HADOOP

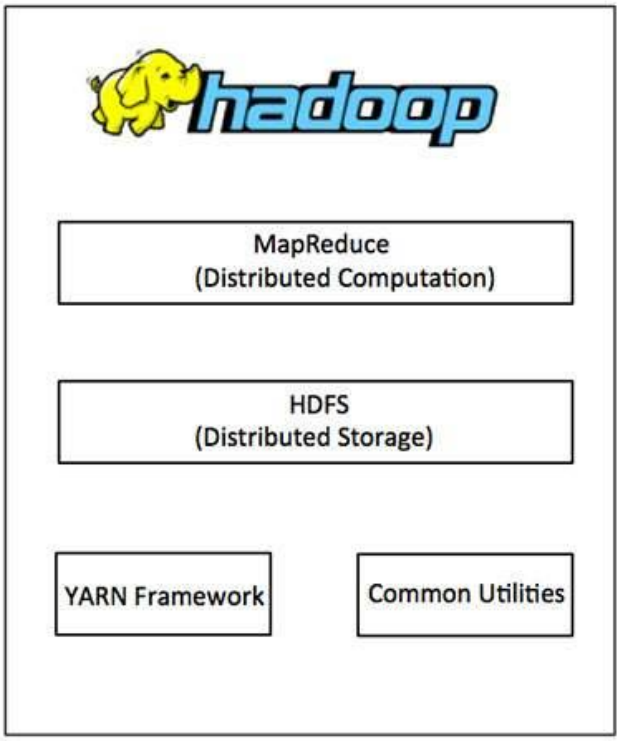
INTRODUCTION TO HADOOP

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

HADOOP ARCHITECTURE

At its core, Hadoop has two major layers namely −

- Processing/Computation layer (MapReduce), and
- Storage layer (Hadoop Distributed File System).



MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules −

- **Hadoop Common** − These are Java libraries and utilities required by other Hadoop modules.
- **Hadoop YARN** − This is a framework for job scheduling and cluster resource management.

HADOOP INSTALLATION ON WINDOWS 10

Prerequisite:

To install Hadoop, you should have Java version 1.8 in your system. Check your java version through this command on command prompt

Command Prompt

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hp>java -version
java version "1.8.0_152"
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)

C:\Users\hp>
```

- If java is not installed in your system, then –
- Go this link –

  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downl...
- Accept the license,

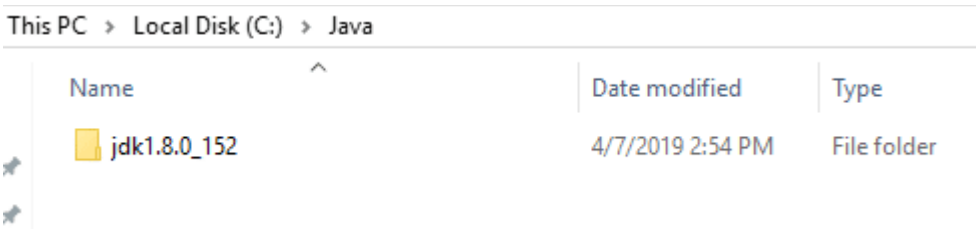
Java SE Development Kit 8u201 Demos and Samples Downloads

You must accept the Oracle BSD License. to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	9.05 MB	jdk-8u201-linux-arm32-vfp-hflt-demos.tar.gz
Linux ARM 64 Hard Float ABI	9.06 MB	jdk-8u201-linux-arm64-vfp-hflt-demos.tar.gz
Linux x86	56.13 MB	jdk-8u201-linux-i586-demos.rpm
Linux x86	55.98 MB	jdk-8u201-linux-i586-demos.tar.gz
Linux x64	56.23 MB	jdk-8u201-linux-x64-demos.rpm
Linux x64	56.08 MB	jdk-8u201-linux-x64-demos.tar.gz
Mac OS X	56.25 MB	jdk-8u201-macosx-x86_64-demos.zip
Solaris SPARC 64-bit	12.2 MB	jdk-8u201-solaris-sparcv9-demos.tar.Z
Solaris SPARC 64-bit	8.46 MB	jdk-8u201-solaris-sparcv9-demos.tar.gz
Solaris x64	12.19 MB	jdk-8u201-solaris-x64-demos.tar.Z
Solaris x64	8.42 MB	jdk-8u201-solaris-x64-demos.tar.gz
Windows x86	56.96 MB	jdk-8u201-windows-i586-demos.zip
Windows x64	56.98 MB	jdk-8u201-windows-x64-demos.zip

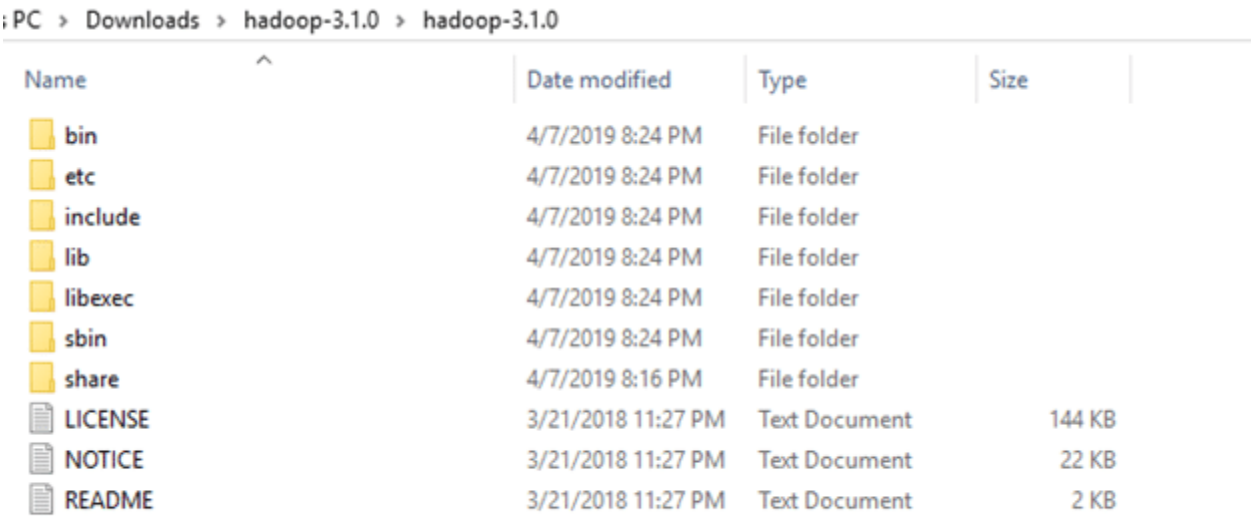
- Download the file according to your operating system. Keep the java folder directly under the local disk directory (C:\Java\jdk1.8.0_152) rather than in Program Files (C:\Program Files\Java\jdk1.8.0_152) as it can create errors afterwards.



- After downloading java version 1.8, download hadoop version 3.1 from this link –

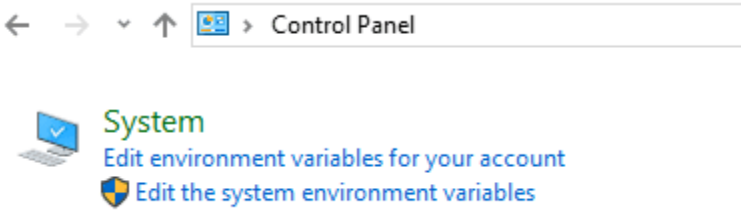
  https://archive.apache.org/dist/hadoop/common/hadoop-3.1.0/hadoop-3...

- Extract it to a folder.

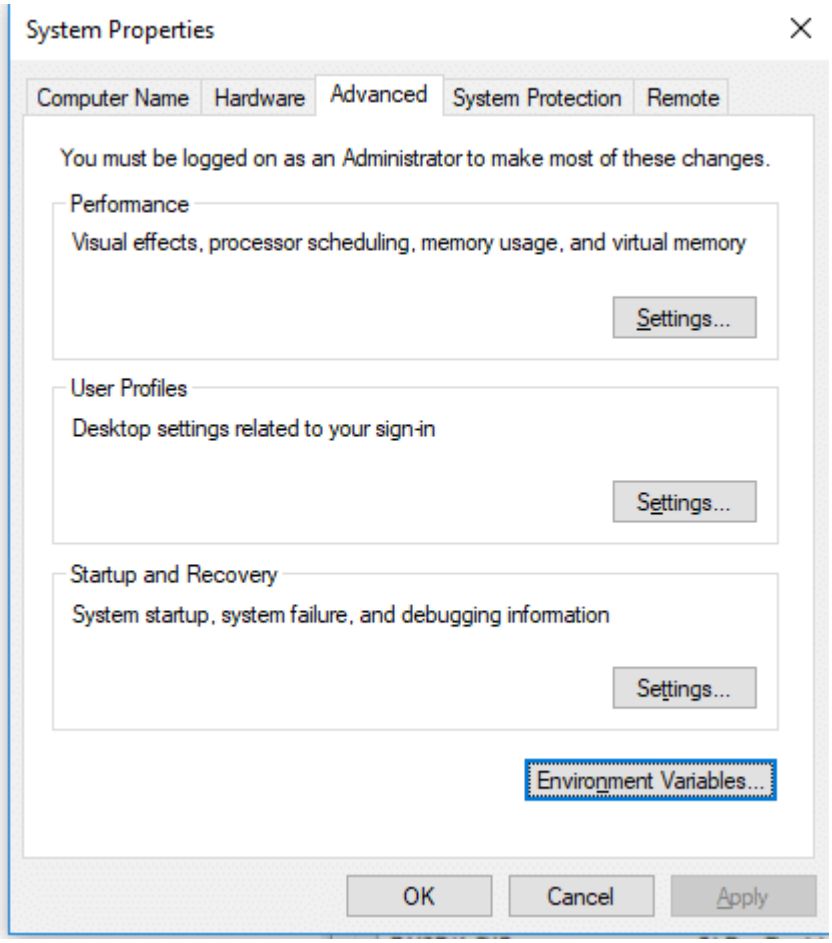


Setup System Environment Variables

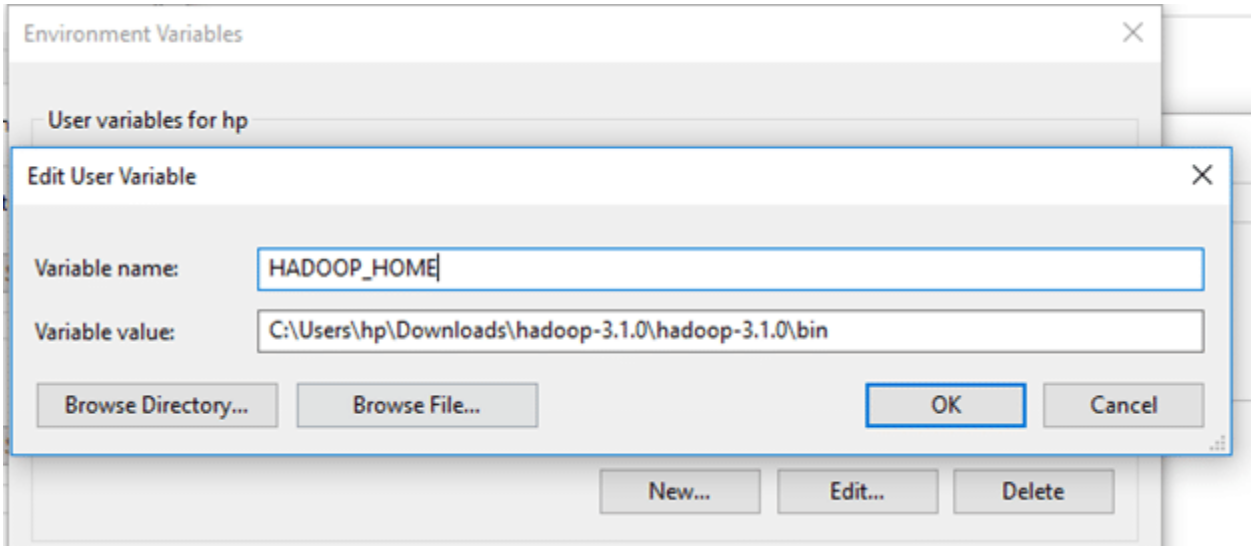
Open control panel to edit the system environment variable



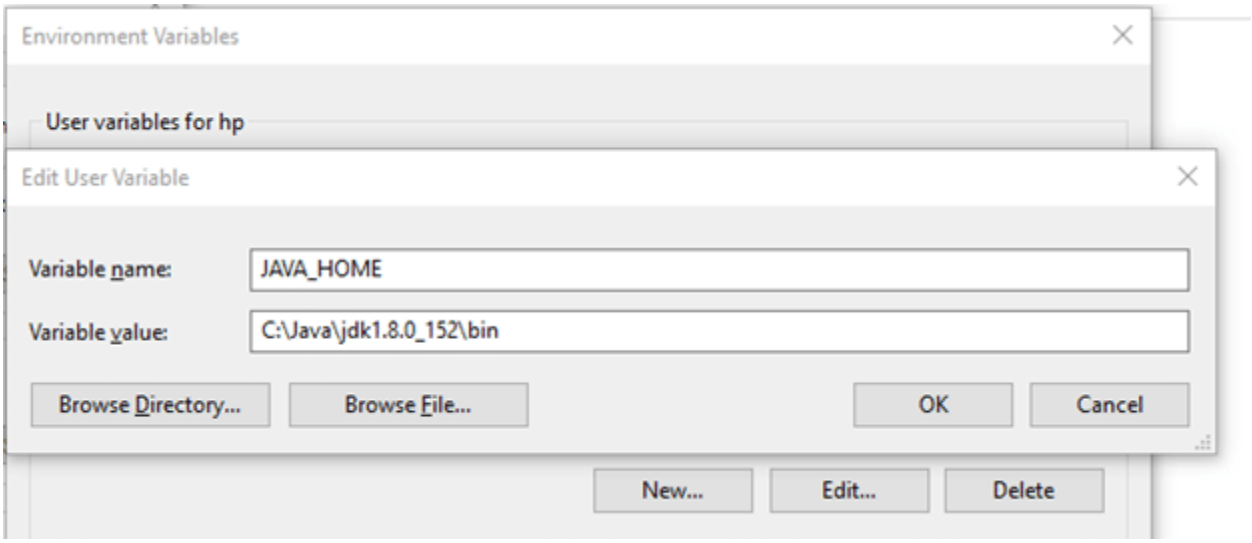
Go to environment variable in system properties



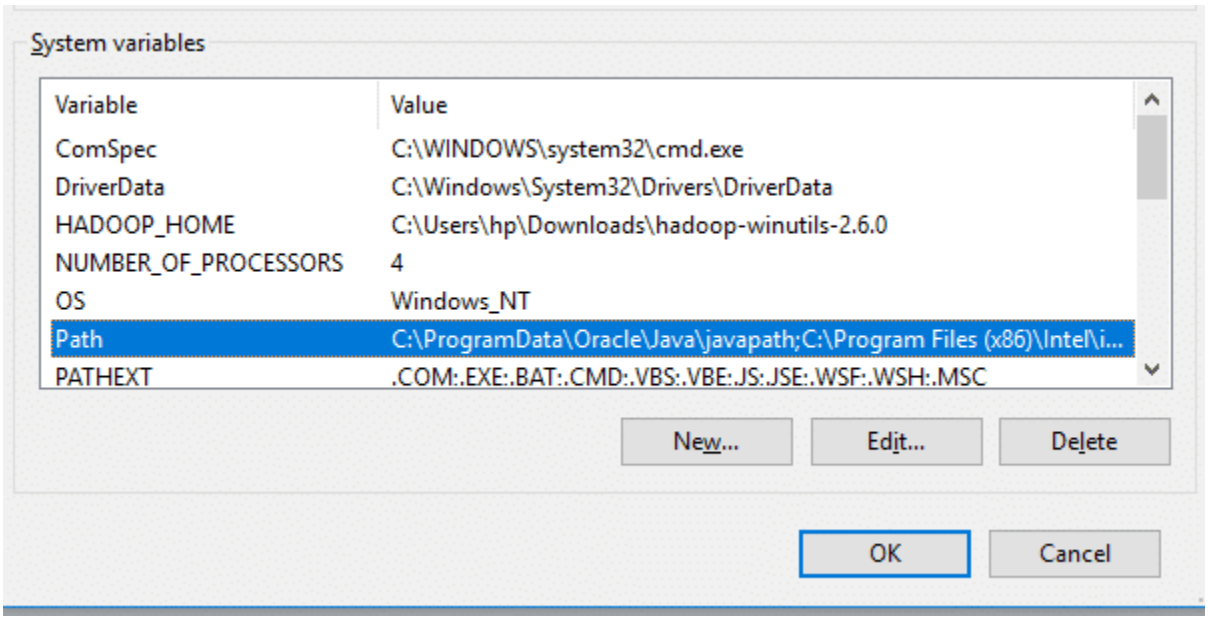
- Create a new user variable. Put the Variable_name as HADOOP_HOME and Variable_value as the path of the bin folder where you extracted hadoop.



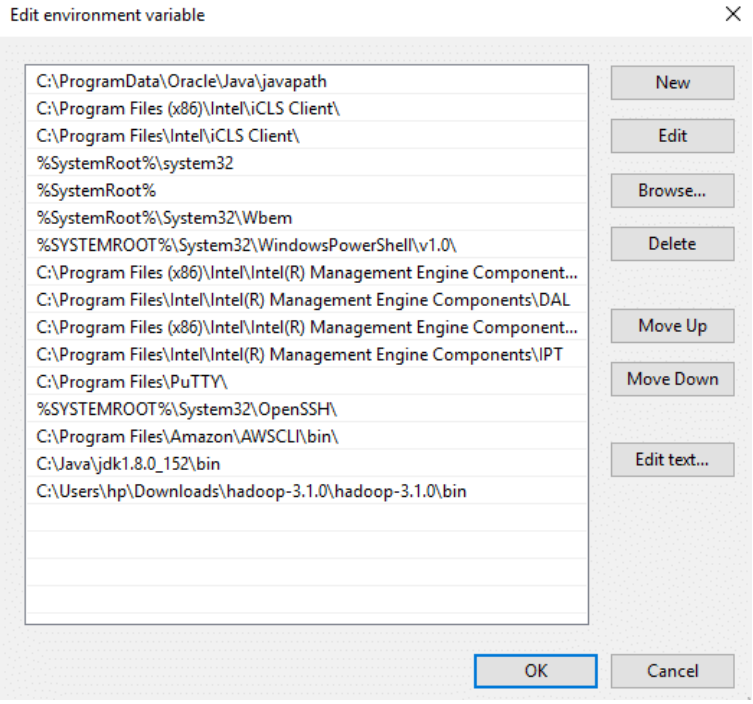
- Likewise, create a new user variable with variable name as JAVA_HOME and variable value as the path of the bin folder in the Java directory.



- Now we need to set Hadoop bin directory and Java bin directory path in system variable path.
- Edit Path in system variable



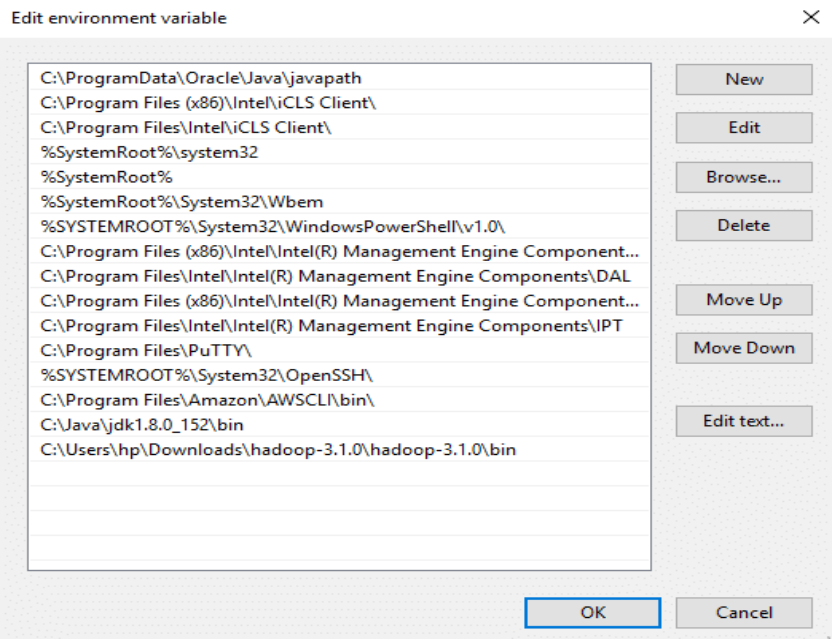
Click on New and add the bin directory path of Hadoop and Java in it.



Configurations

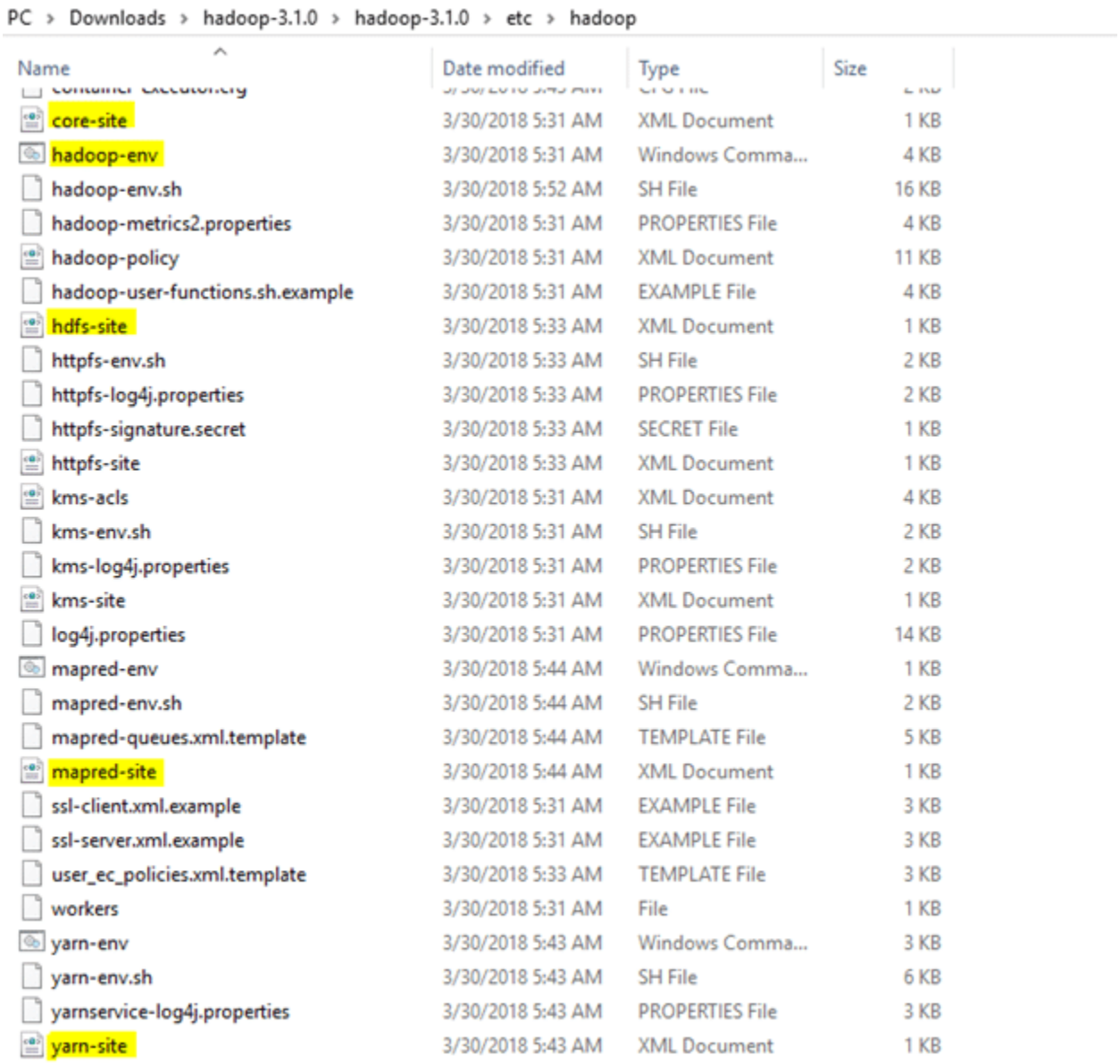
Now we need to edit some files located in the hadoop directory of the etc folder where we installed hadoop. The files that need to be edited have been highlighted.

Click on New and add the bin directory path of Hadoop and Java in it.



Configurations

Now we need to edit some files located in the hadoop directory of the etc folder where we installed hadoop. The files that need to be edited have been highlighted.



1. Edit the file core-site.xml in the hadoop directory. Copy this xml property in the configuration in the file

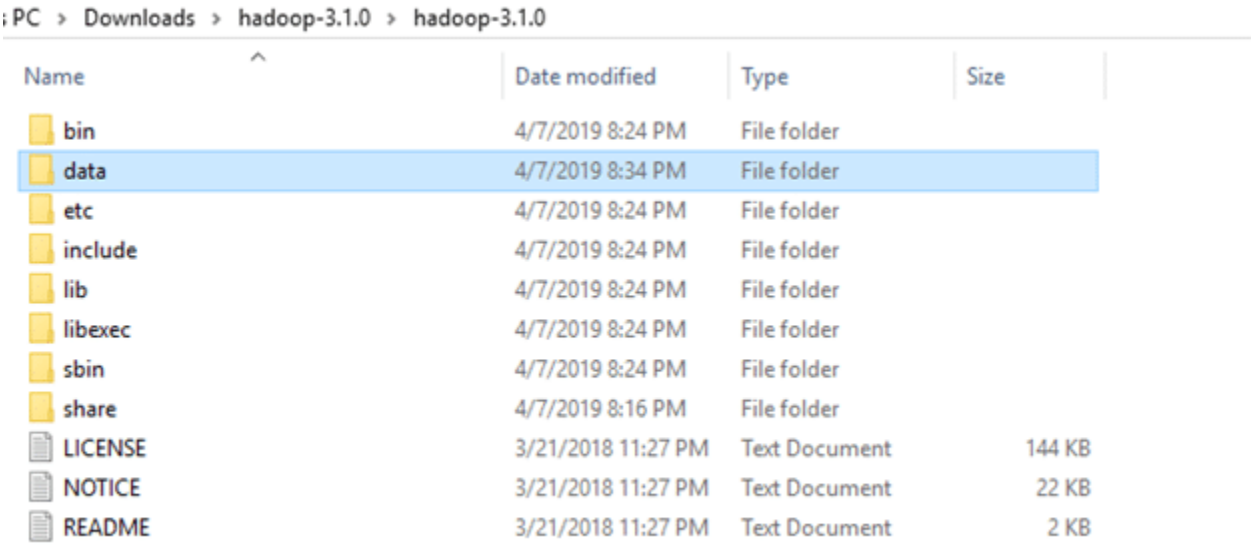
/span>configuration>
  /span>property>
    /span>name>fs.defaultFS/span>/name>
    /span>value>hdfs://localhost:9000</value>

/span>/property>
/span>/configuration>

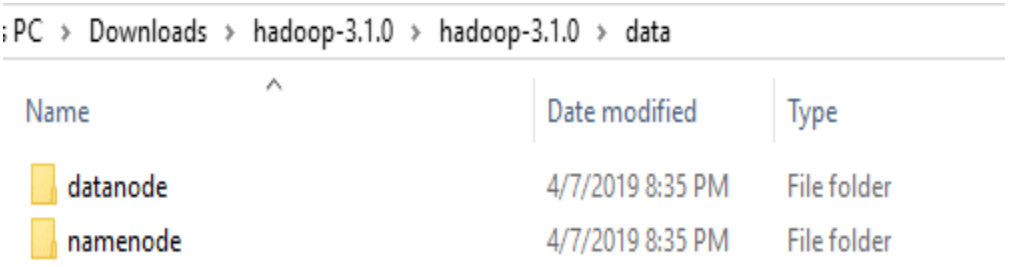
2. Edit mapred-site.xml and copy this property in the cofiguration

/span>configuration>
/span>property>
/span>name>mapreduce.framework.name/span>/name>
/span>value>yarn/span>/value>
/span>/property>
/span>/configuration>

3. Create a folder ‘data’ in the hadoop directory



Create a folder with the name ‘datanode’ and a folder ‘namenode’ in this data directory



4. Edit the file hdfs-site.xml and add below property in the configuration

Note: The path of namenode and datanode across value would be the path of the datanode and namenode folders you just created.

```
/span>configuration>
  /span>property>
    /span>name>dfs.replication/span>/name>
    /span>value>1/span>/value>
  /span>/property>
  /span>property>
    /span>name>dfs.namenode.name.dir/span>/name>
    /span>value>C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-
3.1.0\data\namenode/span>/value>
  /span>/property>
  /span>property>
    /span>name>dfs.datanode.data.dir/span>/name>
    /span>value> C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-
3.1.0\data\datanode/span>/value>
  /span>/property>
/span>/configuration>
```

5. Edit the file yarn-site.xml and add below property in the configuration

```
/span>configuration>
  /span>property>
    /span>name>yarn.nodemanager.aux-services/span>/name>
    /span>value>mapreduce_shuffle/span>/value>
  /span>/property>
  /span>property>
    /span>name>yarn.nodemanager.auxservices.mapreduce.shuffle.class/span>/name>
/span>value>org.apache.hadoop.mapred.ShuffleHandler/span>/value>
  /span>/property>
/span>/configuration>
```

6. Edit hadoop-env.cmd and replace %JAVA_HOME% with the path of the java folder where your jdk 1.8 is installed

hadoop-env - Notepad

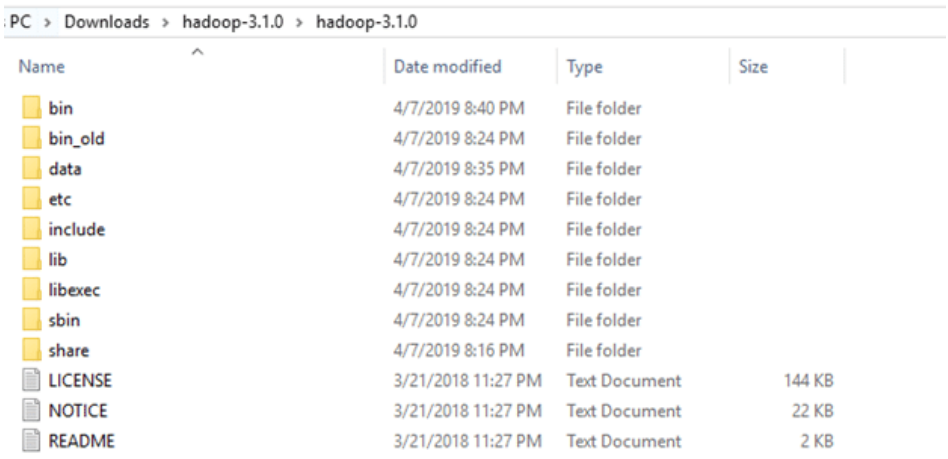
File Edit Format View Help

```
@rem Set Hadoop-specific environment variables here.

@rem The only required environment variable is JAVA_HOME.  All others are
@rem optional.  When running a distributed configuration it is best to
@rem set JAVA_HOME in this file, so that it is correctly defined on
@rem remote nodes.

@rem The java implementation to use.  Required.
set JAVA_HOME=C:\Java\jdk1.8.0_152

@rem The jsvc implementation to use. Jsvc is required to run secure datanodes.
@rem set JSVC_HOME=%JSVC_HOME%

@rem set HADOOP_CONF_DIR=

@rem Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
if exist %HADOOP_HOME%\contrib\capacity-scheduler (
  if not defined HADOOP_CLASSPATH (
    set HADOOP_CLASSPATH=%HADOOP_HOME%\contrib\capacity-scheduler\*.jar
  ) else (
    set HADOOP_CLASSPATH=%HADOOP_CLASSPATH%;%HADOOP_HOME%\contrib\capacity-scheduler\*.jar
  )
)
```

- Hadoop needs windows OS specific files which does not come with default download of hadoop.
- To include those files, replace the bin folder in hadoop directory with the bin folder provided in this github link.

https://github.com/s911415/apache-hadoop-3.1.0-winutils

- Download it as zip file. Extract it and copy the bin folder in it. If you want to save the old bin folder, rename it like bin_old and paste the copied bin folder in that directory.



- Check whether hadoop is successfully installed by running this command on cmd-

hadoop version

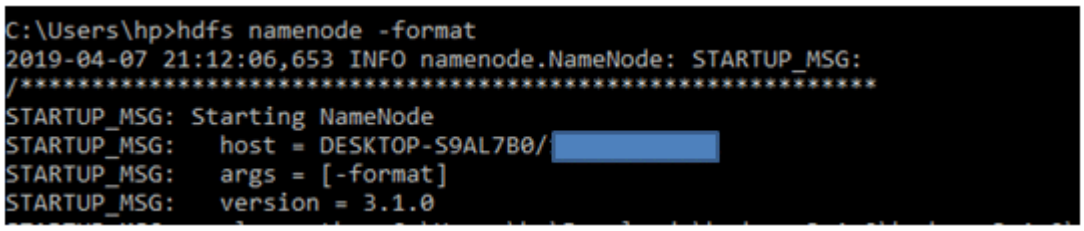
Since it doesn't throw error and successfully shows the hadoop version, that means hadoop is successfully installed in the system.

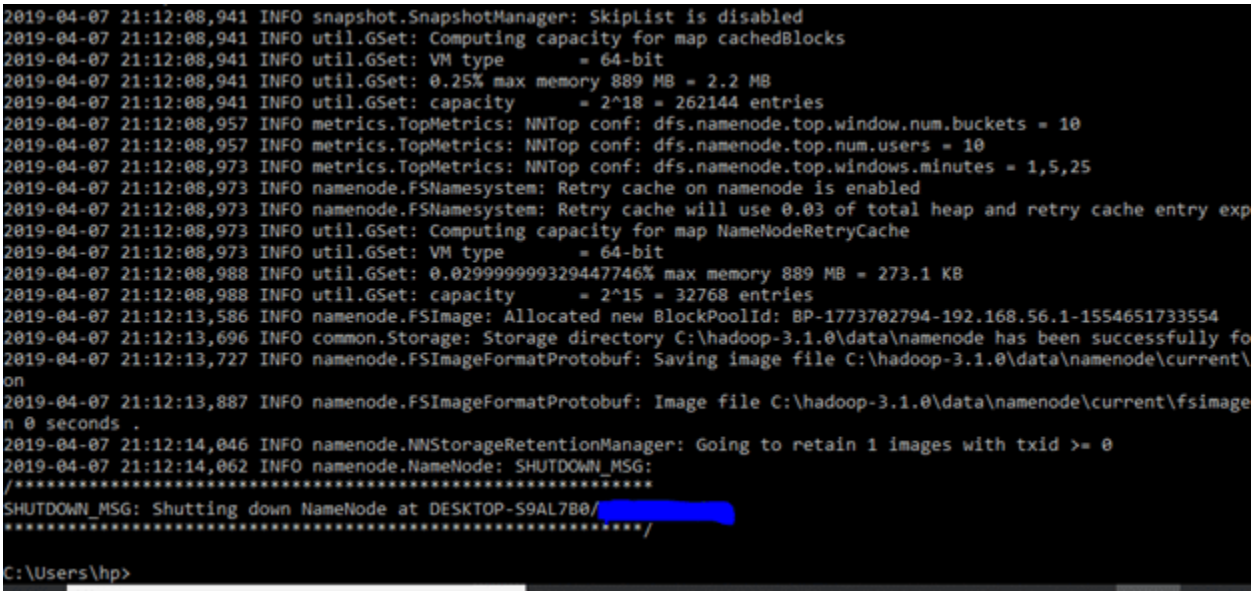
Format the NameNode

- Formatting the NameNode is done once when hadoop is installed and not for running hadoop filesystem, else it will delete all the data inside HDFS. Run this command-

hdfs namenode –format

- It would appear something like this –





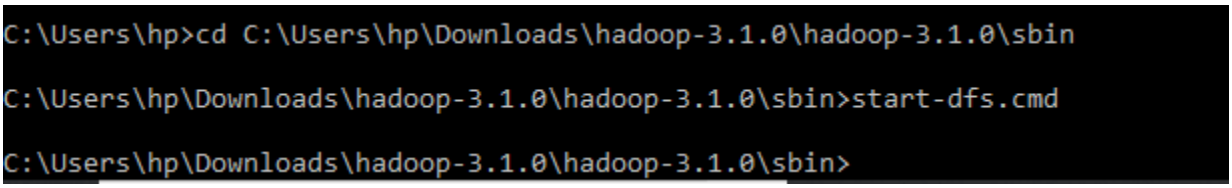
Now change the directory in cmd to sbin folder of hadoop directory with this command,

- (Note: Make sure you are writing the path as per your system)

  cd C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin

- Start namenode and datanode with this command –

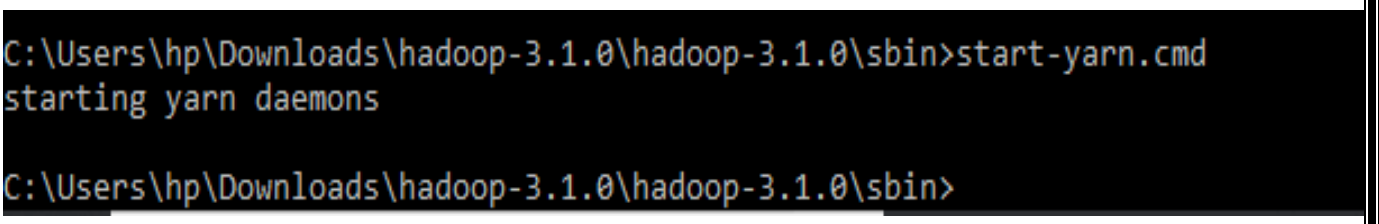
  start-dfs.cmd



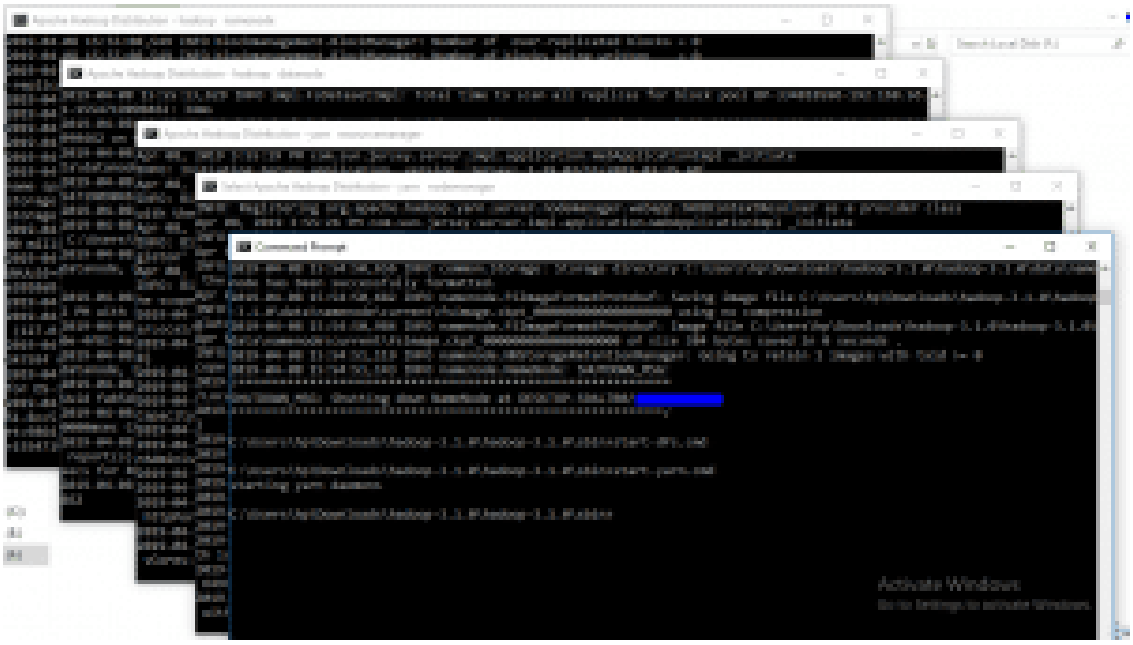
Two more cmd windows will open for NameNode and DataNode

- Now start yarn through this command-

  start-yarn.cmd

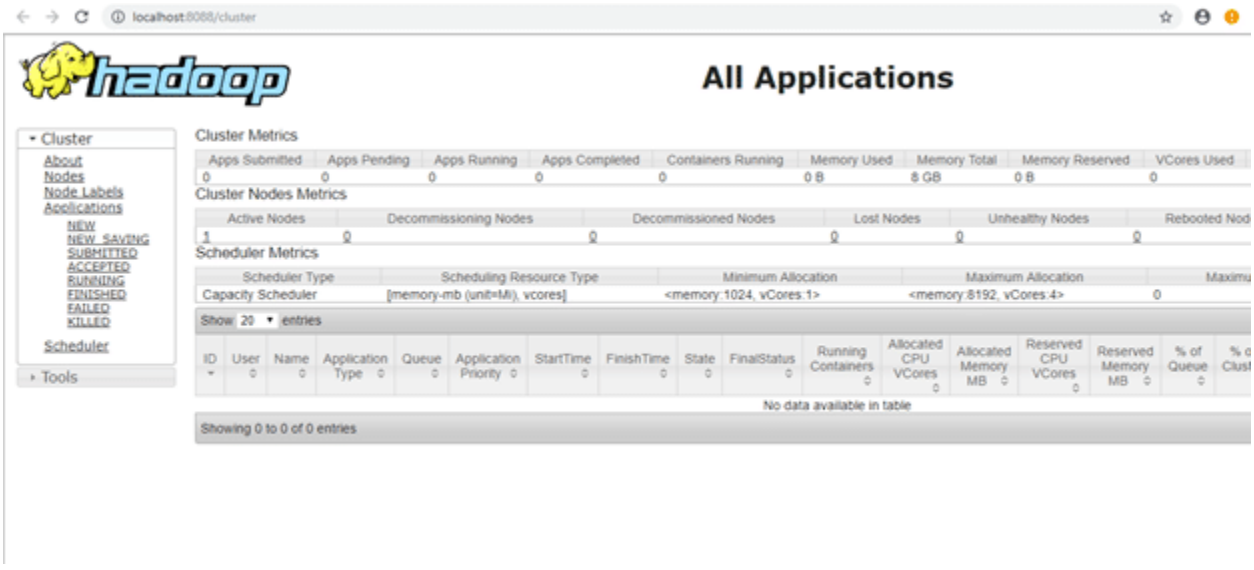


- Two more windows will open, one for yarn resource manager and one for yarn node manager.



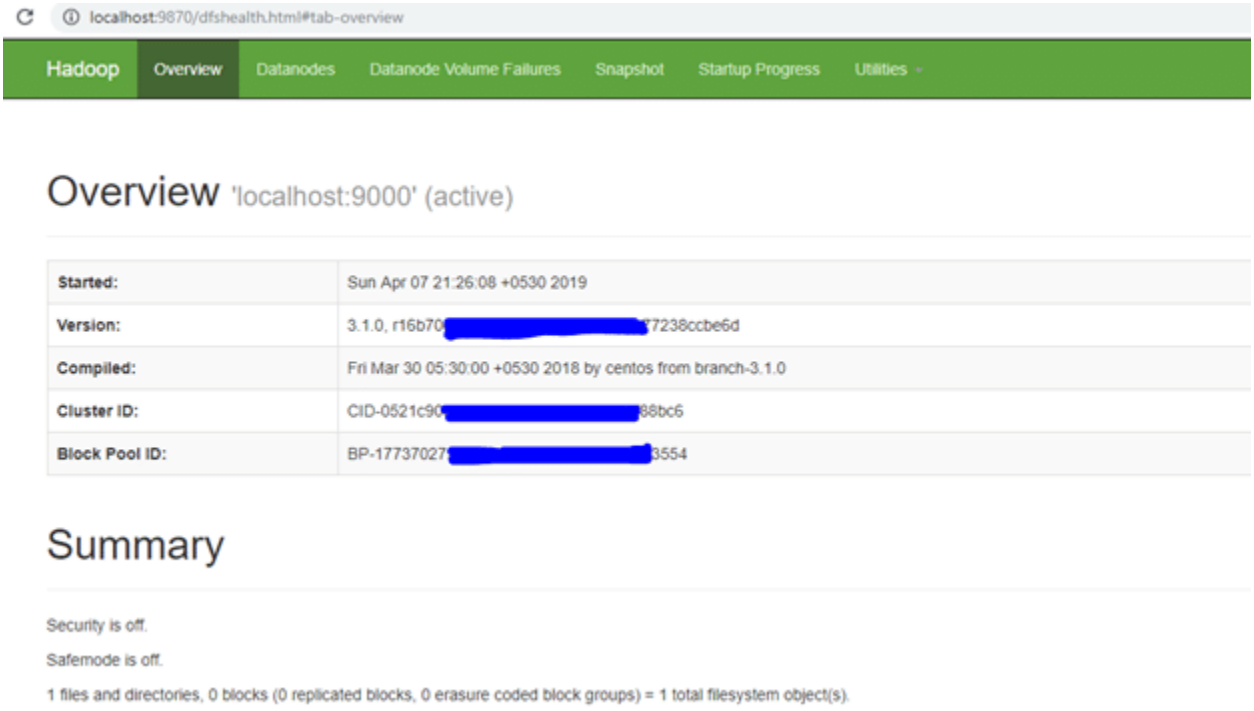
- Note: Make sure all the 4 Apache Hadoop Distribution windows are up n running. If they are not running, you will see an error or a shutdown message. In that case, you need to debug the error.
- To access information about resource manager current jobs, successful and failed jobs, go to this link in browser-

http://localhost:8088/cluster



- To check the details about the hdfs (namenode and datanode),
- Open this link on browser-

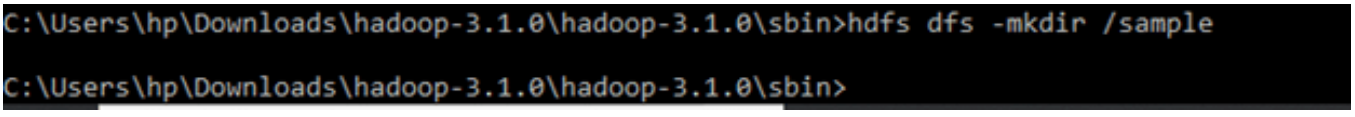
  http://localhost:9870/
- Note: If you are using Hadoop version prior to 3.0.0 – Alpha 1, then use port http://localhost:50070/



WORKING WITH HDFS

- I will be using a small text file in my local file system. To put it in hdfs using hdfs command line tool.
- I will create a directory named ‘sample’ in my hadoop directory using the following command-

hdfs dfs –mkdir /sample



- To verify if the directory is created in hdfs, we will use 'ls' command which will list the files present in hdfs –

hdfs dfs –ls /

```
C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - hp supergroup          0 2019-04-07 23:39 /sample

C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>
```

Then I will copy a text file named 'potatoes' from my local file system to this folder that I just created in hdfs using copyFromLocal command-

hdfs dfs -copyFromLocal C:\Users\hp\Downloads\potatoes.txt /sample

```
C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>hdfs dfs -copyFromLocal C:\Users\hp\Downloads\potatoes.txt /sampl

C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>
```

- To verify if the file is copied to the folder, I will use 'ls' command by specifying the folder name which will read the list of files in that folder –

hdfs dfs –ls /sample

```
C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>hdfs dfs -ls /sample
Found 1 items
-rw-r--r--   1 hp supergroup       3736 2019-04-07 23:39 /sample/potatoes.txt

C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>
```

To view the contents of the file we copied, I will use cat command- hdfs dfs cat /sample/potatoes.txt

```
C:\Users\hp\Downloads\hadoop-3.1.0\hadoop-3.1.0\sbin>hdfs dfs -cat /sample/potatoes.txt
area    temp    size    storage method  texture flavor  moistness
1       1       1       1       1       2.9     3.2     3.0
1       1       1       1       2       2.3     2.5     2.6
1       1       1       1       3       2.5     2.8     2.8
1       1       1       1       4       2.1     2.9     2.4
1       1       1       1       5       1.9     2.8     2.2
1       1       1       2       1       1.8     3.0     1.7
1       1       1       2       2       2.6     3.1     2.4
1       1       1       2       3       3.0     3.0     2.9
1       1       1       2       4       2.2     3.2     2.5
1       1       1       2       5       2.0     2.8     1.9
1       1       1       3       1       1.8     2.6     1.5
1       1       1       3       2       2.0     2.8     1.9
1       1       1       3       3       2.6     2.6     2.6
1       1       1       3       4       2.1     3.2     2.1
1       1       1       3       5       2.5     3.0     2.1
1       1       1       4       1       2.6     3.1     2.4
```

These were some basic hadoop commands. You can refer to this HDFS commands guide to learn more.

https://hadoop.apache.org/docs/r3.1.0/hadoop-project-dist/hadoop-hd...

Cloudera Quickstart VM Installation

What Is Cloudera QuickStart VM?

Cloudera QuickStart VM includes everything that you would need for using CDH, Impala, Cloudera Search, and Cloudera Manager. The Cloudera QuickStart VM uses a package-based install that allows you to work with or without the Cloudera Manager. It has a sample of Cloudera's platform for "Big Data."

Cloudera QuickStart VM Installation - Prerequisites

1. A virtual machine such as Oracle Virtual Box or VMWare
2. RAM of 12+ GB. That is 4+ GB for the operating system and 8+ GB for Cloudera
3. 80GB hard disk

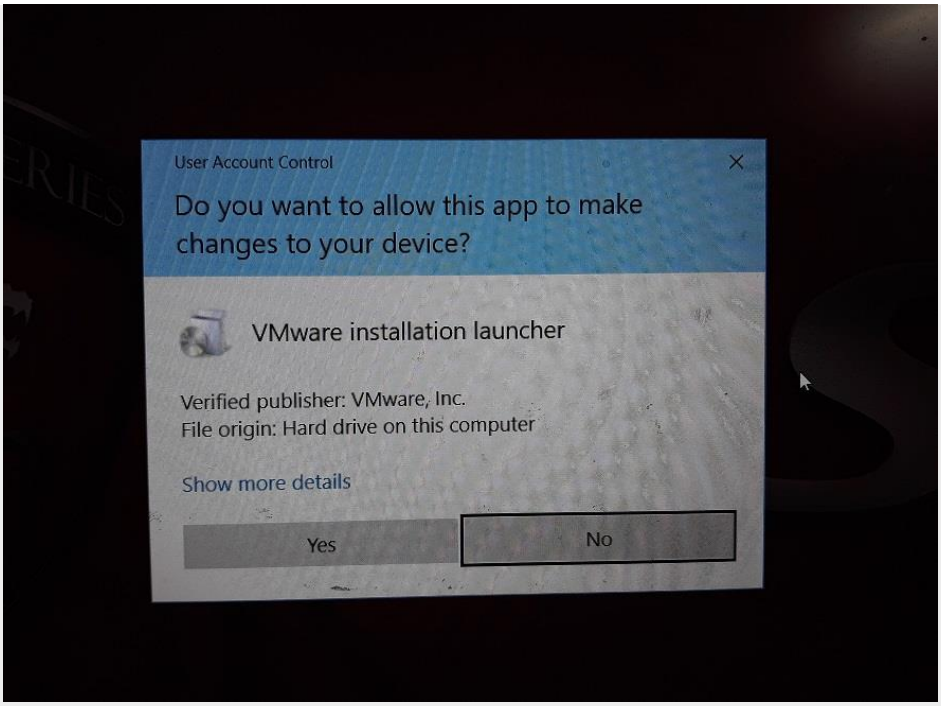
Steps to Install VMware Workstation Player 16 in Windows 10

VMware Workstation Player formally known as VMware Player is a Virtualization software used to run multiple virtual machines on the same hardware. Its available for both Windows and Linux based operating systems**.** It runs on 64 bit operating system, which means that if you have 32 bit operating system, you will not be able to use it.

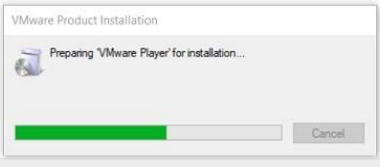
This is the direct link to download VMware Player 16 for Windows 10**.** Please note that this is a 64 bit application. VMware no longer supports 32 bit Operating Systems.In this blog, I will be using VMware Player Version 16.1.0, Installer file name is VMware-player-16.1.0-17198959.exe and is about 220 MB in size.

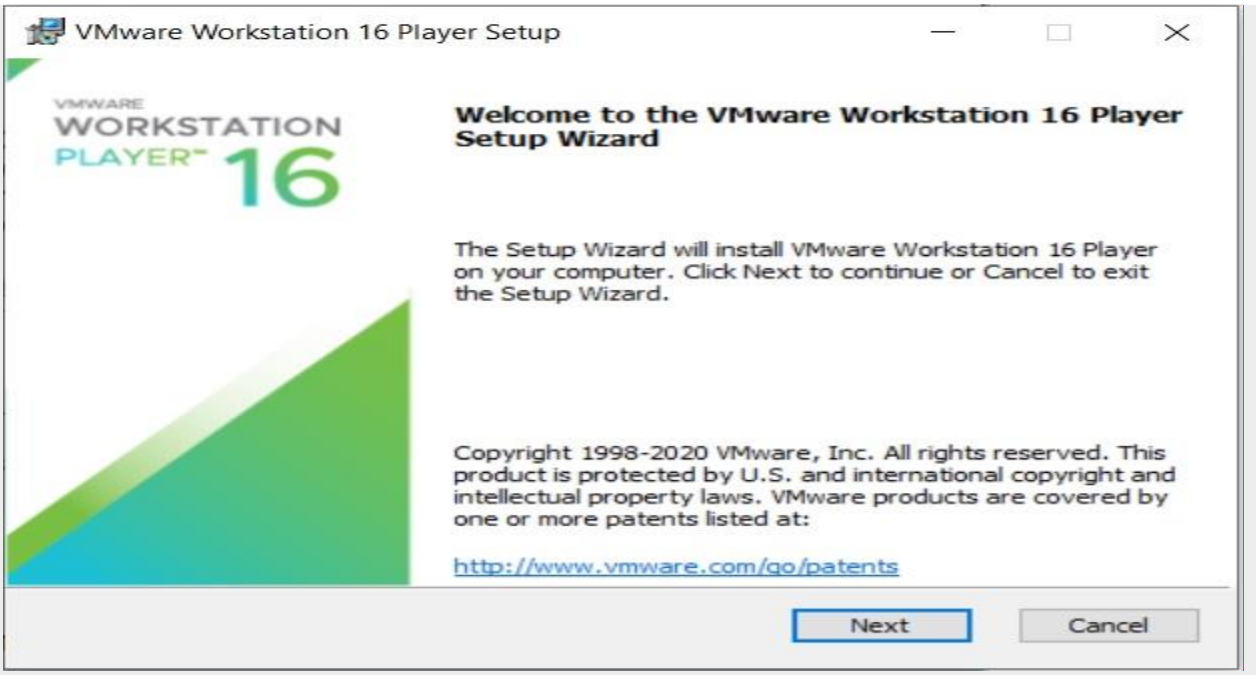
Step 1 – Run the installer

Start the installer by double clicking it. You might see User Account Control Warning. Click Yes to continue.

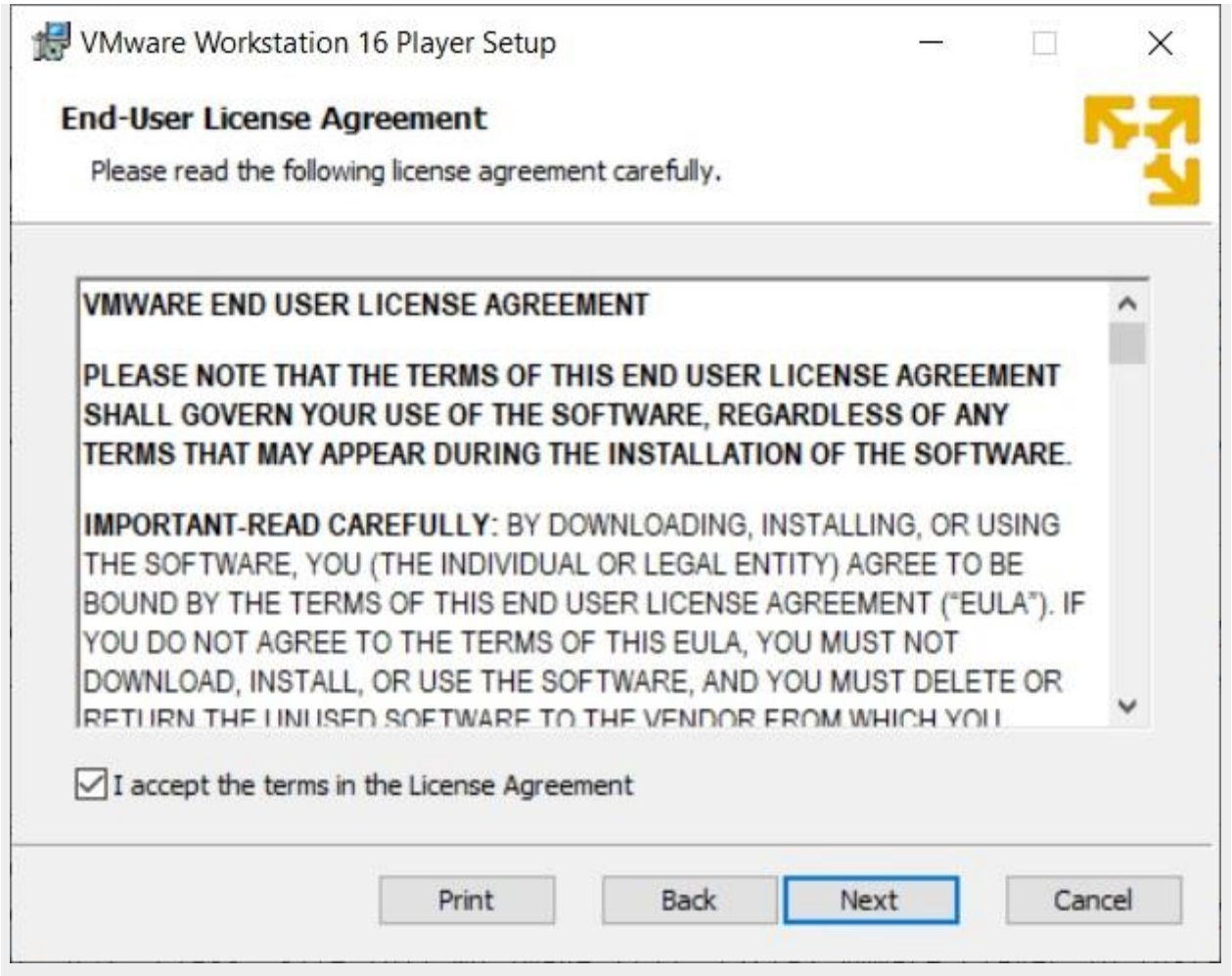


Then, you will see a splash screen. It will prepare the system for installation and then the installation wizard opens.





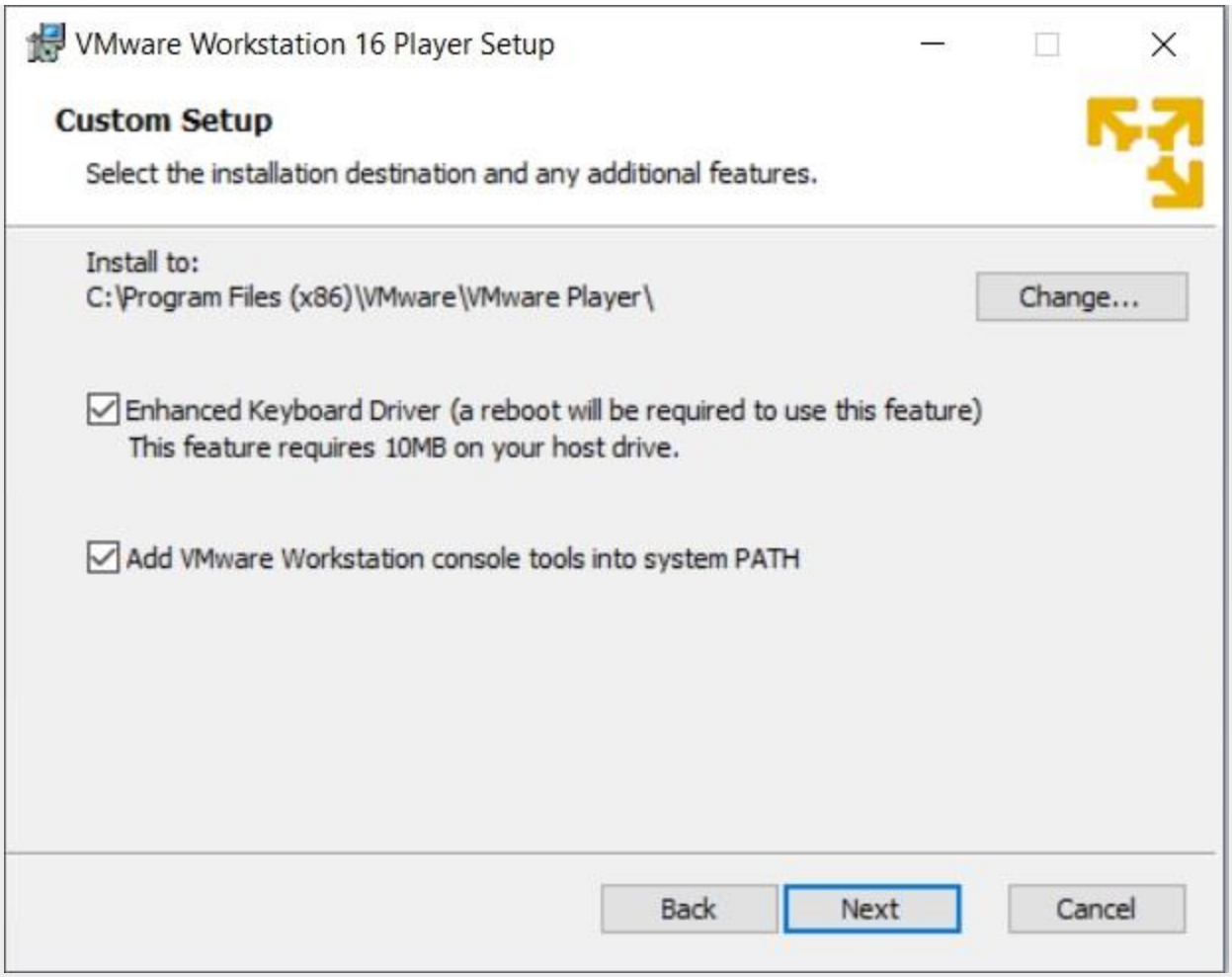
Click next and accept the license terms and click next again to move on to the next screen.



VMware Player 16 Installation – End User Agreement

Step 2 – Custom setup – Enhanced Keyboard driver and Installation directory

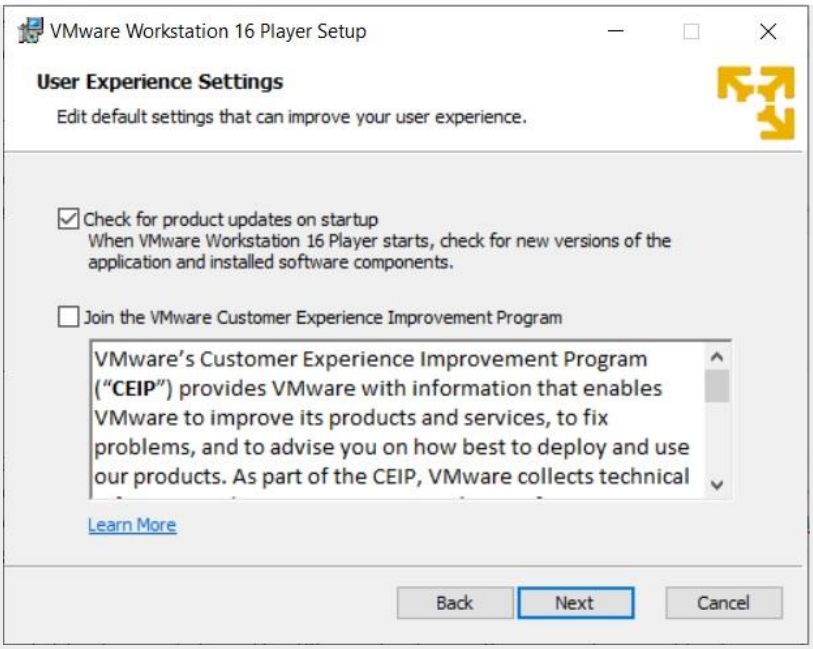
In this dialog box, please select the folder in which you want to install the application. I leave it as it is. Also check the box Enhanced Keyboard Drivers option. Click next.



VMware Player 16 Installation – Custom Setup

Step 3 – User Experience Settings

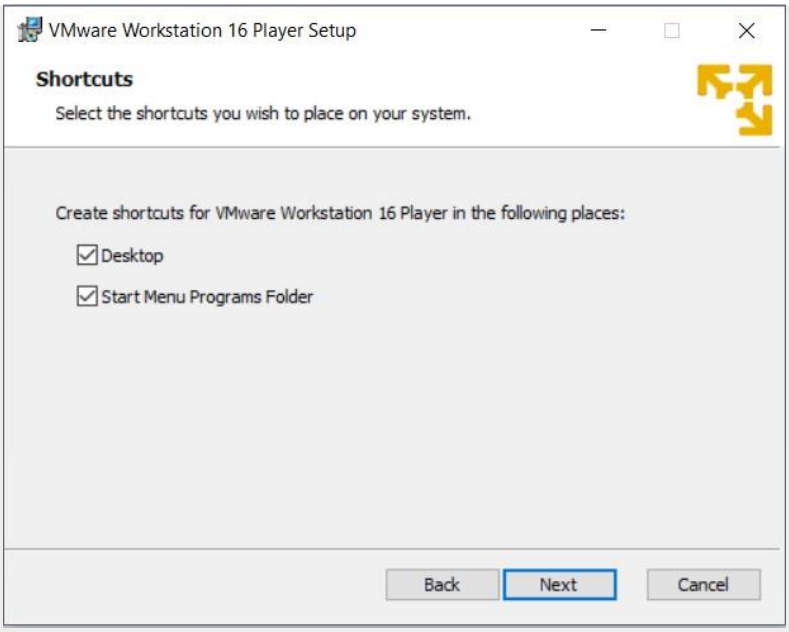
Check the options for Check the product update at Startup and Join the VMware Customer Program. I normally leave it as it is. You can unchecked it if you so desire. Click next



VMware Player 16 Installation – User Experience Settings

Step 4 – Select where the shortcuts will be installed

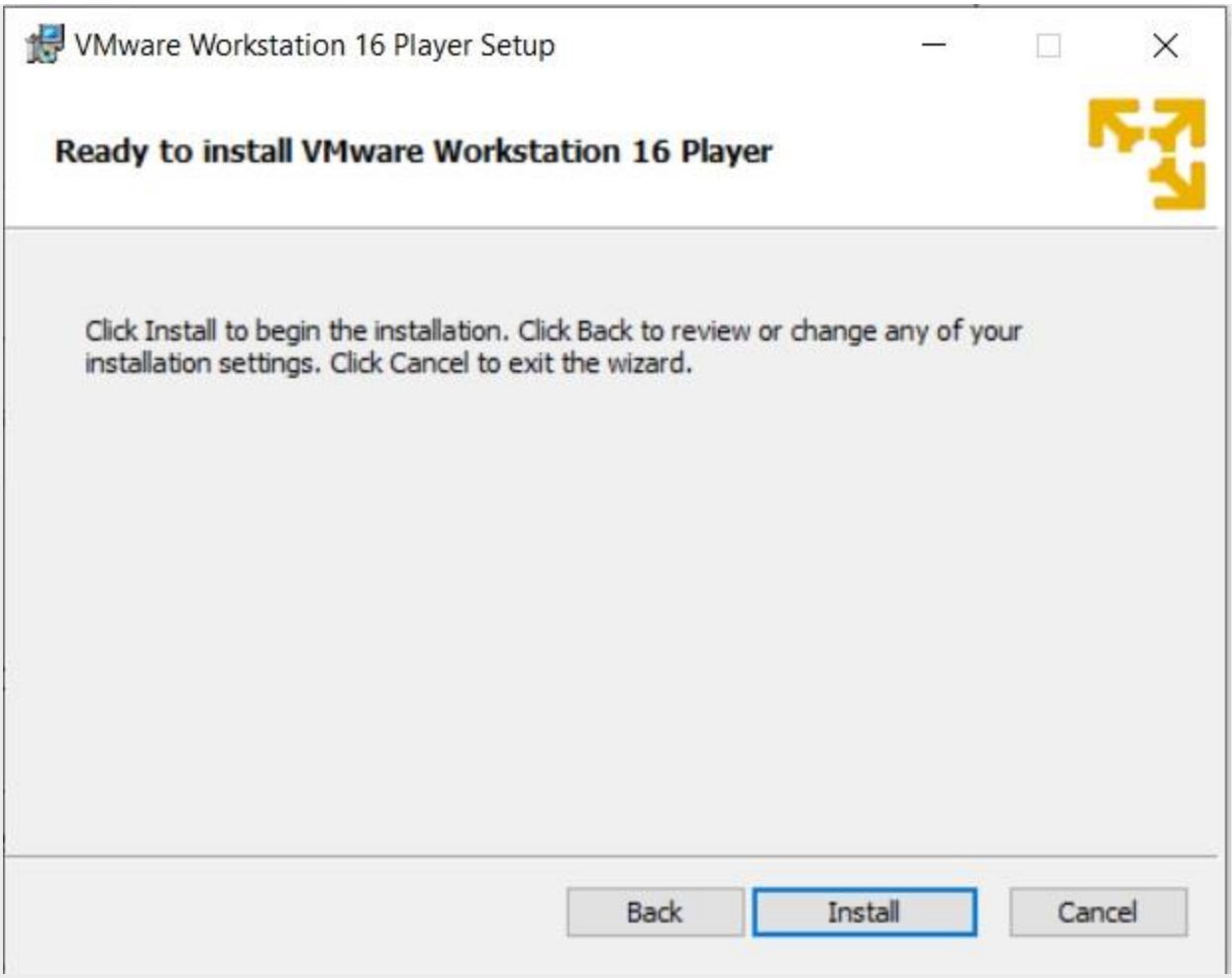
Check the box where the shortcut to run the application will be created. I leave it as it is. Click on next.



VMware Player 16 Installation – Shortcuts

Step 5 – Ready to install

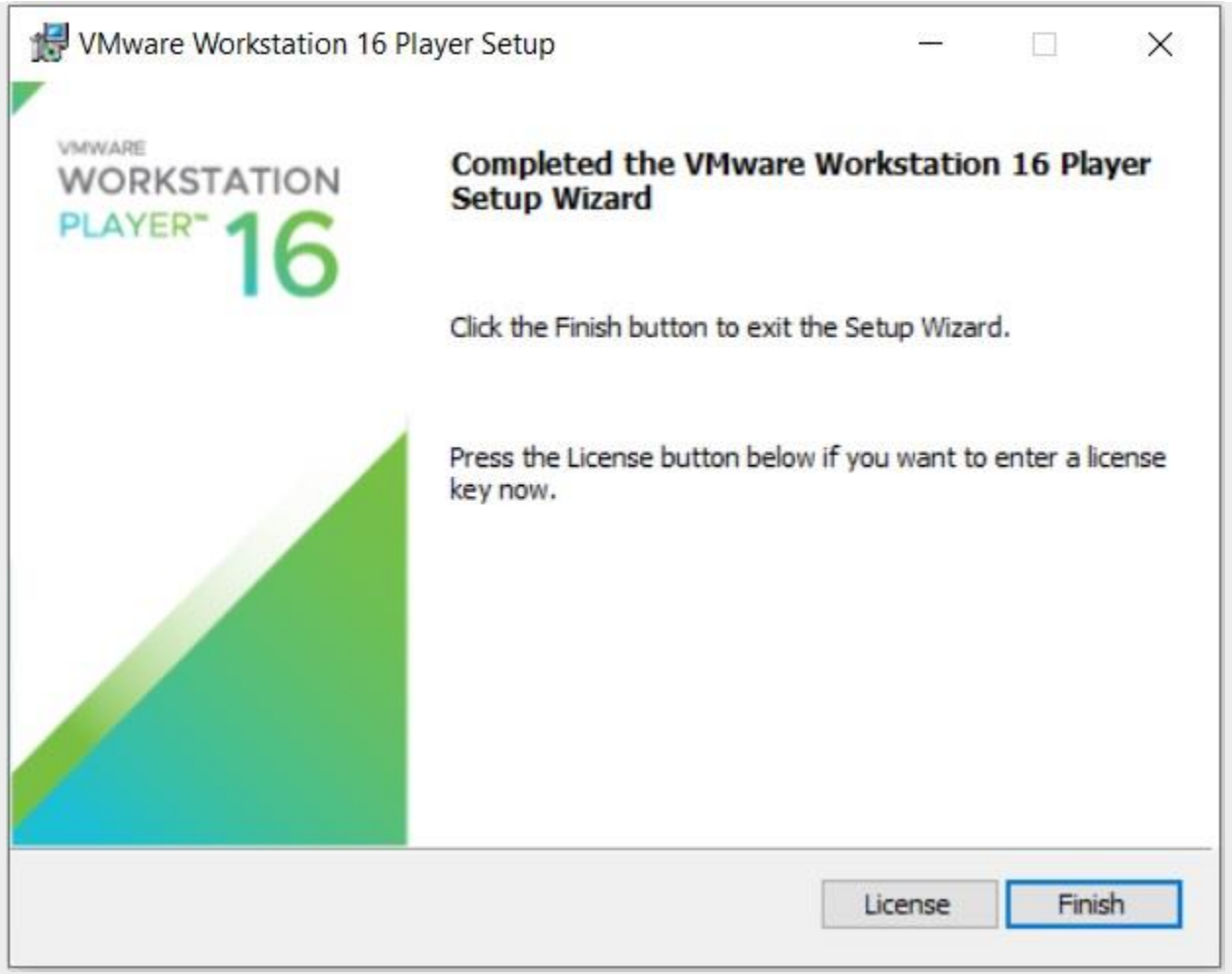
Now the installation wizard is ready to install. Click on install to begin the installation.



VMware Player 16 Installation – Ready to Install

Installation begins, wait for it to complete.

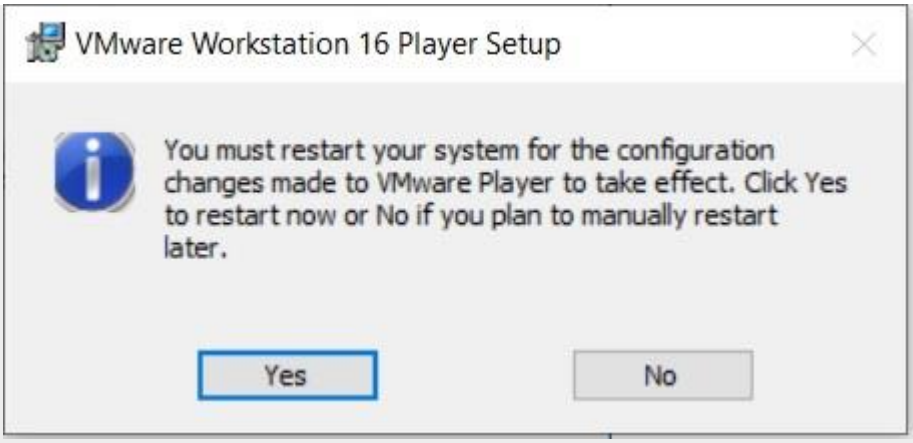
After sometime, you will see installation compete message. You are done.



VMware Player 16 Installation – Installation Complete

Click on Finish to Complete the installation.

You will be asked to restart your system. Click on Yes to restart. Click No, if you want to restart later. But you must restart before using the application, else some features will not work properly.



VMware Player 16 Installation – Reboot Required

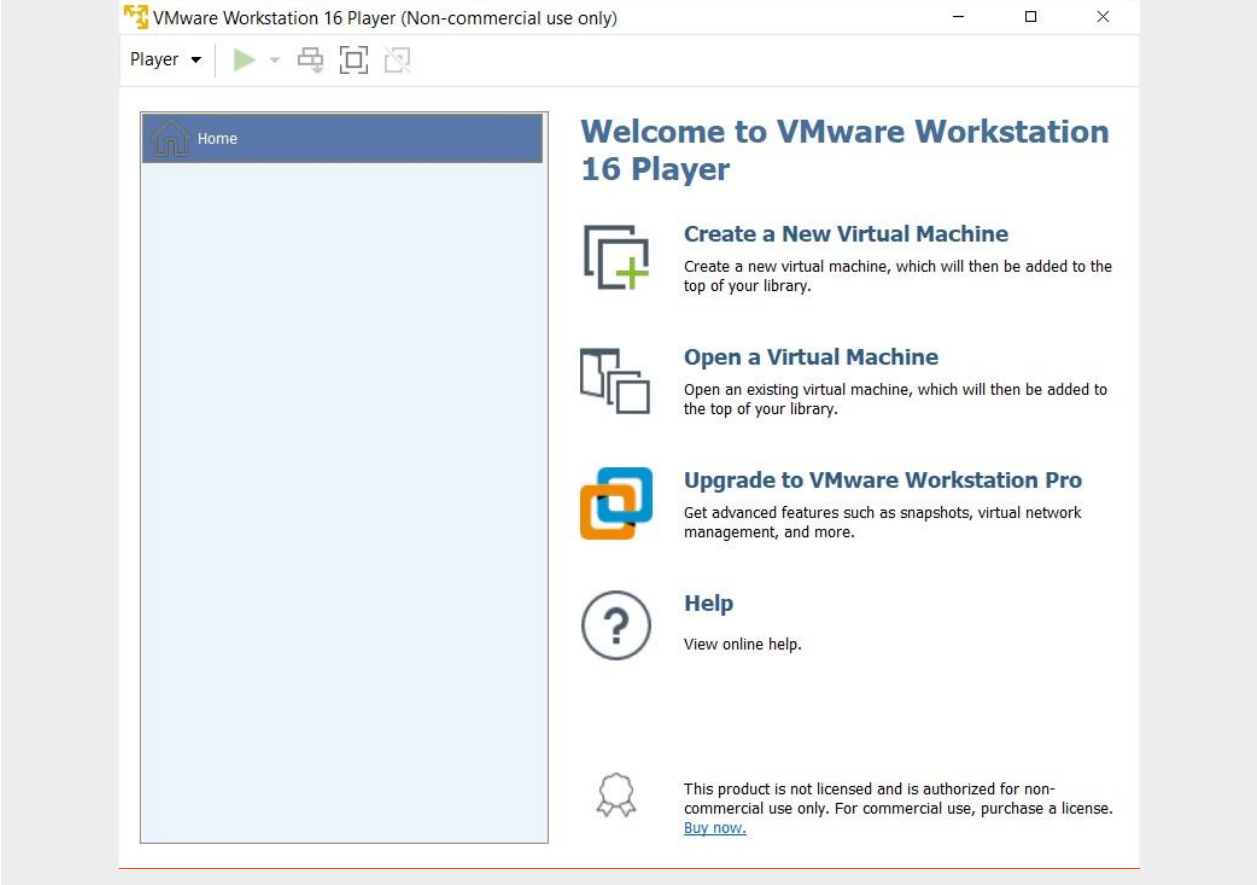
Step 6 – License

Now Run the application. You should see a desktop icon. Douple click on that or use the start menu to navigate to VMware Player option.Once you run the application for the first time, you will be asked for licence. Select the option Use VMware Workstation Player 16 for for free for non commercial use.Click continue.

VMware Player 16 Installation – License

Click on Finish.

Now you will see VMware Workstation Player 16 ready to be used for free for non-commercial purpose.

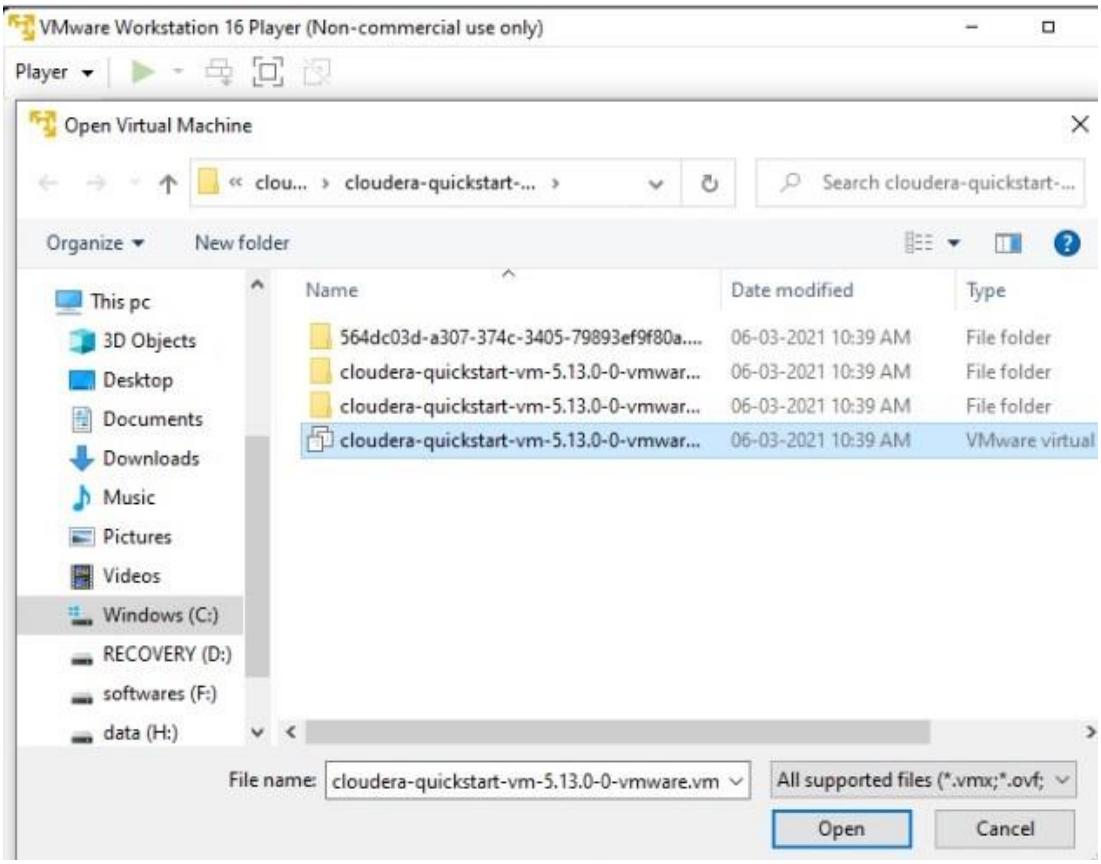


How to start cloudera in vmware,

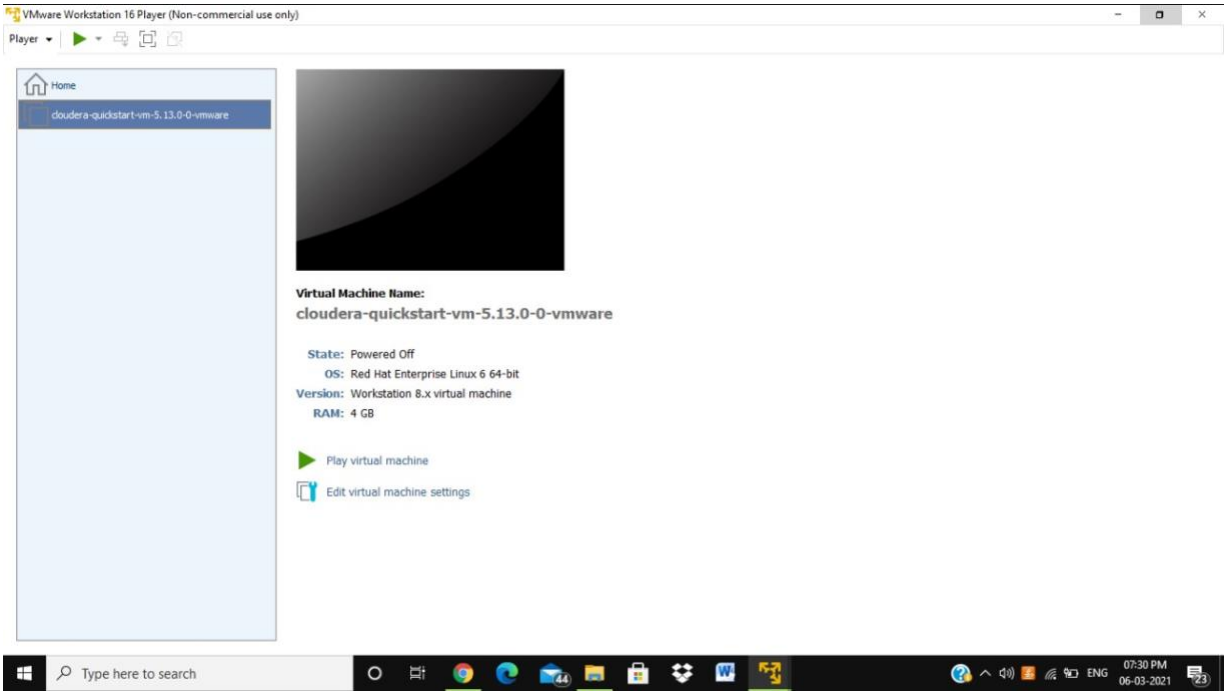
1. Download the cloudera folder and extract it to c-drive.

2.Now open vmware and click on open a virtual machine

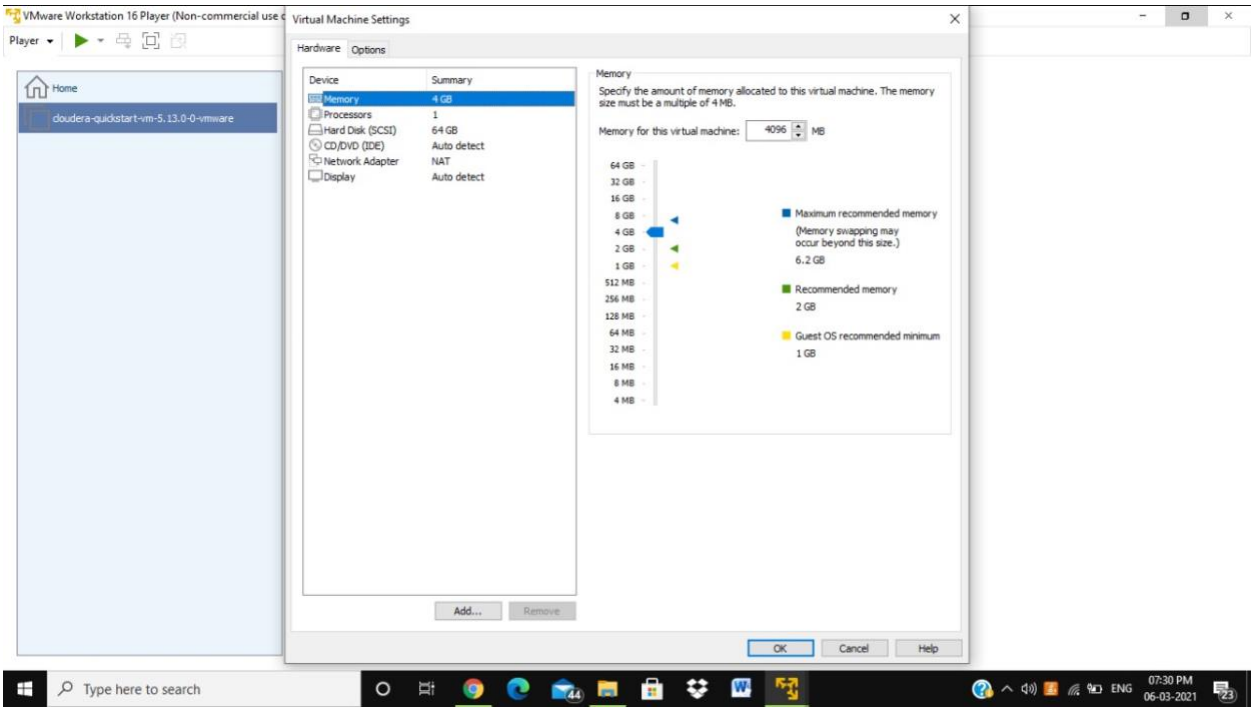
3. select the image file which is extracted from the downloaded cloudera folder(from c drive).



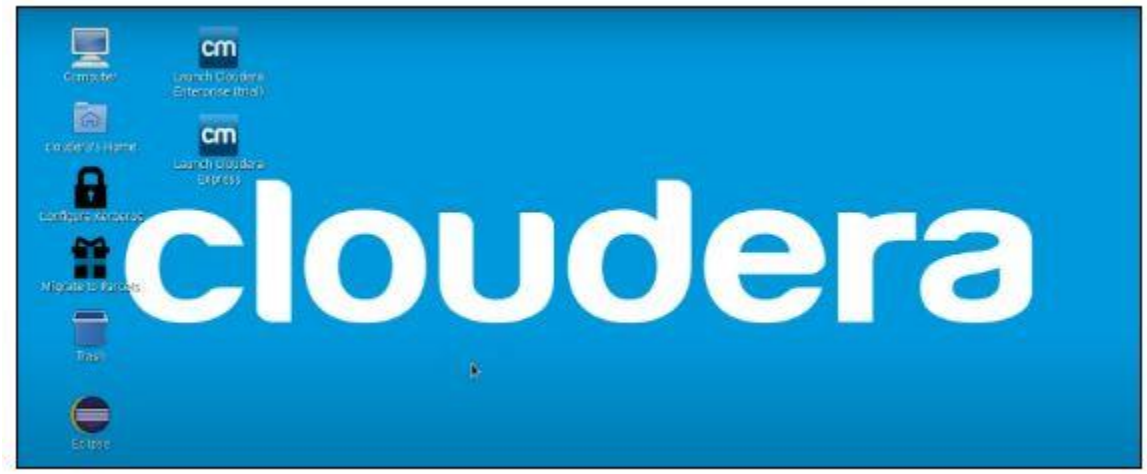
4. then click on edit virtual machine setting



5. choose the number of processor and memory



6. then click ok and click on play virtual machine.



HOW TO CHECK IF HDFS IS WORKING AND ALL THE DEAMONS ARE RUNNING

1. To Check if you have access and if your cluster is working.

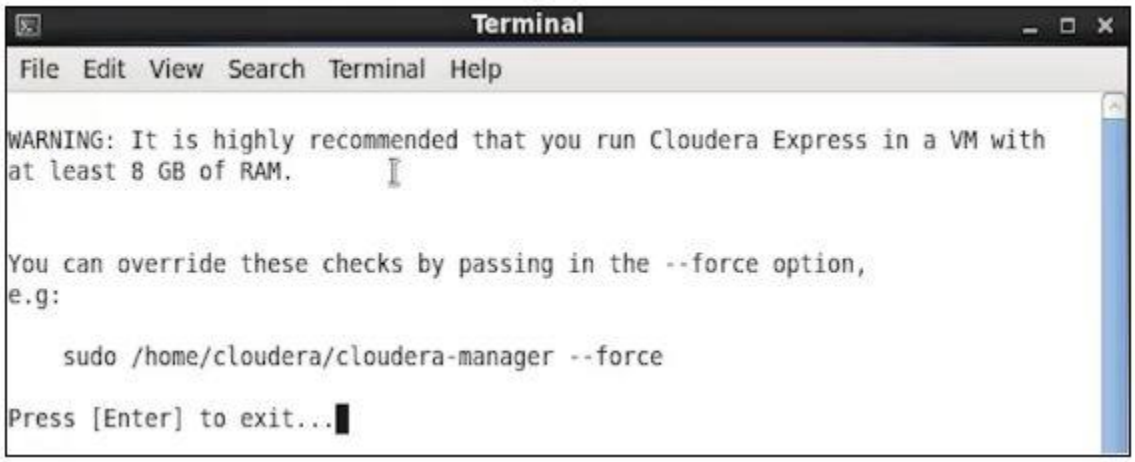
hdfs dfs -ls

It displays what exists on your HDFS location by default

2. To check if every clusters are working

- Once you see that your HDFS access is working fine, you can close the terminal. Then, you have to click on the following icon that says ‘Launch Cloudera Express’.

- Once you click on the express icon, a screen will appear with the following command:



- You are required to copy the command, and run it on a separate terminal. Hence, open a new terminal, and use the below command to close the Cloudera based services. It will restart the services, after which you can access your admin console.

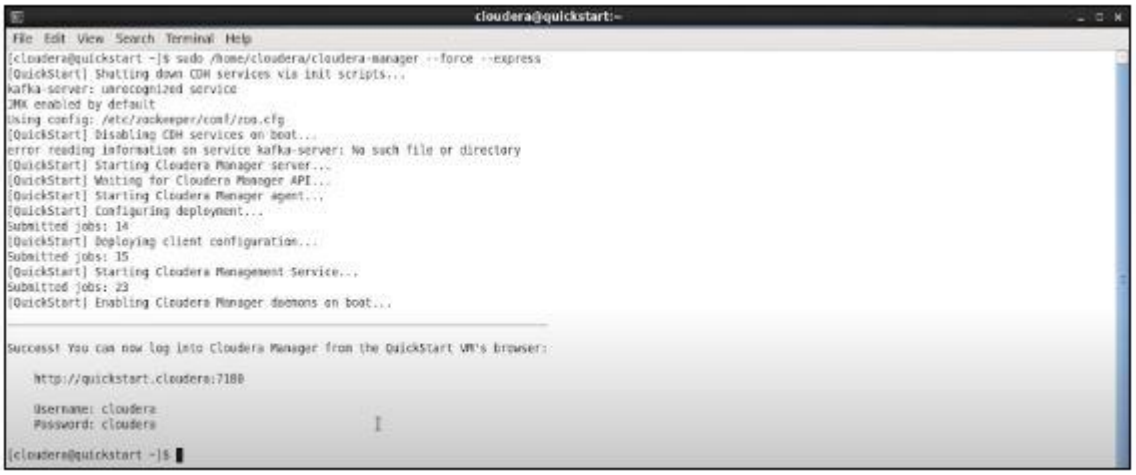


Fig: Restarting services on Cloudera QuickStart VM

- Now that our deployment has been configured, client configurations have also been deployed. Additionally, it has restarted the Cloudera Management Service, which gives access to the Cloudera QuickStart admin console with the help of a username and password.
- Go on and open up the browser and change the port number to 7180.
- You can log in to the Cloudera Manager by providing your username and password.

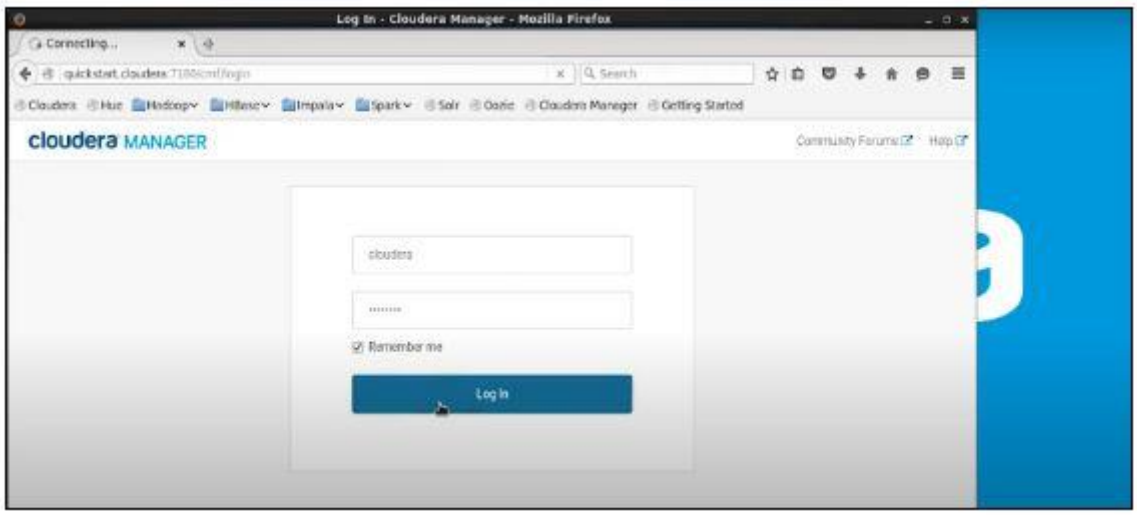


Fig: Logging in to Cloudera Manager

- Since Cloudera is CPU and memory intensive, it could slow down if you haven't assigned enough RAM to the Cloudera cluster. So, it's always recommended to stop or delete the services that you don't need.

- Next, click on the drill-down button associated with each service and select delete to remove it.

```
In [4]: class employee:
          def __init__(self, name, age):
            self.name = name
            self.age = age

          def myfunc(self):
            print("Hello my name is " + self.name)

        emp1 = employee("Jack", 31)

        del emp1.age

        print(emp1.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-05acb4ef51bf> in <module>
     11 del emp1.age
     12
---> 13 print(emp1.age)

AttributeError: 'employee' object has no attribute 'age'
```

Fig: Deleting unnecessary services on Cloudera QuickStart VM

- Before deleting any service, you must remove all the dependencies for that particular service. You can add services to your cluster at any point in time when you need it. You can also fix different configuration issues thereupon.

- In Cloudera Manager, you can fix the health issues or configuration issues within your cluster.

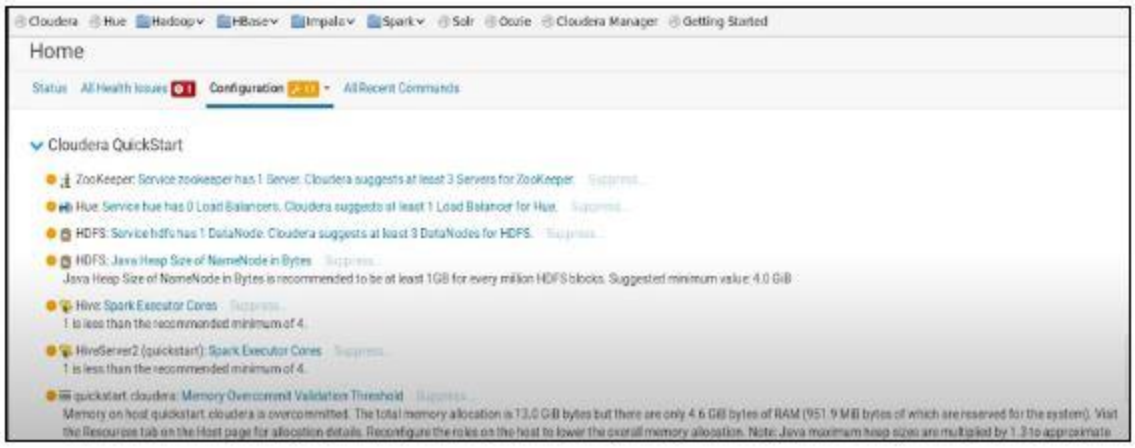


Fig: Solving Health and Configuration Issues on Cloudera QuickStart VM

- You can go ahead and restart the services now. It will ensure that the cluster becomes accessible either by Hue as a web interface or Cloudera QuickStart Terminal, where you can write your commands.

To check if all the deamons working , open terminal in cloudera , use jps

HDFS COMMANDS

1. open the terminal in desktop, command to create new folder

   mkdir foldername

2. use the below command to put the folder into HDFS

   hdfs dfs -put foldername/

3. to view the files under hdfs

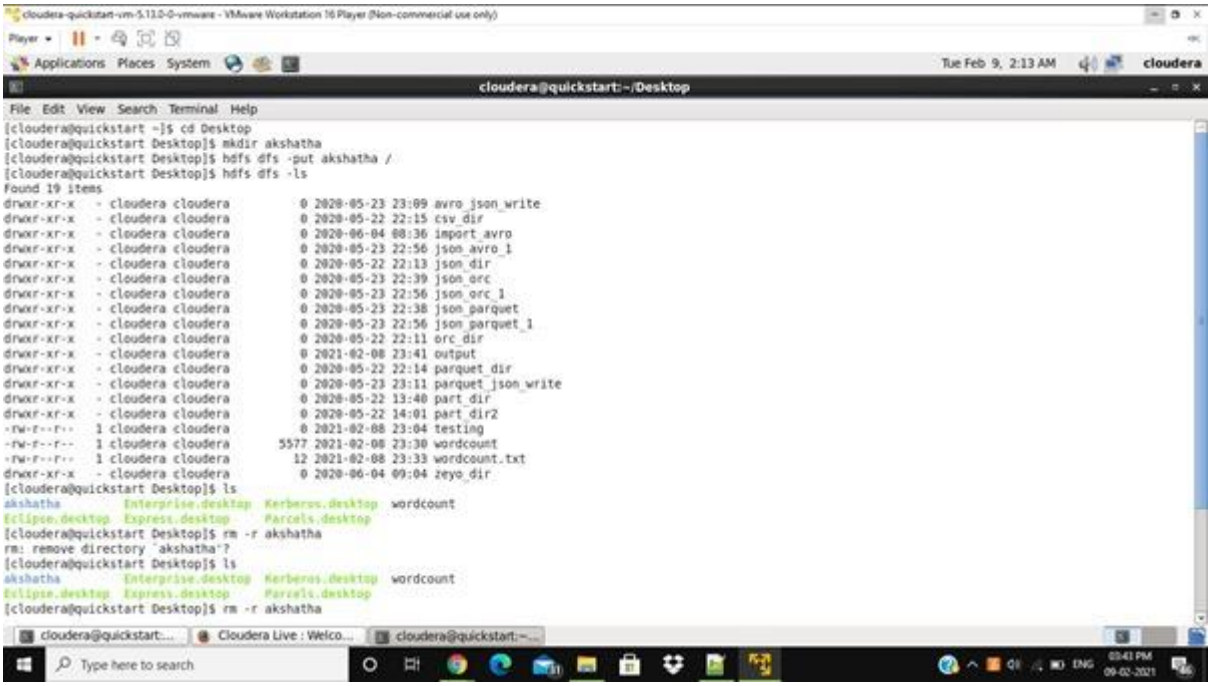
   hdfs dfs –ls

4. to remove the folder

   rm -r foldername

5. to get removed folder from HDFS

   hdfs dfs -get /foldername



Storing Data in HDFS:

1. create a text document

vi filename.txt

2. exit from vi editor , and open terminal , and using the below command the contents in filename.txt will be displayed.

cat filename.txt

3. give the below command to copy the file into HDFS.

hadoop fs –put filename.txt /user/cloudera

4. to verify if the file copied,

hdfs dfs –ls /user/cloudera

MAPREDUCE

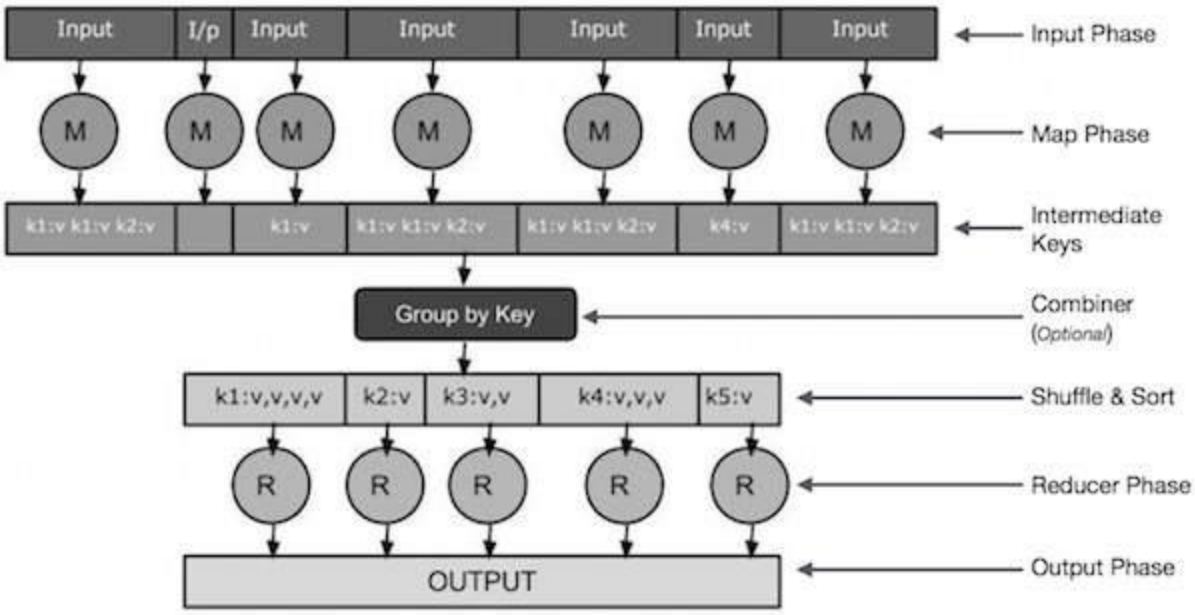
MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

HOW MAPREDUCE WORKS?

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

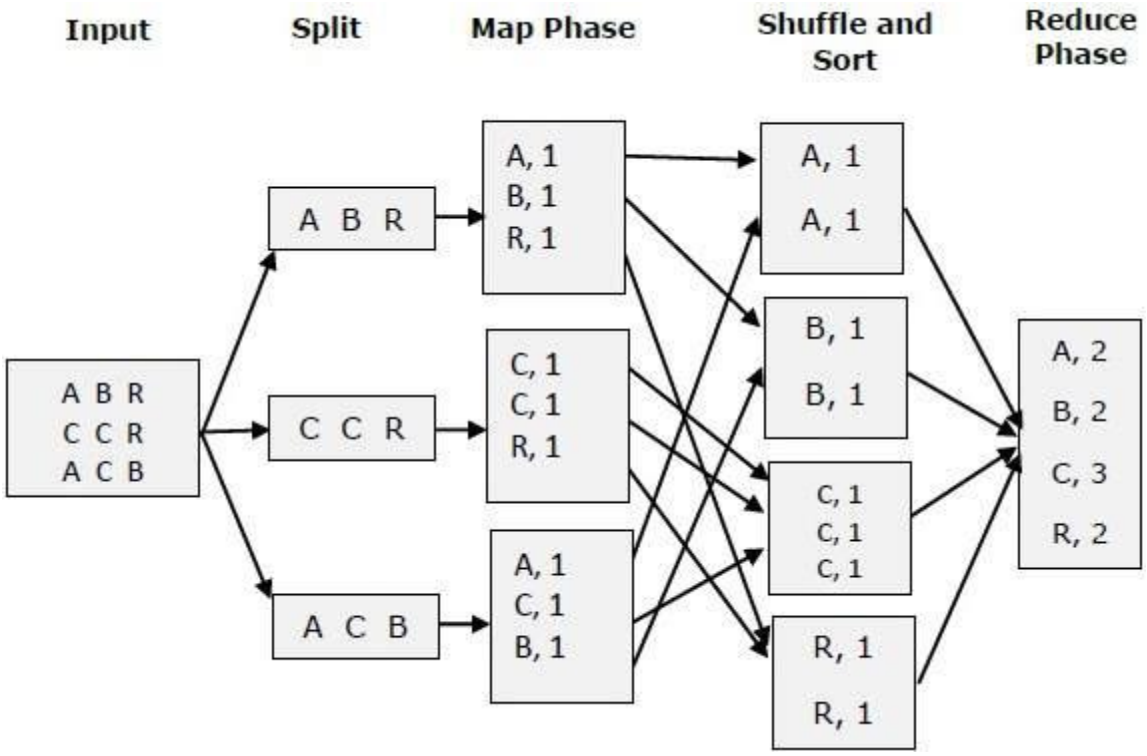
The reduce task is always performed after the map job. Let us now take a close look at each of the phases and try to understand their significance.



- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

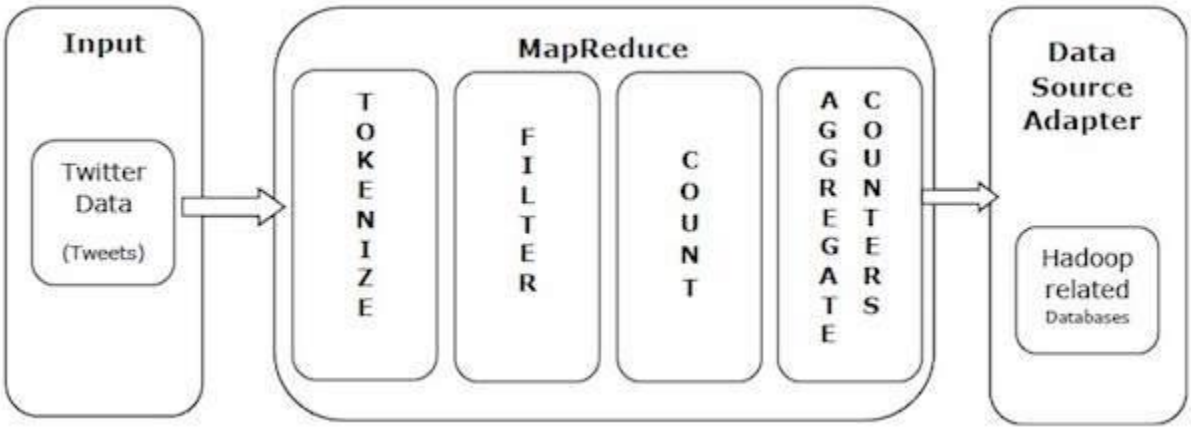
- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** − They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- **Output Phase** − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Let us try to understand the two tasks Map &f Reduce with the help of a small diagram −



MapReduce-Example

Let us take a real-world example to comprehend the power of MapReduce. Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.



As shown in the illustration, the MapReduce algorithm performs the following actions −

- **Tokenize** − Tokenizes the tweets into maps of tokens and writes them as key-value pairs.

- **Filter** − Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** − Generates a token counter per word.
- **Aggregate Counters** − Prepares an aggregate of similar counter values into small manageable units.

Able to Process data with MapReduce:

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.
```

Getting jar files:

```
[cloudera@quickstart ~]$ sudo find / -name "*hadoop*example*.jar"
/usr/lib/hue/apps/oozie/examples/lib/hadoop-examples.jar
/usr/lib/hadoop-0.20-mapreduce/hadoop-examples-mr1.jar
/usr/lib/hadoop-0.20-mapreduce/hadoop-examples-2.6.0-mr1-cdh5.13.0.jar
/usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar
/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.6.0-cdh5.13.0.jar
/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
/usr/share/doc/hadoop-0.20-mapreduce/examples/hadoop-examples-2.6.0-mr1-cdh5.13.0.jar
/usr/share/doc/hadoop-0.20-mapreduce/examples/hadoop-examples.jar
```

Word count is done by using MapReduce :

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/cloudera/srinu.txt /user/clou
21/02/13 22:50:06 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/02/13 22:50:10 INFO input.FileInputFormat: Total input paths to process : 1
21/02/13 22:50:11 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1252)
        at java.lang.Thread.join(Thread.java:1326)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
21/02/13 22:50:11 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1252)
        at java.lang.Thread.join(Thread.java:1326)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
21/02/13 22:50:11 INFO mapreduce.JobSubmitter: number of splits:1
21/02/13 22:50:12 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1613282121420_0001
21/02/13 22:50:14 INFO impl.YarnClientImpl: Submitted application application_1613282121420_0001
21/02/13 22:50:14 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1613282121420_0001/
21/02/13 22:50:14 INFO mapreduce.Job: Running job: job_1613282121420_0001
21/02/13 22:50:43 INFO mapreduce.Job: Job job_1613282121420_0001 running in uber mode : false
21/02/13 22:50:43 INFO mapreduce.Job:  map 0% reduce 0%
21/02/13 22:50:59 INFO mapreduce.Job:  map 100% reduce 0%
21/02/13 22:51:14 INFO mapreduce.Job:  map 100% reduce 100%
21/02/13 22:51:15 INFO mapreduce.Job: Job job_1613282121420_0001 completed successfully
21/02/13 22:51:15 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=125
                FILE: Number of bytes written=287487
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=191
                HDFS: Number of bytes written=87
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=13404
                Total time spent by all reduces in occupied slots (ms)=11733
```

Output for Word count:

```
[cloudera@quickstart ~]$ hadoop dfs -ls /user/cloudera/output
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2021-02-13 22:51 /user/cloudera/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera         87 2021-02-13 22:51 /user/cloudera/output/part-r-00000
[cloudera@quickstart ~]$ hadoop dfs -cat  /user/cloudera/output/part-r-00000
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

arjaguna        1
balireddy       1
bayyavarapu     1
bhargav 1
katragunta      1
praveen 1
sairam  1
srinu   1
[cloudera@quickstart ~]$
```

HBASE

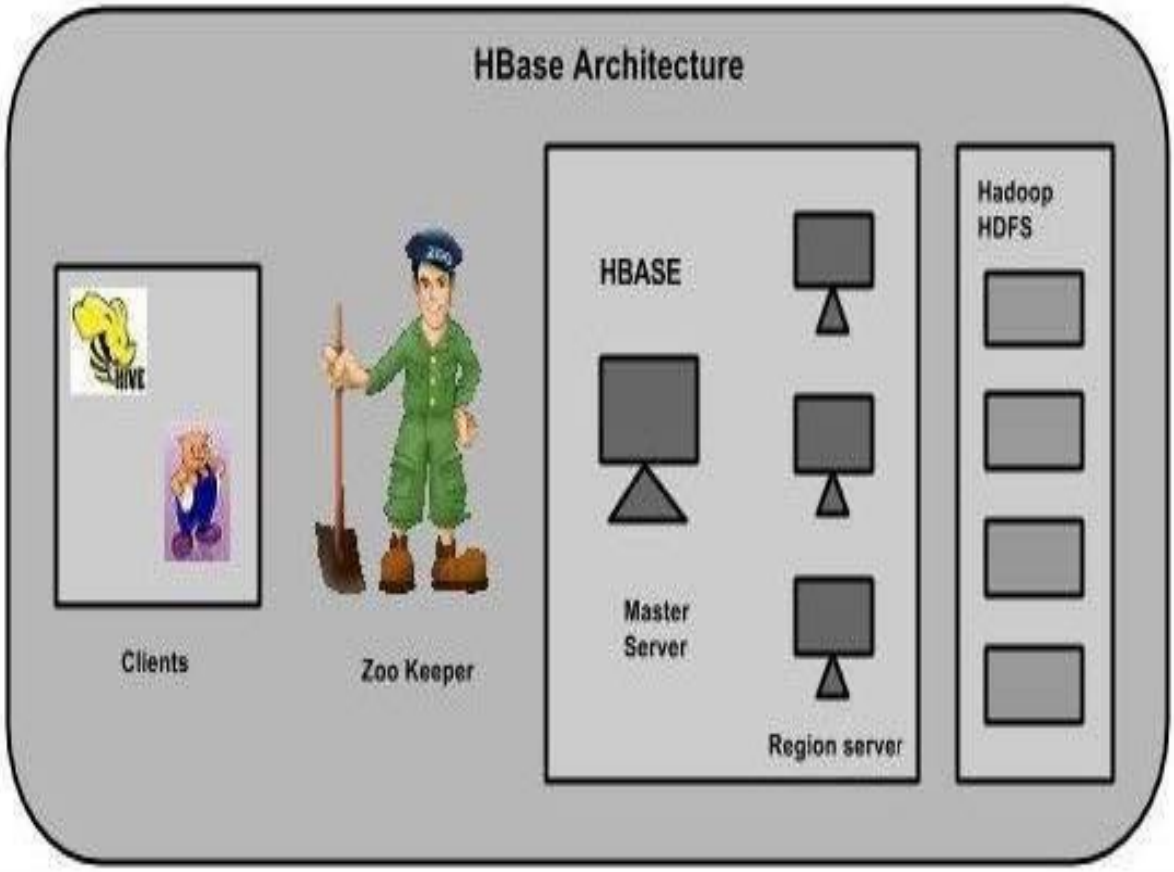
What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

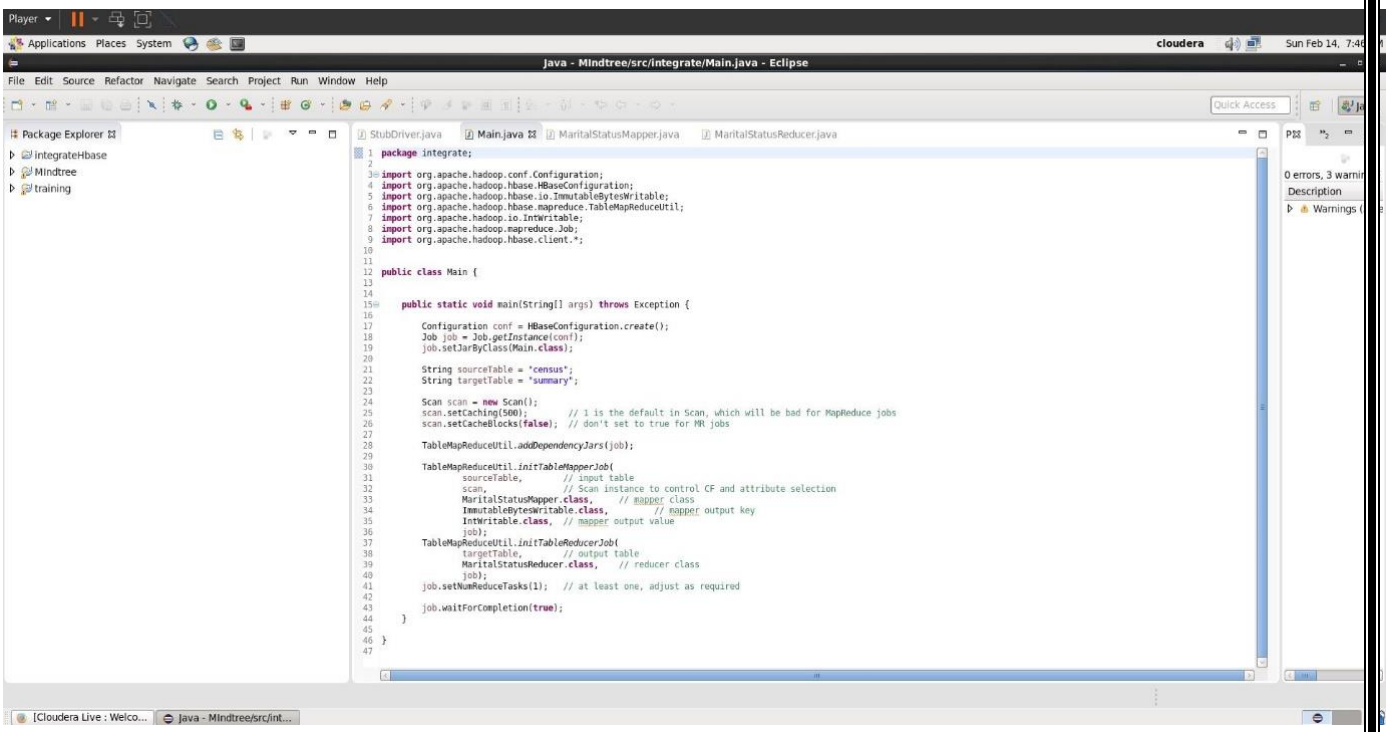
One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



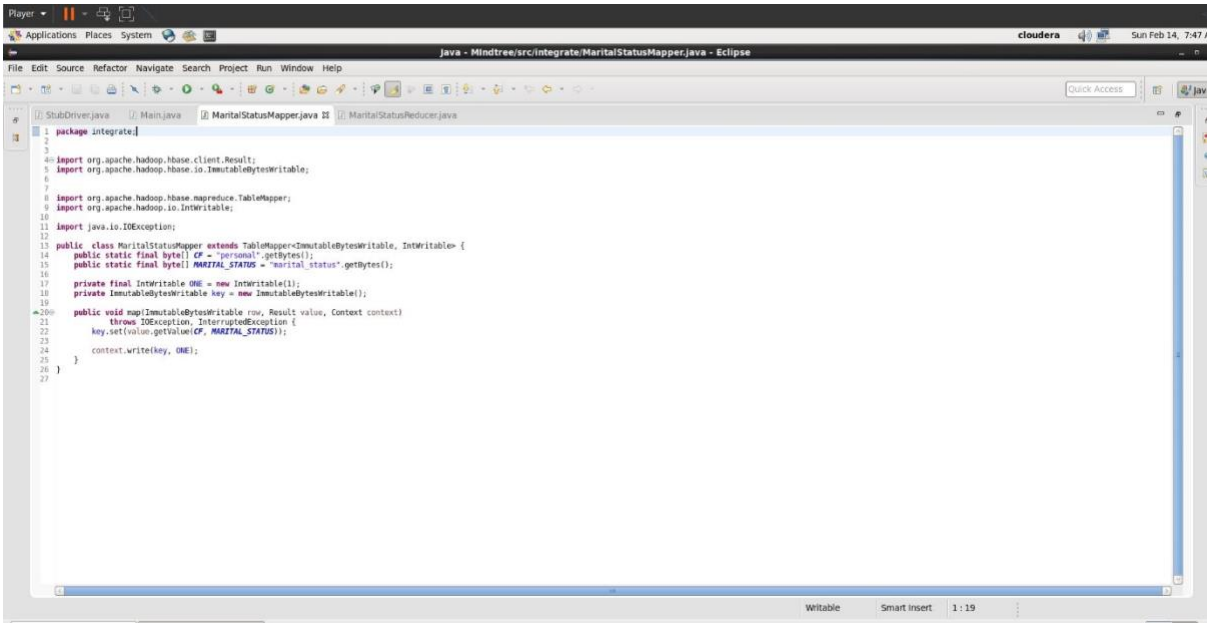
Integrate MapReduce with Hbase:

In this we have three Classes to perform MapReduce with Hbase

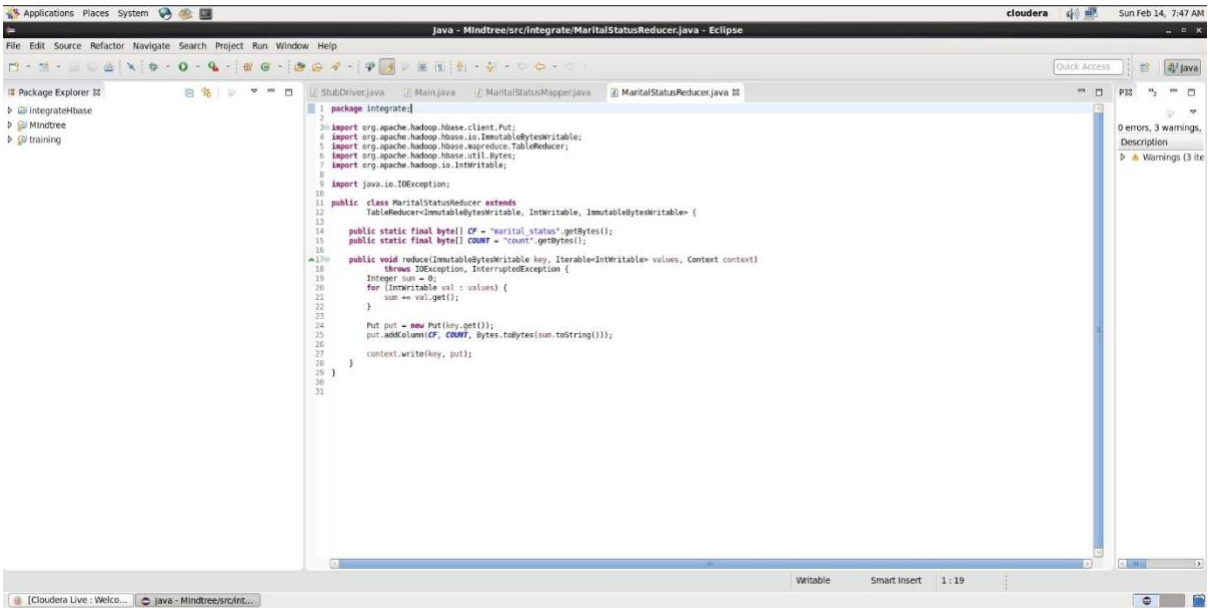
Main Class:



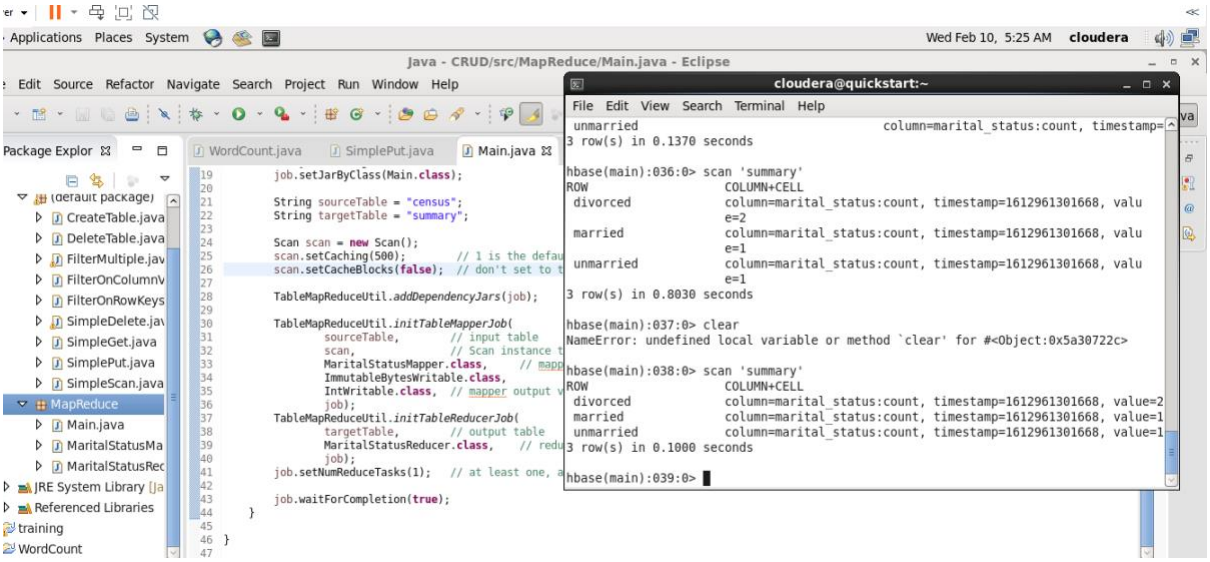
MaritalStatusMapper:



MaritalStatusReducer:



After Running Main Class:



HBase APIs for Java "Hello World" Application

This example is a very simple "hello world" application, using the Cloud Bigtable HBase client library for Java, that illustrates how to:

- Connect to a Cloud Bigtable instance.
- Create a new table.
- Write data to the table.
- Read the data back.
- Delete the table.

Running the sample

The sample uses the HBase APIs to communicate with Cloud Bigtable. The code for this sample is in the GitHub repository GoogleCloudPlatform/cloud-bigtable-examples, in the directory java/hello-world. Using the HBase APIs the sample application connects to Cloud Bigtable and demonstrates some simple operations.

Installing and importing the client library

```
import com.google.cloud.bigtable.hbase.BigtableConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Admin;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;
```

Connecting to Cloud Bigtable

```
// Create the Bigtable connection, use try-with-resources to make sure it gets closed
try (Connection connection = BigtableConfiguration.connect(projectId, instanceId)) {

  // The admin API lets us create, manage and delete tables
  Admin admin = connection.getAdmin();
```

Creating a table

Use the Admin API to create a table.

```
// Create a table with a single column family
HTableDescriptor descriptor = new HTableDescriptor(TableName.valueOf(TABLE_NAME));
descriptor.addFamily(new HColumnDescriptor(COLUMN_FAMILY_NAME));

print("Create table " + descriptor.getNameAsString());
admin.createTable(descriptor);
```

Writing rows to a table

Use the Table class to put rows to the table. For better throughput, consider using the BigtableBufferedMutator class.

```
// Retrieve the table we just created so we can do some reads and writes
Table table = connection.getTable(TableName.valueOf(TABLE_NAME));

// Write some rows to the table
print("Write some greetings to the table");
for (int i = 0; i < GREETINGS.length; i++) {
  // Each row has a unique row key.
  //
  // Note: This example uses sequential numeric IDs for simplicity, but
  // this can result in poor performance in a production application.
  // Since rows are stored in sorted order by key, sequential keys can
  // result in poor distribution of operations across nodes.
  //
  // For more information about how to design a Bigtable schema for the
  // best performance, see the documentation:
  //
  //     https://cloud.google.com/bigtable/docs/schema-design
  String rowKey = "greeting" + i;
```

```
  // Put a single row into the table. We could also pass a list of Puts to write a batch.
  Put put = new Put(Bytes.toBytes(rowKey));
  put.addColumn(COLUMN_FAMILY_NAME, COLUMN_NAME,
Bytes.toBytes(GREETINGS[i]));
  table.put(put);
}
```

Reading a row by its key :-Get a row directly using its key.

```
// Get the first greeting by row key
String rowKey = "greeting0";
Result getResult = table.get(new Get(Bytes.toBytes(rowKey)));
String greeting = Bytes.toString(getResult.getValue(COLUMN_FAMILY_NAME,
COLUMN_NAME));
System.out.println("Get a single greeting by row key");
System.out.printf("\t%s = %s\n", rowKey, greeting);
```

Scanning all table rows

Use the Scan class to get a range of rows.

```
// Now scan across all rows.
Scan scan = new Scan();


print("Scan for all greetings:");
ResultScanner scanner = table.getScanner(scan);
for (Result row : scanner) {
  byte[] valueBytes = row.getValue(COLUMN_FAMILY_NAME, COLUMN_NAME);
  System.out.println('\t' + Bytes.toString(valueBytes));
}
```

Deleting a table :- Delete a table using the Admin API.

```
// Clean up by disabling and then deleting the table
print("Delete the table");
admin.disableTable(table.getName());
admin.deleteTable(table.getName());
```

CRUD OPERATION

Create:

In this we have to create the table ‘census’ by using the below command

```
hbase(main):003:0> create 'census', 'personal', 'professional'
0 row(s) in 2.6780 seconds

=> Hbase::Table - census
```

list and describe:

By using this ‘list’ and ‘describe’ we have to identify what are the tables we have and analyze what type of table it is by the help of ‘describe’ command

```
hbase(main):004:0> list
TABLE
census
1 row(s) in 0.0610 seconds

=> ["census"]
hbase(main):005:0> describe 'census'
Table census is ENABLED
census
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE
 => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'professional', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCK
SIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.3340 seconds

hbase(main):006:0>
```

Put:

By using this put command we have to insert the data into the table

put 'census', 1, 'personal:name', 'Mike Jones'

put 'census', 1, 'personal:marital_status', 'unmarried'

put 'census', 1, 'personal:gender', 'male'

put 'census', 1, 'professional:employed', 'yes'

put 'census', 1, 'professional:education_level', 'high school'

put 'census', 1, 'professional:field', 'construction'

```
hbase(main):006:0> put 'census', 1, 'personal:name', 'Mike Jones'
0 row(s) in 0.4470 seconds

hbase(main):007:0> put 'census', 1, 'personal:marital_status', 'unmarried'
0 row(s) in 0.1200 seconds

hbase(main):008:0> put 'census', 1, 'personal:gender', 'male'
0 row(s) in 0.0170 seconds

hbase(main):009:0> put 'census', 1, 'professional:employed', 'yes'
0 row(s) in 0.0620 seconds

hbase(main):010:0> put 'census', 1, 'professional:education_level', 'high school'
0 row(s) in 0.0220 seconds

hbase(main):011:0> put 'census', 1, 'professional:field', 'construction'
0 row(s) in 0.0300 seconds

hbase(main):012:0> put 'census', 3, 'personal:name', 'Jill Tang'
0 row(s) in 0.0260 seconds

hbase(main):013:0> put 'census', 3, 'personal:marital_status', 'married'
0 row(s) in 0.0270 seconds

hbase(main):014:0> put 'census', 3, 'personal:spouse', 'Jim Tang'
0 row(s) in 0.0310 seconds

hbase(main):015:0> put 'census', 3, 'professional:education_level', 'post-grad'
0 row(s) in 0.0240 seconds

hbase(main):016:0> put 'census', 3, 'personal:gender', 'female'
0 row(s) in 0.0460 seconds

hbase(main):017:0> put 'census', 2, 'personal:name', 'Ben'
0 row(s) in 0.0330 seconds

hbase(main):018:0> put 'census', 2, 'personal:marital_status', 'divorced'
0 row(s) in 0.0620 seconds

hbase(main):019:0> put 'census', 2, 'personal:gender', 'male'
0 row(s) in 0.0330 seconds

hbase(main):020:0> put 'census', 4, 'personal:name', 'Maria'
0 row(s) in 0.0240 seconds

hbase(main):021:0> put 'census', 4, 'personal:marital_status', 'divorced'
```

Scan:

Scan is nothing but displaying the details in the table ..by using this scan command we able to identify the whole data in table

```
hbase(main):023:0> scan 'census'
ROW                                                  COLUMN+CELL
 1                                                   column=personal:gender, timestamp=1613286504479, value=male
 1                                                   column=personal:marital_status, timestamp=1613286481372, value=unmarried
 1                                                   column=personal:name, timestamp=1613286463746, value=Mike Jones
 1                                                   column=professional:education_level, timestamp=1613286528583, value=high school
 1                                                   column=professional:employed, timestamp=1613286519963, value=yes
 1                                                   column=professional:field, timestamp=1613286553082, value=construction
 2                                                   column=personal:gender, timestamp=1613286642617, value=male
 2                                                   column=personal:marital_status, timestamp=1613286632402, value=divorced
 2                                                   column=personal:name, timestamp=1613286619630, value=Ben
 3                                                   column=personal:gender, timestamp=1613286609222, value=female
 3                                                   column=personal:marital_status, timestamp=1613286580441, value=married
 3                                                   column=personal:name, timestamp=1613286570934, value=Jill Tang
 3                                                   column=personal:spouse, timestamp=1613286589388, value=Jim Tang
 3                                                   column=professional:education_level, timestamp=1613286599993, value=post-grad
 4                                                   column=personal:gender, timestamp=1613286668448, value=female
 4                                                   column=personal:marital_status, timestamp=1613286660447, value=divorced
 4                                                   column=personal:name, timestamp=1613286651586, value=Maria
4 row(s) in 0.4120 seconds

hbase(main):024:0>
```

Get:

Get command is using for retrieving the data

```
hbase(main):025:0> get 'census', 1 , 'personal:name'
COLUMN                                               CELL
 personal:name                                       timestamp=1613286463746, value=Mike Jones
1 row(s) in 0.0420 seconds

hbase(main):026:0> get 'census',3, 'professional:education_level'
COLUMN                                               CELL
 professional:education_level                        timestamp=1613286599993, value=post-grad
1 row(s) in 0.0290 seconds

hbase(main):027:0>
```

Delete:

By using this delete command we have to delete the particular column or row

```
hbase(main):009:0> delete 'census', 1, 'personal:marital_status'
0 row(s) in 0.2000 seconds

hbase(main):010:0> get 'census',1
COLUMN                                          CELL
 personal:gender                                timestamp=1613286504479, value=male
 personal:name                                  timestamp=1613286463746, value=Mike Jones
 professional:education_level                   timestamp=1613286528583, value=high school
 professional:employed                          timestamp=1613286519963, value=yes
 professional:field                             timestamp=1613286553082, value=construction
5 row(s) in 0.1130 seconds

hbase(main):011:0>
```

Limit:

Limit is used when we want the top details like top 1 column in table like that

```
hbase(main):003:0> scan 'census'
ROW                                             COLUMN+CELL
 1                                              column=personal:gender, timestamp=1613286504479, value=male
 1                                              column=personal:marital_status, timestamp=1613286481372, value=unmarried
 1                                              column=personal:name, timestamp=1613286463746, value=Mike Jones
 1                                              column=professional:education_level, timestamp=1613286528583, value=high school
 1                                              column=professional:employed, timestamp=1613286519963, value=yes
 1                                              column=professional:field, timestamp=1613286553082, value=construction
 2                                              column=personal:gender, timestamp=1613286642617, value=male
 2                                              column=personal:marital_status, timestamp=1613286632402, value=divorced
 2                                              column=personal:name, timestamp=1613286619630, value=Ben
 3                                              column=personal:gender, timestamp=1613286609222, value=female
 3                                              column=personal:marital_status, timestamp=1613286580441, value=married
 3                                              column=personal:name, timestamp=1613286570934, value=Jill Tang
 3                                              column=personal:spouse, timestamp=1613286589388, value=Jim Tang
 3                                              column=professional:education_level, timestamp=1613286599993, value=post-grad
 4                                              column=personal:gender, timestamp=1613286668448, value=female
 4                                              column=personal:marital_status, timestamp=1613286660447, value=divorced
 4                                              column=personal:name, timestamp=1613286651586, value=Maria
4 row(s) in 0.7450 seconds

hbase(main):004:0> scan 'census',{COLUMNS=>['personal:name'], LIMIT=>1}
ROW                                             COLUMN+CELL
 1                                              column=personal:name, timestamp=1613286463746, value=Mike Jones
1 row(s) in 0.0460 seconds
```

Start row:

By using this start row we print the table based on number like startrow=>1,startrow=>2

```
hbase(main):006:0> scan 'census',{COLUMNS=>['personal:name'], LIMIT=>1, STARTROW=>'3'}
ROW                                             COLUMN+CELL
 3                                              column=personal:name, timestamp=1613286570934, value=Jill Tang
1 row(s) in 0.0510 seconds
```

Stop row:

By using this stop row we able to stop the printing data in the table based on that condition:

```
hbase(main):008:0> scan 'census',{COLUMNS=>['personal:name'],  STARTROW=>'1',  STOPROW=>'3'}
ROW                                             COLUMN+CELL
 1                                              column=personal:name, timestamp=1613286463746, value=Mike Jones
 2                                              column=personal:name, timestamp=1613286619630, value=Ben
2 row(s) in 0.1240 seconds
```

Disable ,enable,drop:

By using this we have to disable and enable the table whenever we need

Drop is done when the table is disabled if we try to drop the table without disabling the table, it shows error

```
hbase(main):011:0> disable 'census'
0 row(s) in 2.7030 seconds

hbase(main):012:0> enable 'census'
0 row(s) in 1.4180 seconds

hbase(main):013:0> drop 'census'

ERROR: Table census is enabled. Disable it first.

Drop the named table. Table must first be disabled:
  hbase> drop 't1'
  hbase> drop 'ns1:t1'
```

Drop is done after the disabling of table:

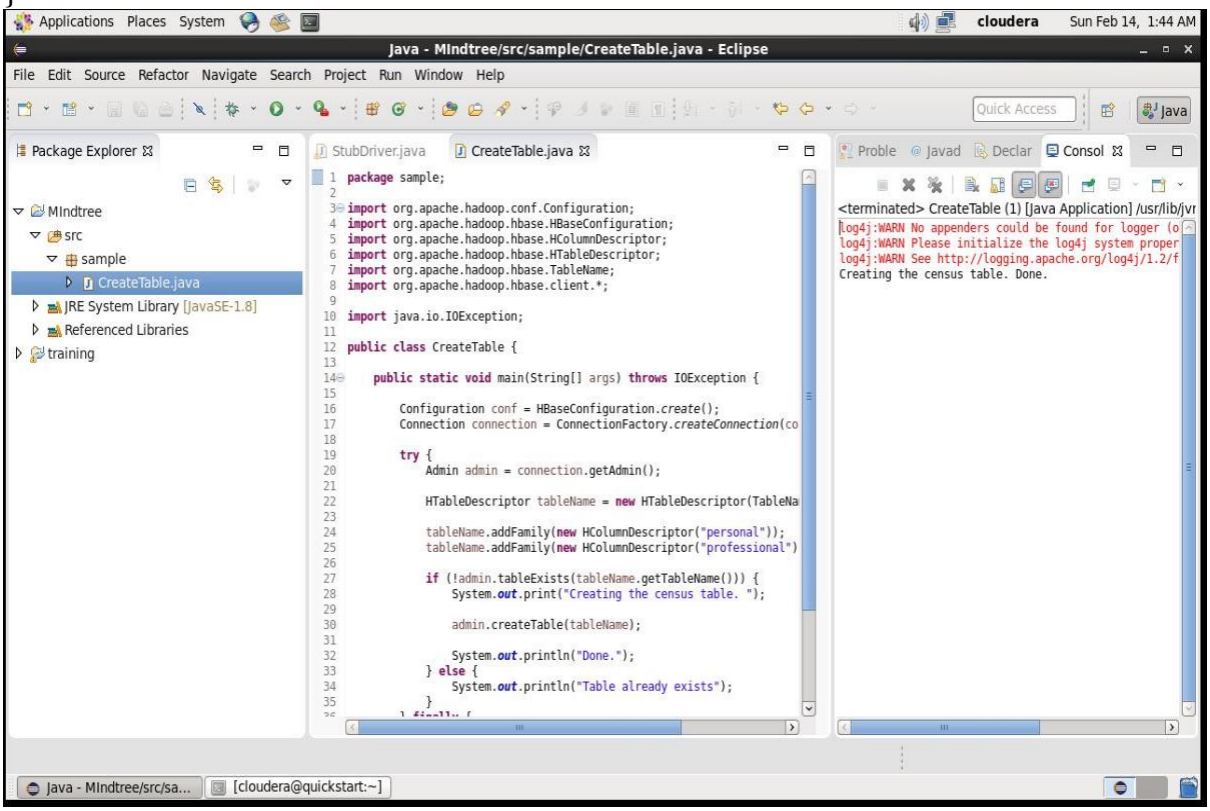
```
hbase(main):014:0> disable 'census'
0 row(s) in 2.3830 seconds

hbase(main):015:0> drop 'census'
0 row(s) in 1.4230 seconds

hbase(main):016:0> list
TABLE
0 row(s) in 0.0780 seconds

=> []
hbase(main):017:0>
```

Able to execute CRUD operations using Hbase Java API

Create : Creating table by using the Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;

import java.io.IOException;

public class CreateTable {

    public static void main(String[] args) throws IOException {

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        try {
```

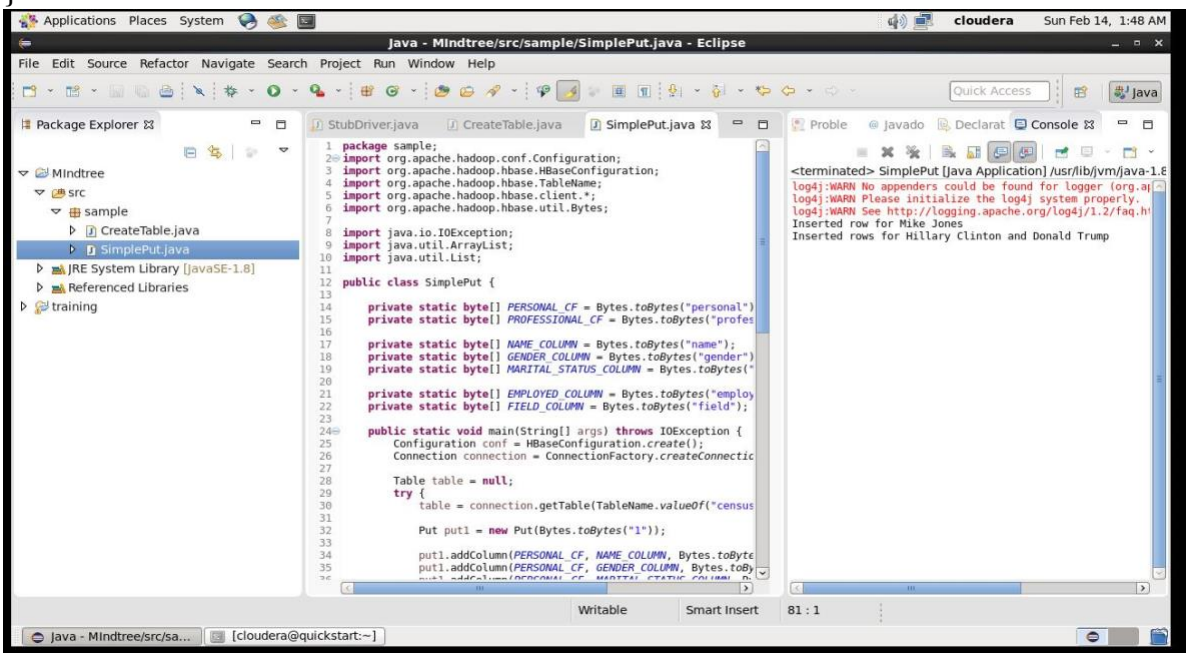
```
        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("census"));

        tableName.addFamily(new HColumnDescriptor("personal"));
        tableName.addFamily(new HColumnDescriptor("professional"));

        if (!admin.tableExists(tableName.getTableName())) {
           System.out.print("Creating the census table. ");

           admin.createTable(tableName);

           System.out.println("Done.");
        } else {
           System.out.println("Table already exists");
        }
      } finally {
        connection.close();
      }

   }
}
```



Put: Put the data in the table by using Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
public class SimplePut {

   private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
   private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");

   private static byte[] NAME_COLUMN = Bytes.toBytes("name");
   private static byte[] GENDER_COLUMN = Bytes.toBytes("gender");
   private static byte[] MARITAL_STATUS_COLUMN = Bytes.toBytes("marital_status");

   private static byte[] EMPLOYED_COLUMN = Bytes.toBytes("employed");
   private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

   public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      Connection connection = ConnectionFactory.createConnection(conf);

      Table table = null;
      try {
         table = connection.getTable(TableName.valueOf("census"));

         Put put1 = new Put(Bytes.toBytes("1"));

         put1.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("Mike Jones"));
         put1.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("male"));
         put1.addColumn(PERSONAL_CF, MARITAL_STATUS_COLUMN,
Bytes.toBytes("unmarried"));
         put1.addColumn(PROFESSIONAL_CF, EMPLOYED_COLUMN,
Bytes.toBytes("yes"));
         put1.addColumn(PROFESSIONAL_CF, FIELD_COLUMN,
Bytes.toBytes("construction"));

         table.put(put1);

         System.out.println("Inserted row for Mike Jones");

         Put put2 = new Put(Bytes.toBytes("2"));
```

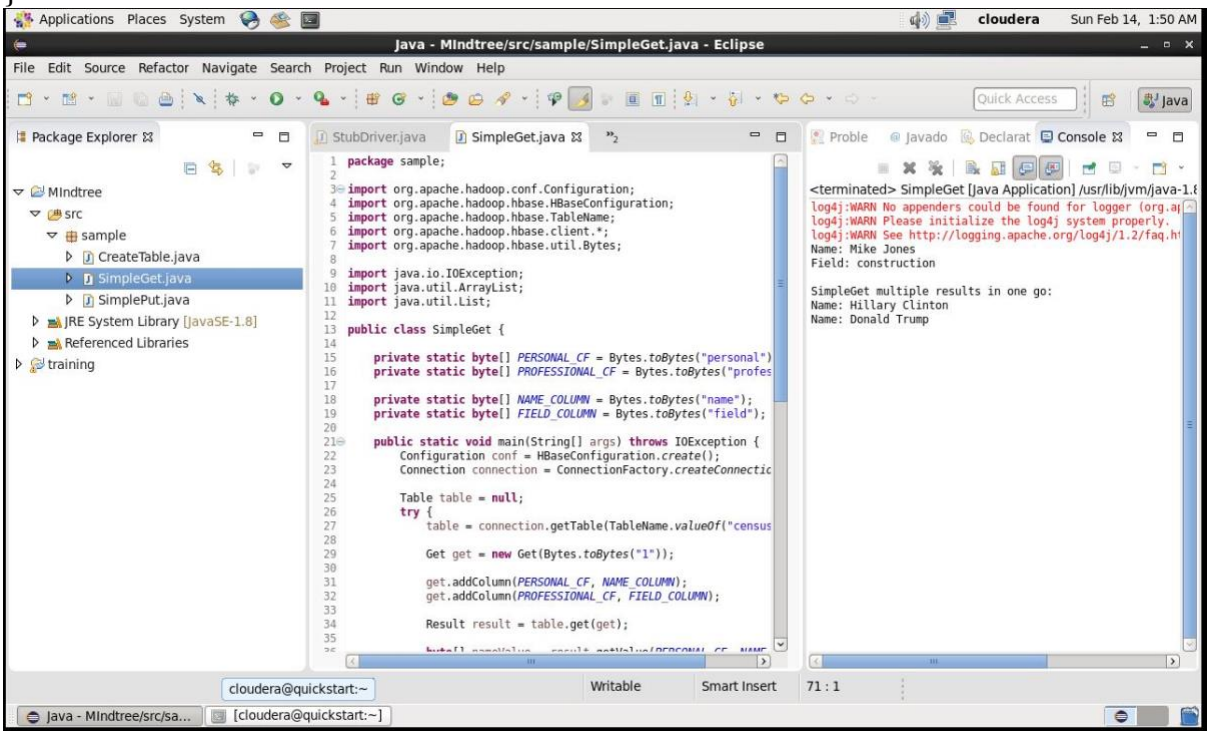
```
        put2.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("Hillary
Clinton"));
        put2.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("female"));
        put2.addColumn(PERSONAL_CF, MARITAL_STATUS_COLUMN,
Bytes.toBytes("married"));
        put2.addColumn(PROFESSIONAL_CF, FIELD_COLUMN, Bytes.toBytes("politics"));

        Put put3 = new Put(Bytes.toBytes("3"));

        put3.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("Donald
Trump"));
        put3.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("male"));
        put3.addColumn(PROFESSIONAL_CF, FIELD_COLUMN, Bytes.toBytes("politics,
real estate"));

        List<Put> putList = new ArrayList<Put>();
        putList.add(put2);
        putList.add(put3);

        table.put(putList);

        System.out.println("Inserted rows for Hillary Clinton and Donald Trump");
      } finally {
        connection.close();
        if (table != null) {
          table.close();
        } }}
}
```



Get: Get the data from the table by using Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class SimpleGet {

   private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
   private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");

   private static byte[] NAME_COLUMN = Bytes.toBytes("name");
   private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

   public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      Connection connection = ConnectionFactory.createConnection(conf);

      Table table = null;
      try {
         table = connection.getTable(TableName.valueOf("census"));

         Get get = new Get(Bytes.toBytes("1"));

         get.addColumn(PERSONAL_CF, NAME_COLUMN);
         get.addColumn(PROFESSIONAL_CF, FIELD_COLUMN);

         Result result = table.get(get);

         byte[] nameValue = result.getValue(PERSONAL_CF, NAME_COLUMN);
         System.out.println("Name: " + Bytes.toString(nameValue));

         byte[] fieldValue = result.getValue(PROFESSIONAL_CF, FIELD_COLUMN);
         System.out.println("Field: " + Bytes.toString(fieldValue));

         System.out.println();
         System.out.println("SimpleGet multiple results in one go:");

         List<Get> getList = new ArrayList<Get>();
         Get get1 = new Get(Bytes.toBytes("2"));
```

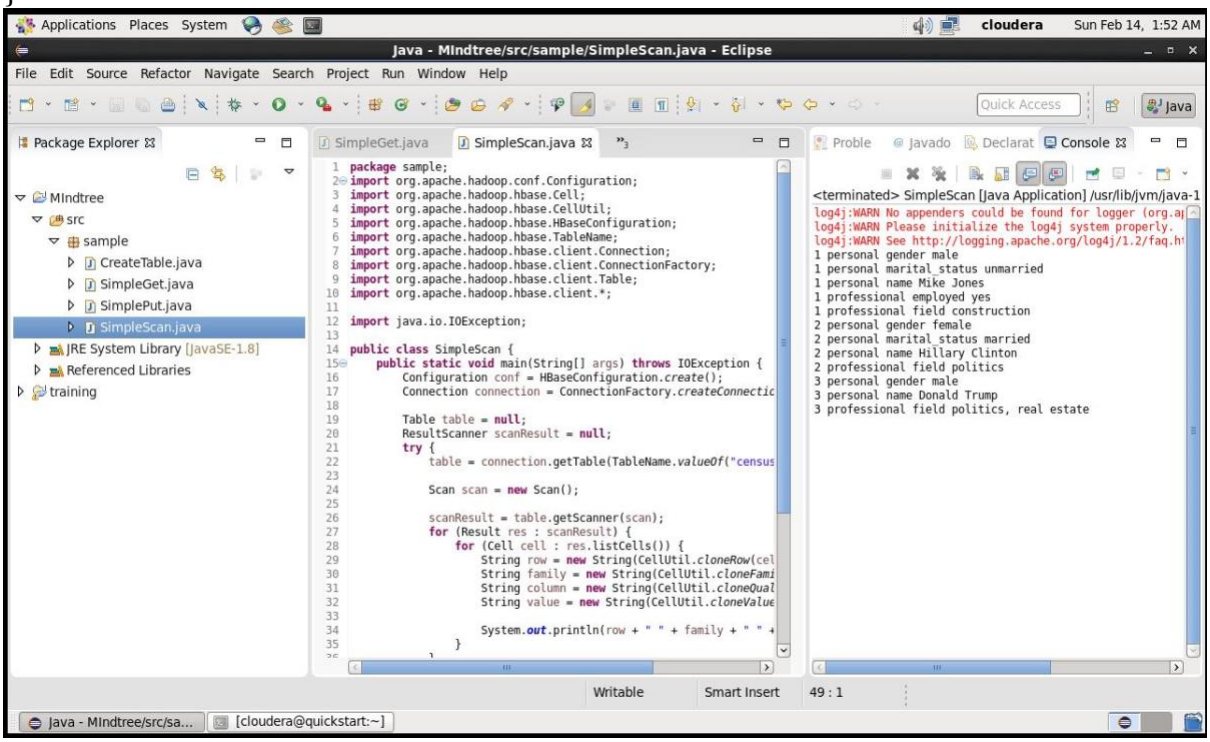
```
        get1.addColumn(PERSONAL_CF, NAME_COLUMN);

        Get get2 = new Get(Bytes.toBytes("3"));
        get1.addColumn(PERSONAL_CF, NAME_COLUMN);

        getList.add(get1);
        getList.add(get2);

        Result[] results = table.get(getList);

        for (Result res : results) {
           nameValue = res.getValue(PERSONAL_CF, NAME_COLUMN);
           System.out.println("Name: " + Bytes.toString(nameValue));
        }
      } finally {
        connection.close();
        if (table != null) {
           table.close();
        }
      }
   }

}
```



Scan: Displaying the data by using 'Scan' with the help of Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.client.*;

import java.io.IOException;

public class SimpleScan {
   public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      Connection connection = ConnectionFactory.createConnection(conf);

      Table table = null;
      ResultScanner scanResult = null;
      try {
         table = connection.getTable(TableName.valueOf("census"));

         Scan scan = new Scan();

         scanResult = table.getScanner(scan);
         for (Result res : scanResult) {
            for (Cell cell : res.listCells()) {
               String row = new String(CellUtil.cloneRow(cell));
               String family = new String(CellUtil.cloneFamily(cell));
               String column = new String(CellUtil.cloneQualifier(cell));
               String value = new String(CellUtil.cloneValue(cell));

               System.out.println(row + " " + family + " " + column + " " + value);
            }
         }
      } finally {
         connection.close();
         if (table != null) {
            table.close();
         }
         if (scanResult != null) {
            scanResult.close();
         }
      }
```

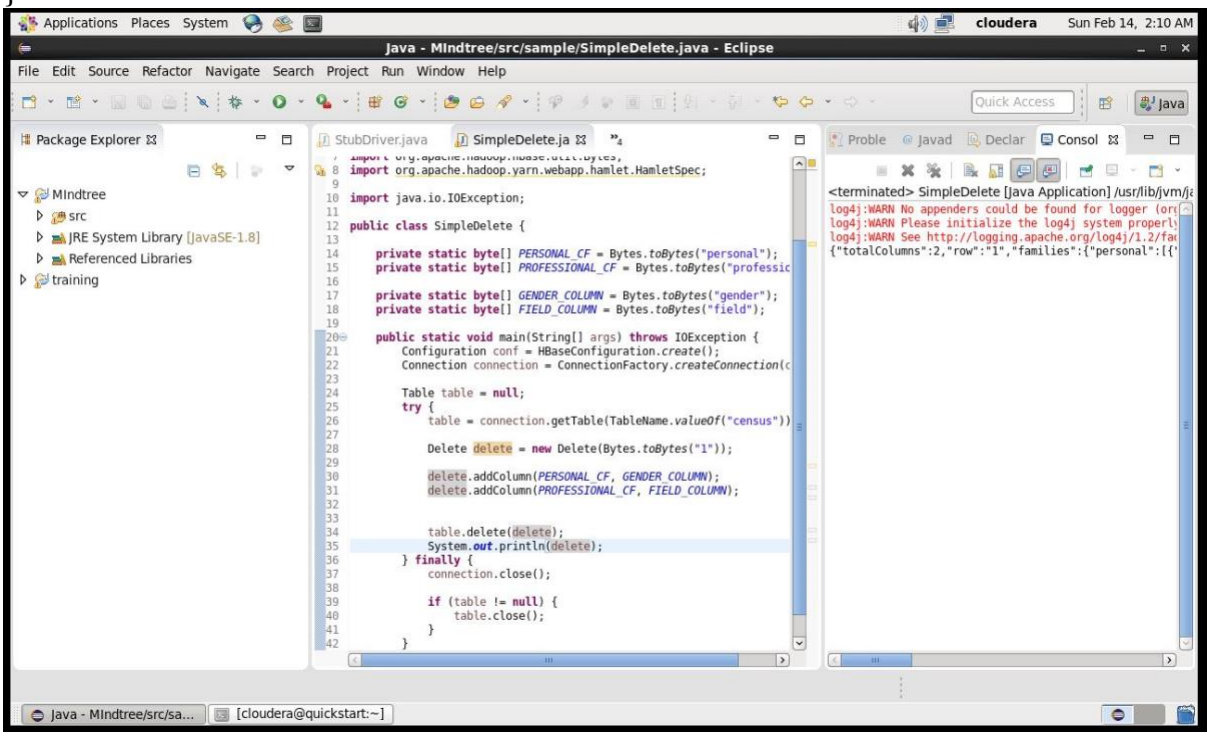
```
    }
}
```



Delete: Delete the data by using Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.yarn.webapp.hamlet.HamletSpec;

import java.io.IOException;

public class SimpleDelete {

    private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
    private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");

    private static byte[] GENDER_COLUMN = Bytes.toBytes("gender");
    private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
```

```
        Table table = null;
        try {
            table = connection.getTable(TableName.valueOf("census"));

            Delete delete = new Delete(Bytes.toBytes("1"));

            delete.addColumn(PERSONAL_CF, GENDER_COLUMN);
            delete.addColumn(PROFESSIONAL_CF, FIELD_COLUMN);

            table.delete(delete);
        } finally {
            connection.close();

            if (table != null) {
                table.close();
            }
        }
    }
}
```



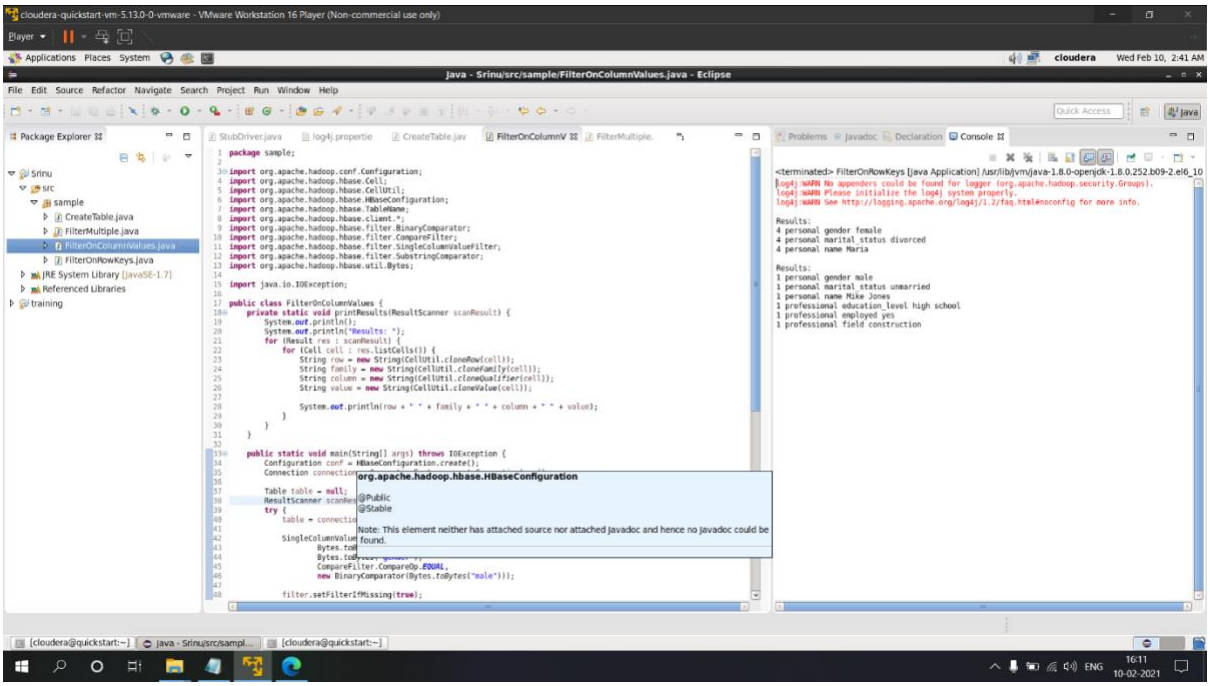
Able to Filter rows based on conditions:

There are Three filter rowConditions :
1) FilterOnColumns
2) FilterOnRowKeys
3) FilterMultiple

FilterOnColumns:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.BinaryComparator;
import org.apache.hadoop.hbase.filter.CompareFilter;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.filter.SubstringComparator;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;

public class FilterOnColumnValues {
   private static void printResults(ResultScanner scanResult) {
      System.out.println();
      System.out.println("Results: ");
      for (Result res : scanResult) {
         for (Cell cell : res.listCells()) {
            String row = new String(CellUtil.cloneRow(cell));
            String family = new String(CellUtil.cloneFamily(cell));
            String column = new String(CellUtil.cloneQualifier(cell));
            String value = new String(CellUtil.cloneValue(cell));

            System.out.println(row + " " + family + " " + column + " " + value);
         }
      }
   }

   public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      Connection connection = ConnectionFactory.createConnection(conf);

      Table table = null;
      ResultScanner scanResult = null;
      try {
         table = connection.getTable(TableName.valueOf("census"));
```

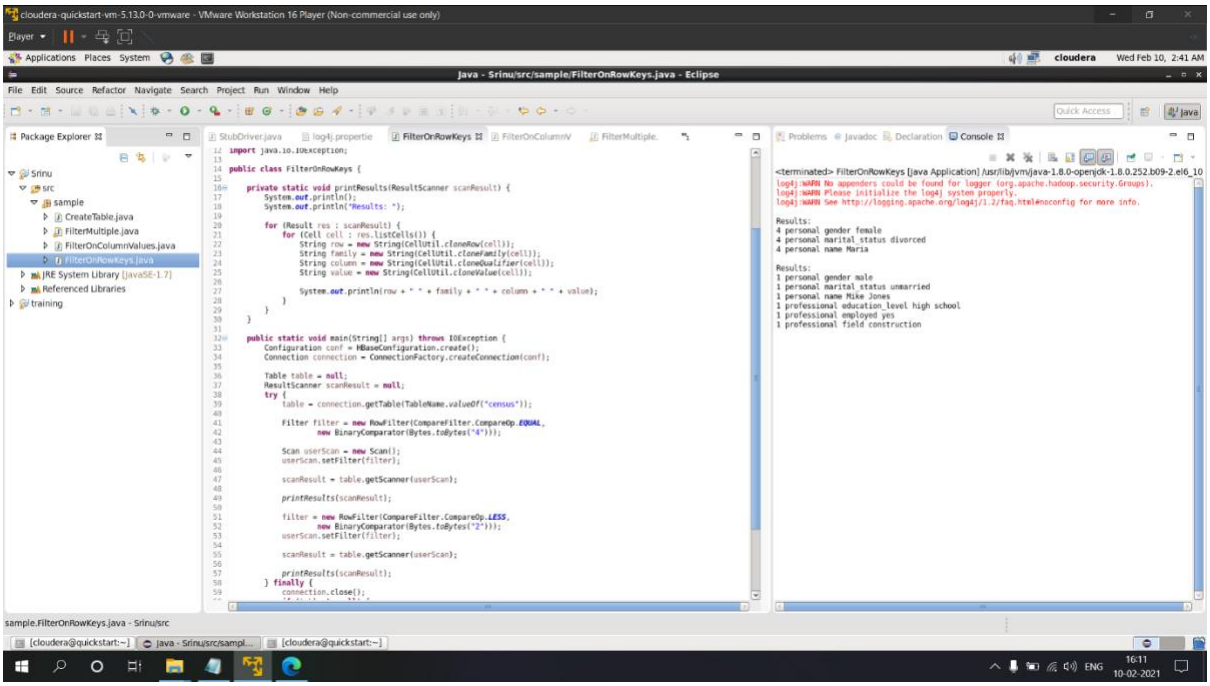
```
        SingleColumnValueFilter filter = new SingleColumnValueFilter(
                Bytes.toBytes("personal"),
                Bytes.toBytes("gender"),
                CompareFilter.CompareOp.EQUAL,
                new BinaryComparator(Bytes.toBytes("male")));

        filter.setFilterIfMissing(true);

        Scan userScan = new Scan();
        userScan.setFilter(filter);

        scanResult = table.getScanner(userScan);

        printResults(scanResult);

        filter = new SingleColumnValueFilter(
                Bytes.toBytes("personal"),
                Bytes.toBytes("name"),
                CompareFilter.CompareOp.EQUAL,
                new SubstringComparator("Jones"));

        userScan.setFilter(filter);
        scanResult = table.getScanner(userScan);

        printResults(scanResult);

      } finally {
        connection.close();
        if (table != null) {
          table.close();
        }
        if (scanResult != null) {
          scanResult.close();
        }
      }
    }
}
```



FilterOnRowKeys:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.*;
import org.apache.hadoop.hbase.filter.RowFilter;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;

public class FilterOnRowKeys {

   private static void printResults(ResultScanner scanResult) {
      System.out.println();
      System.out.println("Results: ");

      for (Result res : scanResult) {
         for (Cell cell : res.listCells()) {
            String row = new String(CellUtil.cloneRow(cell));
            String family = new String(CellUtil.cloneFamily(cell));
            String column = new String(CellUtil.cloneQualifier(cell));
```

```
            String value = new String(CellUtil.cloneValue(cell));

            System.out.println(row + " " + family + " " + column + " " + value);
         }
      }
   }

   public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      Connection connection = ConnectionFactory.createConnection(conf);

      Table table = null;
      ResultScanner scanResult = null;
      try {
         table = connection.getTable(TableName.valueOf("census"));

         Filter filter = new RowFilter(CompareFilter.CompareOp.EQUAL,
               new BinaryComparator(Bytes.toBytes("4")));

         Scan userScan = new Scan();
         userScan.setFilter(filter);

         scanResult = table.getScanner(userScan);

         printResults(scanResult);

         filter = new RowFilter(CompareFilter.CompareOp.LESS,
               new BinaryComparator(Bytes.toBytes("2")));
         userScan.setFilter(filter);

         scanResult = table.getScanner(userScan);

         printResults(scanResult);
      } finally {
         connection.close();
         if (table != null) {
            table.close();
         }
         if (scanResult != null) {
            scanResult.close();
         }
      }
   }
}
```

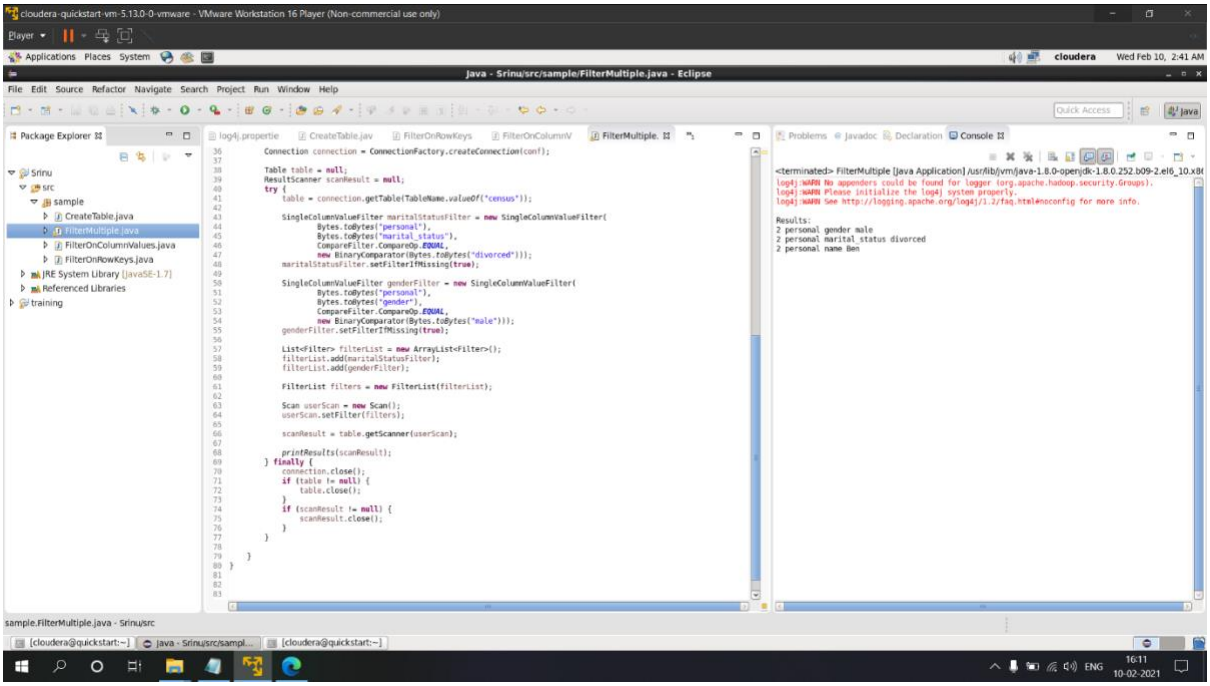


FilterMultiple:

```
import org.apache.commons.lang.time.DateUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class FilterMultiple {

    private static void printResults(ResultScanner scanResult) {
        System.out.println();
        System.out.println("Results: ");
        for (Result res : scanResult) {
            for (Cell cell : res.listCells()) {
                String row = new String(CellUtil.cloneRow(cell));
                String family = new String(CellUtil.cloneFamily(cell));
```

```
            String column = new String(CellUtil.cloneQualifier(cell));
            String value = new String(CellUtil.cloneValue(cell));

            System.out.println(row + " " + family + " " + column + " " + value);
        }
    }
}


public static void main(String[] args) throws IOException {
    Configuration conf = HBaseConfiguration.create();
    Connection connection = ConnectionFactory.createConnection(conf);

    Table table = null;
    ResultScanner scanResult = null;
    try {
        table = connection.getTable(TableName.valueOf("census"));

        SingleColumnValueFilter maritalStatusFilter = new SingleColumnValueFilter(
                Bytes.toBytes("personal"),
                Bytes.toBytes("marital_status"),
                CompareFilter.CompareOp.EQUAL,
                new BinaryComparator(Bytes.toBytes("divorced")));
        maritalStatusFilter.setFilterIfMissing(true);

        SingleColumnValueFilter genderFilter = new SingleColumnValueFilter(
                Bytes.toBytes("personal"),
                Bytes.toBytes("gender"),
                CompareFilter.CompareOp.EQUAL,
                new BinaryComparator(Bytes.toBytes("male")));
        genderFilter.setFilterIfMissing(true);

        List<Filter> filterList = new ArrayList<Filter>();
        filterList.add(maritalStatusFilter);
        filterList.add(genderFilter);

        FilterList filters = new FilterList(filterList);

        Scan userScan = new Scan();
        userScan.setFilter(filters);

        scanResult = table.getScanner(userScan);

        printResults(scanResult);
    } finally {
        connection.close();
```

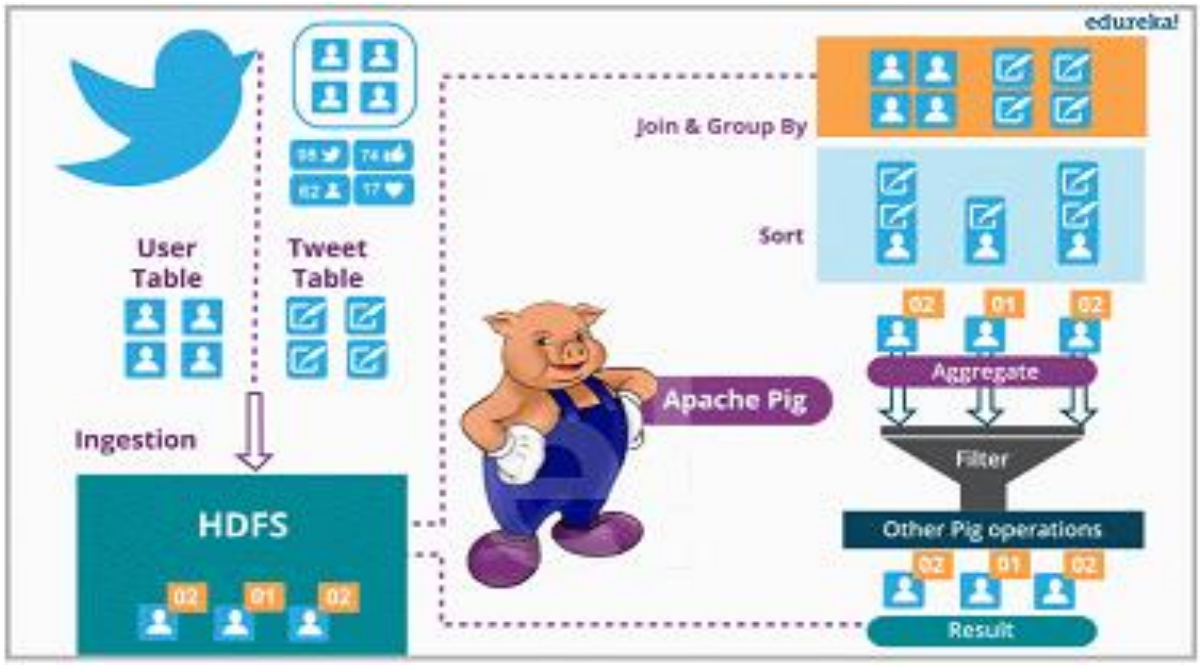
```
            if (table != null) {
                table.close();
            }
            if (scanResult != null) {
                scanResult.close();
            }
        }

    }
}
```



APACHE PIG

INTRODUCTION TO APACHE PIG

Apache Pig is a platform, used to analyze large data sets representing them as data flows. It is designed to provide an abstraction over MapReduce, reducing the complexities of writing a MapReduce program. We can perform data manipulation operations very easily in Hadoop using Apache Pig.



Features of Apache pig

- Pig enables programmers to write complex data transformations without knowing Java.
- Apache Pig has two main components – the Pig Latin language and the Pig Run-time Environment, in which Pig Latin programs are executed.
- For Big Data Analytics, Pig gives a simple data flow language known as Pig Latin which has functionalities similar to SQL like join, filter, limit etc.
- Developers who are working with scripting languages and SQL, leverages Pig Latin. This gives developers ease of programming with Apache Pig. Pig Latin provides various built-

in operators like join, sort, filter, etc to read, write, and process large data sets. Thus it is evident, Pig has a rich set of operators.

Starting with Apache Pig

There are three ways to execute the Pig script:

- **Grunt Shell:** This is Pig's interactive shell provided to execute all Pig Scripts.
- **Script File:** Write all the Pig commands in a script file and execute the Pig script file. This is executed by the Pig Server.
- **Embedded Script:** If some functions are unavailable in built-in operators, we can programmatically create User Defined Functions to bring that functionalities using other languages like Java, Python, Ruby, etc. and embed it in Pig Latin Script file**.** Then, execute that script file.

We will be using grunt shell to execute pig scripts.

To enter into grunt shell, use

```
pig –x local
```

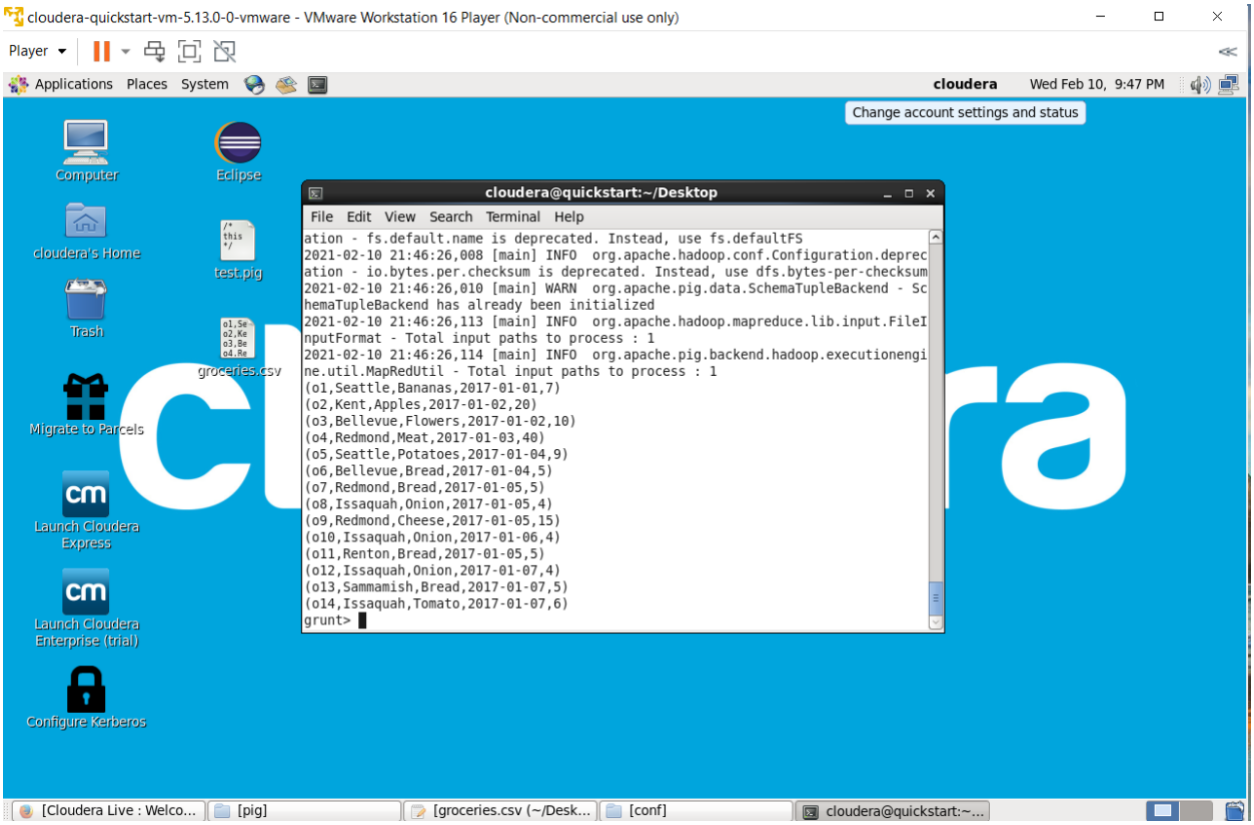
Local Mode

- It executes in a single JVM and is used for development experimenting and prototyping.
- Here, files are installed and run using localhost.
- The local mode works on a local file system. The input and output data stored in the local file system.

Mapreduce mode

```
pig -x mapreduce
```

- It is the default mode.
- In this Pig renders Pig Latin into MapReduce jobs and executes them on the cluster.
- It can be executed against semi-distributed or fully distributed Hadoop installation.
- Here, the input and output data are present on HDFS.



Using pig –x local command , we enter into grunt shell. Now we are ready to run pig scripts.

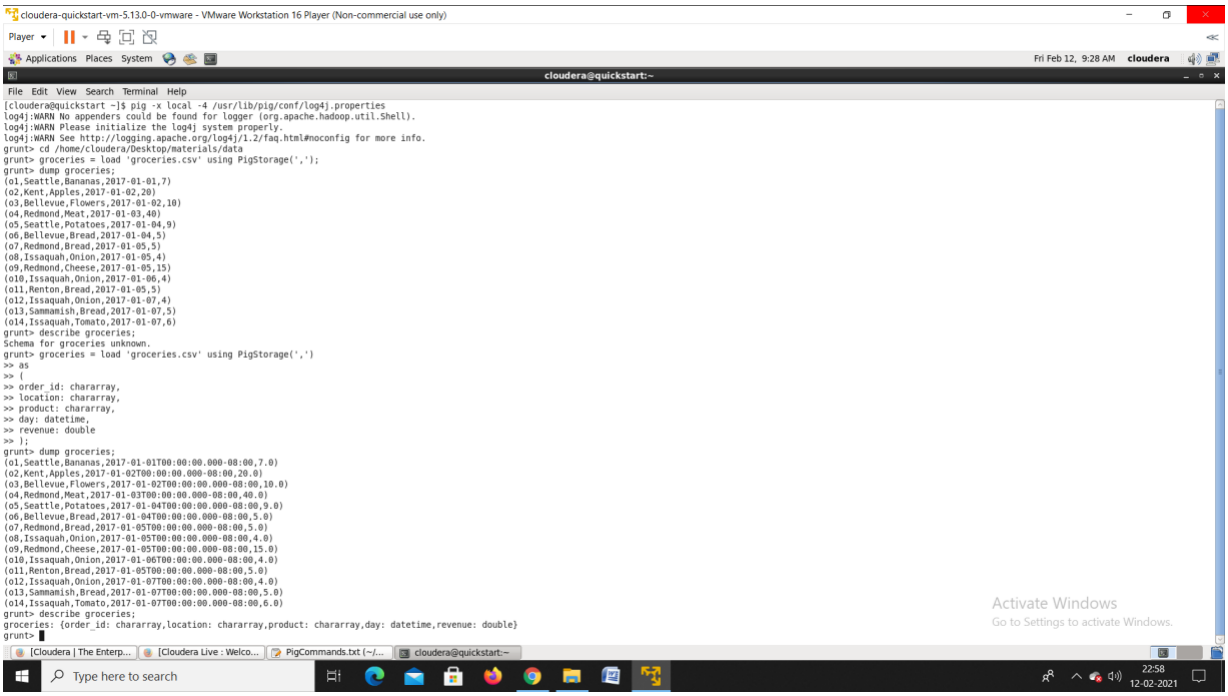
Pig commands

1. Loading data from files: the below command loads a csv file into groceries.

```
groceries = load 'groceries.csv' using PigStorage(',')
as
(
order_id: chararray,
location: chararray,
product: chararray,
day: datetime,
revenue: double
);
```

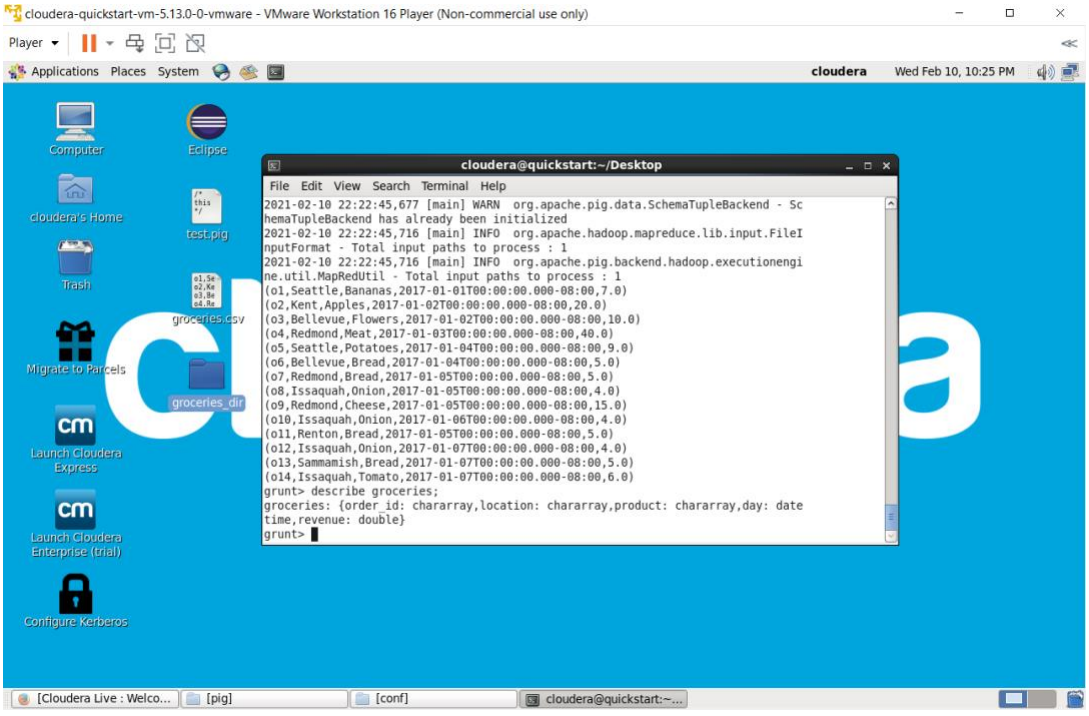
2. dumping the result into terminal

```
dump groceries;
```



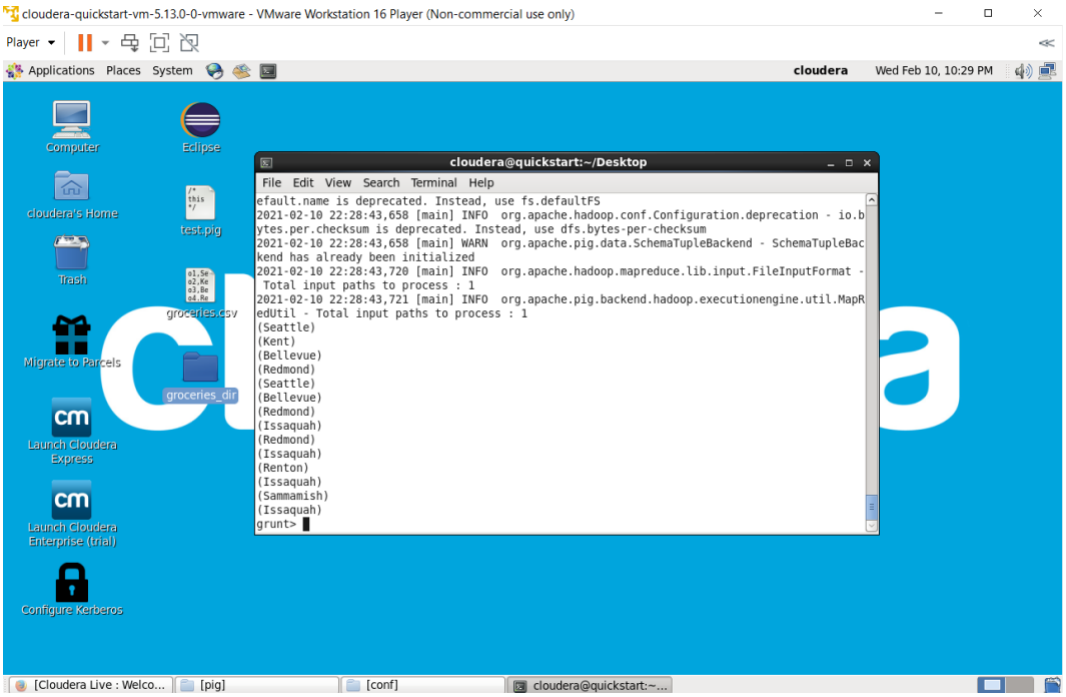
3. to see the schema structure

describe groceries;



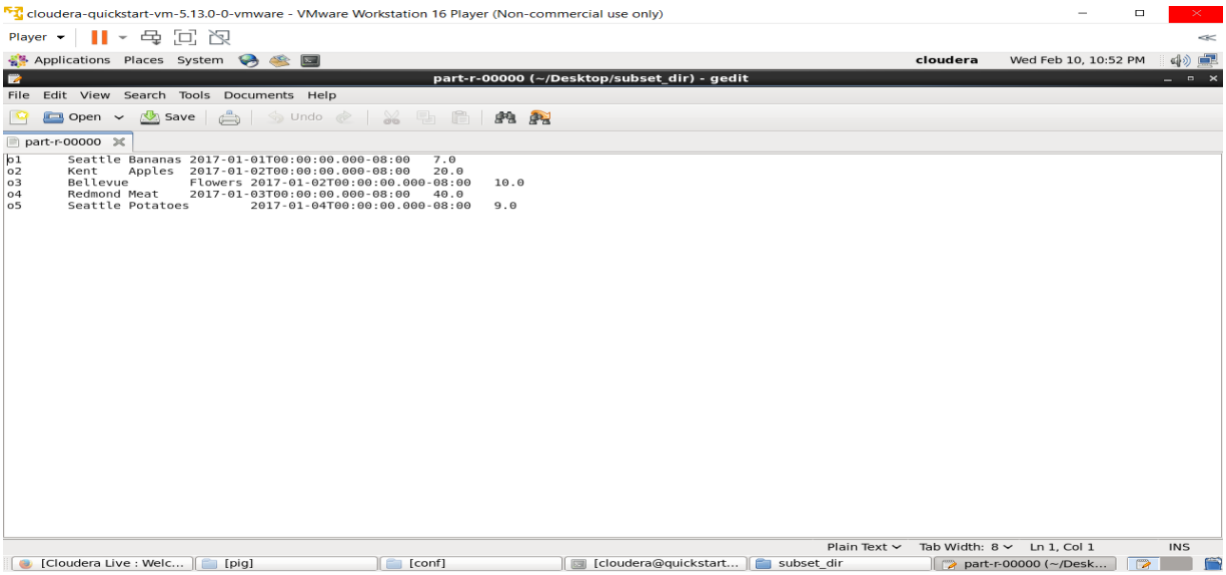
4. using foreach command to display the second column values(by default the column count starts from 0).

```
locations = foreach groceries generate $1
dump locations;
```



5. storing the top five rows into another directory(subset_dir)

```
groceries_subset = limit groceries 5;
store groceries_subset into 'subset_dir' using PigStorage();
```

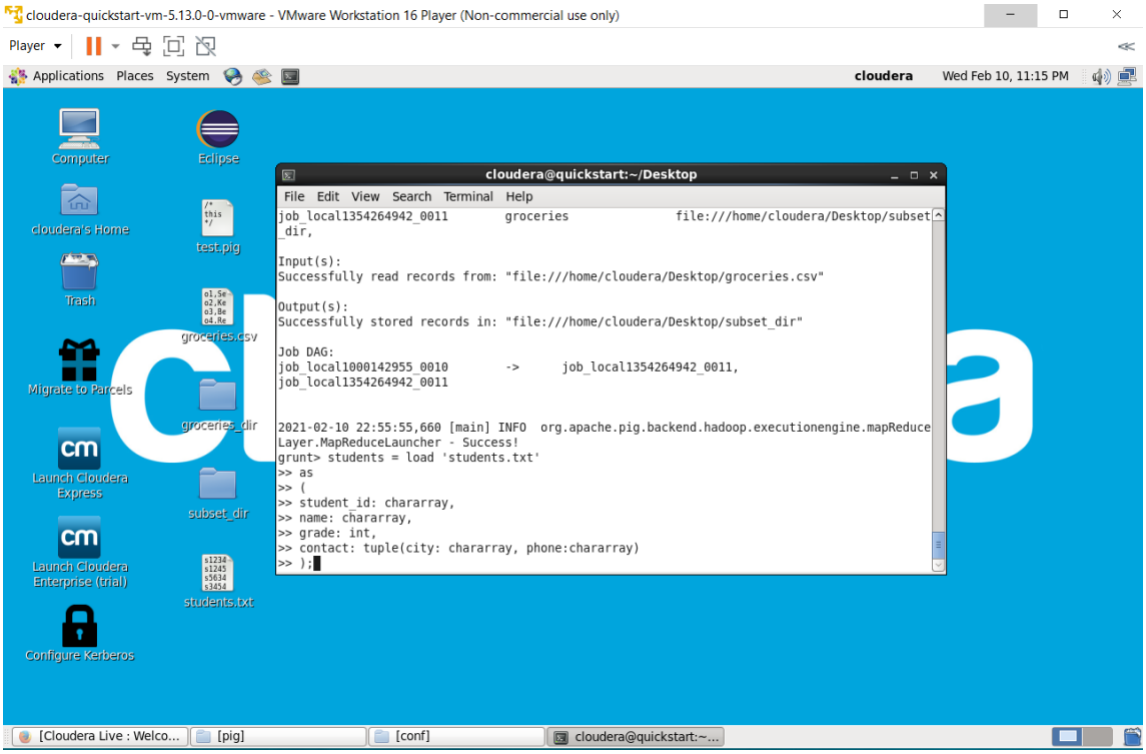


6. using tuple

Tuple is an ordered set of fields which may contain different data types for each field. You can understand it as the records stored in a row in a relational database. A Tuple is a set of cells from a single row as shown in the above image. The elements inside a tuple does not necessarily need to have a schema attached to it. A tuple is represented by '()' symbol.

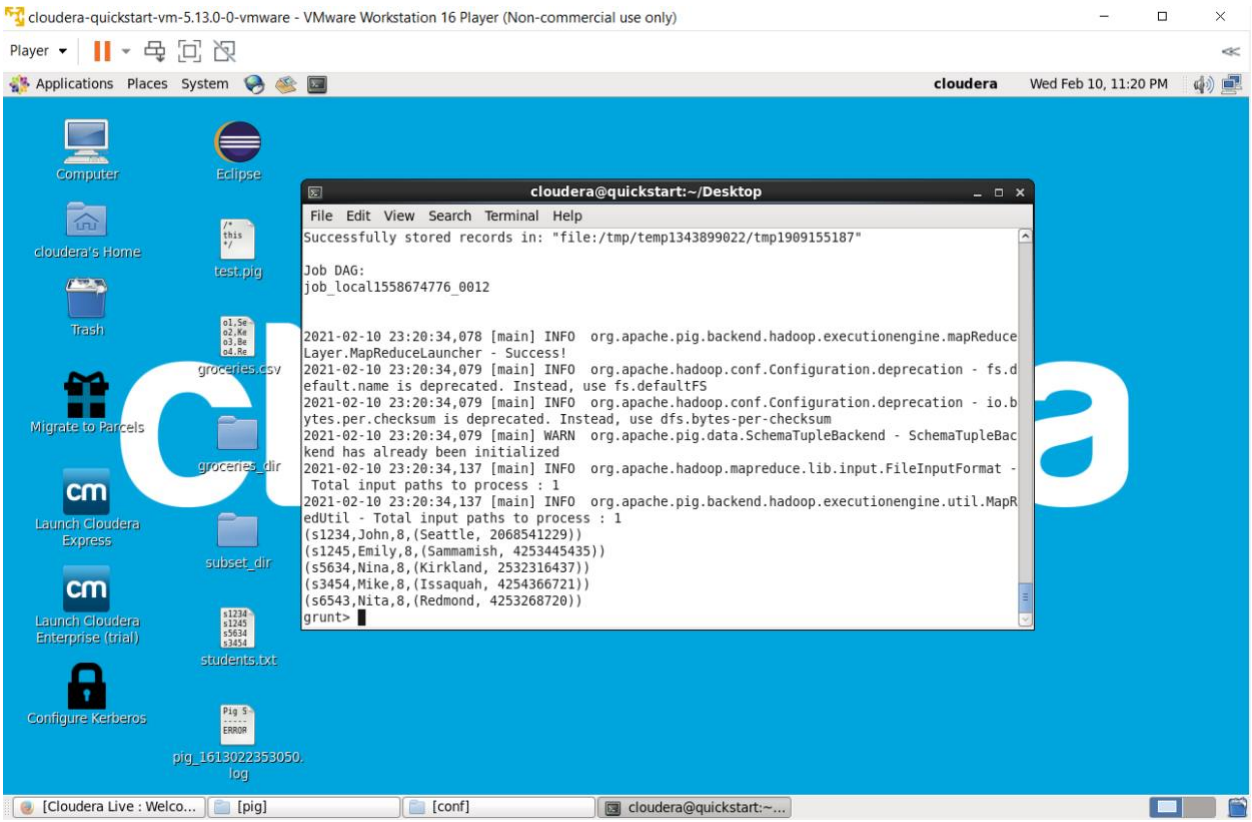
```
students = load 'students.txt'
as
(
student_id: chararray,
name: chararray,
grade: int,
contact: tuple(city: chararray, phone: chararray)
);
```

This creates tuple with city and phone with chararray datatype.



dump students;

describe students;



7. using bag

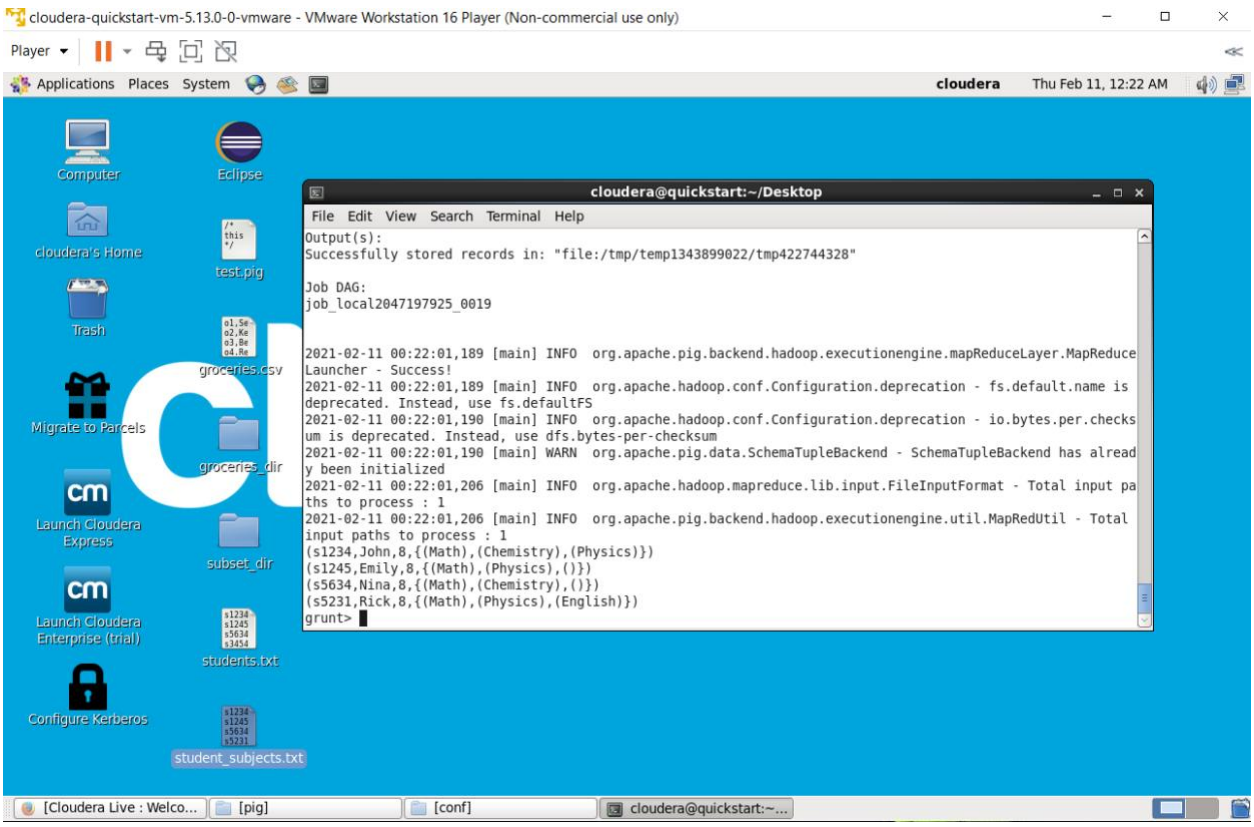
A bag is a collection of a set of tuples and these tuples are subset of rows or entire rows of a table.

A bag can contain duplicate tuples, and it is not mandatory that they need to be unique.

The bag has a flexible schema i.e. tuples within the bag can have different number of fields. A bag can also have tuples with different data types.

A bag is represented by '{}' symbol.

```
student_subjects = load 'student_subjects.txt'
as
(
student_id: chararray,
name: chararray,
grade: int,
subject1: chararray,
subject2: chararray,
subject3: chararray
);
student_subject_bag = foreach student_subjects generate $0, $1, $2, TOBAG($3, $4, $5);
```

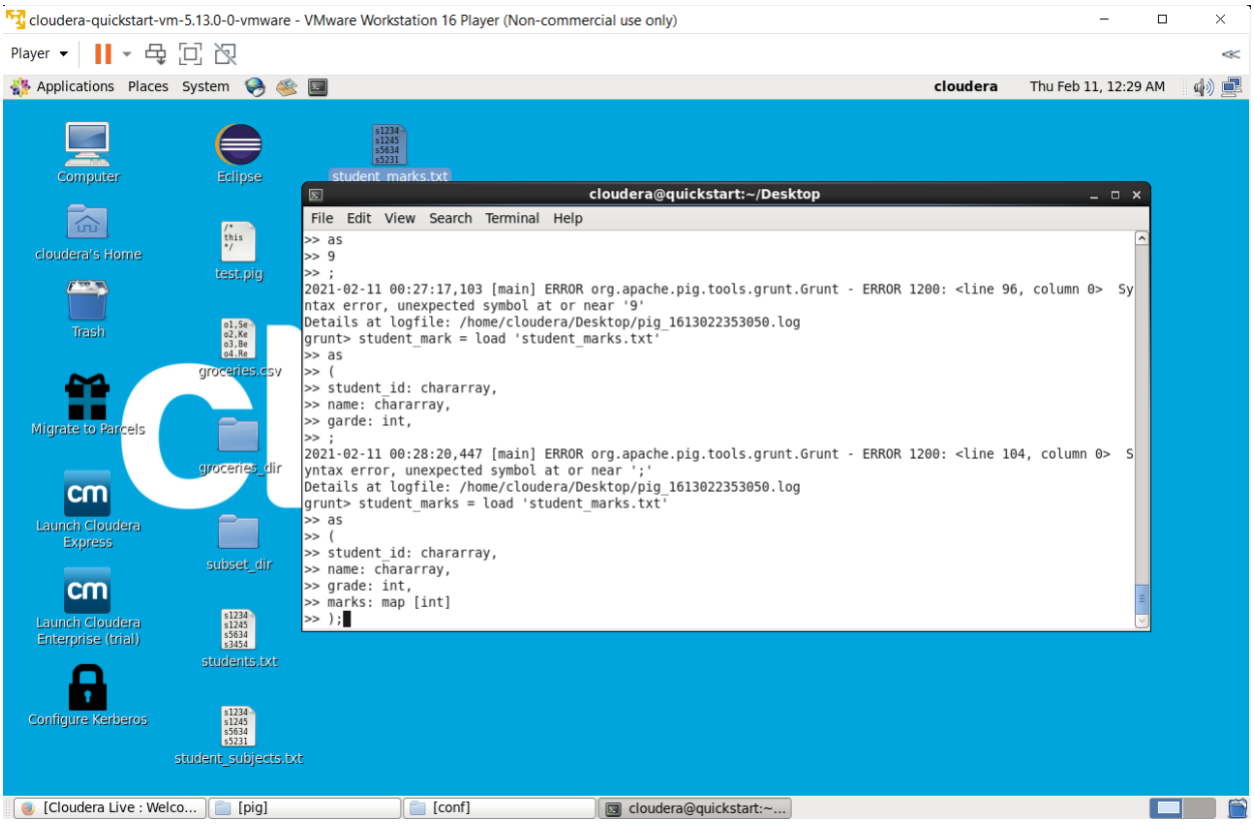


8. using map

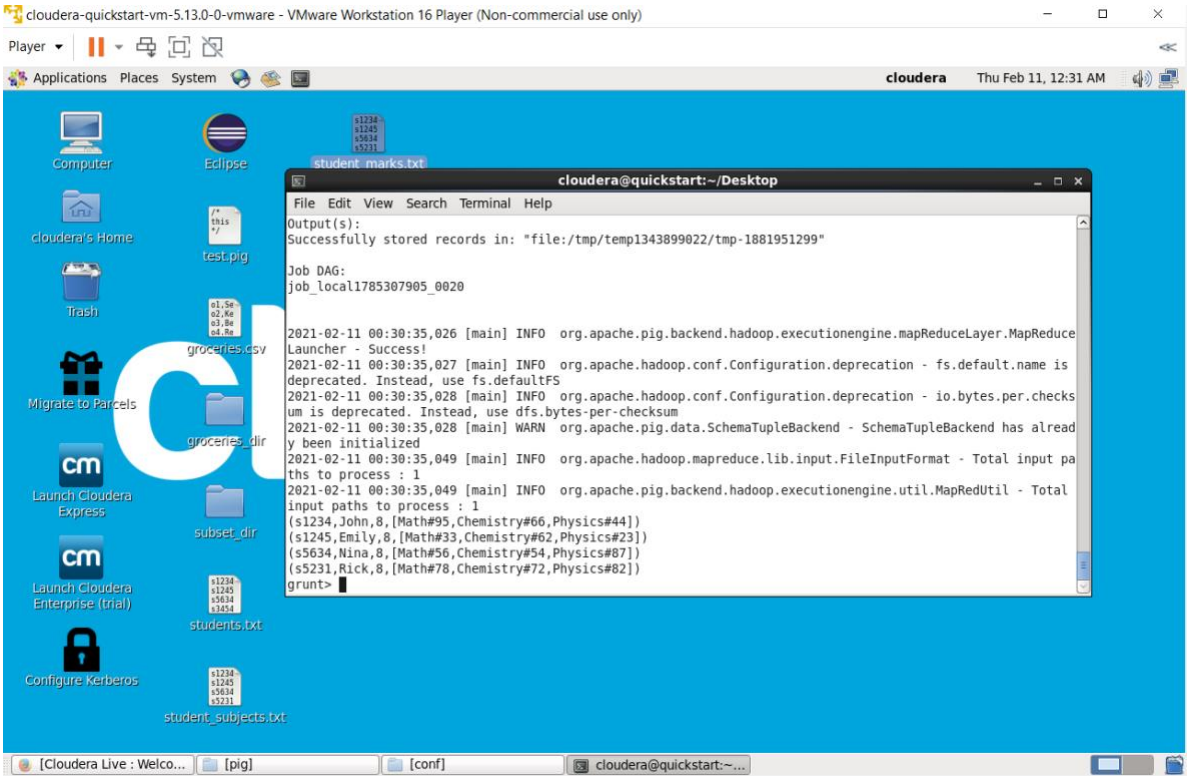
A map is key-value pairs used to represent data elements. The key must be a chararray [] and should be unique like column name, so it can be indexed and value associated with it can be accessed on basis of the keys. The value can be of any data type.

Maps are represented by '[]' symbol and key-value are separated by '#' symbol, as you can see in the above image.

```
student_marks = load 'student_marks.txt'
as
(
student_id: chararray,
name: chararray,
grade: int,
marks: map [int]
);
```



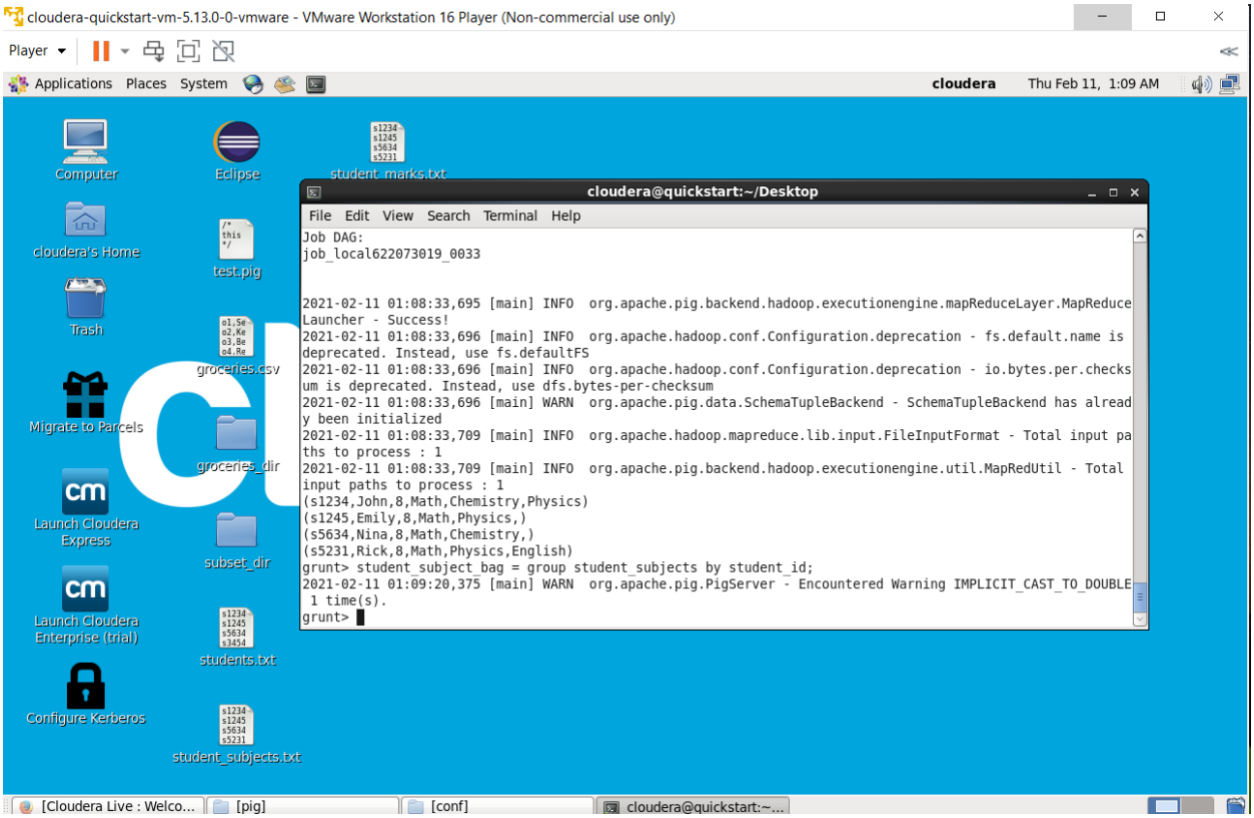
dump student_marks;



9. using group

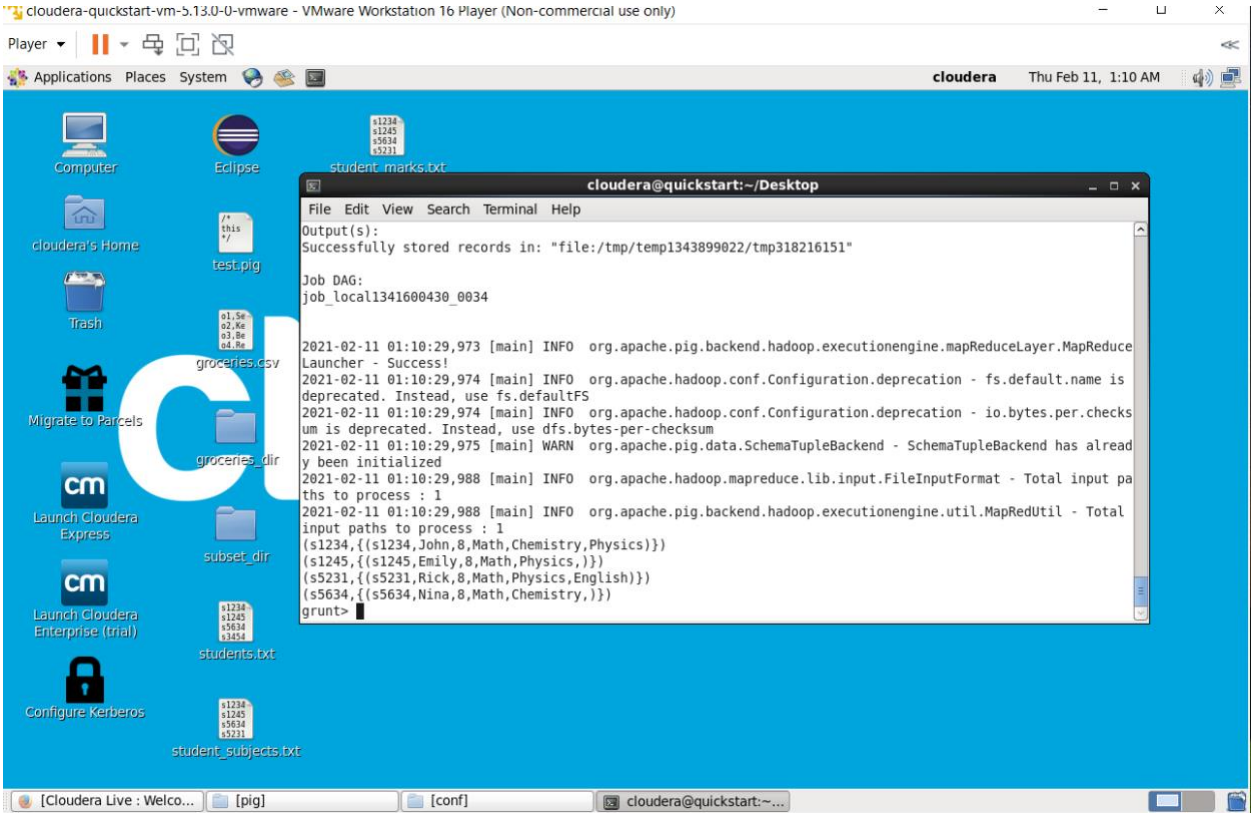
Group command is used to put the items into a single container based on conditions. The below command groups by student_id and stores in student_subject_bag.

student_subject_bag = group student_subjects by student_id;



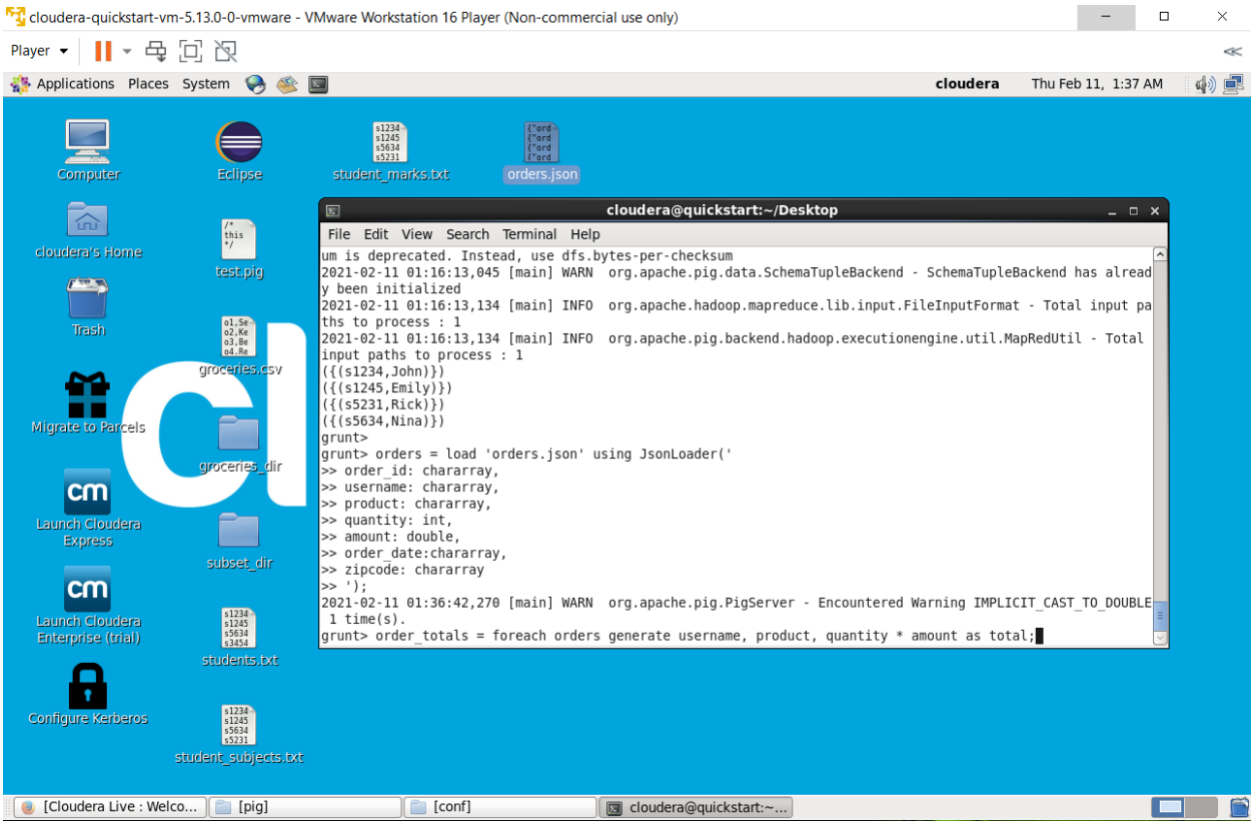
describe student_subject_bag;

dump student_subject_bag;



10. loading json file: this load a json file into orders.

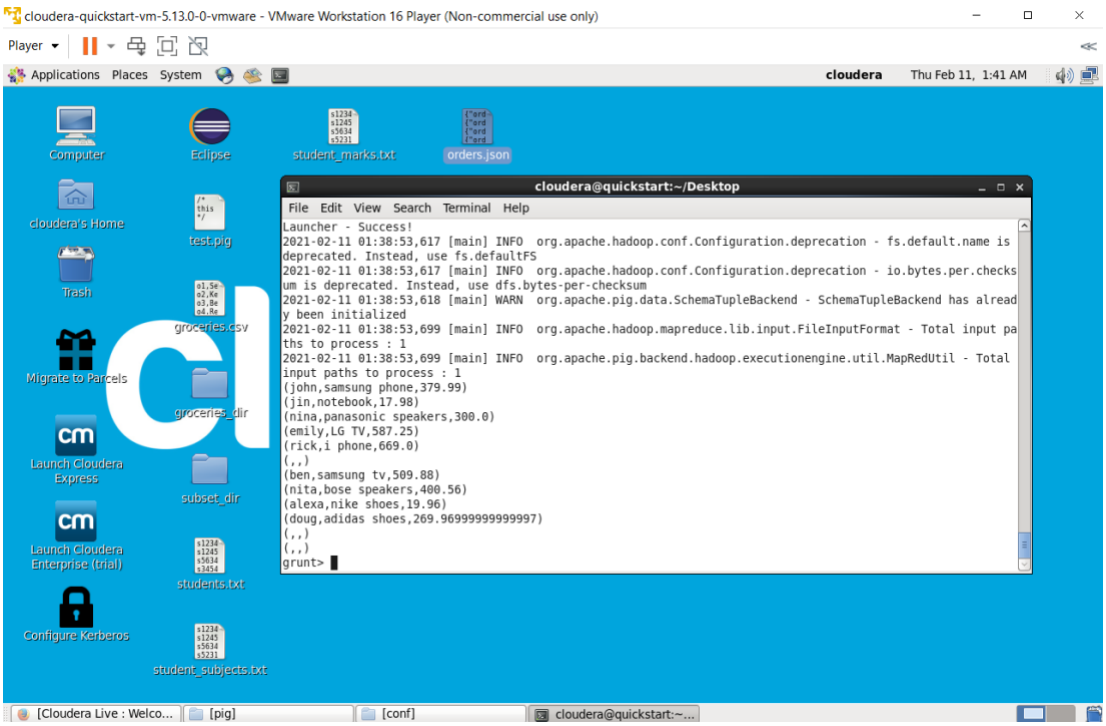
```
orders = load 'orders.json' using JsonLoader('
order_id: chararray,
username: chararray,
product: chararray,
quantity: int,
amount: double,
order_date:chararray,
zipcode: chararray
');
```



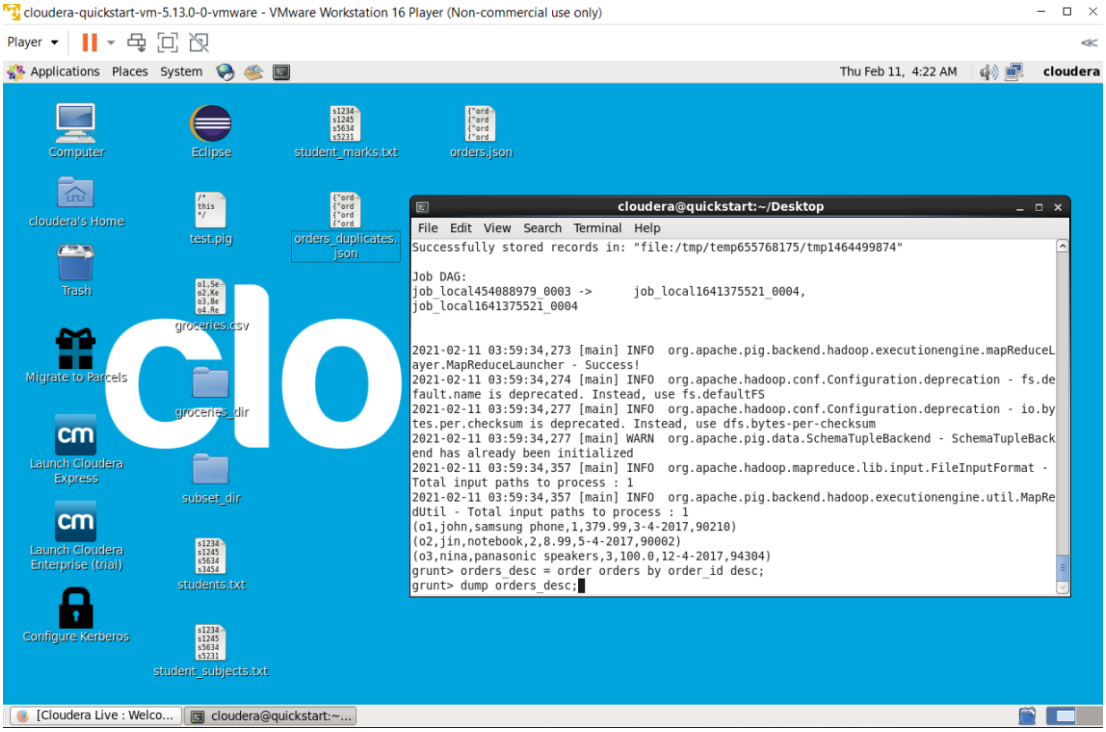
11. calculates the total quantity

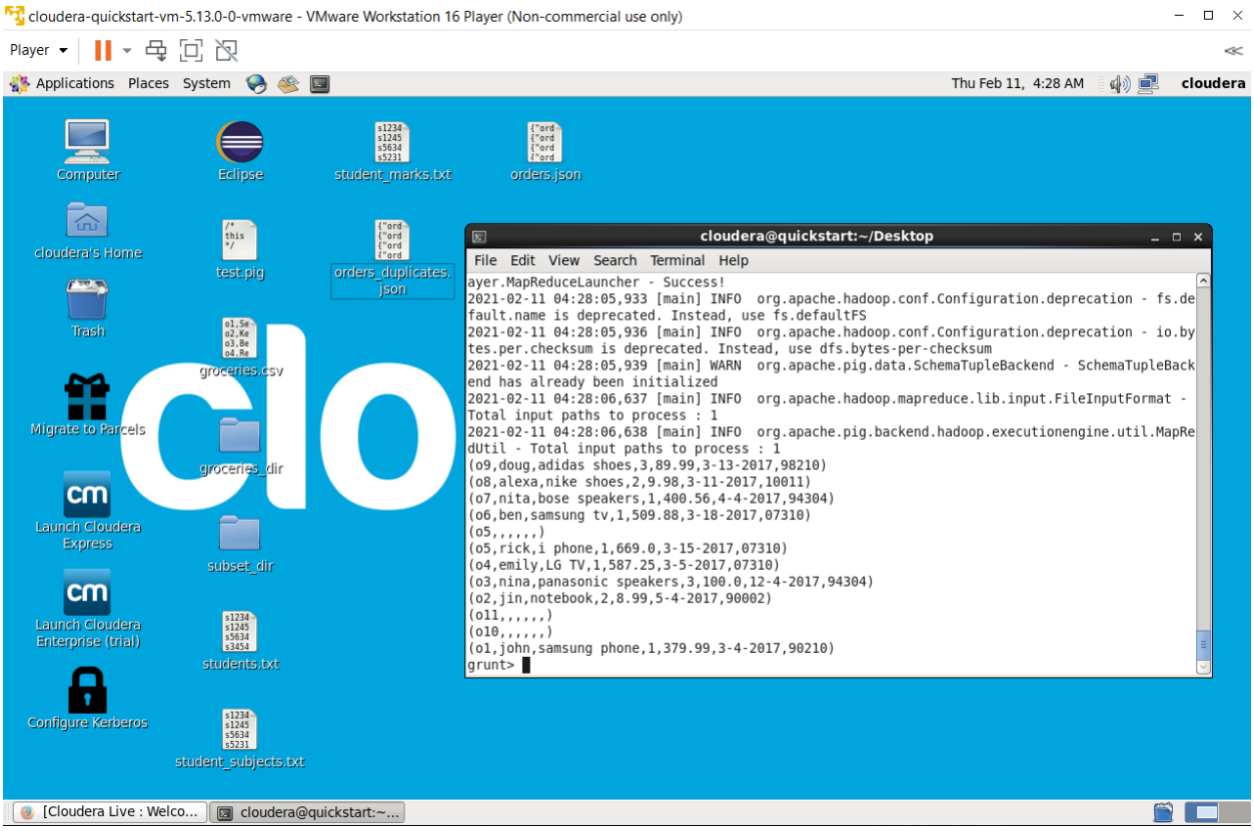
order_totals = foreach orders generate username, product, quantity * amount as total;

dump order_totals;



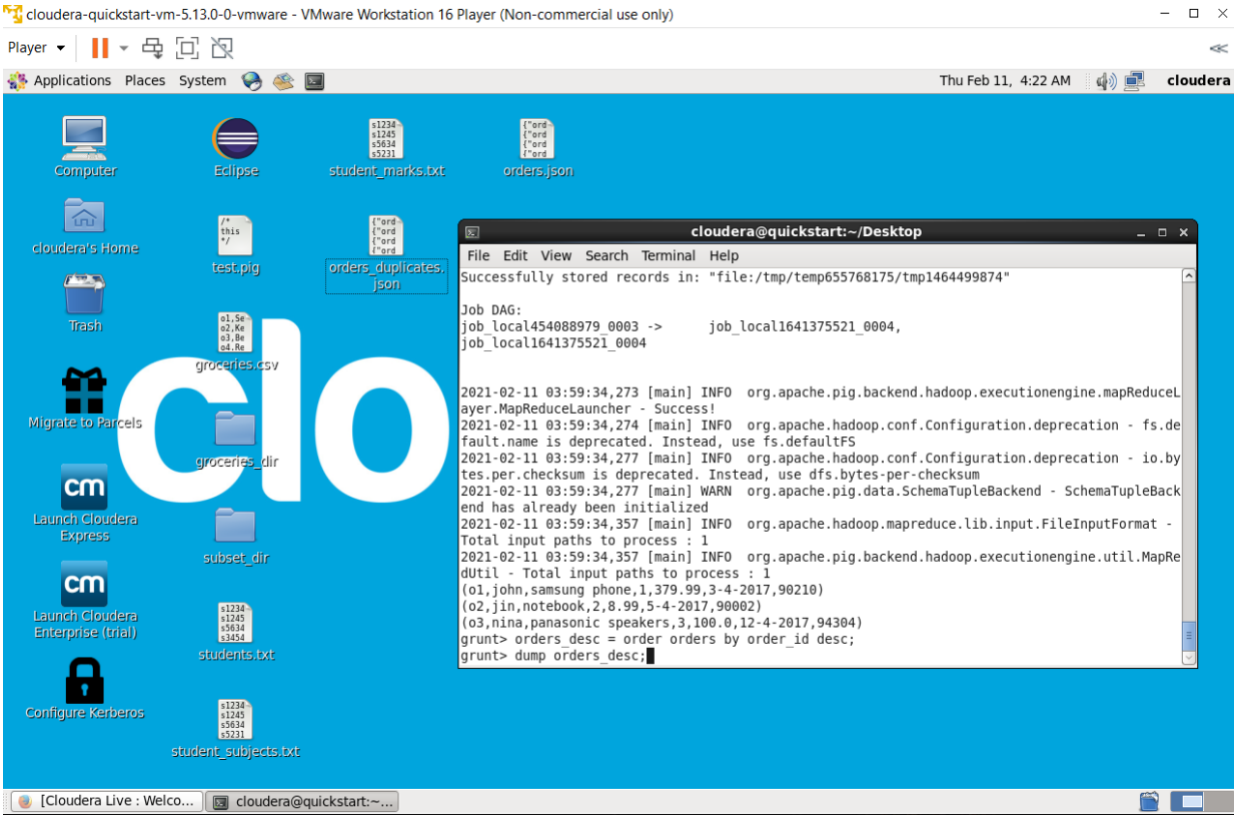
Using order keyword

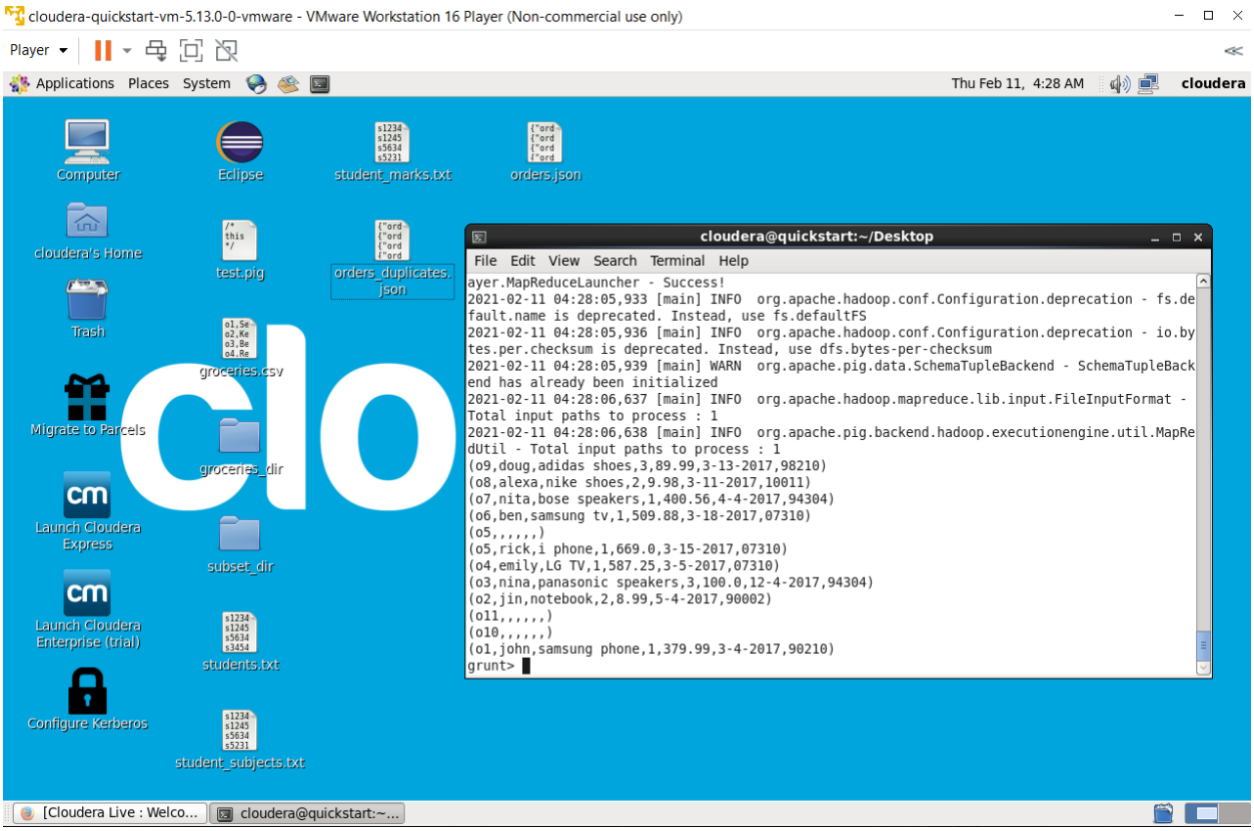




Using order keyword : order keyword sort in ascending order by default

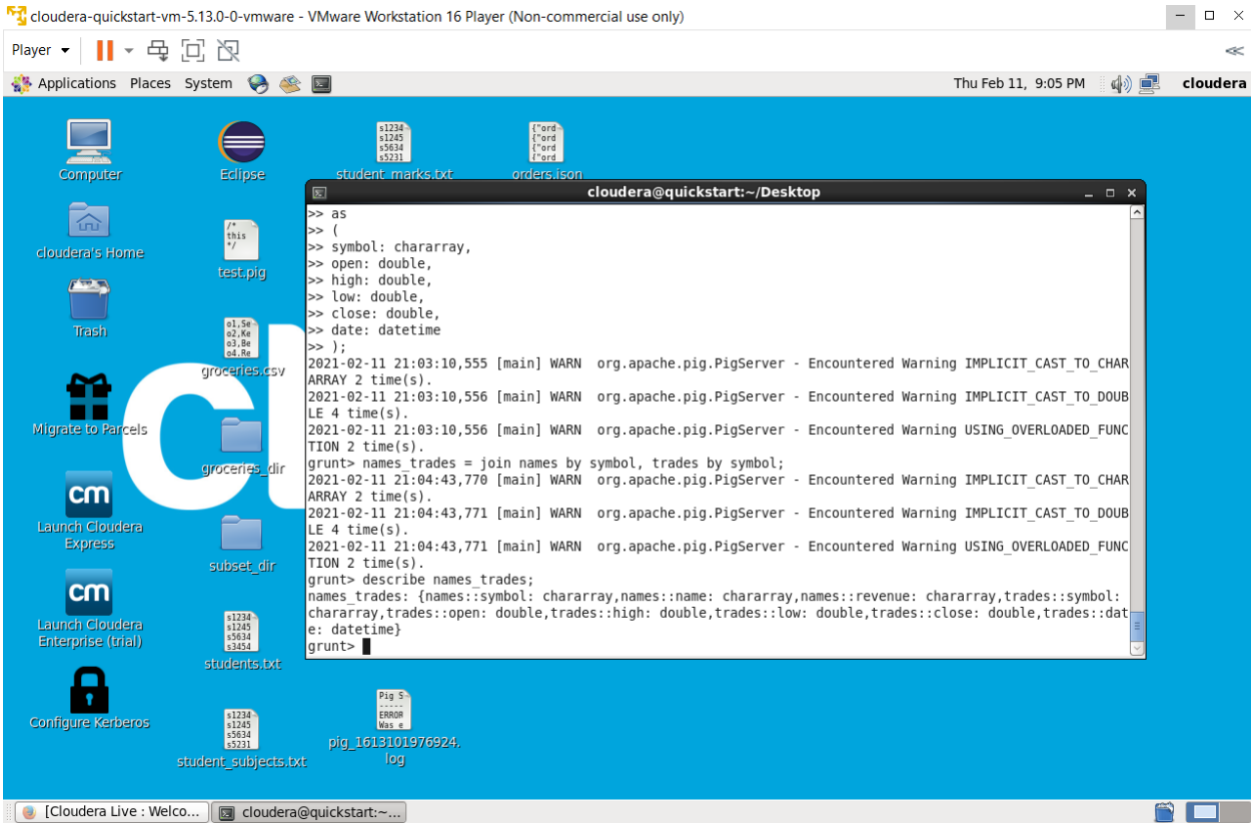
Orders_desc = order orders by order_id desc;

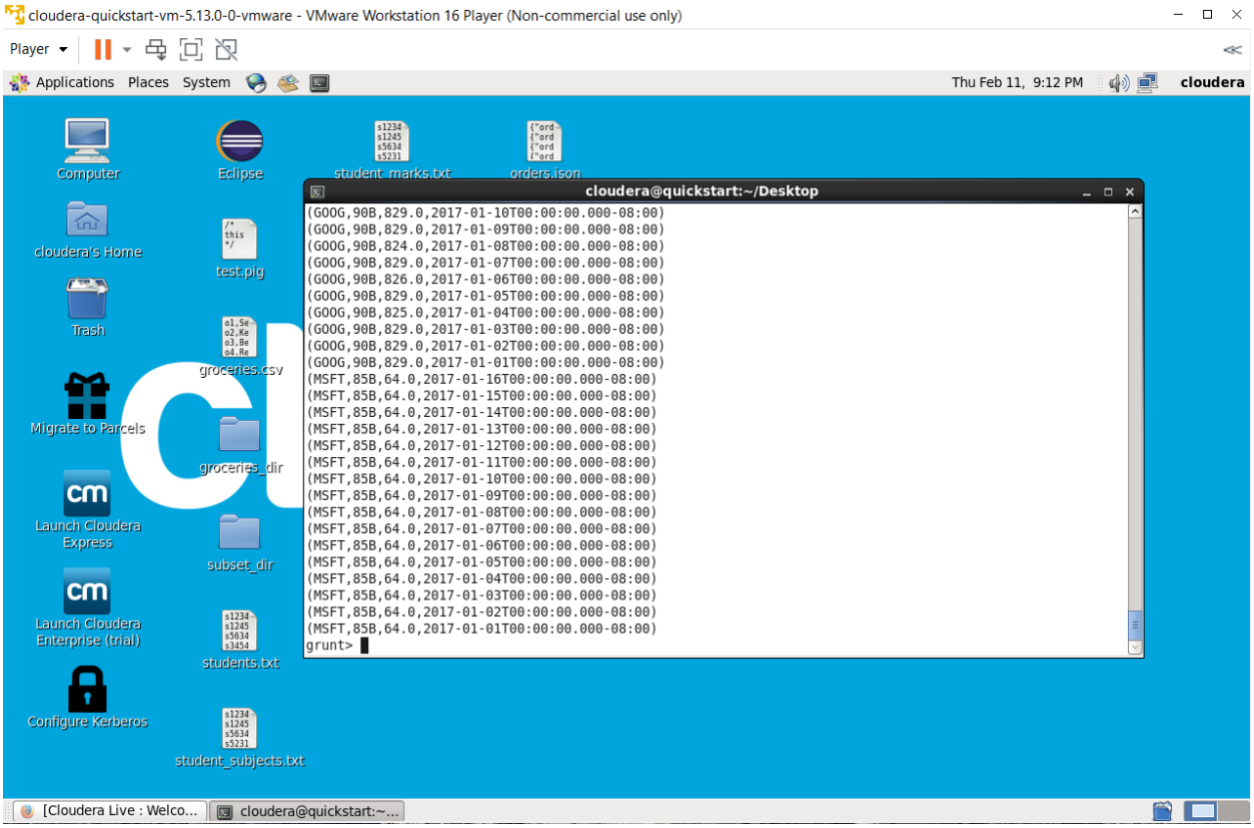




Using joins: joins are used to join two tables on common fields.

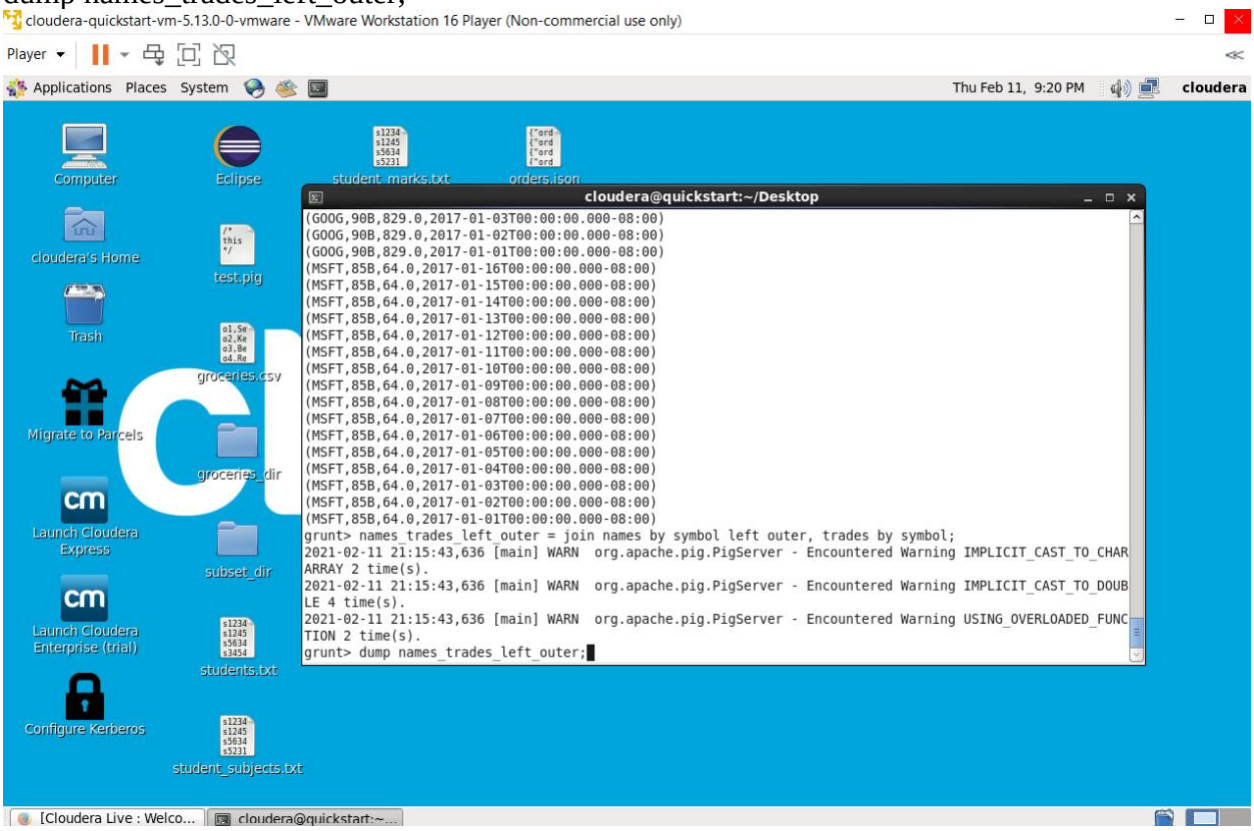
```
trades = load 'trades.csv' using PigStorage(',')
as
(
symbol: chararray,
open: double,
high: double,
low: double,
close: double,
date: datetime
);
describe trades;
names_trades = join names by symbol, trades by symbol;
```

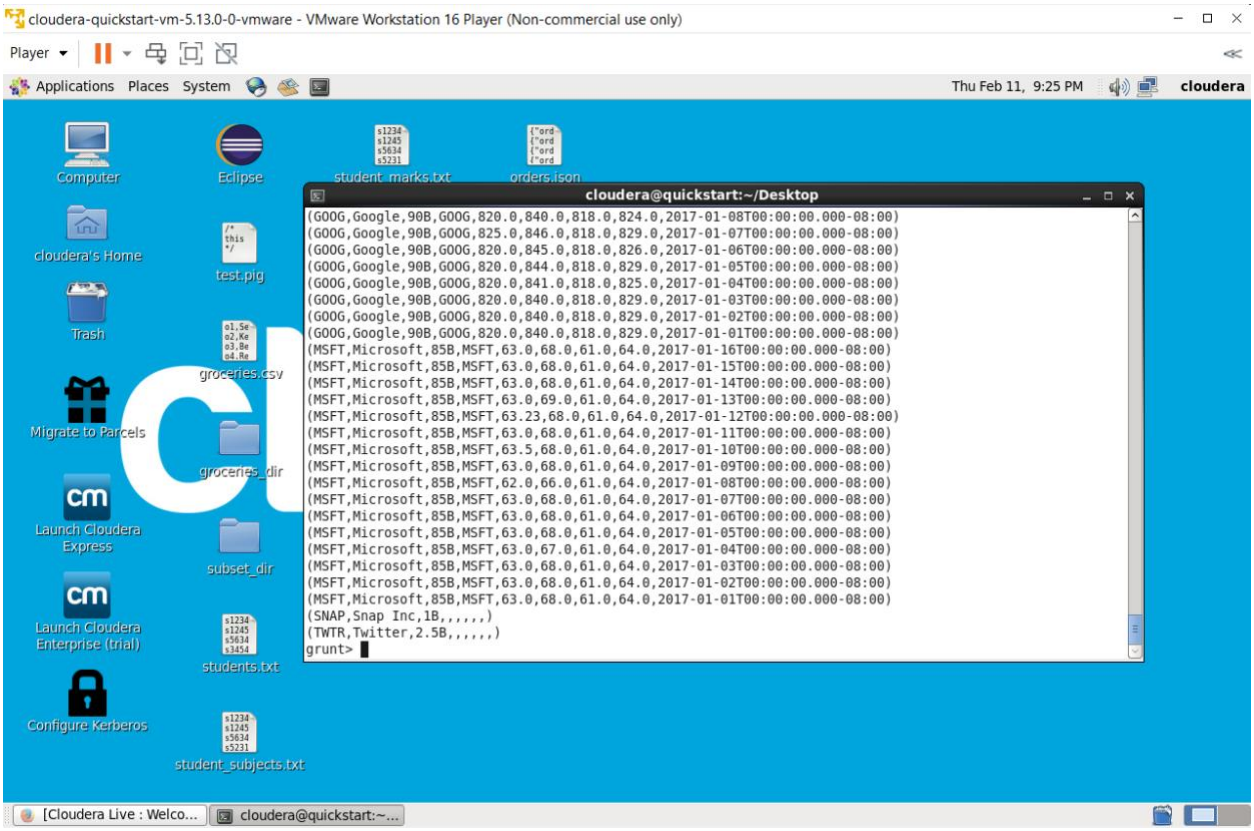

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Thu Feb 11, 9:05 PM
cloudera
Computer
Eclipse
student_marks.txt
orders.json
cloudera's Home
test.pig
Trash
groceries.csv
Migrate to Parcels
groceries_dir
Launch Cloudera Express
subset_dir
Launch Cloudera Enterprise (trial)
students.txt
Configure Kerberos
student_subjects.txt
pig_1613101976924.log
cloudera@quickstart:~/Desktop
>> as
>> (
>> symbol: chararray,
>> open: double,
>> high: double,
>> low: double,
>> close: double,
>> date: datetime
>>);
2021-02-11 21:03:10,555 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 2 time(s).
2021-02-11 21:03:10,556 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 4 time(s).
2021-02-11 21:03:10,556 [main] WARN org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNCTION 2 time(s).
grunt> names_trades = join names by symbol, trades by symbol;
2021-02-11 21:04:43,770 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 2 time(s).
2021-02-11 21:04:43,771 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 4 time(s).
2021-02-11 21:04:43,771 [main] WARN org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNCTION 2 time(s).
grunt> describe names_trades;
names_trades: {names::symbol: chararray,names::name: chararray,names::revenue: chararray,trades::symbol: chararray,trades::open: double,trades::high: double,trades::low: double,trades::close: double,trades::date: datetime}
grunt>
[Cloudera Live : Welco...
cloudera@quickstart:~...


cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Thu Feb 11, 9:12 PM
cloudera
Computer
Eclipse
student_marks.txt
orders.json
cloudera's Home
test.pig
Trash
groceries.csv
Migrate to Parcels
groceries_dir
Launch Cloudera Express
subset_dir
Launch Cloudera Enterprise (trial)
students.txt
Configure Kerberos
student_subjects.txt
cloudera@quickstart:~/Desktop
(GOOG,90B,829.0,2017-01-10T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-09T00:00:00.000-08:00)
(GOOG,90B,824.0,2017-01-08T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-07T00:00:00.000-08:00)
(GOOG,90B,826.0,2017-01-06T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-05T00:00:00.000-08:00)
(GOOG,90B,825.0,2017-01-04T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-03T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-02T00:00:00.000-08:00)
(GOOG,90B,829.0,2017-01-01T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-16T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-15T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-14T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-13T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-12T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-11T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-10T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-09T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-08T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-07T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-06T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-05T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-04T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-03T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-02T00:00:00.000-08:00)
(MSFT,85B,64.0,2017-01-01T00:00:00.000-08:00)
grunt>
[Cloudera Live : Welco...
cloudera@quickstart:~...

Using left outer: all the rows of left table and matched one's in the right table will be the result of the left outer join.

names_trades_left_outer = join names by symbol left outer, trades by symbol;
dump names_trades_left_outer;



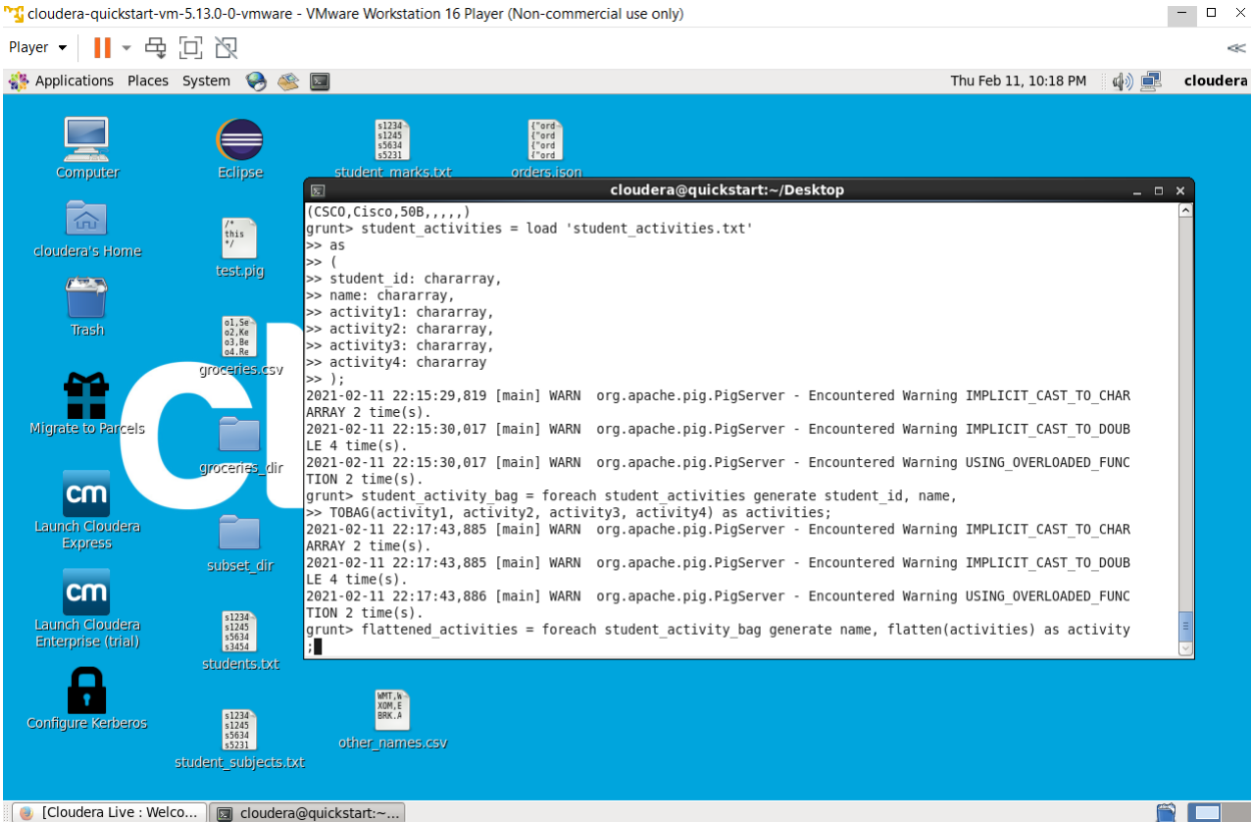


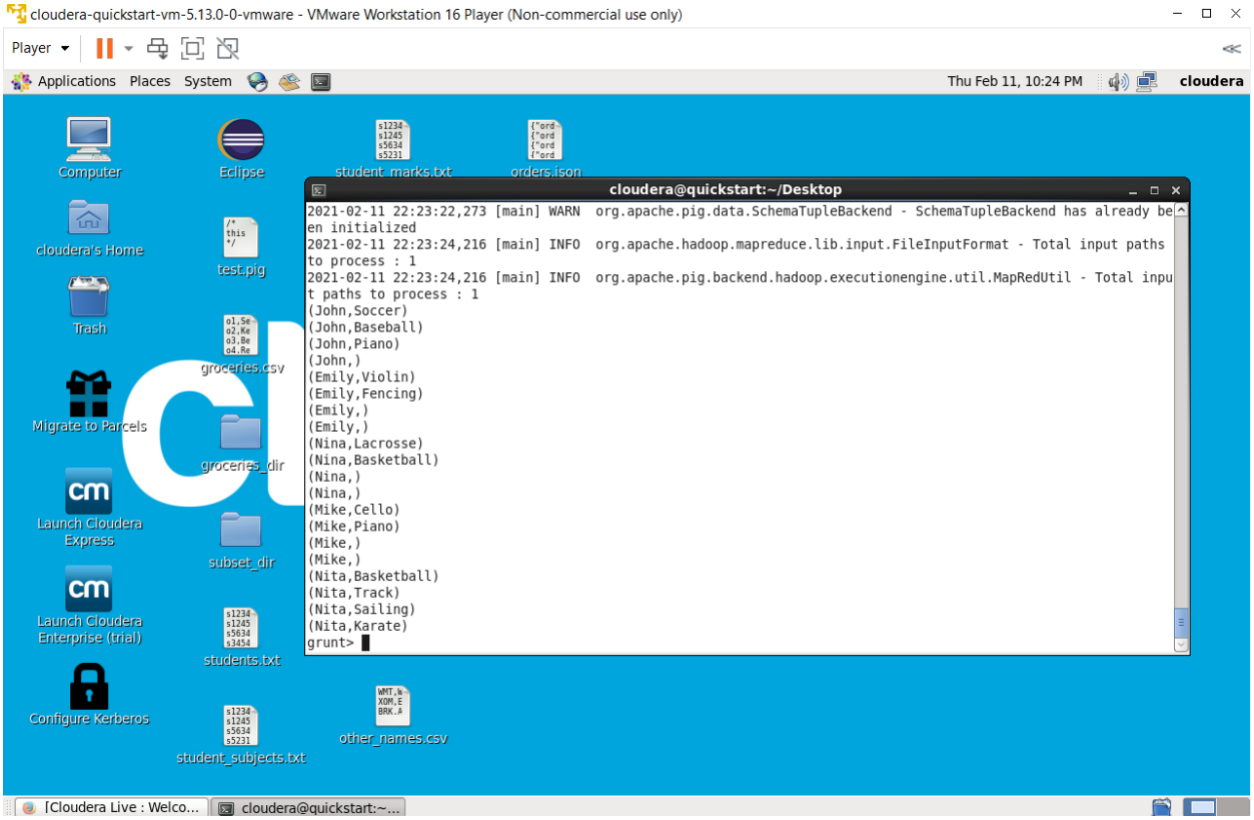
Using flatten command:

This will spilt the values to the particular column.

flattened_activities = foreach student_activity_bag generate name, flatten(activities) as activity;

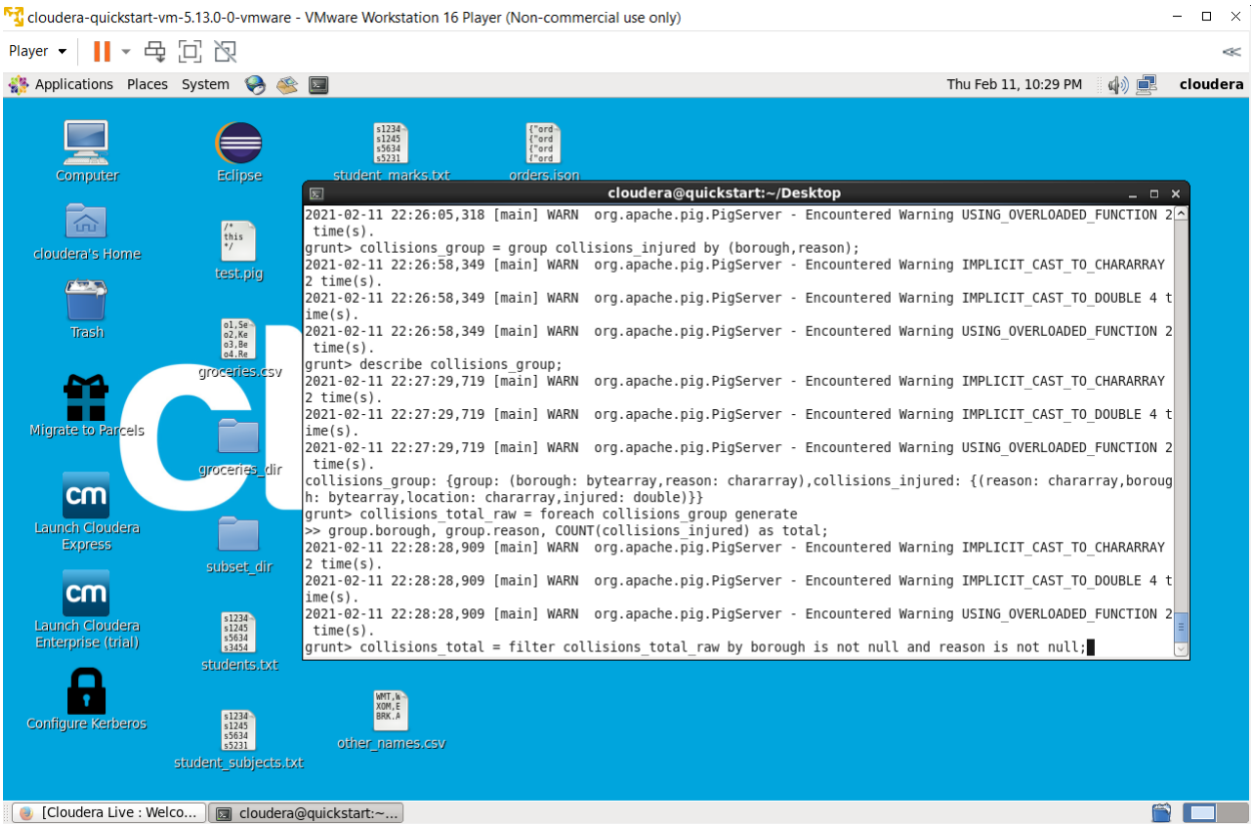
dump flattened_activities;

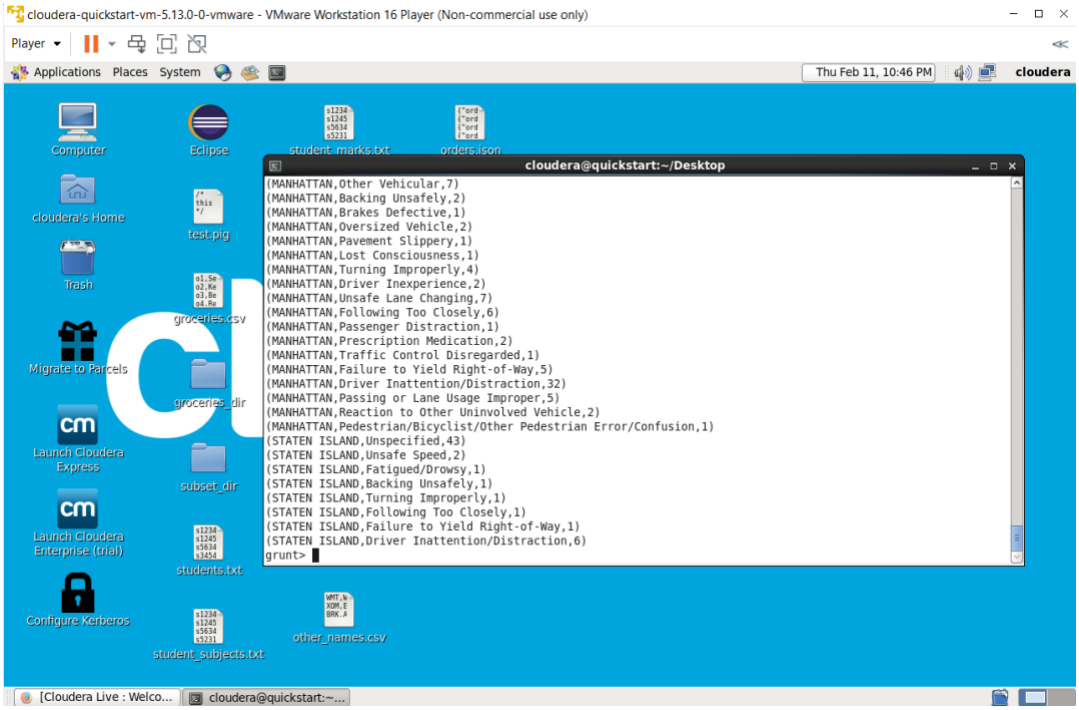

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Thu Feb 11, 10:18 PM
cloudera
Computer
Eclipse
student_marks.txt
orders.json
cloudera's Home
test.pig
Trash
groceries.csv
Migrate to Parcels
groceries_dir
Launch Cloudera Express
subset_dir
Launch Cloudera Enterprise (trial)
students.txt
Configure Kerberos
student_subjects.txt
other_names.csv
cloudera@quickstart:~/Desktop
(CSCO,Cisco,50B,,,,,)
grunt> student_activities = load 'student_activities.txt'
>> as
>> (
>> student_id: chararray,
>> name: chararray,
>> activity1: chararray,
>> activity2: chararray,
>> activity3: chararray,
>> activity4: chararray
>>);
2021-02-11 22:15:29,819 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHAR
ARRAY 2 time(s).
2021-02-11 22:15:30,017 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUB
LE 4 time(s).
2021-02-11 22:15:30,017 [main] WARN org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNC
TION 2 time(s).
grunt> student_activity_bag = foreach student_activities generate student_id, name,
>> TOBAG(activity1, activity2, activity3, activity4) as activities;
2021-02-11 22:17:43,885 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHAR
ARRAY 2 time(s).
2021-02-11 22:17:43,885 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUB
LE 4 time(s).
2021-02-11 22:17:43,886 [main] WARN org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNC
TION 2 time(s).
grunt> flattened_activities = foreach student_activity_bag generate name, flatten(activities) as activity
;
[Cloudera Live : Welco...
cloudera@quickstart:~...


cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Thu Feb 11, 10:24 PM
cloudera
Computer
Eclipse
student_marks.txt
orders.json
cloudera's Home
test.pig
Trash
groceries.csv
Migrate to Parcels
groceries_dir
Launch Cloudera Express
subset_dir
Launch Cloudera Enterprise (trial)
students.txt
Configure Kerberos
student_subjects.txt
other_names.csv
cloudera@quickstart:~/Desktop
2021-02-11 22:23:22,273 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already be
en initialized
2021-02-11 22:23:24,216 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths
to process : 1
2021-02-11 22:23:24,216 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total inpu
t paths to process : 1
(John,Soccer)
(John,Baseball)
(John,Piano)
(John,)
(Emily,Violin)
(Emily,Fencing)
(Emily,)
(Emily,)
(Nina,Lacrosse)
(Nina,Basketball)
(Nina,)
(Nina,)
(Mike,Cello)
(Mike,Piano)
(Mike,)
(Mike,)
(Nita,Basketball)
(Nita,Track)
(Nita,Sailing)
(Nita,Karate)
grunt>
[Cloudera Live : Welco...
cloudera@quickstart:~...

1.Using filters to clean the data

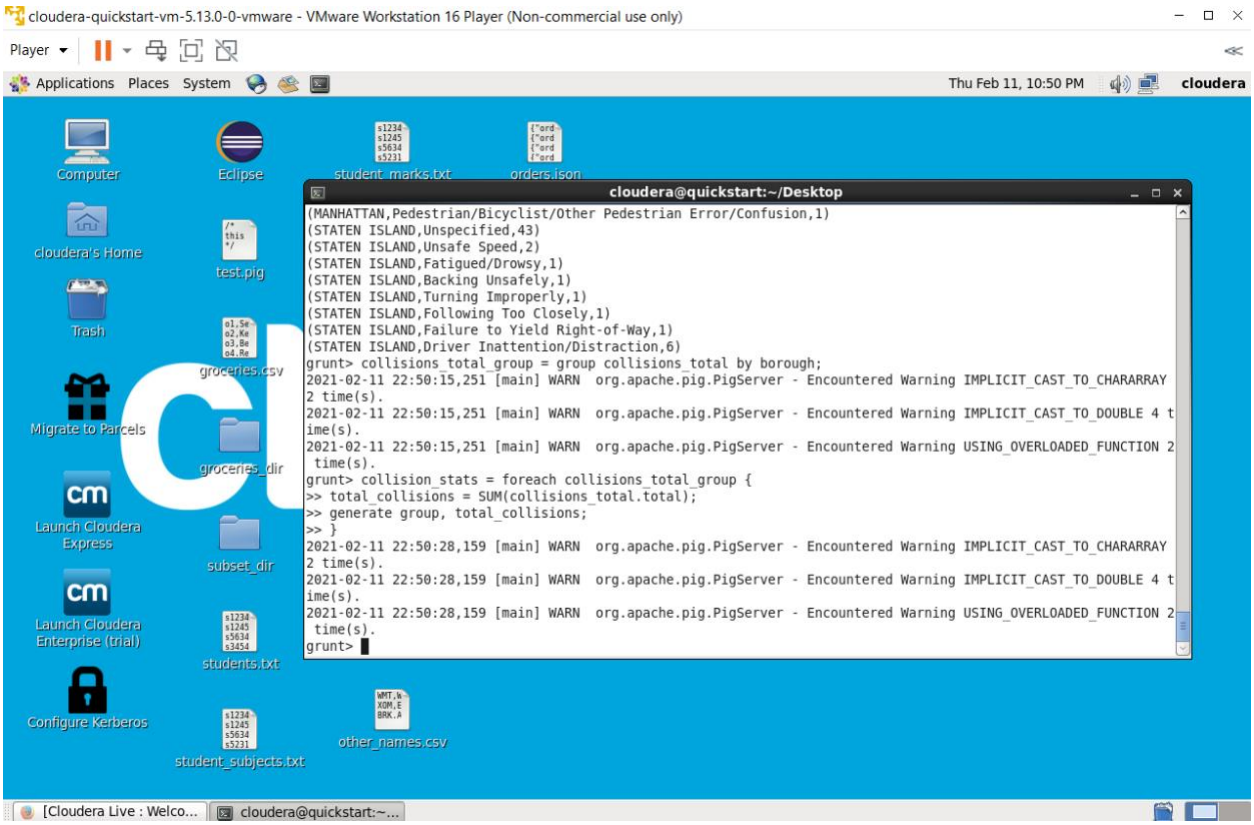
collisions_total = filter collisions_total_raw by borough is not null and reason is not null;

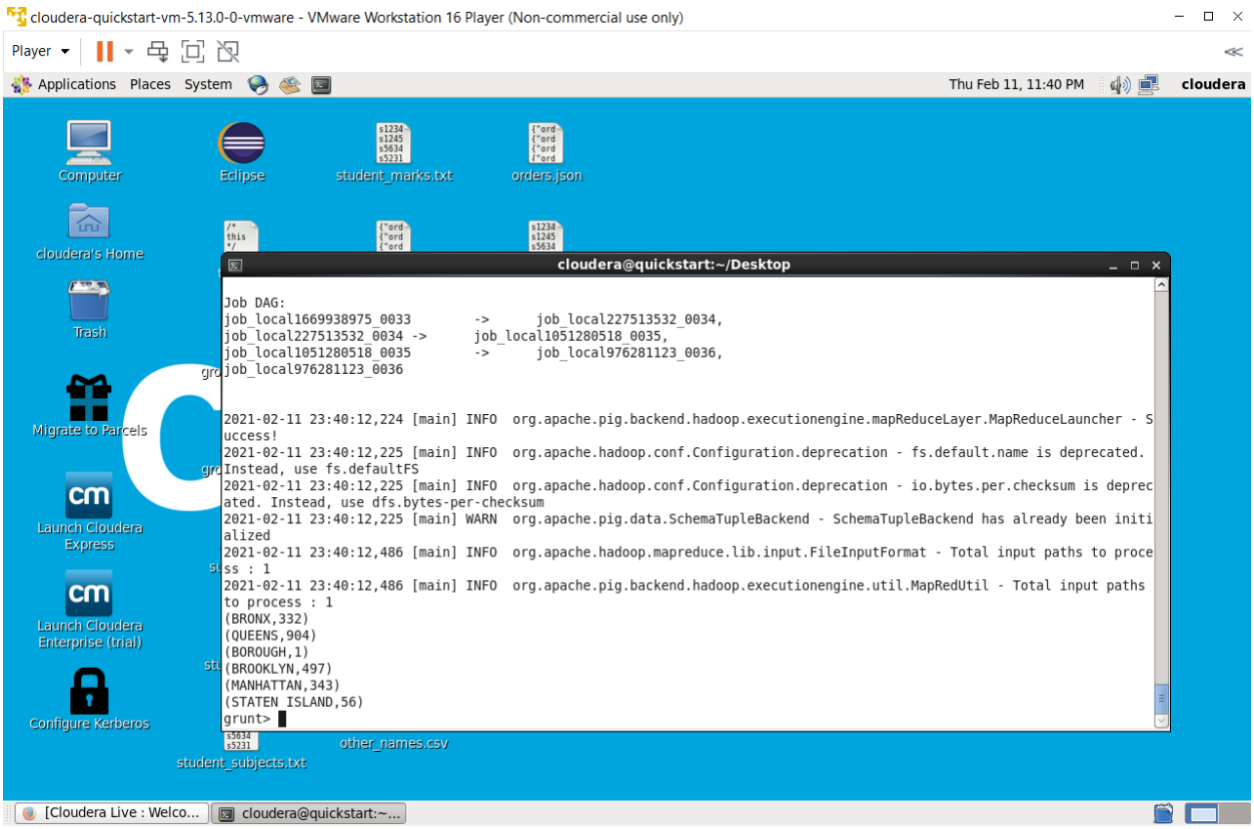




Using nested foreach:

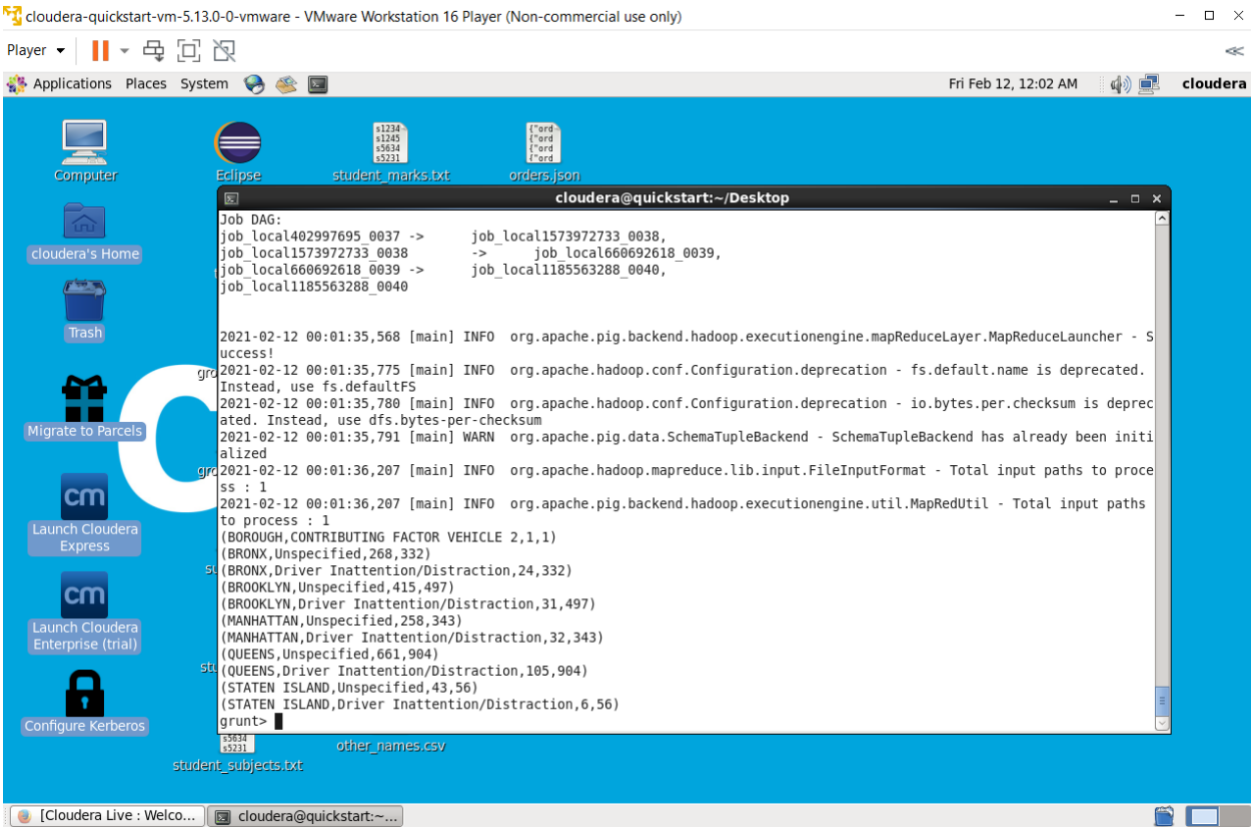
foreach inside foreach
collision_stats = foreach collisions_total_group {
total_collisions = SUM(collisions_total.total);
generate group, total_collisions;
}





Foreach with highest keyword

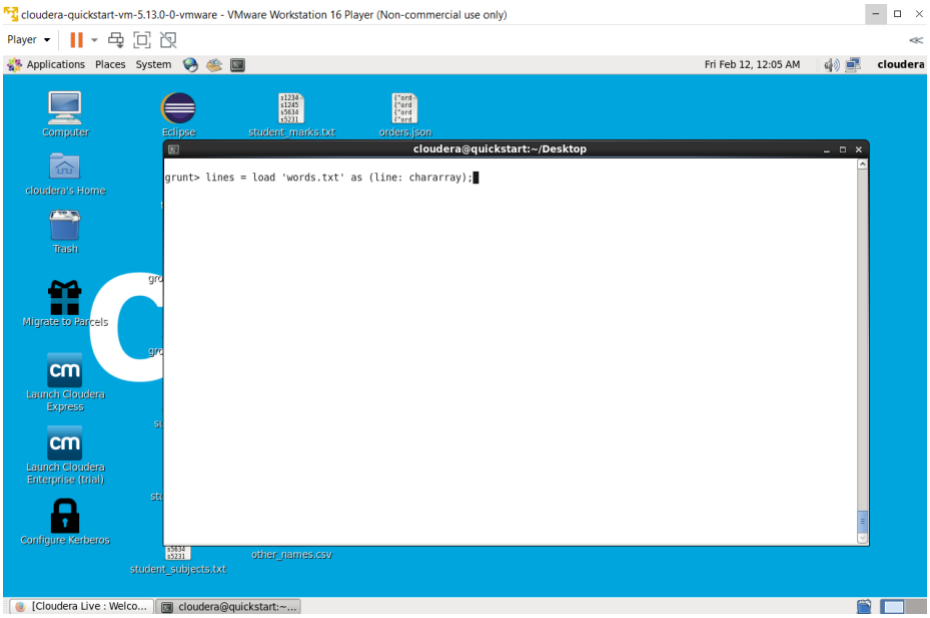
```
collision_stats = foreach collisions_total_group {
total_collisions = SUM(collisions_total.total);
sorted_collisions = order collisions_total by total desc;
highest_num_collisions = limit sorted_collisions 2;
generate flatten(highest_num_collisions), total_collisions;
}
```



Executing MapReduce using Pig

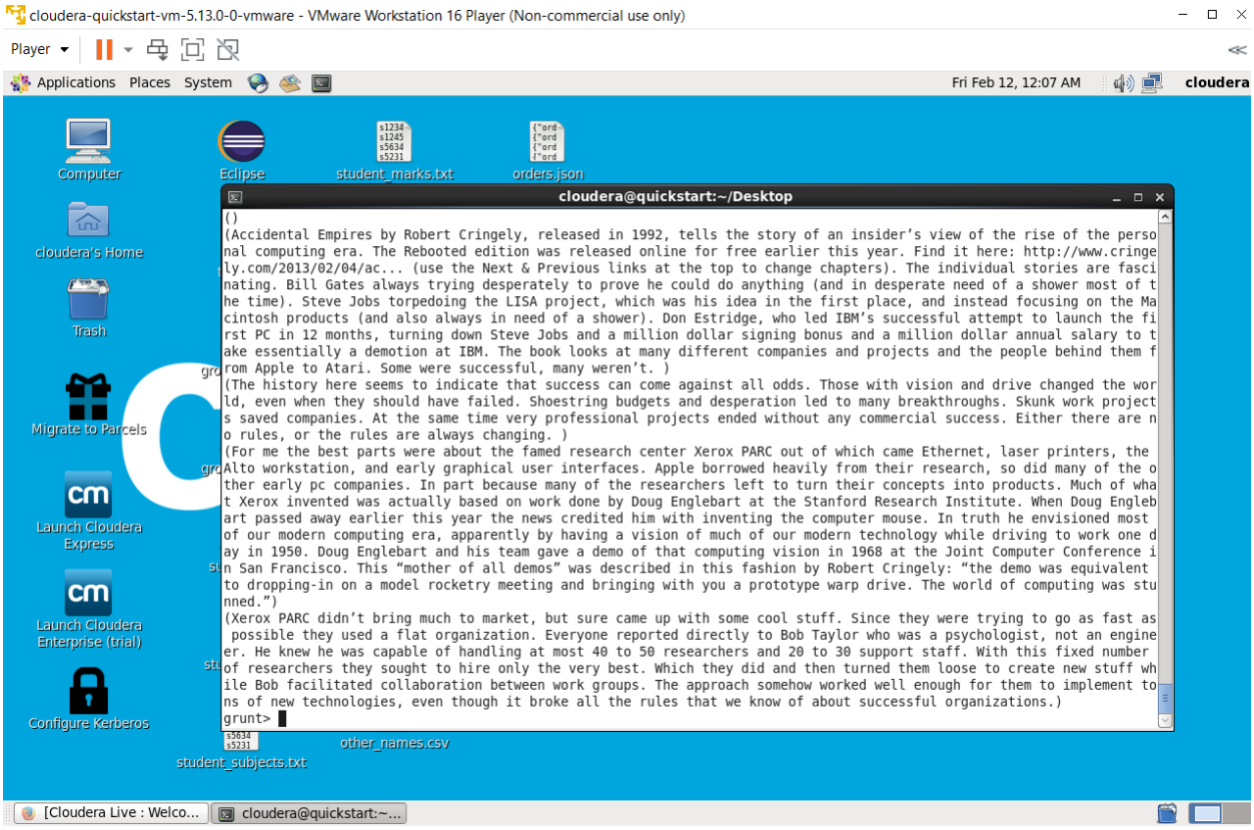
Counting number of words in text file

1. lines = load 'words.txt' as (line: chararray);



2. displays the contents

dump lines;



3. the following command splits into each line

word_bag = foreach lines generate TOKENIZE(line) as bag_of_words;

4. the following command splits into each word

words = foreach word_bag generate flatten(bag_of_words) as word;

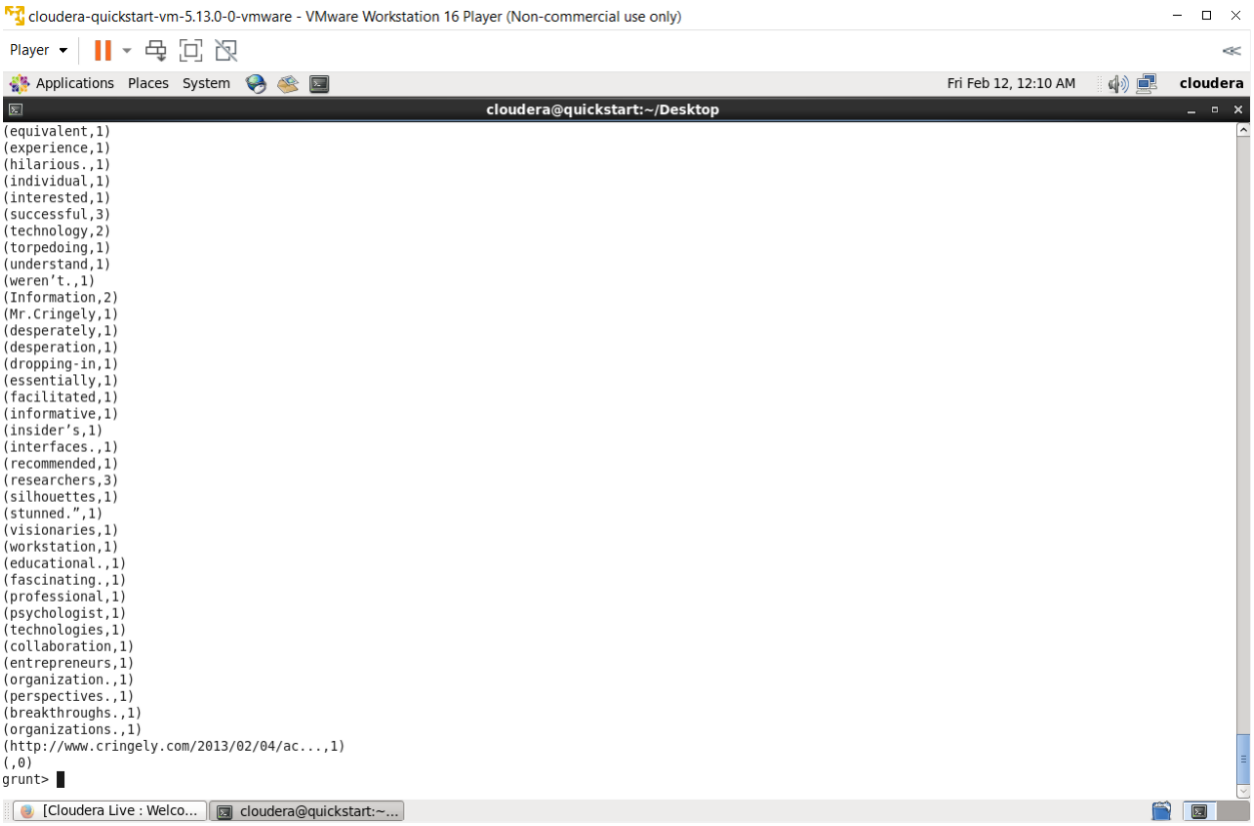
5. The following command groups the similar words and count the number of word in it and displays the same.

word_group = group words by word;

.word_count = foreach word_group generate group, COUNT(words);

describe word_count;

dump word_count;


cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Fri Feb 12, 12:10 AM
cloudera
cloudera@quickstart:~/Desktop
(equivalent,1)
(experience,1)
(hilarious.,1)
(individual,1)
(interested,1)
(successful,3)
(technology,2)
(torpedoing,1)
(understand,1)
(weren't.,1)
(Information,2)
(Mr.Cringely,1)
(desperately,1)
(desperation,1)
(dropping-in,1)
(essentially,1)
(facilitated,1)
(informative,1)
(insider's,1)
(interfaces.,1)
(recommended,1)
(researchers,3)
(silhouettes,1)
(stunned.",1)
(visionaries,1)
(workstation,1)
(educational.,1)
(fascinating.,1)
(professional,1)
(psychologist,1)
(technologies,1)
(collaboration,1)
(entrepreneurs,1)
(organization.,1)
(perspectives.,1)
(breakthroughs.,1)
(organizations.,1)
(http://www.cringely.com/2013/02/04/ac...,1)
(,0)
grunt>
[Cloudera Live : Welco...
cloudera@quickstart:~...

APACHE HIVE

WHAT IS HIVE

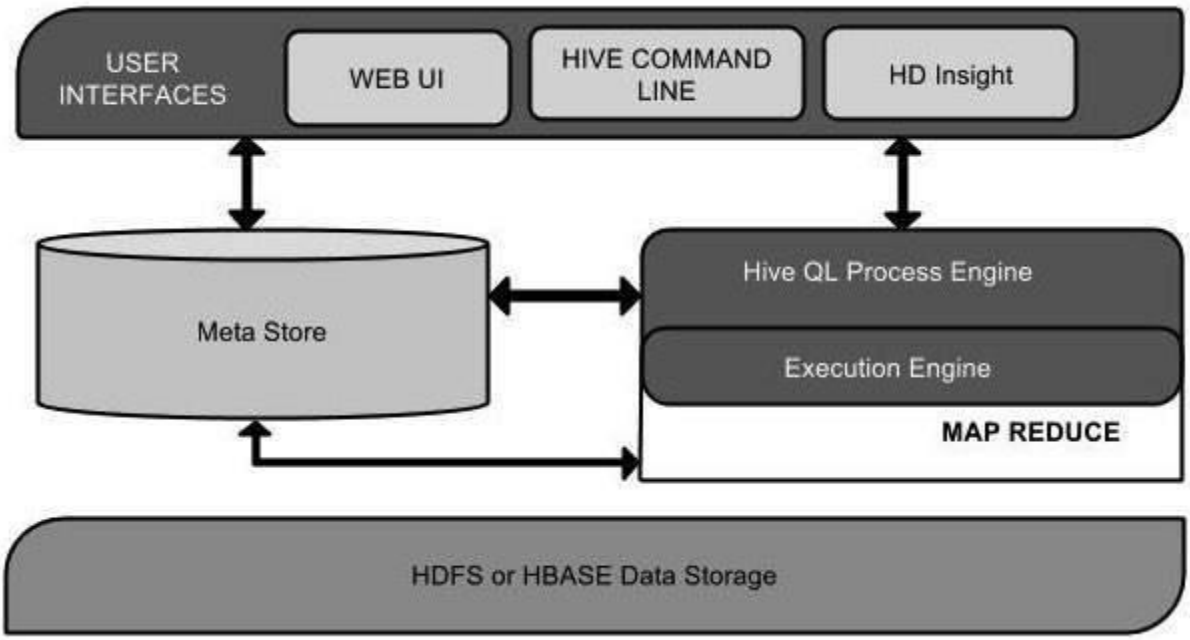
Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

FEATURES OF HIVE

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

ARCHITECTURE OF HIVE

The following component diagram depicts the architecture of Hive:



This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

HIVE INSTALLATION

STEP 1: Make sure java and Hadoop are already present.

STEP 2: You can download it by visiting the following link http://apache.petsads.us/hive/hive-0.14.0/. Let us assume it gets downloaded onto the /Downloads directory. Here, we download Hive

archive named “apache-hive-0.14.0-bin.tar.gz” for this tutorial. The following command is used to verify the download:

```
$ cd Downloads
$ ls
```

On successful download, you get to see the following response:

```
apache-hive-0.14.0-bin.tar.gz
```

STEP 3: The following command is used to verify the download and extract the hive archive:

```
$ tar zxvf apache-hive-0.14.0-bin.tar.gz
$ ls
```

On successful download, you get to see the following response:

```
apache-hive-0.14.0-bin apache-hive-0.14.0-bin.tar.gz
```

STEP 4: We need to copy the files from the super user “su -”. The following commands are used to copy the files from the extracted directory to the /usr/local/hive” directory.

```
$ su -
passwd:

# cd /home/user/Download
# mv apache-hive-0.14.0-bin /usr/local/hive
# exit
```

STEP 5:

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
```

```
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

The following command is used to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

STEP 6:To configure Hive with Hadoop, you need to edit the **hive-env.sh** file, which is placed in the **$HIVE_HOME/conf** directory. The following commands redirect to Hive **config** folder and copy the template file:

```
$ cd $HIVE_HOME/conf
$ cp hive-env.sh.template hive-env.sh
```

Edit the **hive-env.sh** file by appending the following line:

```
export HADOOP_HOME=/usr/local/hadoop
```

STEP 7: The following command is used to download Apache Derby. It takes some time to download.

```
$ cd ~
$ wget http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz
```

The following command is used to verify the download:

```
$ ls
```

On successful download, you get to see the following response:

```
db-derby-10.4.2.0-bin.tar.gz
```

STEP 8:

```
$ tar zxvf db-derby-10.4.2.0-bin.tar.gz
$ ls
```

On successful download, you get to see the following response:

```
db-derby-10.4.2.0-bin db-derby-10.4.2.0-bin.tar.gz
```

STEP 9: We need to copy from the super user "su -". The following commands are used to copy the files from the extracted directory to the /usr/local/derby directory:

```
$ su -
passwd:
# cd /home/user
# mv db-derby-10.4.2.0-bin /usr/local/derby
# exit
```

STEP 10: You can set up the Derby environment by appending the following lines to **~/.bashrc** file:

```
export DERBY_HOME=/usr/local/derby
export PATH=$PATH:$DERBY_HOME/bin
Apache Hive
18
export
CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools
.jar
```

The following command is used to execute **~/.bashrc** file:

```
$ source ~/.bashrc
```

Create a directory named data in $DERBY_HOME directory to store Metastore data.

```
$ mkdir $DERBY_HOME/data
```

STEP 11: Configuring Metastore means specifying to Hive where the database is stored. You can do this by editing the hive-site.xml file, which is in the $HIVE_HOME/conf directory. First of all, copy the template file using the following command:

```
$ cd $HIVE_HOME/conf
$ cp hive-default.xml.template hive-site.xml
```

Edit **hive-site.xml** and append the following lines between the <configuration> and </configuration> tags:

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby://localhost:1527/metastore_db;create=true </value>
  <description>JDBC connect string for a JDBC metastore </description>
</property>
```

Create a file named jpox.properties and add the following lines into it:

```
javax.jdo.PersistenceManagerFactoryClass =
org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

STEP 12: Before running Hive, you need to create the **/tmp** folder and a separate Hive folder in HDFS. Here, we use the **/user/hive/warehouse** folder. You need to set write permission for these newly created folders as shown below:

```
chmod g+w
```

Now set them in HDFS before verifying Hive. Use the following commands:

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

The following commands are used to verify Hive installation:

```
$ cd $HIVE_HOME
$ bin/hive
```

On successful installation of Hive, you get to see the following response:

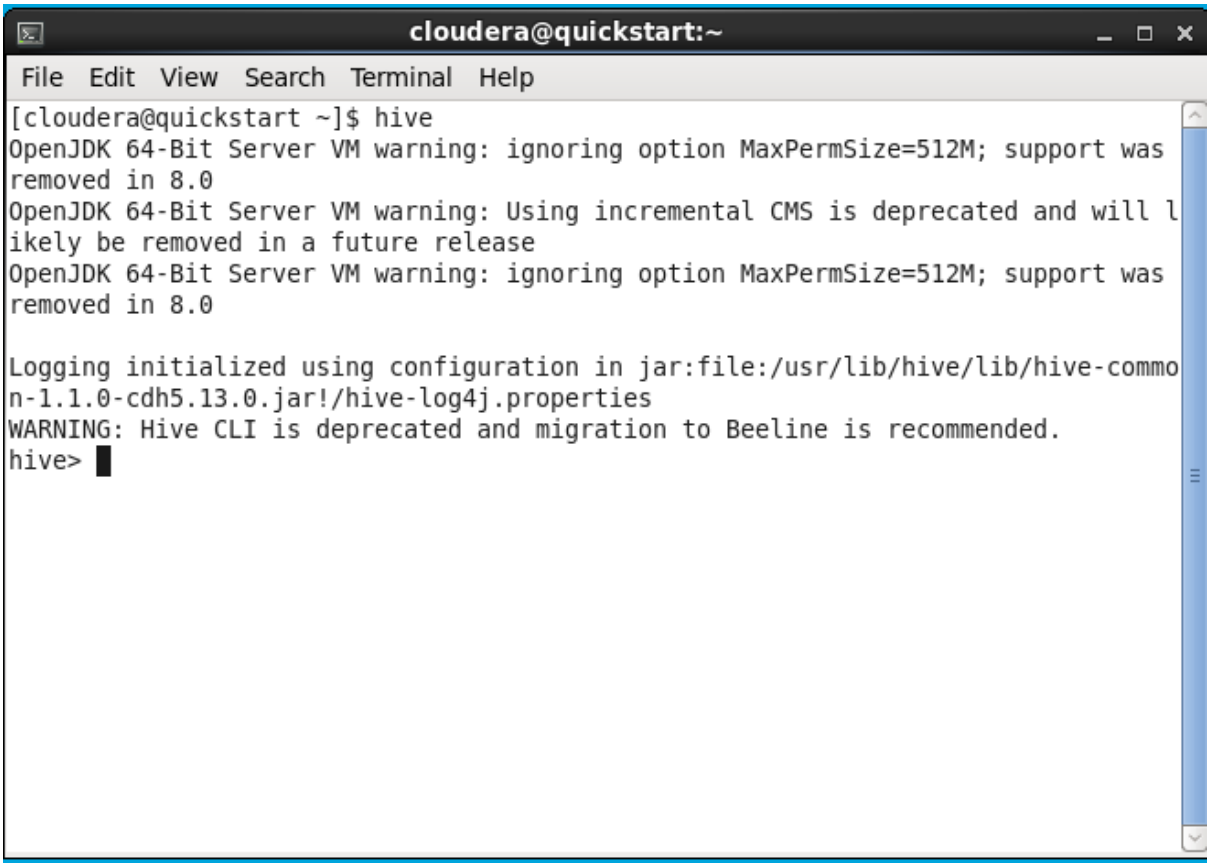
```
Logging initialized using configuration in jar:file:/home/hadoop/hive-0.9.0/lib/hive-common-
0.9.0.jar!/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201312121621_1494929084.txt
…….
hive>
```

The following sample command is executed to display all the tables:

```
hive> show tables;
OK
Time taken: 2.798 seconds
hive>
```

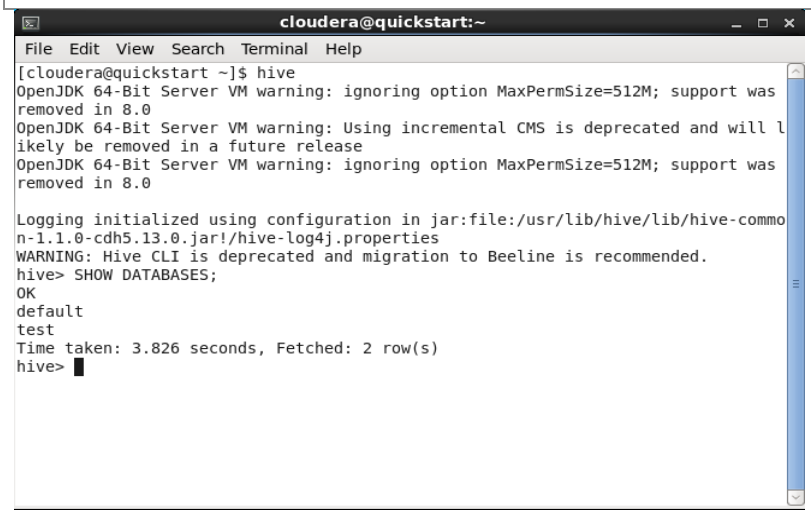
HIVE COMMANDS

$hive – opens up hive



The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
```



Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

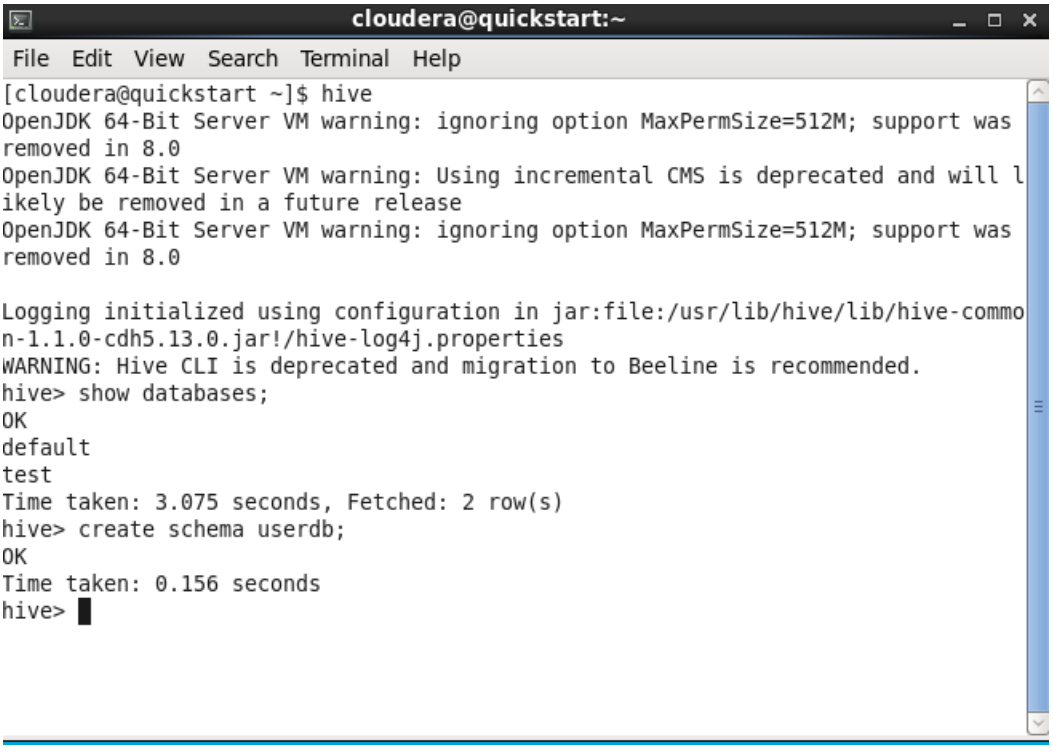
```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

or

```
hive> CREATE SCHEMA userdb;
```



Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

```
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name
[RESTRICT|CASCADE];
```

The following queries are used to drop a database. Let us assume that the database name is **userdb**.

```
hive> DROP DATABASE IF EXISTS userdb;
```

The following query drops the database using **CASCADE**. It means dropping respective tables before dropping the database.

```
hive> DROP DATABASE IF EXISTS userdb CASCADE;
```

The following query drops the database using **SCHEMA**.

```
hive> DROP SCHEMA userdb;
```

```
hive>
    >
    > drop database if exists userdb;
OK
Time taken: 0.216 seconds
hive> show databases;
OK
default
test
Time taken: 0.034 seconds, Fetched: 2 row(s)
hive>
```

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]
```

```
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

The following query creates a table named **employee** using the above data.

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String);
```

```
hive> create table if not exists employee(eid int, name String, salary String, d
estination String);
OK
Time taken: 3.609 seconds
hive>
```

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

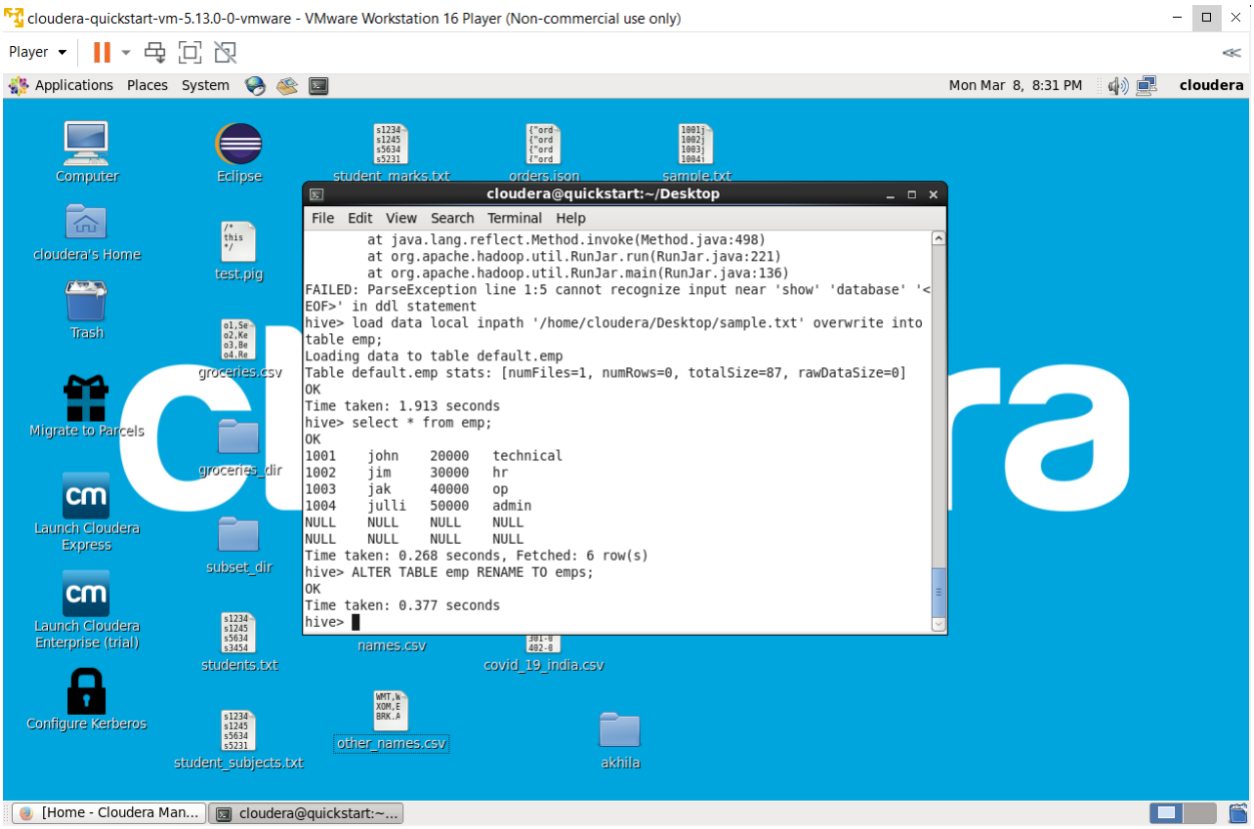
Syntax

The syntax for load data is as follows:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

The following query loads the given text into the table.

```
hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'
OVERWRITE INTO TABLE employee;
```



Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new_name

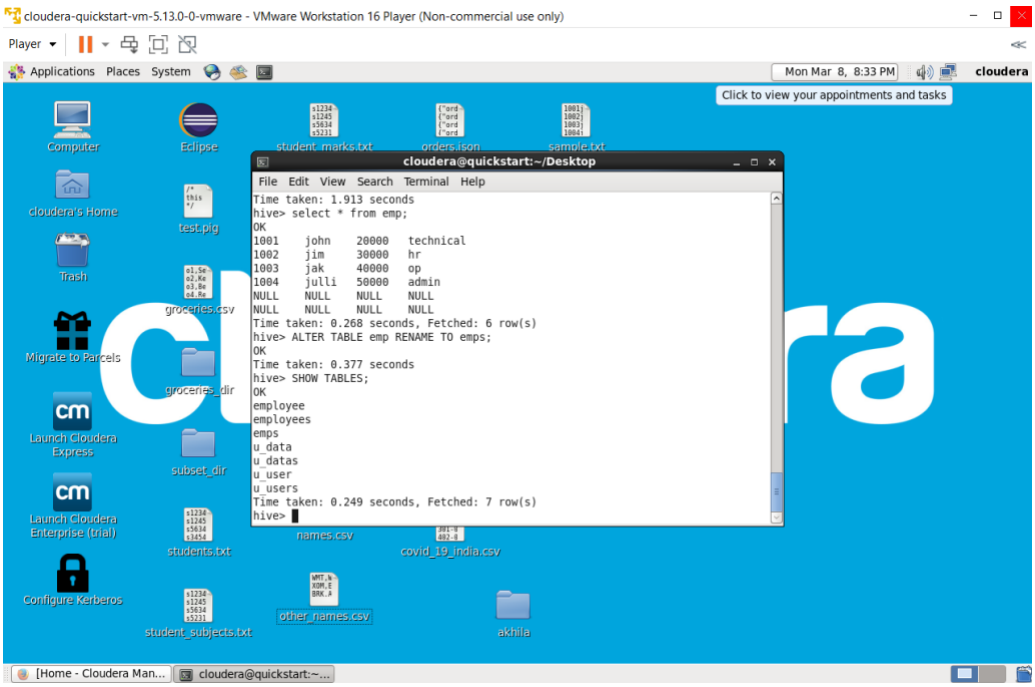
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])

ALTER TABLE name DROP [COLUMN] column_name

ALTER TABLE name CHANGE column_name new_name new_type

ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

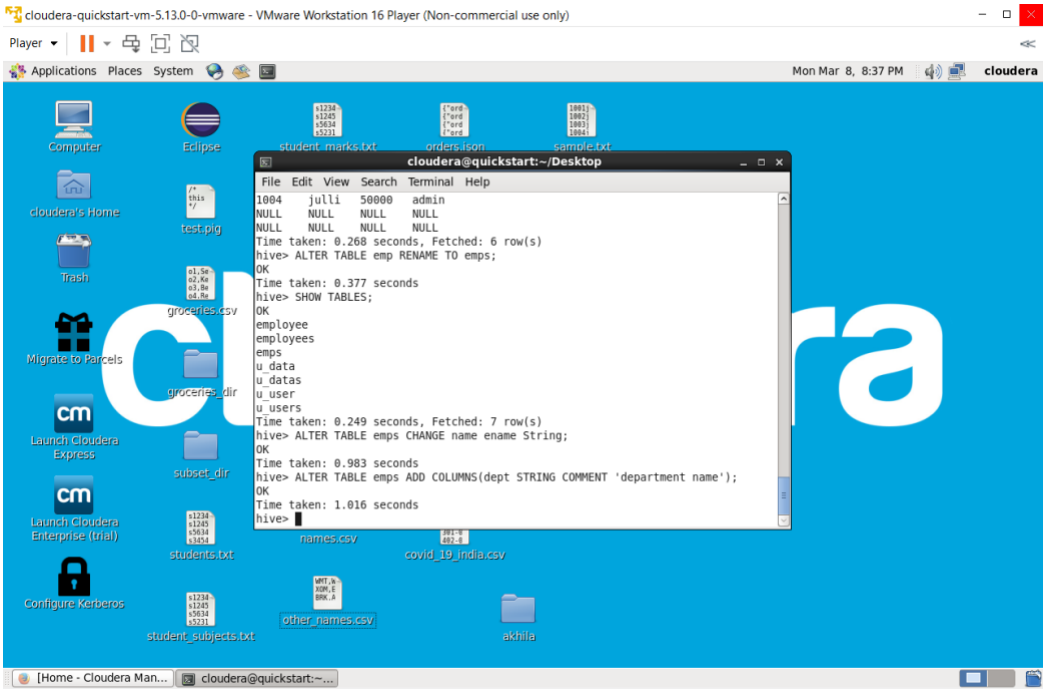
Example: hive> ALTER TABLE employee CHANGE name ename String;



ADD COLUMNS STATEMENT

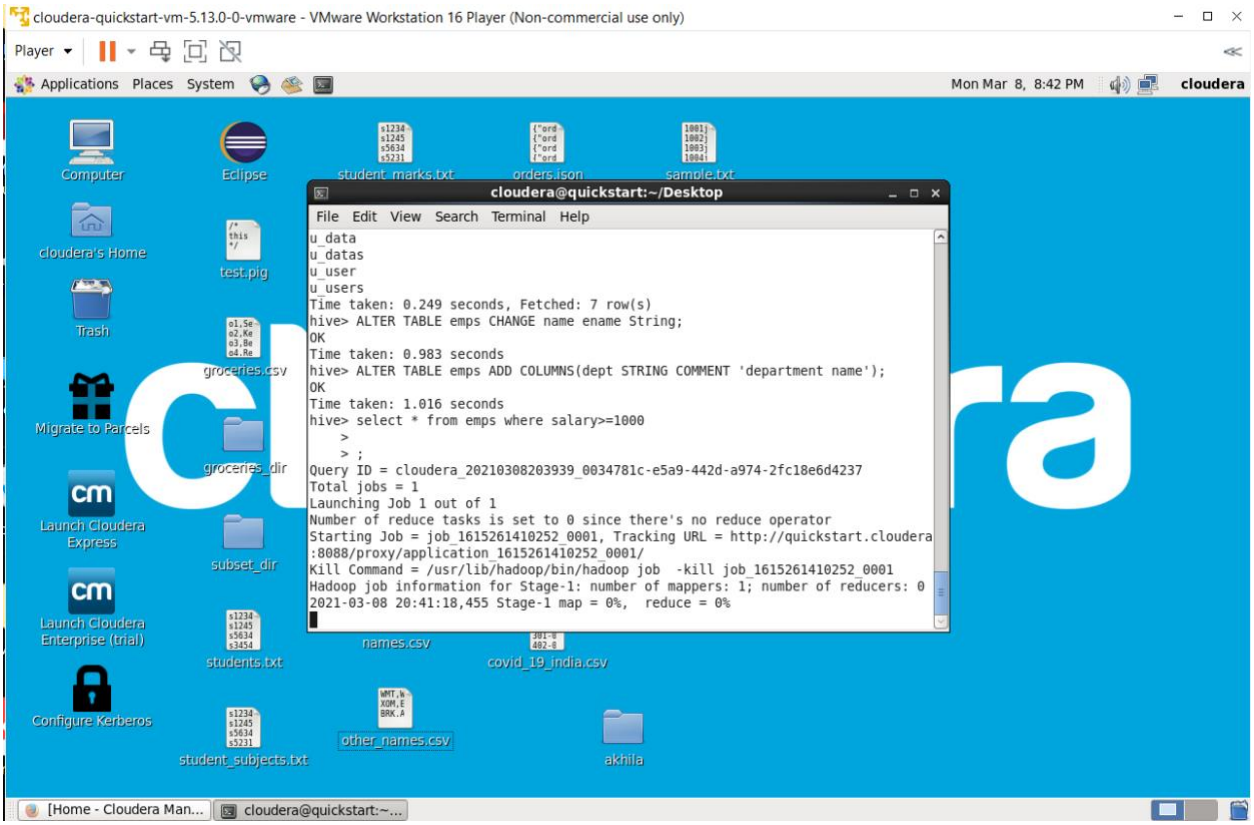
The following query adds a column named dept to the employee table.

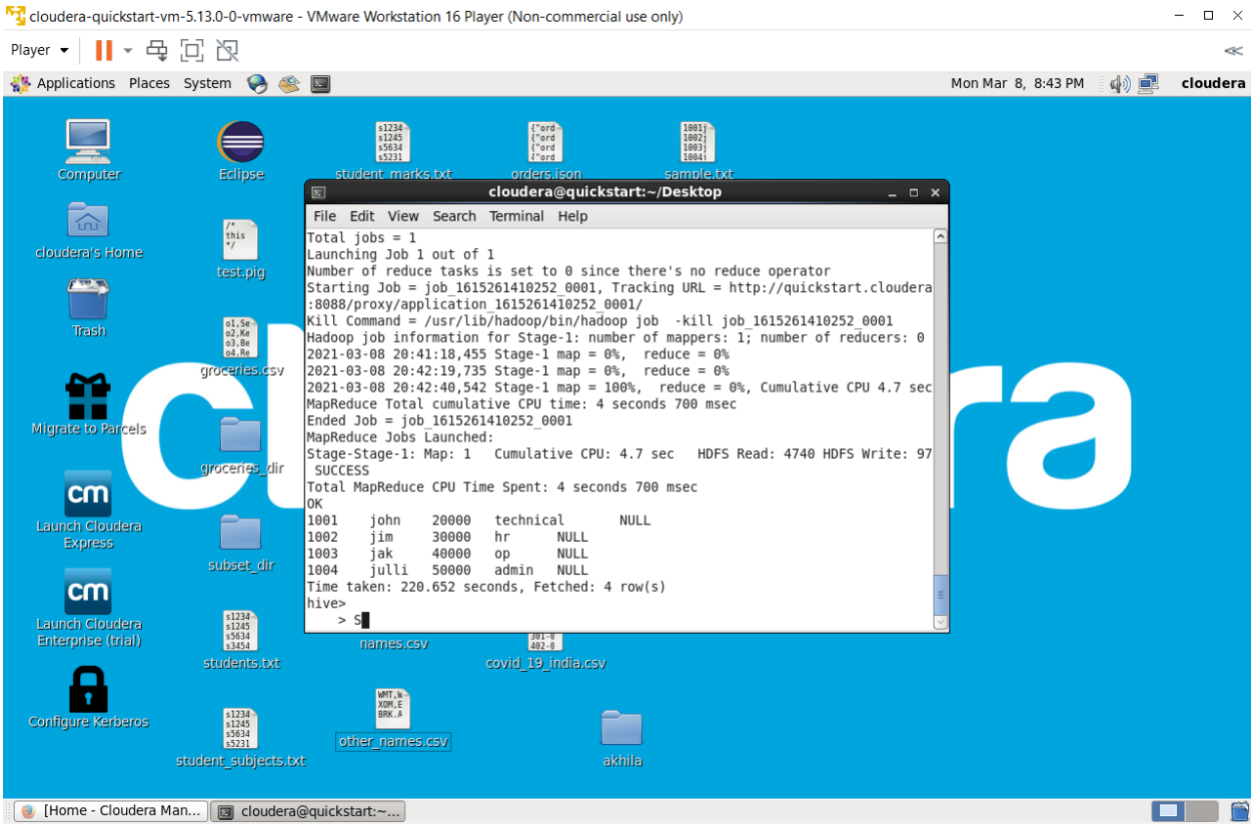
```
hive> ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');
```



Select query

hive> select * from emps where salary>=1000





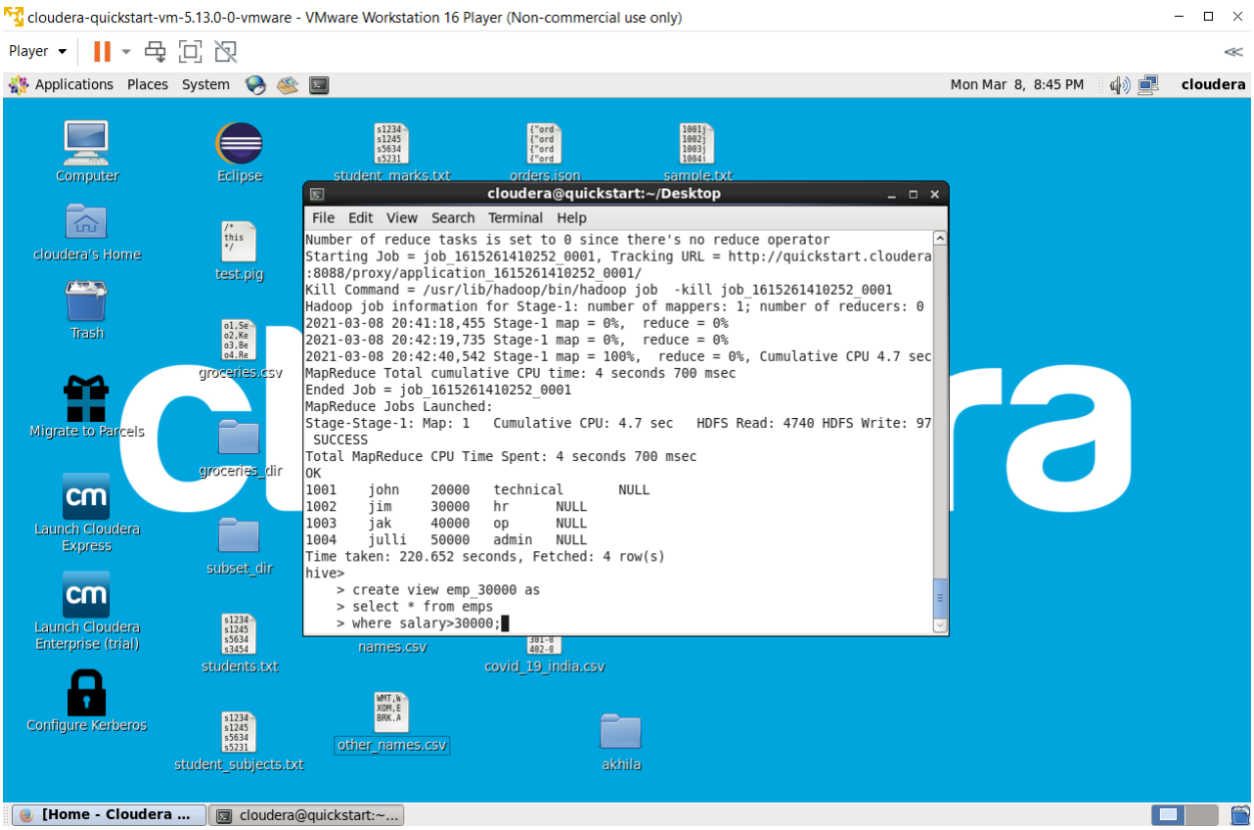
Creating views

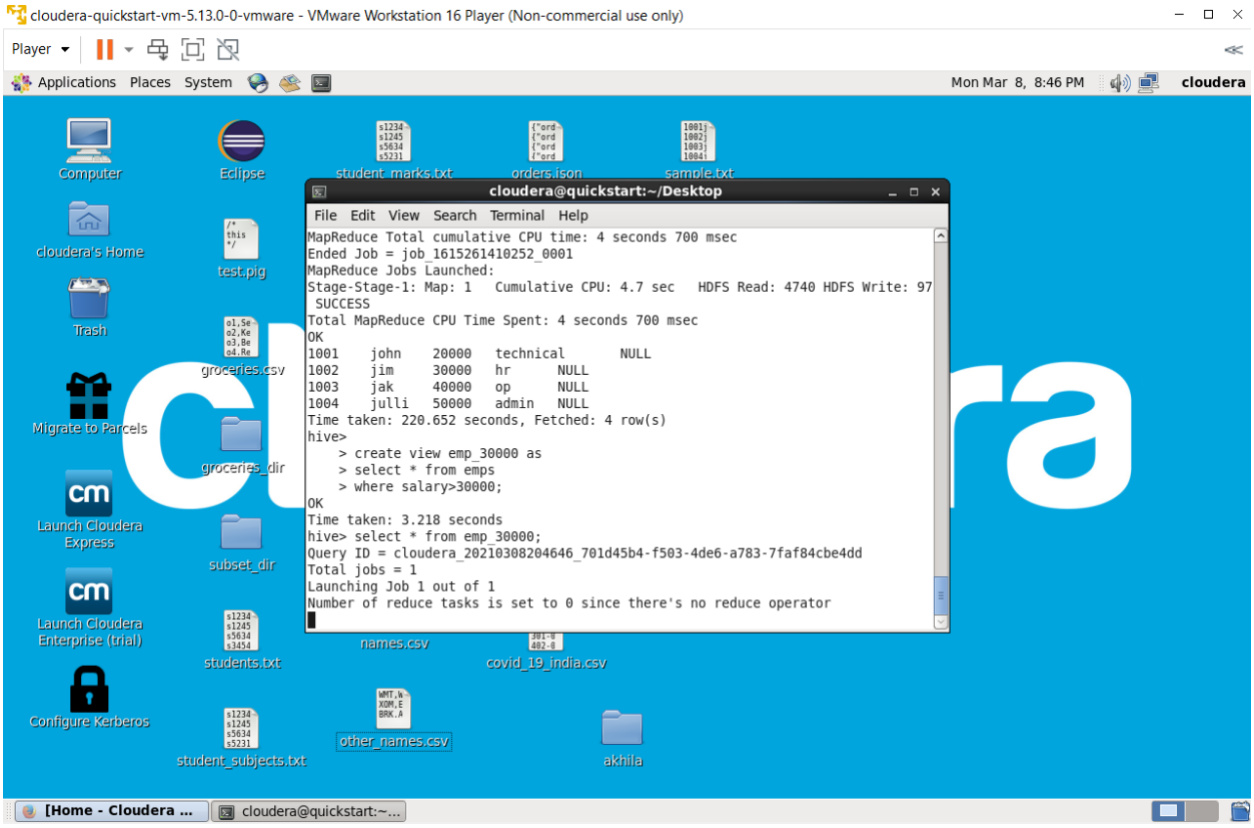
hive>

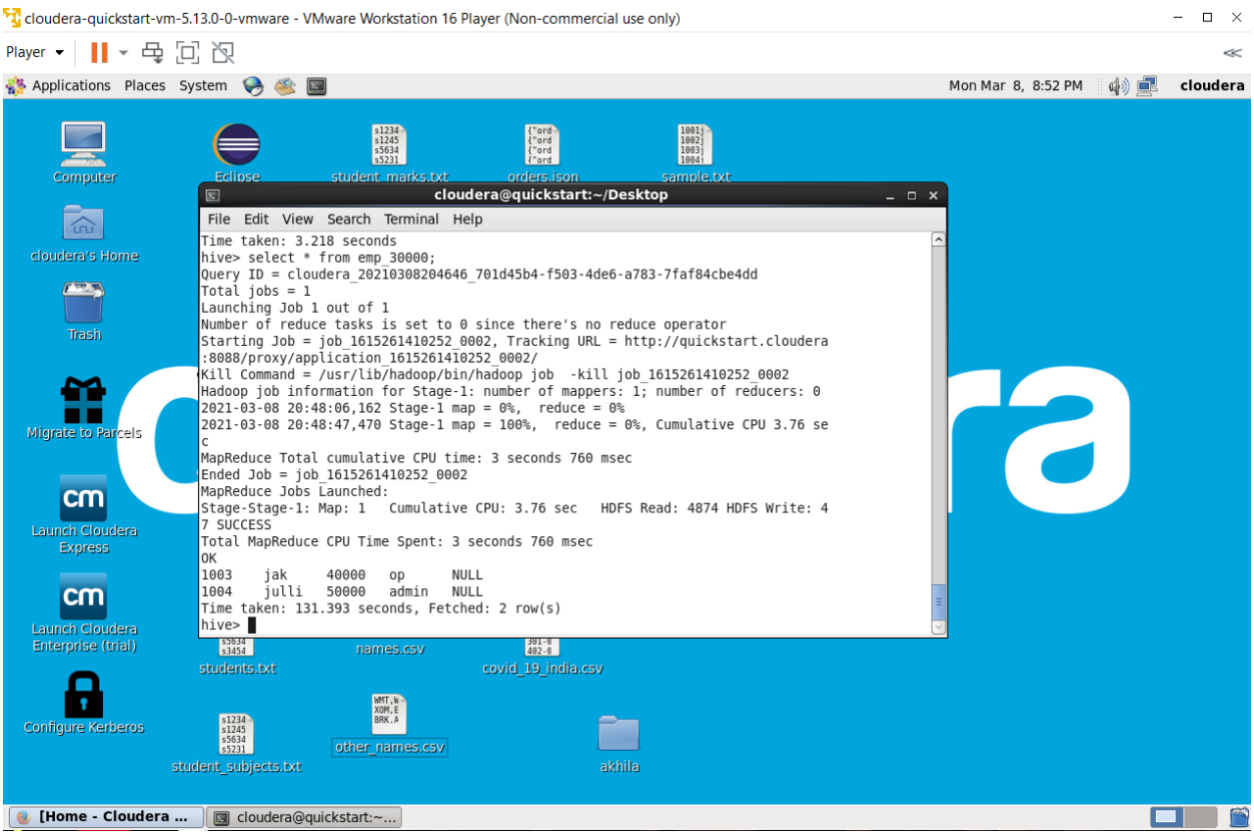
> create view emp_30000 as

> select * from emps

> where salary>30000;


cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Mon Mar 8, 8:45 PM
cloudera
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1615261410252_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1615261410252_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1615261410252_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-03-08 20:41:18,455 Stage-1 map = 0%, reduce = 0%
2021-03-08 20:42:19,735 Stage-1 map = 0%, reduce = 0%
2021-03-08 20:42:40,542 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.7 sec
MapReduce Total cumulative CPU time: 4 seconds 700 msec
Ended Job = job_1615261410252_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 4.7 sec HDFS Read: 4740 HDFS Write: 97 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 700 msec
OK
1001 john 20000 technical NULL
1002 jim 30000 hr NULL
1003 jak 40000 op NULL
1004 julli 50000 admin NULL
Time taken: 220.652 seconds, Fetched: 4 row(s)
hive>
> create view emp_30000 as
> select * from emps
> where salary>30000;
Computer
Eclipse
cloudera's Home
Trash
Migrate to Parcels
Launch Cloudera Express
Launch Cloudera Enterprise (trial)
Configure Kerberos
test.pig
groceries.csv
groceries_dir
subset_dir
students.txt
student_subjects.txt
student_marks.txt
orders.json
sample.txt
names.csv
covid_19_india.csv
other_names.csv
akhila
[Home - Cloudera ...
cloudera@quickstart:~...

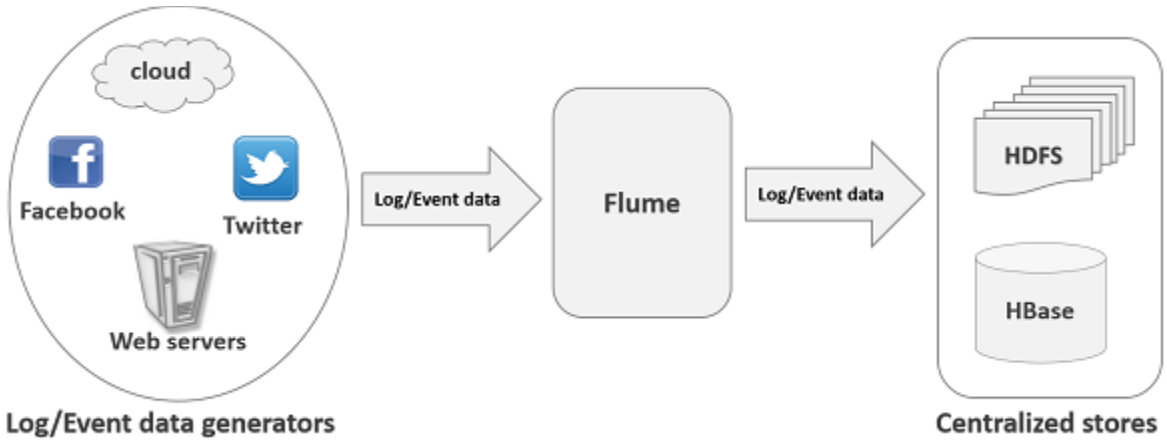

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Mon Mar 8, 8:46 PM
cloudera
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
MapReduce Total cumulative CPU time: 4 seconds 700 msec
Ended Job = job_1615261410252_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 4.7 sec HDFS Read: 4740 HDFS Write: 97 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 700 msec
OK
1001 john 20000 technical NULL
1002 jim 30000 hr NULL
1003 jak 40000 op NULL
1004 julli 50000 admin NULL
Time taken: 220.652 seconds, Fetched: 4 row(s)
hive>
> create view emp_30000 as
> select * from emps
> where salary>30000;
OK
Time taken: 3.218 seconds
hive> select * from emp_30000;
Query ID = cloudera_20210308204646_701d45b4-f503-4de6-a783-7faf84cbe4dd
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Computer
Eclipse
cloudera's Home
Trash
Migrate to Parcels
Launch Cloudera Express
Launch Cloudera Enterprise (trial)
Configure Kerberos
test.pig
groceries.csv
groceries_dir
subset_dir
students.txt
student_subjects.txt
student_marks.txt
orders.json
sample.txt
names.csv
covid_19_india.csv
other_names.csv
akhila
[Home - Cloudera ...
cloudera@quickstart:~...


cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 16 Player (Non-commercial use only)
Player
Applications Places System
Mon Mar 8, 8:52 PM cloudera
Computer
Eclipse
student_marks.txt
orders.json
sample.txt
cloudera's Home
Trash
Migrate to Parcels
Launch Cloudera Express
Launch Cloudera Enterprise (trial)
Configure Kerberos
students.txt
names.csv
covid_19_india.csv
other_names.csv
student_subjects.txt
akhila
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
Time taken: 3.218 seconds
hive> select * from emp_30000;
Query ID = cloudera_20210308204646_701d45b4-f503-4de6-a783-7faf84cbe4dd
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1615261410252_0002, Tracking URL = http://quickstart.cloudera
:8088/proxy/application_1615261410252_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1615261410252_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-03-08 20:48:06,162 Stage-1 map = 0%, reduce = 0%
2021-03-08 20:48:47,470 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.76 se
c
MapReduce Total cumulative CPU time: 3 seconds 760 msec
Ended Job = job_1615261410252_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 3.76 sec HDFS Read: 4874 HDFS Write: 4
7 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 760 msec
OK
1003 jak 40000 op NULL
1004 julli 50000 admin NULL
Time taken: 131.393 seconds, Fetched: 2 row(s)
hive>
[Home - Cloudera ...
cloudera@quickstart:~...

APACHE FLUME

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 in to 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 in to 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 destinations types.
- Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Flume can be scaled horizontally.

INSTALLING FLUME

First of all, download the latest version of Apache Flume software from the website https://flume.apache.org/.

Step 1: Open the website. Click on the download link on the left-hand side of the home page. It will take you to the download page of Apache Flume**.**

Step 2 : In the Download page, you can see the links for binary and source files of Apache Flume. Click on the link apache-flume-1.6.0-bin.tar.gz

You will be redirected to a list of mirrors where you can start your download by clicking any of these mirrors. In the same way, you can download the source code of Apache Flume by clicking on apache-flume-1.6.0-src.tar.gz.

Step 3 : Create a directory with the name Flume in the same directory where the installation directories of Hadoop, HBase, and other software were installed (if you have already installed any) as shown below.

$ mkdir Flume

Step 4: Extract the downloaded tar files as shown below.

$ cd Downloads/
$ tar zxvf apache-flume-1.6.0-bin.tar.gz
$ tar zxvf apache-flume-1.6.0-src.tar.gz

Step 5: Move the content of apache-flume-1.6.0-bin.tar file to the Flume directory created earlier as shown below. (Assume we have created the Flume directory in the local user named Hadoop.)

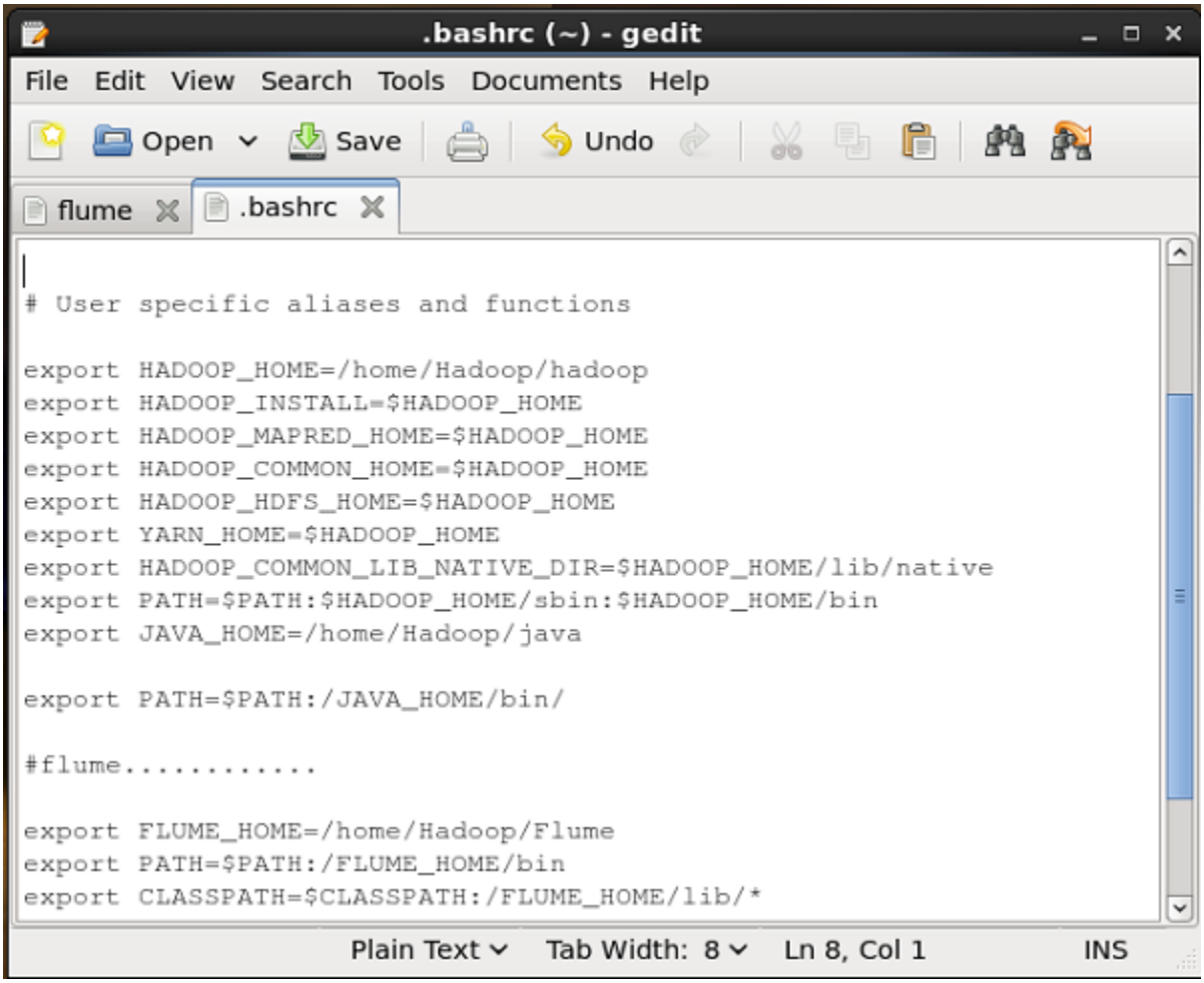
$ mv apache-flume-1.6.0-bin.tar/* /home/Hadoop/Flume/

CONFIGURING FLUME

To configure Flume, we have to modify three files namely, **flume-env.sh, flumeconf.properties,** and **bash.rc**.

Setting the Path / Classpath

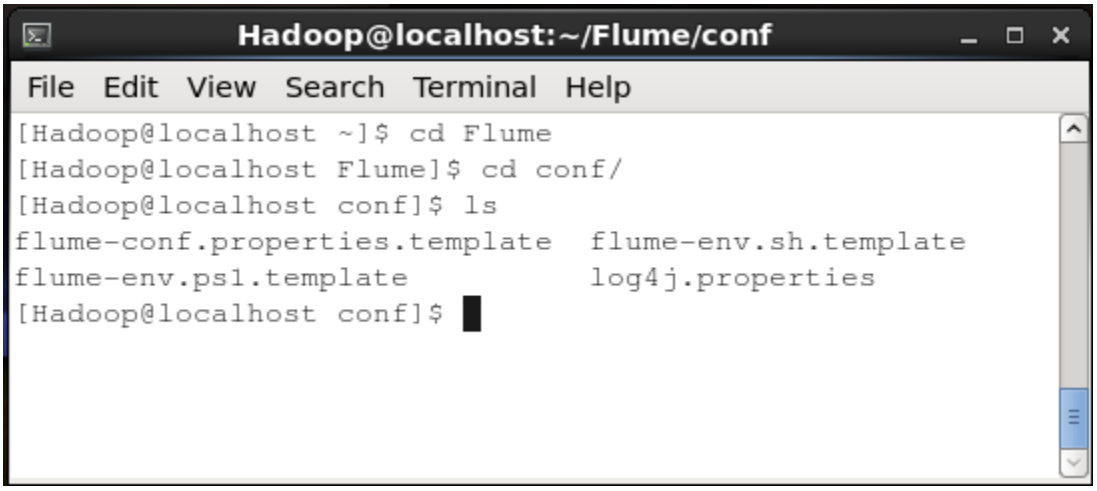
In the **.bashrc** file, set the home folder, the path, and the classpath for Flume as shown below.



conf Folder

If you open the **conf** folder of Apache Flume, you will have the following four files −

- flume-conf.properties.template,
- flume-env.sh.template,
- flume-env.ps1.template, and
- log4j.properties.

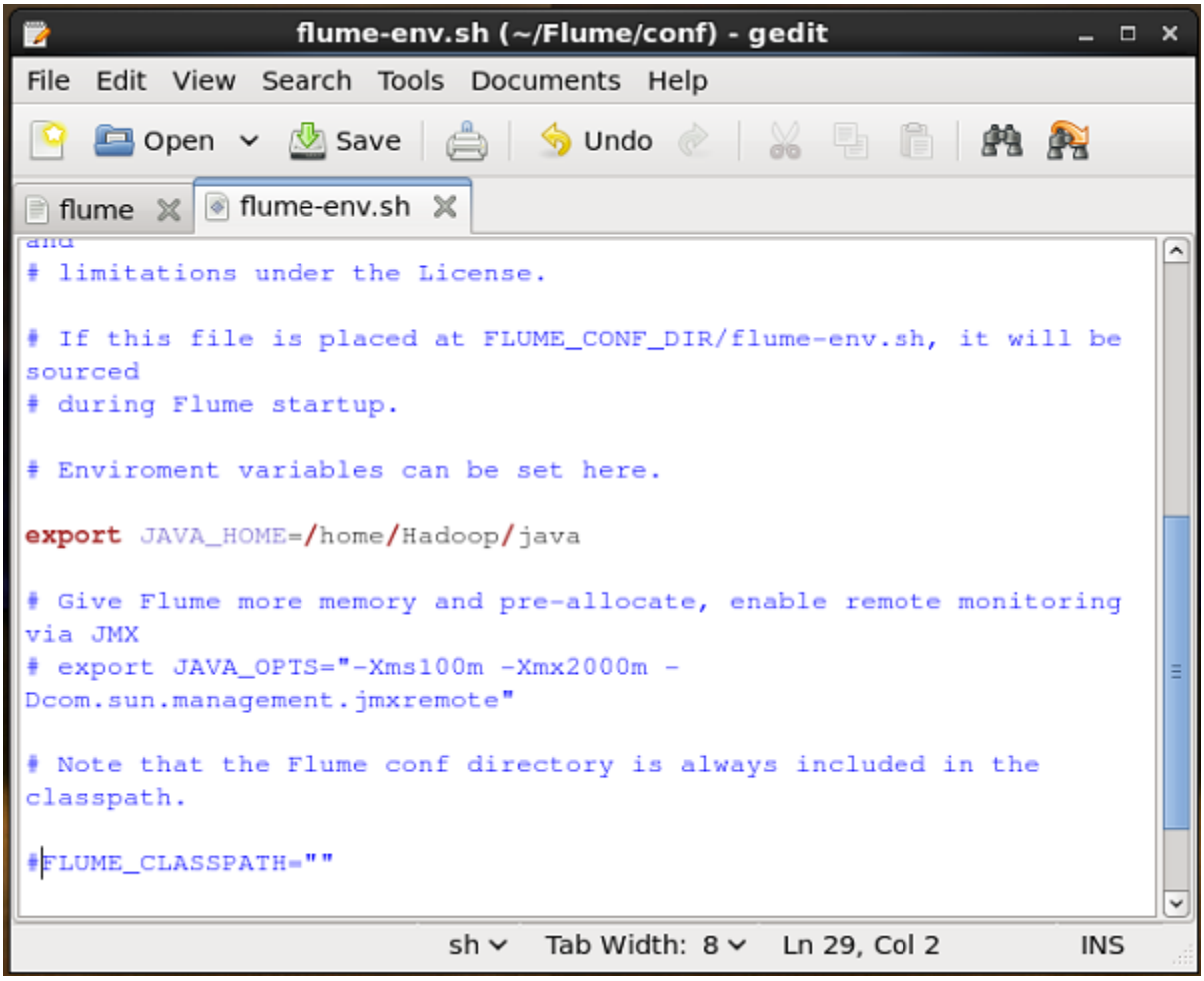


Now rename

- **flume-conf.properties.template** file as **flume-conf.properties** and
- **flume-env.sh.template** as **flume-env.sh**

flume-env.sh

Open **flume-env.sh** file and set the **JAVA_Home** to the folder where Java was installed in your system.



Verifying the Installation

Verify the installation of Apache Flume by browsing through the **bin** folder and typing the following command.

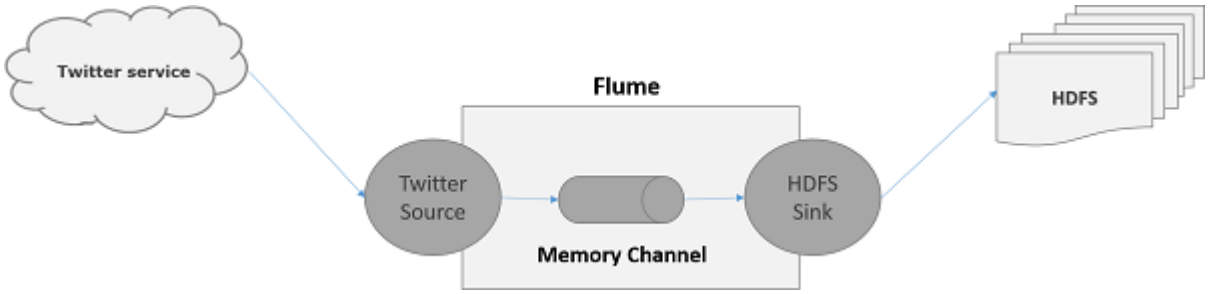
$ ./flume-ng

If you have successfully installed Flume, you will get a help prompt of Flume as shown below.

FETCHING TWITTER DATA

Using Flume, we can fetch data from various services and transport it to centralized stores (HDFS and HBase). a webserver generates log data and this data is collected by an agent in Flume. The channel buffers this data to a sink, which finally pushes it to centralized stores.

We will create an application and get the tweets from it using the experimental twitter source provided by Apache Flume. We will use the memory channel to buffer these tweets and HDFS sink to push these tweets into the HDFS.



To fetch Twitter data, we will have to follow the steps given below −

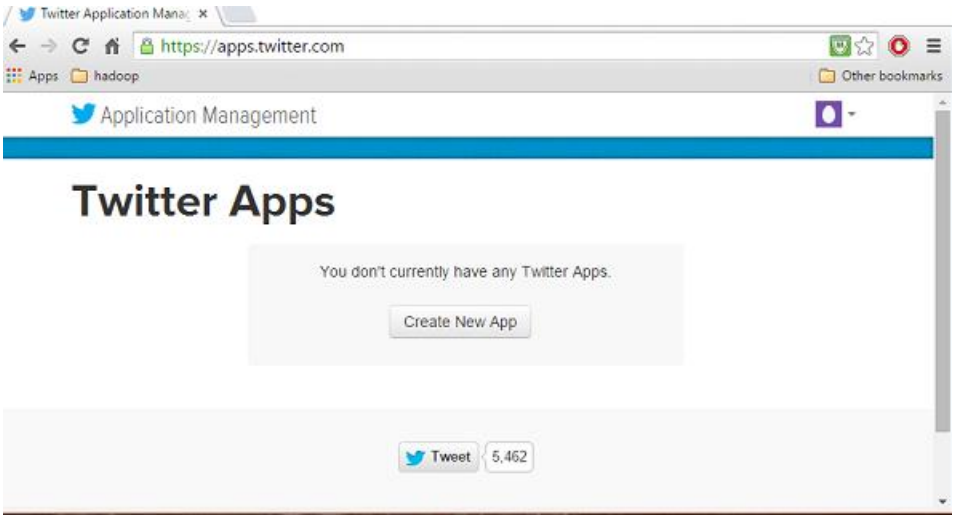
- Create a twitter Application
- Install / Start HDFS
- Configure Flume

Creating a Twitter Application

In order to get the tweets from Twitter, it is needed to create a Twitter application. Following are the steps that needs to be followed to create a Twitter application.

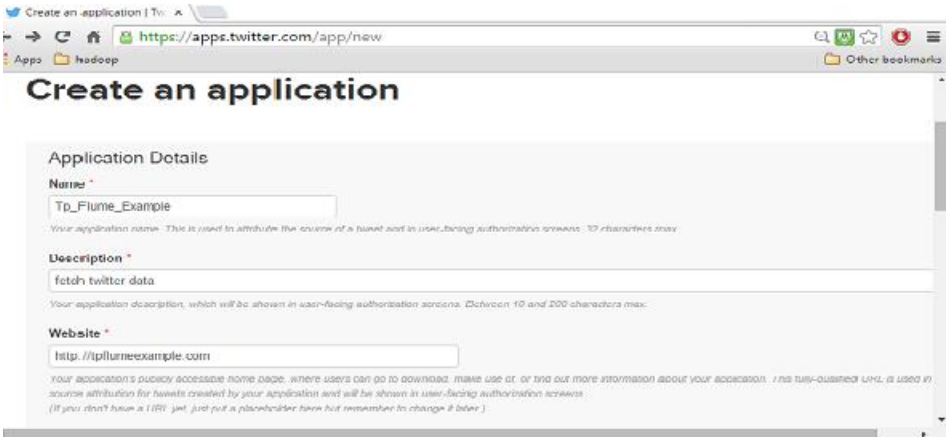
Step 1

To create a Twitter application, go to https://apps.twitter.com/. Sign in to your Twitter account. We will have a Twitter Application Management window where we can create, delete, and manage Twitter Apps.



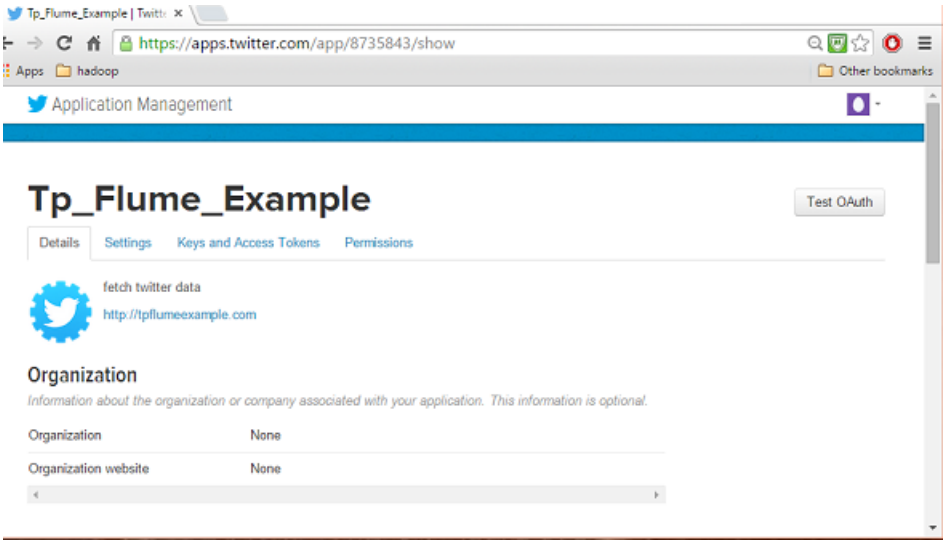
Step 2

Click on the **Create New App** button. We will be redirected to a window where we will get an application form in which we have to fill in your details in order to create the App. While filling the website address, give the complete URL pattern, for example, http://example.com.



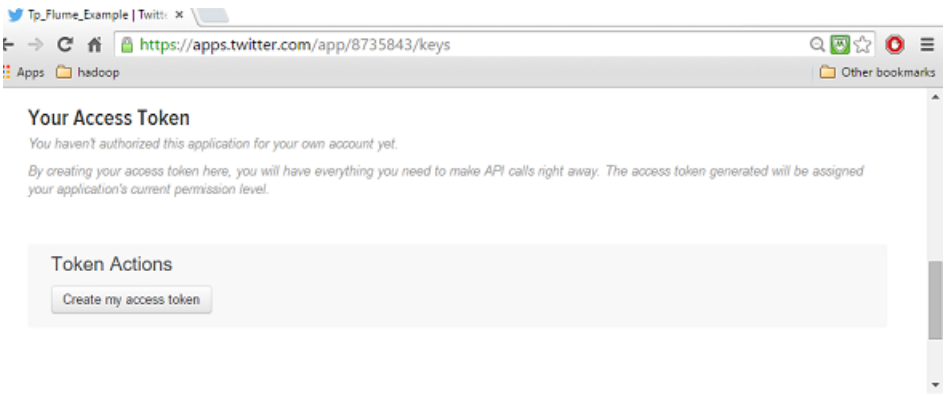
Step 3

Fill in the details, accept the **Developer Agreement** when finished, click on the **Create your Twitter application button** which is at the bottom of the page. If everything goes fine, an App will be created with the given details as shown below.



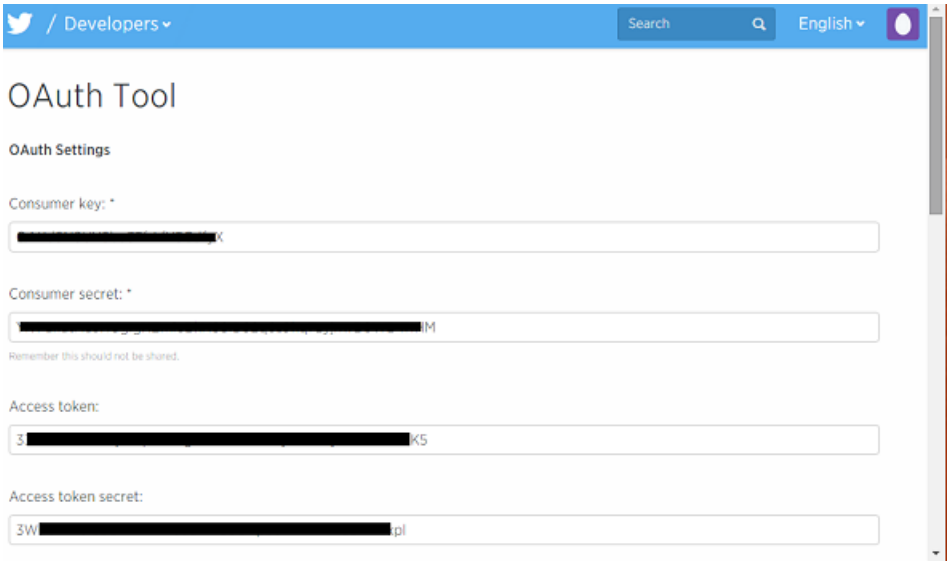
Step 4

Under **keys and Access Tokens** tab at the bottom of the page, we can observe a button named **Create my access token**. Click on it to generate the access token.



Step 5

Finally, click on the **Test OAuth** button which is on the right side top of the page. This will lead to a page which displays our **Consumer key, Consumer secret, Access token,** and **Access token secret**. Copy these details. These are useful to configure the agent in Flume.



Starting HDFS

Since we are storing the data in HDFS, we need to install / verify Hadoop. Start Hadoop and create a folder in it to store Flume data. Follow the steps given below before configuring Flume.

Step 1: Install / Verify Hadoop

Install Hadoop. If Hadoop is already installed in our system, verify the installation using Hadoop version command, as shown below.

```
$ hadoop version
```

Step 2: Starting Hadoop

Browse through the **sbin** directory of Hadoop and start yarn and Hadoop dfs (distributed file system) as shown below.

```
cd /$Hadoop_Home/sbin/
$ start-dfs.sh
localhost: starting namenode, logging to
   /home/Hadoop/hadoop/logs/hadoop-Hadoop-namenode-localhost.localdomain.out
localhost: starting datanode, logging to
   /home/Hadoop/hadoop/logs/hadoop-Hadoop-datanode-localhost.localdomain.out
```

```
Starting secondary namenodes [0.0.0.0]
starting secondarynamenode, logging to
   /home/Hadoop/hadoop/logs/hadoop-Hadoop-secondarynamenode-localhost.localdomain.out

$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to
   /home/Hadoop/hadoop/logs/yarn-Hadoop-resourcemanager-localhost.localdomain.out
localhost: starting nodemanager, logging to
   /home/Hadoop/hadoop/logs/yarn-Hadoop-nodemanager-localhost.localdomain.out
```

Step 3: Create a Directory in HDFS

In Hadoop DFS, we can create directories using the command **mkdir**. Browse through it and create a directory with the name **twitter_data** in the required path as shown below.

```
$cd /$Hadoop_Home/bin/
$ hdfs dfs -mkdir hdfs://localhost:9000/user/Hadoop/twitter_data
```

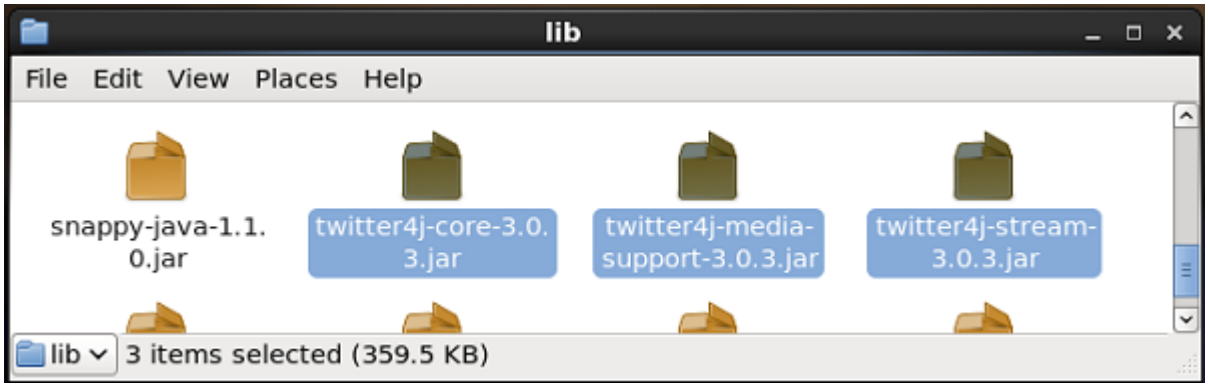
Configuring Flume

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder. The example uses an experimental source provided by Apache Flume named **Twitter 1% Firehose** Memory channel and HDFS sink.

Twitter 1% Firehose Source

This source is highly experimental. It connects to the 1% sample Twitter Firehose using streaming API and continuously downloads tweets, converts them to Avro format, and sends Avro events to a downstream Flume sink.

We will get this source by default along with the installation of Flume. The **jar** files corresponding to this source can be located in the **lib** folder as shown below.



Setting the classpath

Set the **classpath** variable to the **lib** folder of Flume in **Flume-env.sh** file as shown below.

export CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*

This source needs the details such as **Consumer key, Consumer secret, Access token,** and **Access token secret** of a Twitter application. While configuring this source, we have to provide values to the following properties −

- **Channels**
- **Source type : org.apache.flume.source.twitter.TwitterSource**
- **consumerKey** − The OAuth consumer key
- **consumerSecret** − OAuth consumer secret
- **accessToken** − OAuth access token
- **accessTokenSecret** − OAuth token secret
- **maxBatchSize** − Maximum number of twitter messages that should be in a twitter batch. The default value is 1000 (optional).
- **maxBatchDurationMillis** − Maximum number of milliseconds to wait before closing a batch. The default value is 1000 (optional).

Channel:- We are using the memory channel. To configure the memory channel, we *must* provide value to the type of the channel.

- **type** − It holds the type of the channel. In our example, the type is **MemChannel**.
- **Capacity** − It is the maximum number of events stored in the channel. Its default value is 100 (optional).
- **TransactionCapacity** − It is the maximum number of events the channel accepts or sends. Its default value is 100 (optional).

HDFS Sink:- This sink writes data into the HDFS. To configure this sink, we *must* provide the following details.

- **Channel**
- **type** − hdfs
- **hdfs.path** − the path of the directory in HDFS where data is to be stored.

And we can provide some optional values based on the scenario. Given below are the optional properties of the HDFS sink that we are configuring in our application.

- **fileType** − This is the required file format of our HDFS file. **SequenceFile, DataStream** and **CompressedStream** are the three types available with this stream. In our example, we are using the **DataStream**.
- **writeFormat** − Could be either text or writable.
- **batchSize** − It is the number of events written to a file before it is flushed into the HDFS. Its default value is 100.
- **rollsize** − It is the file size to trigger a roll. It default value is 100.
- **rollCount** − It is the number of events written into the file before it is rolled. Its default value is 10.

Example – Configuration File:- Given below is an example of the configuration file. Copy this content and save as **twitter.conf** in the conf folder of Flume.

```
# Naming the components on the current agent.
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

# Describing/Configuring the source
TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource
TwitterAgent.sources.Twitter.consumerKey = Your OAuth consumer key
TwitterAgent.sources.Twitter.consumerSecret = Your OAuth consumer secret
TwitterAgent.sources.Twitter.accessToken = Your OAuth consumer key access token
TwitterAgent.sources.Twitter.accessTokenSecret = Your OAuth consumer key access token secret
TwitterAgent.sources.Twitter.keywords = tutorials point,java, bigdata, mapreduce, mahout, hbase, nosql

# Describing/Configuring the sink

TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/Hadoop/twitter_data/
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

# Describing/Configuring the channel
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100
```

Binding the source and sink to the channel

TwitterAgent.sources.Twitter.channels = MemChannel

TwitterAgent.sinks.HDFS.channel = MemChannel

Execution

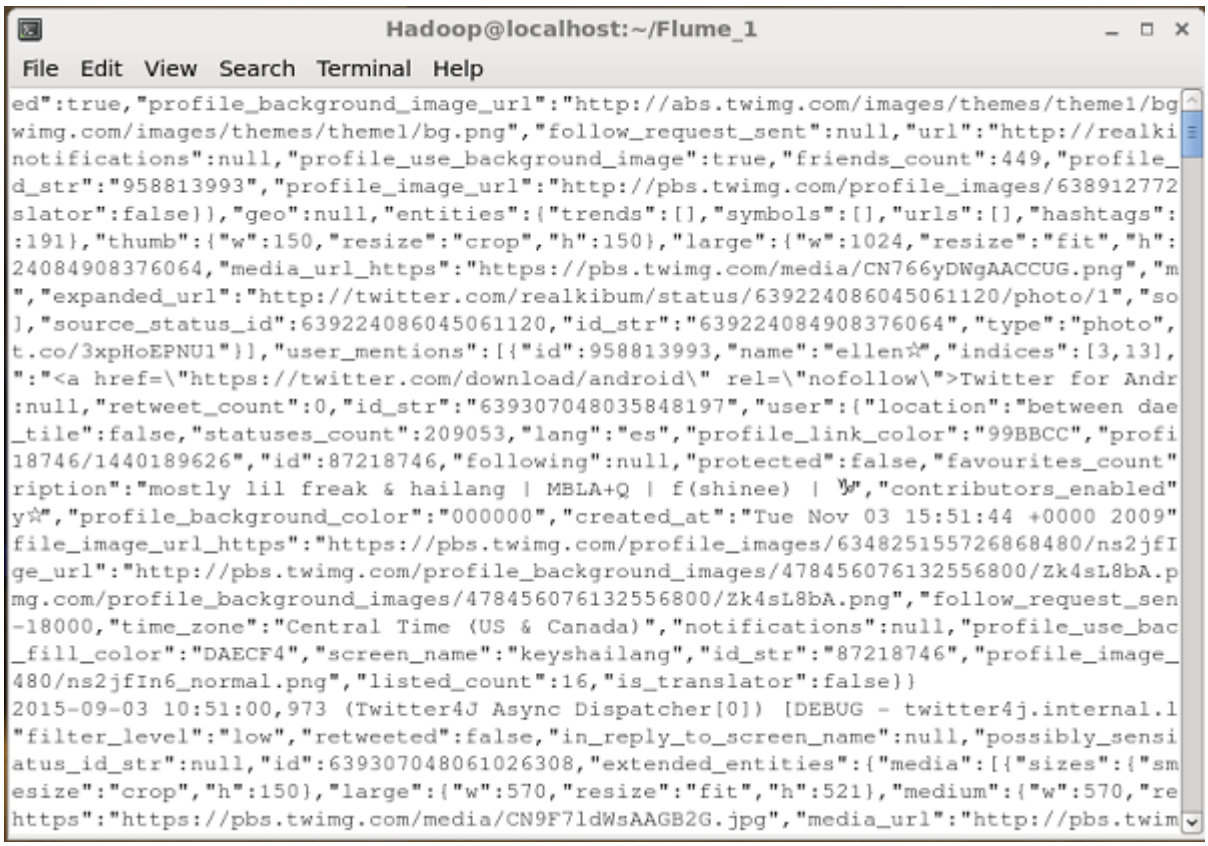
Browse through the Flume home directory and execute the application as shown below.

$ cd $FLUME_HOME

$ bin/flume-ng agent --conf ./conf/ -f conf/twitter.conf

Dflume.root.logger=DEBUG,console -n TwitterAgent

If everything goes fine, the streaming of tweets into HDFS will start. Given below is the snapshot of the command prompt window while fetching tweets.

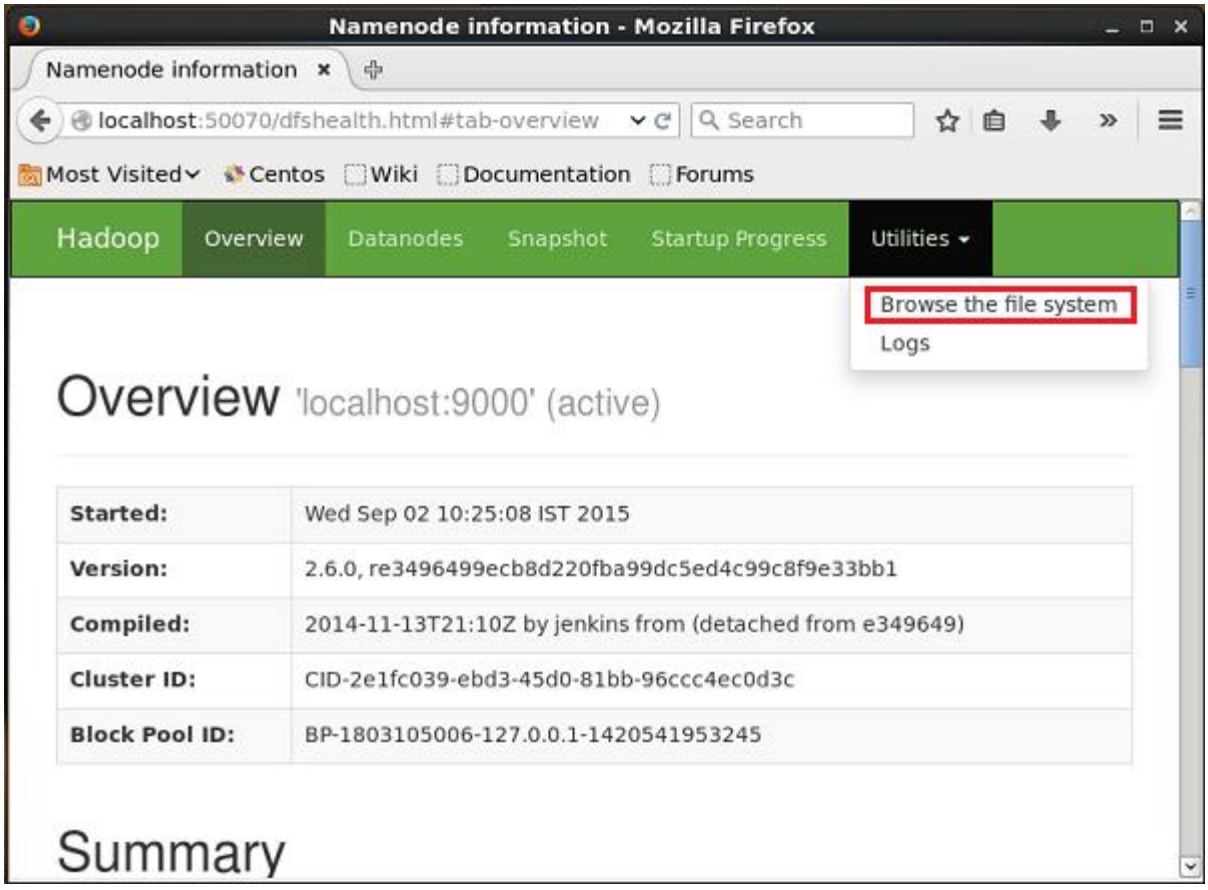


VERIFYING HDFS

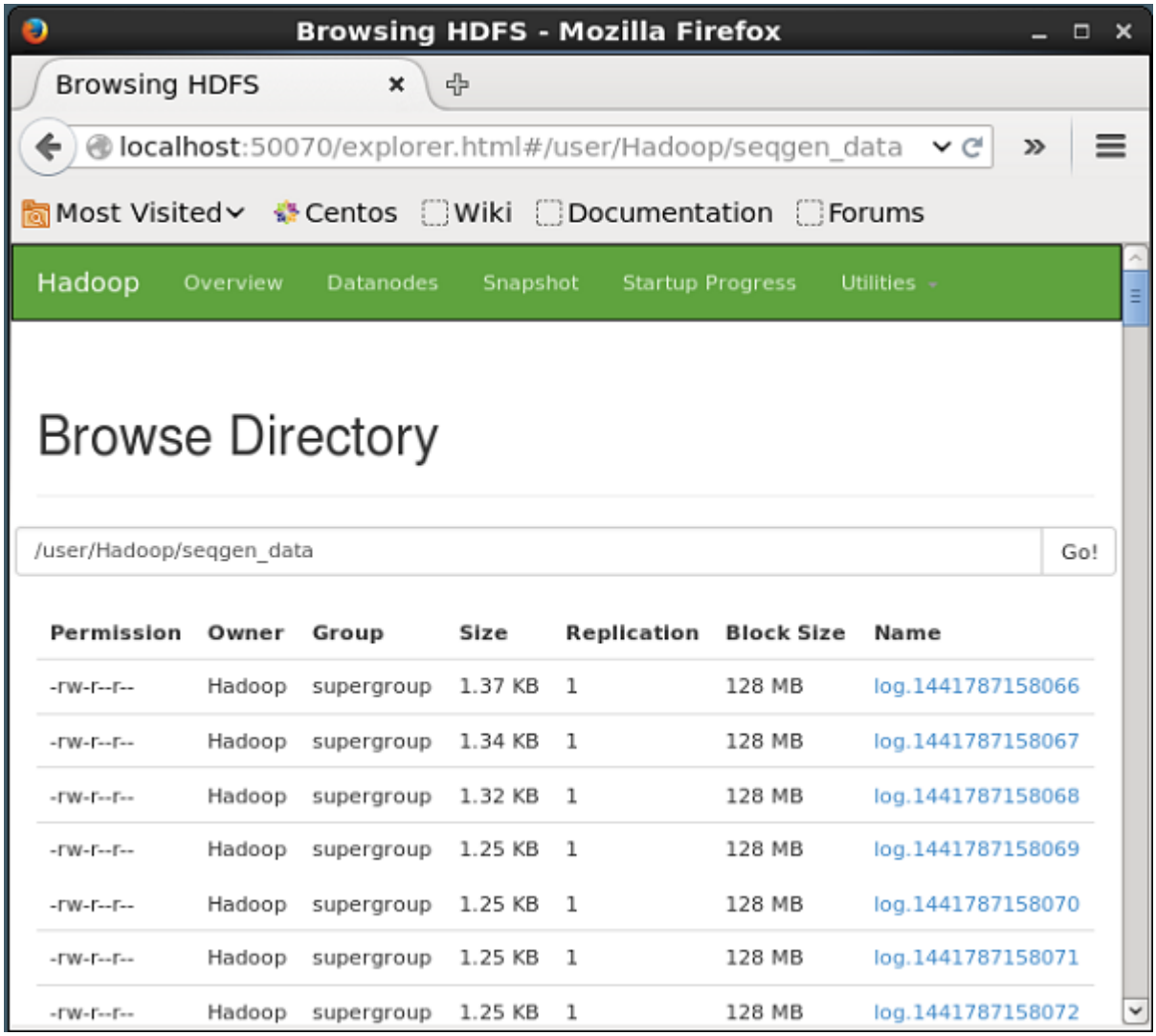
We can access the Hadoop Administration Web UI using the URL given below.

http://localhost:50070/

Click on the dropdown named **Utilities** on the right-hand side of the page. We can see two options as shown in the snapshot given below.



Click on **Browse the file system** and enter the path of the HDFS directory where we have stored the tweets. In our example, the path will be **/user/Hadoop/twitter_data/**. Then, we can see the list of twitter log files stored in HDFS as given below.



Configuring Flume

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder. The example uses a **sequence generator source**, a **memory channel**, and an **HDFS sink**.

Sequence Generator Source

It is the source that generates the events continuously. It maintains a counter that starts from 0 and increments by 1. It is used for testing purpose. While configuring this source, you must provide values to the following properties −

- **Channels**
- **Source type** − seq

Channel

We are using the **memory** channel. To configure the memory channel, we *must* provide a value to the type of the channel. Given below are the list of properties that we need to supply while configuring the memory channel −

- **type** − It holds the type of the channel. In our example the type is MemChannel.
- **Capacity** − It is the maximum number of events stored in the channel. Its default value is 100. (optional)
- **TransactionCapacity** − It is the maximum number of events the channel accepts or sends. Its default is 100. (optional).

HDFS Sink

This sink writes data into the HDFS. To configure this sink, we *must* provide the following details.

- **Channel**
- **type** − hdfs
- **hdfs.path** − the path of the directory in HDFS where data is to be stored.

And we can provide some optional values based on the scenario. Given below are the optional properties of the HDFS sink that we are configuring in our application.

- **fileType** − This is the required file format of our HDFS file. **SequenceFile, DataStream** and **CompressedStream** are the three types available with this stream. In our example, we are using the **DataStream**.
- **writeFormat** − Could be either text or writable.

- **batchSize** − It is the number of events written to a file before it is flushed into the HDFS. Its default value is 100.
- **rollsize** − It is the file size to trigger a roll. It default value is 100.
- **rollCount** − It is the number of events written into the file before it is rolled. Its default value is 10.

Example – Configuration File

Given below is an example of the configuration file. Copy this content and save as **seq_gen .conf** in the conf folder of Flume.

```
# Naming the components on the current agent

SeqGenAgent.sources = SeqSource
SeqGenAgent.channels = MemChannel
SeqGenAgent.sinks = HDFS

# Describing/Configuring the source
SeqGenAgent.sources.SeqSource.type = seq

# Describing/Configuring the sink
SeqGenAgent.sinks.HDFS.type = hdfs
SeqGenAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/Hadoop/seqgen_data/
SeqGenAgent.sinks.HDFS.hdfs.filePrefix = log
SeqGenAgent.sinks.HDFS.hdfs.rollInterval = 0
SeqGenAgent.sinks.HDFS.hdfs.rollCount = 10000
SeqGenAgent.sinks.HDFS.hdfs.fileType = DataStream

# Describing/Configuring the channel
SeqGenAgent.channels.MemChannel.type = memory
SeqGenAgent.channels.MemChannel.capacity = 1000
SeqGenAgent.channels.MemChannel.transactionCapacity = 100
```

Binding the source and sink to the channel

SeqGenAgent.sources.SeqSource.channels = MemChannel

SeqGenAgent.sinks.HDFS.channel = MemChannel

Execution

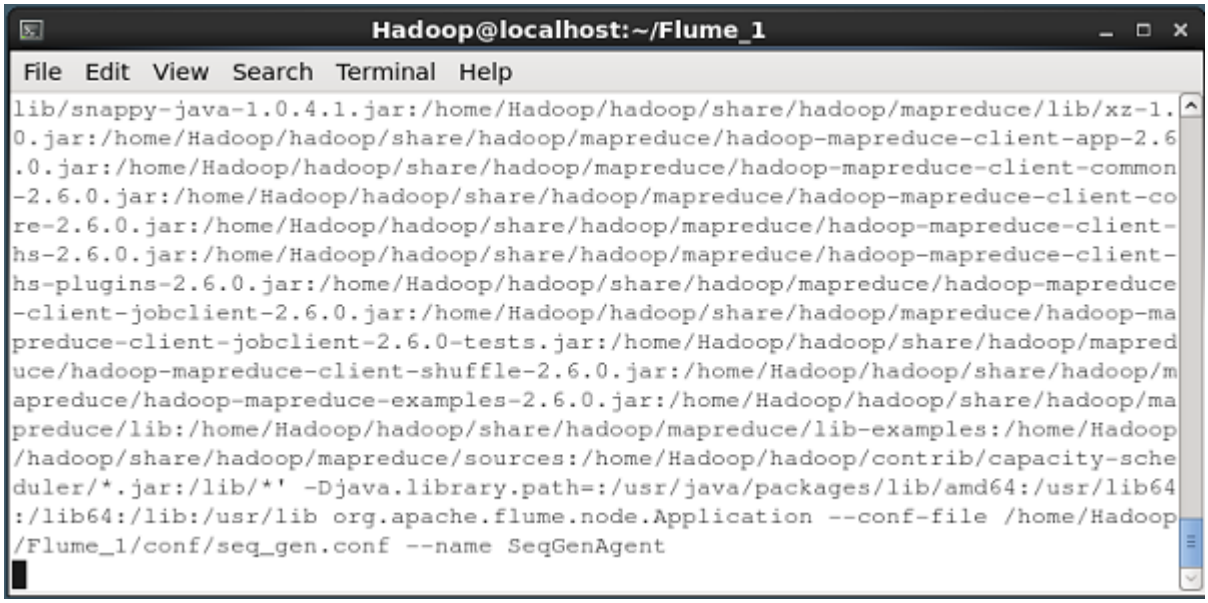
Browse through the Flume home directory and execute the application as shown below.

$ cd $FLUME_HOME

$./bin/flume-ng agent --conf $FLUME_CONF --conf-file $FLUME_CONF/seq_gen.conf
--name SeqGenAgent

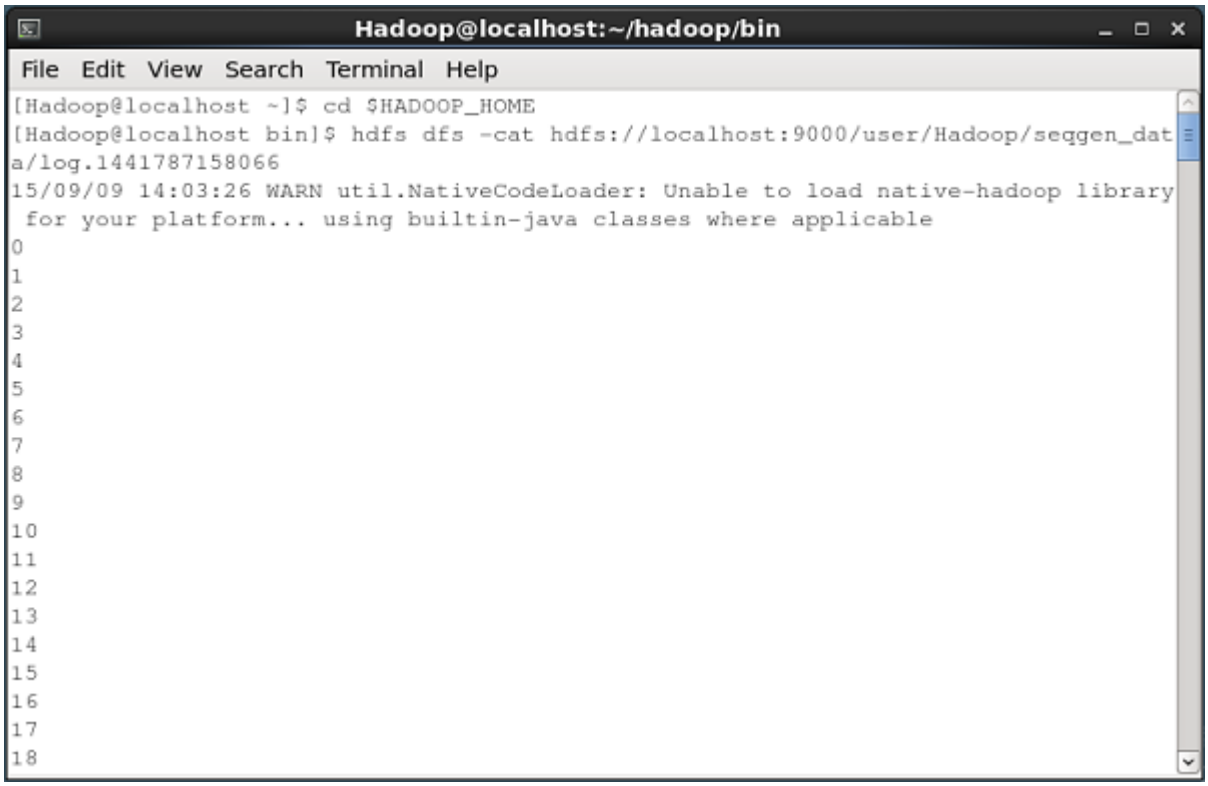
If everything goes fine, the source starts generating sequence numbers which will be pushed into the HDFS in the form of log files.

Given below is a snapshot of the command prompt window fetching the data generated by the sequence generator into the HDFS.



Verifying the Contents of the File

All these log files contain numbers in sequential format. We can verify the contents of these file in the file system using the **cat** command as shown below.

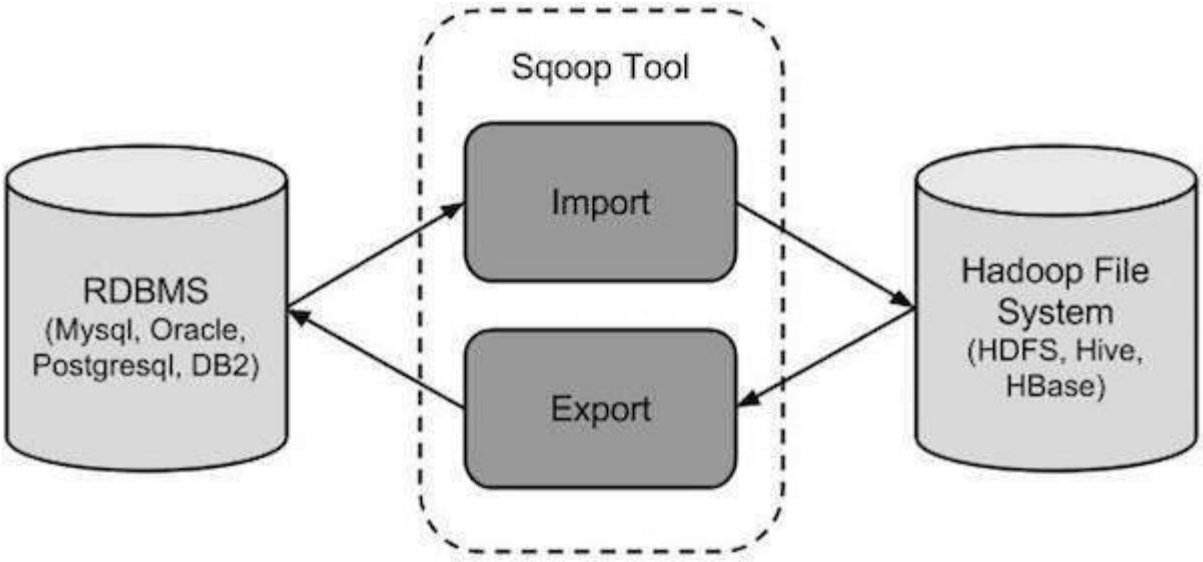


SQOOP

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases.

HOW SQOOP WORKS?

The following image describes the workflow of Sqoop.



Sqoop Import

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

Downloading Sqoop

We can download the latest version of Sqoop. We are using version 1.4.5, that is, **sqoop-1.4.5.bin__hadoop-2.0.4-alpha.tar.gz**.

Installing Sqoop

The following commands are used to extract the Sqoop tar ball and move it to "/usr/lib/sqoop" directory.

```
$tar -xvf sqoop-1.4.4.bin__hadoop-2.0.4-alpha.tar.gz
$ su
password:

# mv sqoop-1.4.4.bin__hadoop-2.0.4-alpha /usr/lib/sqoop
#exit
```

Configuring bashrc

We have to set up the Sqoop environment by appending the following lines to ~/**.bashrc** file −

```
#Sqoop
export SQOOP_HOME=/usr/lib/sqoop export PATH=$PATH:$SQOOP_HOME/bin
```

The following command is used to execute ~/**.bashrc** file.

```
$ source ~/.bashrc
```

Configuring Sqoop

To configure Sqoop with Hadoop, we need to edit the **sqoop-env.sh** file, which is placed in the **$SQOOP_HOME/conf** directory. First of all, Redirect to Sqoop config directory and copy the template file using the following command −

```
$ cd $SQOOP_HOME/conf
```

```
$ mv sqoop-env-template.sh sqoop-env.sh
```

Open **sqoop-env.sh** and edit the following lines −

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Download and Configure mysql-connector-java

We can download **mysql-connector-java-5.1.30.tar.gz** file.

The following commands are used to extract mysql-connector-java tarball and move **mysql-connector-java-5.1.30-bin.jar** to /usr/lib/sqoop/lib directory.

```
$ tar -zxf mysql-connector-java-5.1.30.tar.gz
$ su
password:

# cd mysql-connector-java-5.1.30
# mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

Verifying Sqoop

The following command is used to verify the Sqoop version.

```
$ cd $SQOOP_HOME/bin
$ sqoop-version
```

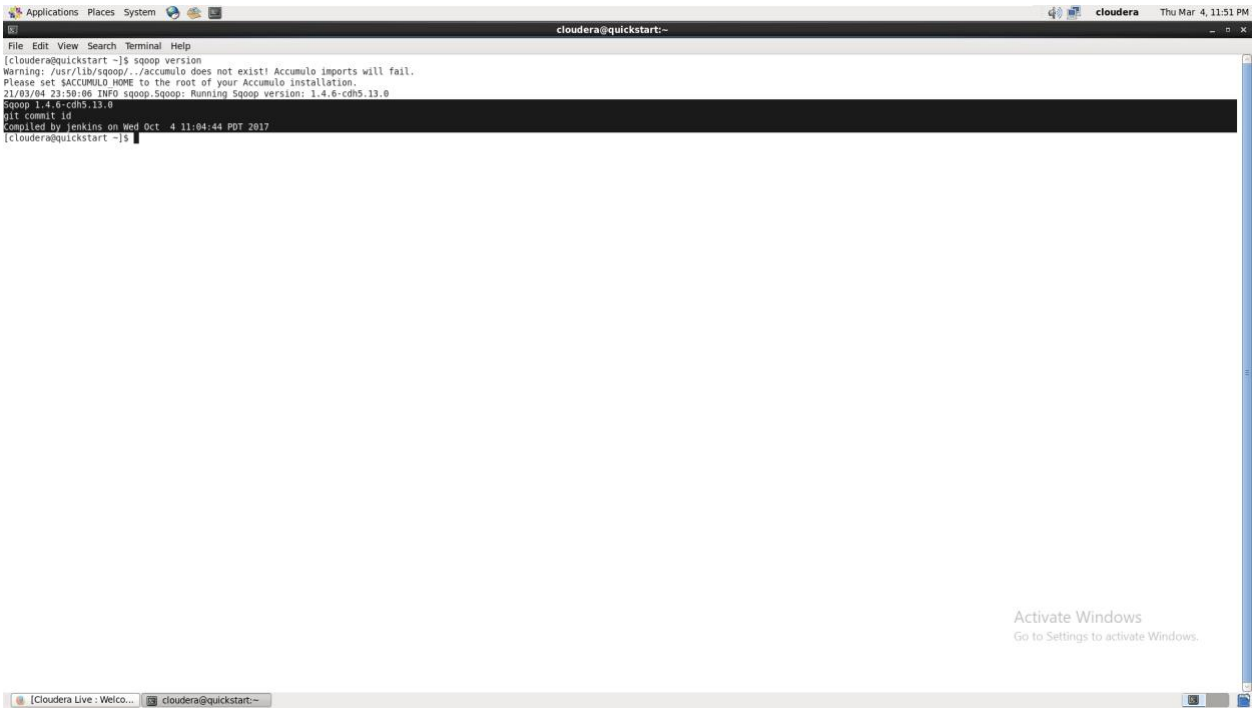
Expected output −

14/12/17 14:52:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2eddbe83
Compiled by abe on Fri Aug 1 11:19:26 PDT 2014

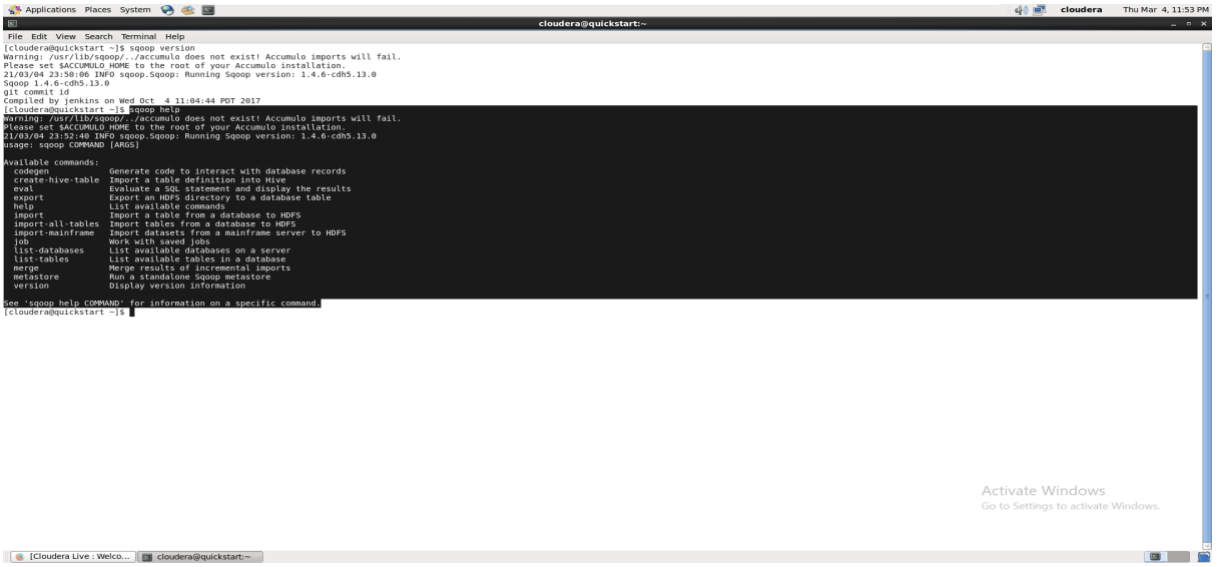
Sqoop installation is complete.

On Cloudera VM

If we are using cloudera,then check if all the demons are working using jps command and then open terminal to check sqoop version.

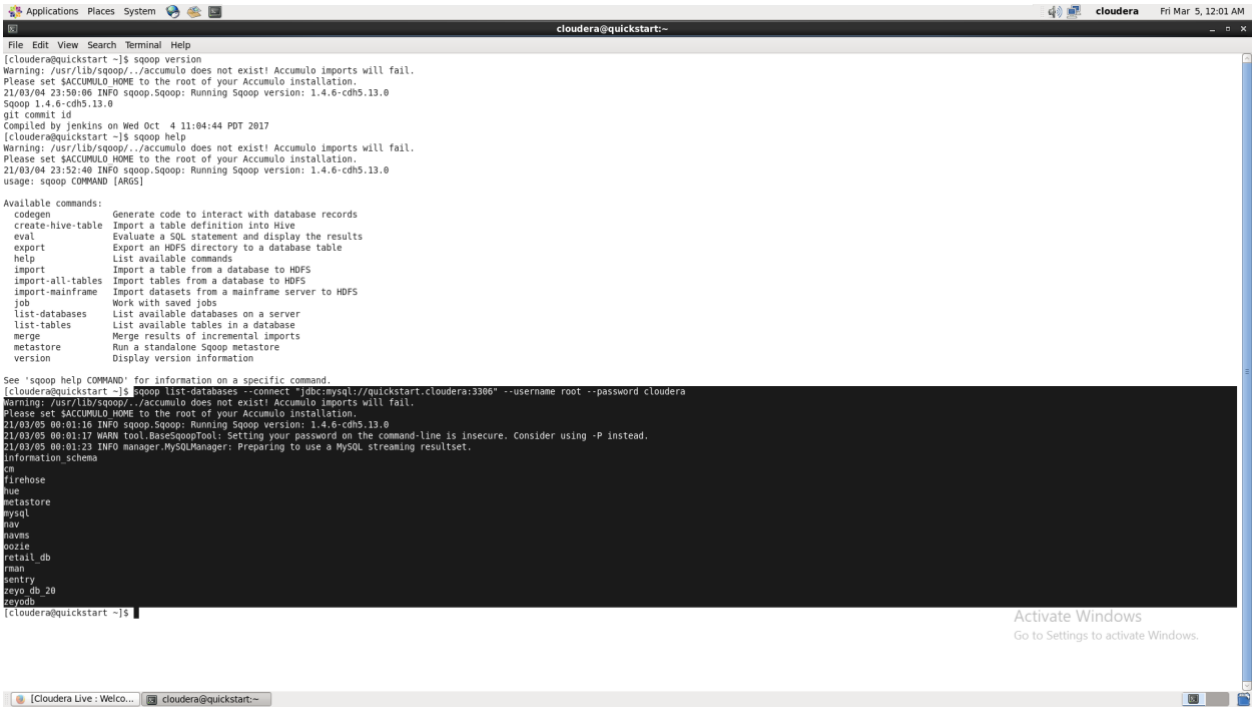


If we want any information regarding sqoop commands we can use the **sqoop help** command.



Use following command to connect to database

sqoop list-databases –connect “jdbc://quickstart.cloudera:3306” –username root –password cloudera



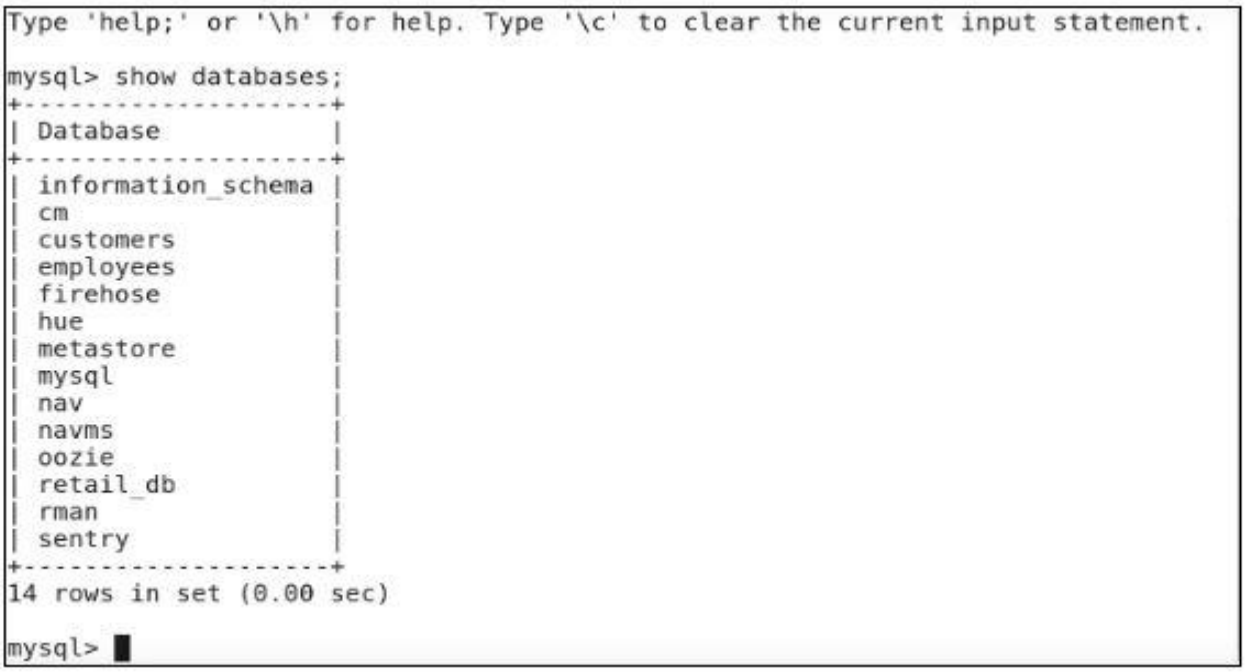
To connect with mysql

mysql –uroot -pcloudera

After typing the above commands, you will get into MySQL.

show databases;

This will display a list of 12 databases present in MySQL



As seen in the above image, MySQL comes with a standard set of databases. We will use the **retail_db** database. To use this database and check the tables in it:

use retail_db;

By typing the command above, that database will be set as the default in MySQL. To check the list of tables present inside the **retail_db** database, we have to type the following command:

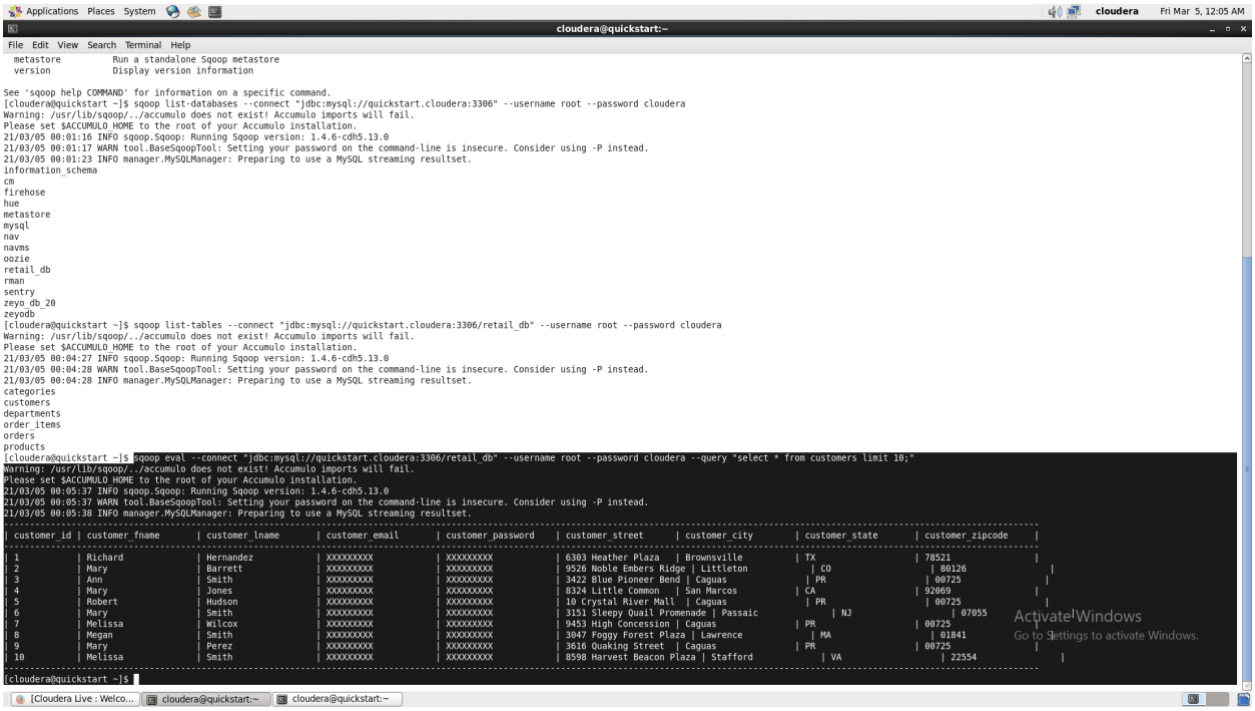
show tables;

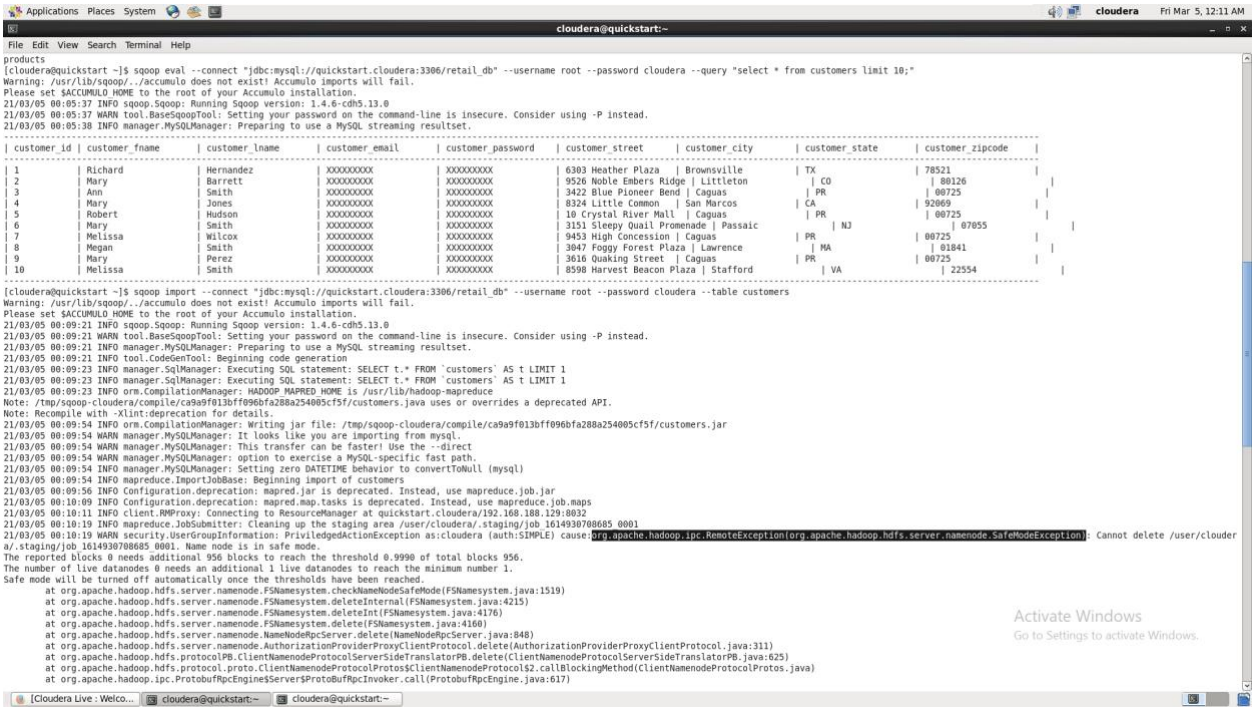
After this, open another terminal in Cloudera and type the following:

hostname -f //Checks the hostname

sqoop list-databases --connect jdbc:mysql://localhost/ --password cloudera --username root;

After this, we would have to connect to the MySQL database and import the customer's database to HDFS. sqoop eval --connect jdbc:mysql://localhost/ --password cloudera --username root –query select * fro customers limit 10"




Applications Places System
cloudera Fri Mar 5, 12:11 AM
cloudera@quickstart:~
File Edit View Search Terminal Help
products
[cloudera@quickstart ~]$ sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username root --password cloudera --query "select * from customers limit 10;"
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
21/03/05 00:05:37 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
21/03/05 00:05:37 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
21/03/05 00:05:38 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
customer_id	customer_fname	customer_lname	customer_email	customer_password	customer_street	customer_city	customer_state	customer_zipcode
1	Richard	Hernandez	XXXXXXXXX	XXXXXXXXX	6303 Heather Plaza	Brownsville	TX	78521
2	Mary	Barrett	XXXXXXXXX	XXXXXXXXX	9526 Noble Embers Ridge	Littleton	CO	80126
3	Ann	Smith	XXXXXXXXX	XXXXXXXXX	3422 Blue Pioneer Bend	Caguas	PR	00725
4	Mary	Jones	XXXXXXXXX	XXXXXXXXX	8324 Little Common	San Marcos	CA	92069
5	Robert	Hudson	XXXXXXXXX	XXXXXXXXX	10 Crystal River Mall	Caguas	PR	00725
6	Mary	Smith	XXXXXXXXX	XXXXXXXXX	3151 Sleepy Quail Promenade	Passaic	NJ	07055
7	Melissa	Wilcox	XXXXXXXXX	XXXXXXXXX	9453 High Concession	Caguas	PR	00725
8	Megan	Smith	XXXXXXXXX	XXXXXXXXX	3047 Foggy Forest Plaza	Lawrence	MA	01841
9	Mary	Perez	XXXXXXXXX	XXXXXXXXX	3616 Quaking Street	Caguas	PR	00725
10	Melissa	Smith	XXXXXXXXX	XXXXXXXXX	8598 Harvest Beacon Plaza	Stafford	VA	22554
[cloudera@quickstart ~]$ sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username root --password cloudera --table customers
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
21/03/05 00:09:21 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
21/03/05 00:09:21 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
21/03/05 00:09:21 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
21/03/05 00:09:21 INFO tool.CodeGenTool: Beginning code generation
21/03/05 00:09:23 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `customers` AS t LIMIT 1
21/03/05 00:09:23 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `customers` AS t LIMIT 1
21/03/05 00:09:23 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-cloudera/compile/ca9a9f013bff096bfa288a254005cf5f/customers.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
21/03/05 00:09:54 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-cloudera/compile/ca9a9f013bff096bfa288a254005cf5f/customers.jar
21/03/05 00:09:54 WARN manager.MySQLManager: It looks like you are importing from mysql.
21/03/05 00:09:54 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
21/03/05 00:09:54 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
21/03/05 00:09:54 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
21/03/05 00:09:54 INFO mapreduce.ImportJobBase: Beginning import of customers
21/03/05 00:09:56 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
21/03/05 00:10:09 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
21/03/05 00:10:11 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/192.168.188.129:8032
21/03/05 00:10:19 INFO mapreduce.JobSubmitter: Cleaning up the staging area /user/cloudera/.staging/job_1614930708685_0001
21/03/05 00:10:19 WARN security.UserGroupInformation: PriviledgedActionException as:cloudera (auth:SIMPLE) cause:org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.hdfs.server.namenode.SafeModeException): Cannot delete /user/cloudera/.staging/job_1614930708685_0001. Name node is in safe mode.
The reported blocks 0 needs additional 956 blocks to reach the threshold 0.9990 of total blocks 956.
The number of live datanodes 0 needs an additional 1 live datanodes to reach the minimum number 1.
Safe mode will be turned off automatically once the thresholds have been reached.
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkNameNodeSafeMode(FSNamesystem.java:1519)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.deleteInternal(FSNamesystem.java:4215)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.deleteInt(FSNamesystem.java:4176)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.delete(FSNamesystem.java:4160)
at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.delete(NameNodeRpcServer.java:848)
at org.apache.hadoop.hdfs.server.namenode.AuthorizationProviderProxyClientProtocol.delete(AuthorizationProviderProxyClientProtocol.java:311)
at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolServerSideTranslatorPB.delete(ClientNamenodeProtocolServerSideTranslatorPB.java:625)
at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos$ClientNamenodeProtocol$2.callBlockingMethod(ClientNamenodeProtocolProtos.java)
at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java:617)
Activate Windows
Go to Settings to activate Windows.
[Cloudera Live : Welco...
cloudera@quickstart:~
cloudera@quickstart:~


Applications Places System
cloudera Fri Mar 5, 12:11 AM
cloudera@quickstart:~
File Edit View Search Terminal Help
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.deleteInt(FSNamesystem.java:4176)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.delete(FSNamesystem.java:4160)
at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.delete(NameNodeRpcServer.java:848)
at org.apache.hadoop.hdfs.server.namenode.AuthorizationProviderProxyClientProtocol.delete(AuthorizationProviderProxyClientProtocol.java:311)
at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolServerSideTranslatorPB.delete(ClientNamenodeProtocolServerSideTranslatorPB.java:625)
at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos$ClientNamenodeProtocol$2.callBlockingMethod(ClientNamenodeProtocolProtos.java)
at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java:617)
at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:1073)
at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2226)
at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2222)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:422)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1917)
at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2220)
at org.apache.hadoop.ipc.Client.call(Client.java:1504)
at org.apache.hadoop.ipc.Client.call(Client.java:1441)
at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.invoke(ProtobufRpcEngine.java:230)
at com.sun.proxy.$Proxy11.delete(Unknown Source)
at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.delete(ClientNamenodeProtocolTranslatorPB.java:550)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInvocationHandler.java:260)
at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:104)
at com.sun.proxy.$Proxy12.delete(Unknown Source)
at org.apache.hadoop.hdfs.DFSClient.delete(DFSClient.java:2067)
at org.apache.hadoop.hdfs.DistributedFileSystem$13.doCall(DistributedFileSystem.java:687)
at org.apache.hadoop.hdfs.DistributedFileSystem$13.doCall(DistributedFileSystem.java:683)
at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
at org.apache.hadoop.hdfs.DistributedFileSystem.delete(DistributedFileSystem.java:683)
at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:255)
at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1307)
at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1304)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:422)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1917)
at org.apache.hadoop.mapreduce.Job.submit(Job.java:1304)
at org.apache.hadoop.mapreduce.Job.waitForCompletion(Job.java:1325)
at org.apache.sqoop.mapreduce.ImportJobBase.doSubmitJob(ImportJobBase.java:203)
at org.apache.sqoop.mapreduce.ImportJobBase.runJob(ImportJobBase.java:176)
at org.apache.sqoop.mapreduce.ImportJobBase.runImport(ImportJobBase.java:273)
at org.apache.sqoop.manager.SqlManager.importTable(SqlManager.java:692)
at org.apache.sqoop.manager.MySQLManager.importTable(MySQLManager.java:127)
at org.apache.sqoop.tool.ImportTool.importTable(ImportTool.java:513)
at org.apache.sqoop.tool.ImportTool.run(ImportTool.java:621)
at org.apache.sqoop.Sqoop.run(Sqoop.java:147)
at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:70)
at org.apache.sqoop.Sqoop.runSqoop(Sqoop.java:183)
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:234)
at org.apache.sqoop.Sqoop.runTool(Sqoop.java:243)
at org.apache.sqoop.Sqoop.main(Sqoop.java:252)
[cloudera@quickstart ~]$ sudo -u hdfs hdfs dfsadmin -safemode leave
Safe mode is OFF
[cloudera@quickstart ~]$
Activate Windows
Go to Settings to activate Windows.
[Cloudera Live : Welco...
cloudera@quickstart:~
cloudera@quickstart:~