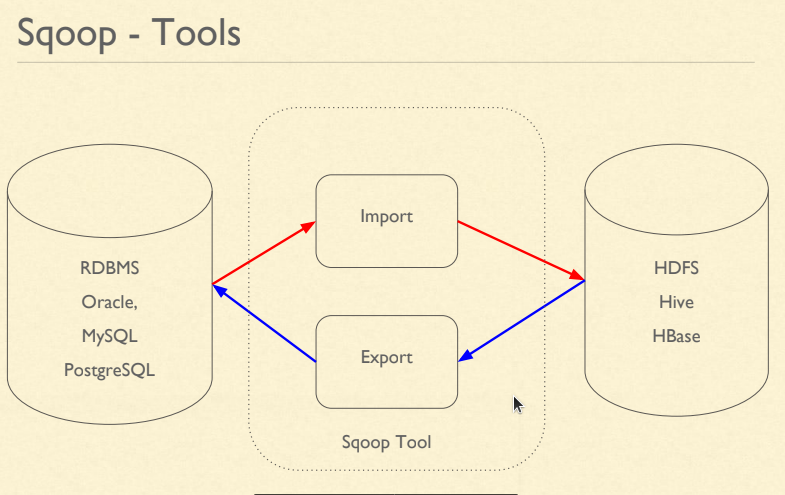
# Sqoop Introduction

It is exciting to know the reason behind the name Sqoop. Sqoop got its name from **“SQL to Hadoop & Hadoop to SQL”**.

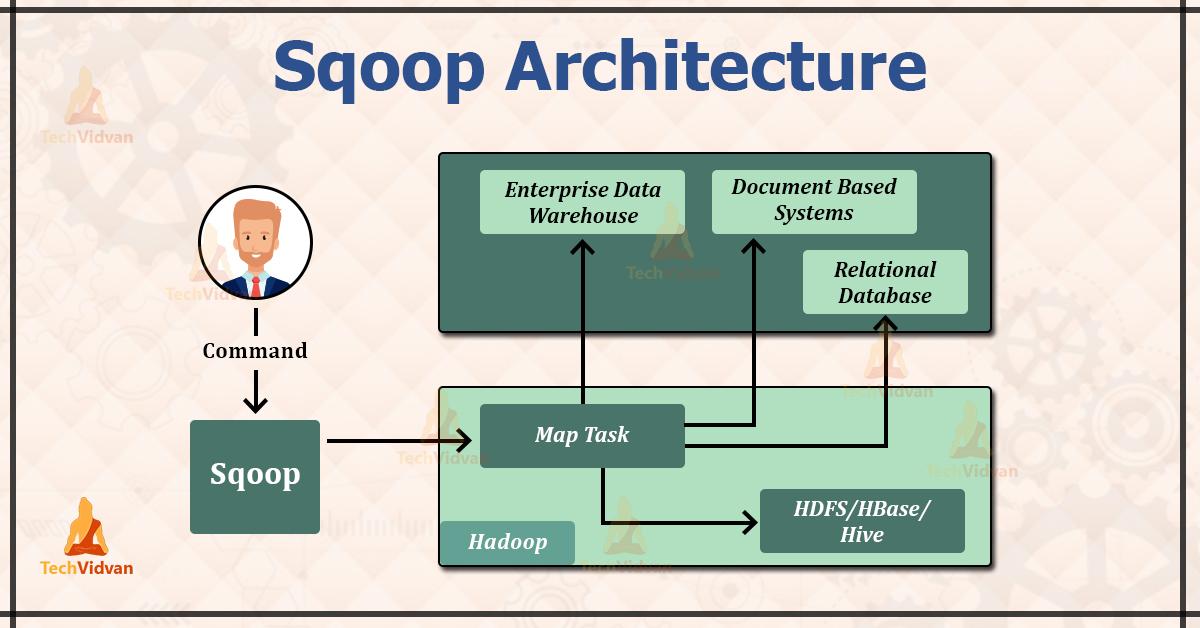
Apache Sqoop is a tool by Apache Software Foundation for transferring data between the Hadoop and the relational database servers like MySQL, SQLite, Oracle RDB, Teradata, Postgres, Netezza, and many more.

In simple words, Sqoop is useful for importing data from relational databases such as Oracle, MySQL to Hadoop HDFS, and for exporting data from HDFS to relational databases.

Apache Sqoop can transfer bulkier data efficiently between the Hadoop system and the external data stores like enterprise data warehouses, RDBMS, etc.



### **Sqoop Architecture and Working**

****

The above image depicts Sqoop Architecture.

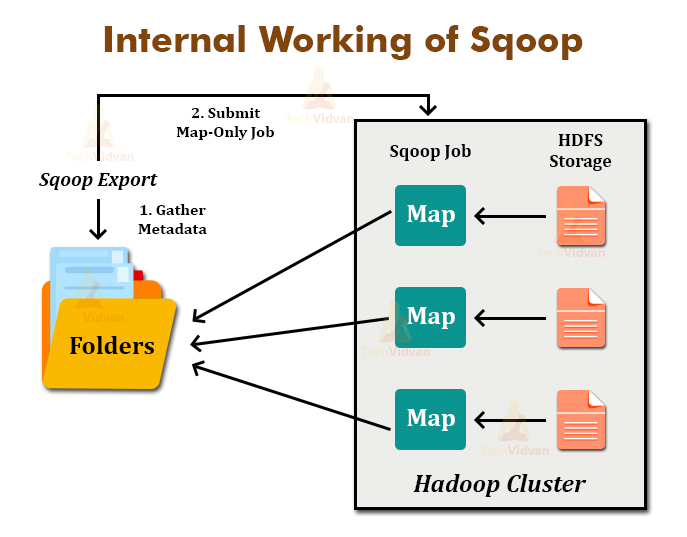
Apache Sqoop provides the command-line interface to its end users. We can also access Sqoop via Java APIs. The Sqoop commands which are submitted by the end-user are read and parsed by the Sqoop. The Sqoop launches the Hadoop Map only job for importing or exporting data.

No Reduce job is launched because the Reduce phase is needed only when the aggregations are performed. Apache Sqoop just imports and exports data, and hence it does not perform any aggregations due to which we don’t require a Reduce phase.

Apache Sqoop parses the arguments which are provided in the command line and launches the Map only job. The Map only job launches multiple mappers depending on the number defined by the user in the command line.

For import, each mapper task is assigned with the part of data that is to be imported on the basis of the key defined in a command line. For getting higher performance, Sqoop distributes input data equally amongst all the mappers.

Each mapper then creates a connection with the database by using the JDBC and fetches part of the data assigned by the Sqoop. They then write those data into HDFS or HBase or Hive on the basis of the option provided in the command line.



Sqoop Export also works in the same way. The Sqoop Export tool exports the set of files from the Hadoop Distributed File System back to the Relational Database. The files which are given as an input to the Sqoop contain records. These records are called rows in a table.

When the user submits it Job, then it is mapped into the Map Tasks that bring chunks of data from the Hadoop Distributed File System. These chunks are then exported to any structured data destination.

By combining all these chunks of data, the user receives the entire data at destination, which is generally an RDBMS such as MYSQL, SQL Server, Oracle, etc.

**Sqoop provides features like:**

* Full Load.
* Incremental Load.
* Parallel import/export.
* Import results of SQL query.
* Compression.
* Connectors for all major RDBMS Databases.
* Kerberos Security Integration.
* Load data directly into Hive/Hbase.

## **Sqoop installation**

**Download sqoop :** [**http://archive.apache.org/dist/sqoop/1.4.6/**](http://archive.apache.org/dist/sqoop/1.4.6/)

With the below tar version :

[**sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha.tar.gz**](http://archive.apache.org/dist/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz)

Extract the contents by right clicking it ,extract (or) use command as :

tar -xvf [**sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha.tar.gz**](http://archive.apache.org/dist/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz)

Update environment variables with filepath of extracted file package :

---in terminal type command to edit the environment variables;

$ gedit .bashrc

In the bashrc file :give the command as :

export SQOOP\_HOME=/home/hdoop/sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha

export PATH=$PATH:$SQOOP\_HOME/bin

Save the file; type command as :

source ~/.bashrc

----- To check whether sqoop installed ==>sqoop help

Commands shows that sqoop installed;

Now go to the folder where sqoop file is extracted,inside it go to conf and insert hadoop path inside its env.sh file:

~/Downloads$ cd sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha/

~/Downloads/sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha$ cd conf

~/Downloads/sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha/conf$ ls

~/Downloads/sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha/conf$ nano sqoop-env-template.sh

Inside give the following that is pasting the hadoop path

export HADOOP\_COMMON\_HOME=/home/hdoop/hadoop-3.3.1

export HADOOP\_MAPRED\_HOME=/home/hdoop/hadoop-3.3.1

download jar commons-lang-2.6.jar and commons-lang3-3.12.0.jar and paste in $sqoophome/lib folder<https://mirrors.tuna.tsinghua.edu.cn/apache//commons/lang/binaries/>

Click on the link and download:

[commons-lang-2.6-bin.tar.gz](https://mirrors.tuna.tsinghua.edu.cn/apache//commons/lang/binaries/commons-lang-2.6-bin.tar.gz)

Then extract the folder and in extracted file there will be 3 commons jar folder copy that and paste it in sqoop extracted folder in lib;

--Download the mysql connector java (jdbc) from the below link:

<https://dev.mysql.com/downloads/connector/j/?os=26>

Extract the file and inside that copy the mysql connector-jar- 8.0.27

$tar -xvf mysql-connector-java-8.0.27.tar.gz

and paste it in an extracted file of sqoop, inside the bin folder. Save it.

**Connecting to mysql :**

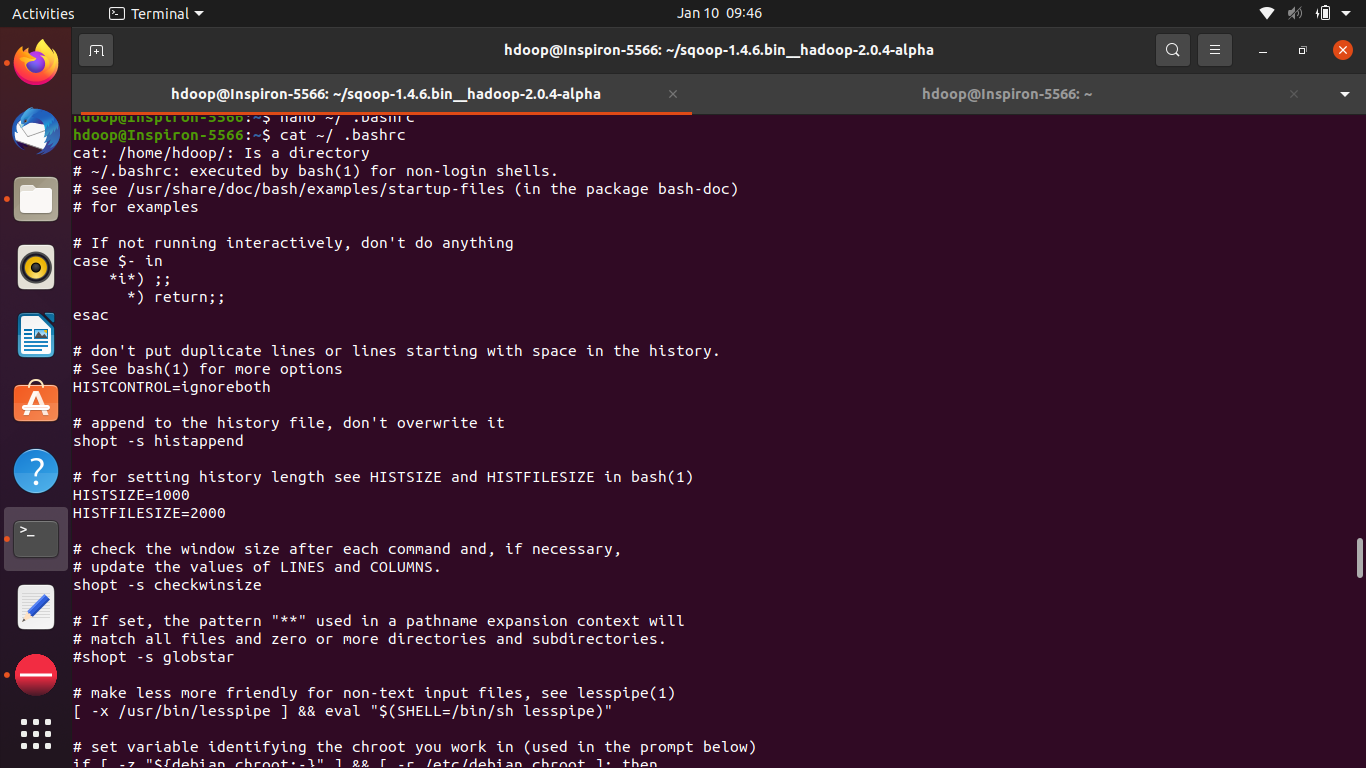
/usr/bin/mysql -u root -p

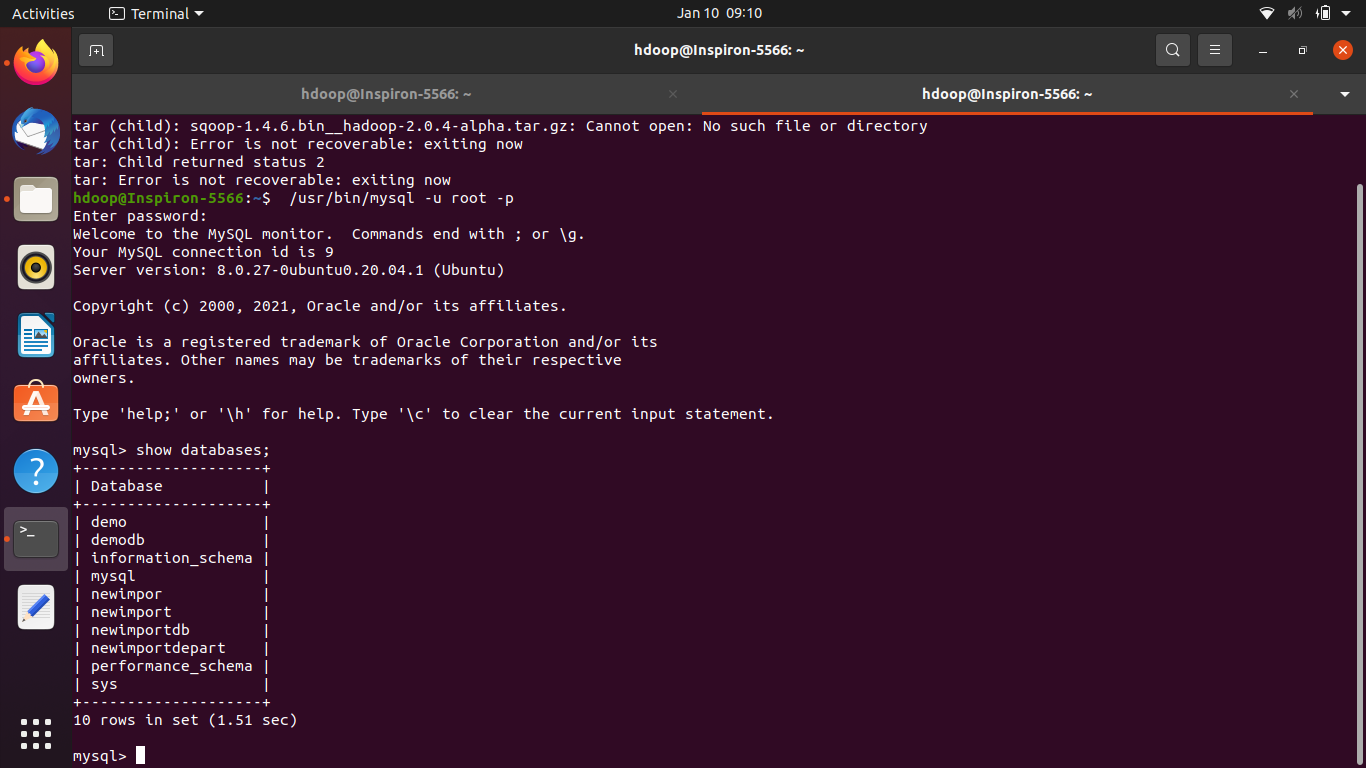
mysql> show databases;

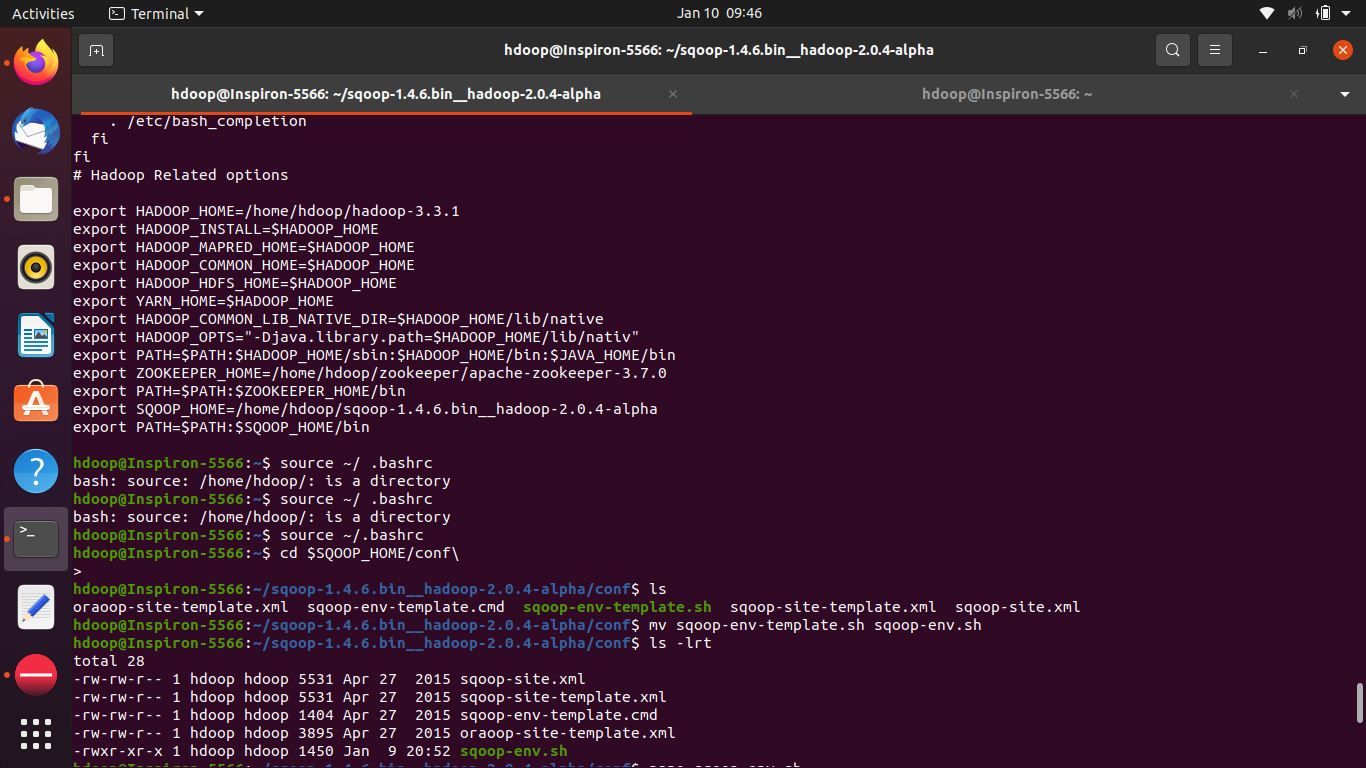
mysql> use employeedb;

mysql> show tables;

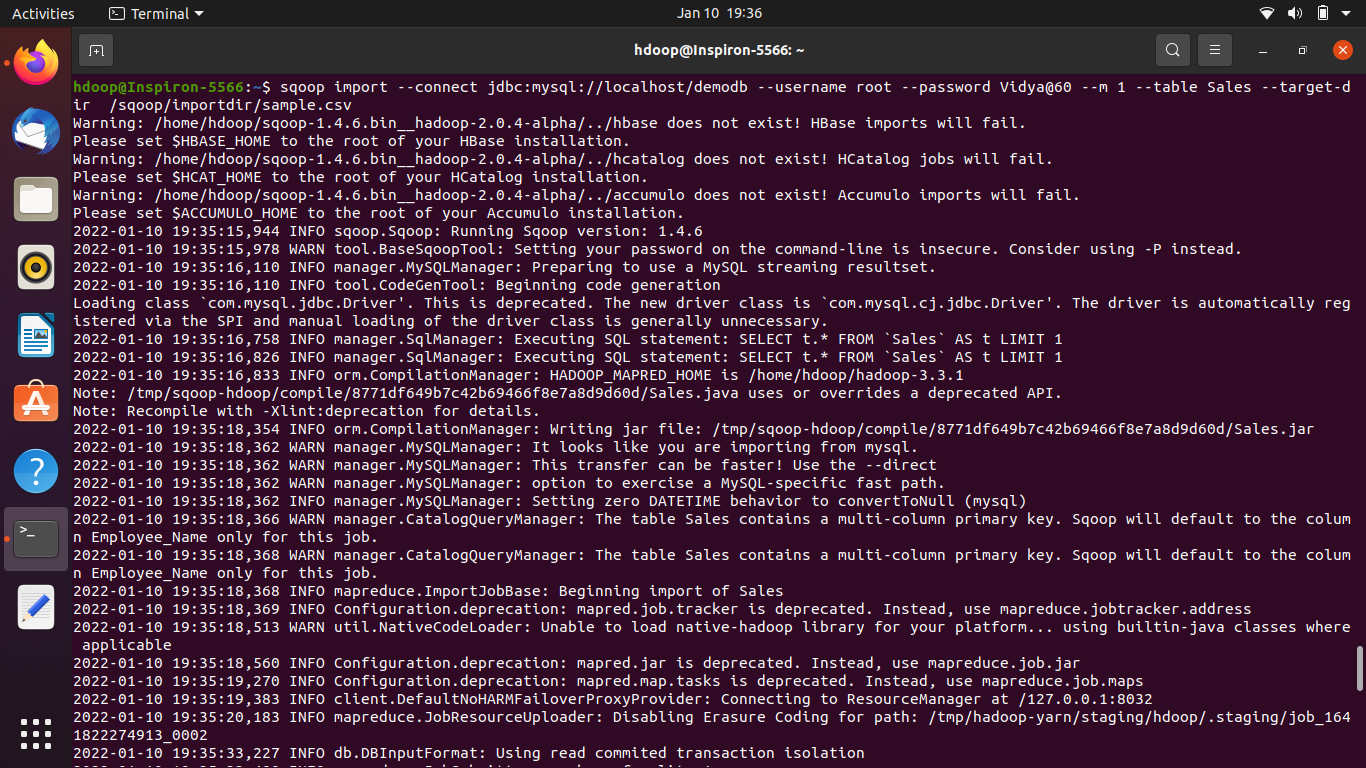
mysql> select \* from employee;

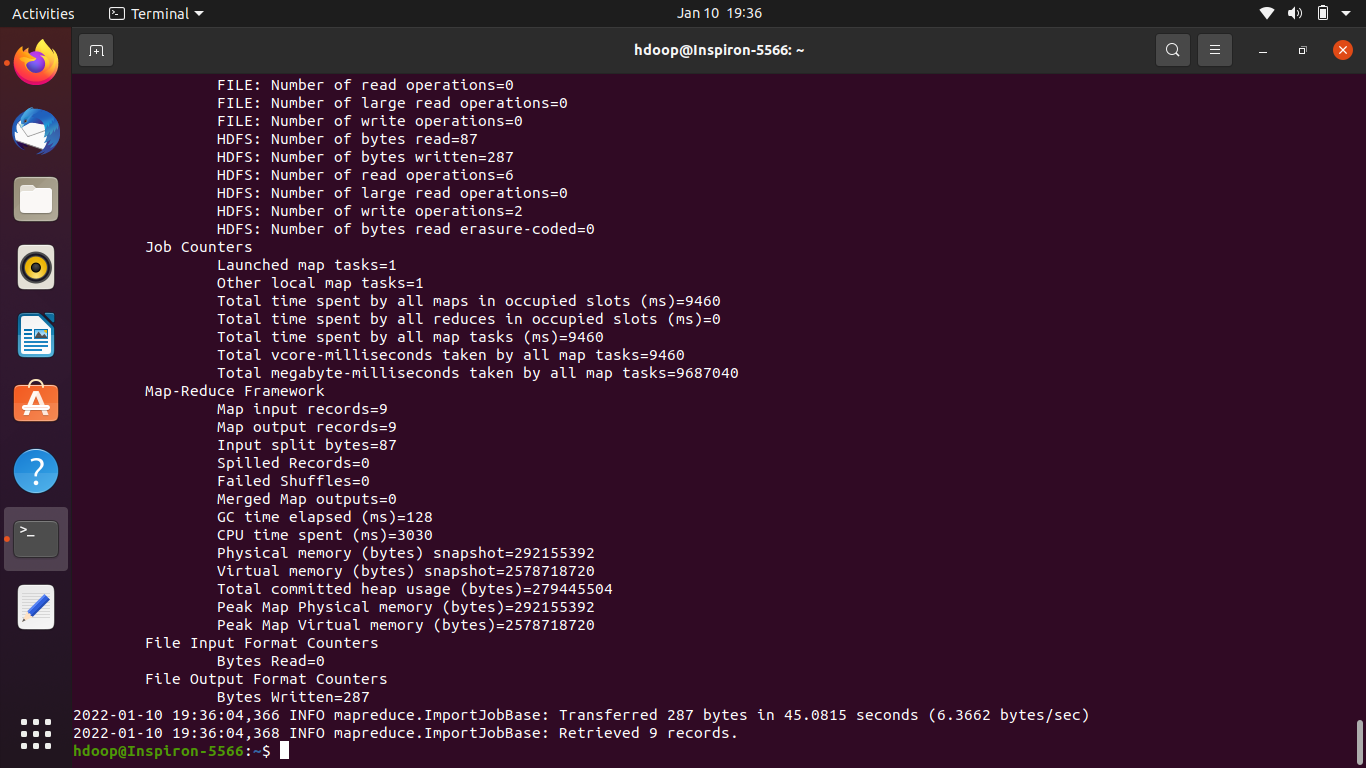






$ sqoop import --connect jdbc:mysql://localhost/demodb --username root --password Vidya@60 --m 1 --table Sales --target-dir /sqoop/importdir/



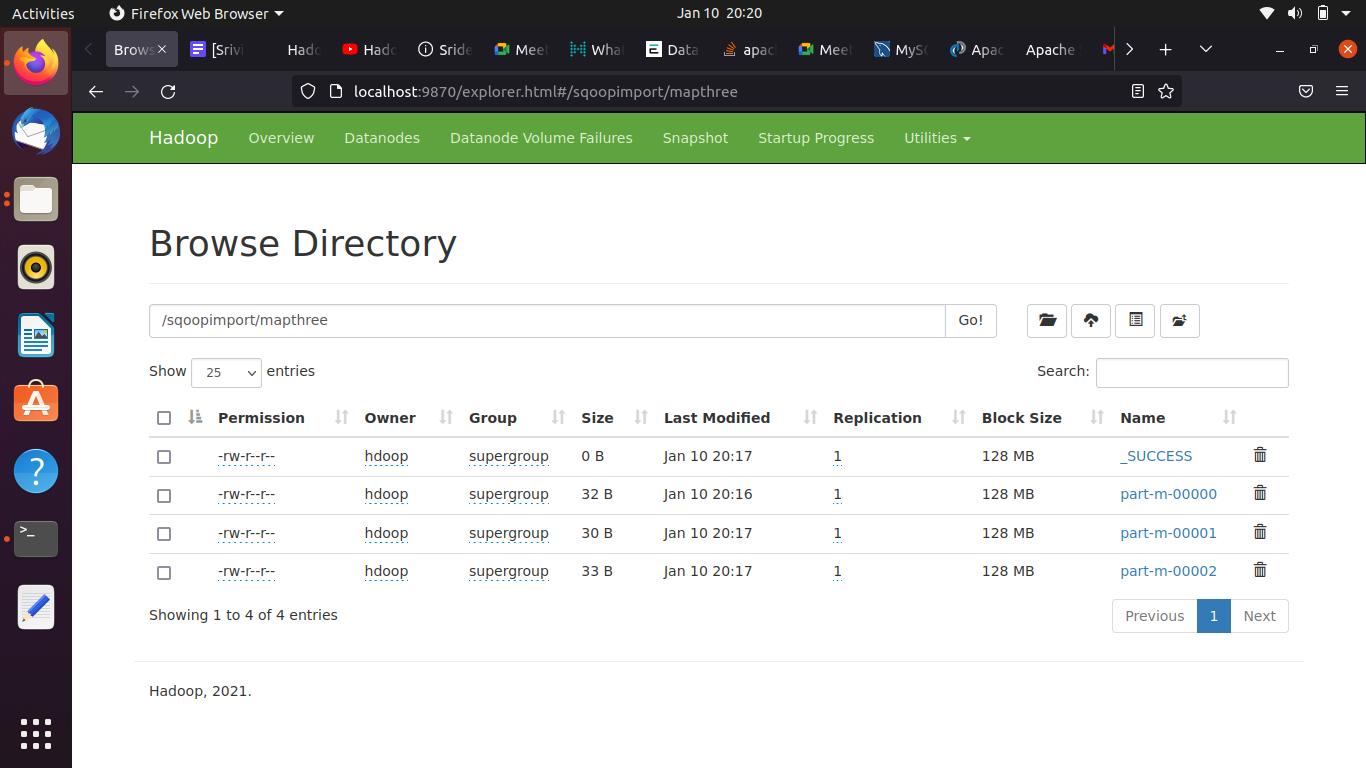






## 

$ sqoop import --connect jdbc:mysql://localhost/demodb --username root -P --table employees --target-dir /sqoopimport/mapthree --m 3



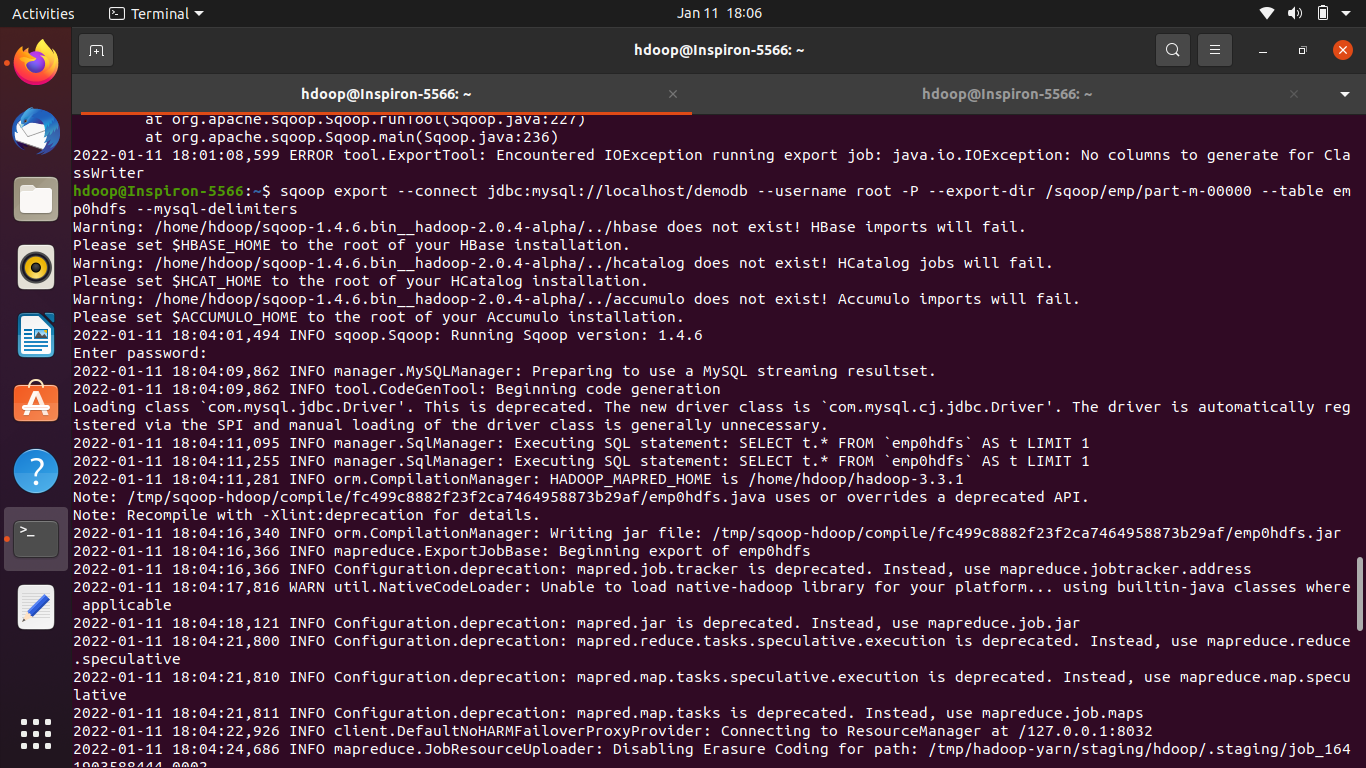
**Create table emp0hdfs according to the data of the file in hdfs data stored:**

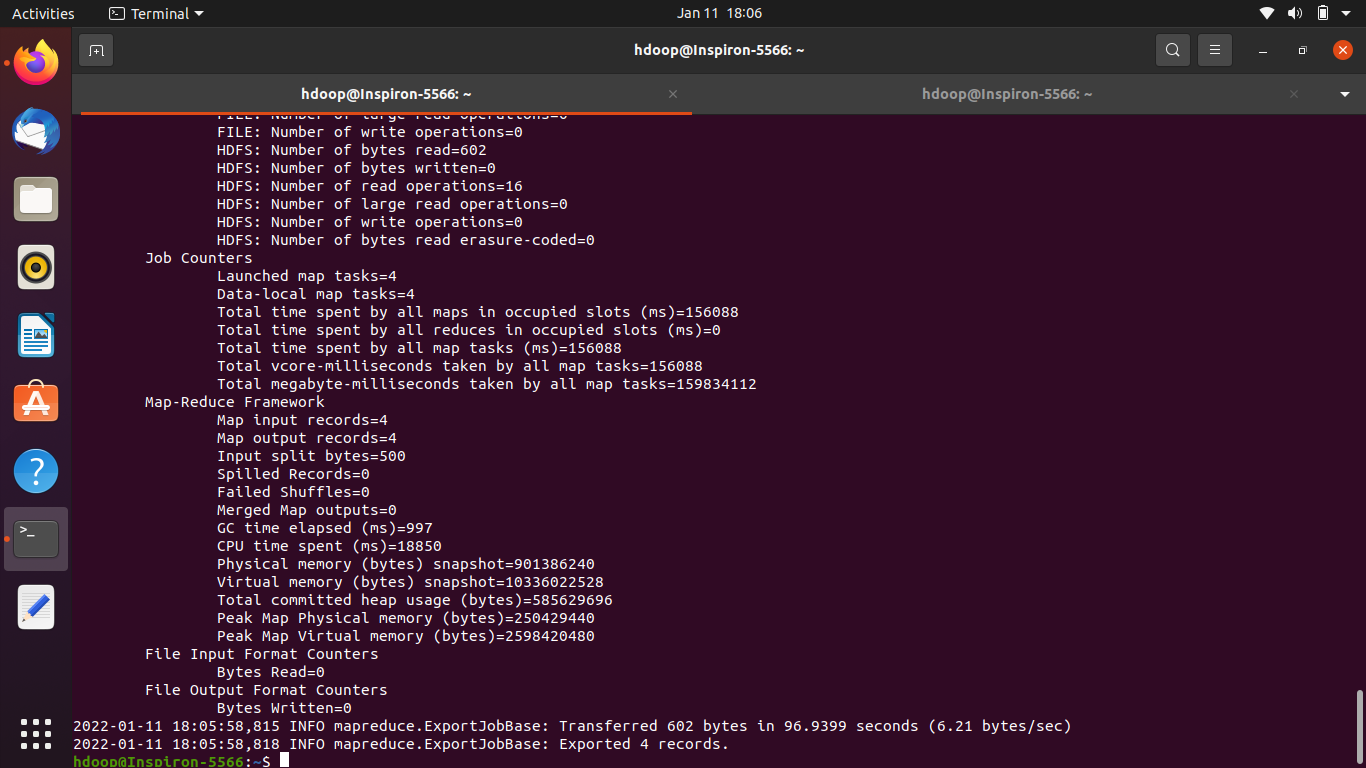
mysql> create table emp0hdfs(gender varchar(20),id int primary key,name varchar(20));

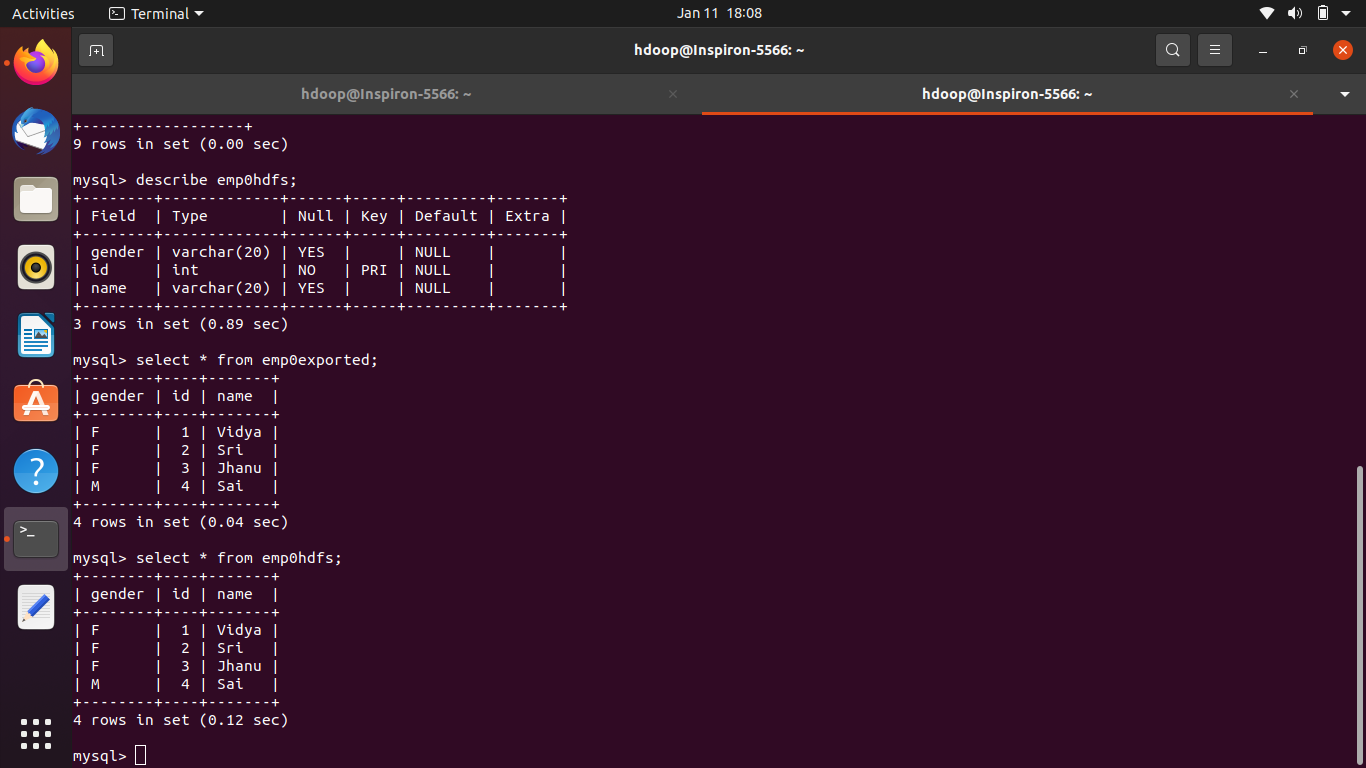
mysql> desc emp0exported;

mysql> select \* from emp0exported;

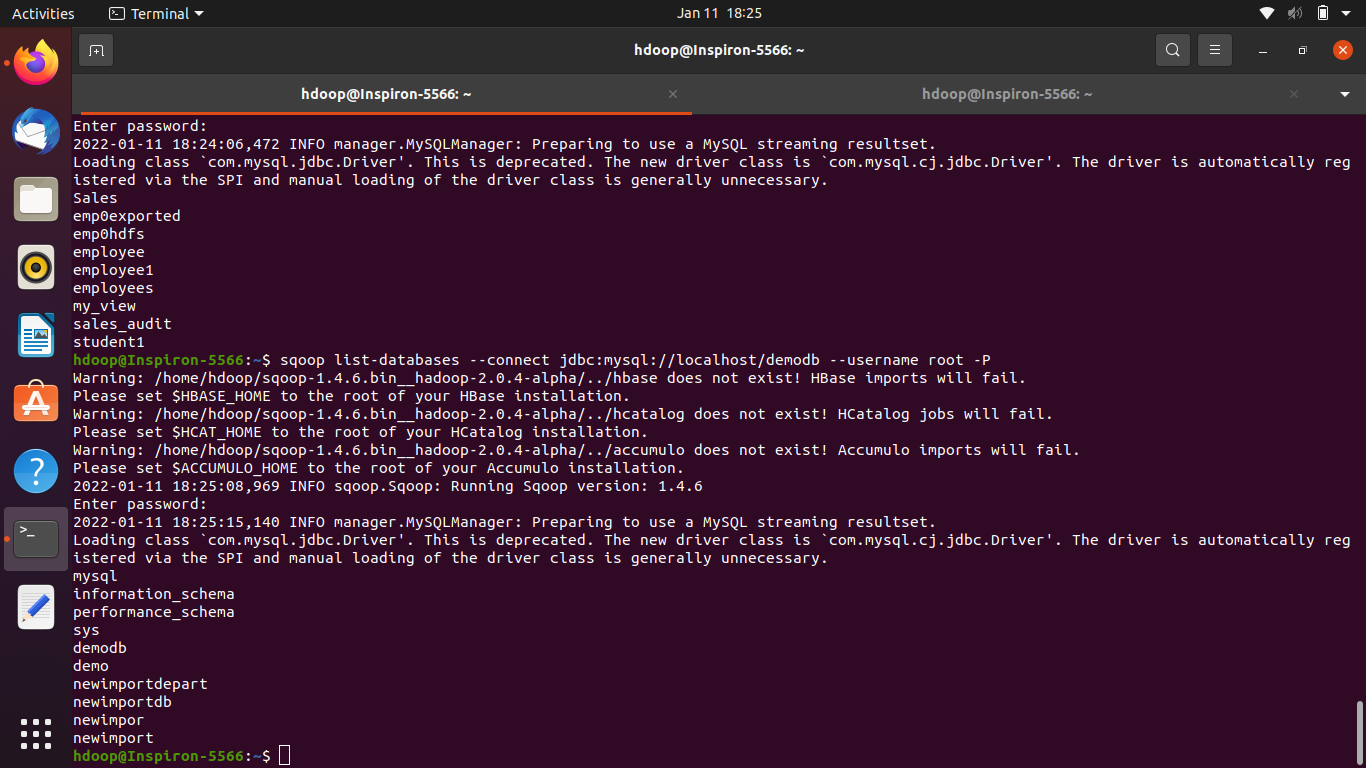
sqoop export --connect jdbc:mysql://localhost/demodb --username root -P --export-dir /sqoop/emp/part-m-00000 --table emp0hdfs --mysql-delimiters







$ sqoop list-tables --connect jdbc:mysql://localhost/demodb --username root -P



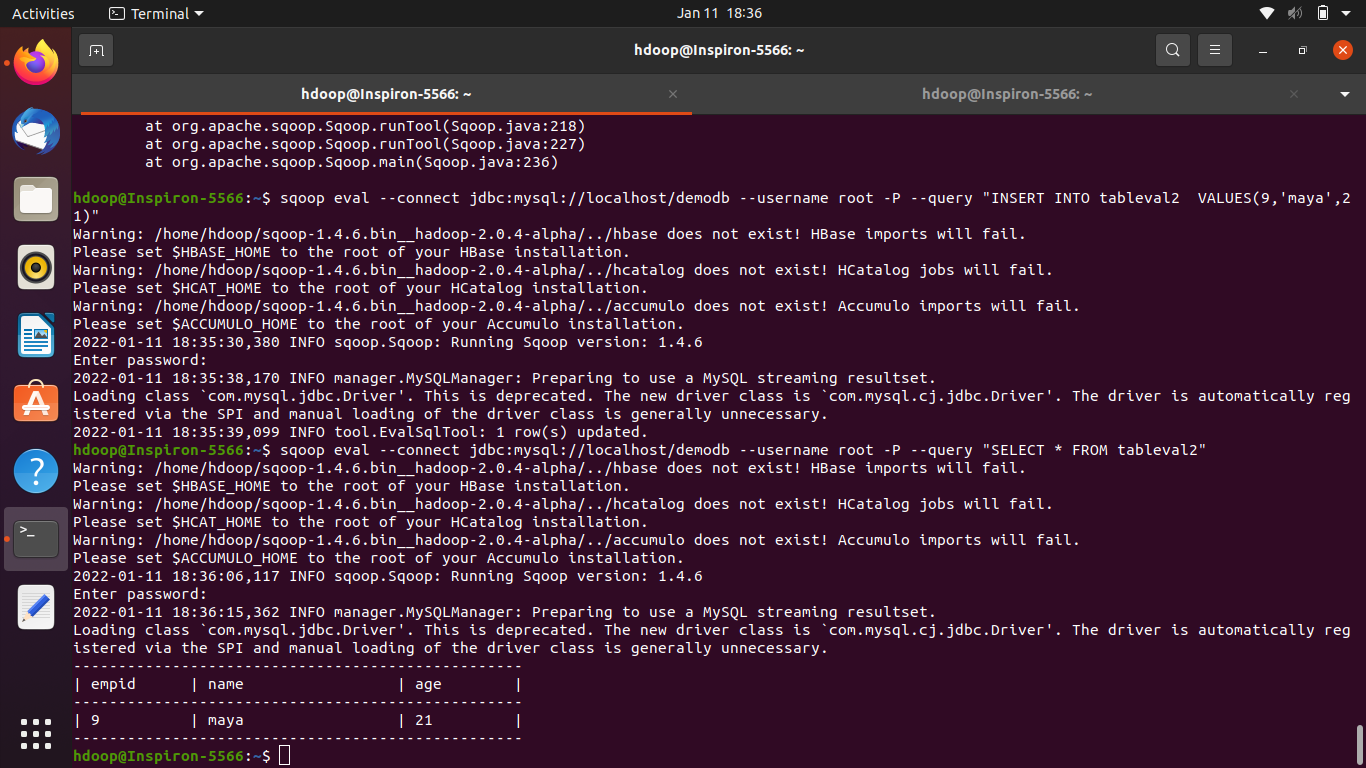
$ sqoop list-databases --connect jdbc:mysql://localhost/demodb --username root -P

$ sqoop eval --connect jdbc:mysql://localhost/demodb --username root -P --query "CREATE TABLE tableval2(empid INT PRIMARY KEY,name varchar(20),age INT)"

sqoop eval --connect jdbc:mysql://localhost/demodb --username root -P --query "INSERT INTO tableval2 VALUES(9,'maya',21)"

$ sqoop eval --connect jdbc:mysql://localhost/demodb --username root -P --query "SELECT

\* FROM tableval2"



# --incremental lastmodified

$ sqoop import \

> --connect jdbc:mysql://localhost/demodb \

> --username root \

> --password Vidya@60 \

> --table Sales \

> --split-by Year \

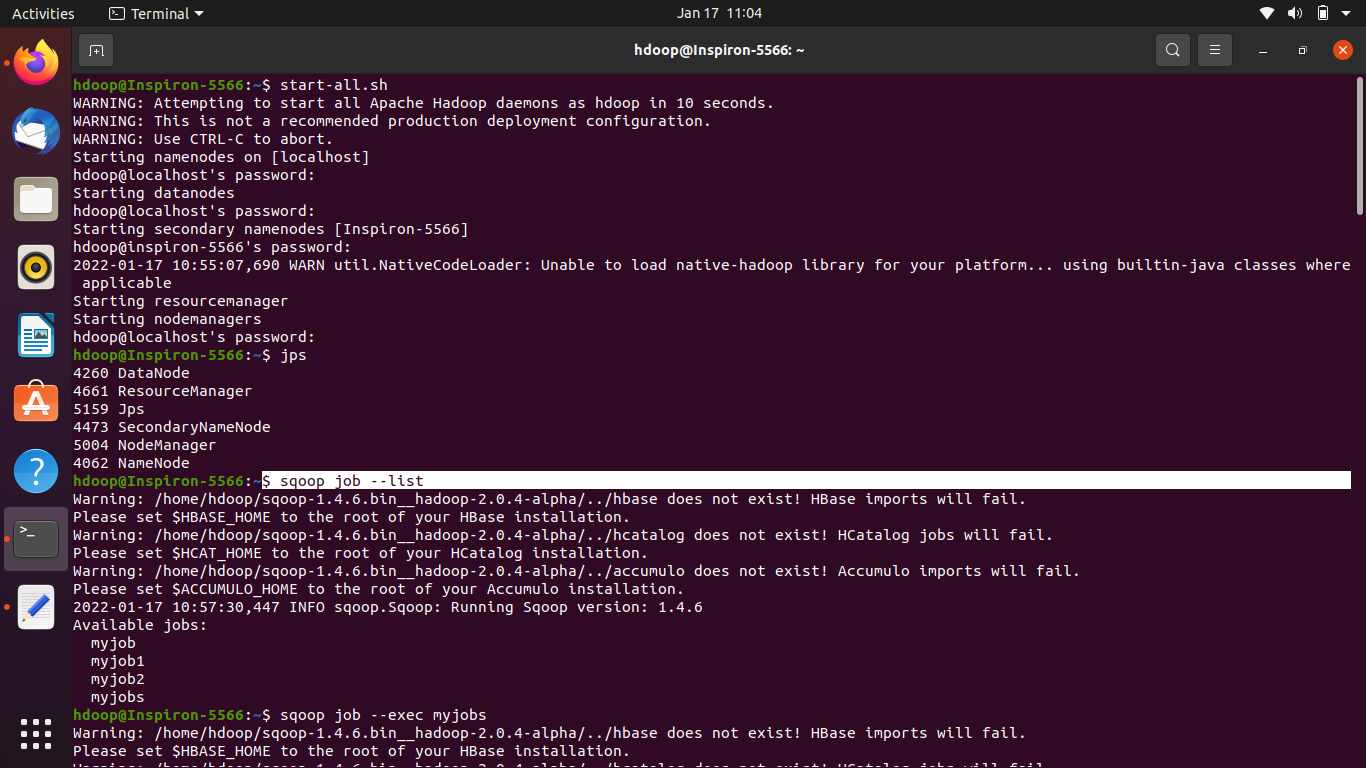
> --target-dir /user/sqoop/orders3 \

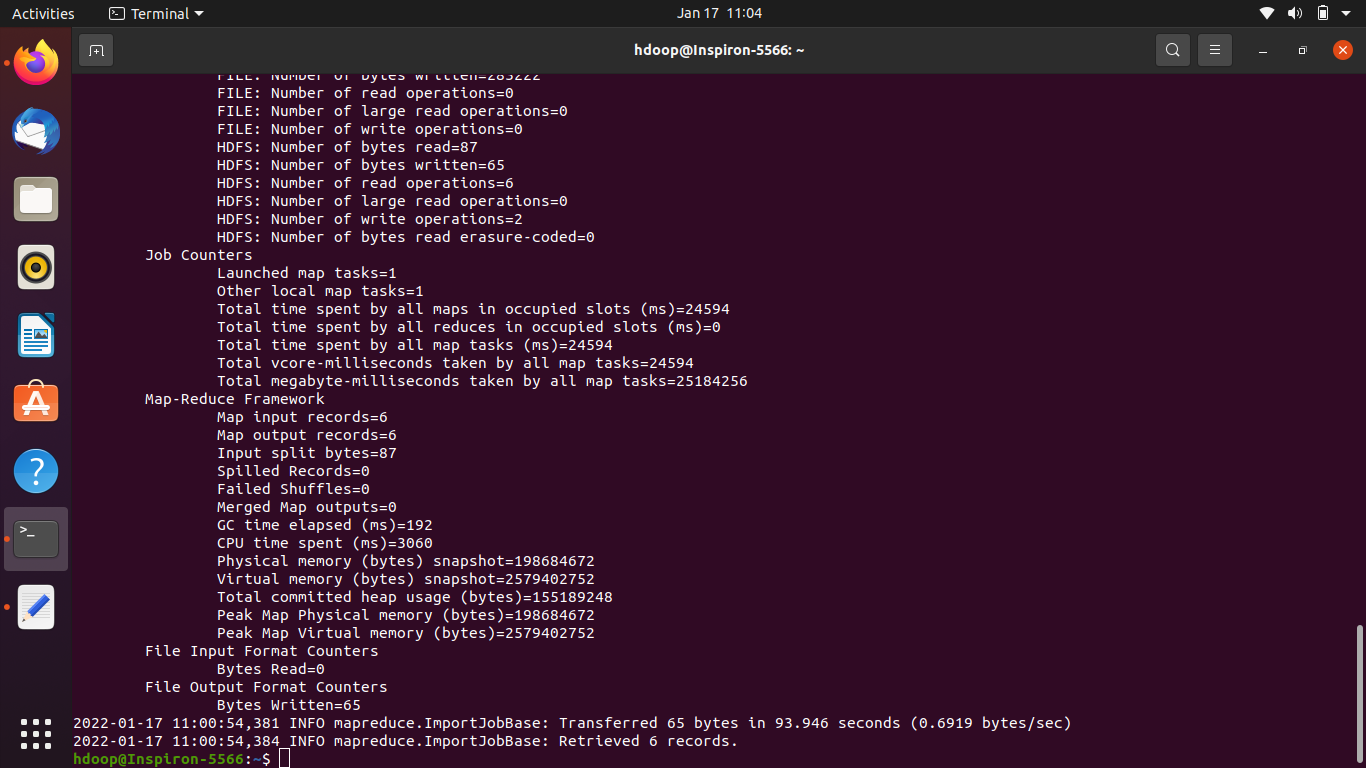
> --as-textfile

$ sqoop job --create myjob -- import --connect "jdbc:mysql://localhost/demodb" --username root --password Vidya@06 --table my\_view -m 1 --taget-dir /sqoop/job/

$ sqoop job --list

$ sqoop job --exec myjobs





hdfs dfs -ls -R /user/sqoop/orders3

2022-01-13 12:13:27,822 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

-rw-r--r-- 1 hdoop supergroup 0 2022-01-13 12:09 /user/sqoop/orders3/\_SUCCESS

-rw-r--r-- 1 hdoop supergroup 98 2022-01-13 12:09 /user/sqoop/orders3/part-m-00000

-rw-r--r-- 1 hdoop supergroup 101 2022-01-13 12:09 /user/sqoop/orders3/part-m-00001

-rw-r--r-- 1 hdoop supergroup 88 2022-01-13 12:09 /user/sqoop/orders3/part-m-00002

export FLUME\_HOME=/home/hdoop/apache-flume-1.9.0-bin

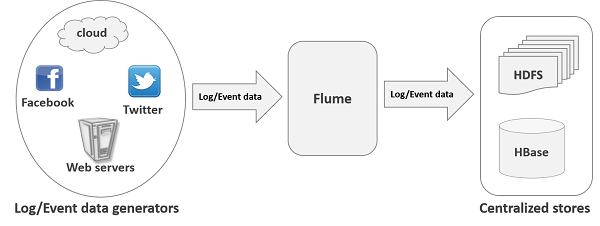
export PATH=$PATH:$FLUME\_HOME/bin

export CLASSPATH=$CLASSPATH:/FLUME\_HOME/lib/\*

## **What is Flume?**

Apache Flume is a tool/service/data ingestion mechanism for collecting, aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.

Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



## **Applications of Flume**

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data into Hadoop for analysis. Here, Apache Flume comes to our rescue.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

## **Advantages of Flume**

Here are the advantages of using Flume −

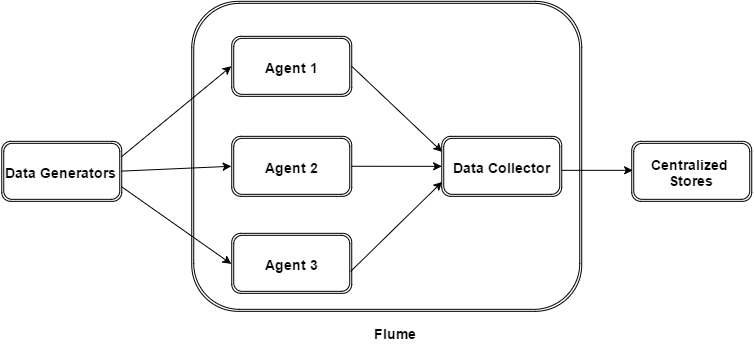
* Using Apache Flume we can store the data into any of the centralized stores (HBase, HDFS).
* When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
* Flume provides the feature of **contextual routing**.
* The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
* Flume is reliable, fault tolerant, scalable, manageable, and customizable.

## **Features of Flume**

Some of the notable features of Flume are as follows −

* Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.
* Using Flume, we can get the data from multiple servers immediately into Hadoop.
* Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.
* Flume supports a large set of sources and destination types.
* Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
* Flume can be scaled horizontally.

The following illustration depicts the basic architecture of Flume. As shown in the illustration, **data generators** (such as Facebook, Twitter) generate data which gets collected by individual Flume **agents** running on them. Thereafter, a **data collector** (which is also an agent) collects the data from the agents which is aggregated and pushed into a centralized store such as HDFS or HBase.



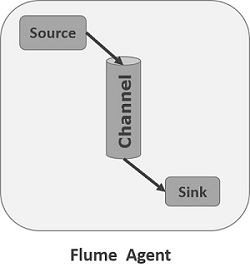
## **Flume Event**

An **event** is the basic unit of the data transported inside **Flume**. It contains a payload of byte array that is to be transported from the source to the destination accompanied by optional headers. A typical Flume event would have the following structure −



## **Flume Agent**

An **agent** is an independent daemon process (JVM) in Flume. It receives the data (events) from clients or other agents and forwards it to its next destination (sink or agent). Flume may have more than one agent. Following diagram represents a **Flume Agent**

****

As shown in the diagram a Flume Agent contains three main components namely, **source**, **channel**, and **sink**.

### **Source**

A **source** is the component of an Agent which receives data from the data generators and transfers it to one or more channels in the form of Flume events.

Apache Flume supports several types of sources and each source receives events from a specified data generator.

**Example** − Avro source, Thrift source, twitter 1% source etc.

### **Channel**

A **channel** is a transient store which receives the events from the source and buffers them till they are consumed by sinks. It acts as a bridge between the sources and the sinks.

These channels are fully transactional and they can work with any number of sources and sinks.

**Example** − JDBC channel, File system channel, Memory channel, etc.

### **Sink**

A **sink** stores the data into centralized stores like HBase and HDFS. It consumes the data (events) from the channels and delivers it to the destination. The destination of the sink might be another agent or the central stores.

**Example** − HDFS sink

**Note** − A flume agent can have multiple sources, sinks and channels. We have listed all the supported sources, sinks, channels in the Flume configuration chapter of this tutorial.

## **Additional Components of Flume Agent**

What we have discussed above are the primitive components of the agent. In addition to this, we have a few more components that play a vital role in transferring the events from the data generator to the centralized stores.

### **Interceptors**

Interceptors are used to alter/inspect flume events which are transferred between source and channel.

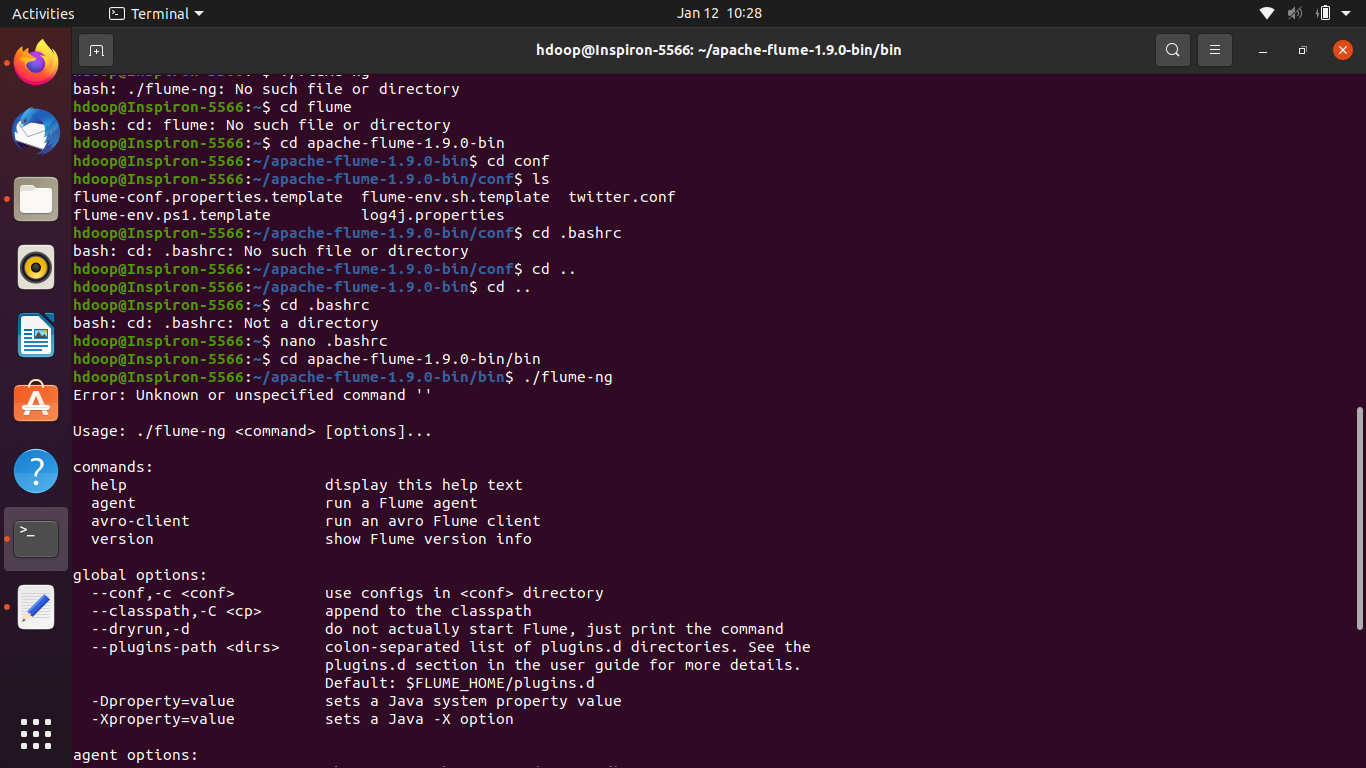
### **Channel Selectors**

These are used to determine which channel is to be opted to transfer the data in case of multiple channels. There are two types of channel selectors −

* **Default channel selectors** − These are also known as replicating channel selectors; they replicate all the events in each channel.
* **Multiplexing channel selectors** − These decide the channel to send an event based on the address in the header of that event.

### **Sink Processors**

These are used to invoke a particular sink from the selected group of sinks. These are used to create failover paths for your sinks or load balance events across multiple sinks from a channel.



sqoop job --create myjob -- import --connect "jdbc:mysql://localhost/demodb" --username root --password Vidya@06 --table my\_view -m 1 --taget-dir /sqoop/job/

* **Create twitter.conf file and copy below conf in flume/conf folder**

TwitterAgent.sources = Twitter

TwitterAgent.channels = MemChannel

TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource

TwitterAgent.sources.Twitter.channels = MemChannel

TwitterAgent.sources.Twitter.consumerKey = hfMViwc1oE7NIyvuOi6jgEQU6

TwitterAgent.sources.Twitter.consumerSecret = IuLS7LOC0ZhPrYm21GMOJMr7pITAs3104KgFVVGqALIN284A6j

TwitterAgent.sources.Twitter.accessToken= 963790698402103297-CpwHth34oZXG1ViY9M75ijuixko7QoC

TwitterAgent.sources.Twitter.accessTokenSecret = EuJoVH4YpVTAIYzNgTfuMwdoD6VqsUKS6zOHDMD1zpKLw

TwitterAgent.sources.Twitter.keywords = bigdata , hive, mysql, database, oracle

TwitterAgent.sinks.HDFS.channel = MemChannel

TwitterAgent.sinks.HDFS.type = hdfs

TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/File/Twit

TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream

TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text

TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000

TwitterAgent.sinks.HDFS.hdfs.rollSize = 0

TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.channels.MemChannel.type = memory

TwitterAgent.channels.MemChannel.capacity = 10000

TwitterAgent.channels.MemChannel.transactionCapacity = 1000

* https://stackoverflow.com/questions/49060244/exception-org-apache-hadoop-ipc-rpcexception-rpc-response-exceeds-maximum-da

Open hdfs-site.xml and paste this:

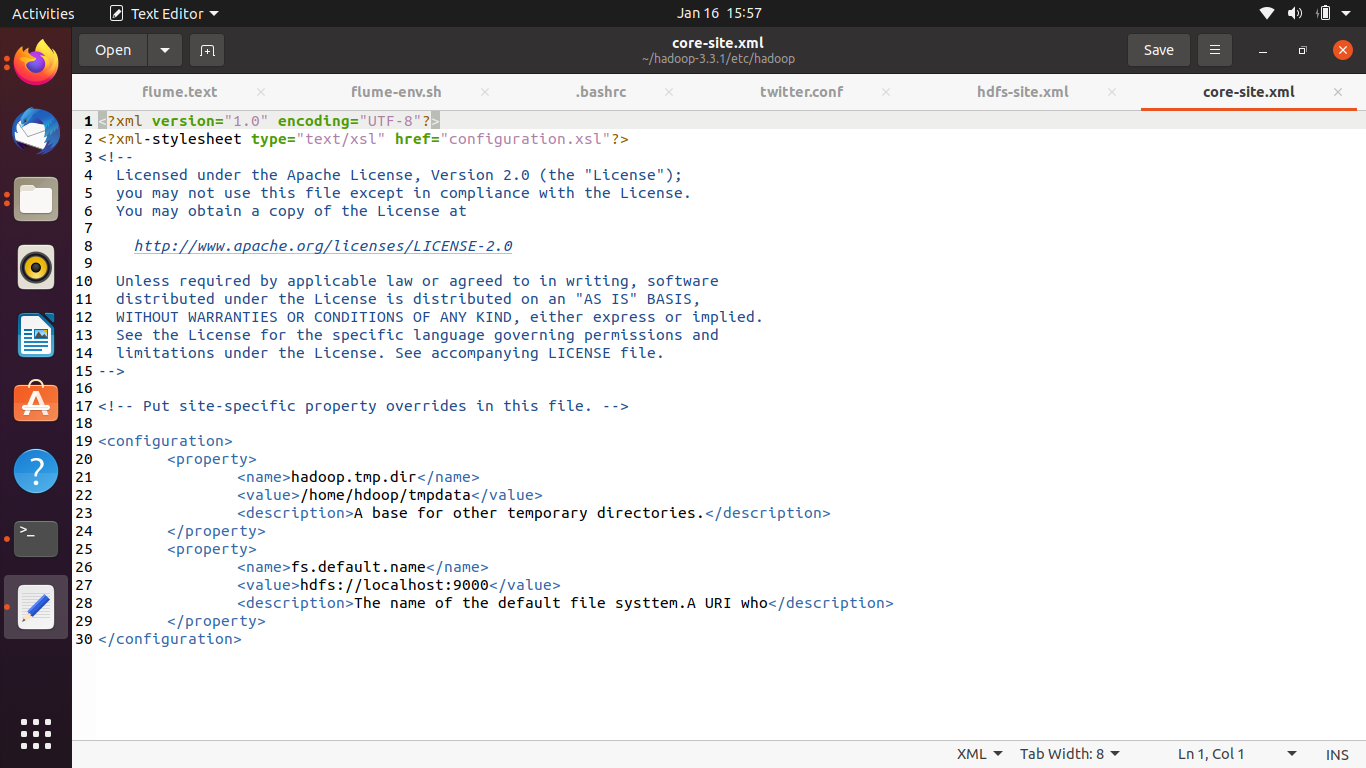
<property>

<name>ipc.maximum.data.length</name>

<value>134217728</value>

</property>

Open core-site.xml



Check guava-27.0-jre.jar is available in flume/lib folder if lesser version is available download his jar file from the below link

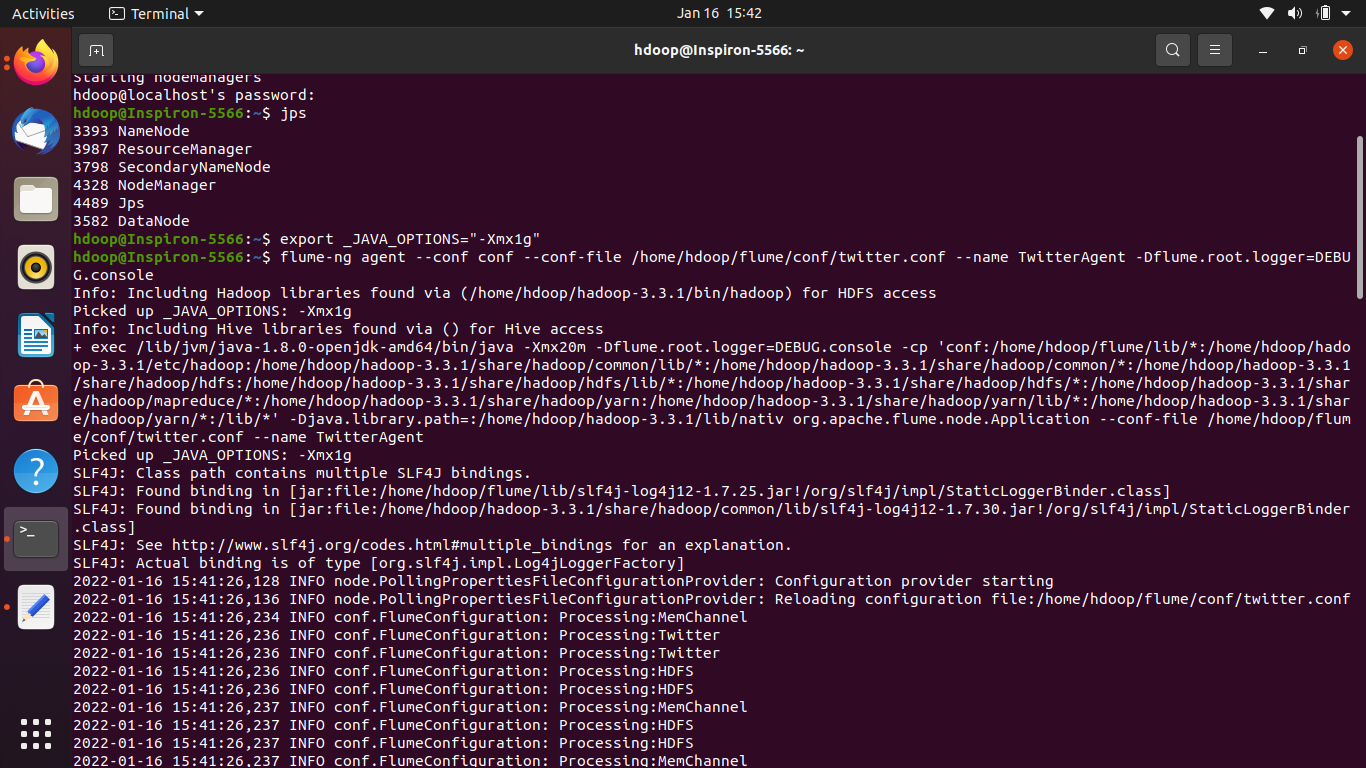
<https://www.google.com/url?q=https://jar-download.com/artifacts/com.google.guava/guava/27.0-jre/source-code&sa=D&source=hangouts&ust=1642414483554000&usg=AOvVaw3mClATmz7TO28RyZSqEq8l>

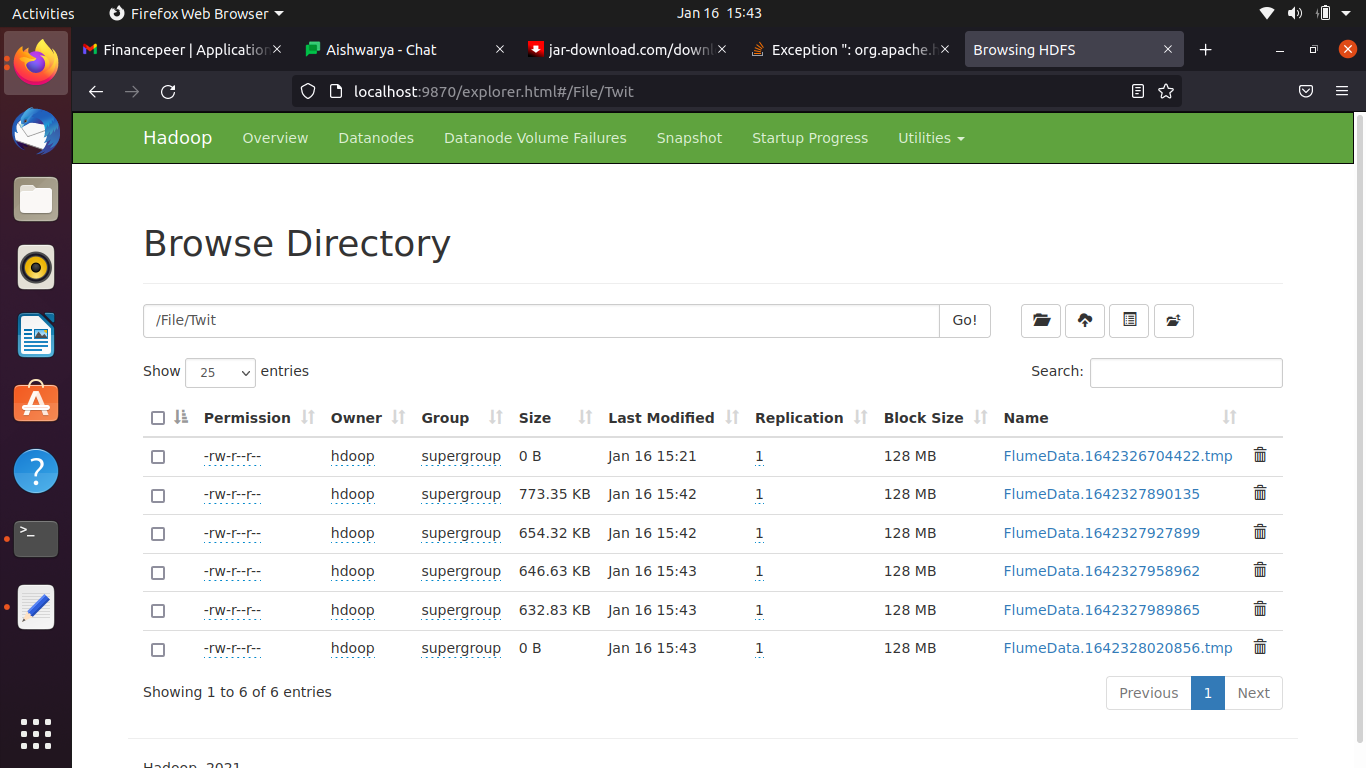
<https://jar-download.com/download-handling.php>

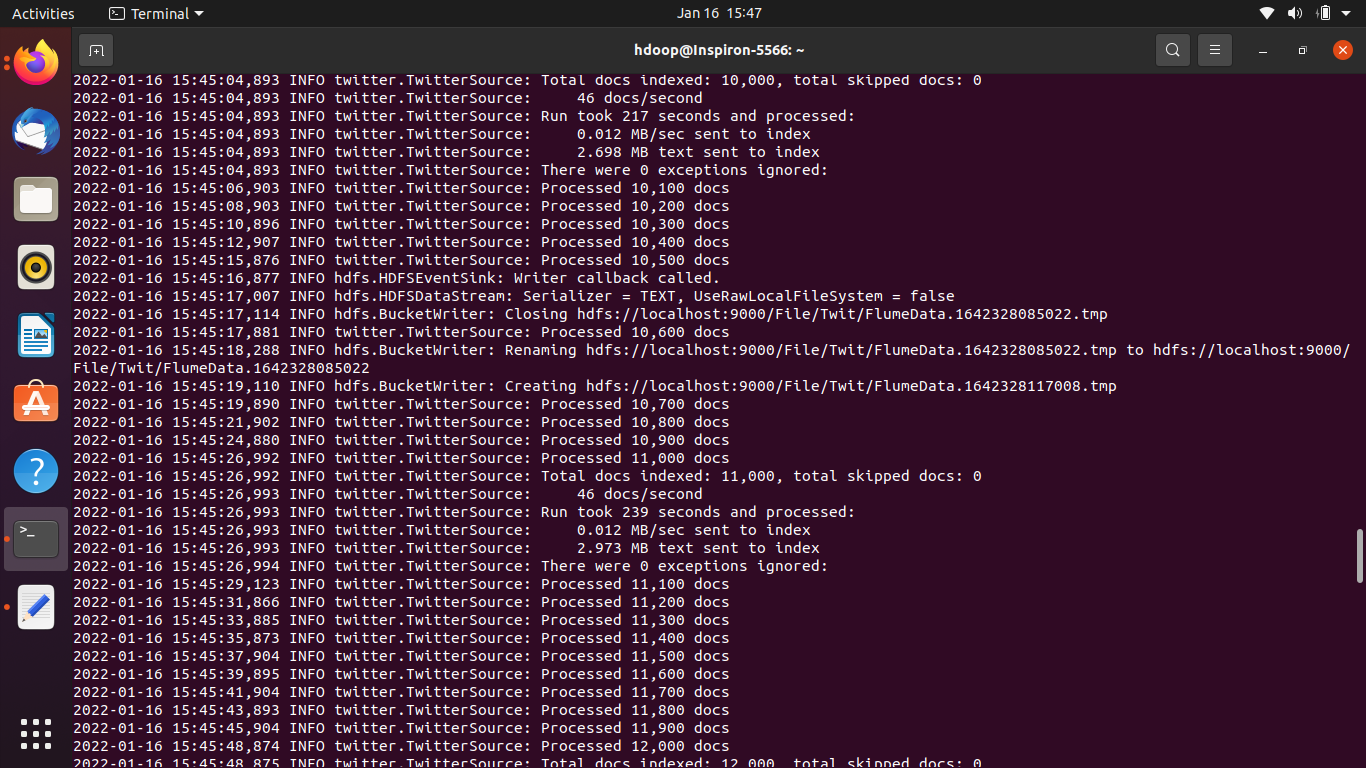
Type flume-ng version to check whether flume is downloaded or not on the terminal

$ export \_JAVA\_OPTIONS="-Xmx1g"

$ flume-ng agent --conf conf --conf-file /home/hdoop/flume/conf/twitter.conf --name TwitterAgent -Dflume.root.logger=DEBUG.console







Write belo pythoncode

import csv

import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key

CSV\_URL = 'https://www.alphavantage.co/query?function=TIME\_SERIES\_INTRADAY\_EXTENDED&symbol=IBM&interval=15min&slice=year1month1&apikey=4X0A1IGHNXUX93SO'

with requests.Session() as s:

download = s.get(CSV\_URL)

decoded\_content = download.content.decode('utf-8')

cr = csv.reader(decoded\_content.splitlines(), delimiter=',')

my\_list = list(cr)

for row in my\_list:

print(row)

Create conf file for stockdata and paste this

agent1.sources = tail

agent1.channels = Channel-2

agent1.sinks = sink-1

agent1.sources.tail.type = exec

agent1.sources.tail.command = python3 /home/hdoop/stock\_data.py

agent1.sources.tail.channels = Channel-2

agent1.sinks.sink-1.channel = Channel-2

agent1.sinks.sink-1.type = hdfs

agent1.sinks.sink-1.hdfs.path = hdfs://localhost:9000/File/stock

agent1.sinks.sink-1.hdfs.fileType = DataStream

agent1.sinks.sink-1.hdfs.rollInterval = 60

agent1.sinks.sink-1.hdfs.rollSize = 0

agent1.sinks.sink-1.hdfs.rollCount = 0

agent1.channels.Channel-2.type = memory

# 