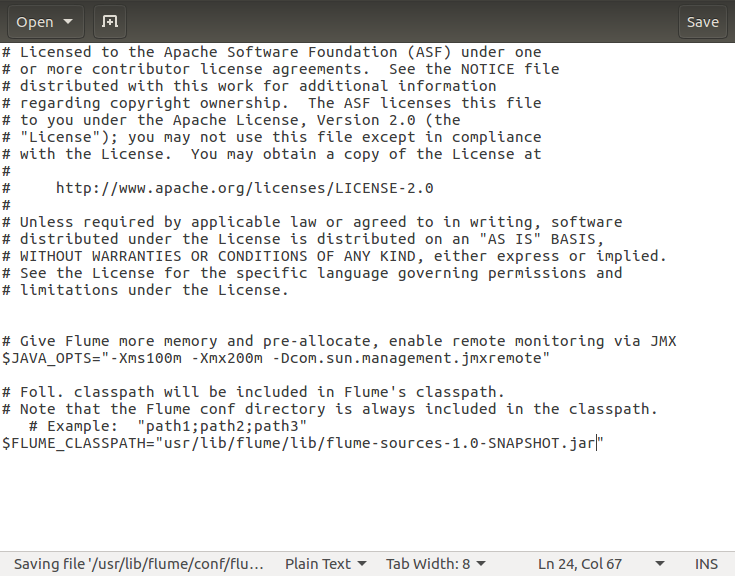
* We use the **flume-ng** command. If the help desk opens then flume has been successfully installed.



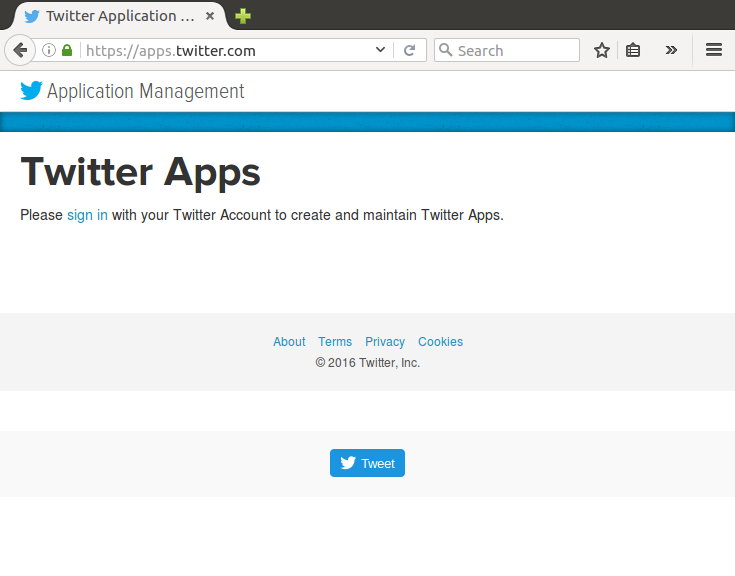
* Now we move to **/usr/libflume/lib** and copy a jar file named **flume-sources-1.0-SNAPSHOT.jar** into this directory.



* After that we open the **flume-env.ps1** file and add the **FLUME\_CLASSPATH** in it.

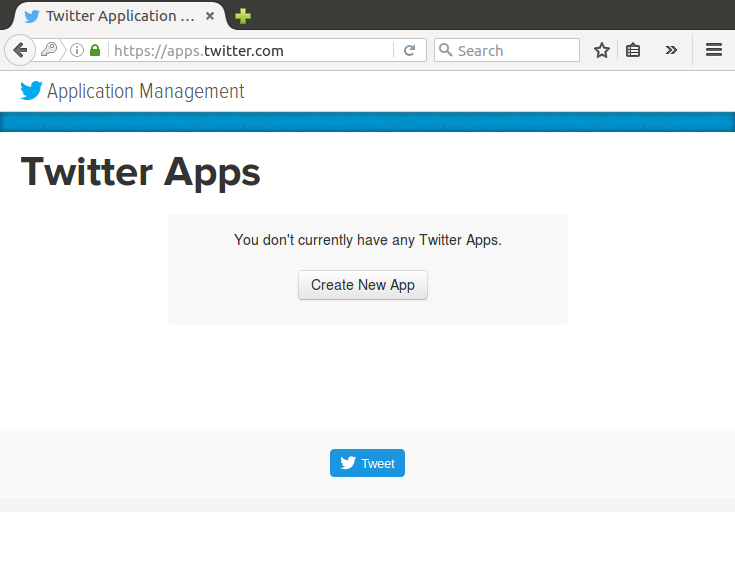


* Now we open the website **dev.twitter.com/apps** in the Mozilla Firefox Browser.

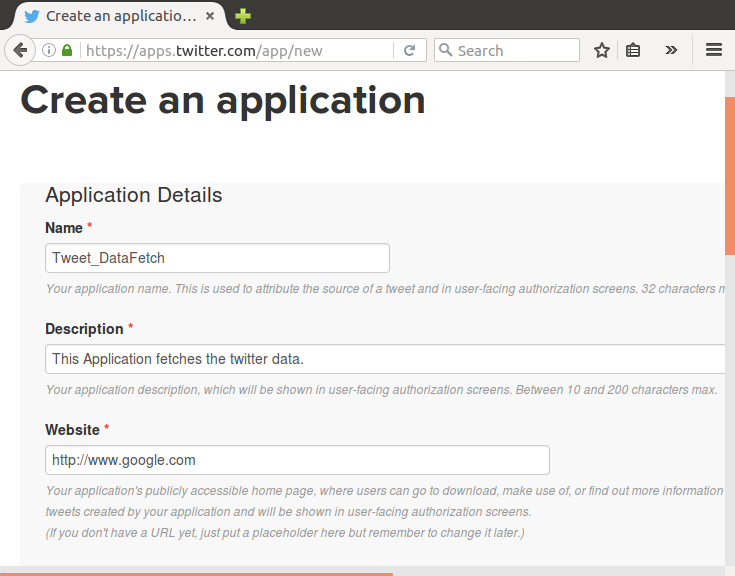


* We will now see the website suggesting us to sign in. So, we sign into our twitter account.

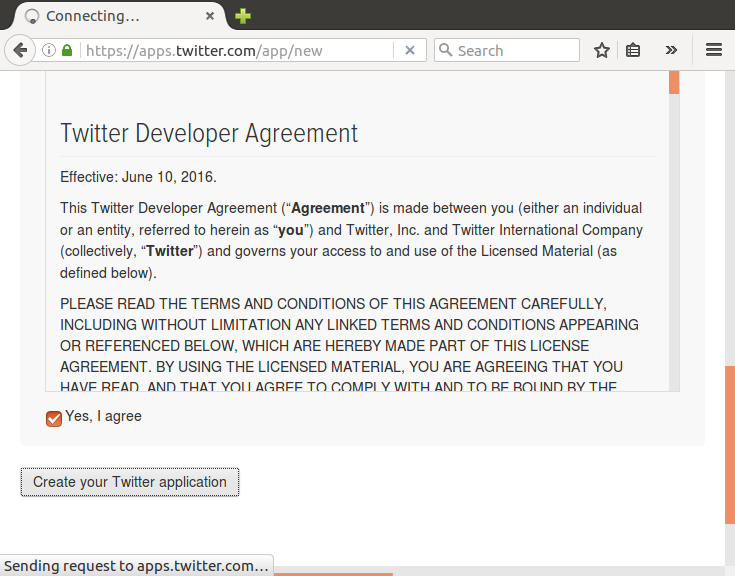
Click on **Create New App.**



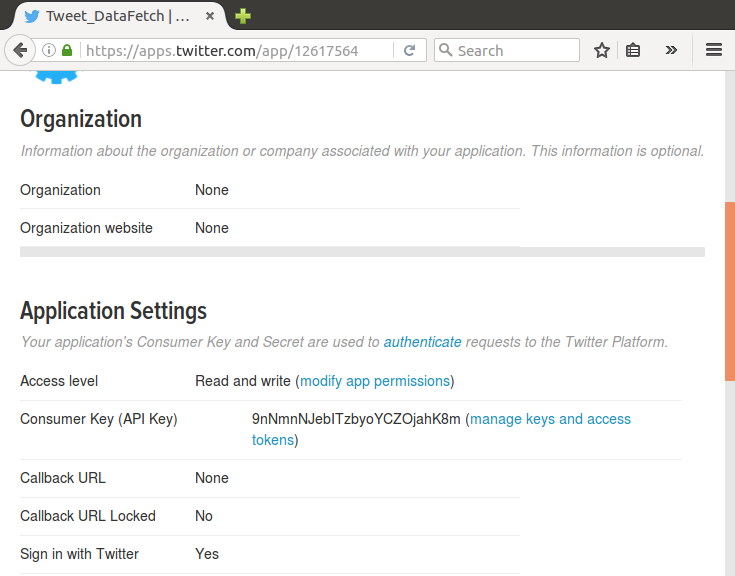
* Fill in all the required fields to make the application and use the website as **google.com**.



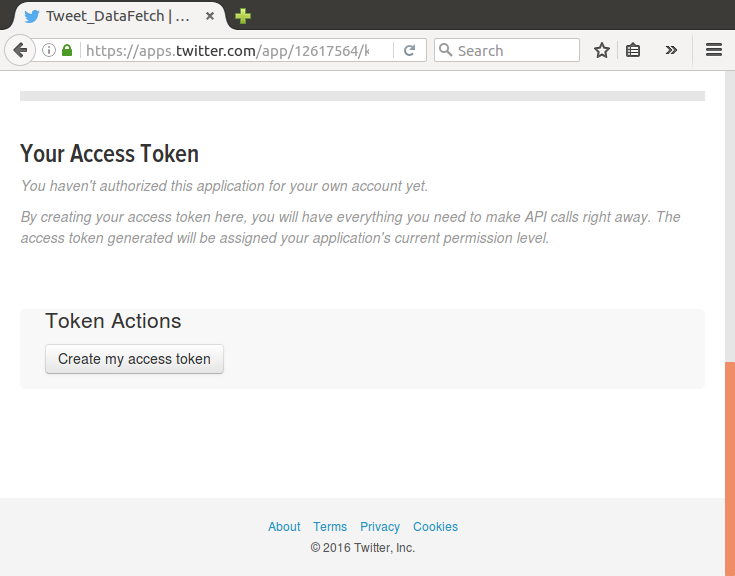
* Now, scroll down and tick the option **Yes, I agree** and then click **Create your Twitter application.**



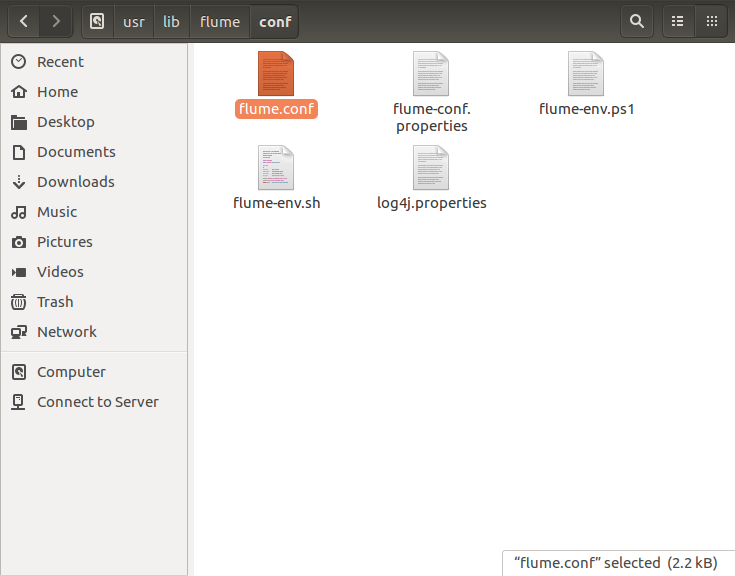
* Click on **manage keys and access tokens.**



* Now click on **Create my access token.**



* Now, we open **flume.conf** file in the directory **/usr/lib/flume/conf** and then change the following keys in the file. These keys will be obtained from the page above.

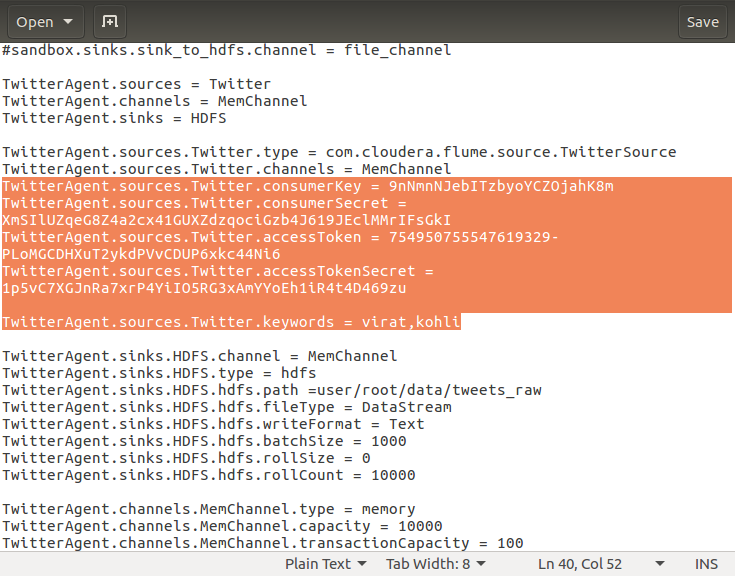


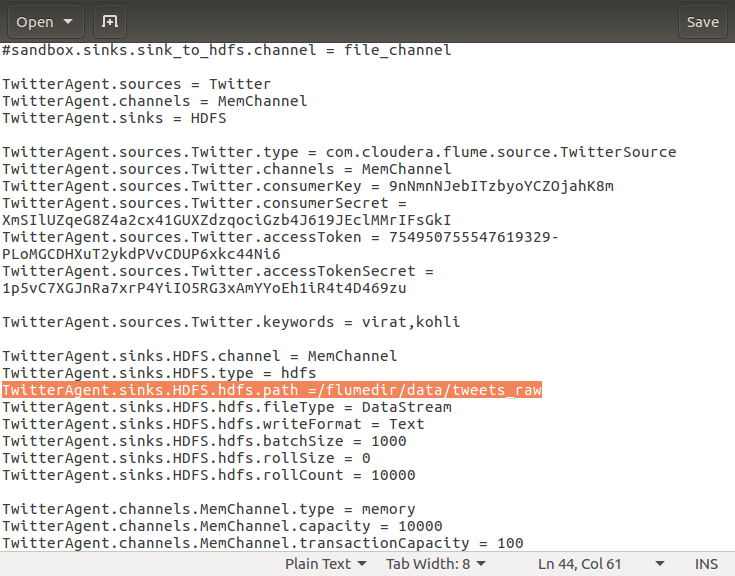
* These are the keys which we will change in the flume.conf file:

**Access Token, Access Token Secret, Consumer Key (API Key), Consumer Secret (API Secret)**



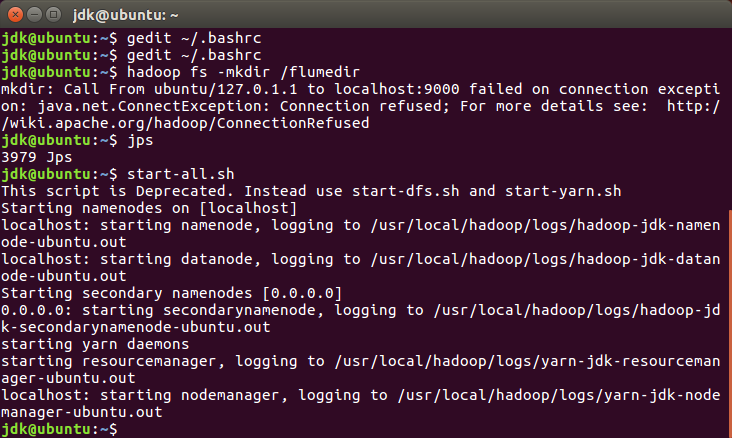
* The keys are changed in this. Also add the keywords that we want to extract from twitter.



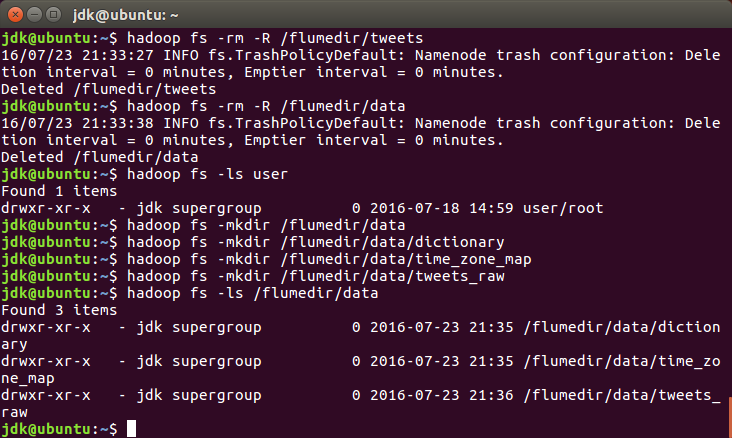


**DATA EXTRACTION FROM TWITTER:**

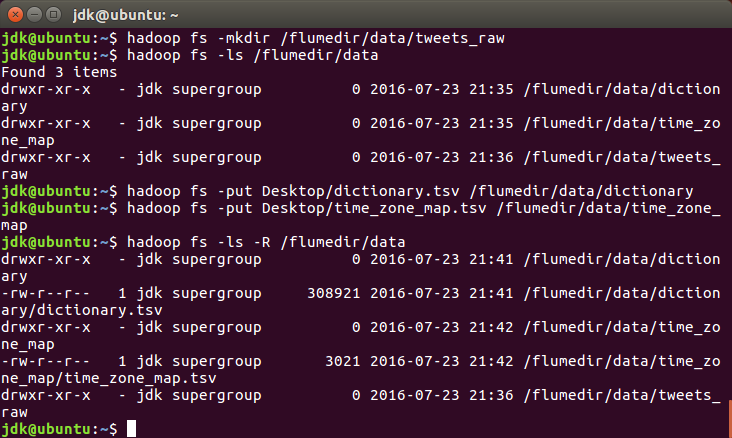
* Start all the services using the **start-all.sh** command.



* Now, make the following new directories:

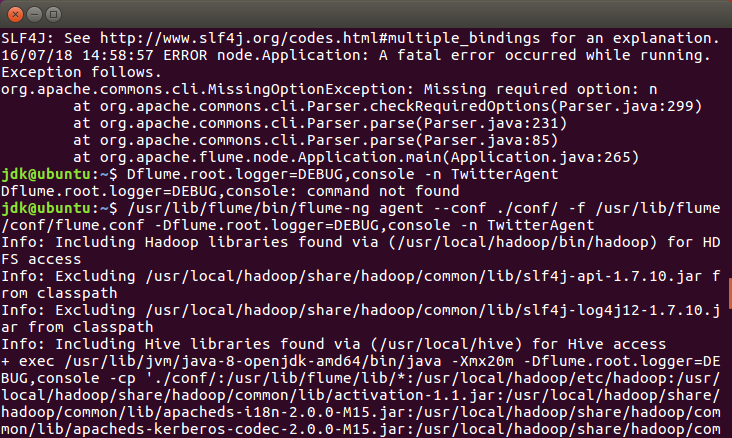


* Then we copy the files **dictionary.tsv** and **time\_zone\_map.tsv** to their respective directories:

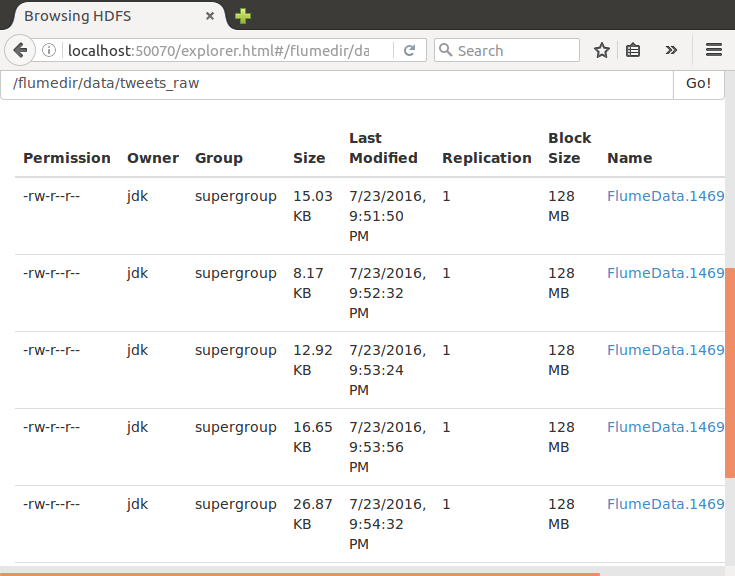


* We will now start the flume agent using the following command:

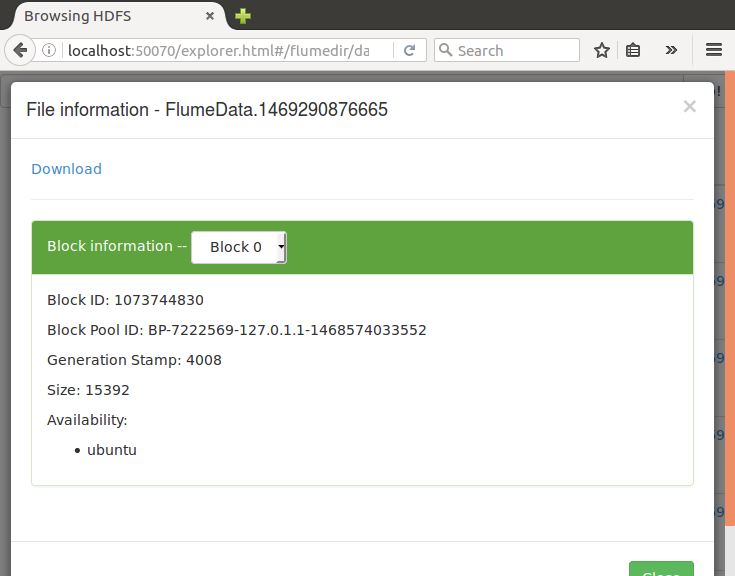
**/usr/lib/flume/bin/flume-ng agent --conf ./conf/ -f /usr/lib/flume/conf/flume.conf -Dflume.root.logger=DEBUG,console -n TwitterAgent**



* This is the list of twitter data extracted which contains the keyword as specified in the conf file.



* We can check the files by downloading them and seeing the tweets relating to the keyword.



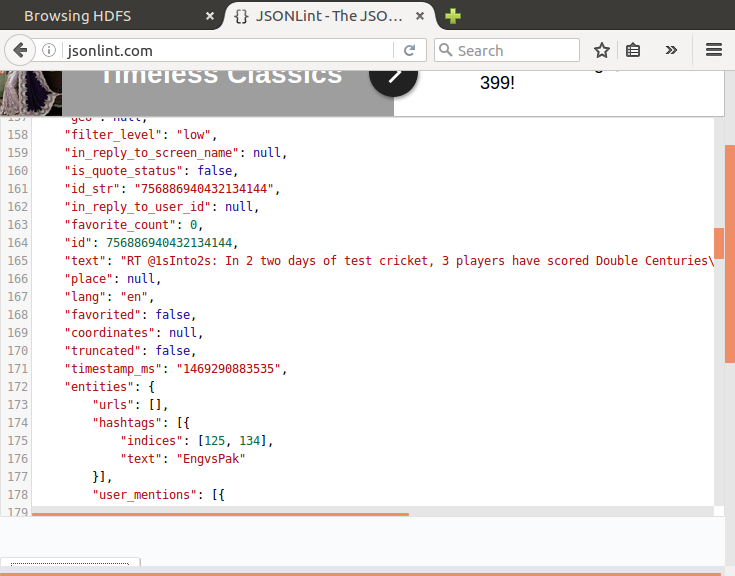
* The file downloaded will be like this:



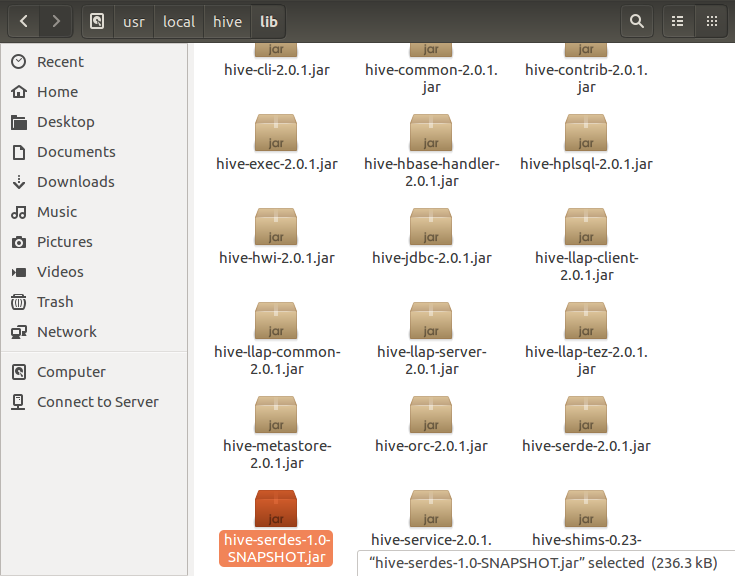
* If we clean this downloaded file using the **JSON Validator** we will see the tweets in a human readable format as shown:



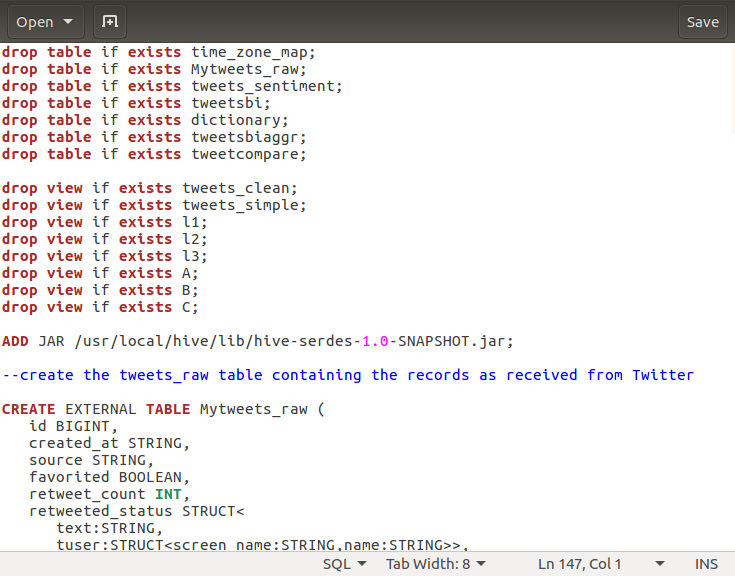
Then click on Validate JSON. The output that will become is as follows:

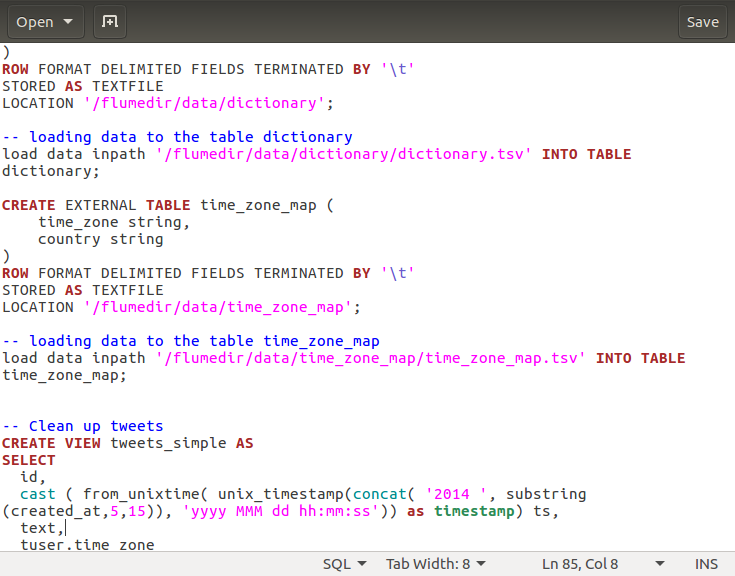
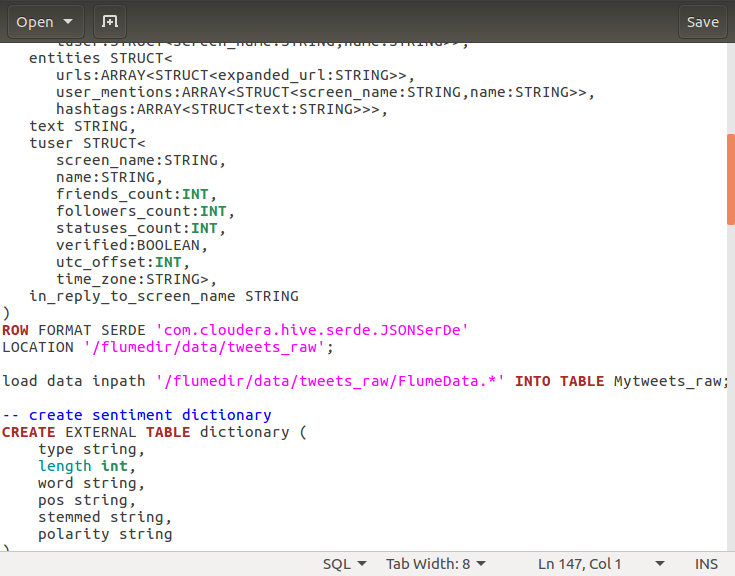


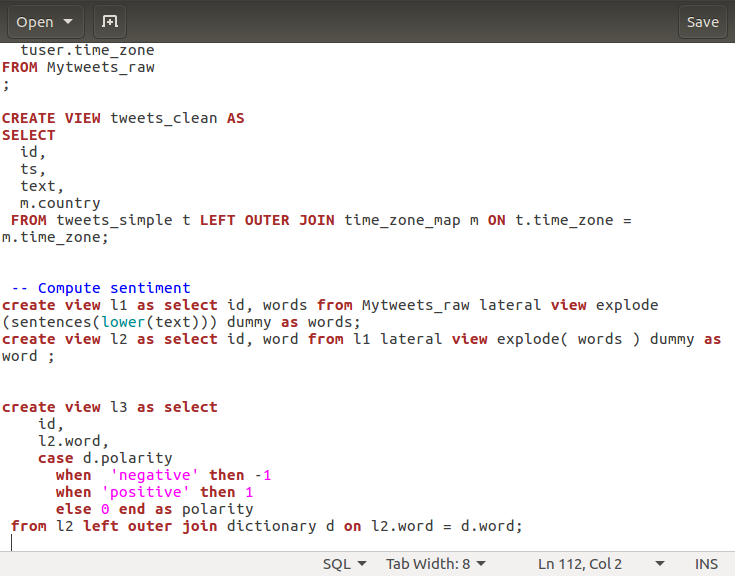
* Now we copy the **hive-serdes-1.0-SNAPSHOT.jar** file into the directory **/usr/local/hive/lib**. This will be used by the hive shell to extract the clean data from the downloaded data into the hive table.

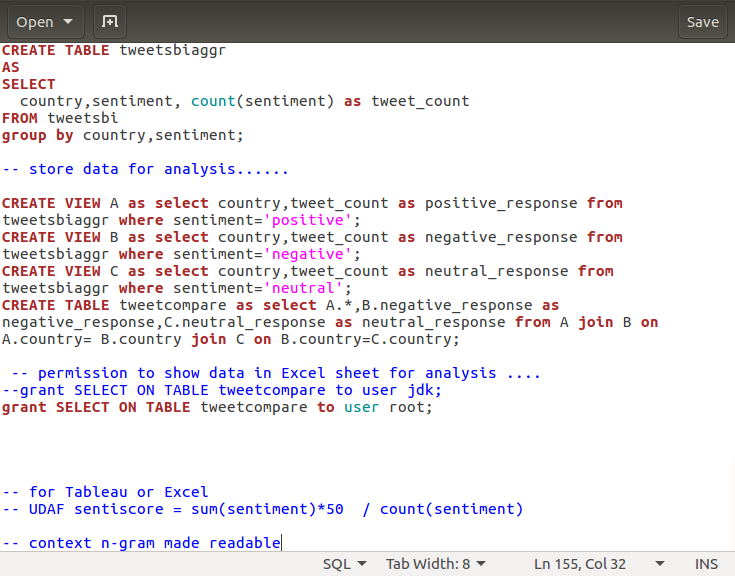
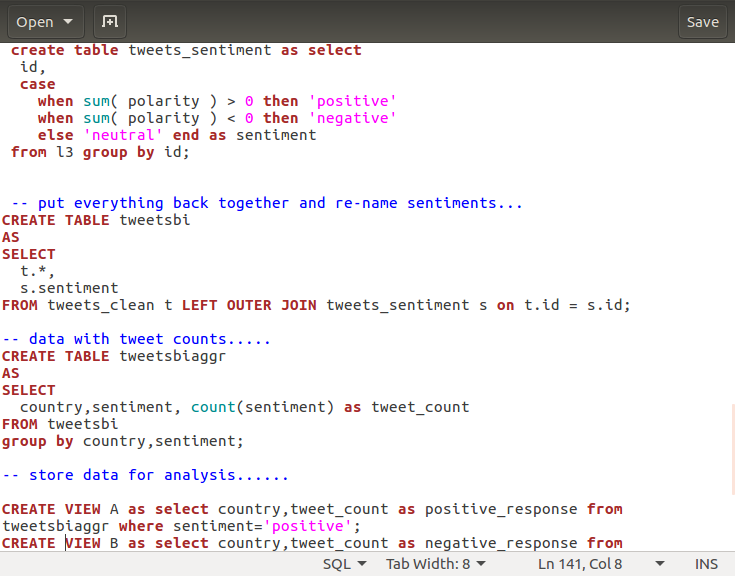


* Now we create a file **tweets.sql** which is on the Desktop.

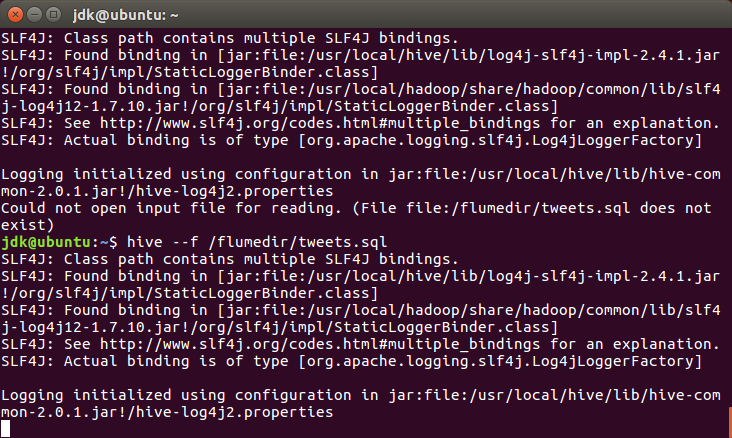




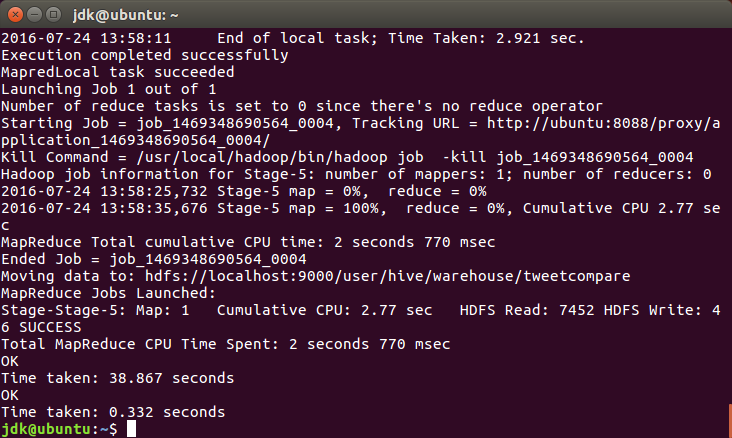




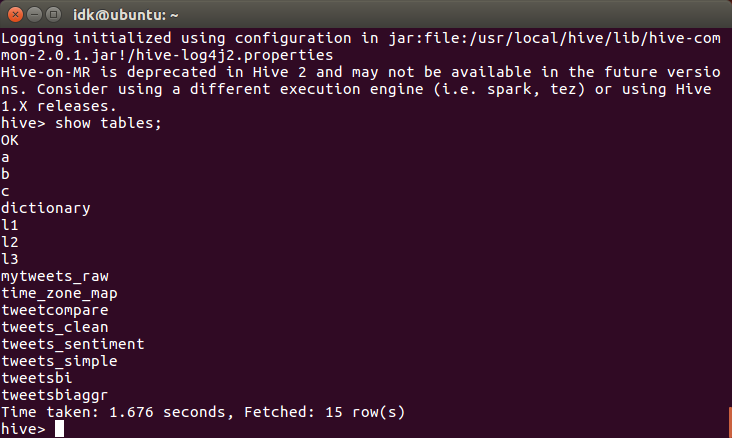
* Now run the tweets.sql file using the hive command.



* After running the script, we receive this output as SUCCESS.

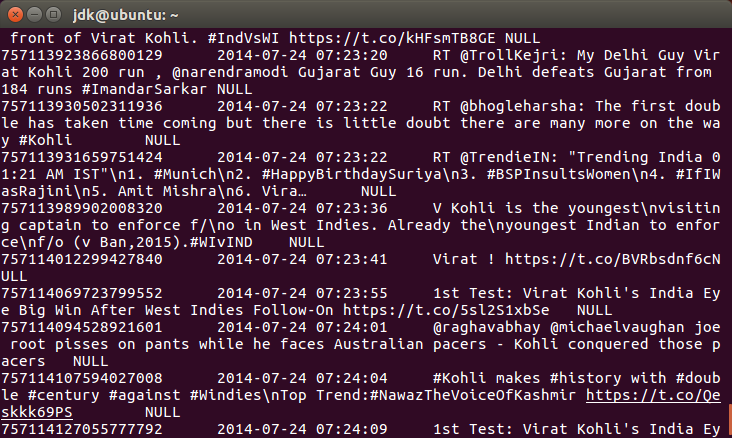


* Now we look into all the created tables in the **hive shell** and **default database**.

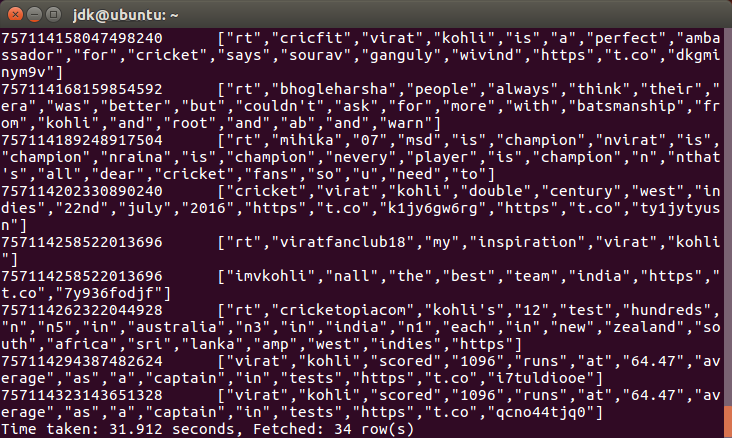


* The outputs can be seen as follows:

Tweets\_simple



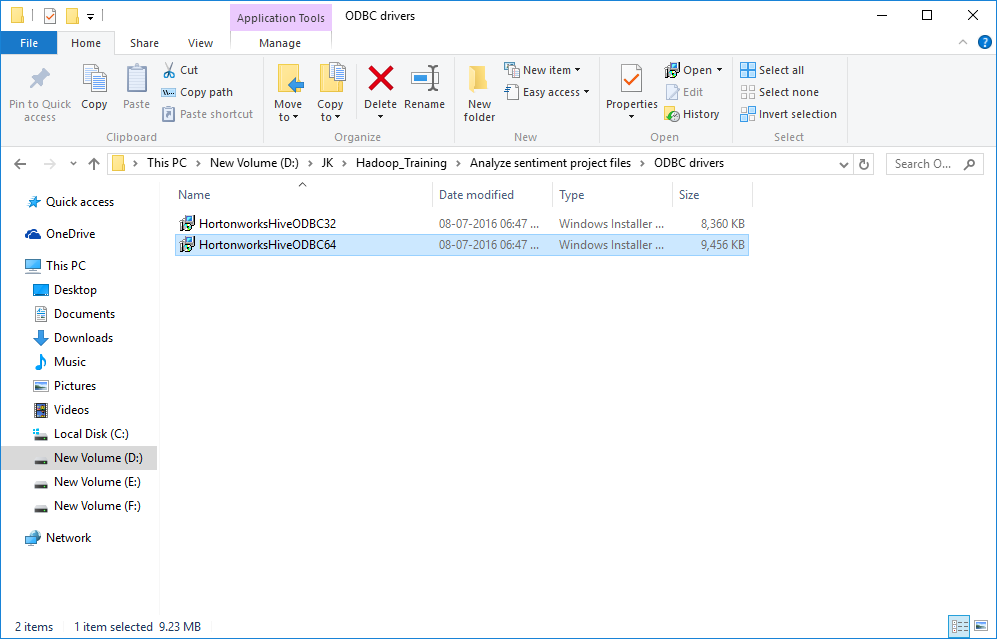
Tweets\_clean



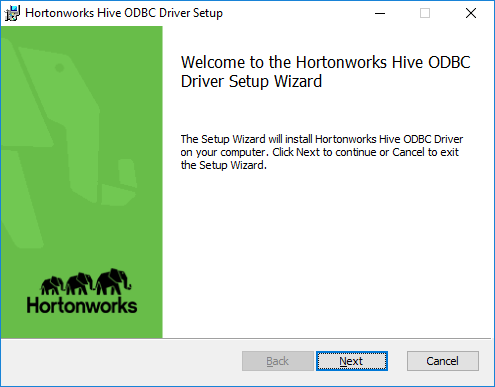
Tweets\_sentiment

**ODBC DRIVER INSTALLATION:**

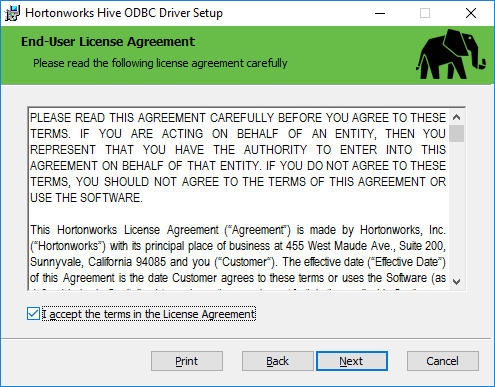
* Select the 64/32 bit driver according to the system.



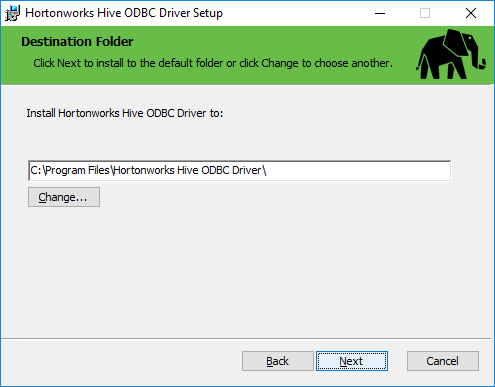
* Click on Next.



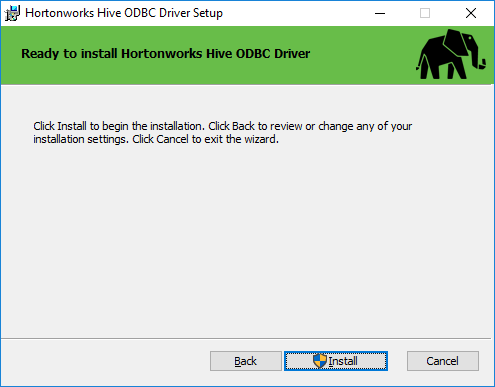
* Accept the agreement and then click on Next.



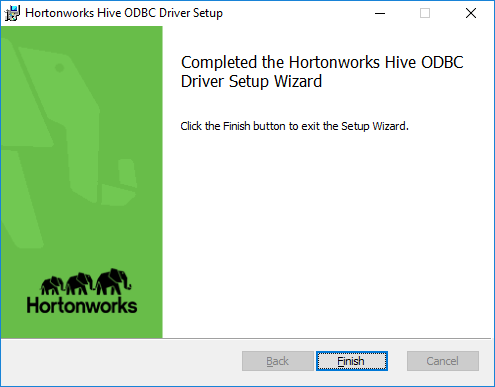
* Specify the location and then click Next.



* Click on Install button.

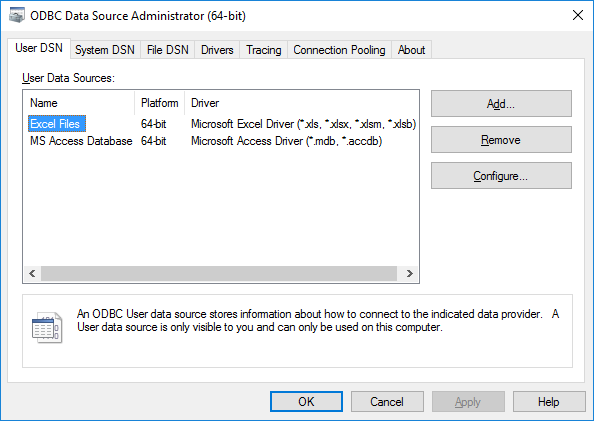


* Finally, click on Finish.

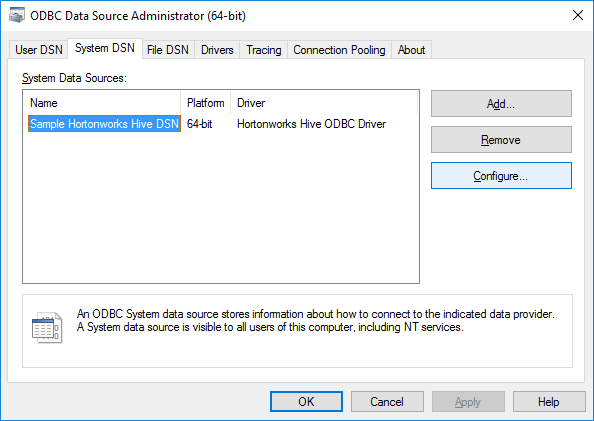


**CONFIGURE ODBC DRIVER:**

* Open Windows control panel. Open Administrative Tools. Double click Data Sources (ODBC) to open ODBC Data Source Administrator.



* Open the System DSN tab and click on Configure.



* Change the port no. and host name as follows:

