Aim: To Install Cloudera Quickstart VM on VirtualBox

Cloudera is a software that provides a platform for data analytics, data warehousing, and machine learning. Initially, Cloudera started as an open-source Apache Hadoop distribution project, commonly known as Cloudera Distribution for Hadoop or CDH. It contains Apache Hadoop and other related projects where all the components are 100% open-source under Apache License.

Cloudera provides virtual machine images of complete Apache Hadoop clusters, making it easy to get started with Cloudera CDH. The following topics will be covered in this assignment on Cloudera QuickStart VM Installation.

1. What is Cloudera QuickStart VM?
2. Cloudera QuickStart VM Installation - Prerequisites
3. Downloading the Cloudera QuickStart VM
4. Cloudera QuickStart VM Installation on windows

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."

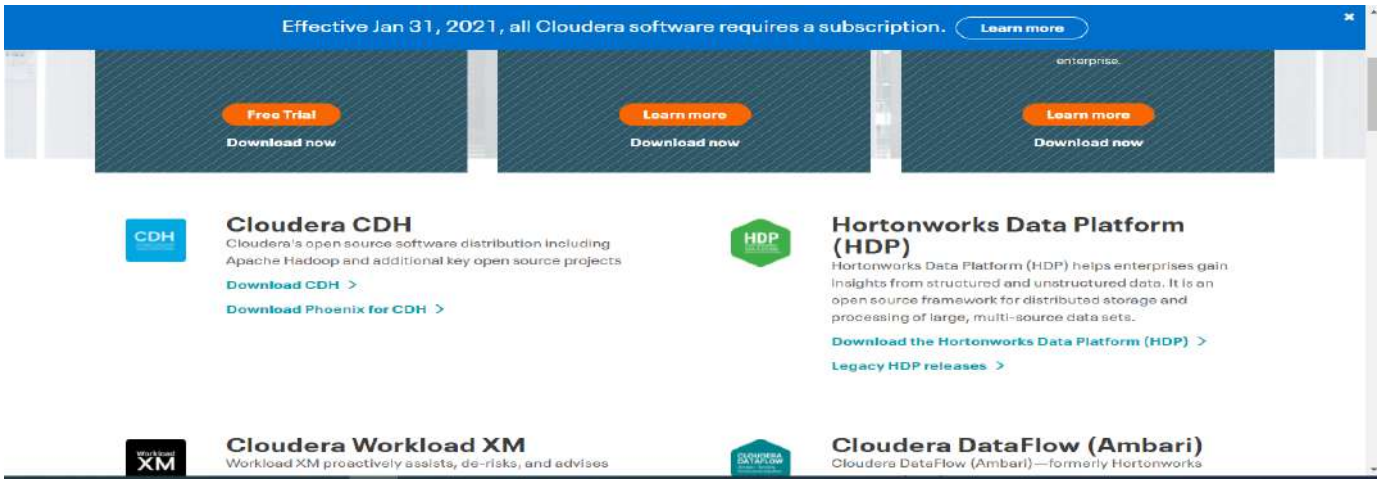
Cloudera QuickStart VM Installation - Prerequisites

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

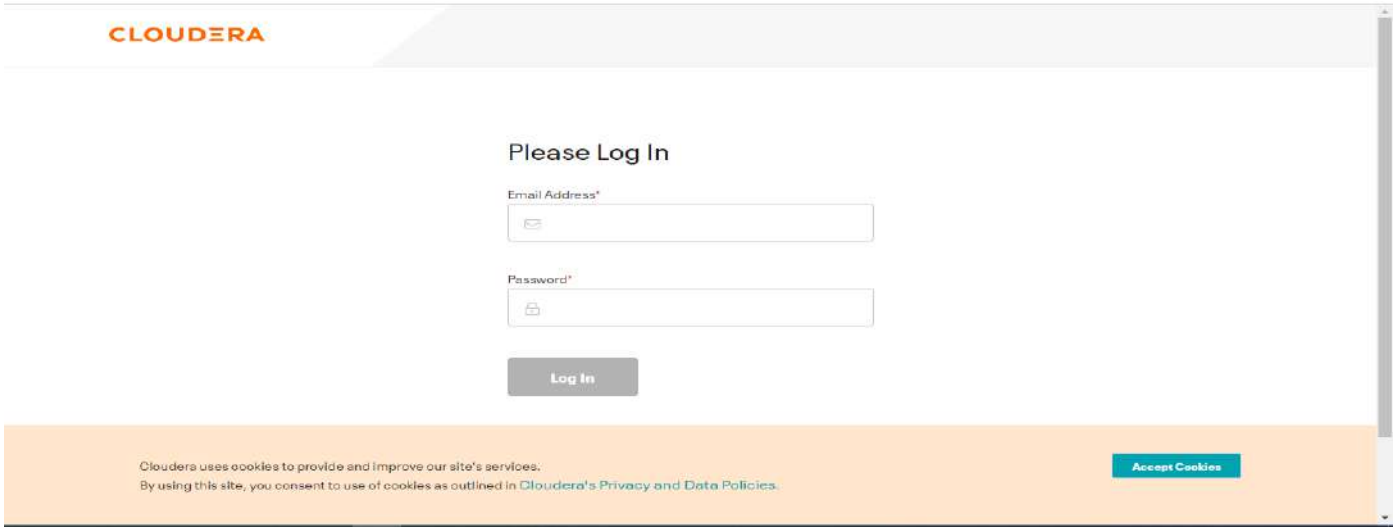
80GB hard disk

Download Oracle Virtual Box from https://www.virtualbox.org/wiki/Downloads and install it in your system

Link: https://www.cloudera.com/downloads.html

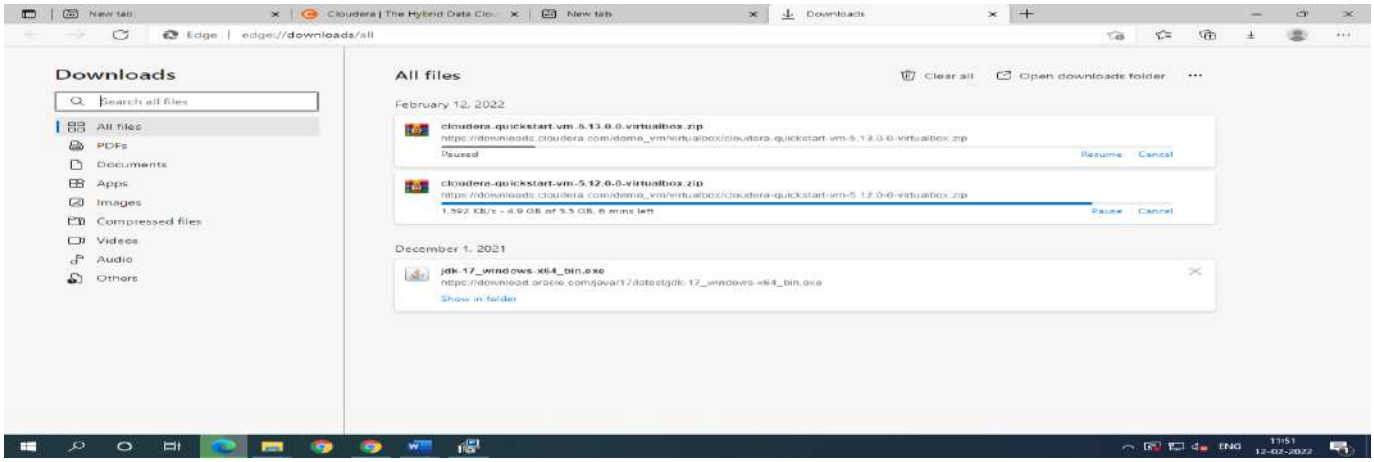


If already an account, then login else first register the account



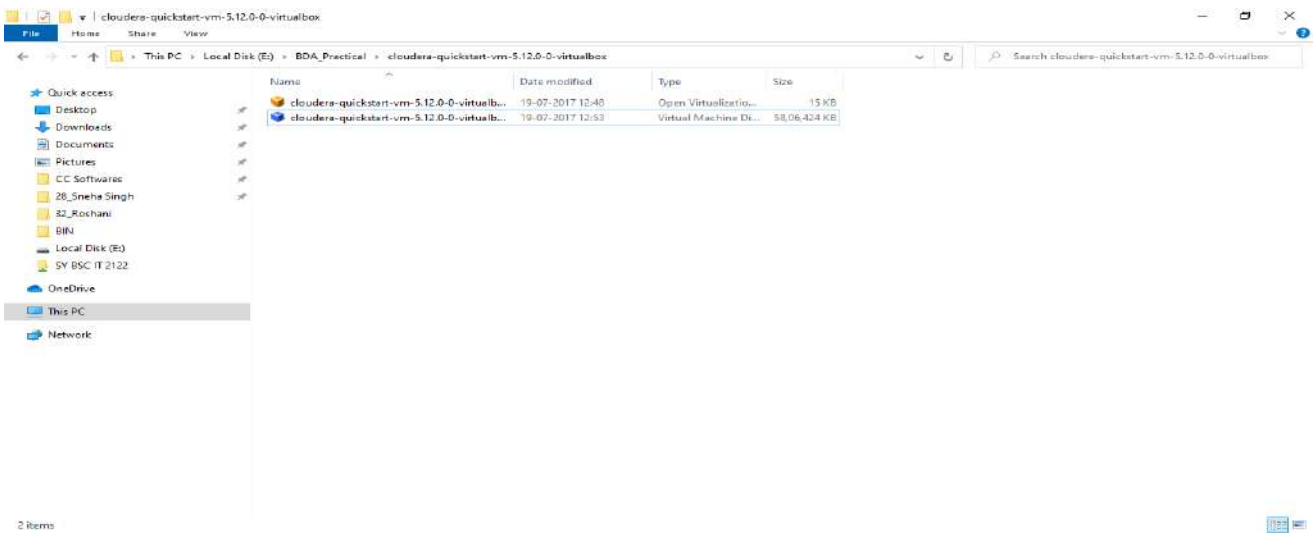
cloudera download link –

https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip



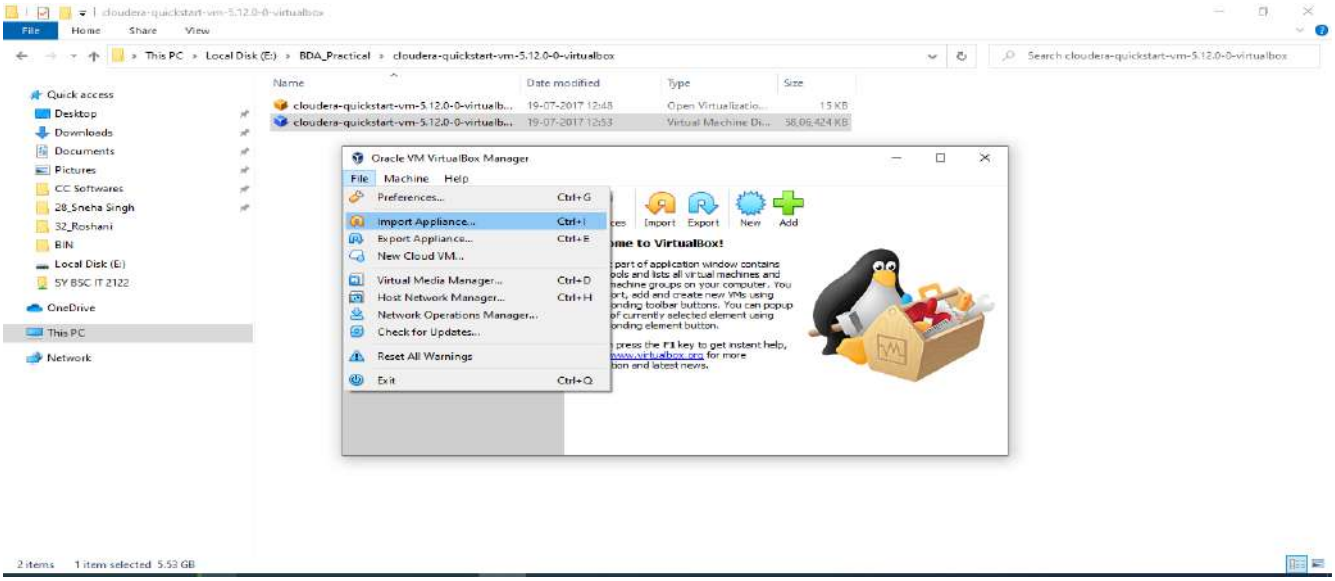
Downloading the Cloudera QuickStart VM

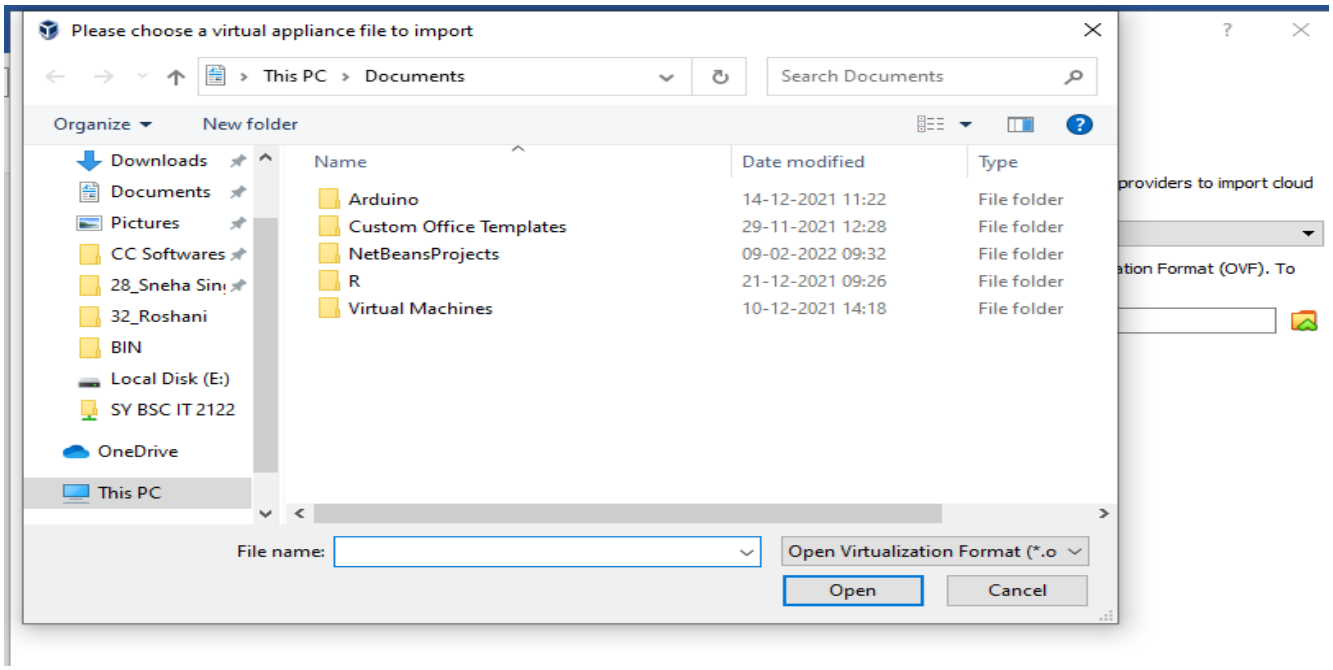
- The Cloudera QuickStart VMs are openly available as Zip archives in VirtualBox, VMware and KVM formats. To download the VM, search for https://www.cloudera.com/downloads.html and select the appropriate version of CDH that you require.
- Click on the 'GET IT NOW' button, and it will prompt you to fill in your details.
- Once the file is downloaded, go to the download folder and unzip these files. It can then be used to set up a single node Cloudera cluster.
- Shown below are the two virtual images of Cloudera QuickStart VM.
- Now that the downloading process is done with, let's move forward with this Cloudera QuickStart VM Installation guide and see the actual process.



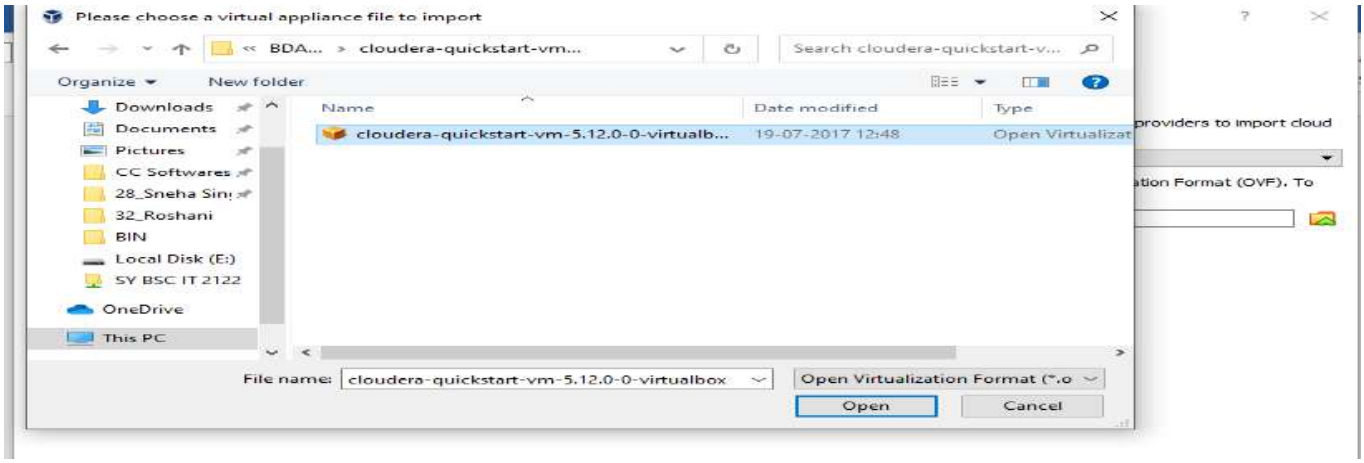
Cloudera QuickStart VM Installation

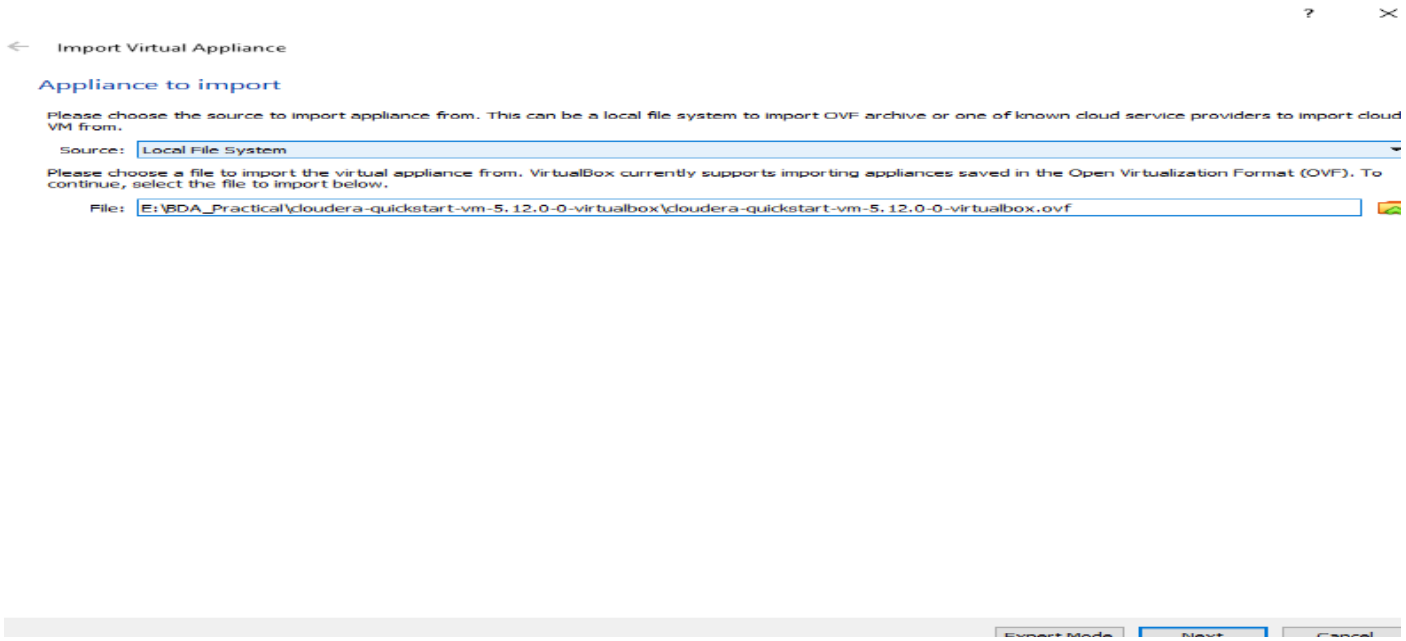
- Before setting up the Cloudera Virtual Machine, you would need to have a virtual machine such as VMware or Oracle VirtualBox on your system.
- In this case, we are using Oracle VirtualBox to set up the Cloudera QuickStart VM.
- In order to download and install the Oracle VirtualBox on your operating system, click on the following link: Oracle VirtualBox(https://www.virtualbox.org/wiki/Downloads).
- To set up the Cloudera QuickStart VM in your Oracle VirtualBox Manager, click on 'File' and then select 'Import Appliance'.
- Choose the QuickStart VM image by looking into your downloads. Click on 'Open' and then 'Next'. Now you can see the specifications, then click on 'Import'. This will start importing the virtual disk image .vmdk file into your VM box.

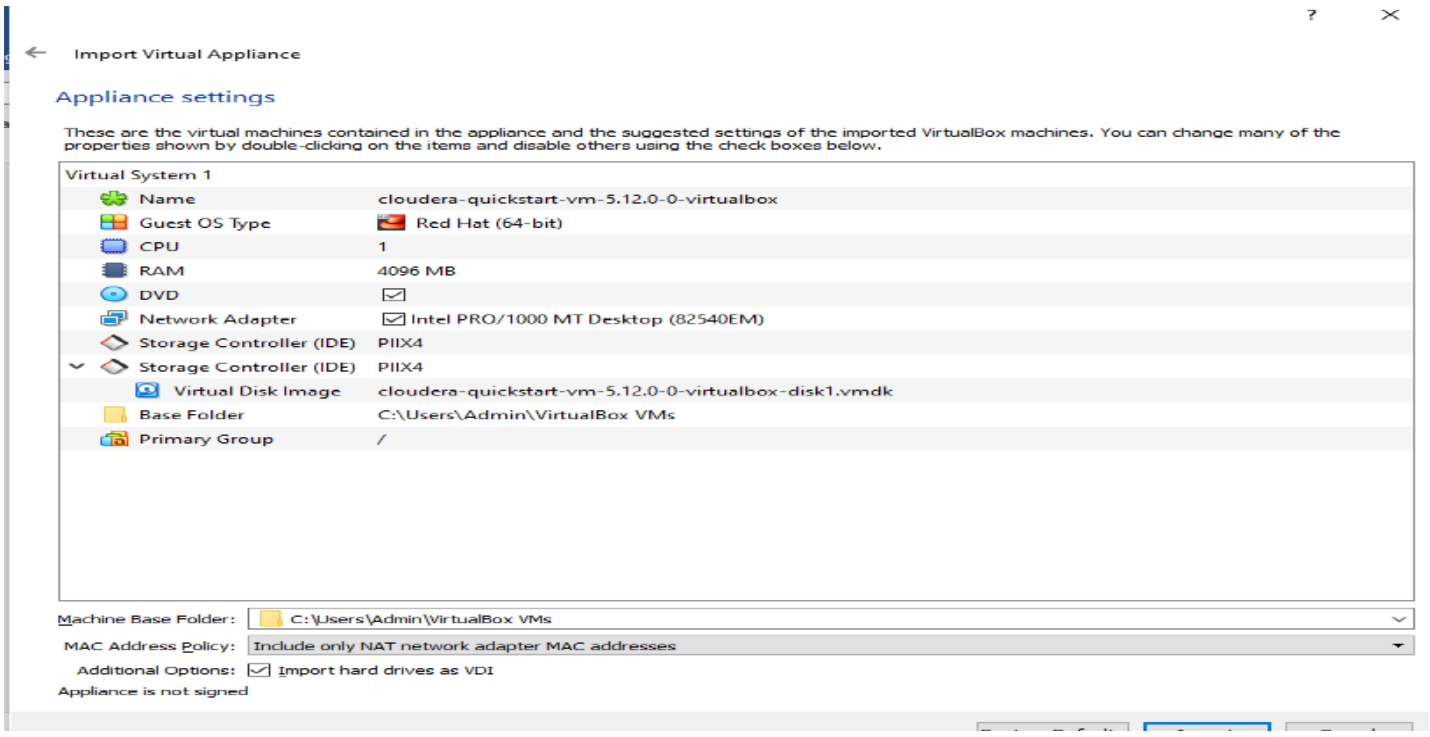


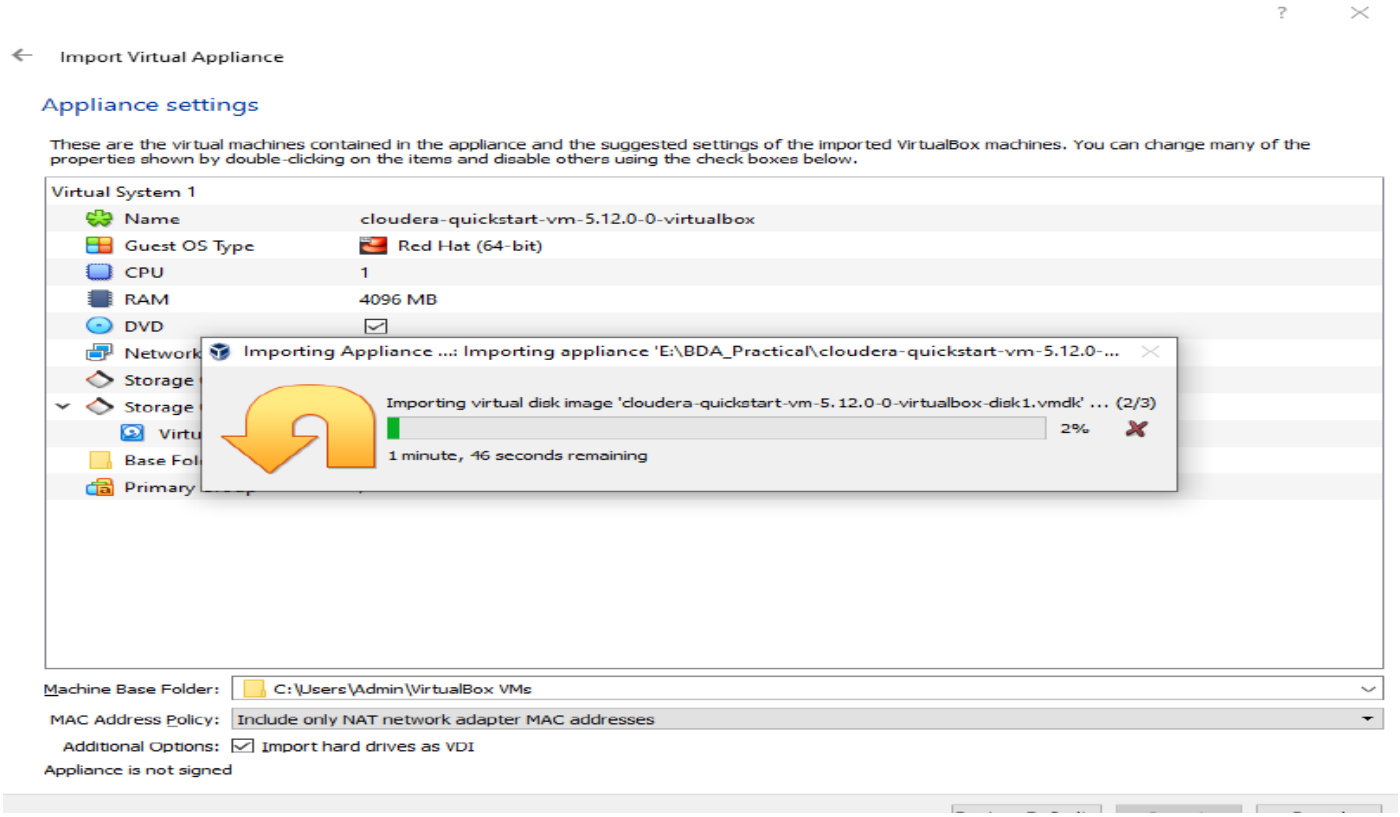


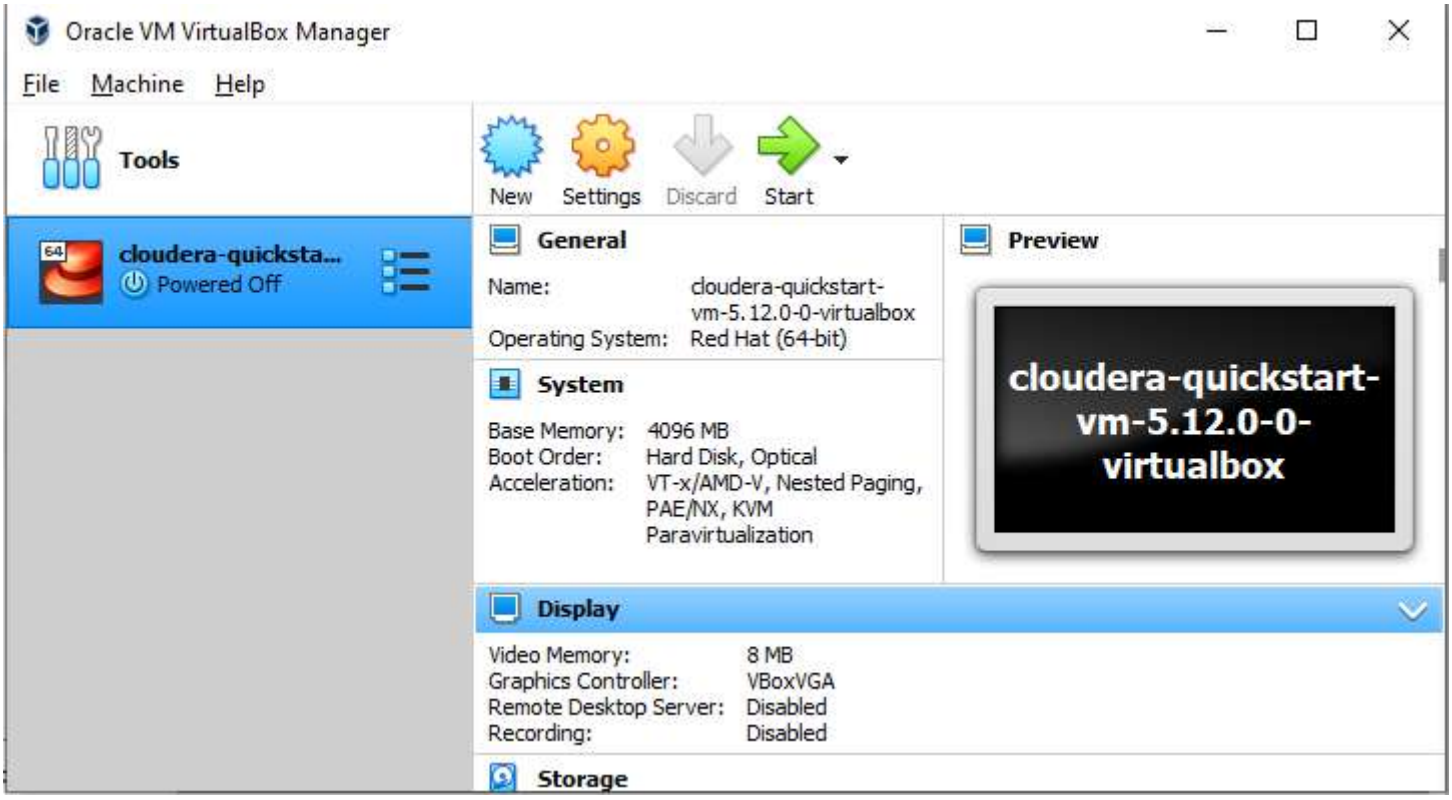
- Click on "Open" and Wait for a while, as the importing finishes.
- The next step is to go ahead and set up a Cloudera QuickStart VM for practice. Once the importing is complete, you can see the Cloudera QuickStart VM on the left side panel.


Please choose a virtual appliance file to import
BDA... > cloudera-quickstart-vm...
Search cloudera-quickstart-v...
Organize
New folder
Downloads
Documents
Pictures
CC Softwares
28_Sneha Sin
32_Roshani
BIN
Local Disk (E:)
SY BSC IT 2122
OneDrive
This PC
Name
Date modified
Type
cloudera-quickstart-vm-5.12.0-0-virtualb...
19-07-2017 12:48
Open Virtualizat
File name: cloudera-quickstart-vm-5.12.0-0-virtualbox
Open Virtualization Format (*.o
Open
Cancel

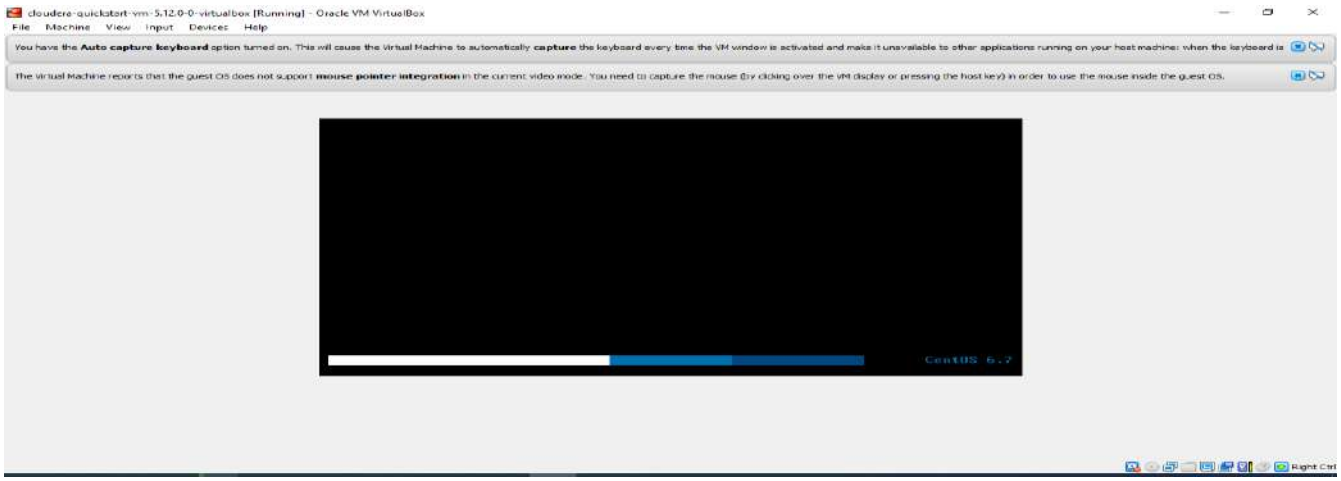

Import Virtual Appliance
Appliance to import
Please choose the source to import appliance from. This can be a local file system to import OVF archive or one of known cloud service providers to import cloud VM from.
Source: Local File System
Please choose a file to import the virtual appliance from. VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.
File: E:\BDA_Practical\cloudera-quickstart-vm-5.12.0-0-virtualbox\cloudera-quickstart-vm-5.12.0-0-virtualbox.ovf

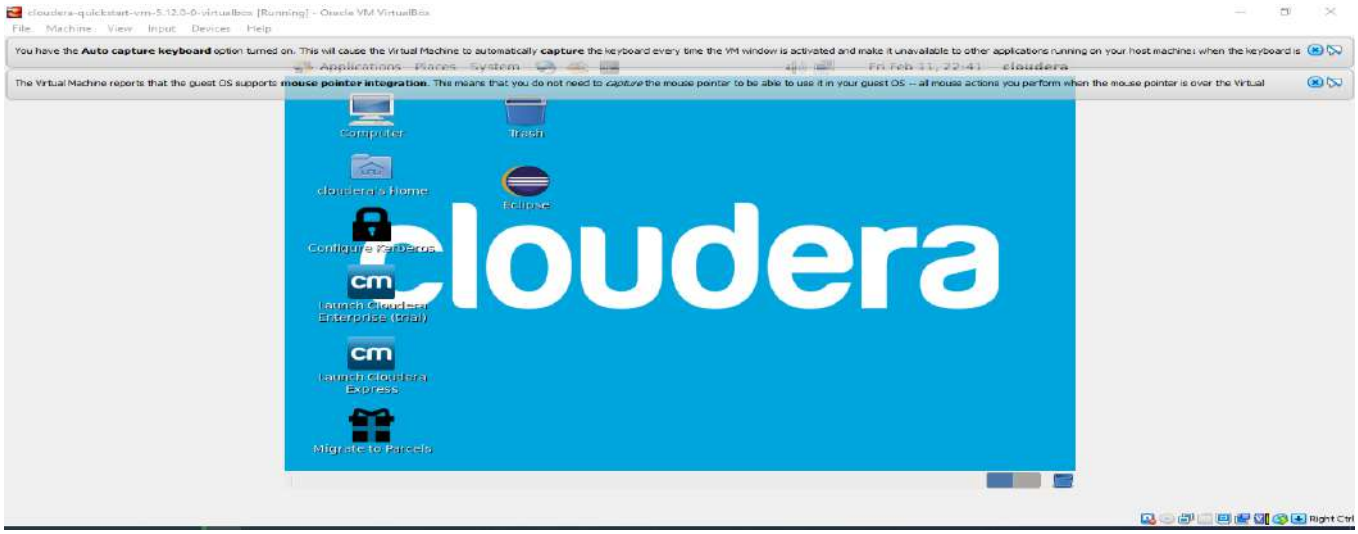

Import Virtual Appliance
Appliance settings
These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.
Virtual System 1
Name cloudera-quickstart-vm-5.12.0-0-virtualbox
Guest OS Type Red Hat (64-bit)
CPU 1
RAM 4096 MB
DVD
Network Adapter Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE) PIIX4
Storage Controller (IDE) PIIX4
Virtual Disk Image cloudera-quickstart-vm-5.12.0-0-virtualbox-disk1.vmdk
Base Folder C:\Users\Admin\VirtualBox VMs
Primary Group /
Machine Base Folder: C:\Users\Admin\VirtualBox VMs
MAC Address Policy: Include only NAT network adapter MAC addresses
Additional Options: Import hard drives as VDI
Appliance is not signed

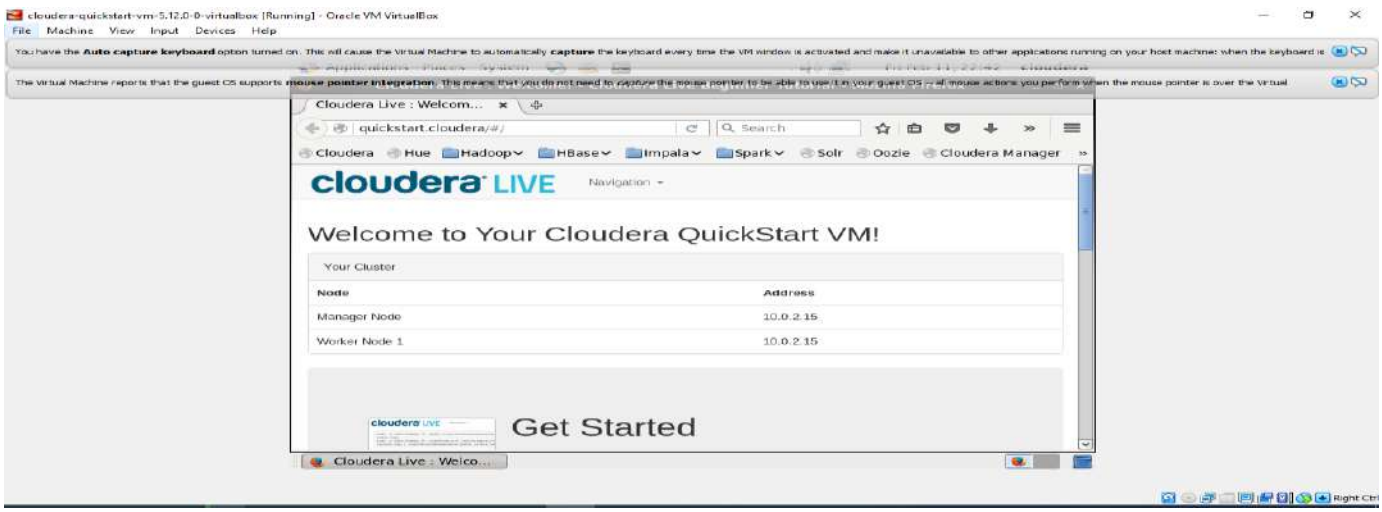




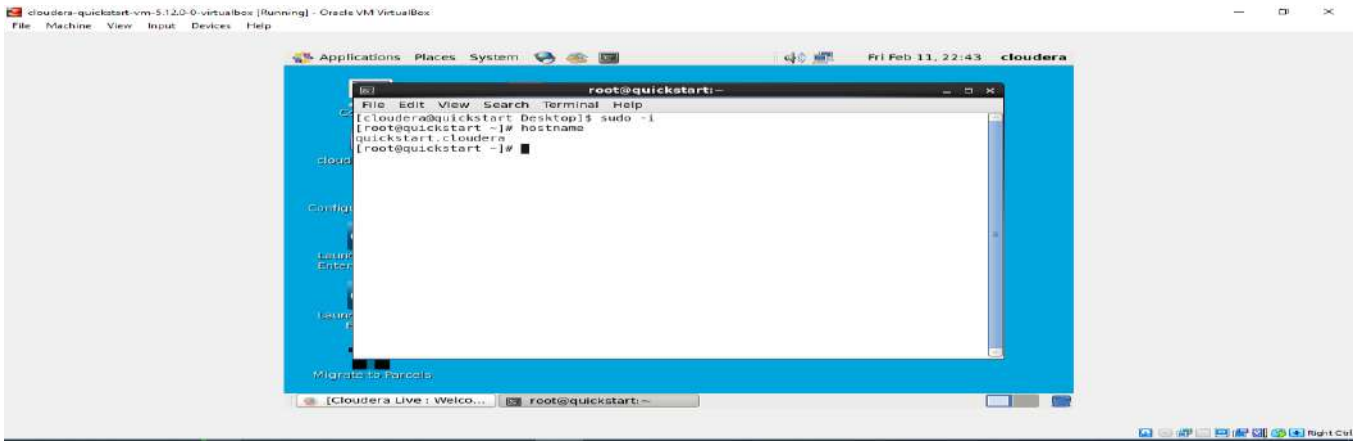
- The next step will be going ahead and starting the machine by clicking the ‘Start’ symbol on top.
- Once your machine comes on, it will look like this:

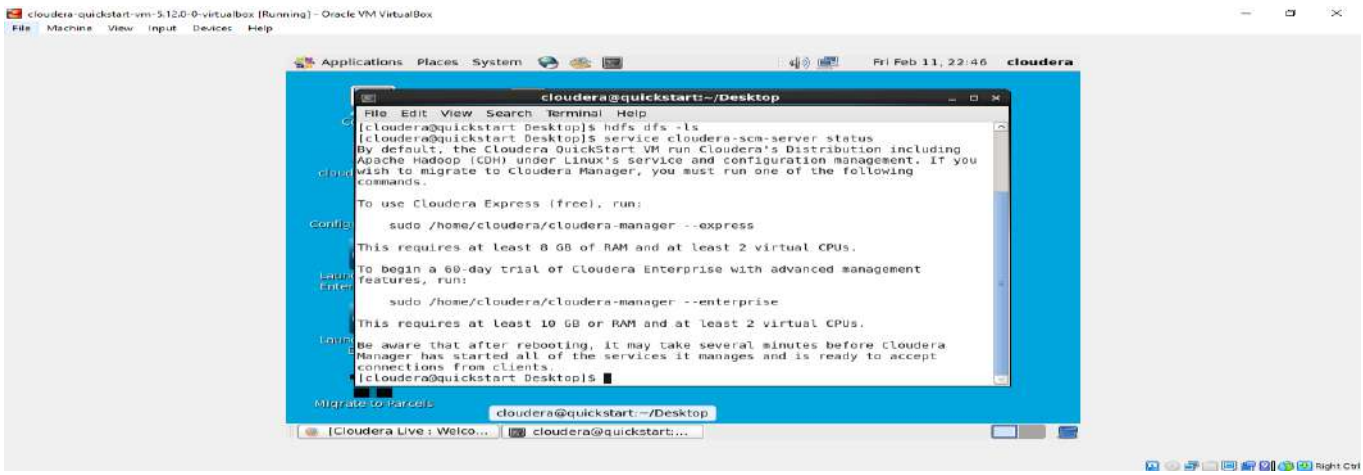




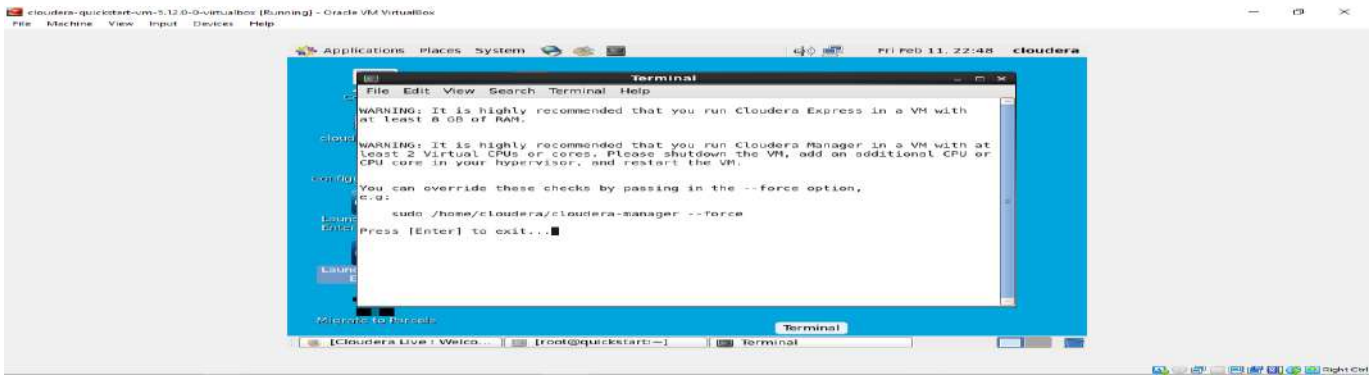


- Next, we have to follow a few steps to gain admin console access. You need to click on the terminal present on top of the desktop screen, and type in the following:
    1. **hostname** # This shows the hostname which will be quickstart.cloudera
    2. **hdfs dfs -ls /** # Checks if you have access and if your cluster is working. It displays what exists on your HDFS location by default
    3. **service cloudera-scm-server status** # Tells what command you have to type to use cloudera express free
    4. **su** - #Login as root
    5. service cloudera-scm-server status # The password for root is cloudera
- 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’.

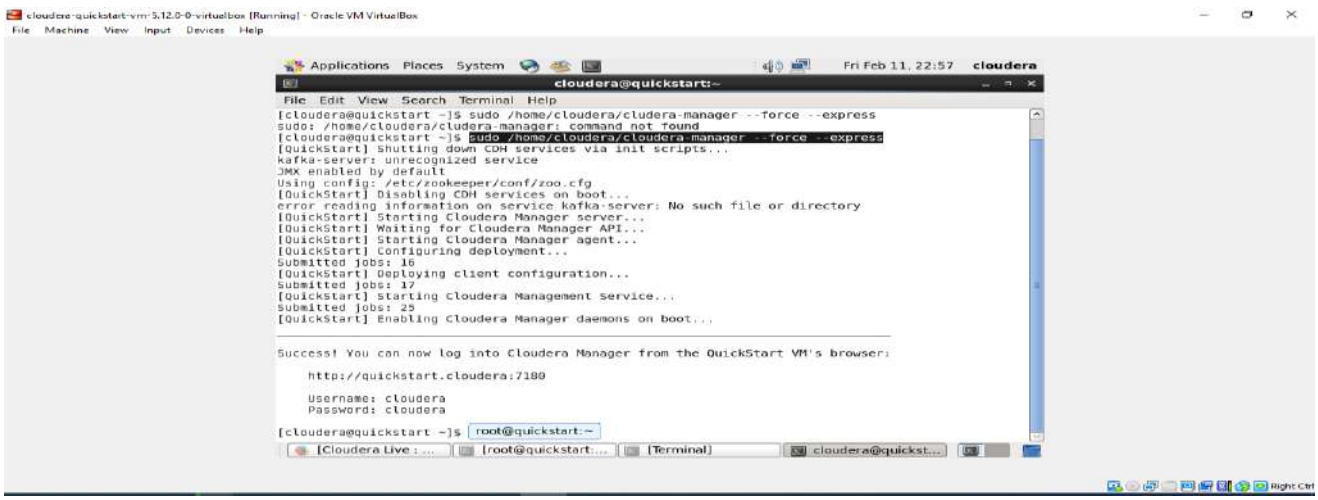




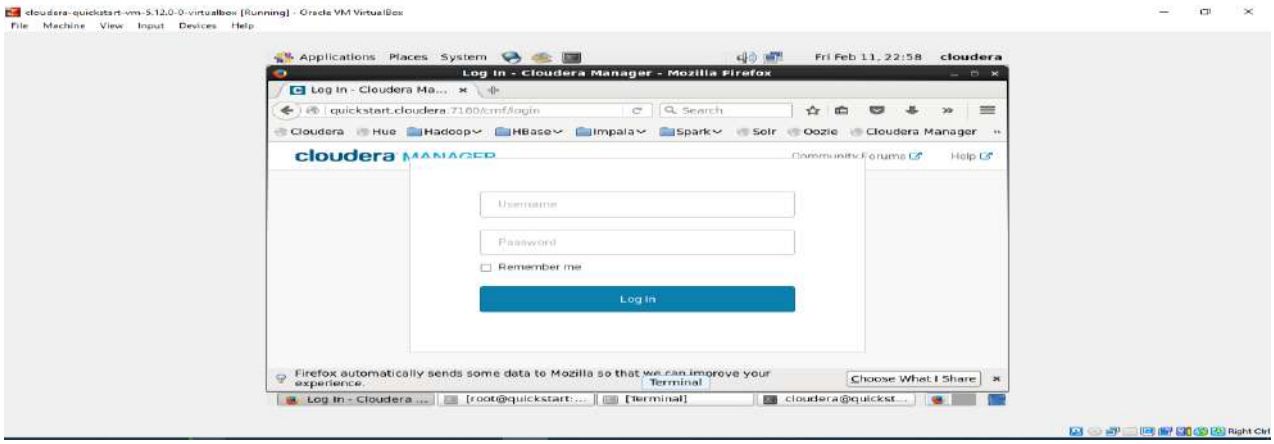
- You are required to copy the command, and run it on a separate terminal. Hence, open a new terminal, and use the below command to close the Cloudera based services. It will restart the services, after which you can access your admin console.



sudo /home/cloudera/cloudera-manager –force --express



- 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.



- 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.

Aim: To implement Various Hadoop HDFS Commands

What is Hadoop?

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in Hadoop, data resides in a distributed file system which is called as a Hadoop Distributed File system. The processing model is based on 'Data Locality' concept wherein computational logic is sent to cluster nodes (server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

Hadoop Distributed File System (HDFS) - Data Storage and Management

This is the most important component of the Hadoop ecosystem. HDFS is Hadoop's primary storage system. Hadoop Distributed File System (HDFS) is a Java-based file system that provides reliable, fault tolerance and accessible data storage for the big data. HDFS is a distributed file system that runs on conventional hardware. HDFS is already configured with the default settings for many installations. Typically, a large cluster configuration is required. Hadoop interacts directly with HDFS using commands. Wen comes to HDFS, there are also two components can be identified, which are known as Name Node and Data Node.

Name Node

It is also known as Master node. Here, it does not store actual data or datasets. Name Node stores the Meta data, for an example, the number of calls transform from a tower, their position, where the end users are getting the call, the Data node data and other details. Basically, this contains files and directories. The tasks of Name node can be recognized as follows.

- Managing file system namespace
- Controlling the access of clients to files
- Executing file system through naming, opening, closing files and directories

Data Node

Data node is called as Slave. Data node is responsible for the effective storage of data in HDFS. The data node completes read and write operations on customer request. They also send signals, known as heartbeats, to the name node. These heartbeats show the status of the data node. Replica Block of Data node consists of two files in the file system. The first file is for data and the second for registry metadata. HDFS metadata contains a data control. At startup, each Data node is connected to the appropriate Name node and grasp. The ID of the Data Node namespace and the software version are controlled by the handshake. If a discrepancy is detected, Data Node is automatically disabled. When comes to tasks of Data node, those can be detailed as follows.

- This is consisting of operations like block replica creation, deletion, and replication according to the instruction of Name node
- Managing data storage of the system

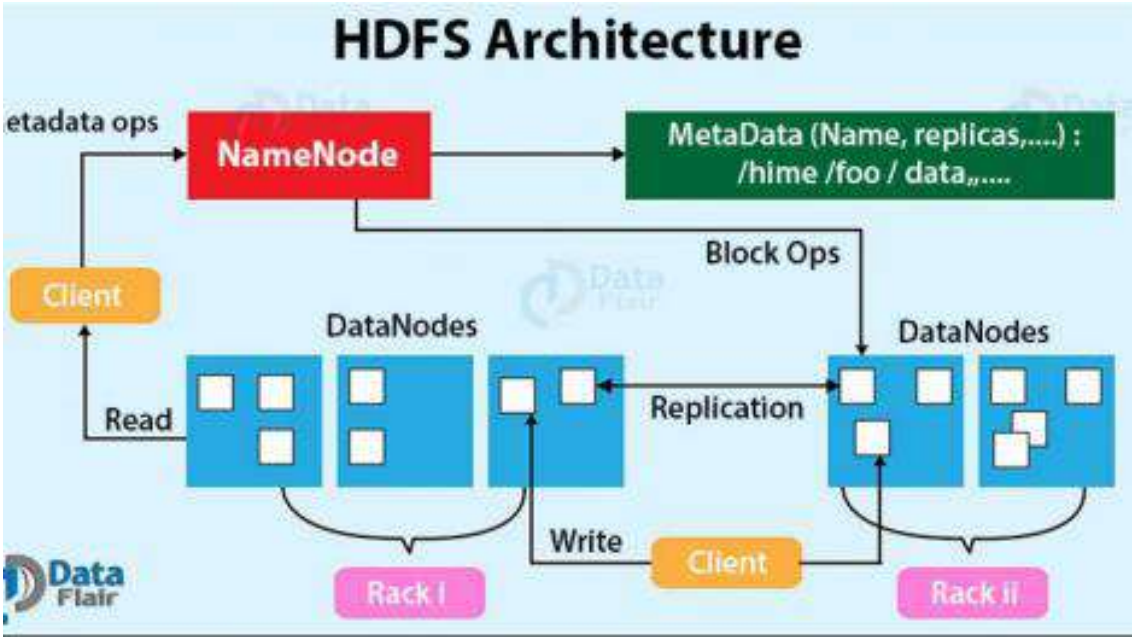
Processing and Computation – Hadoop MapReduce

When comes to Hadoop MapReduce, that is the main component of the Hadoop, that provides data processing. MapReduce is can be identified as an easy-to-write application framework that processes the large amount of structured and unstructured data stored in the Hadoop distributed file system.

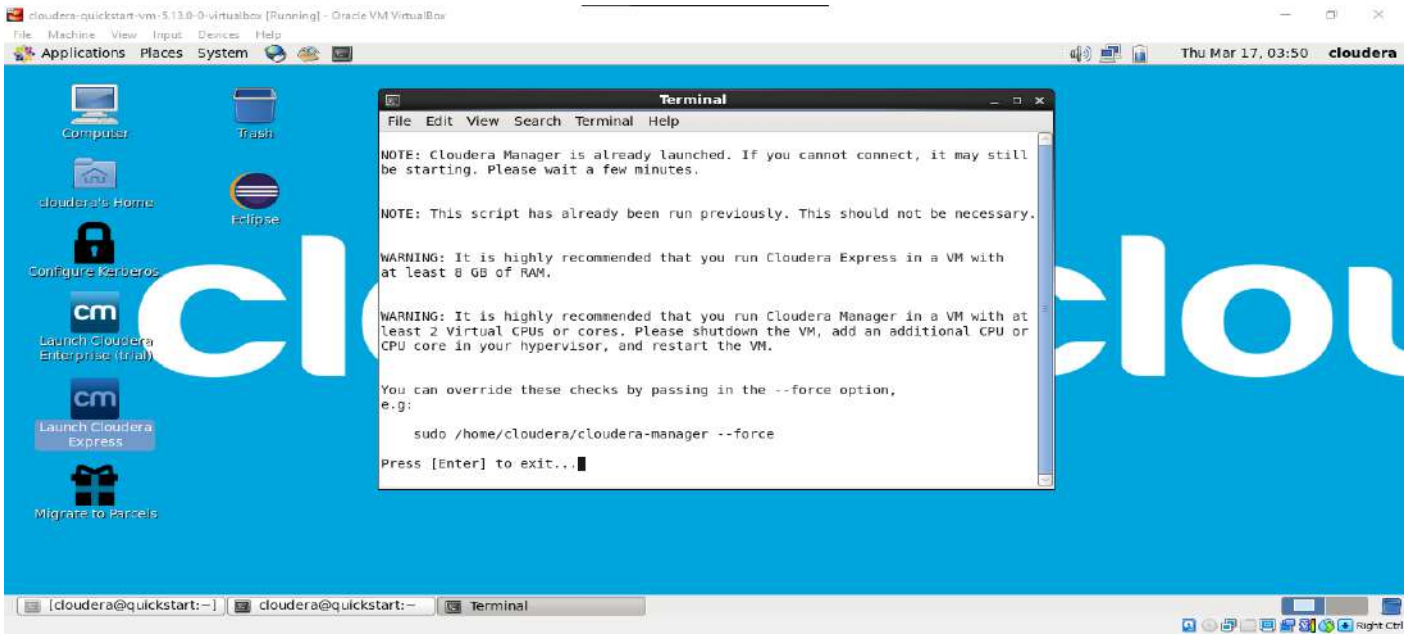
MapReduce programs are parallel, so they are very useful for large-scale data analysis using multiple clusters. Therefore, this parallelism increases the speed and reliability of the cluster. In MapReduce, there are two functions, Map function and Reduce function.

Two functions can be identified, map function and reduce function.

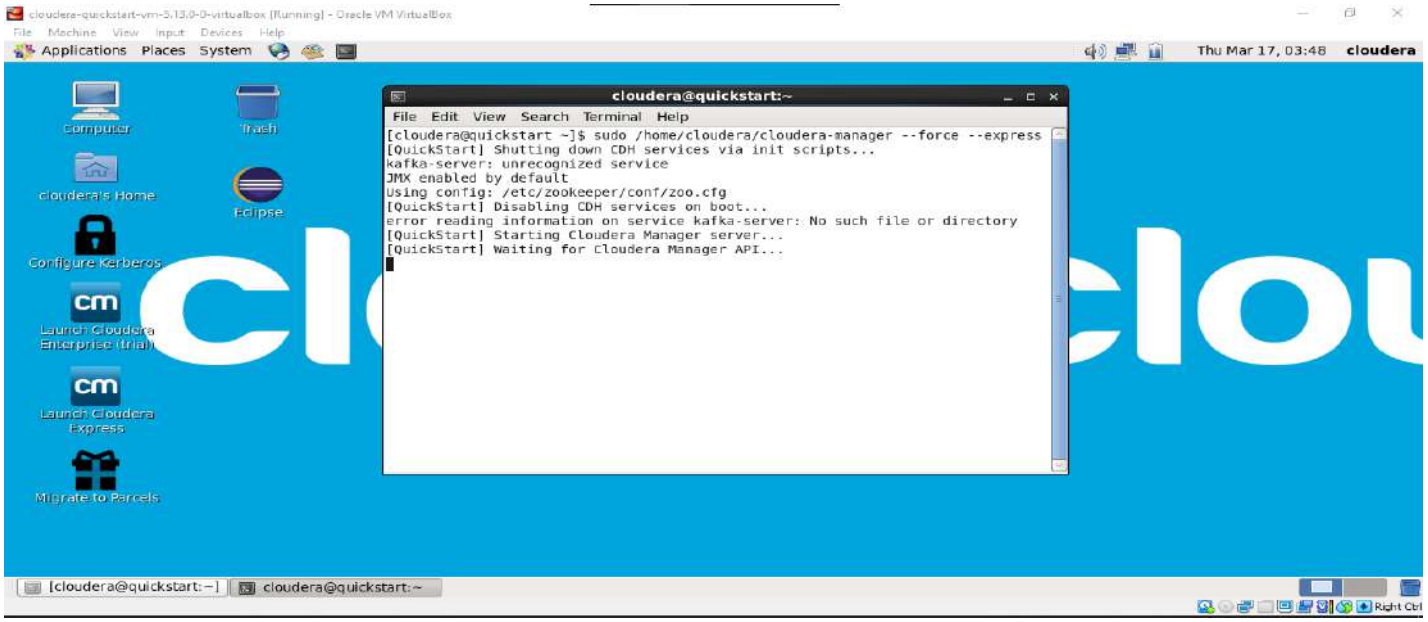
- The map function retrieves a data set and converts it to another data set. Each element is divided into processing (key / value pairs).
- The Reduce function accepts the Map output as an input and integrates these data nodes based on the key and changes the key value accordingly.

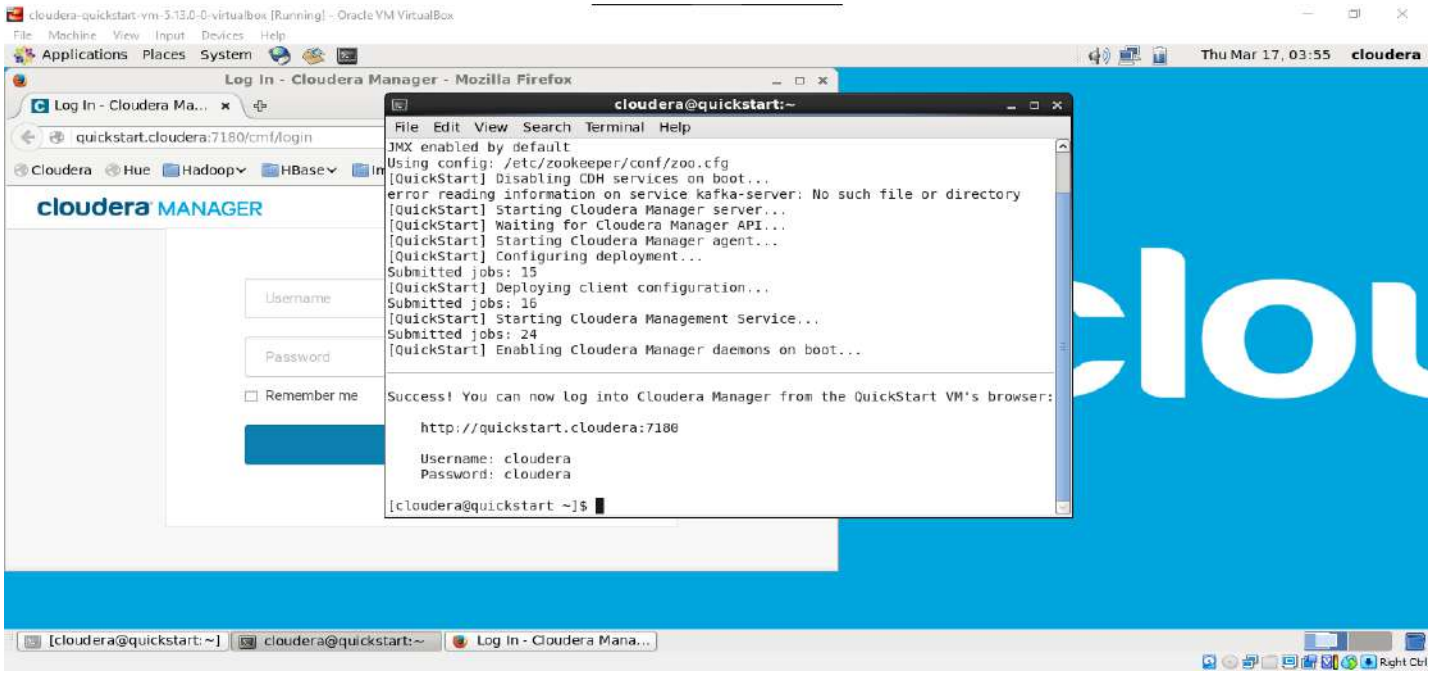


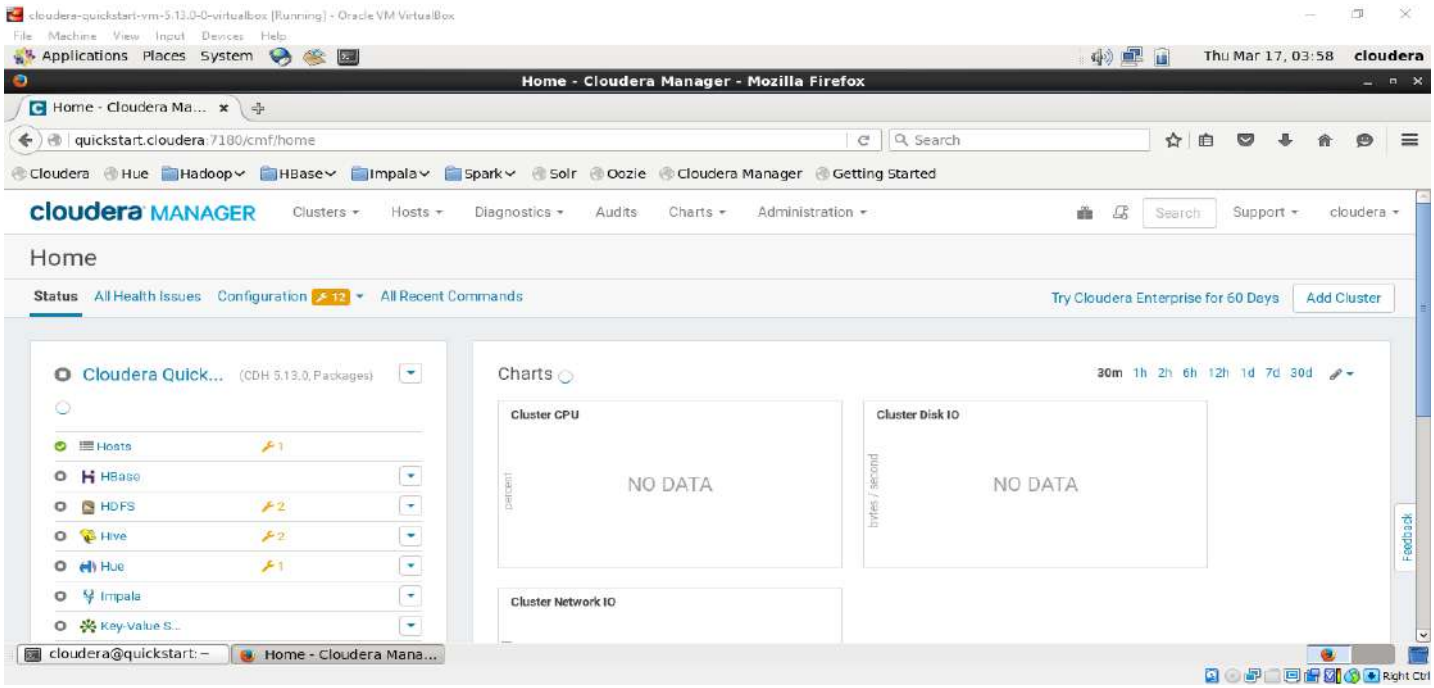
Copy the command from the Launch Cloudera Express



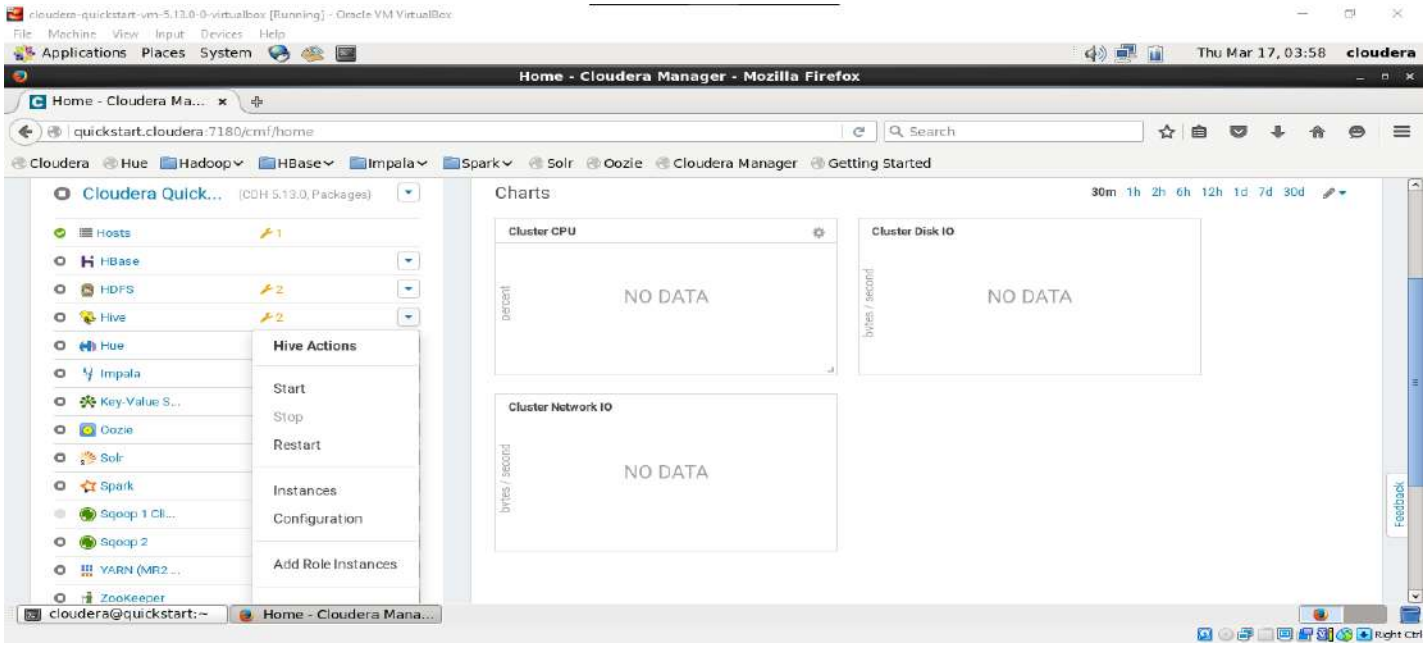
Type **sudo /home/cloudera/cloudera-manager --force --express** in terminal

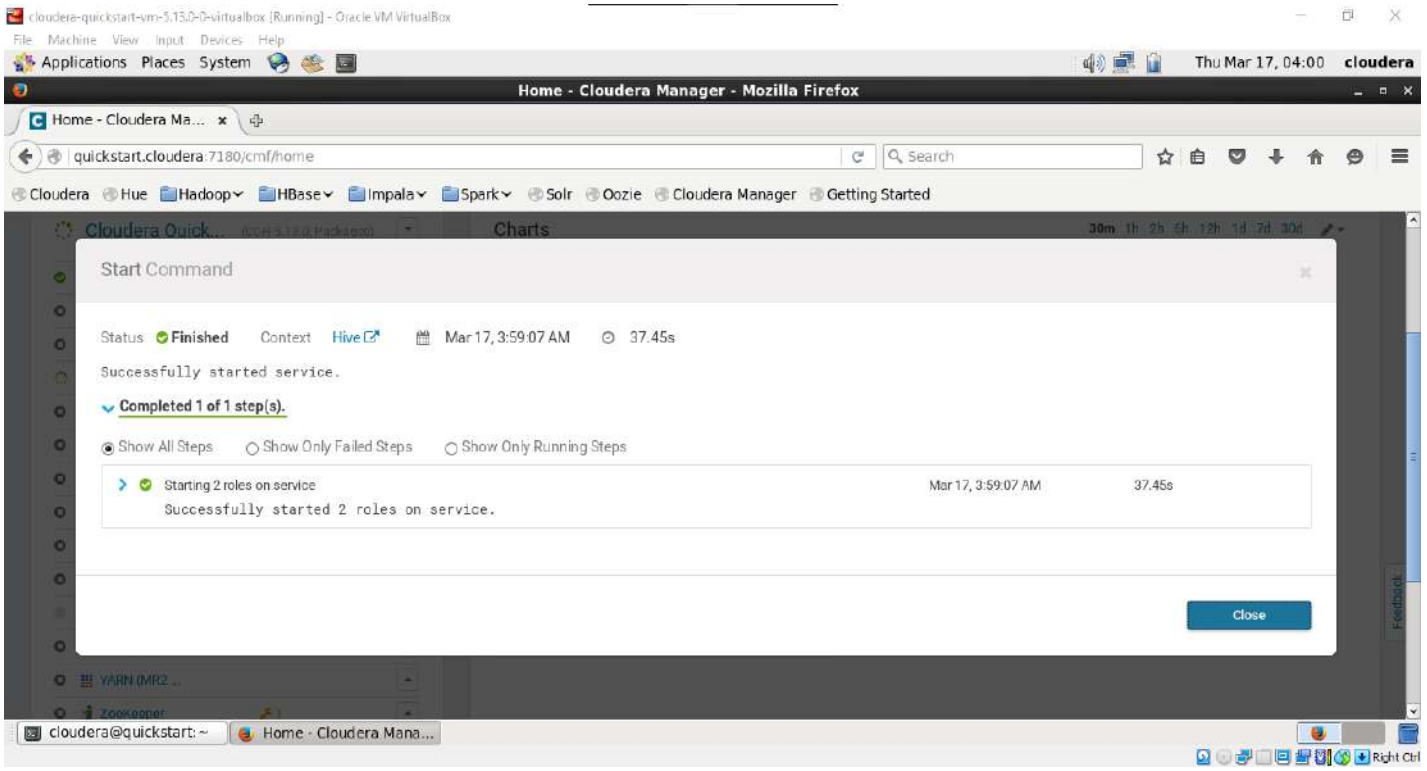






Click on down arrow beside **HDFS** and select **start**

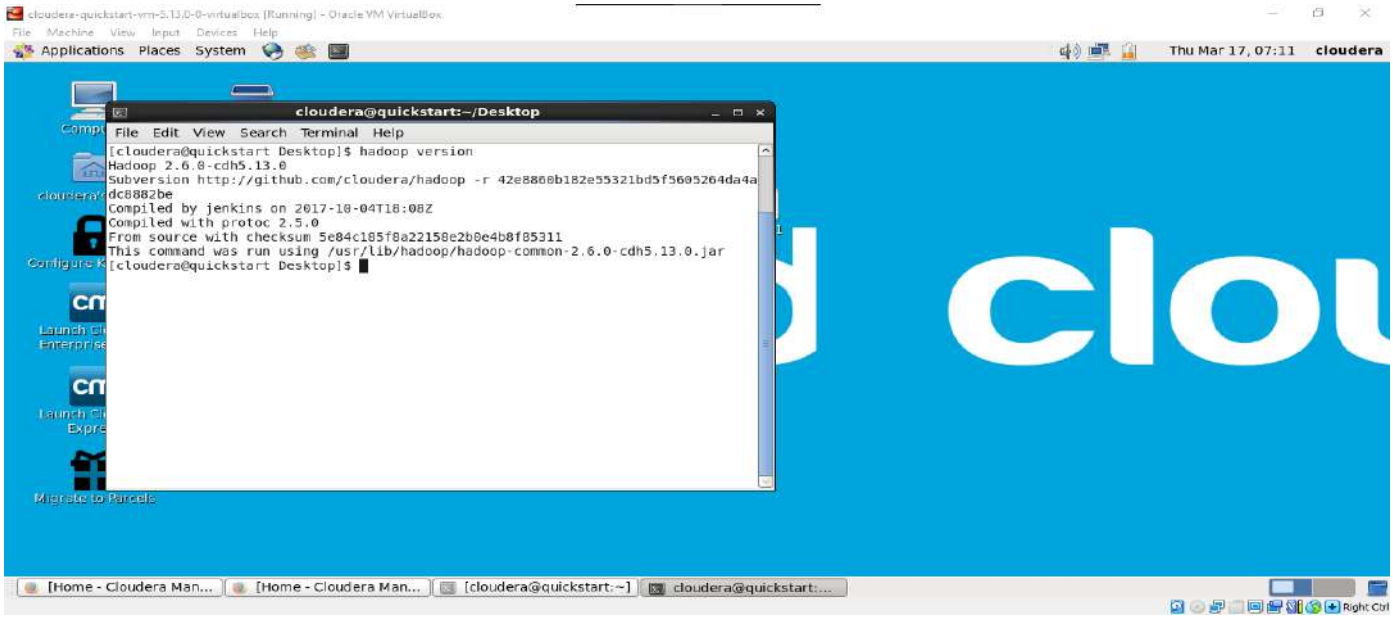




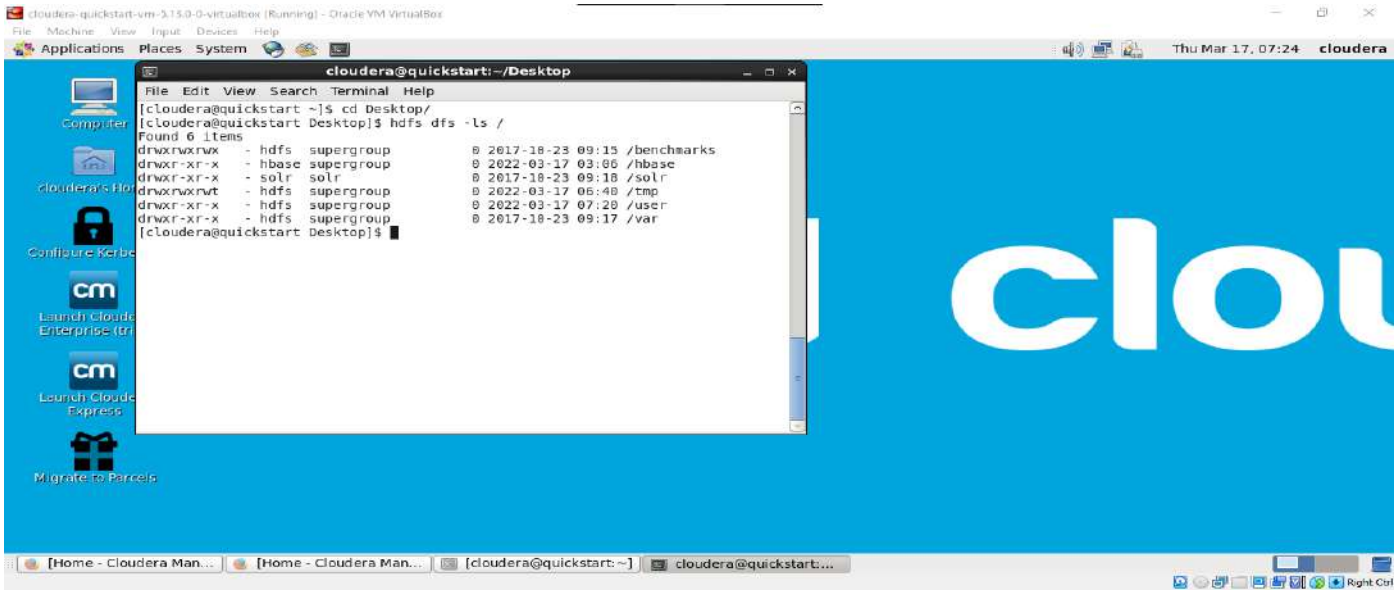
Hadoop HDFS Commands

First go to the **cd Desktop/**

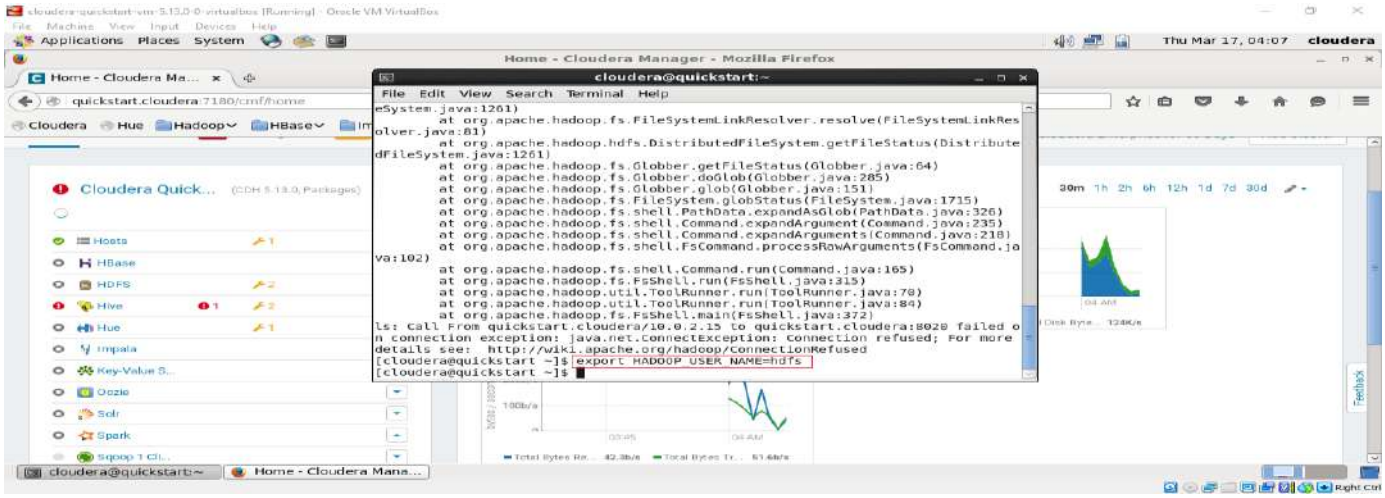
1. **hadoop version** is used to check version of Hadoop system



2. **hdfs dfs -ls /** lists all the files and folders present in HDFS location. It lists the contents of the directory specified by path, names, permissions, owner, size and modification date for each entry. hdfs dfs is the command specific to HDFS.

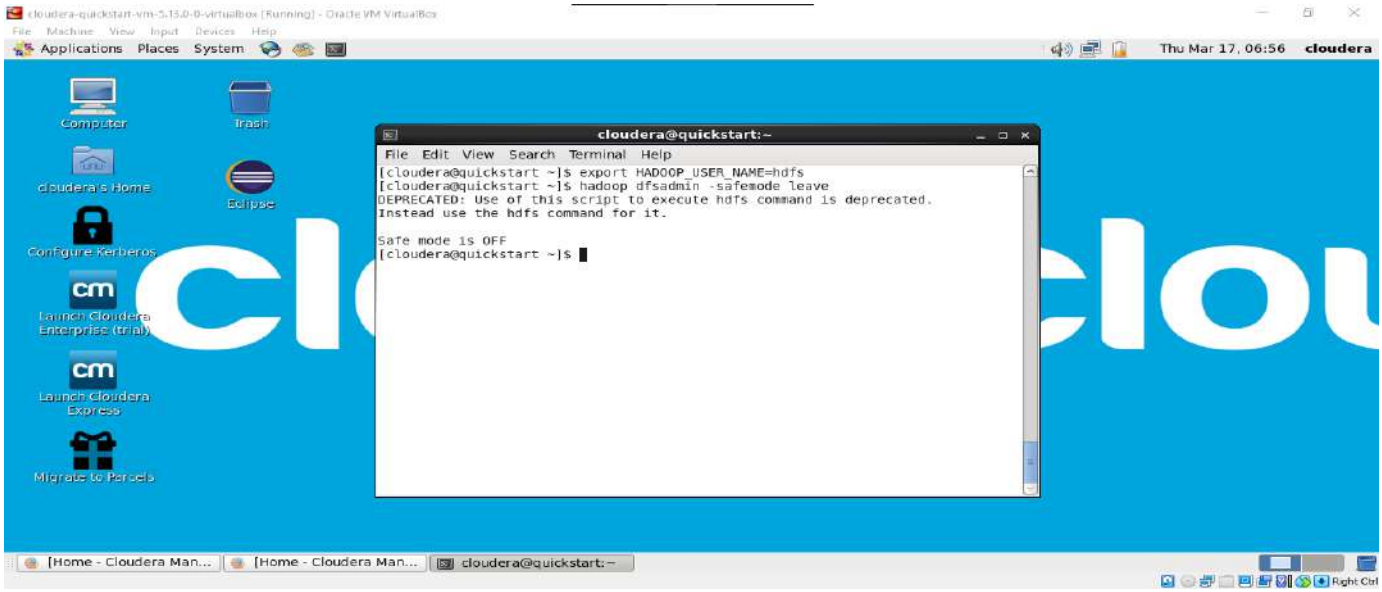


3. If getting any error on permission access use **export HADOOP_USER_NAME=hdfs** after this command run previous command again and use **hdfs dfs -ls /** to check if new directory "rjclocal" is created.

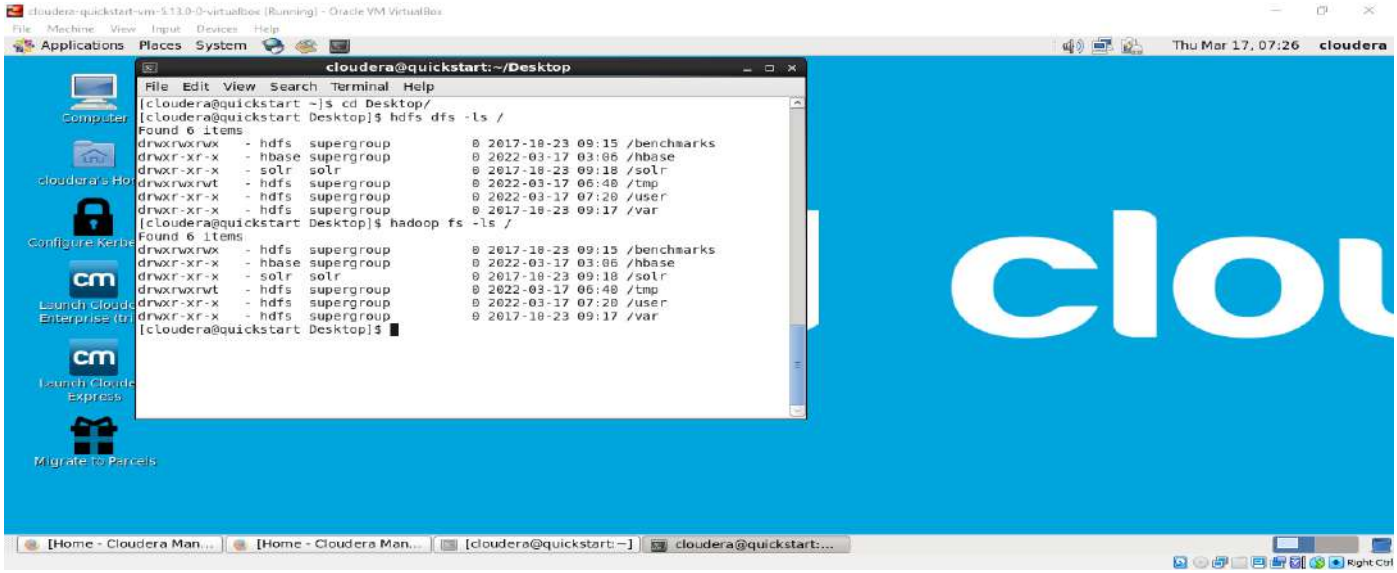


Restart then execute the **hdfs dfs –ls /**

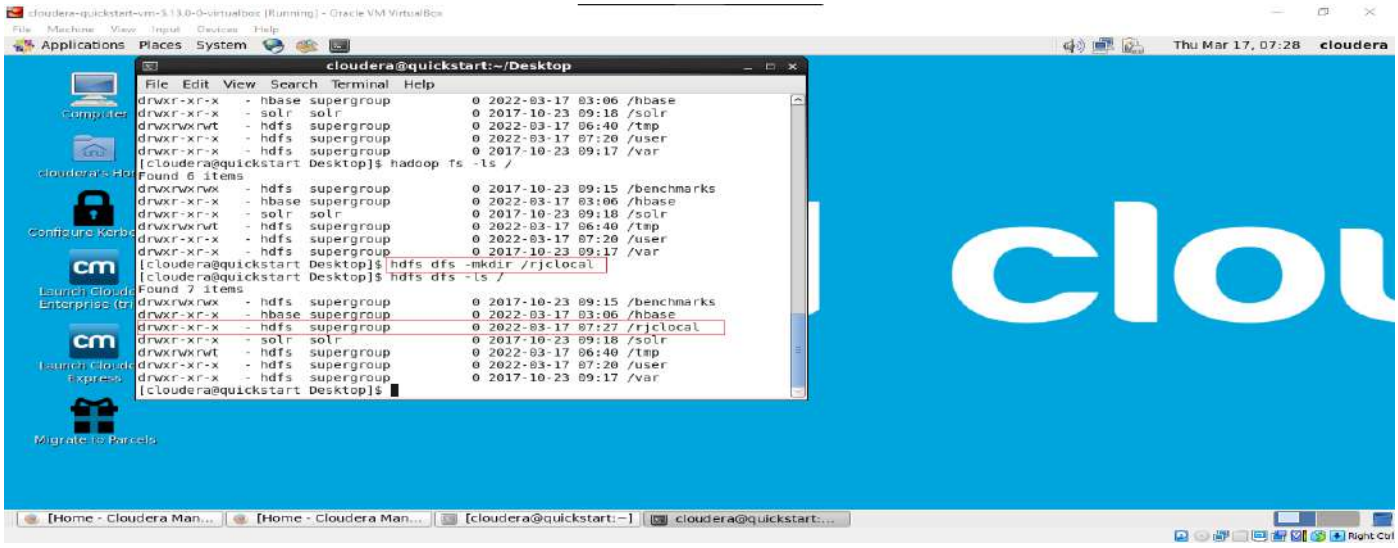
4. If getting error on safe mode then run **hadoop dfadmin -safemode leave**



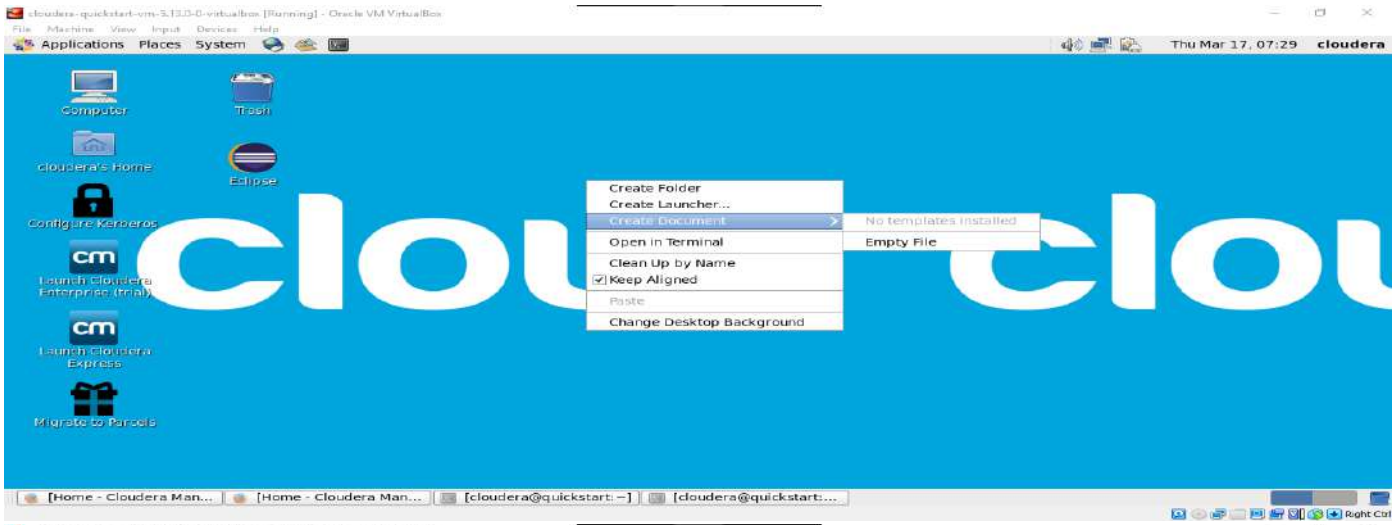
5. **hadoop fs -ls /** displays a list of content of a directory specified in the path provided by the user.

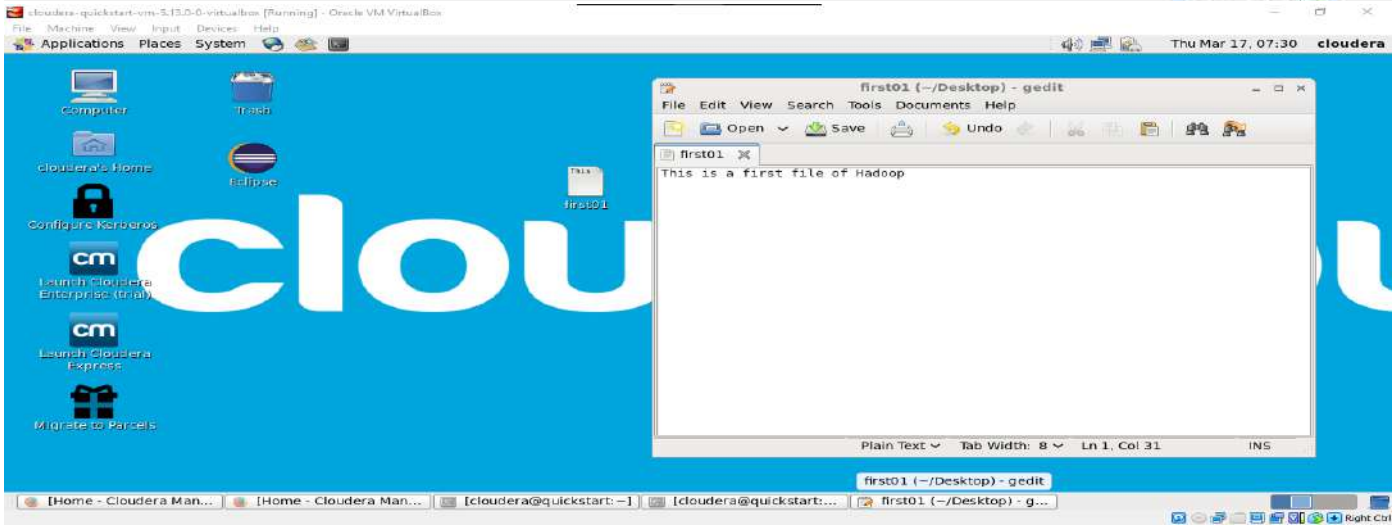


6. **hdfs dfs -mkdir /rjclocal**

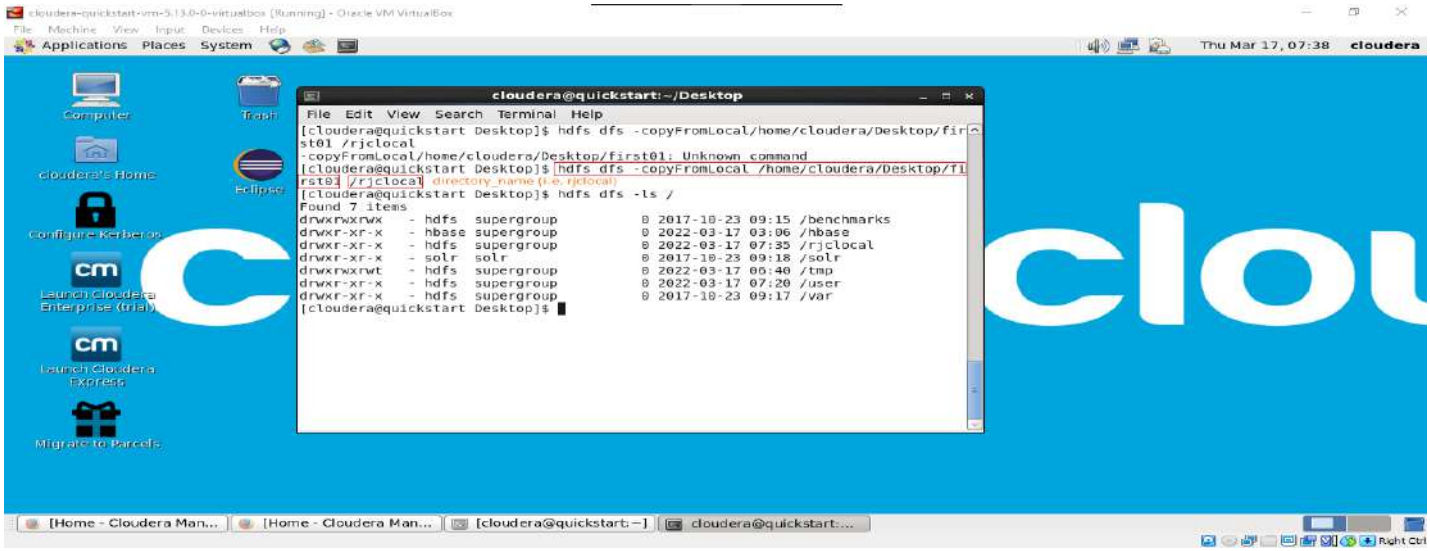


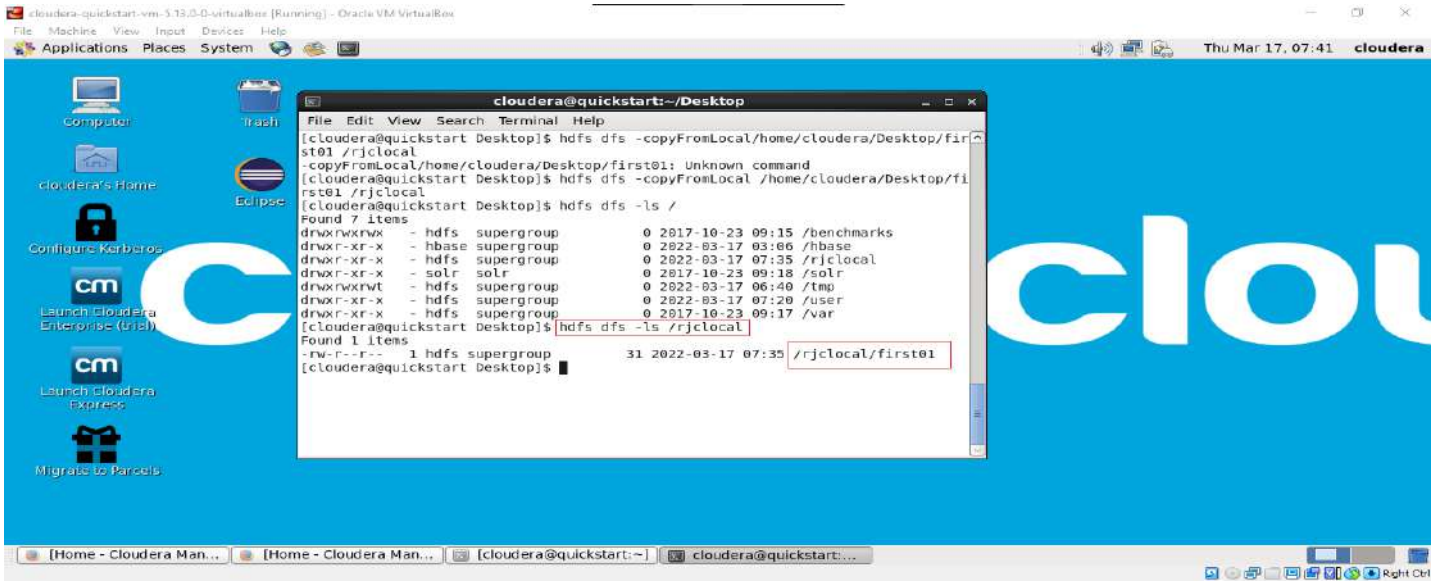
7. **Create a new file** on desktop



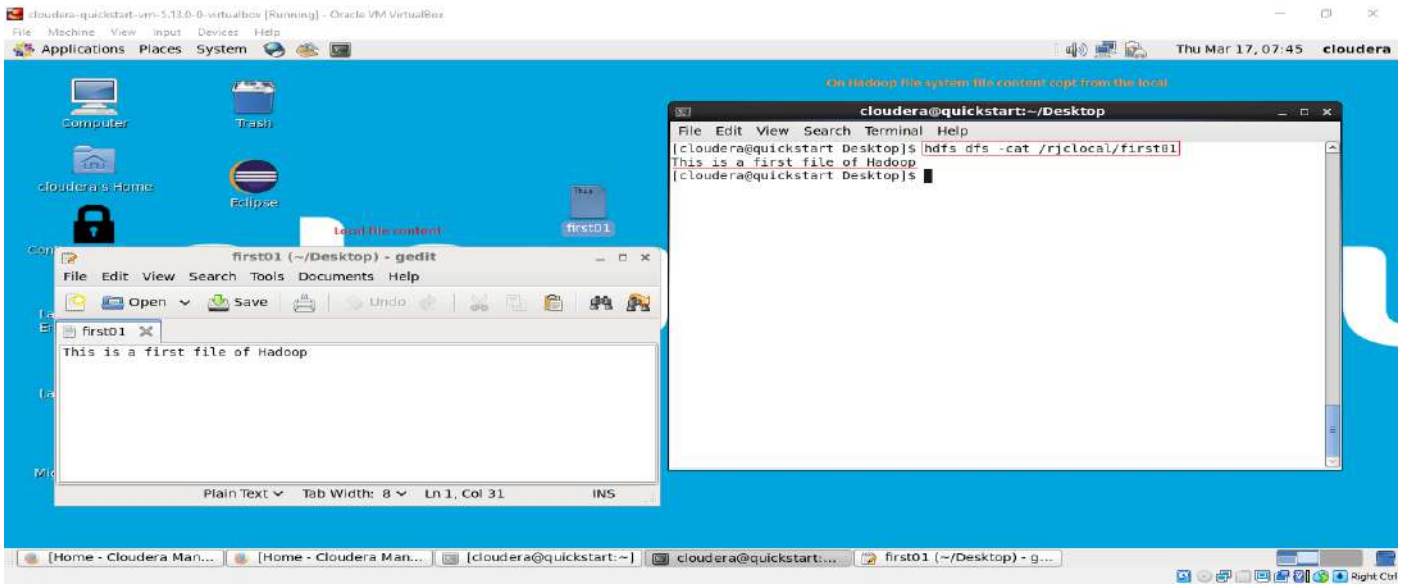


8. Use **hdfs dfs -copyFromLocal /home/cloudera/Desktop/<file_name> /<directory_name>**- this command copies **<file_name>** file to HDFS **<directory_name>** directory and use **hdfs dfs -ls /rjclocal** to check is file is successfully copied



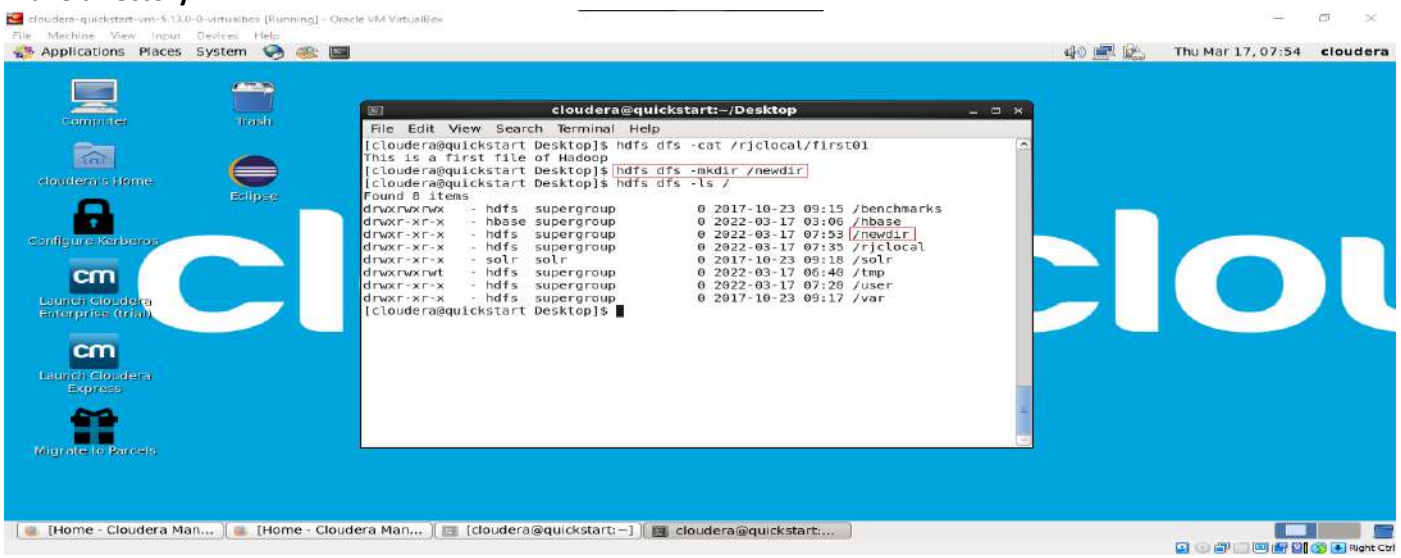


9. **hdfs dfs -cat /<directory_name>/<file_name>** reads content of file

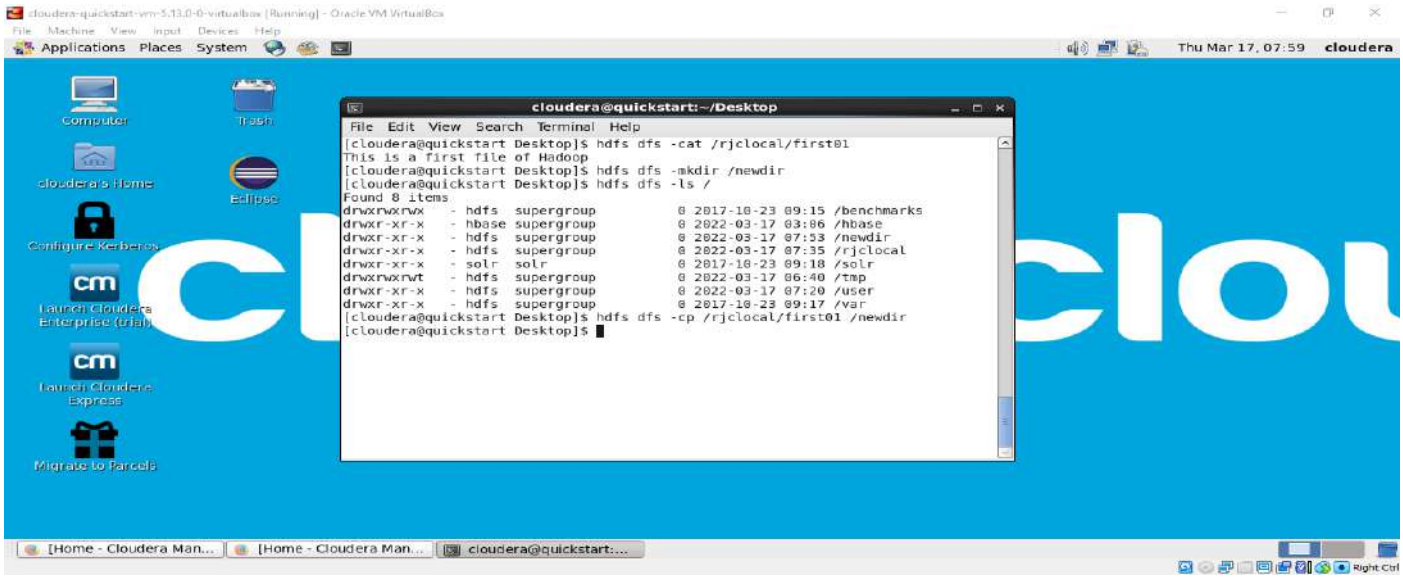


10. First we create a new directory in HDFS "**newdir**" **hdfs dfs -mkdir /<dir_name>** and then use **hdfs dfs -cp /<dir1_name>/<dir_file_name> /<dir2_name>** then check if file is copied successfully by using **hdfs dfs -ls /<dir2_name>** **cp** command is used to copy file from one directory to another within HDFS

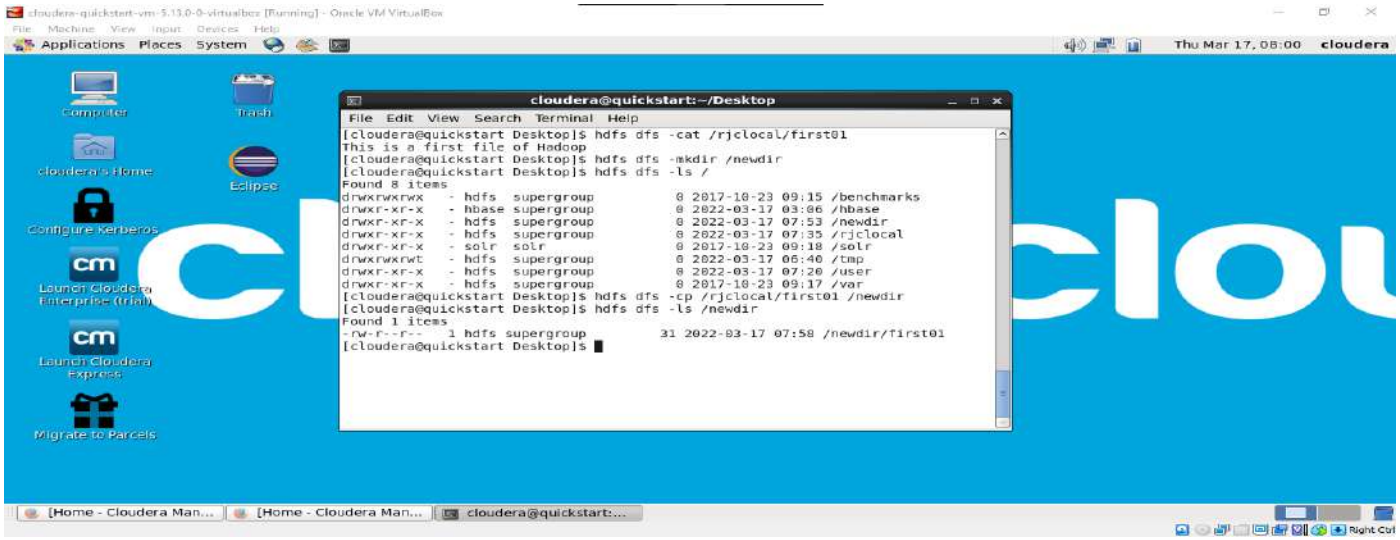
Make directory



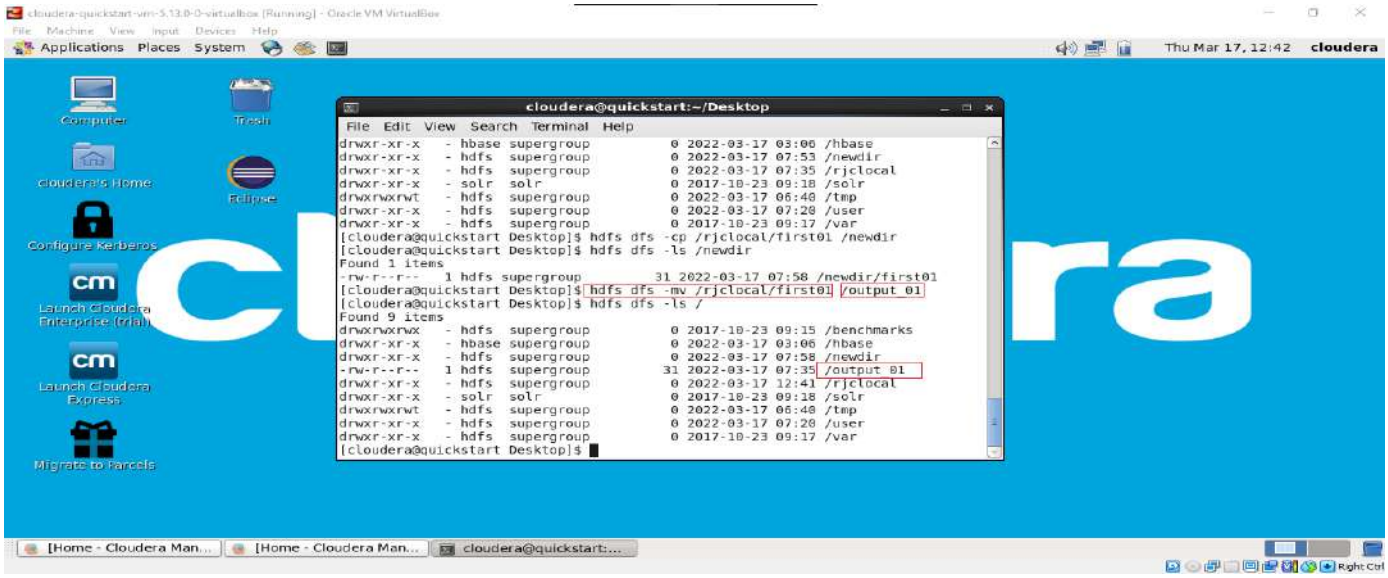
Copy the file from one directory to other directory



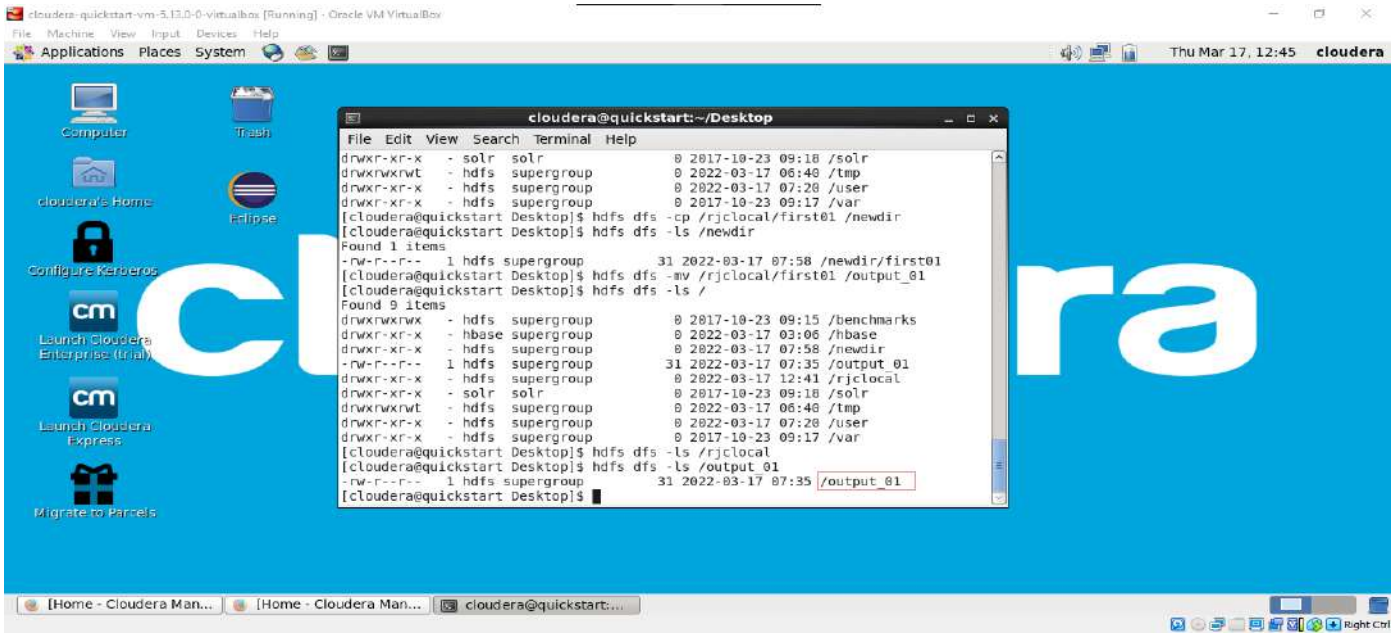
Check the newdir file is copy or not using "hdfs dfs –ls /<dir_name>"



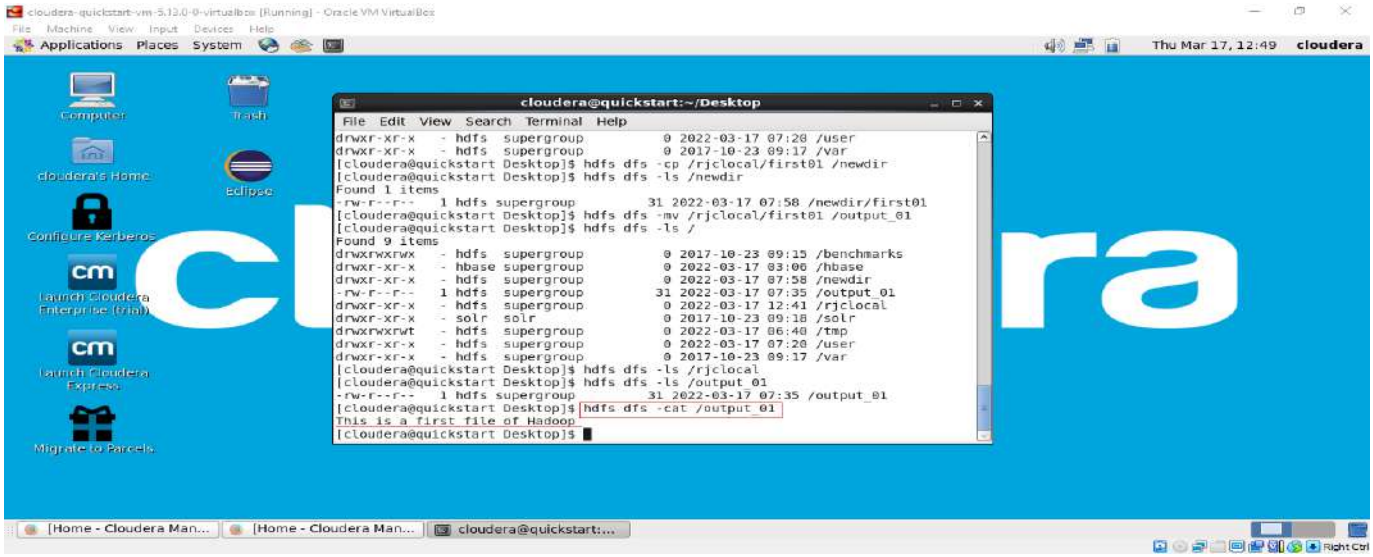
11. **hdfs dfs -mv /<dir1_name>/<file1_name> /output_abc** moves file **<file1_name>** from source to destination within HDFS



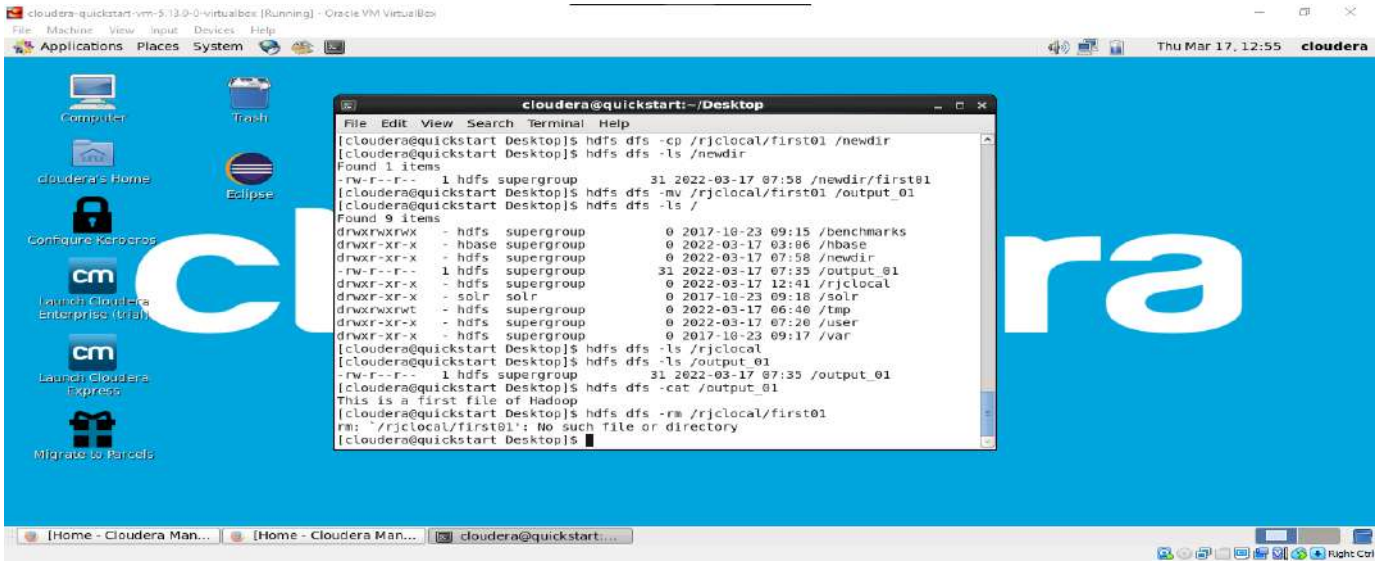
After moving file source folder to destination folder



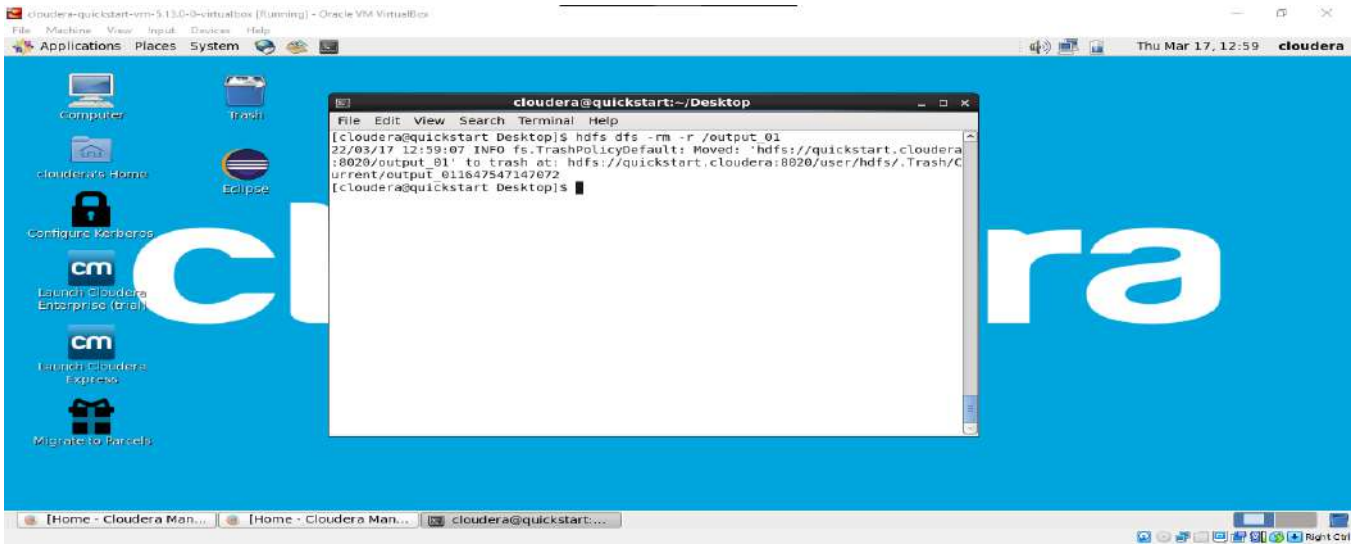
Use **hdfs dfs -cat /<file1_name>** command to read moves file



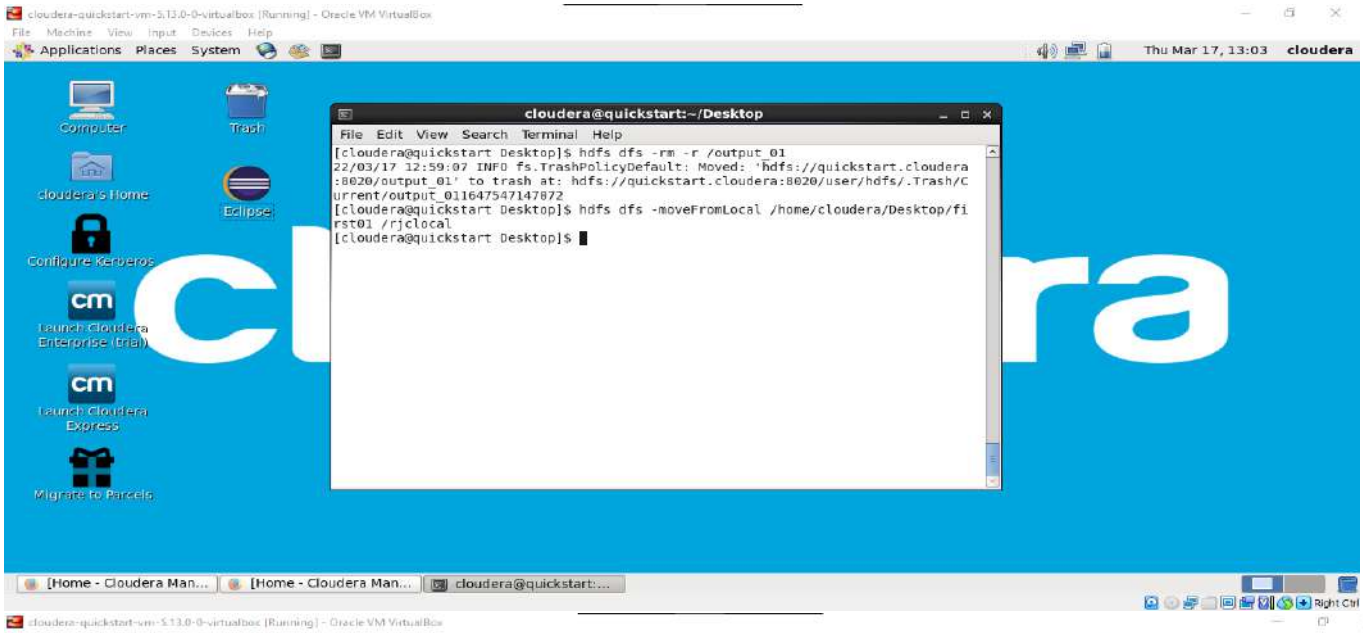
12. **hdfs dfs -rm /<dir_name>/<file_name>** deletes objects and directories full of objects

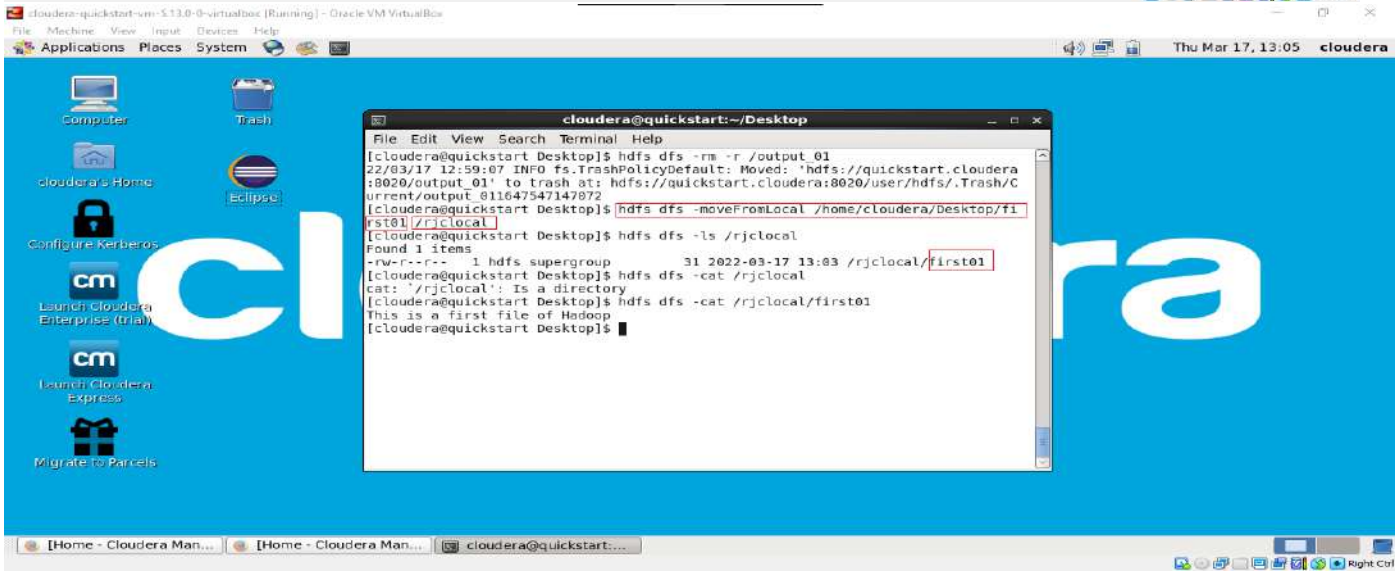


13. **hdfs dfs -rm -r /output_abc** removes directory with objects

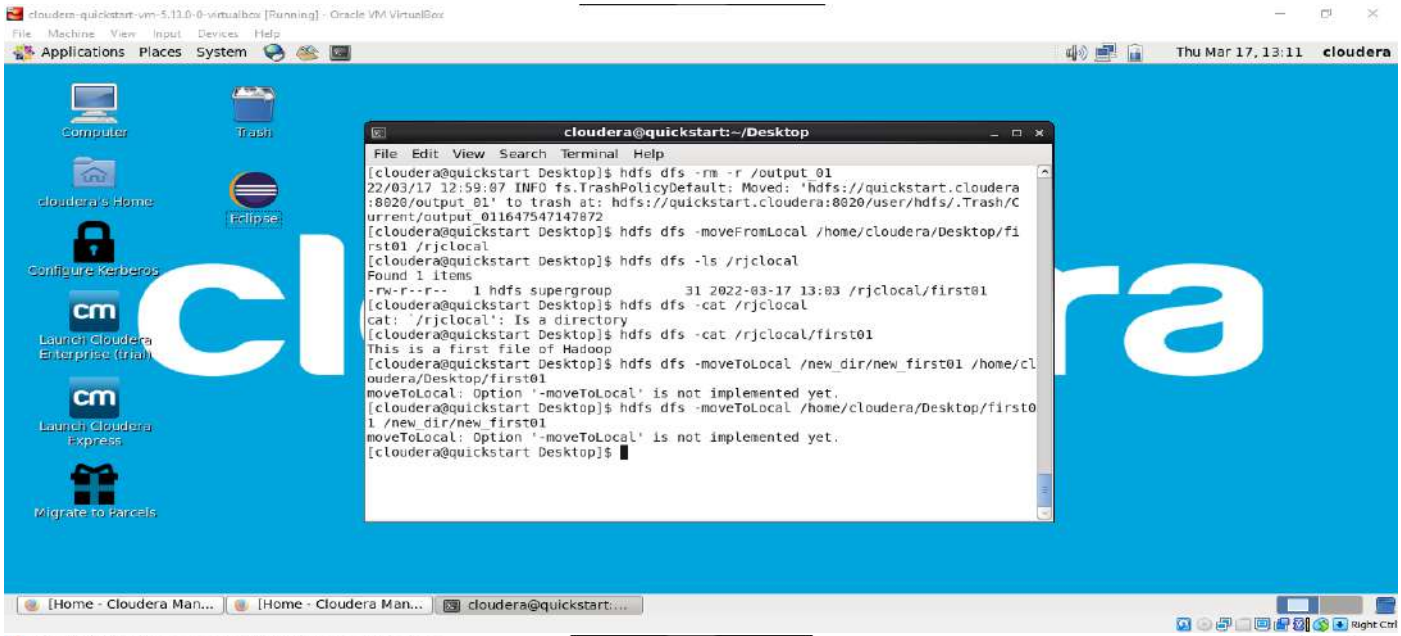


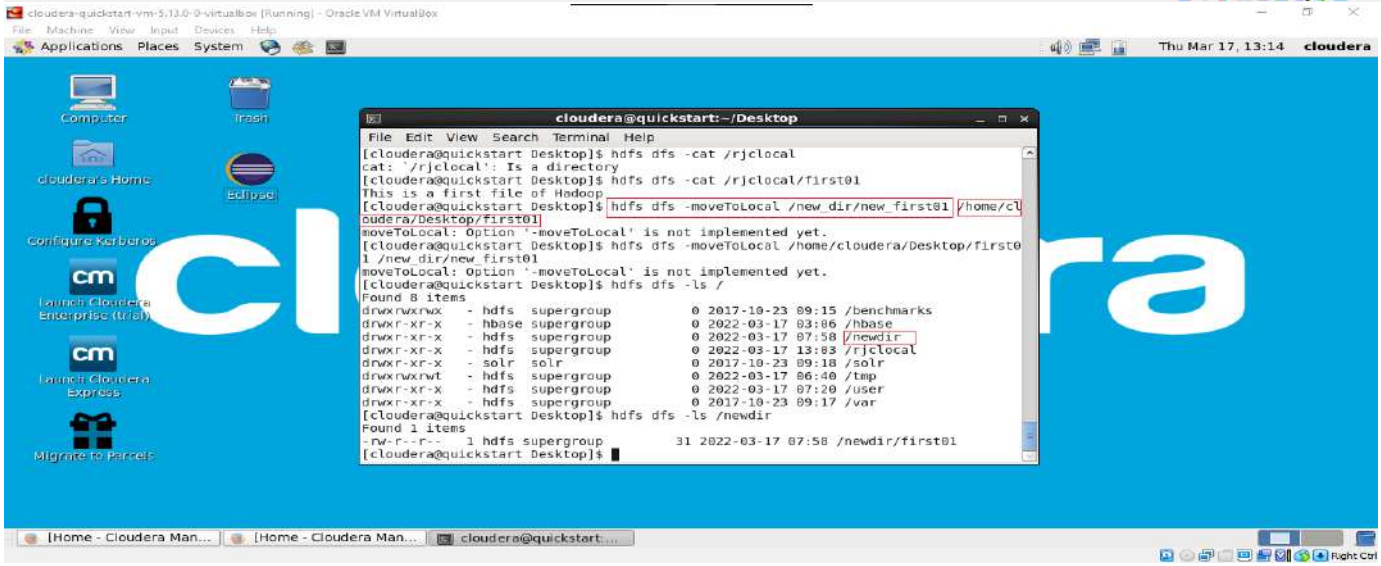
14. **hdfs dfs -moveFromLocal /home/cloudera/Desktop/<file_name> /<dir_name>**moves files from local file system to HDFS



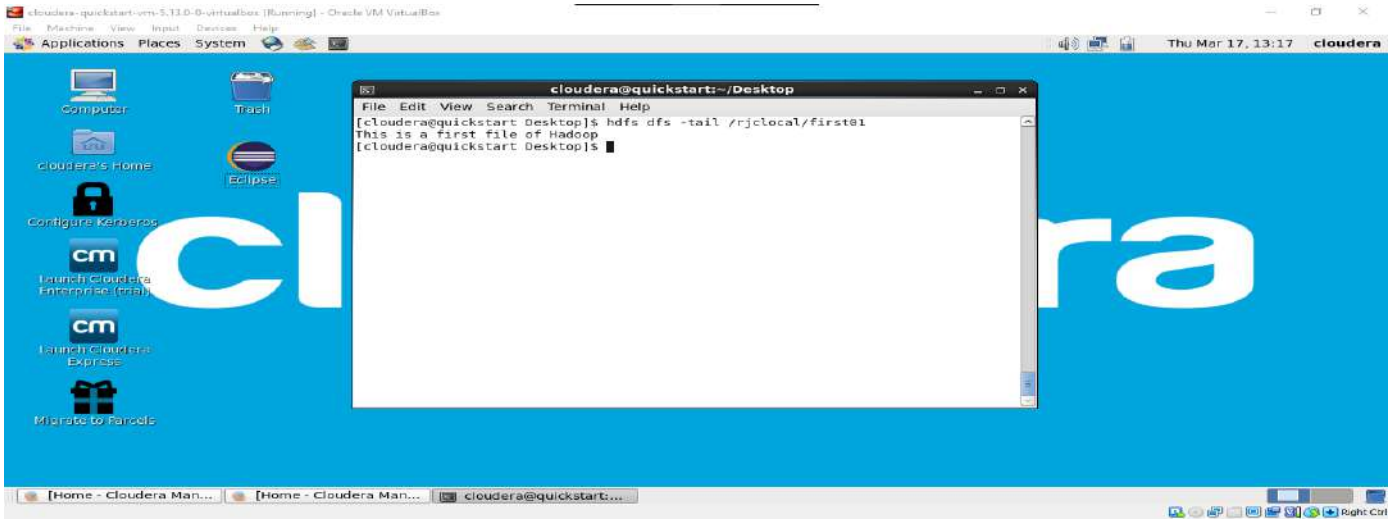


15. **hdfs dfs -moveToLocal /<new_dir_name>/<file_name> /home/cloudera/Desktop/<file_name>** moves files.

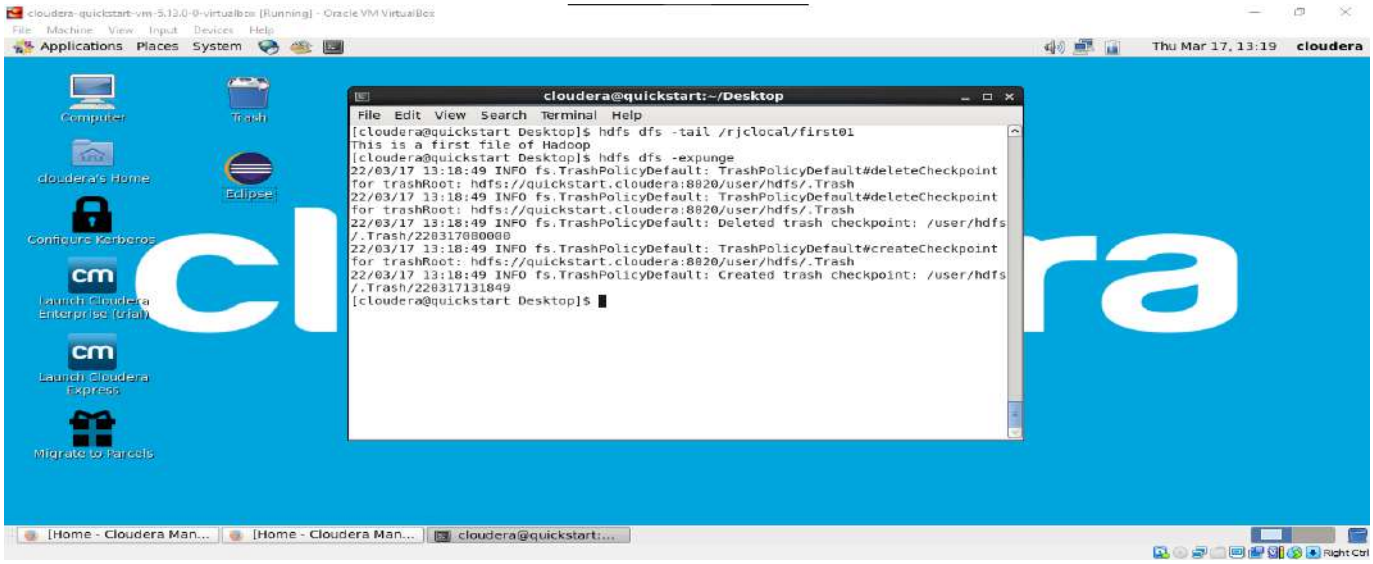




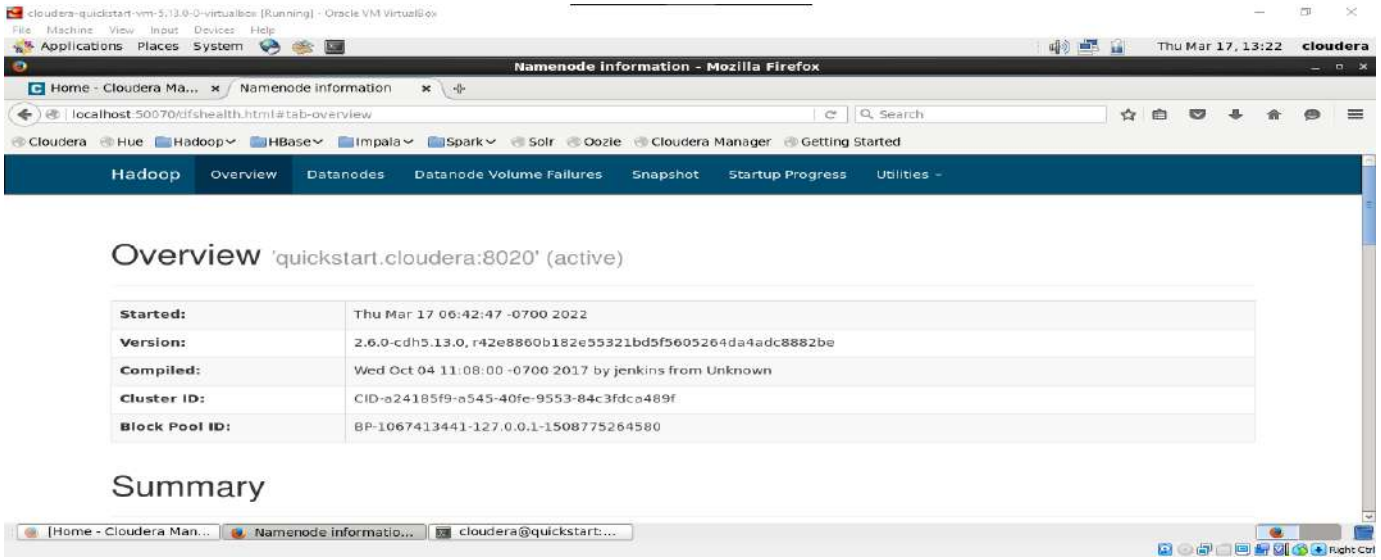
16. **hdfs dfs -tail /<dir_name>/<file_name>** shows last 1KB of file



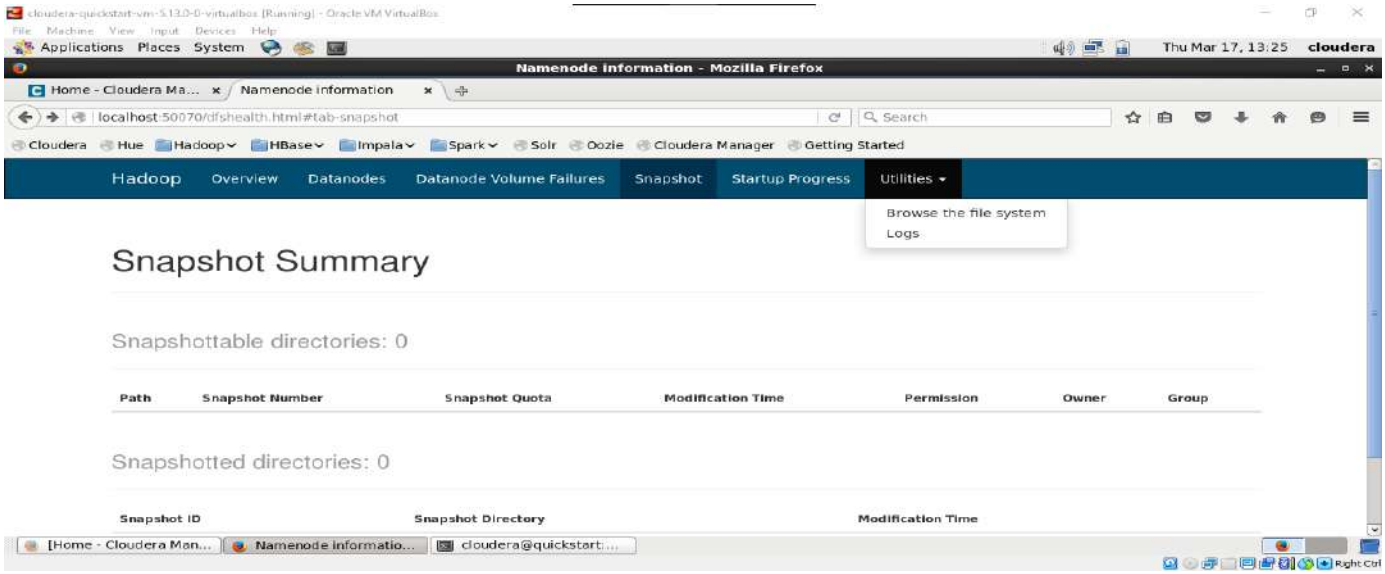
17. **hdfs dfs -expunge** empties the trash available in HDFS

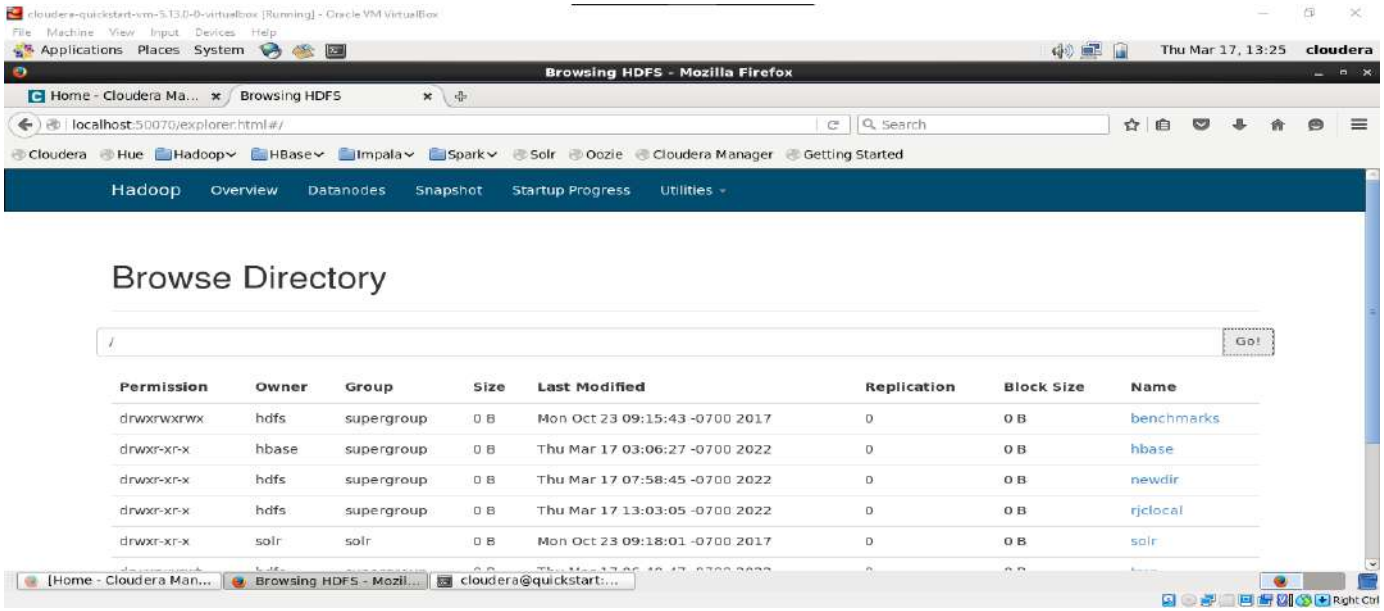


18. First go to local host at **localhost:50070** click on Utilities and select browse the file system

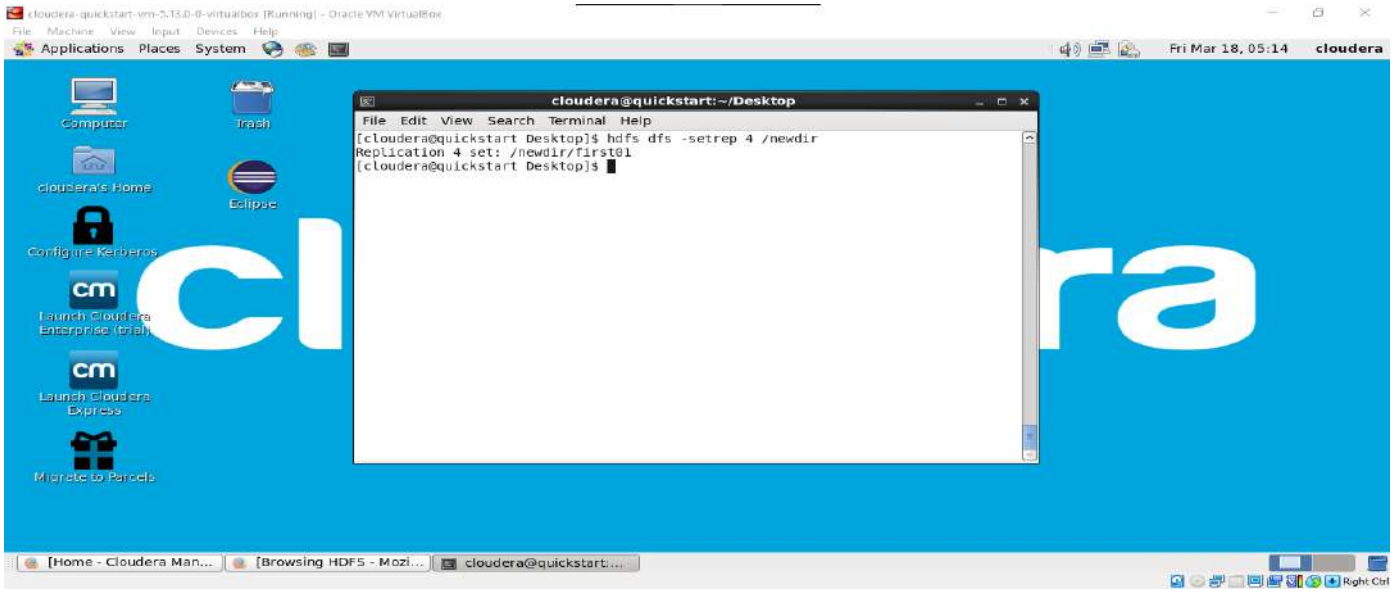


Type **/** in test area and click go to see current directories in file system

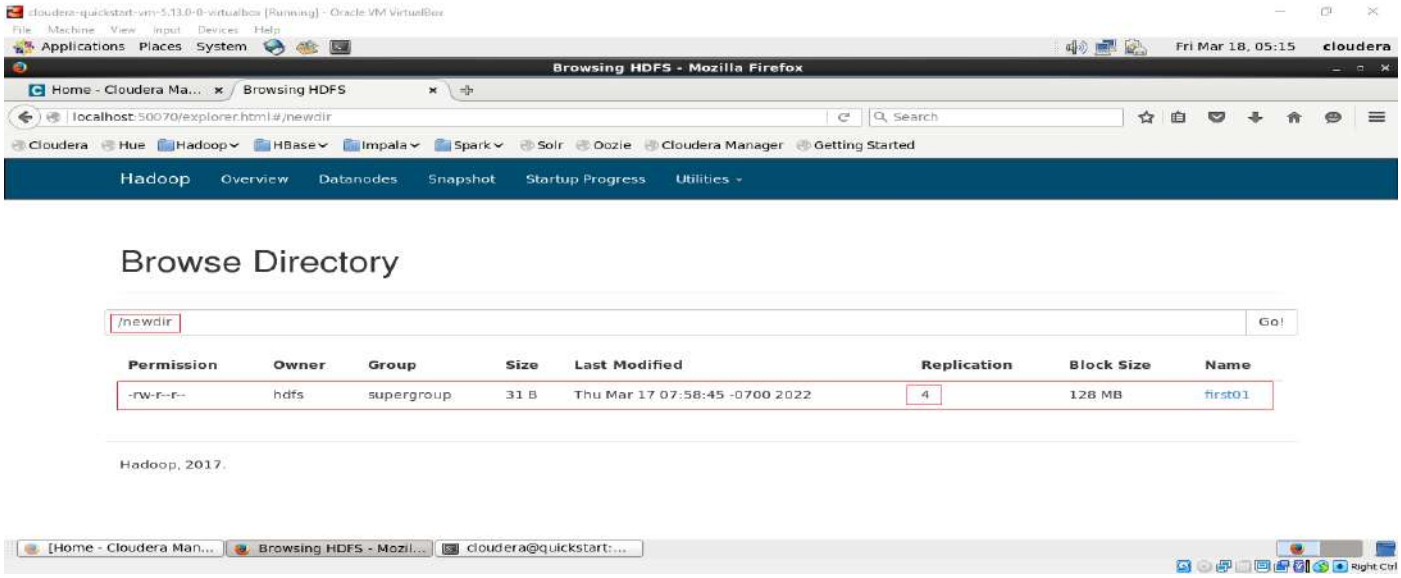




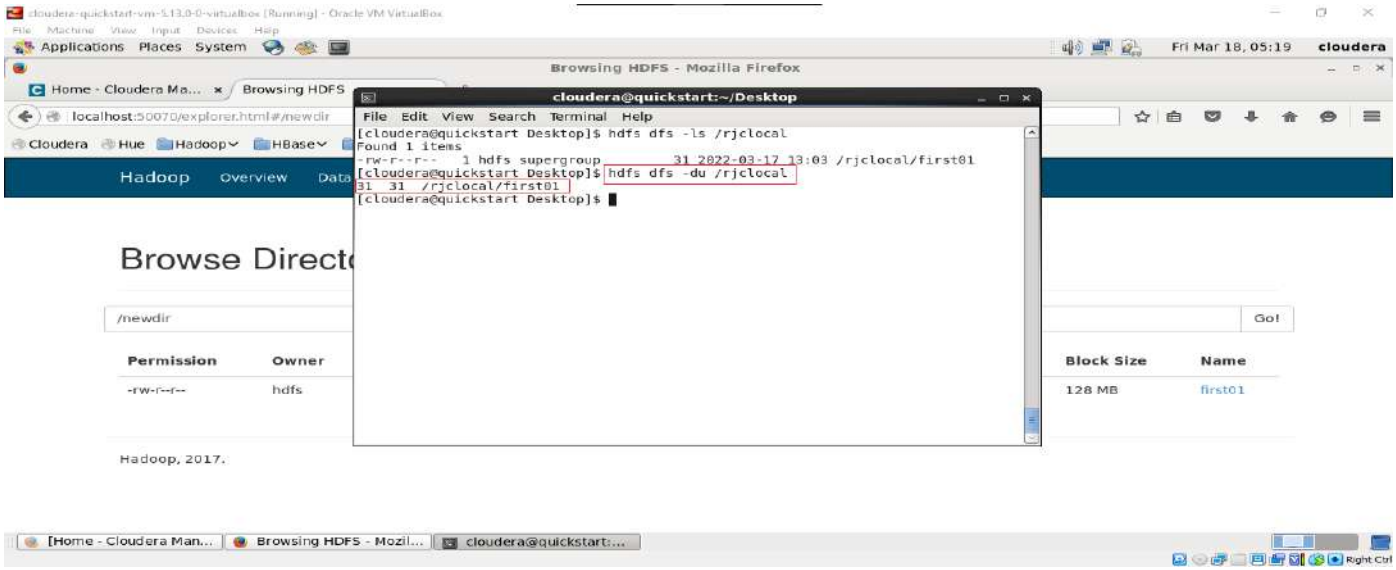
hdfs dfs -setrep 4 /<dir_name> sets replication of files in specifies directory 4 times



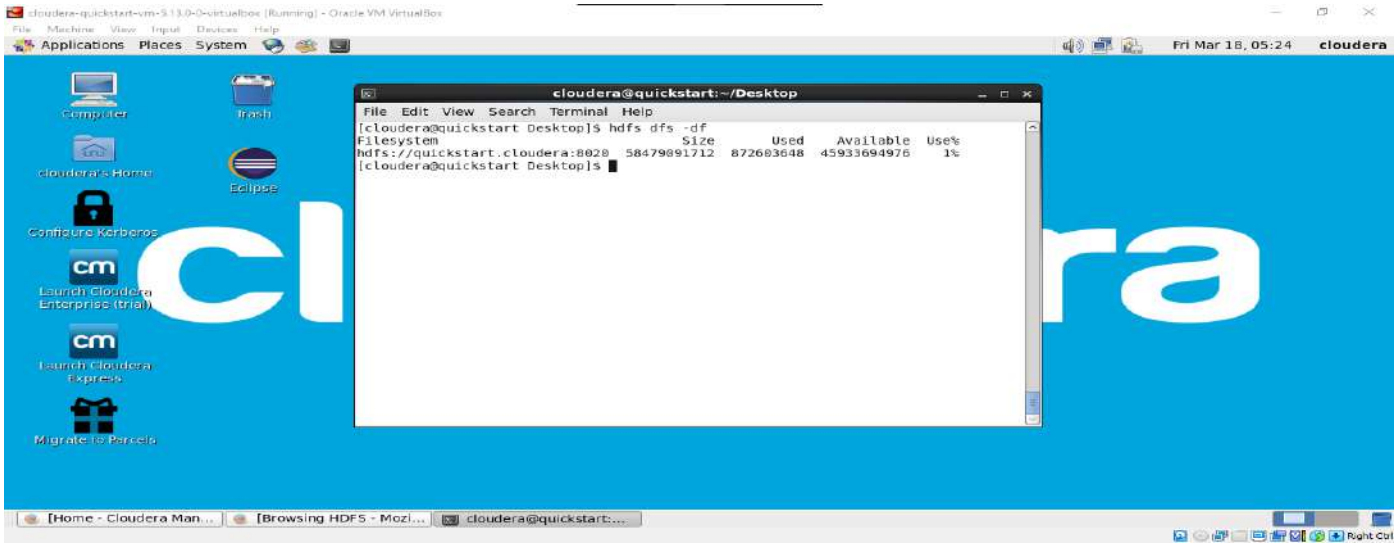
Check **replication** of file within **specified directory**



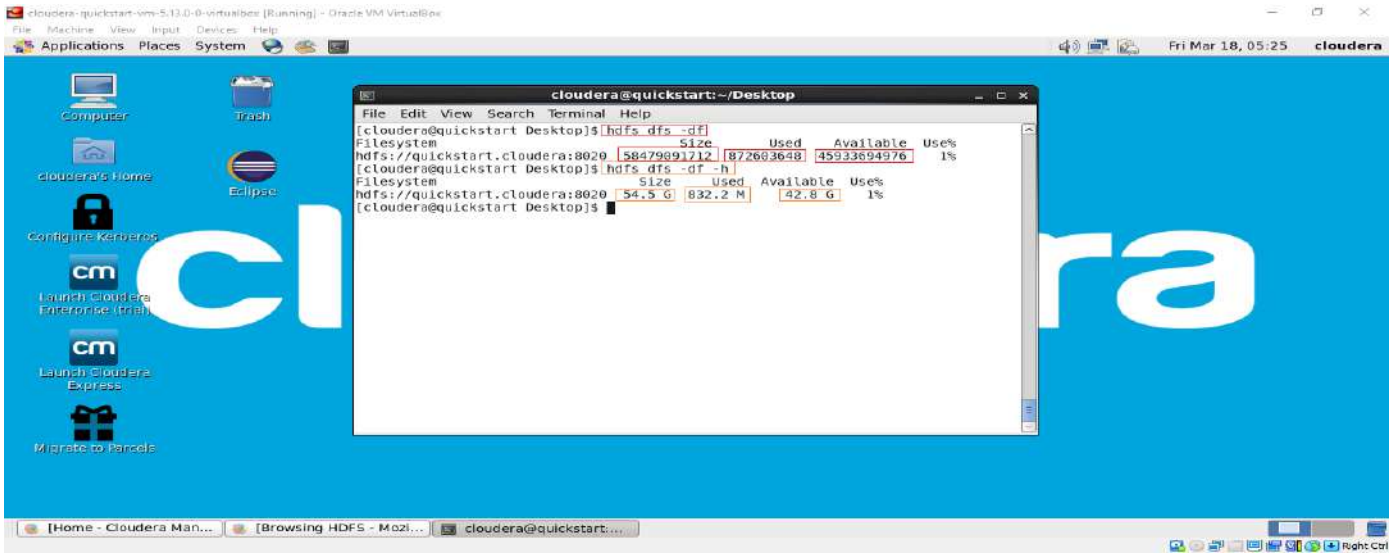
19. **hdfs dfs -du /<dir_name>** gives size of a directory in HDFS



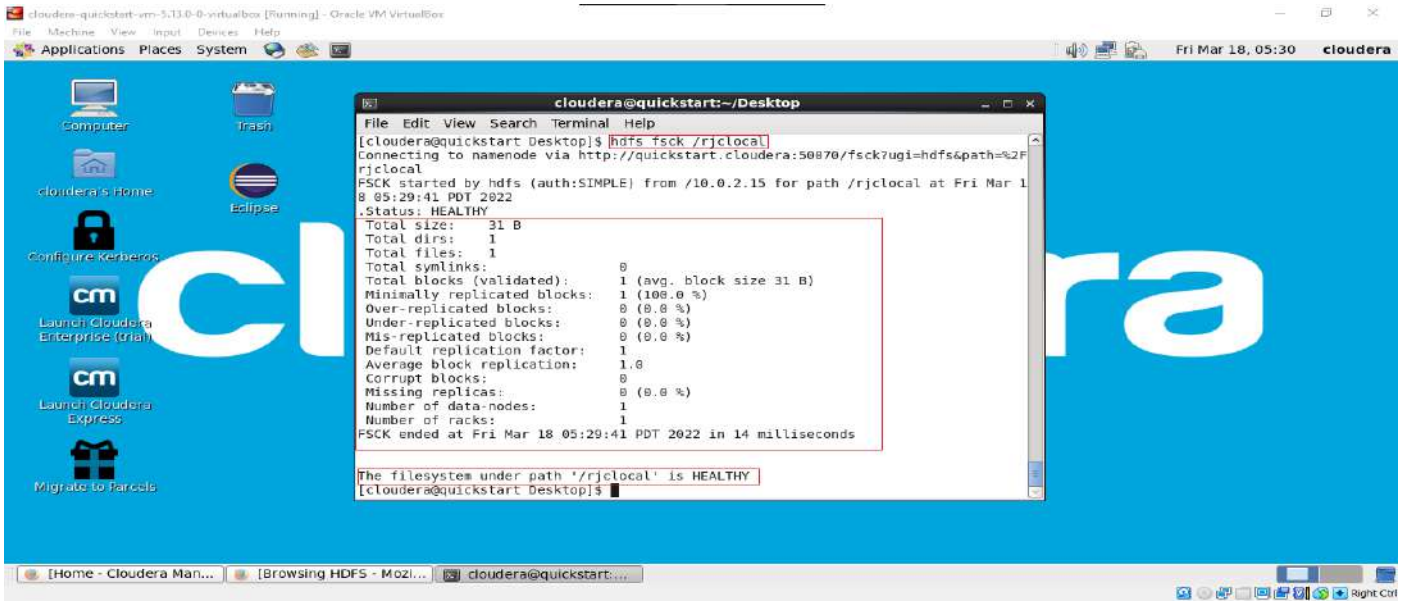
20. **hdfs dfs -df** gives **amount of space in use** and **space available** on currently mounted file system



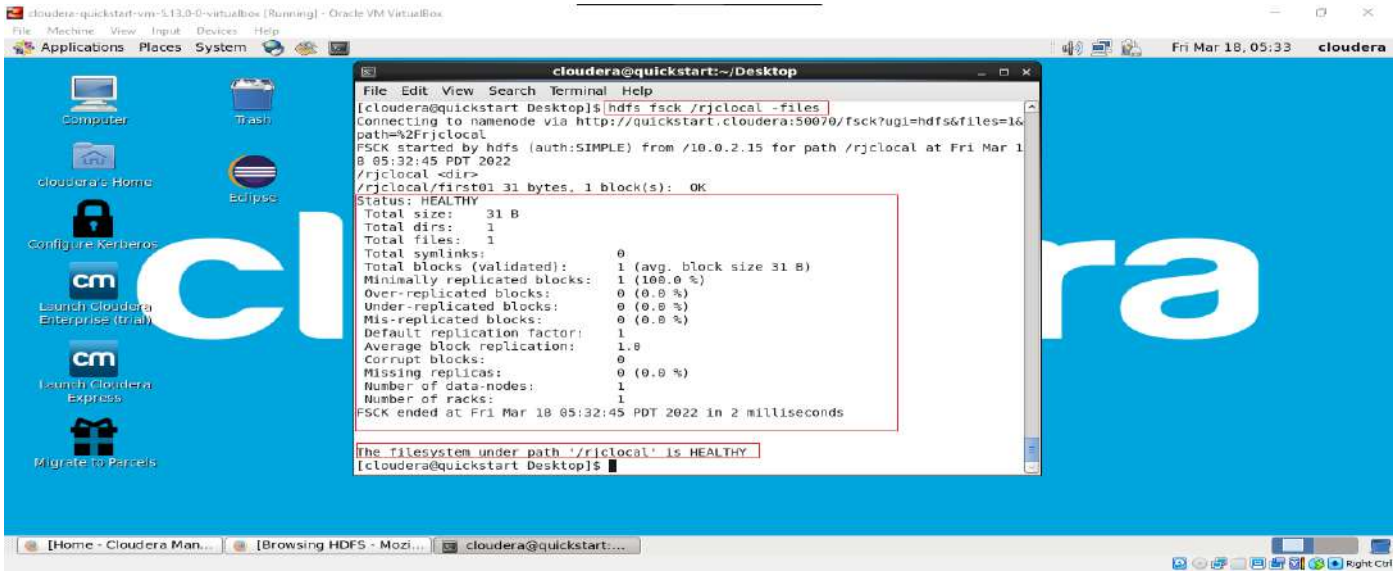
hdfs dfs -df -h gives **space in gb and mb abbreviation**



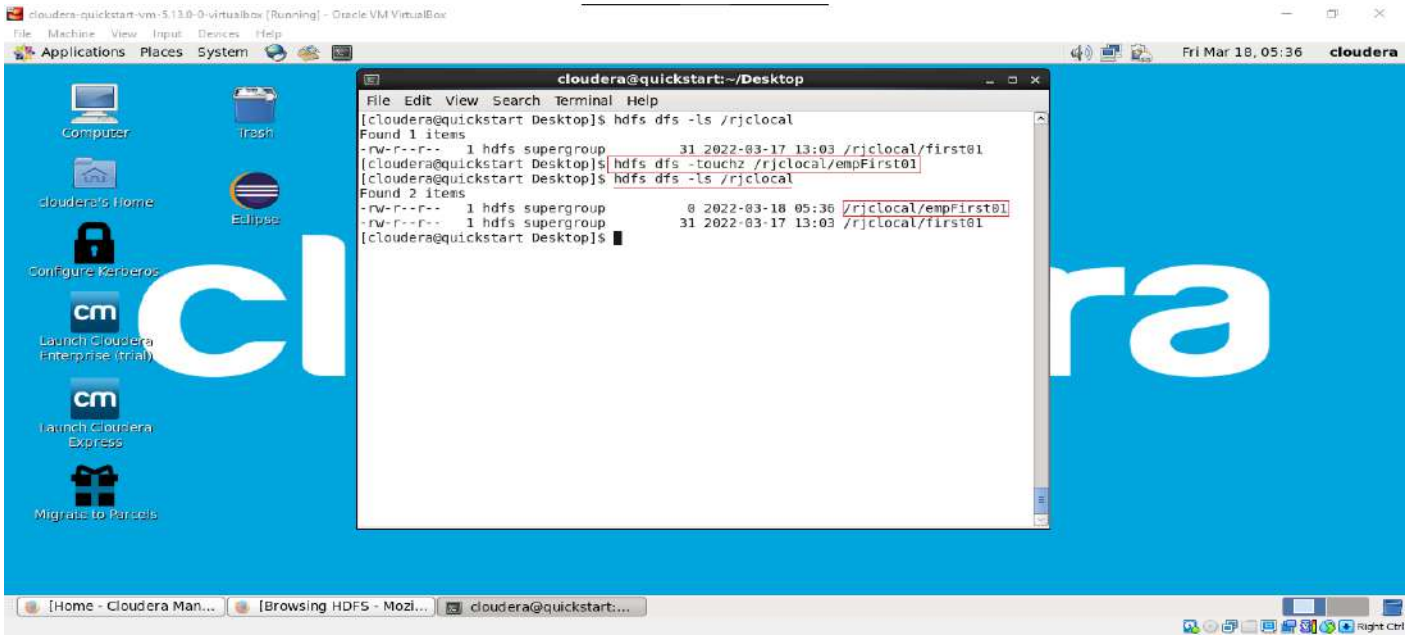
21. **hdfs fsck /<dir_name> checks health of HDFS** and **moves corrupted files to lost+found** directory or It delete a corrupted file present in HDFS



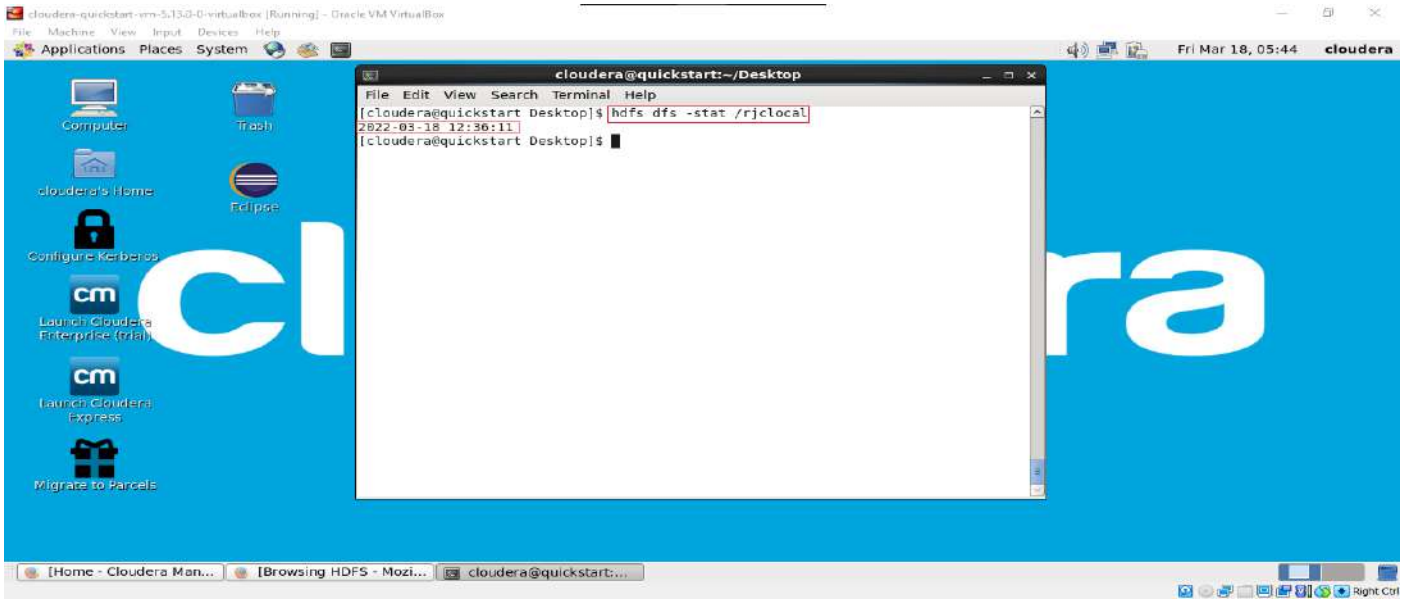
hdfs fsck /rjclocal -files prints out the files being checked



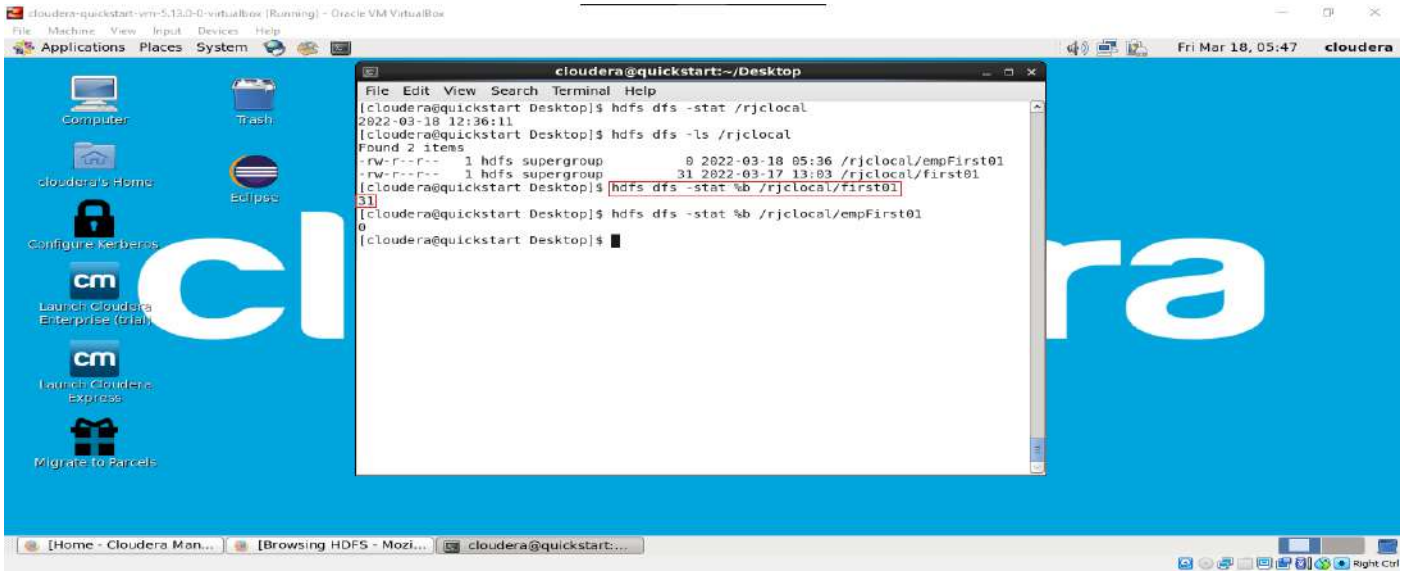
22. **hdfs dfs -touchz /<dir_name>/<emt_file_name>** creates an empty file



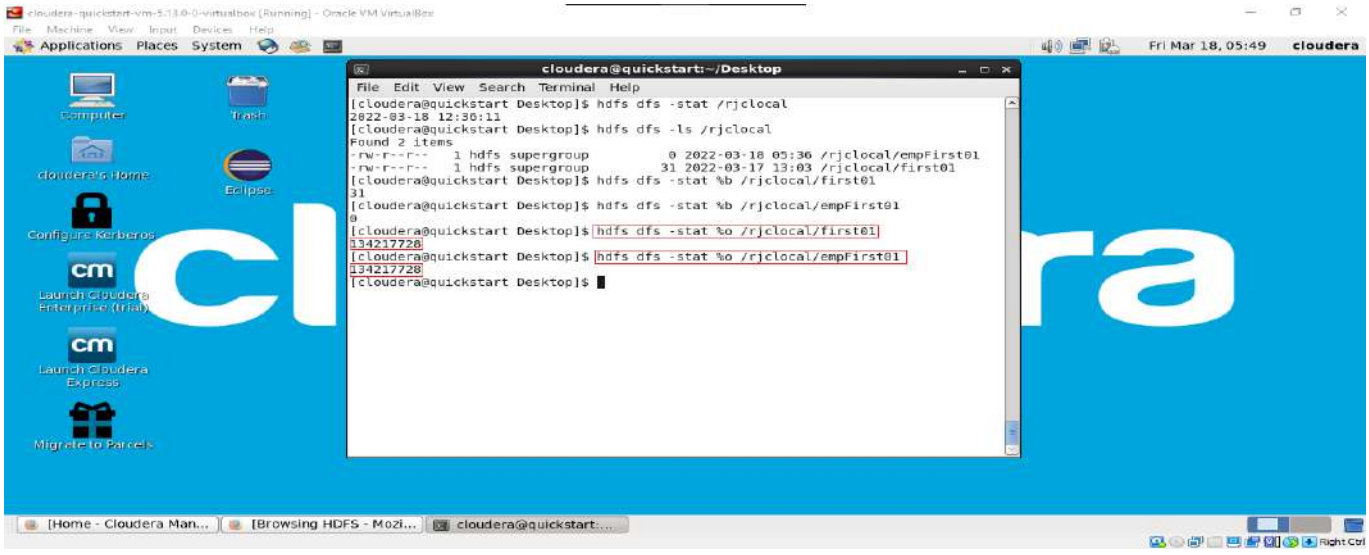
23. **hdfs dfs -stat /<dir_name>** prints statistics about the file or directory it shows recent date of modification



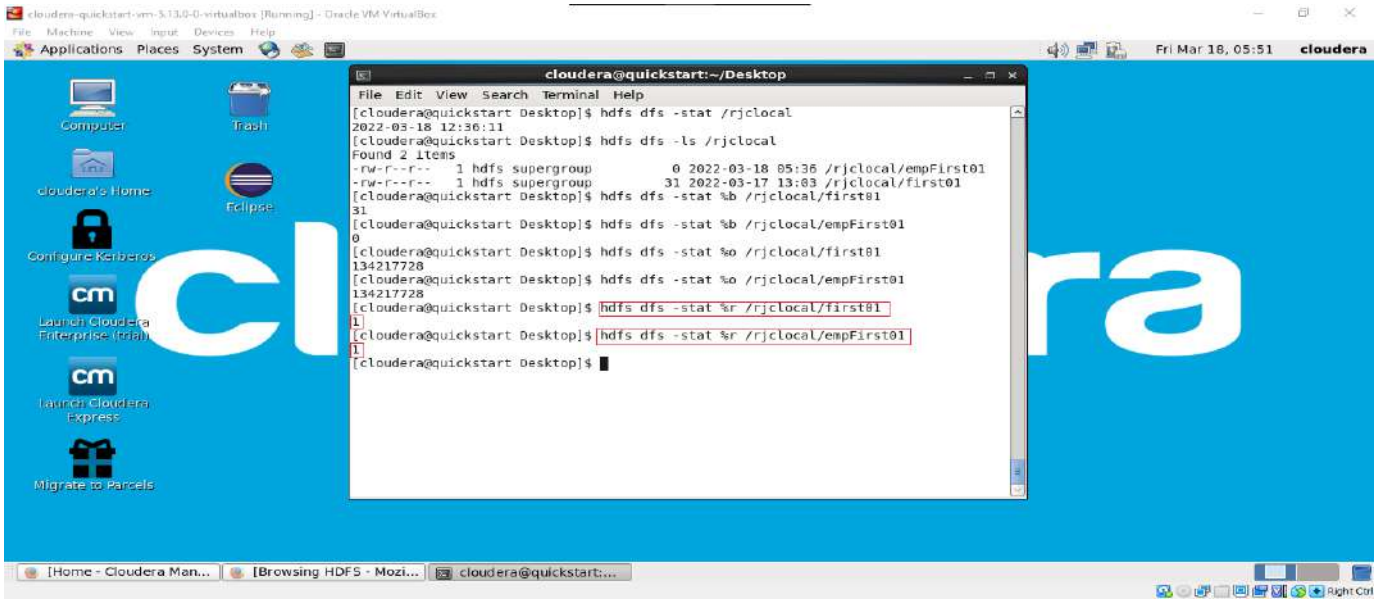
hdfs -stat %b /<dir_name>/<file_name> gives file **size in bytes**



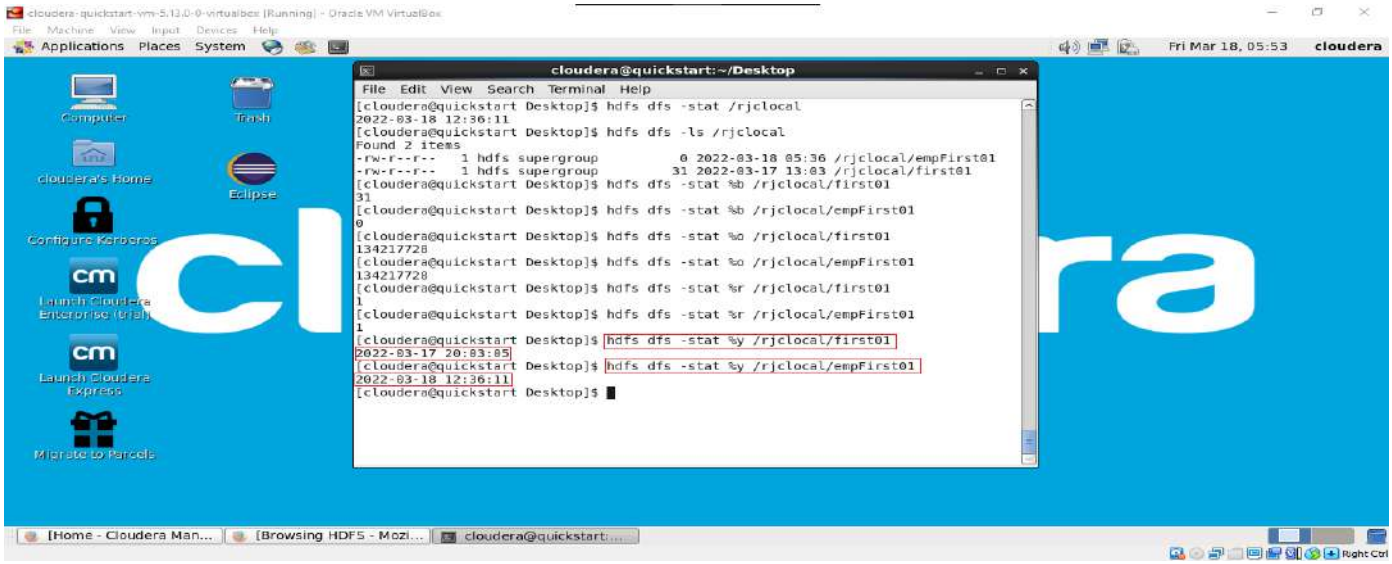
hdfs dfs -stat %o /<dir_name>/<file_name> gives **block size** of file



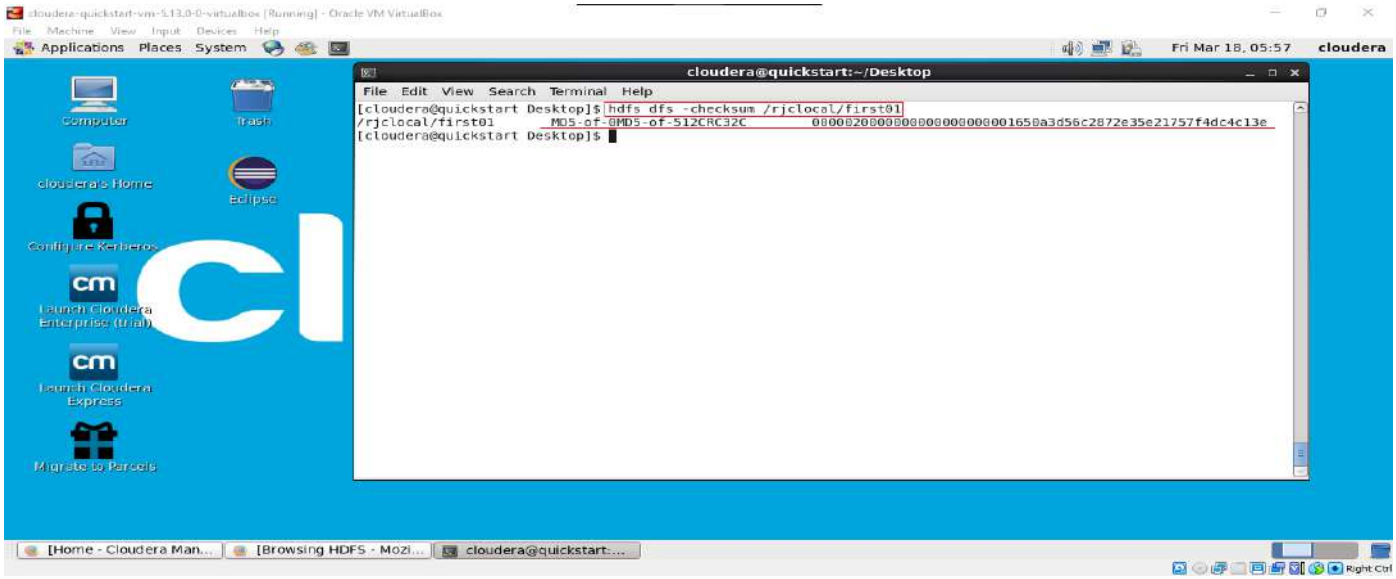
hdfs dfs -stat %r /<dir_name>/<file_name> gives **no. of replication** of file



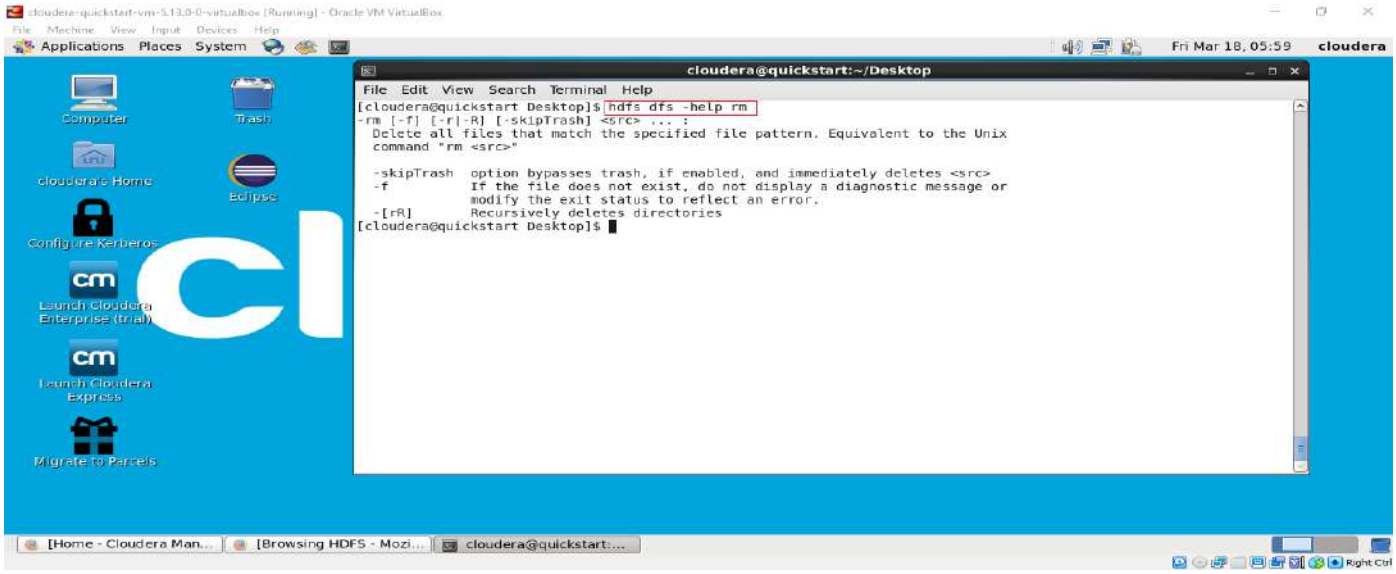
hdfs dfs -stat %y /<dir_name>/<file_name> gives **last modification date**



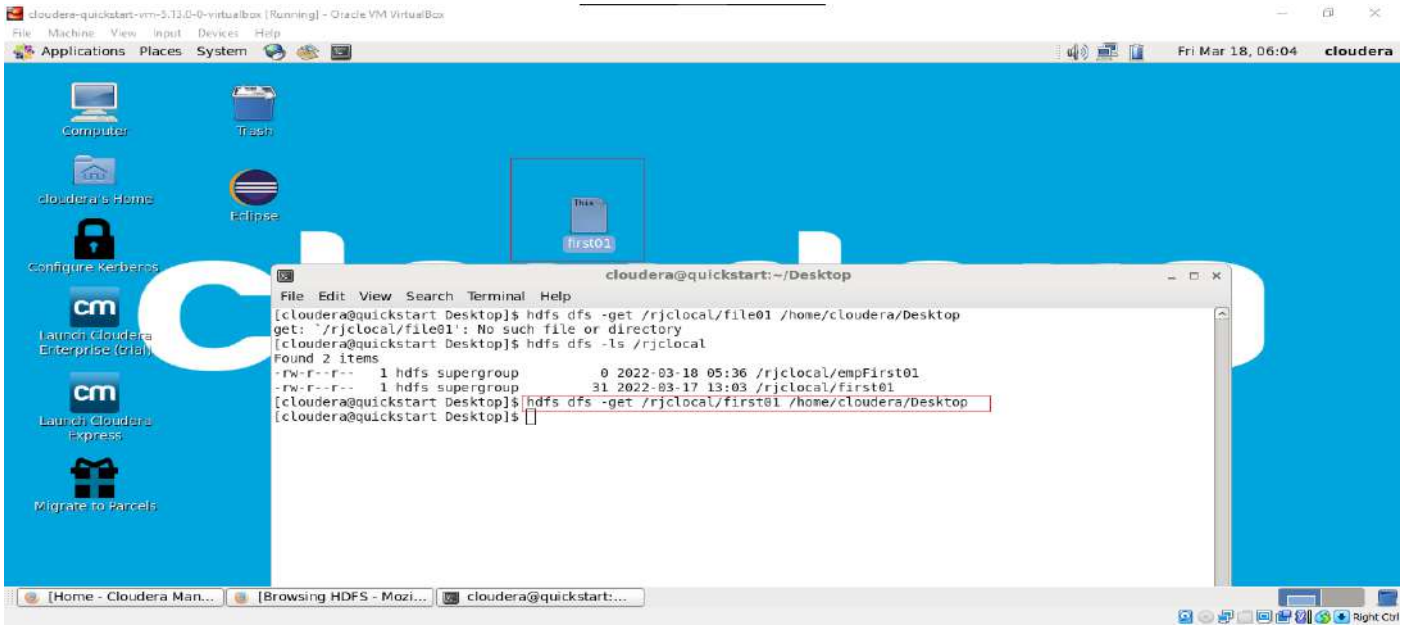
24. **hdfs dfs -checksum /<dir_name>/<file_name>** gives **checksum of file**



25. **hdfs dfs -help rm** shows **syntax of whereas** commands



26. **hdfs dfs -get /<dir_name>/<file_name> /home/cloudera/Desktop** copies file from **HDFS to local file system**



Aim: To Implement WordCount problem using Hadoop MapReduce in Eclipse: (Without Combiner & With Combiner)

What is MapReduce?

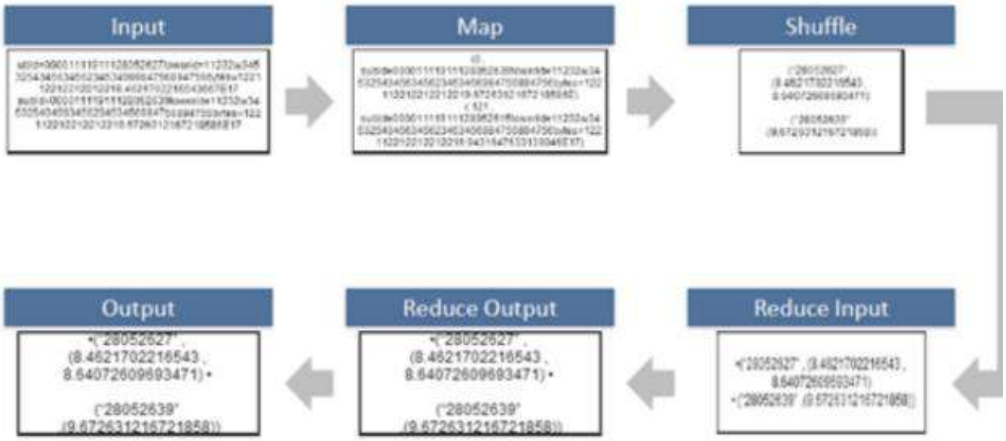
A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form. It was developed in 2004, on the basis of paper titled as "MapReduce:

Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

Steps in Map Reduce

- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.

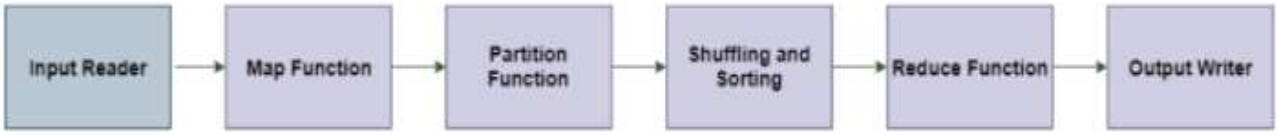


Sort and Shuffle

The sort and shuffle occur on the output of Mapper and before the reducer. When the Mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each Mapper <k2,v2>, we collect all the values for each unique key k2. This output from the shuffle phase in the form of <k2, list(v2)> is sent as input to reducer phase.

Data Flow In MapReduce

MapReduce is used to compute the huge amount of data. To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.



Phases of MapReduce data flow

Input reader

The input reader reads the upcoming data and splits it into the data blocks of the appropriate size (64 MB to 128 MB). Each data block is associated with a Map function.

Once input reads the data, it generates the corresponding key-value pairs. The input files reside in HDFS.

Map function

The map function process the upcoming key-value pairs and generated the corresponding output key-value pairs. The map input and output type may be different from each other.

Partition function

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.

Shuffling and Sorting

The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation time.

The sorting operation is performed on input data for Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

Reduce function

The Reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output.

Output writer

Once the data flow from all the above phases, Output writer executes. The role of Output writer is to write the Reduce output to the stable storage.

To Implement WordCount problem using Hadoop MapReduce in Eclipse:

Hadoop WordCount operation occurs in 3 stages –

- Mapper Phase
- Shuffle Phase
- Reducer Phase

Hadoop WordCount - Mapper Phase Execution

- The text from the input text file is tokenized into words to form a key value pair with all the words present in the input text file. The key is the word from the input file and value is '1'.
- For instance if you consider the sentence "An elephant is an animal".
- The mapper phase in the WordCount example will split the string into individual tokens i.e. words. In this case, the entire sentence will be split into 5 tokens (one for each word) with a value 1 as shown below –

  Key-Value pairs from Hadoop Map Phase Execution-
  (an,1)
  (elephant,1)
  (is,1)
  (an,1)
  (animal,1)

Hadoop WordCount Example- Shuffle Phase Execution

- After the map phase execution is completed successfully, shuffle phase is executed automatically wherein the key-value pairs generated in the map phase are taken as input and then sorted in alphabetical order.
- After the shuffle phase is executed from the WordCount example code, the output will look like this –
  (an,1)
  (an,1)
  (animal,1)
  (elephant,1)
  (is,1)

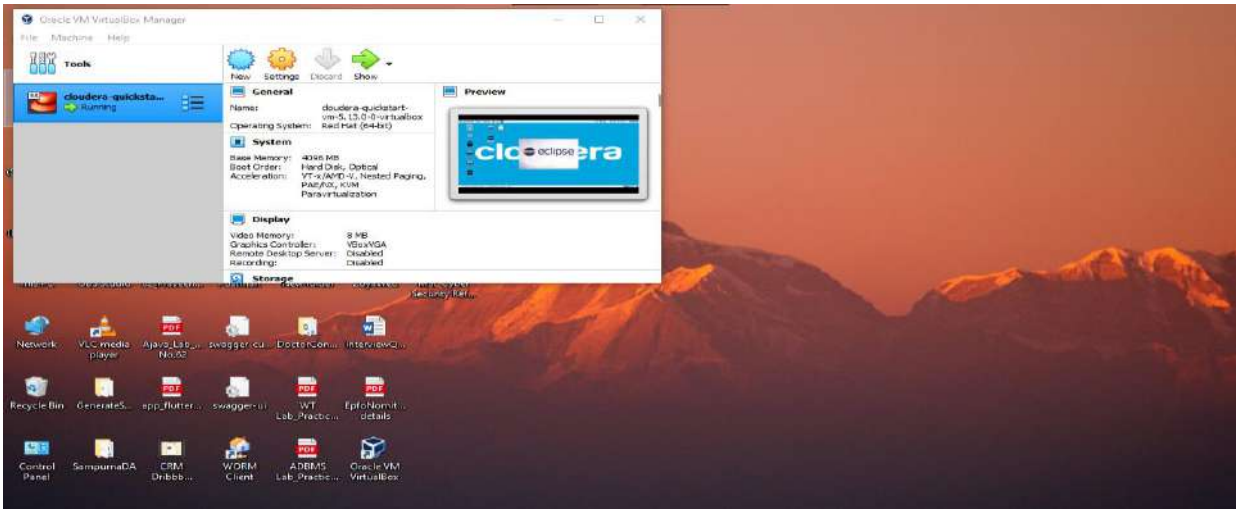
Hadoop WordCount Example- Reducer Phase Execution

- In the reduce phase, all the keys are grouped together and the values for similar keys are added up to find the occurrences for a particular word.

- It is like an aggregation phase for the keys generated by the map phase. The reducer phase takes the output of shuffle phase as input and then reduces the key-value pairs to unique keys with values added up.
- In our example “An elephant is an animal.” is the only word that appears twice in the sentence.
- After the execution of the reduce phase of MapReduce WordCount example program, appears as a key only once but with a count of 2 as shown below –
  (an,2)
  (animal,1)
  (elephant,1)
  (is,1)
- This is how the MapReduce word count program executes and outputs the number of occurrences of a word in any given input file.
- An important point to note during the execution of the WordCount example is that the mapper class in the WordCount program will execute completely on the entire input file and not just a single sentence.
- Suppose if the input file has 15 lines then the mapper class will split the words of all the 15 lines and form initial key value pairs for the entire dataset.
- The reducer execution will begin only after the mapper phase is executed successfully.

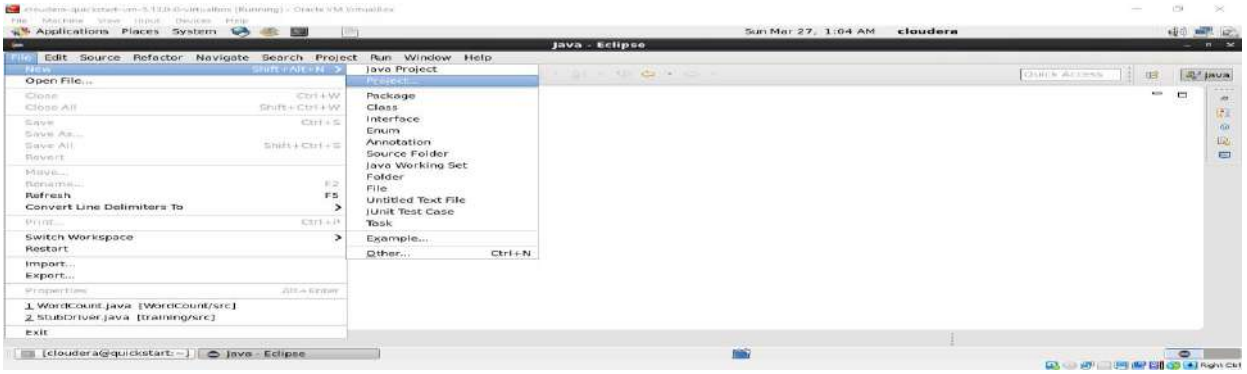
Steps for Word Count in Cloudera: (Without Combiner)

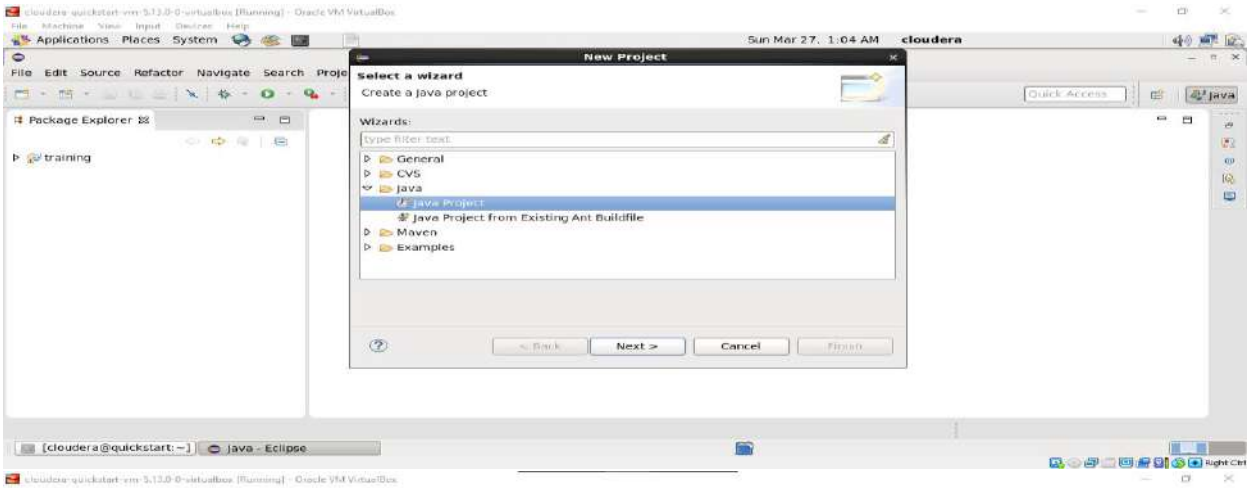
1. Open VirtualBox and then start Cloudera VM

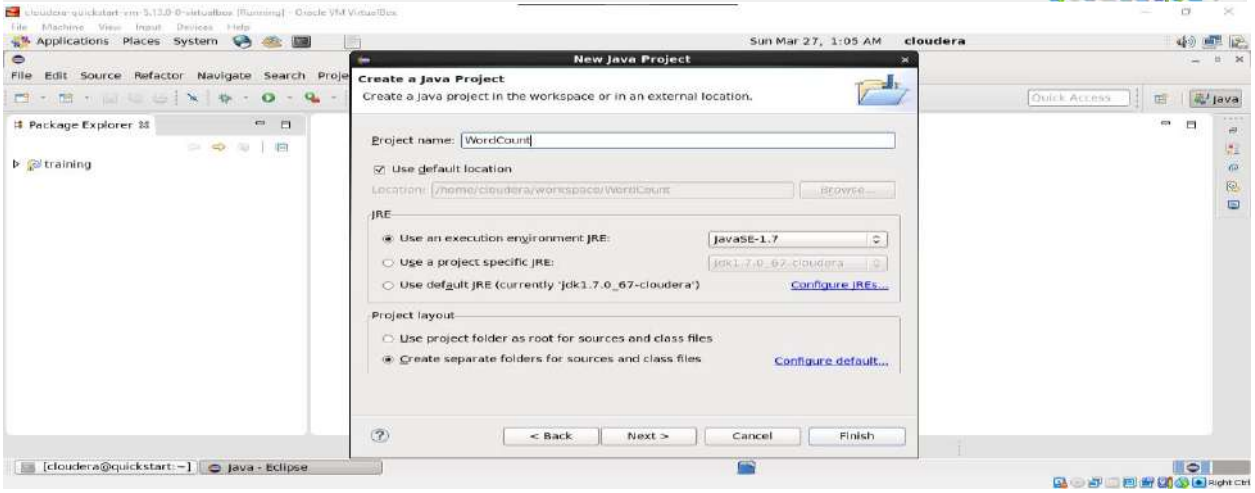


2. Open Eclipse

3. Create a New Java Project: **File** > **New** > **Project** > Java Project > Next Set project name as "**WordCount**".

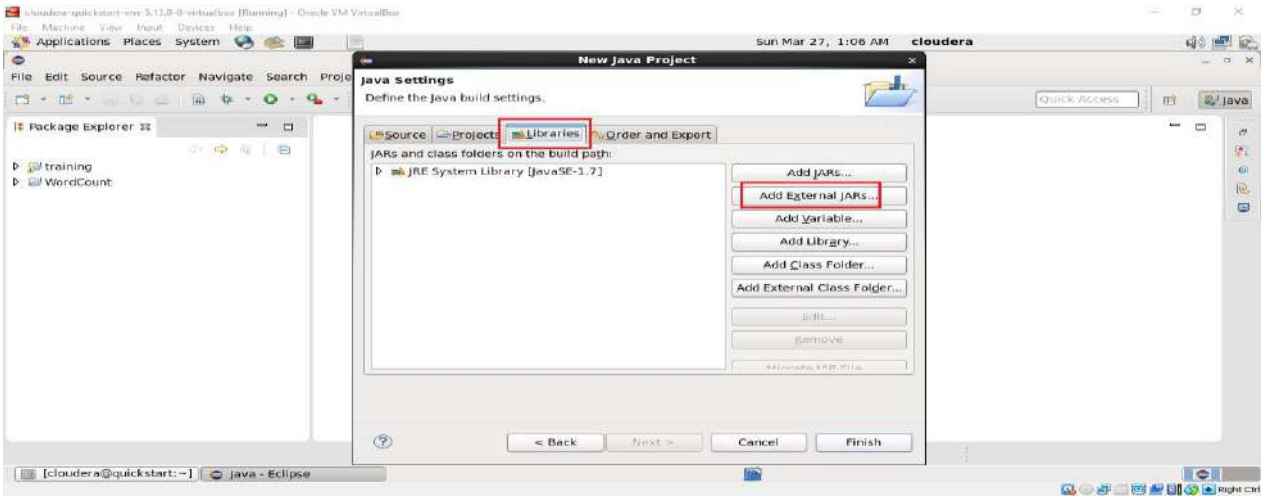


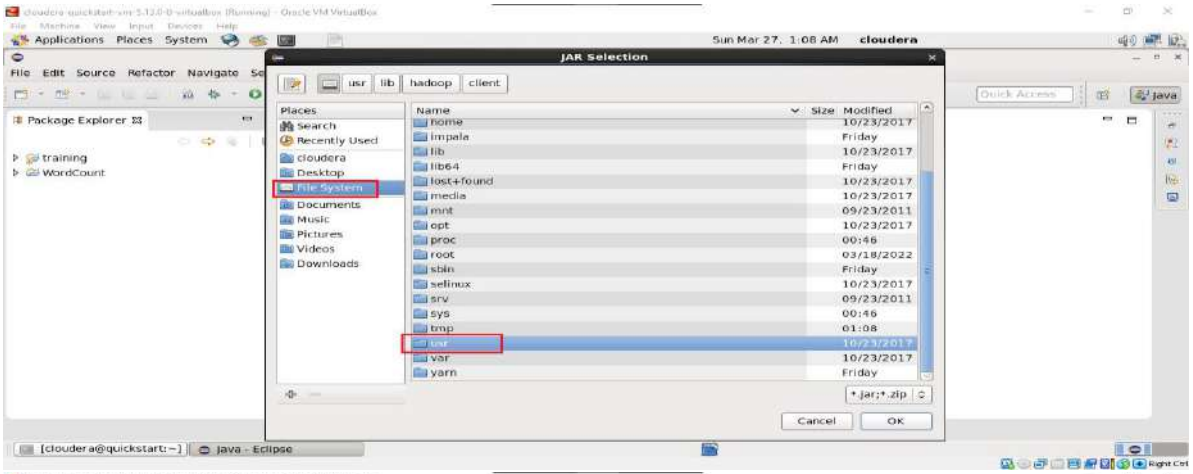


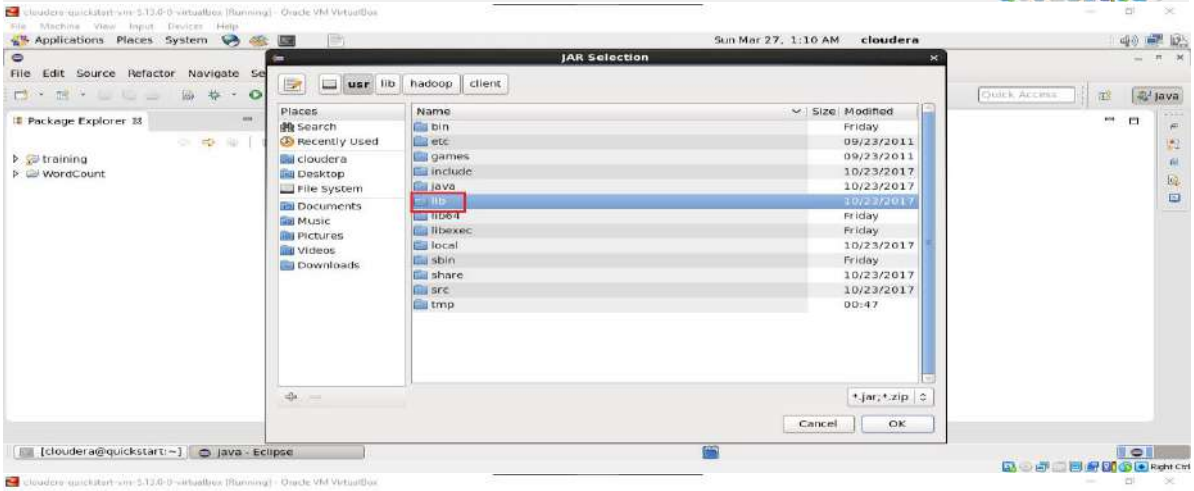


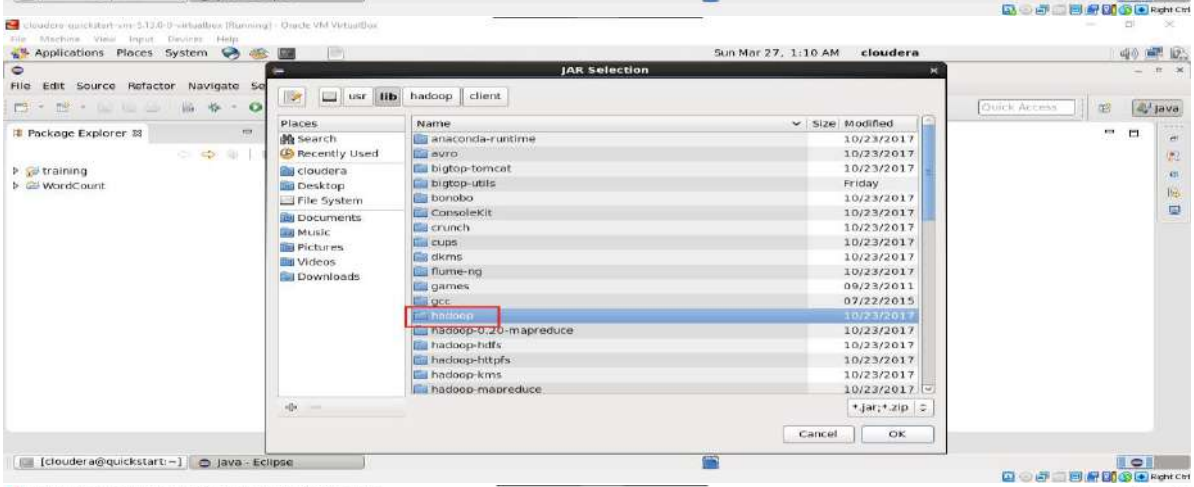
4. Adding the **hadoop libraries** to the project click on:

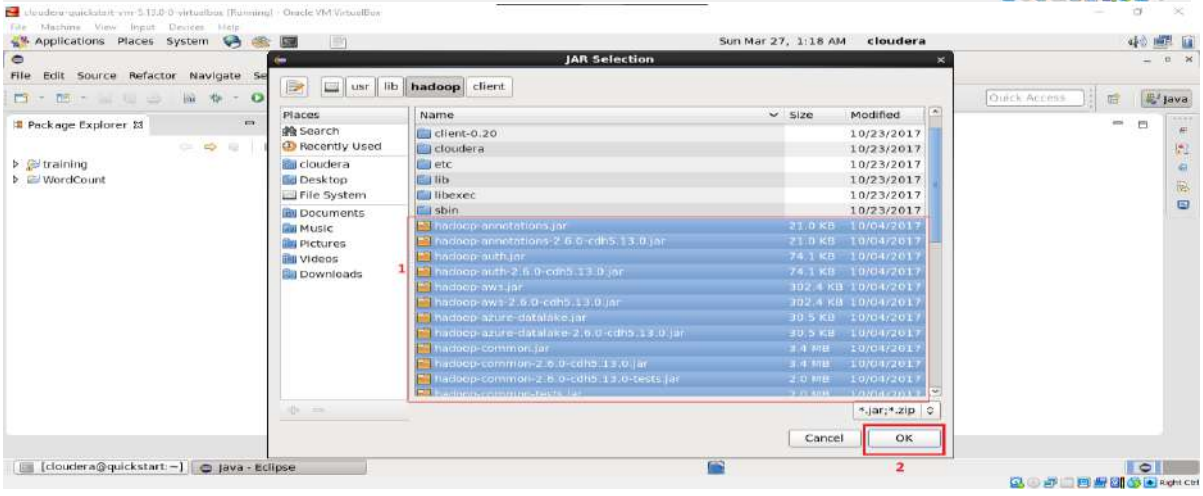
Libraries > Add External JARs > File System > usr > lib > hadoop > client > select all JAR files > OK > Finish.

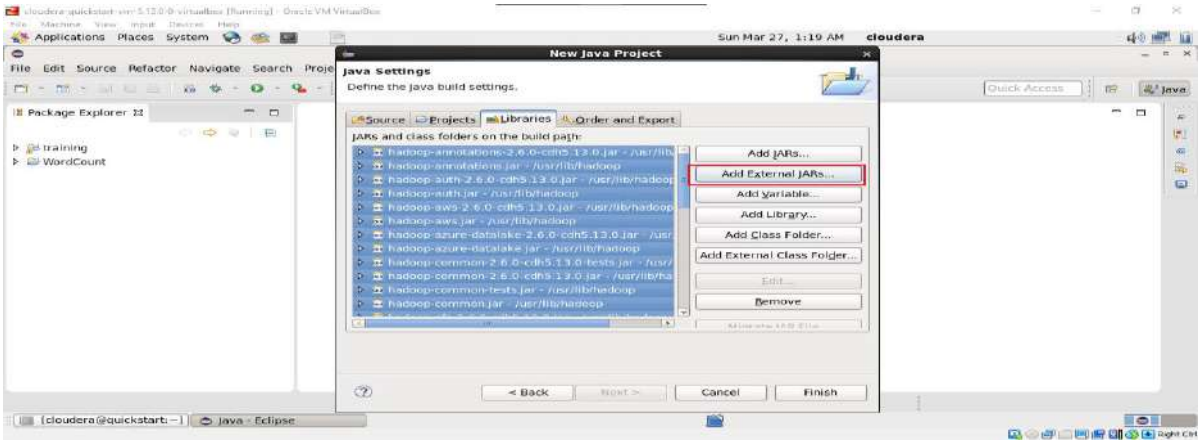


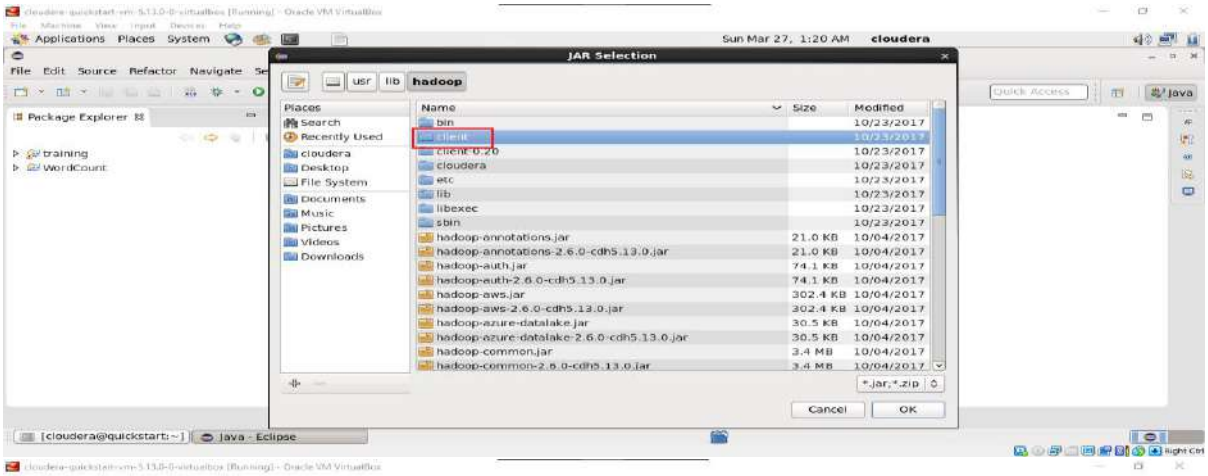

JAR Selection
usr lib hadoop client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name Size Modified
home 10/23/2017
impala Friday
lib 10/23/2017
lib64 Friday
lost+found 10/23/2017
media 10/23/2017
mnt 09/23/2011
opt 10/23/2017
proc 00:46
root 03/18/2022
sbin Friday
selinux 10/23/2017
srv 09/23/2011
sys 00:46
tmp 01:08
usr 10/23/2017
var 10/23/2017
yarn Friday
.jar;.zip
Cancel
OK

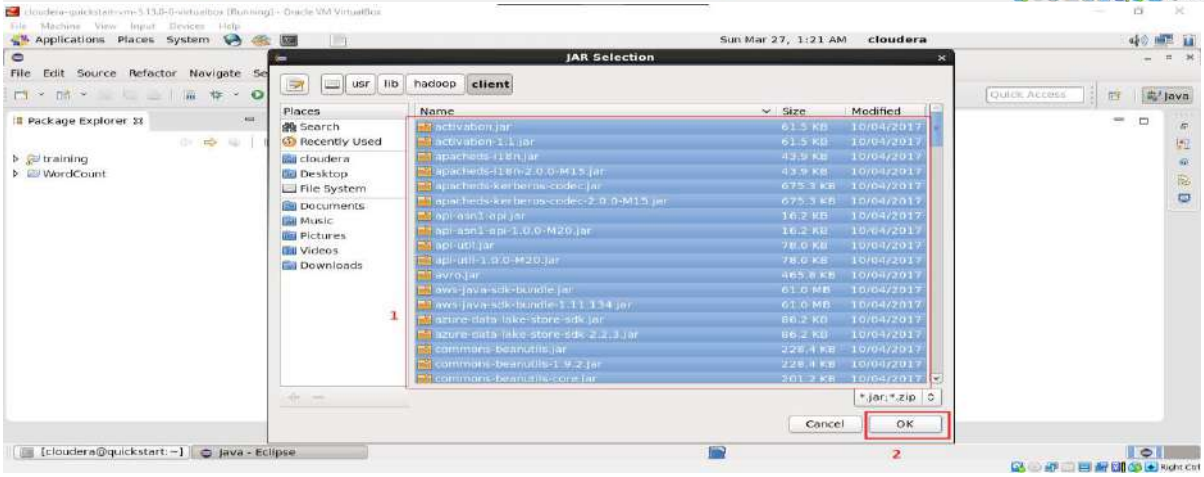

JAR Selection
usr lib hadoop client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name Size Modified
bin Friday
etc 09/23/2011
games 09/23/2011
include 10/23/2017
java 10/23/2017
lib 10/23/2017
lib64 Friday
libexec Friday
local 10/23/2017
sbin Friday
share 10/23/2017
src 10/23/2017
tmp 00:47
.jar;.zip
Cancel
OK

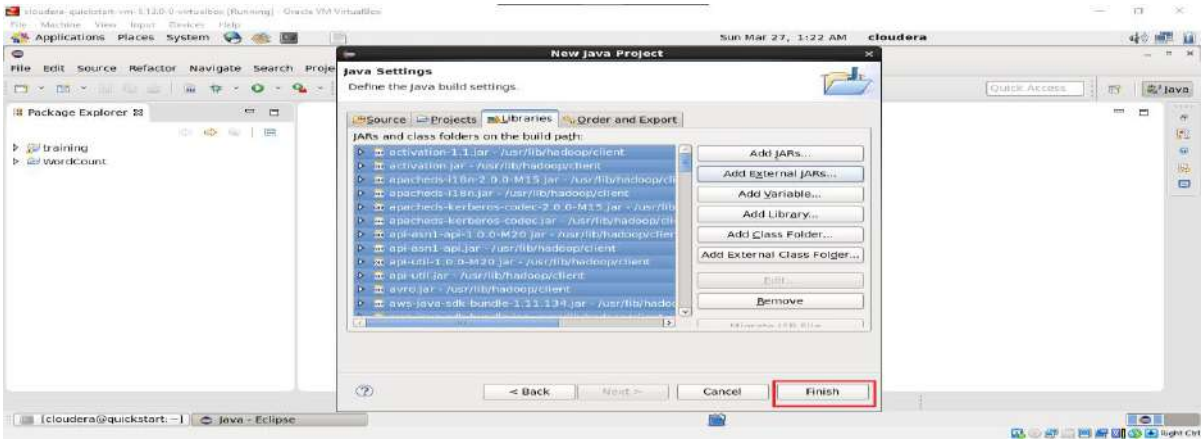

JAR Selection
usr lib hadoop client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name Size Modified
anaconda-runtime 10/23/2017
avro 10/23/2017
bigtop-tomcat 10/23/2017
bigtop-utils Friday
bonobo 10/23/2017
ConsoleKit 10/23/2017
crunch 10/23/2017
cups 10/23/2017
dkms 10/23/2017
flume-ng 10/23/2017
games 09/23/2011
gcc 07/22/2015
hadoop 10/23/2017
hadoop-0.20-mapreduce 10/23/2017
hadoop-hdfs 10/23/2017
hadoop-httpfs 10/23/2017
hadoop-kms 10/23/2017
hadoop-mapreduce 10/23/2017
.jar;.zip
Cancel
OK


JAR Selection
usr lib hadoop client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name Size Modified
client-0.20 10/23/2017
cloudera 10/23/2017
etc 10/23/2017
lib 10/23/2017
libexec 10/23/2017
sbin 10/23/2017
hadoop-annotations.jar 21.0 KB 10/04/2017
hadoop-annotations-2.6.0-cdh5.13.0.jar 21.0 KB 10/04/2017
hadoop-auth.jar 74.1 KB 10/04/2017
hadoop-auth-2.6.0-cdh5.13.0.jar 74.1 KB 10/04/2017
hadoop-aws.jar 302.4 KB 10/04/2017
hadoop-aws-2.6.0-cdh5.13.0.jar 302.4 KB 10/04/2017
hadoop-azure-datalake.jar 30.5 KB 10/04/2017
hadoop-azure-datalake-2.6.0-cdh5.13.0.jar 30.5 KB 10/04/2017
hadoop-common.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0-tests.jar 2.0 MB 10/04/2017
.jar;.zip
Cancel
OK
1
2

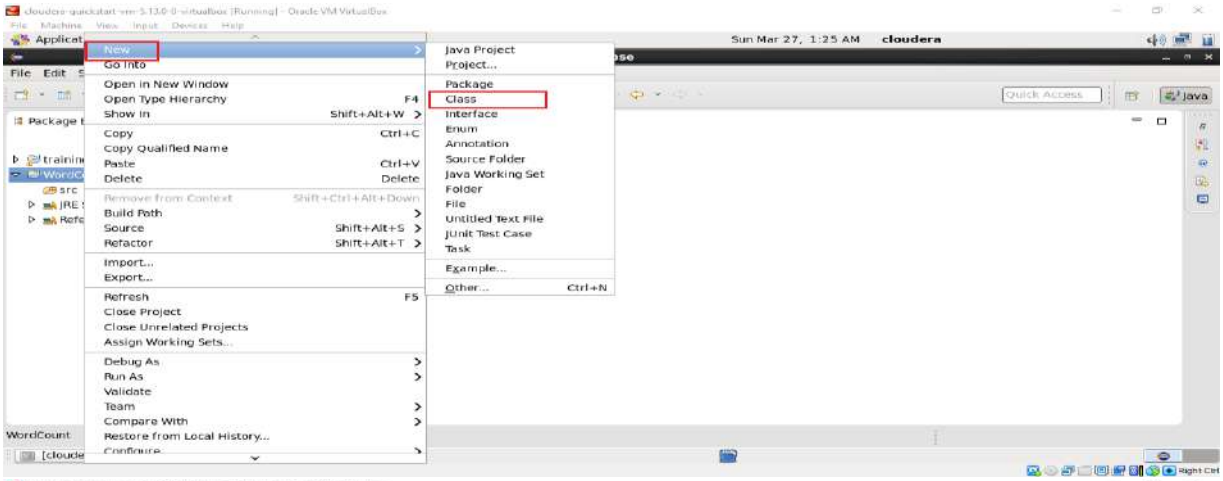


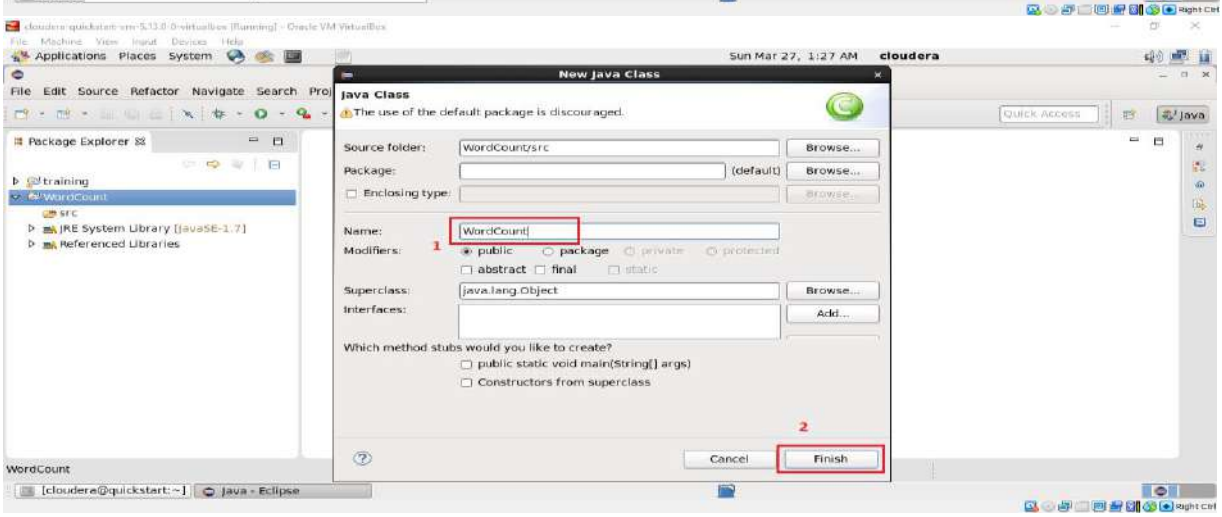






5. **Right click** on the name of project
   "**WordCount**" > New > Class (Don't write anything for package) > class name "**WordCount**" > Finish
   Then WordCount.java will open if not then open WordCount.java file manually





6. Source Code:

```
1  //Packages
2  import java.io.IOException;
16
17 public class WordCount {
18     //Map Logic
19     public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
31
32     //Reducer
33     public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
46
47     //Main Function
48     public static void main(String[] args) throws Exception {
62 }
63
```

```
//Packages

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
```

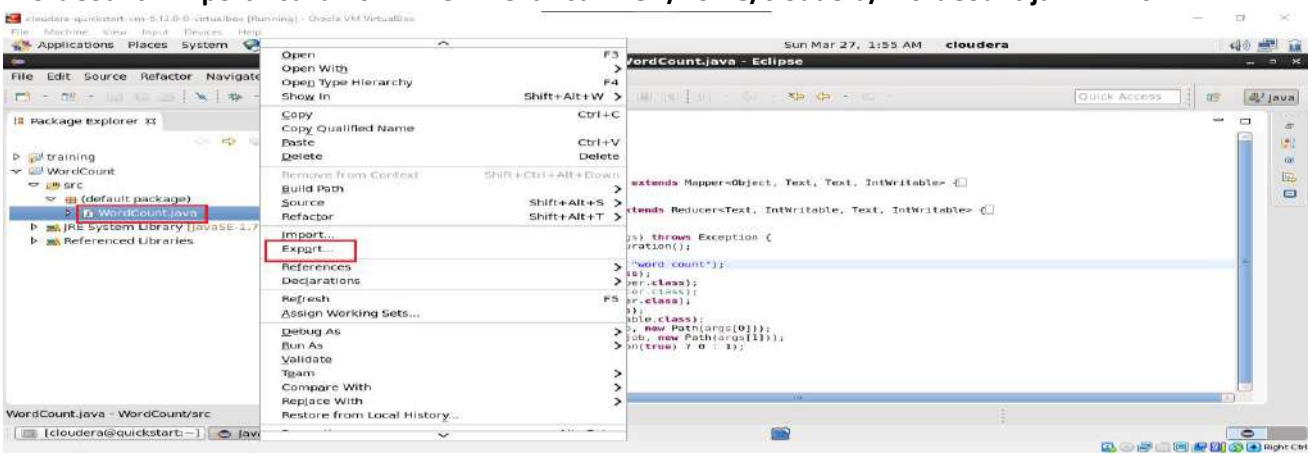
```
    //Map Logic
    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    //Reducer
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }

    }

    //Main Function
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
      //job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

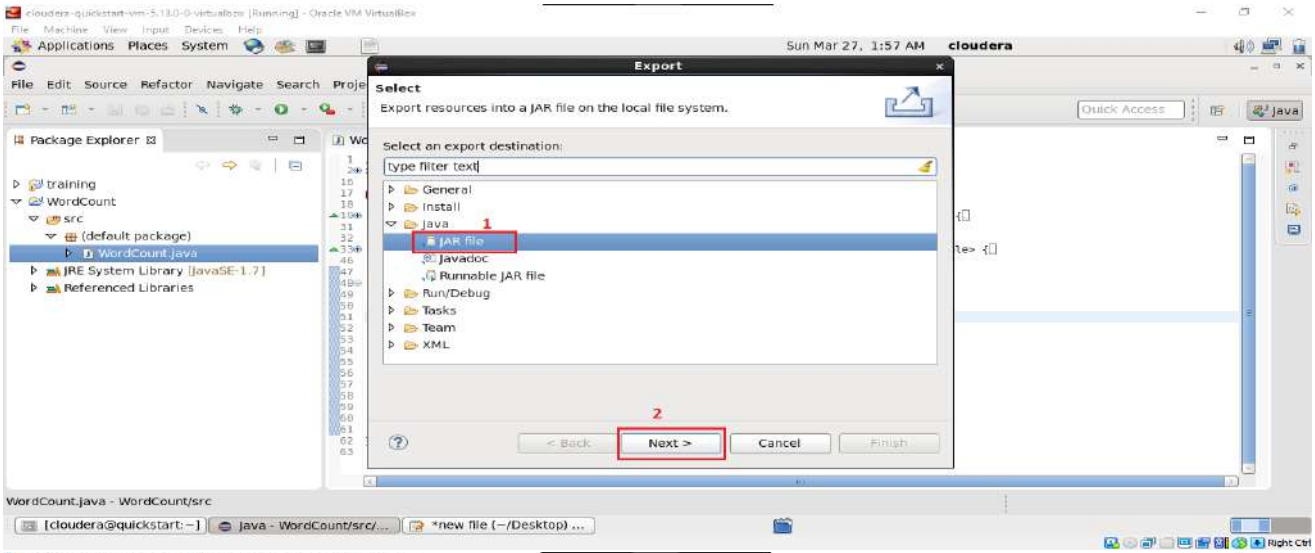
```java
//Packages
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

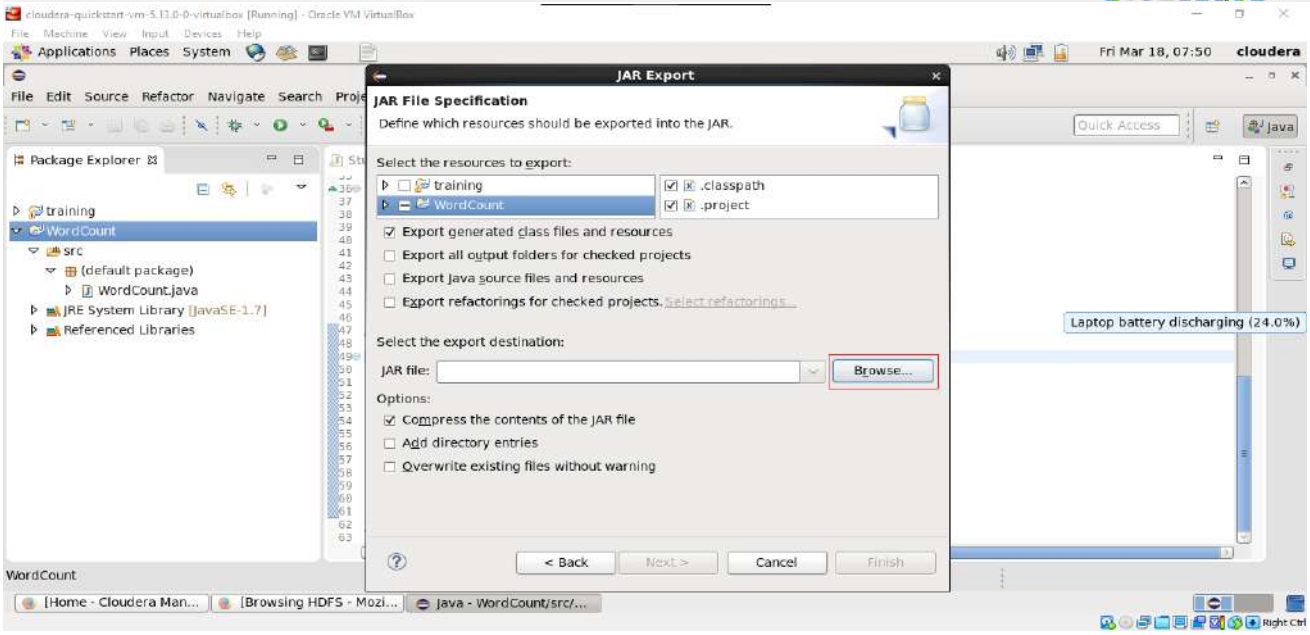
```java
  //Map Logic
  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
          StringTokenizer itr = new StringTokenizer(value.toString());
          while (itr.hasMoreTokens()) {
              word.set(itr.nextToken());
              context.write(word, one);
          }
      }
  }

  //Reducer
  public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
      private IntWritable result = new IntWritable();

      public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException, InterruptedException {
          int sum = 0;
          for (IntWritable val : values) {
              sum += val.get();
          }
          result.set(sum);
          context.write(key, result);
      }

  }

    //Main Function
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        //job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

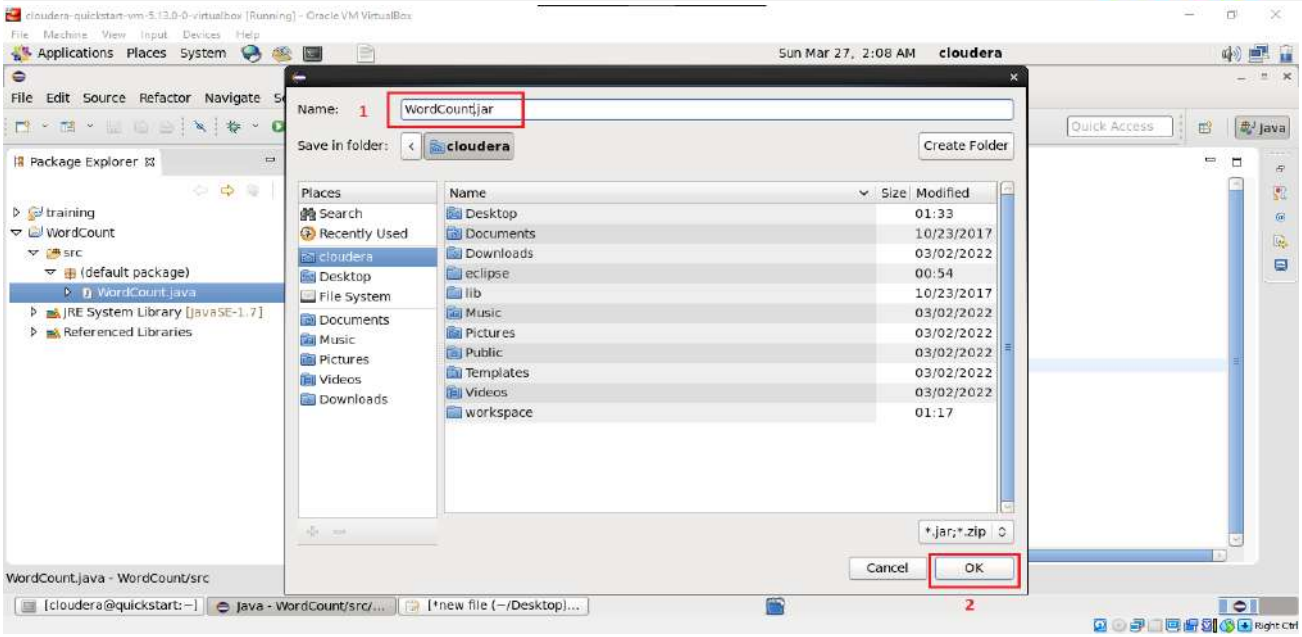
Note: We are running the code **without combiner**. That is why we commented the combiner line **in main function**.

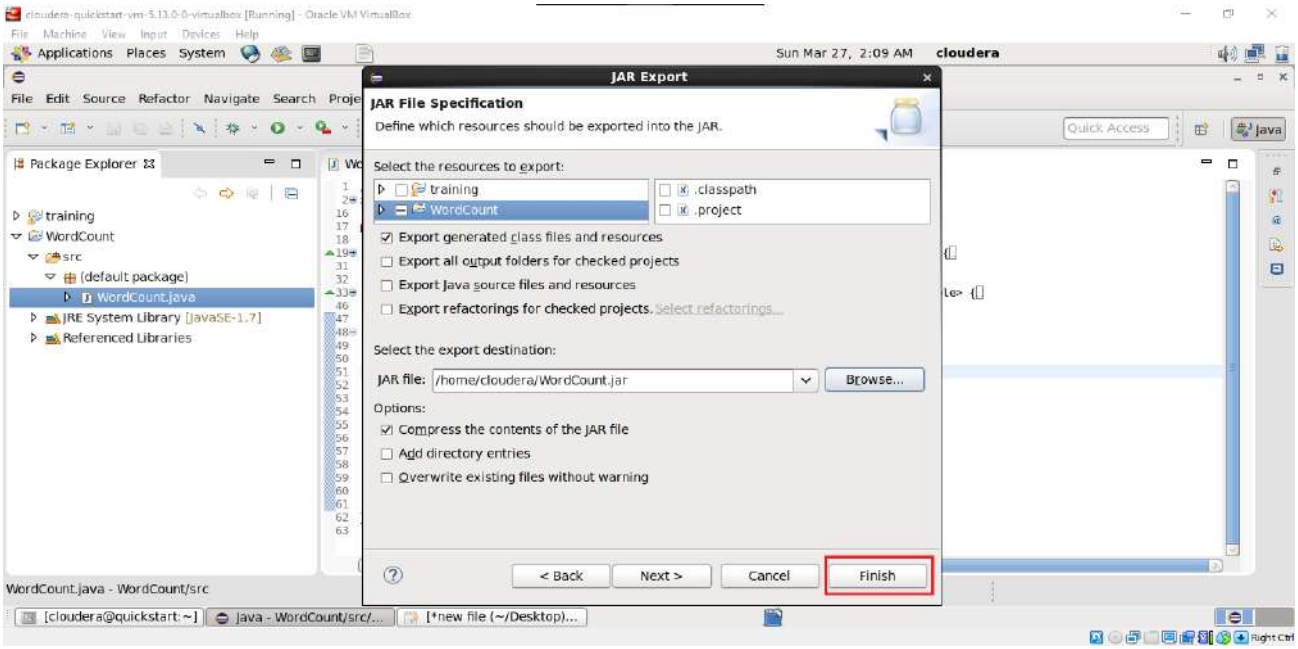
7. **Right click** on the **project name**
   WordCount > Export > Java > JAR File > Next > Jar file "/home/cloudera/WordCount.jar"> Finish.




Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Run/Debug
Tasks
Team
XML
1
2
< Back
Next >
Cancel
Finish


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
training
WordCount
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects.
Select the export destination:
JAR file:
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish


Name:
1
WordCount.jar
Save in folder:
cloudera
Create Folder
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
Desktop 01:33
Documents 10/23/2017
Downloads 03/02/2022
eclipse 00:54
lib 10/23/2017
Music 03/02/2022
Pictures 03/02/2022
Public 03/02/2022
Templates 03/02/2022
Videos 03/02/2022
workspace 01:17
.jar;.zip
Cancel
OK
2



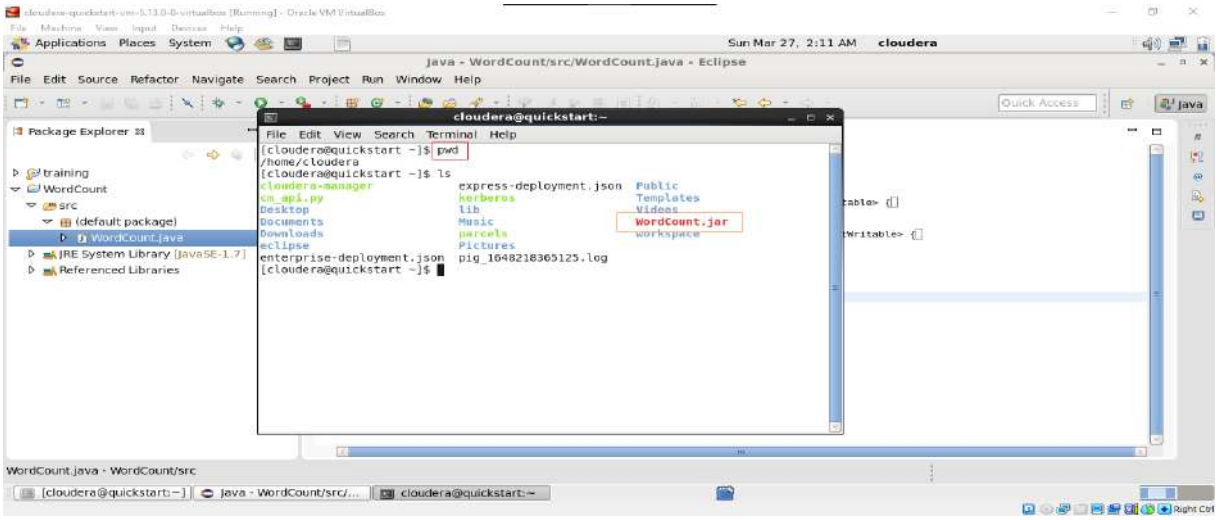
Verify jar file from terminal by using Open terminal & type " **ls** " There it will show WordCount.jar

Check current working directory

pwd

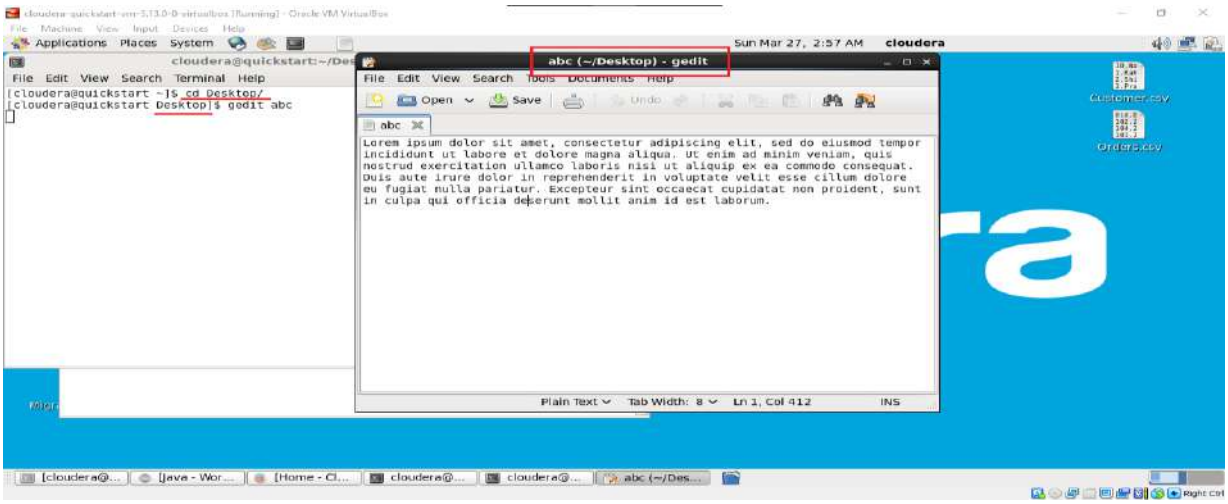
Check the list

ls



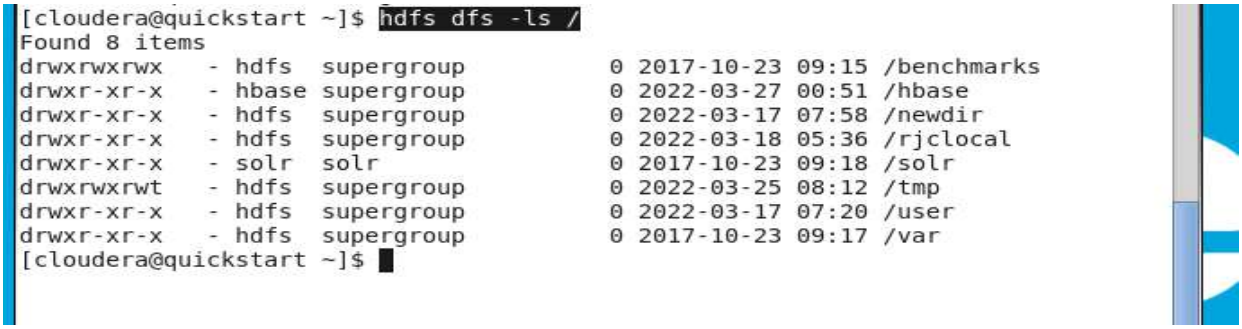
8. We need to create an **input file** in local file system (On Desktop)

Creating an input file named as "**abc**".



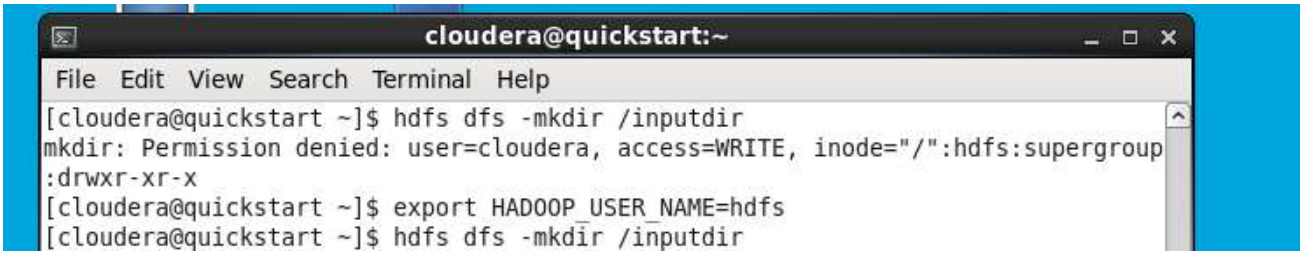
Here listing all the directory present in

hdfs using hdfs dfs -ls /



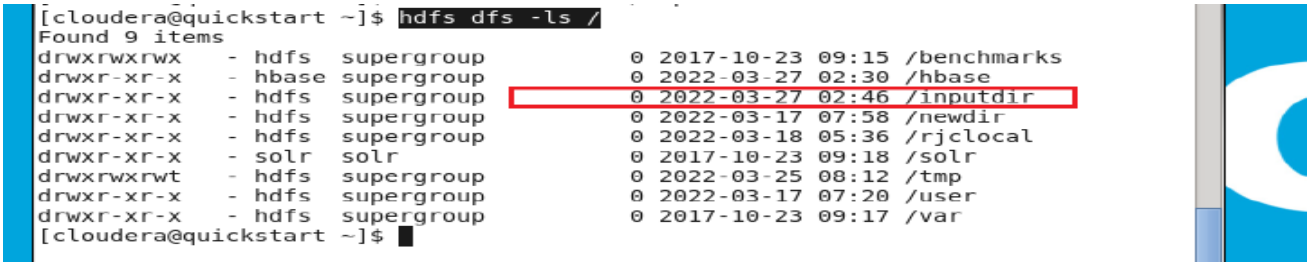
9. Now we have to move this input file to hdfs. For this we create a direcory on hdfs using command

hdfs dfs -mkdir /inputdir



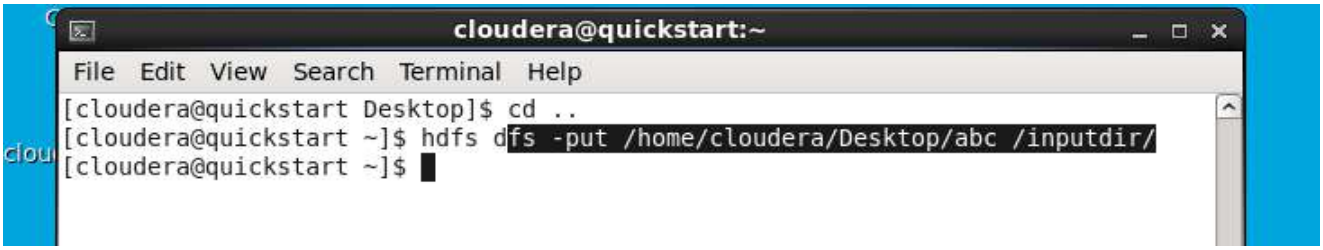
Then we can verify whether this directory is created or not using ls command

hdfs dfs -ls /



Move the input file to this directory created in hdfs by using either **put** command or **copyFromLocal** command.

hdfs dfs -put /home/cloudera/Desktop/abc /inputdir/



Now checking whether the "**abc**" present in **/inputdir** directory of hdfs or not using command

hdfs dfs –ls /inputdir

```
[cloudera@quickstart ~]$ hdfs dfs -ls /inputdir/
Found 1 items
-rw-r--r--   1 hdfs supergroup        446 2022-03-27 02:55 /inputdir/abc
[cloudera@quickstart ~]$
```

As we can see "**abc**" file is present in **/inputdir** directory of hdfs. Now we will see the content of this file using command

hdfs dfs –cat /inputdir/abc

```
[cloudera@quickstart ~]$ hdfs dfs -cat /inputdir/abc
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
 officia deserunt mollit anim id est laborum.
[cloudera@quickstart ~]$
```

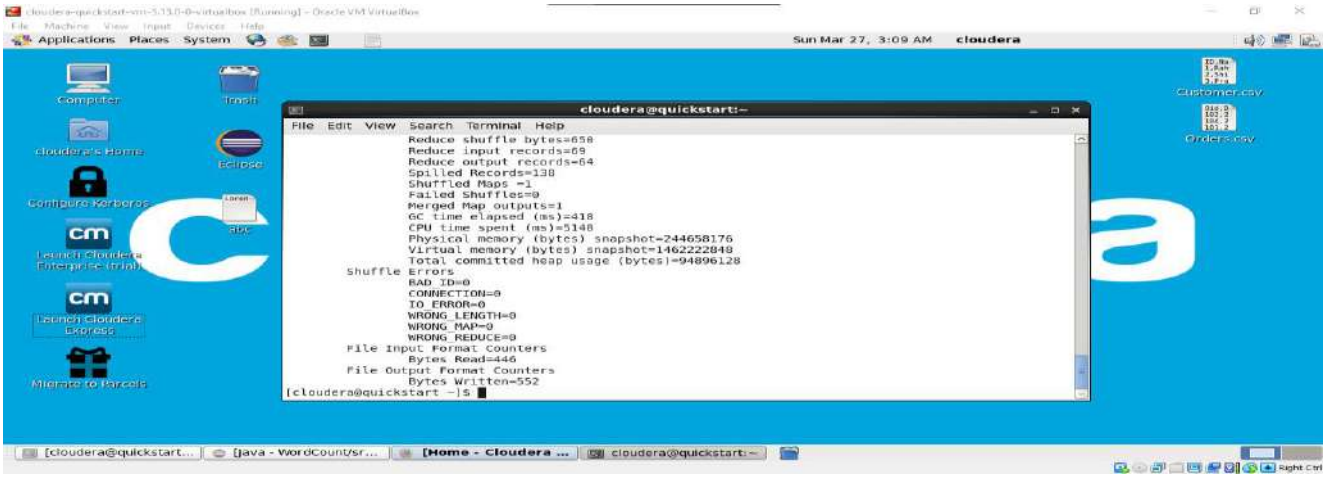
10. Running Mapreduce Program on Hadoop,

Syntax : **hadoop jar jarFileName.jar ClassName /InputFileAddress /outputdir**

hadoop jar /home/cloudera/WordCount.jar WordCount /inputdir/abc /outputdir

```
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/WordCount.jar WordCount /inputdir/abc /outputdir
22/03/27 03:05:46 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:
8032
```

Map-Reduce Framework



As we can see in the above output,

Combine input records=0

Combine output records=0

Note: We are getting this because we have **commented the Combiner line in main function.**

And Reduce shuffle bytes coming as,

Reduce shuffle bytes=1876

So when we are not using combiner 1876 bytes acting as an input for the reducer.

11. Then we can verify the content of tempop1 directory and in that part-r file has the actual output by using the command Hdfs dfs -cat /tempop1/part-r-00000 This will give us final output. The same file can also be accessed using a browser. For every execution of this program we need to delete the output directory or give a new name to the output directory every time. 1st we are checking whether the tempop1 directory is created in hdfs or not using command

hdfs dfs -ls /

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 10 items
drwxrwxrwx   - hdfs   supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase  supergroup          0 2022-03-27 02:30 /hbase
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 02:55 /inputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:58 /newdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 03:08 /outputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-18 05:36 /rjclocal
drwxr-xr-x   - solr   solr                0 2017-10-23 09:18 /solr
drwxrwxrwt   - hdfs   supergroup          0 2022-03-25 08:12 /tmp
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:20 /user
drwxr-xr-x   - hdfs   supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart ~]$
```

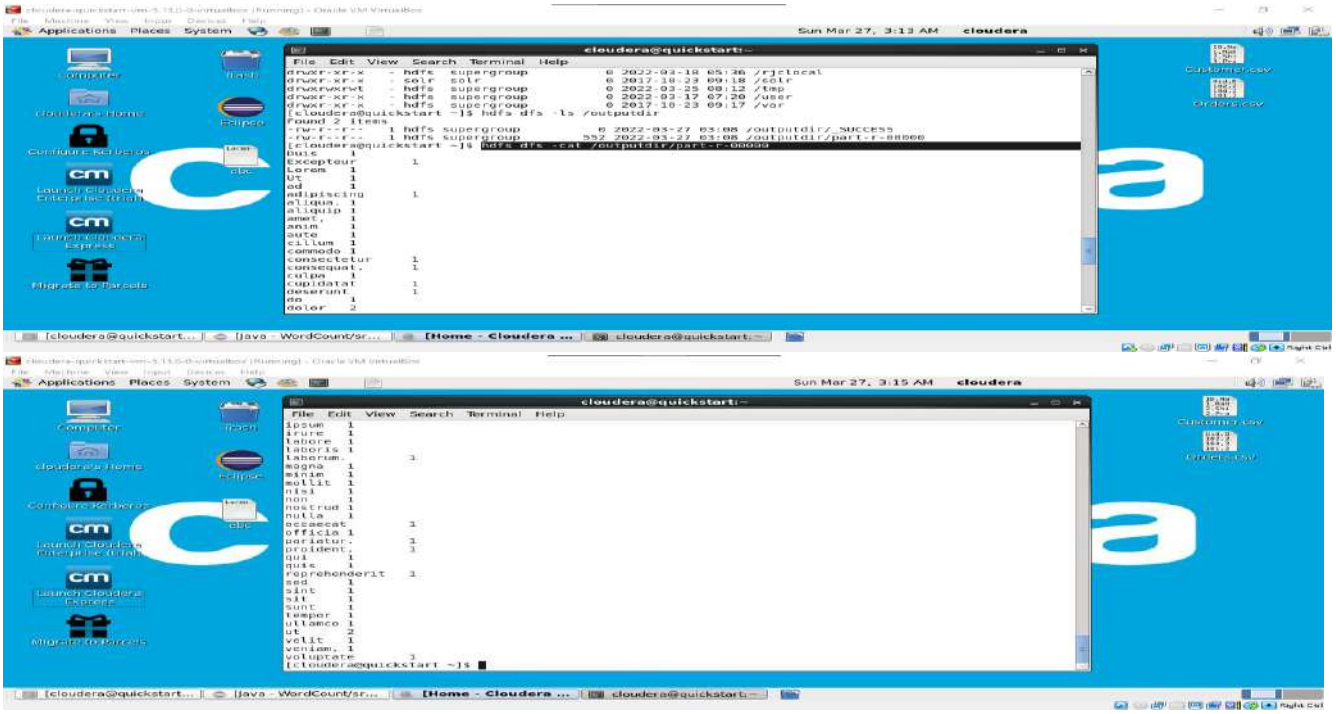
Now let's check what we have inside this **outputdir** directory using command as

hdfs dfs –ls /outputdir

```
[cloudera@quickstart ~]$ hdfs dfs -ls /outputdir
Found 2 items
-rw-r--r--   1 hdfs supergroup          0 2022-03-27 03:08 /outputdir/_SUCCESS
-rw-r--r--   1 hdfs supergroup        552 2022-03-27 03:08 /outputdir/part-r-00000
[cloudera@quickstart ~]$
```

Now we want to read the content of the part-r-00000 file which present inside the outputdir using command

hdfs dfs -cat /outputdir/part-r-00000

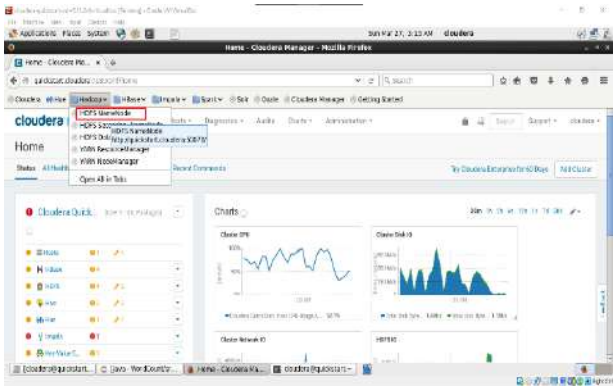


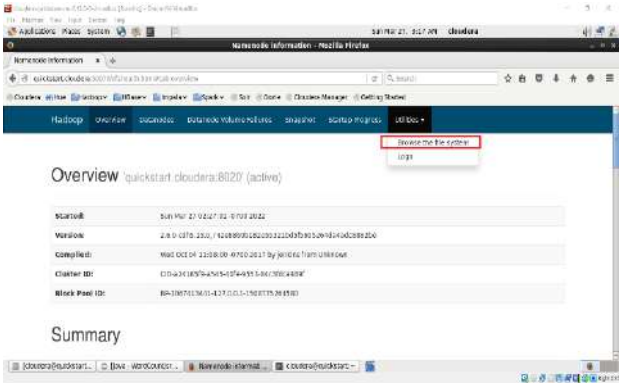
It will give the count of number of times each word has occurred as output.

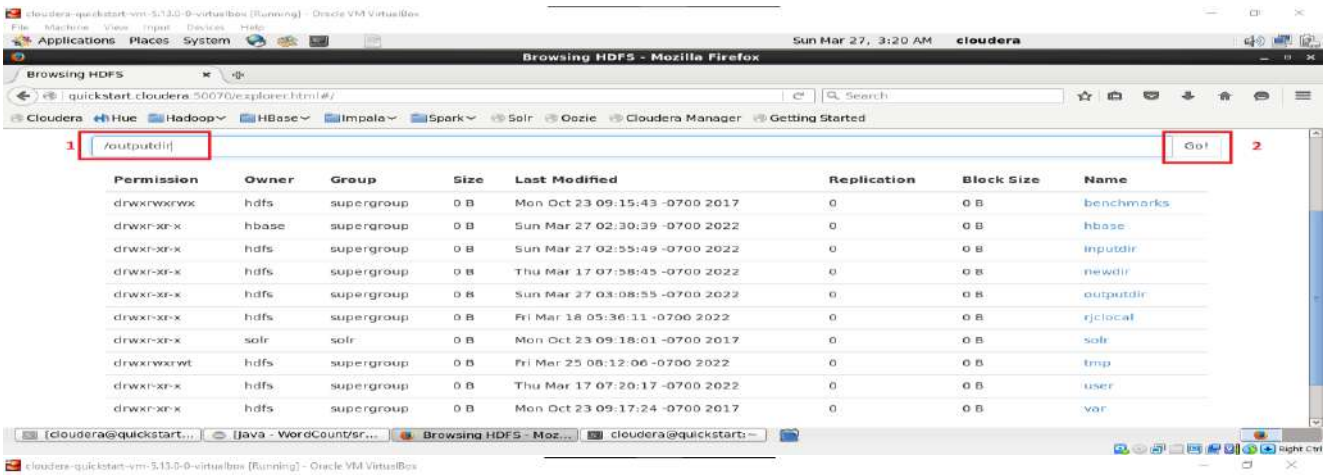
12. The same file can also be accessed using a browser.

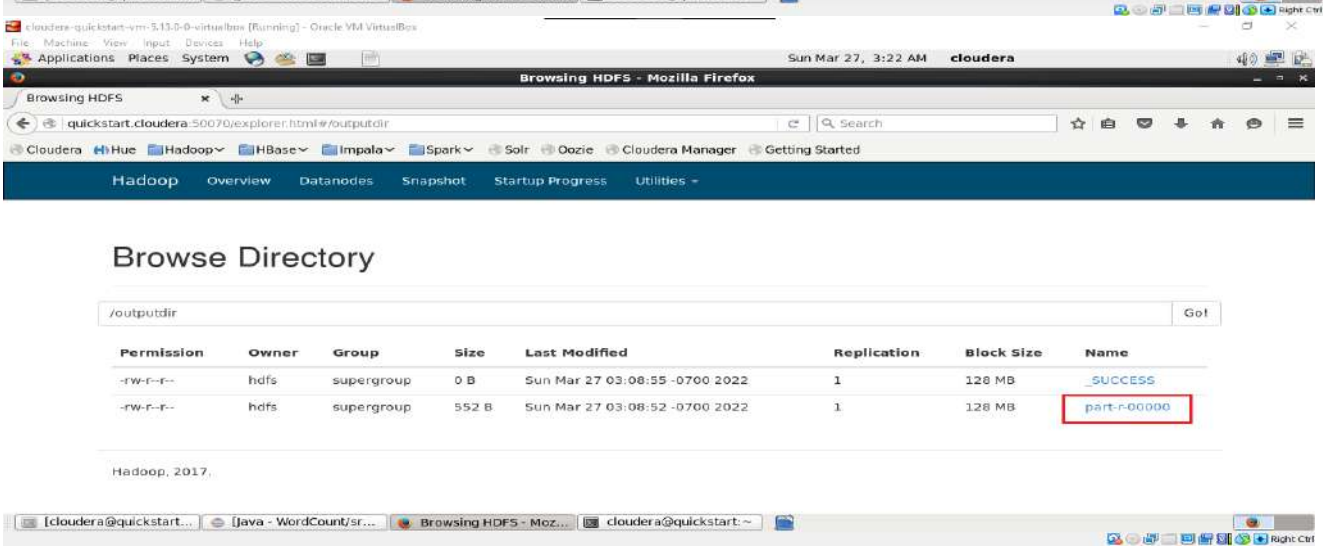
Browse the Directory by

Hadoop->HDFS Namenode->Ultilities ->Browse the file system

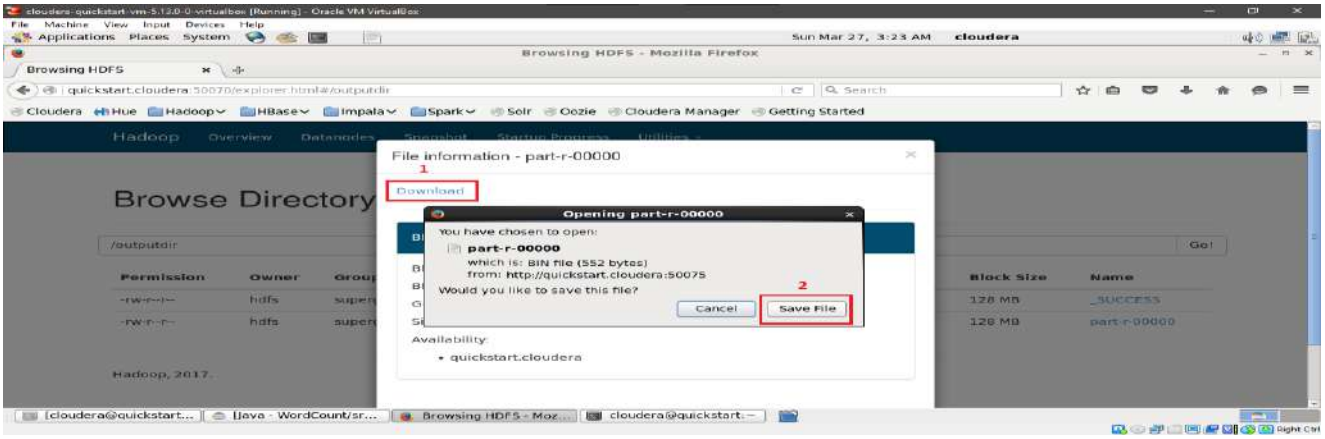


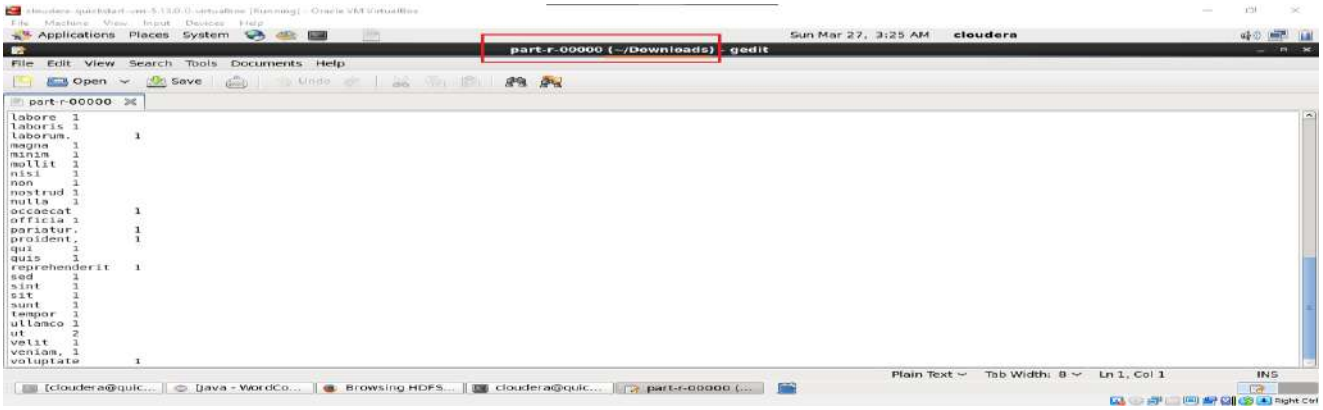






Now **downloading the part-r-00000 file.**

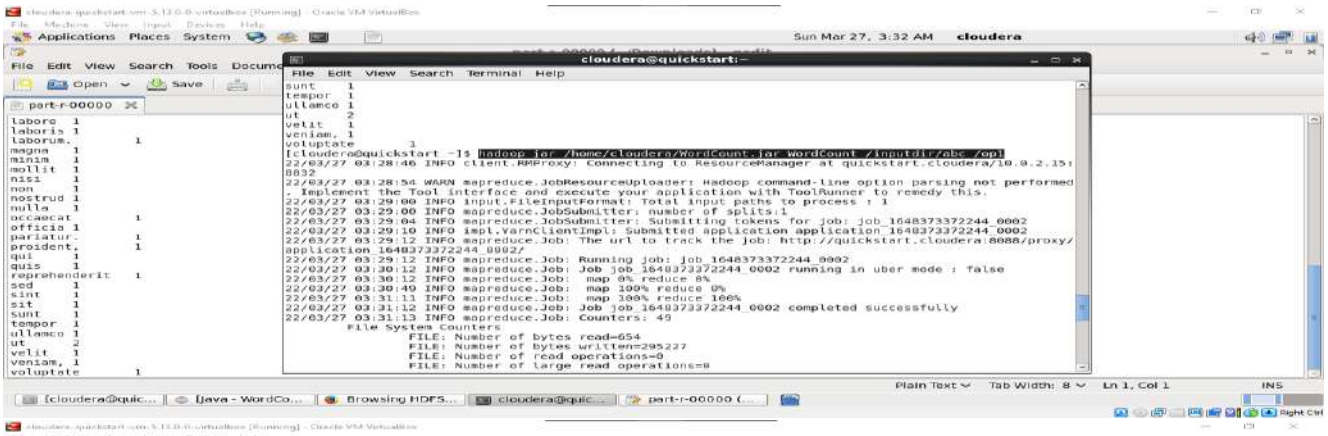


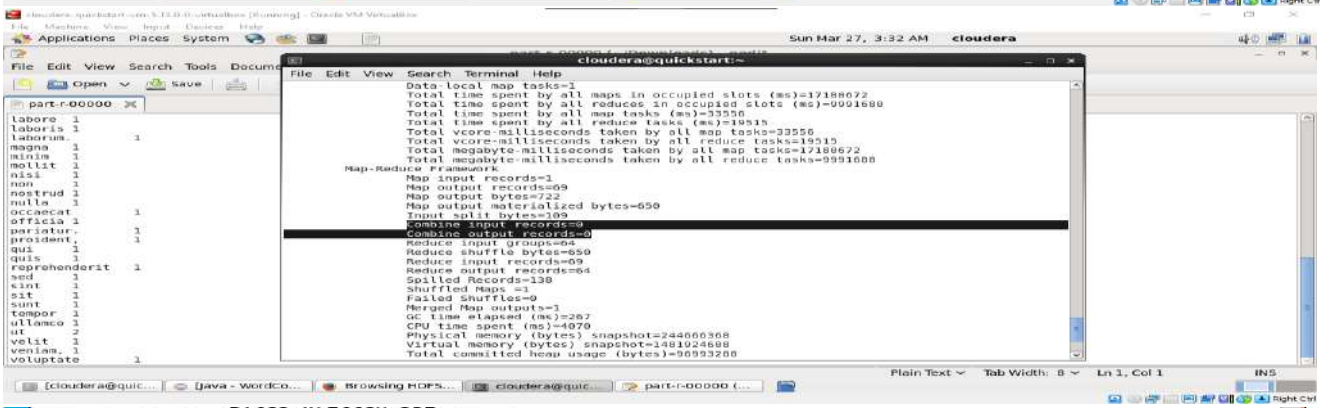


For every execution of this program we need to delete the output directory or give a new name to the output directory every time.

Inside the part-r-00000 file it will have the same output as we are getting after executing using command

hadoop jar /home/cloudera/WordCount.jar WordCount /inputdir/abc /op1






```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 11 items
drwxrwxrwx   - hdfs   supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase  supergroup          0 2022-03-27 02:30 /hbase
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 02:55 /inputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:58 /newdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 03:31 /op1
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 03:08 /outputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-18 05:36 /rjclocal
drwxr-xr-x   - solr   solr                0 2017-10-23 09:18 /solr
drwxrwxrwt   - hdfs   supergroup          0 2022-03-25 08:12 /tmp
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:20 /user
drwxr-xr-x   - hdfs   supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart ~]$ hdfs dfs -ls /op1
Found 2 items
-rw-r--r--   1 hdfs supergroup          0 2022-03-27 03:31 /op1/_SUCCESS
-rw-r--r--   1 hdfs supergroup        552 2022-03-27 03:31 /op1/part-r-00000
[cloudera@quickstart ~]$
```

Aim: To determine maximum temperature using Hadoop MapReduce

Dataset – temperature.txt

0067011990999991950051507004+68750+023550FM-12+038299999V0203301N00671220001CN9999999N9+00001+99999999999

0043011990999991950051512004+68750+023550FM-12+038299999V0203201N00671220001CN9999999N9+00221+99999999999

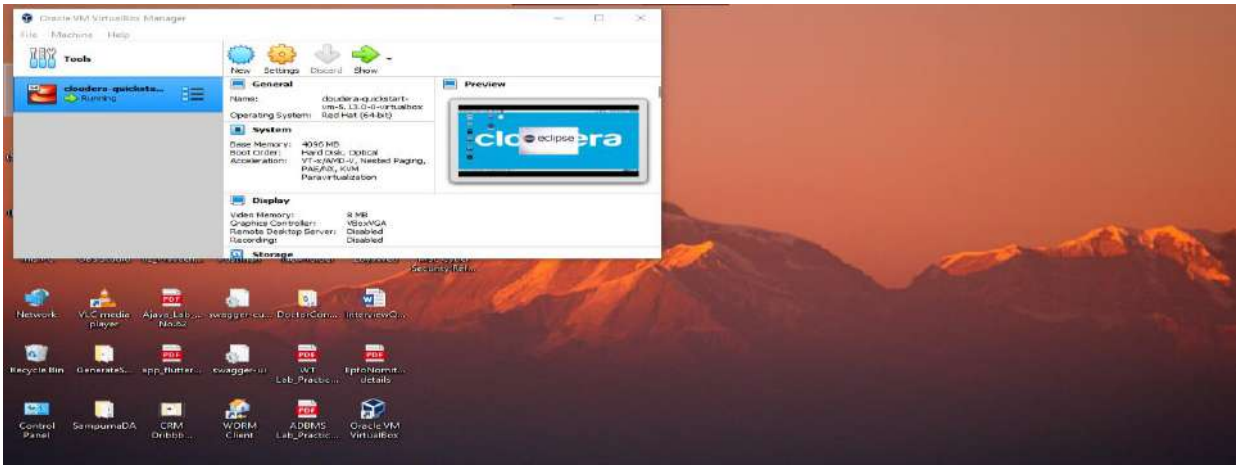
0043011990999991950051518004+68750+023550FM-12+038299999V0203201N00261220001CN9999999N9-00111+99999999999

0043012650999991949032412004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+01111+99999999999

0043012650999991949032418004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+00781+99999999999

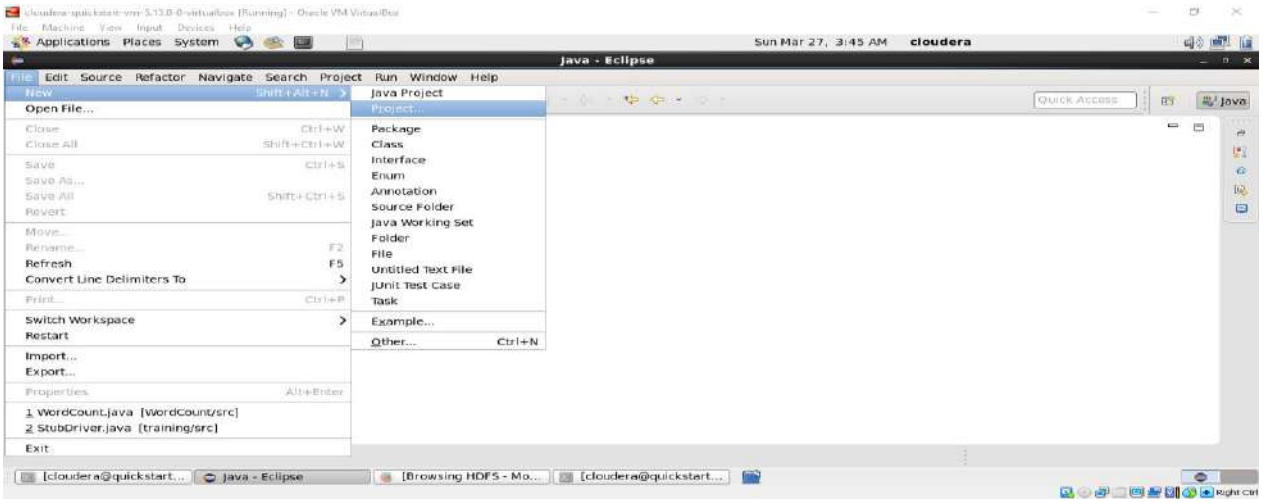
Steps to determine maximum temperature using Hadoop MapReduce in Cloudera:

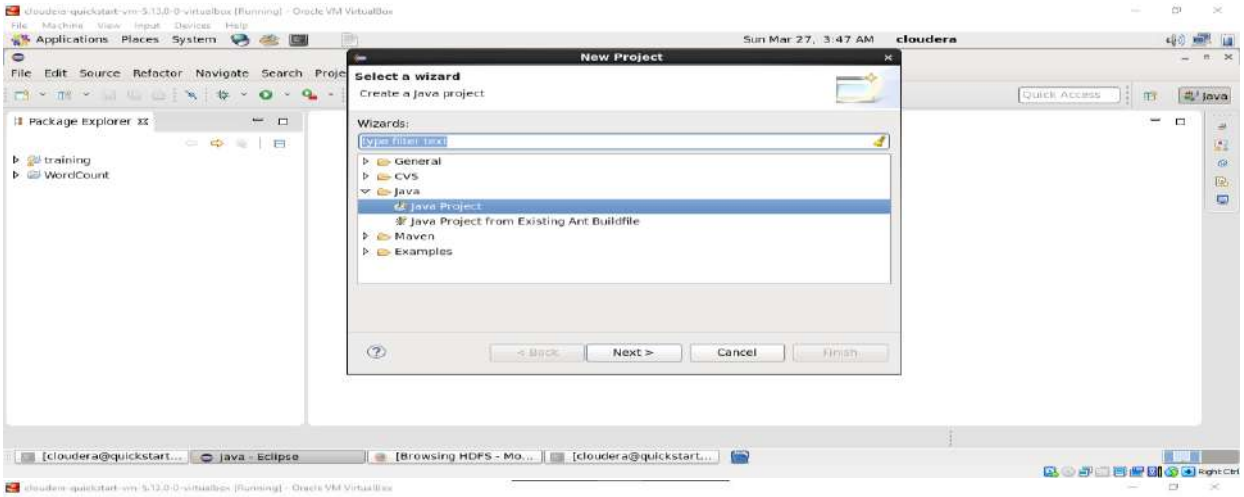
1. Open VirtualBox and then start Cloudera VM

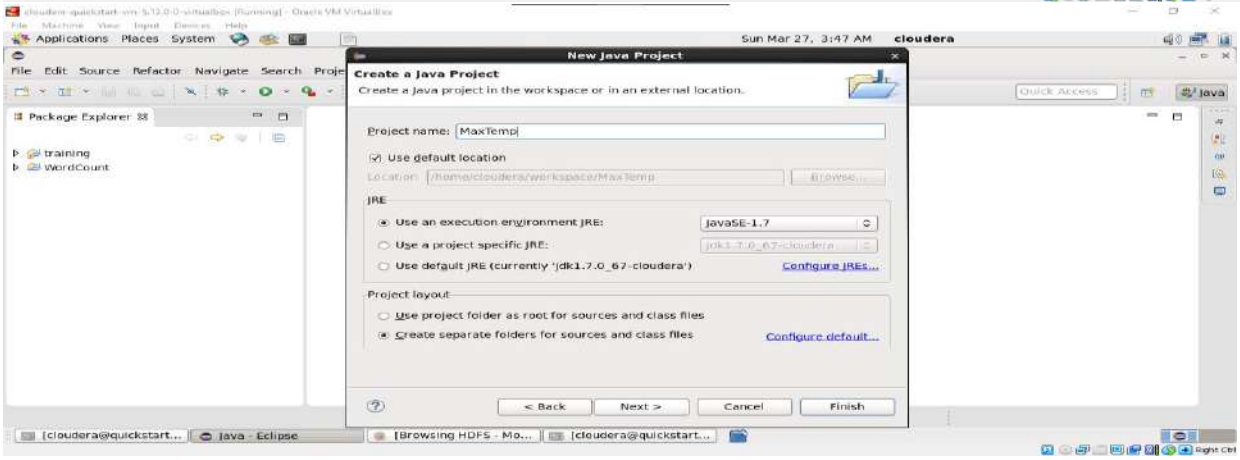


2. Open Eclipse

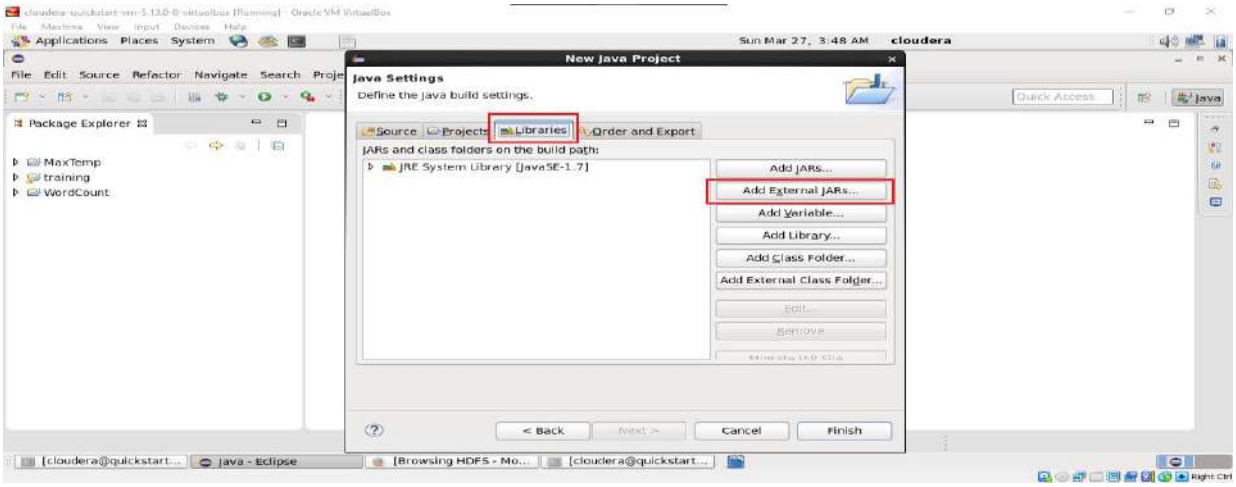
3. Create a **New Java Project**: File > New > Project > Java Project > Next Set project name as "**MaxTemp**"".

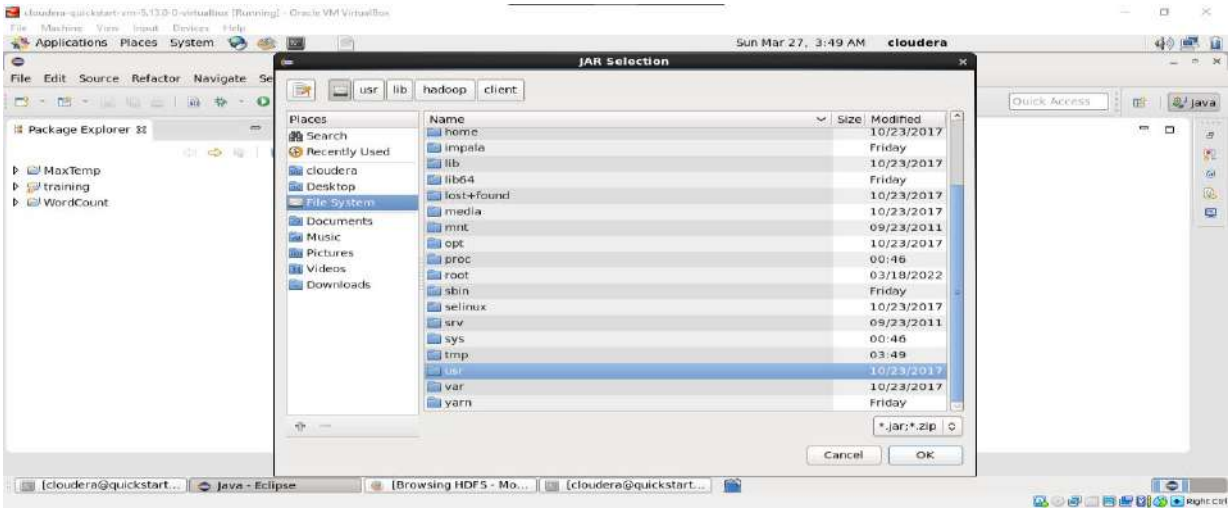


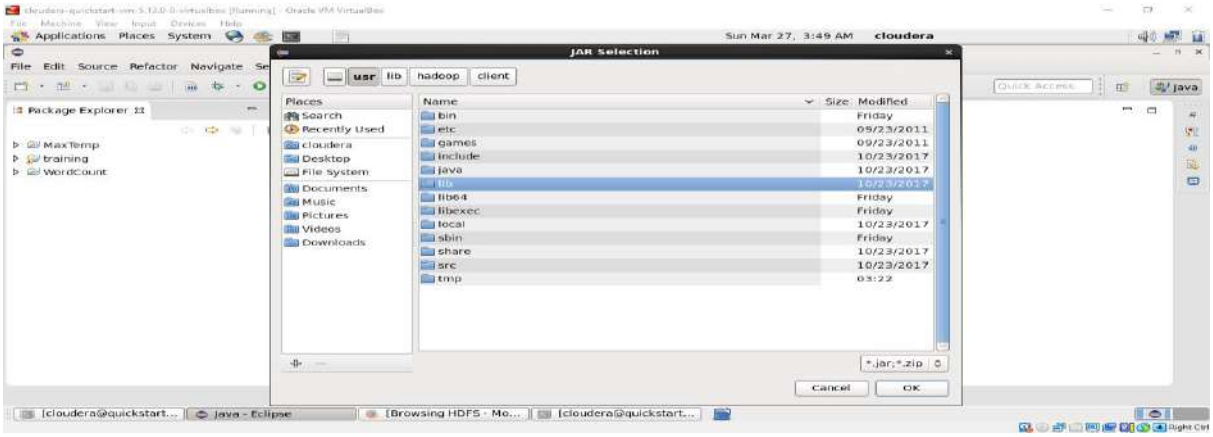




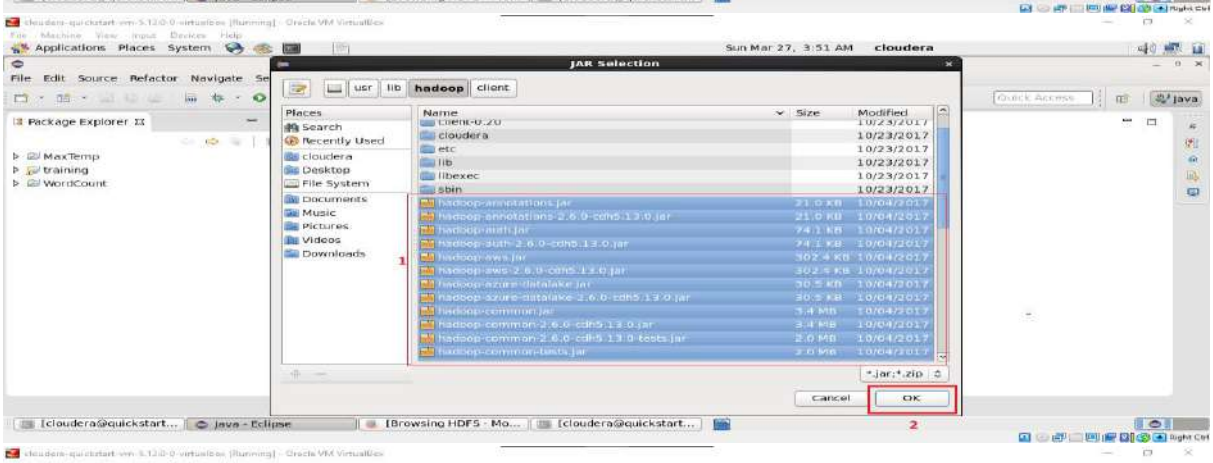
4. Adding the **hadoop libraries** to the project click on:
Libraries > Add External JARs > File System > usr > lib > hadoop > client > select all JAR files > OK > Finish.

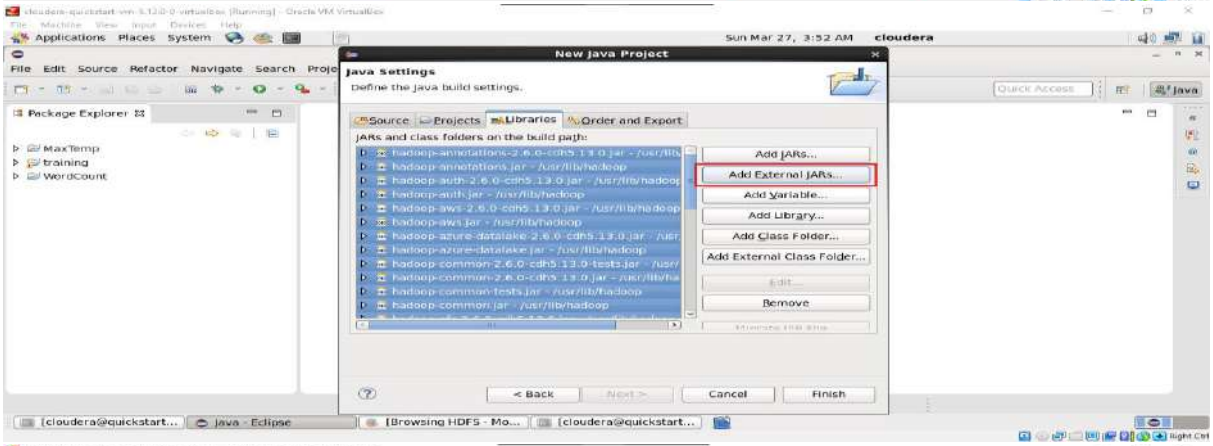


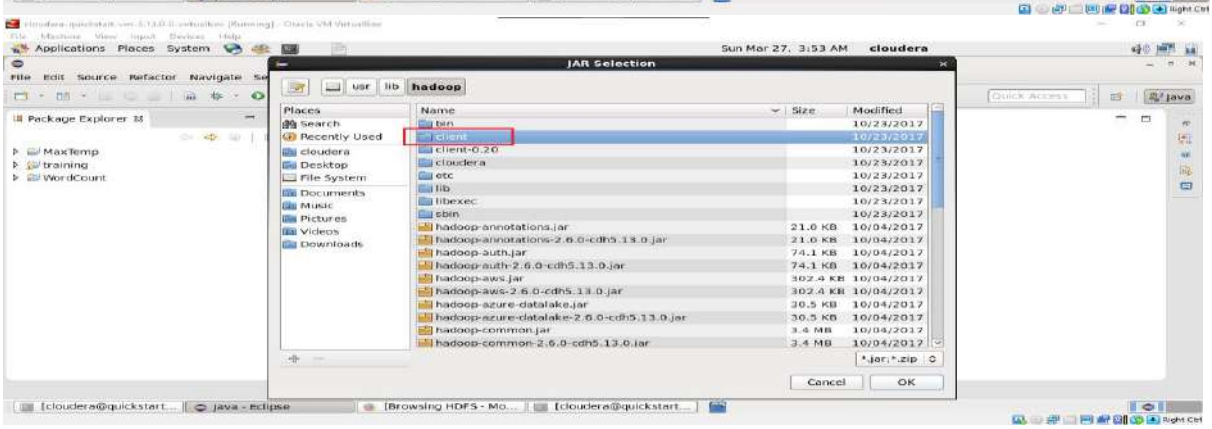


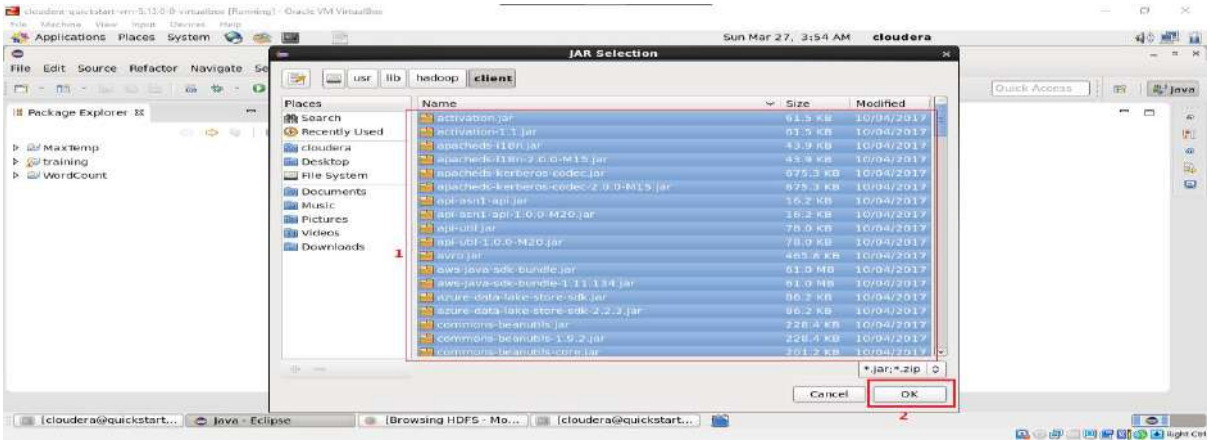

JAR Selection
usr lib hadoop client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name Size Modified
bin Friday
etc 09/23/2011
games 09/23/2011
include 10/23/2017
java 10/23/2017
lib 10/23/2017
lib64 Friday
libexec Friday
local 10/23/2017
sbin Friday
share 10/23/2017
src 10/23/2017
tmp 03:22
.jar;.zip
Cancel
OK

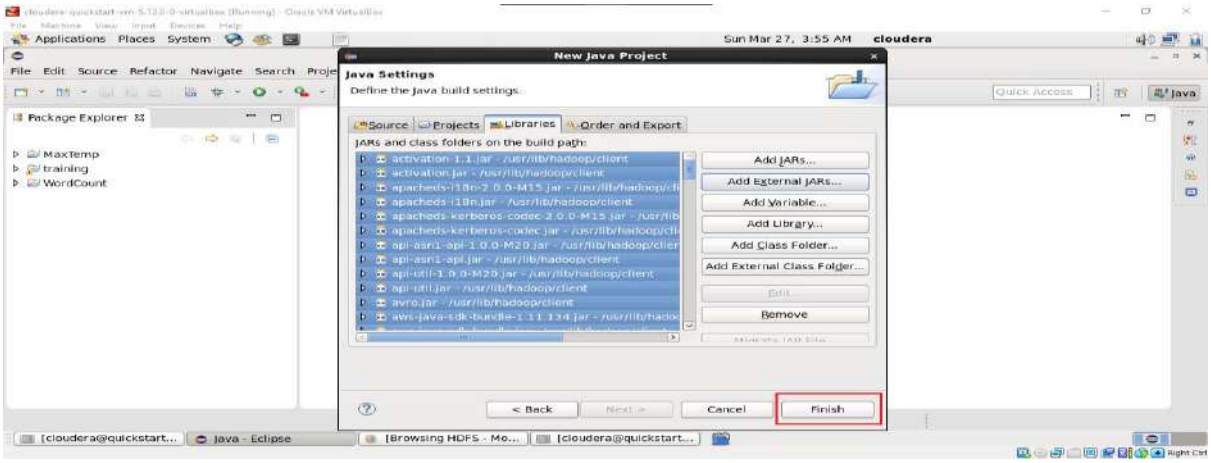

JAR Selection
usr lib hadoop client
Name Size Modified
anaconda-runtime 10/23/2017
avro 10/23/2017
bigtop-tomcat 10/23/2017
bigtop-utils Friday
bonobo 10/23/2017
consolekit 10/23/2017
crunch 10/23/2017
cups 10/23/2017
dkms 10/23/2017
flume-ng 10/23/2017
games 09/23/2011
gcc 07/22/2015
hadoop 10/23/2017
hadoop-0.20-mapreduce 10/23/2017
hadoop-hdfs 10/23/2017
hadoop-httpfs 10/23/2017
hadoop-kms 10/23/2017
hadoop-mapreduce 10/23/2017
.jar;.zip
Cancel
OK


JAR Selection
usr lib hadoop client
Name Size Modified
client-0.20 10/23/2017
cloudera 10/23/2017
etc 10/23/2017
lib 10/23/2017
libexec 10/23/2017
sbin 10/23/2017
hadoop-annotations.jar 21.0 KB 10/04/2017
hadoop-annotations-2.6.0-cdh5.13.0.jar 21.0 KB 10/04/2017
hadoop-auth.jar 74.1 KB 10/04/2017
hadoop-auth-2.6.0-cdh5.13.0.jar 74.1 KB 10/04/2017
hadoop-aws.jar 302.4 KB 10/04/2017
hadoop-aws-2.6.0-cdh5.13.0.jar 302.4 KB 10/04/2017
hadoop-azure-datalake.jar 30.5 KB 10/04/2017
hadoop-azure-datalake-2.6.0-cdh5.13.0.jar 30.5 KB 10/04/2017
hadoop-common.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0-tests.jar 2.0 MB 10/04/2017
hadoop-common-tests.jar 2.0 MB 10/04/2017
1
.jar;.zip
Cancel
OK
2

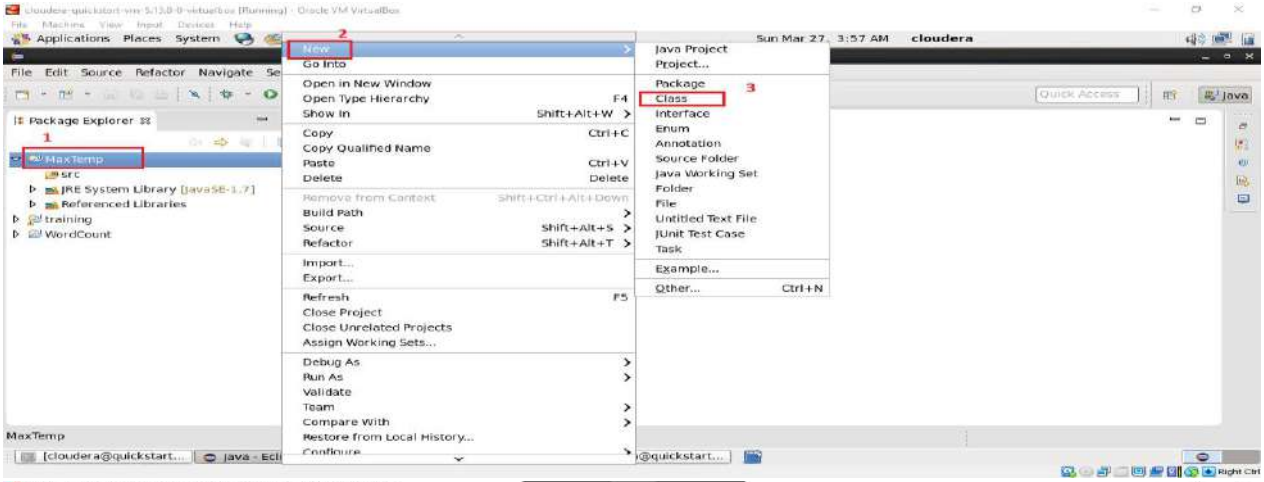

New Java Project
Java Settings
Define the Java build settings.
Source Projects Libraries Order and Export
JARs and class folders on the build path:
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
< Back
Next >
Cancel
Finish

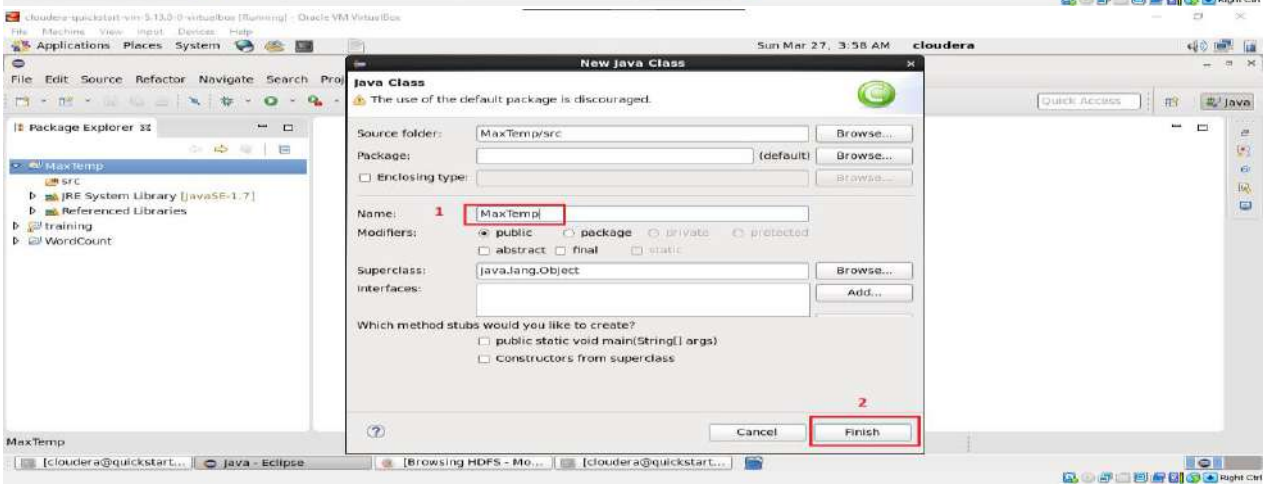

JAR Selection
usr lib hadoop
Name Size Modified
bin 10/23/2017
client 10/23/2017
client-0.20 10/23/2017
cloudera 10/23/2017
etc 10/23/2017
lib 10/23/2017
libexec 10/23/2017
sbin 10/23/2017
hadoop-annotations.jar 21.0 KB 10/04/2017
hadoop-annotations-2.6.0-cdh5.13.0.jar 21.0 KB 10/04/2017
hadoop-auth.jar 74.1 KB 10/04/2017
hadoop-auth-2.6.0-cdh5.13.0.jar 74.1 KB 10/04/2017
hadoop-aws.jar 302.4 KB 10/04/2017
hadoop-aws-2.6.0-cdh5.13.0.jar 302.4 KB 10/04/2017
hadoop-azure-datalake.jar 30.5 KB 10/04/2017
hadoop-azure-datalake-2.6.0-cdh5.13.0.jar 30.5 KB 10/04/2017
hadoop-common.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0.jar 3.4 MB 10/04/2017
.jar;.zip
Cancel
OK



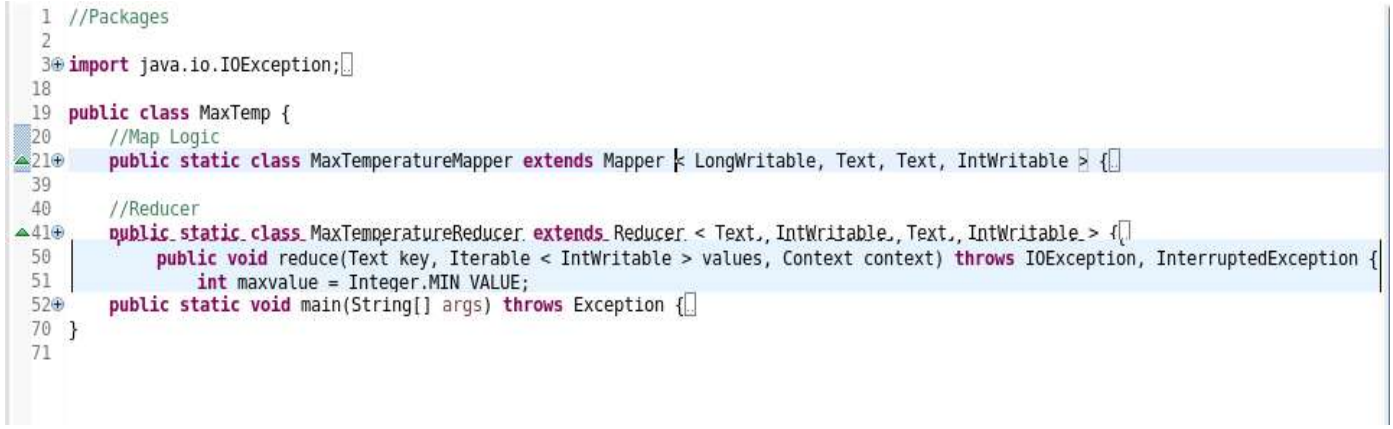


5. **Right click** on the name of project
"**MaxTemp**" > New > Class (Don't write anything for package) > class name "**MaxTemp**" > Finish
Then WordCount.java will open if not then open MaxTemp.java file manually





6. Source Code:



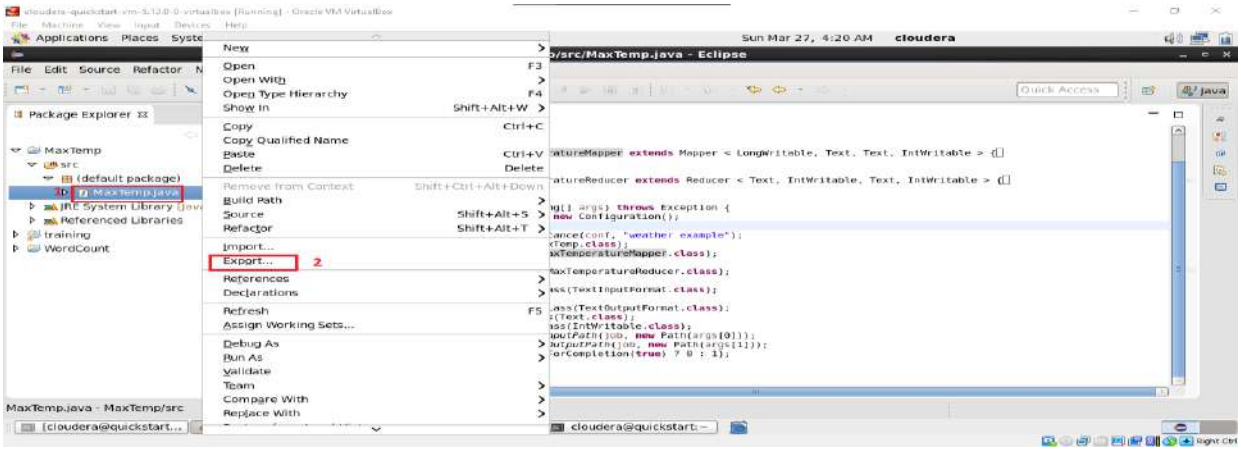

```
 //Packages

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;

public class MaxTemp {
    //Map Logic
    public static class MaxTemperatureMapper extends Mapper < LongWritable, Text, Text,
IntWritable > {

        public static final int MISSING = 9999;

        public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
            String line = value.toString();
            String year = line.substring(15, 19);
            int airtemp;
            if (line.charAt(87) == '+') {
                airtemp = Integer.parseInt(line.substring(88, 92));
            } else
                airtemp = Integer.parseInt(line.substring(87, 92));
            String q = line.substring(92, 93);
            if (airtemp != MISSING && q.matches("[01459]")) {
                context.write(new Text(year), new IntWritable(airtemp));
            }
        }
    }

    //Reducer
    public static class MaxTemperatureReducer extends Reducer < Text, IntWritable, Text,
IntWritable > {
```

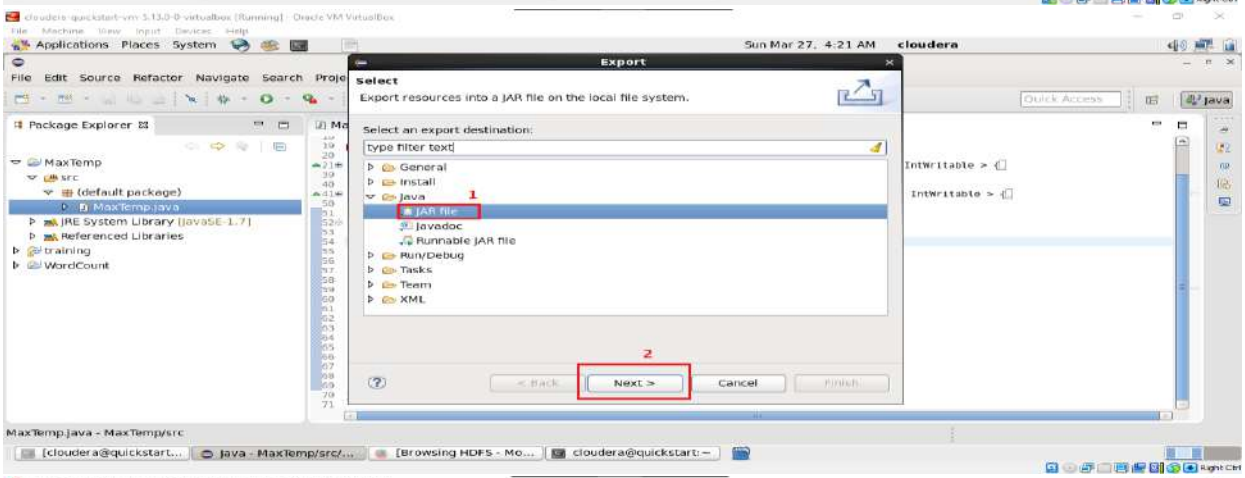
```
        public void reduce(Text key, Iterable < IntWritable > values, Context context) throws
IOException, InterruptedException {
            int maxvalue = Integer.MIN_VALUE;
            for (IntWritable value: values) {
                maxvalue = Math.max(maxvalue, value.get());
            }
            context.write(key, new IntWritable(maxvalue));
        }
    }

    //Main Function
    public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();

            Job job = Job.getInstance(conf, "weather example");
            job.setJarByClass(MaxTemp.class);
            job.setMapperClass(MaxTemperatureMapper.class);

            job.setReducerClass(MaxTemperatureReducer.class);

            job.setInputFormatClass(TextInputFormat.class);

            job.setOutputFormatClass(TextOutputFormat.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

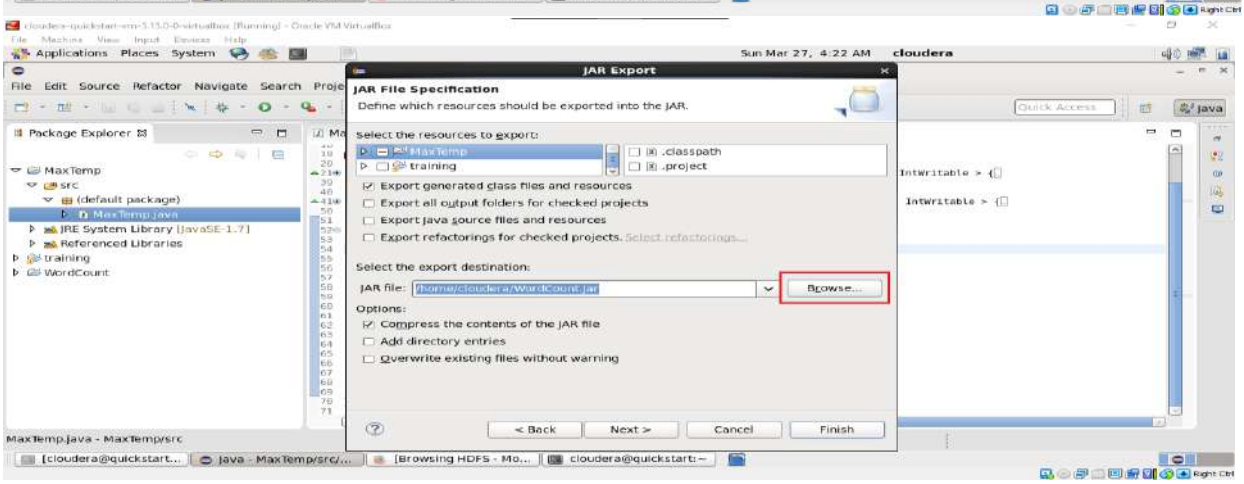
```
1  //Packages
2
3  import java.io.IOException;
4  import org.apache.hadoop.fs.Path;
5  import org.apache.hadoop.io.LongWritable;
6
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.io.IntWritable;
9  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
13
14 import org.apache.hadoop.mapreduce.Job;
15 import org.apache.hadoop.mapreduce.Mapper;
16 import org.apache.hadoop.mapreduce.Reducer;
17 import org.apache.hadoop.conf.Configuration;
18
21     public static class MaxTemperatureMapper extends Mapper < LongWritable, Text, Text, IntWritable > {
22
23         public static final int MISSING = 9999;
24
25         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
26             String line = value.toString();
27             String year = line.substring(15, 19);
28             int airtemp;
29             if (line.charAt(87) == '+') {
30                 airtemp = Integer.parseInt(line.substring(88, 92));
31             } else
32                 airtemp = Integer.parseInt(line.substring(87, 92));
33             String q = line.substring(92, 93);
34             if (airtemp != MISSING && q.matches("[01459]")) {
35                 context.write(new Text(year), new IntWritable(airtemp));
36             }
37         }
38     }
```

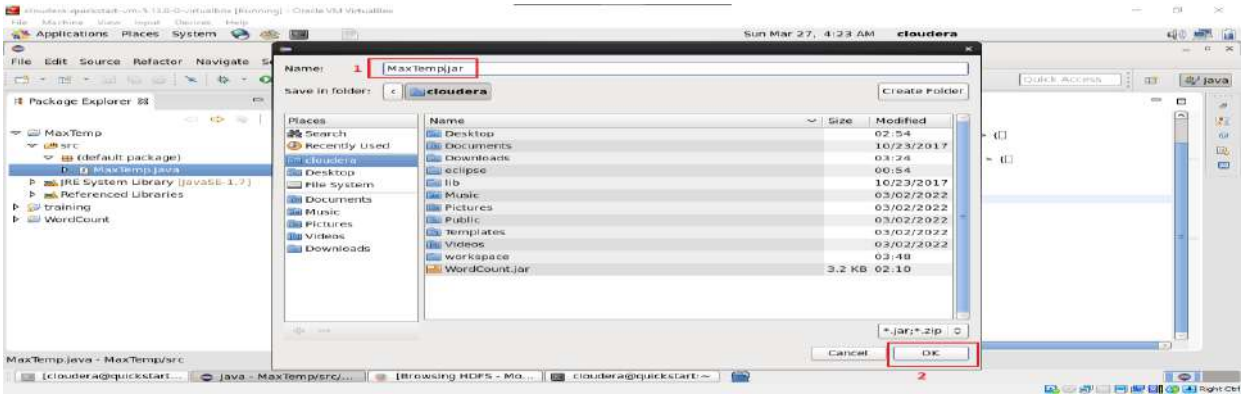
```
    //Reducer
    public static class MaxTemperatureReducer extends Reducer < Text, IntWritable, Text, IntWritable > {
        public void reduce(Text key, Iterable < IntWritable > values, Context context) throws IOException, InterruptedException {
            int maxvalue = Integer.MIN_VALUE;
            for (IntWritable value: values) {
                maxvalue = Math.max(maxvalue, value.get());
            }
            context.write(key, new IntWritable(maxvalue));
        }
    }

    //Main Function
    public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();

            Job job = Job.getInstance(conf, "weather example");
            job.setJarByClass(MaxTemp.class);
            job.setMapperClass(MaxTemperatureMapper.class);

            job.setReducerClass(MaxTemperatureReducer.class);

            job.setInputFormatClass(TextInputFormat.class);

            job.setOutputFormatClass(TextOutputFormat.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
```

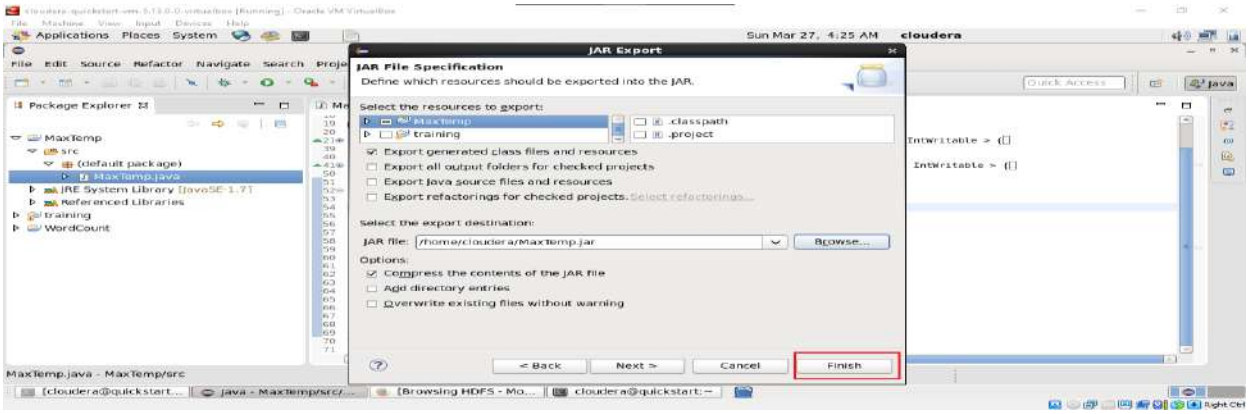
7. **Right click** on the **project name**
   MaxTemp> Export > Java > JAR File > Next > Jar file "/home/cloudera/MaxTemp.jar"> Finish.











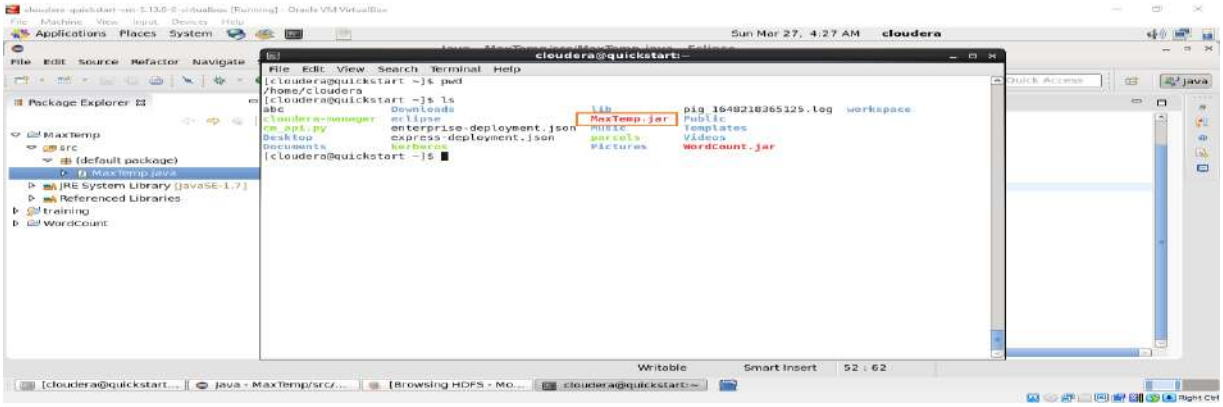
Verify jar file from terminal by using Open terminal & type " **ls** " There it will show MaxTemp.jar

Check current working directory

pwd

Check the list

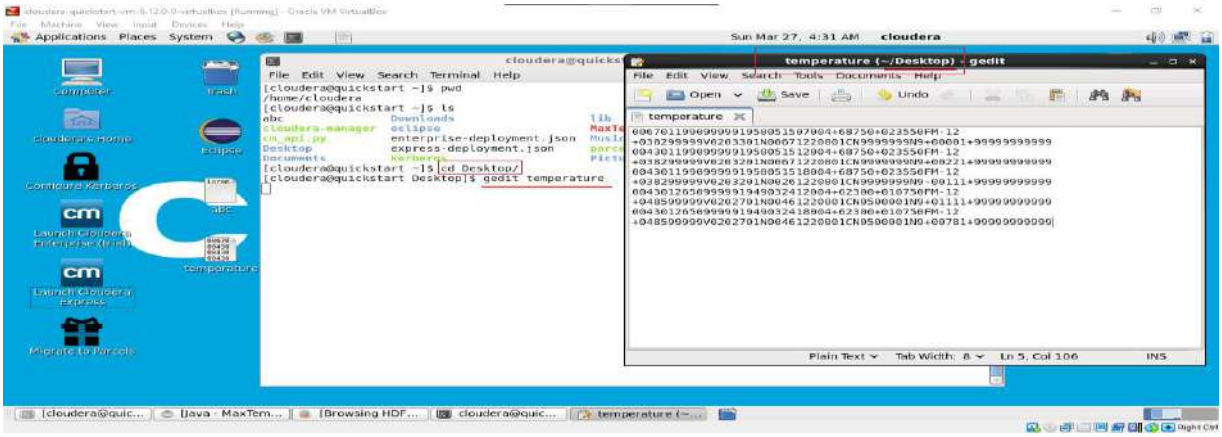
ls



8. We need to <u>create</u> an **input file** in <u>local file system (On Desktop)</u>

Creating an input file named as "**temperature**".

Paste the **DataSet**



Here listing all the directory present in

hdfs using hdfs dfs -ls /

```
[cloudera@quickstart Desktop]$ hdfs dfs -ls /
Found 11 items
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2022-03-27 02:30 /hbase
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 02:55 /inputdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-17 07:58 /newdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 03:31 /op1
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 03:08 /outputdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-18 05:36 /rjclocal
drwxr-xr-x   - solr  solr                0 2017-10-23 09:18 /solr
drwxrwxrwt   - hdfs  supergroup          0 2022-03-25 08:12 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2022-03-17 07:20 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart Desktop]$
```

9. Now we have to move this input file to hdfs. For this we create a direcory on hdfs using command

hdfs dfs -mkdir /tempip

```
[cloudera@quickstart Desktop]$ hdfs dfs -mkdir /tempip
[cloudera@quickstart Desktop]$
```

If getting any permission related error than execute below command

export HADOOP_USER_NAME=hdfs

Then we can verify whether this directory is created or not using ls command

hdfs dfs -ls /

```
[cloudera@quickstart Desktop]$ hdfs dfs -ls /
Found 12 items
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2022-03-27 02:30 /hbase
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 02:55 /inputdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-17 07:58 /newdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 03:31 /op1
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 03:08 /outputdir
drwxr-xr-x   - hdfs  supergroup          0 2022-03-18 05:36 /rjclocal
drwxr-xr-x   - solr  solr                0 2017-10-23 09:18 /solr
drwxr-xr-x   - hdfs  supergroup          0 2022-03-27 04:37 /tempip
drwxrwxrwt   - hdfs  supergroup          0 2022-03-25 08:12 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2022-03-17 07:20 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart Desktop]$
```

Move the input file to this directory created in hdfs by using either **put** command or **copyFromLocal** command

hdfs dfs -put /home/cloudera/Desktop/temperature /tempip/

```
[cloudera@quickstart Desktop]$ hdfs dfs -put /home/cloudera/Desktop/temperature /tempip/
[cloudera@quickstart Desktop]$
```

Now checking whether the "**temperature**" present in /**tempip** directory of hdfs or not using command

hdfs dfs –ls /tempip

```
[cloudera@quickstart Desktop]$ hdfs dfs -ls /tempip
Found 1 items
-rw-r--r--   1 hdfs supergroup        530 2022-03-27 04:37 /tempip/temperature
[cloudera@quickstart Desktop]$
```

As we can see "**temperature**" file is present in / **tempip** directory of hdfs. Now we will see the content of this file using command

hdfs dfs –cat /tempip/temperature

```
[cloudera@quickstart Desktop]$ hdfs dfs -cat /tempip/temperature
0067011990999991950051507004+68750+023550FM-12+038299999V0203301N00671220001CN9999999N9+00001+99999999
999
0043011990999991950051512004+68750+023550FM-12+038299999V0203201N00671220001CN9999999N9+00221+99999999
999
0043011990999991950051518004+68750+023550FM-12+038299999V0203201N00261220001CN9999999N9-00111+99999999
999
0043012650999991949032412004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+01111+99999999
999
0043012650999991949032418004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+00781+99999999
999
[cloudera@quickstart Desktop]$
```

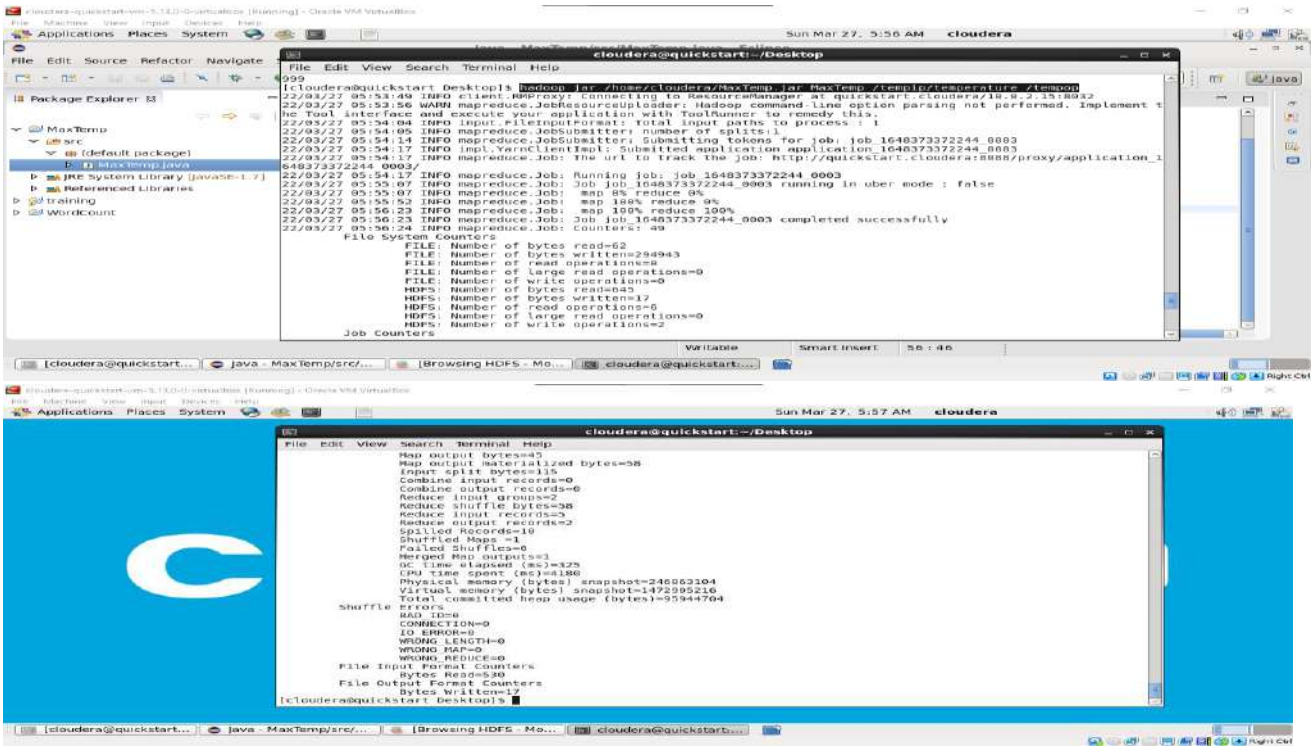
10. Running Mapreduce Program on Hadoop,

Syntax : **hadoop jar jarFileName.jar ClassName /InputFileAddress /outputdir**

hadoop jar /home/cloudera/MaxTemp.jar MaxTemp /tempip/temperature /tempop1

```
[cloudera@quickstart Desktop]$ hadoop jar /home/cloudera/MaxTemp.jar MaxTemp /tempip/temperature /tempop
22/03/27 05:53:49 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
22/03/27 05:53:56 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement t
he Tool interface and execute your application with ToolRunner to remedy this.
```

Map-Reduce Framework



11. Then we can verify the content of outputdir directory and in that part-r file has the actual output by using the command Hdfs dfs -cat /outputdir/part-r-00000 This will give us final output. The same file can also be accessed using a browser. For every execution of this program we need to delete the output directory or give a new name to the output directory every time. 1st we are checking whether the outputdir directory is created in hdfs or not using command

hdfs dfs -ls /

```
[cloudera@quickstart Desktop]$ hdfs dfs -ls /
Found 13 items
drwxrwxrwx   - hdfs   supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase  supergroup          0 2022-03-27 05:48 /hbase
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 02:55 /inputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:58 /newdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 03:31 /op1
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 03:08 /outputdir
drwxr-xr-x   - hdfs   supergroup          0 2022-03-18 05:36 /rjclocal
drwxr-xr-x   - solr   solr                0 2017-10-23 09:18 /solr
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 04:37 /tempip
drwxr-xr-x   - hdfs   supergroup          0 2022-03-27 05:56 /tempop
drwxrwxrwt   - hdfs   supergroup          0 2022-03-25 08:12 /tmp
drwxr-xr-x   - hdfs   supergroup          0 2022-03-17 07:20 /user
drwxr-xr-x   - hdfs   supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart Desktop]$
```

Now let's check what we have inside this **tempop** directory using command as

hdfs dfs –ls /tempop

```
[cloudera@quickstart Desktop]$ hdfs dfs -ls /tempop
Found 2 items
-rw-r--r--   1 hdfs supergroup          0 2022-03-27 05:56 /tempop/_SUCCESS
-rw-r--r--   1 hdfs supergroup         17 2022-03-27 05:56 /tempop/part-r-00000
[cloudera@quickstart Desktop]$
```

Now we want to read the content of the part-r-00000 file which present inside the **tempop** using command

hdfs dfs -cat /tempop/part-r-00000

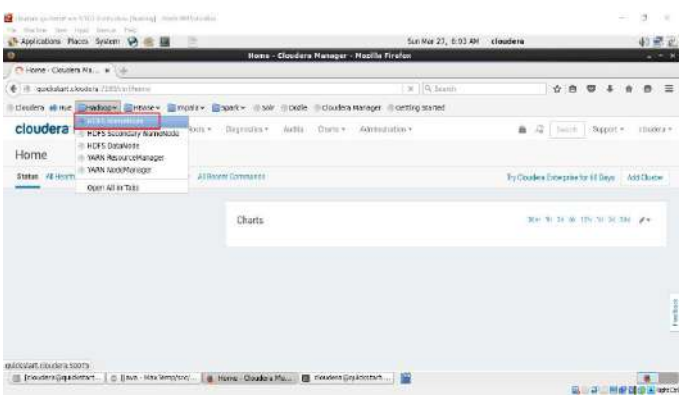
```
[cloudera@quickstart Desktop]$ hdfs dfs -cat /tempop/part-r-00000
1949    111
1950    22
[cloudera@quickstart Desktop]$
```

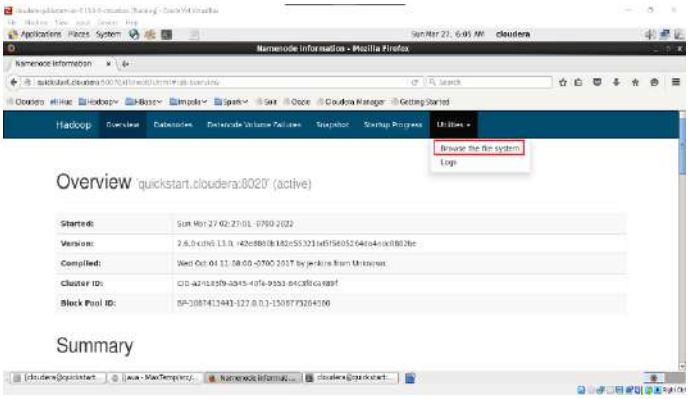
So the maximum temperature for the year 1949 is 111 and for the year 1950 is 22.

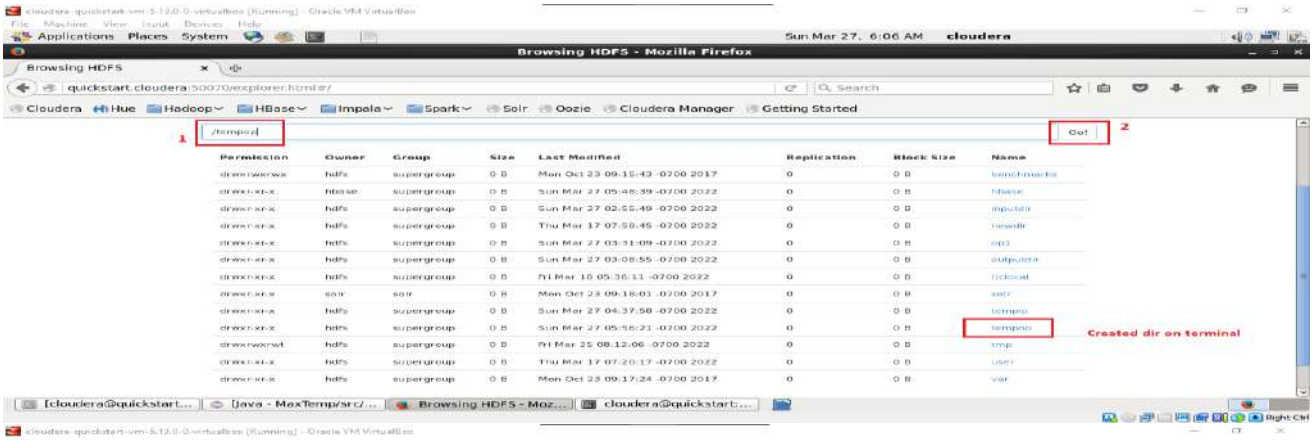
12. The same file can also be accessed using a browser.

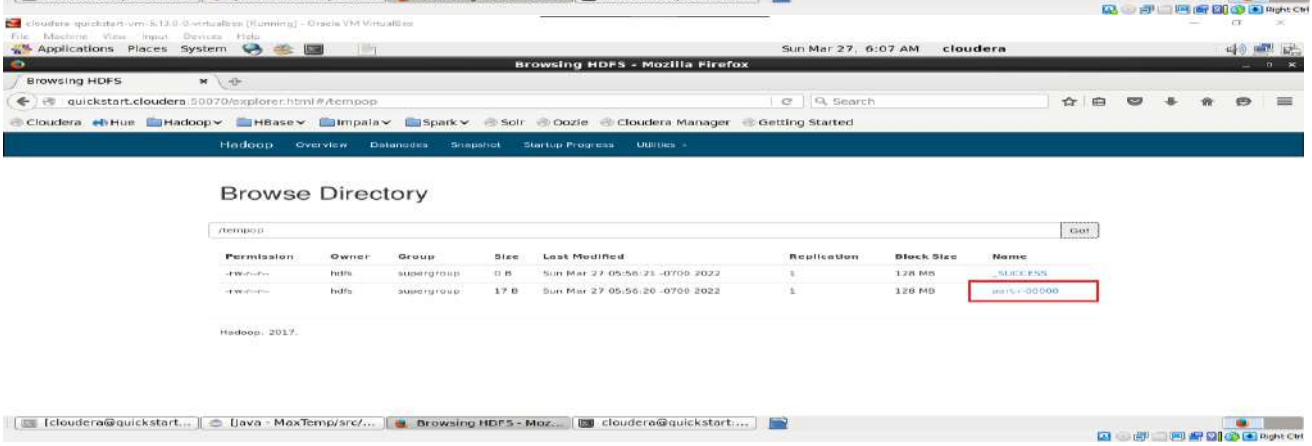
Browse the Directory by

Hadoop->HDFS Namenode->Ultilities ->Browse the file system

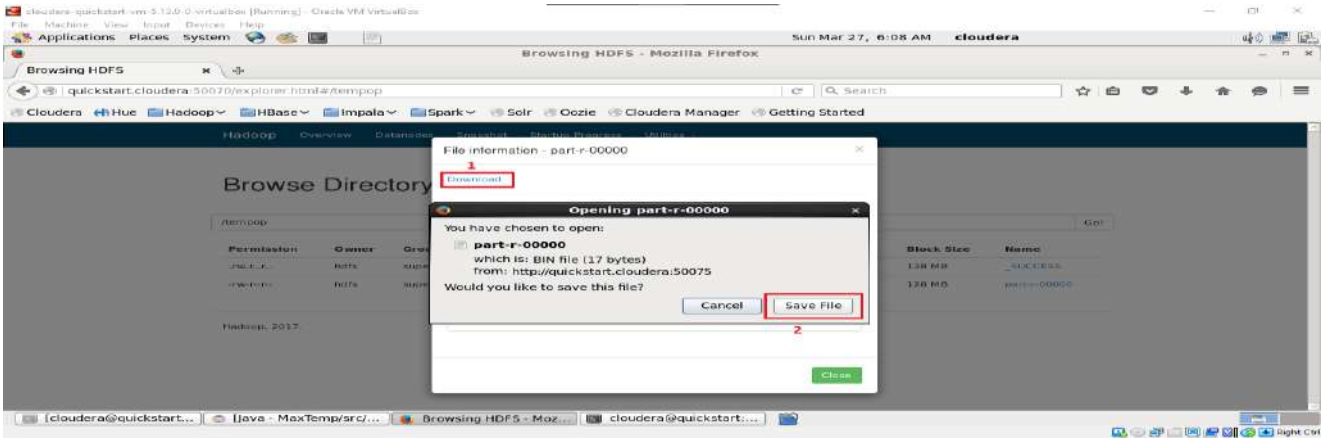


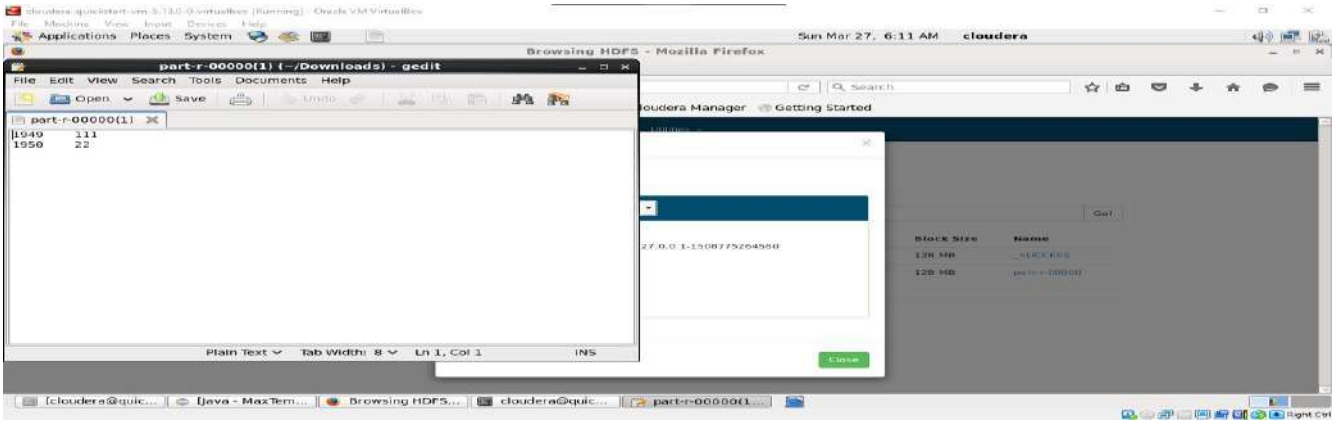






Now **downloading the part-r-00000 file.**





For every execution of this program we need to delete the output directory or give a new name to the output directory every time.

Inside the part-r-00000 file it will have the same output as we are getting after executing using command.

hadoop jar /home/cloudera/MaximumTemp.jar MaxTemp /tempip/temperature /tempop1

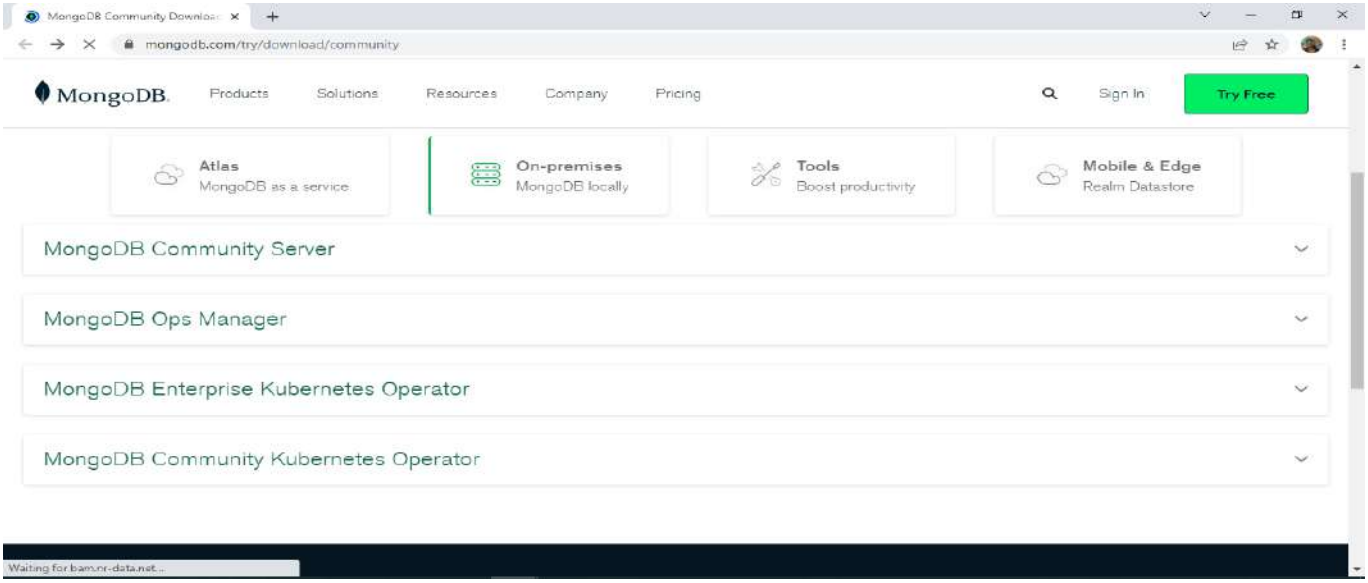
Aim: A- Installing MongoDB

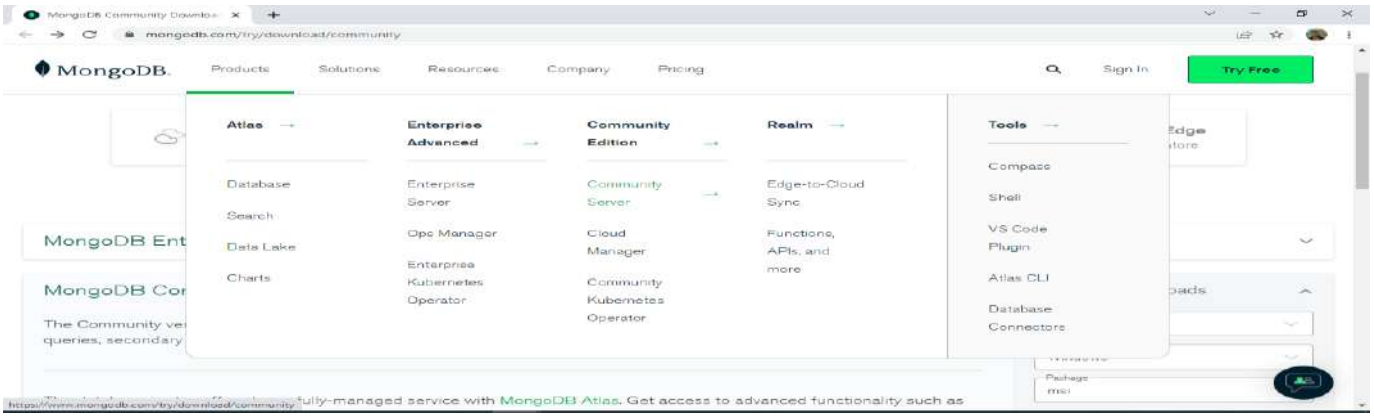
What is MongoDB?

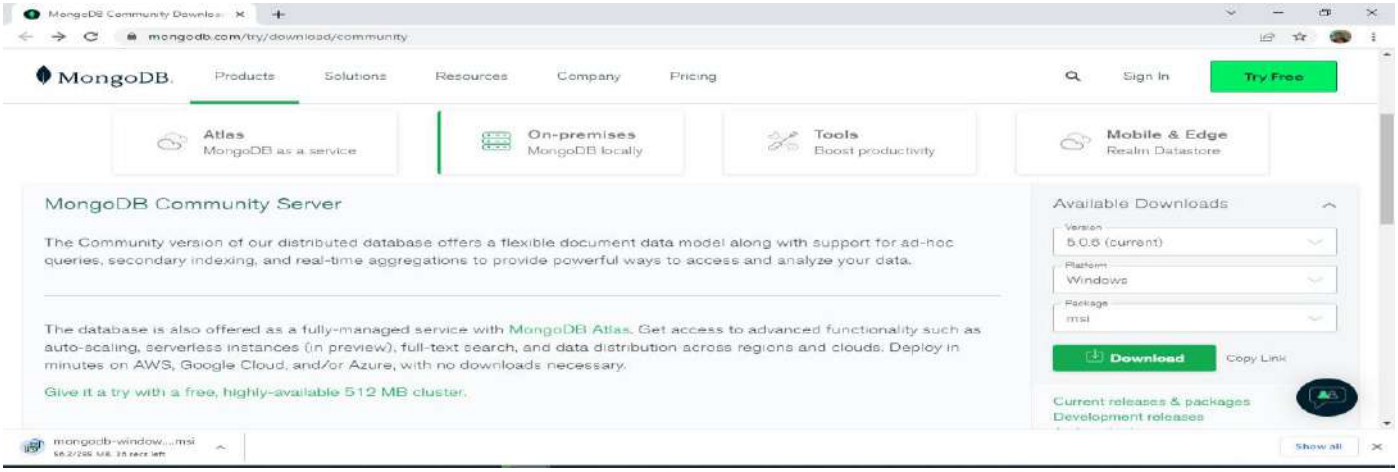
MongoDB is a document-oriented **NoSQL database used for high volume data storage**. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.

Download & Install MongoDB on Windows

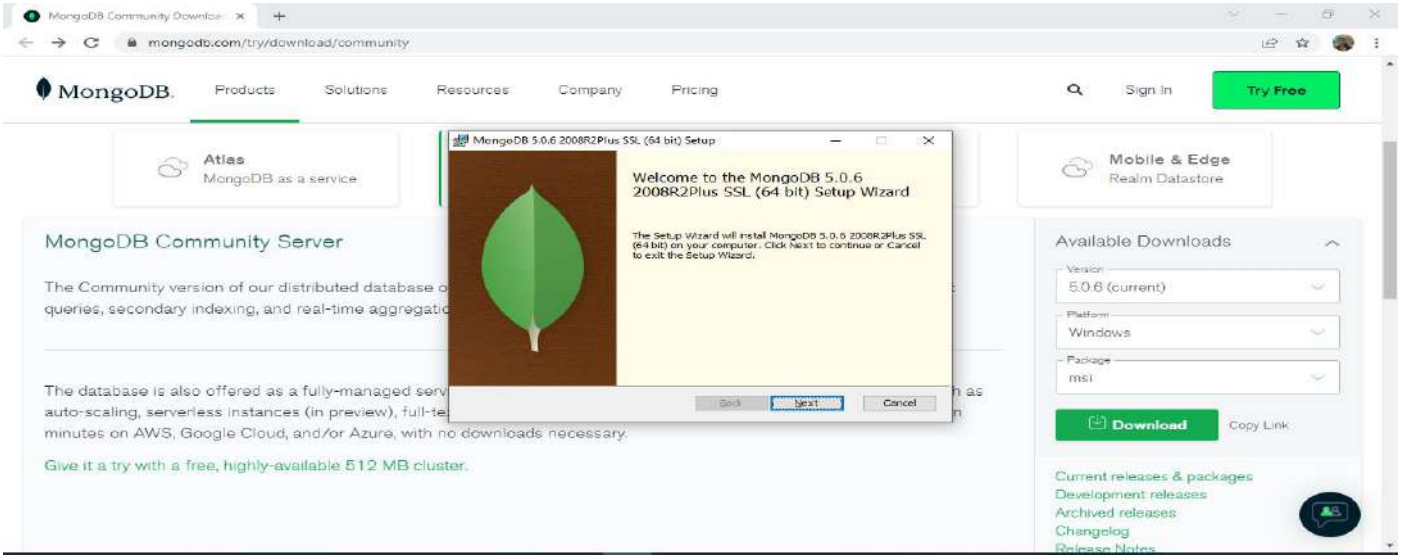
1) Go to link (https://www.mongodb.com/try/download/community) and Download MongoDB Community Server. We will install the 64-bit version for Windows.



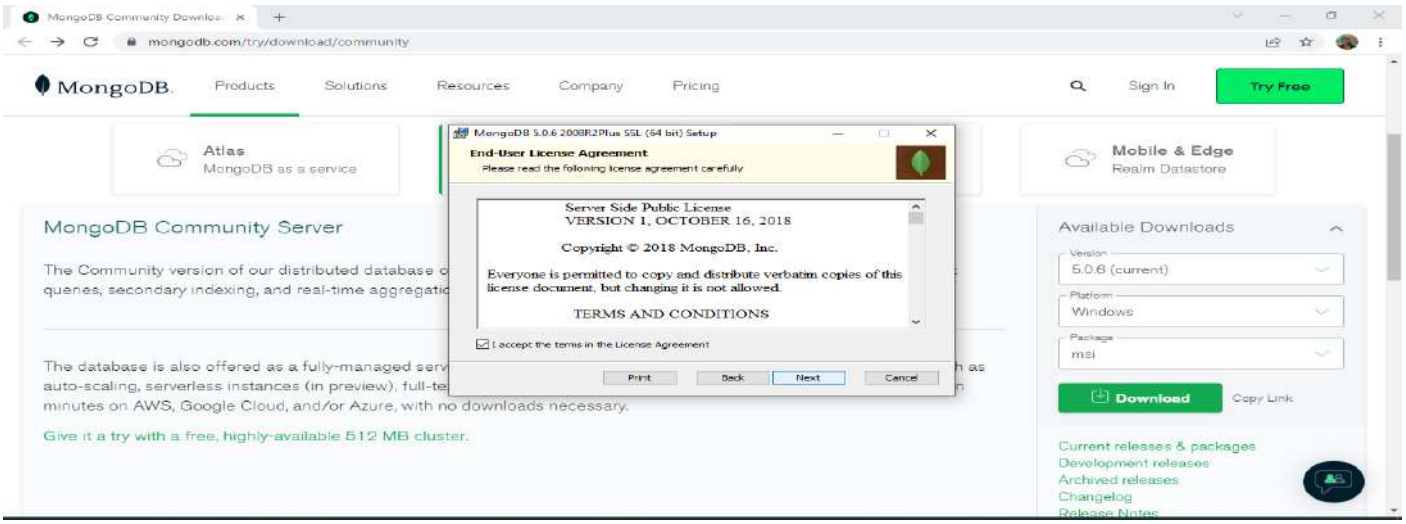




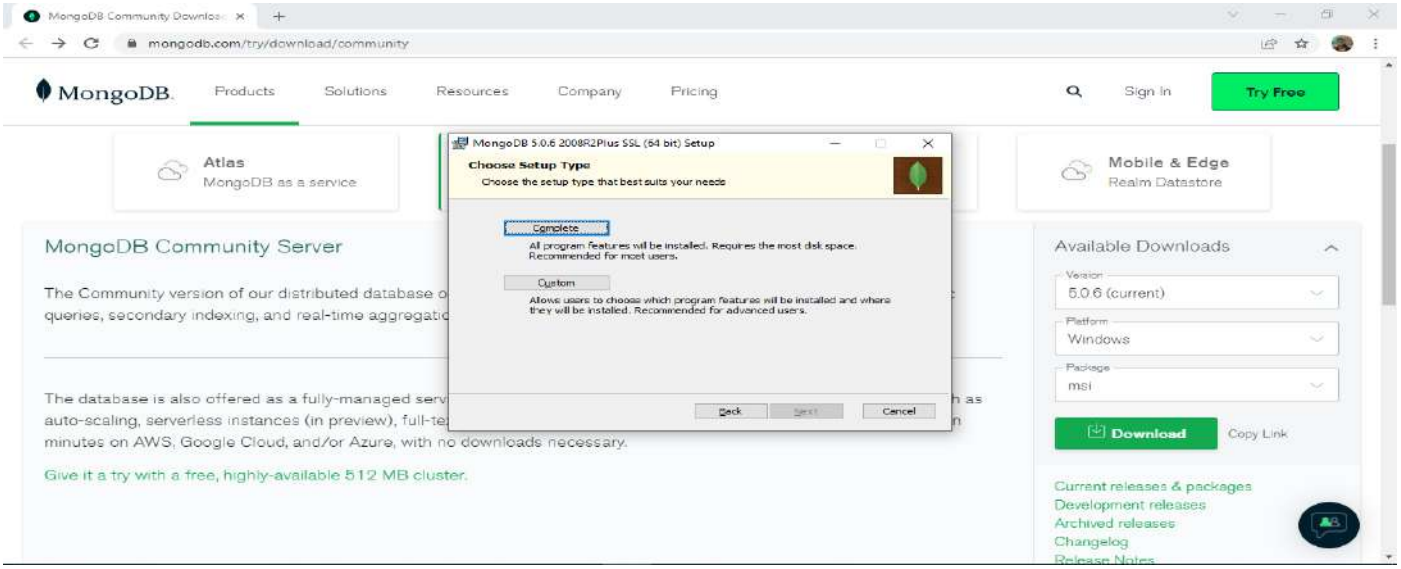
2) Once download is complete open the msi file. Click Next in the start-up screen



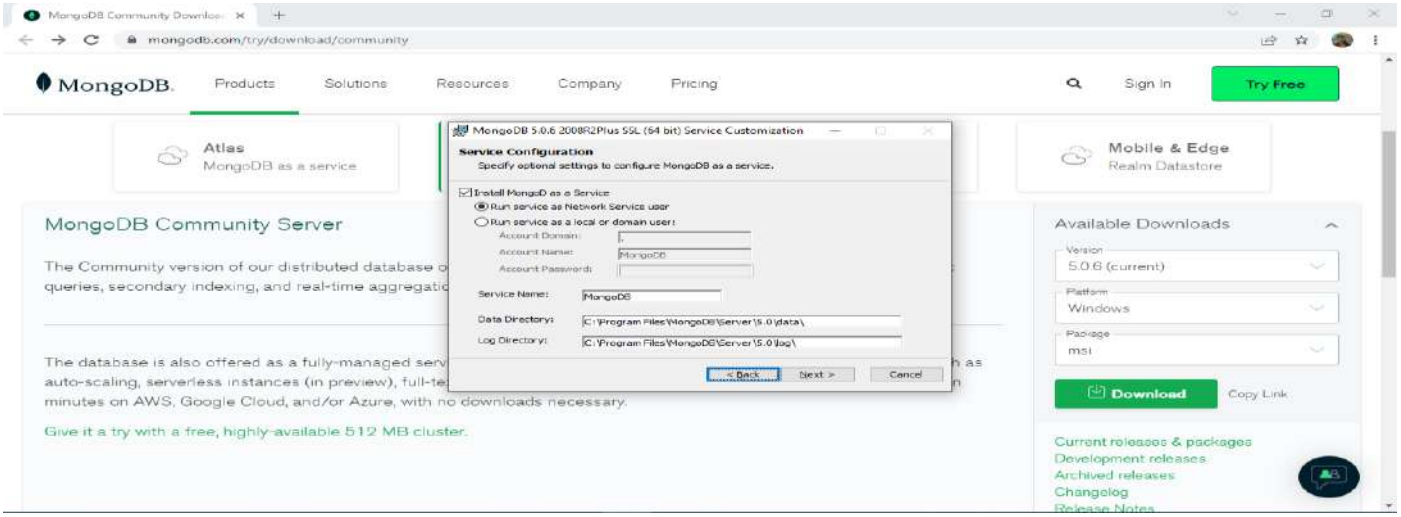
3) Accept the End-User License Agreement and **Click Next**



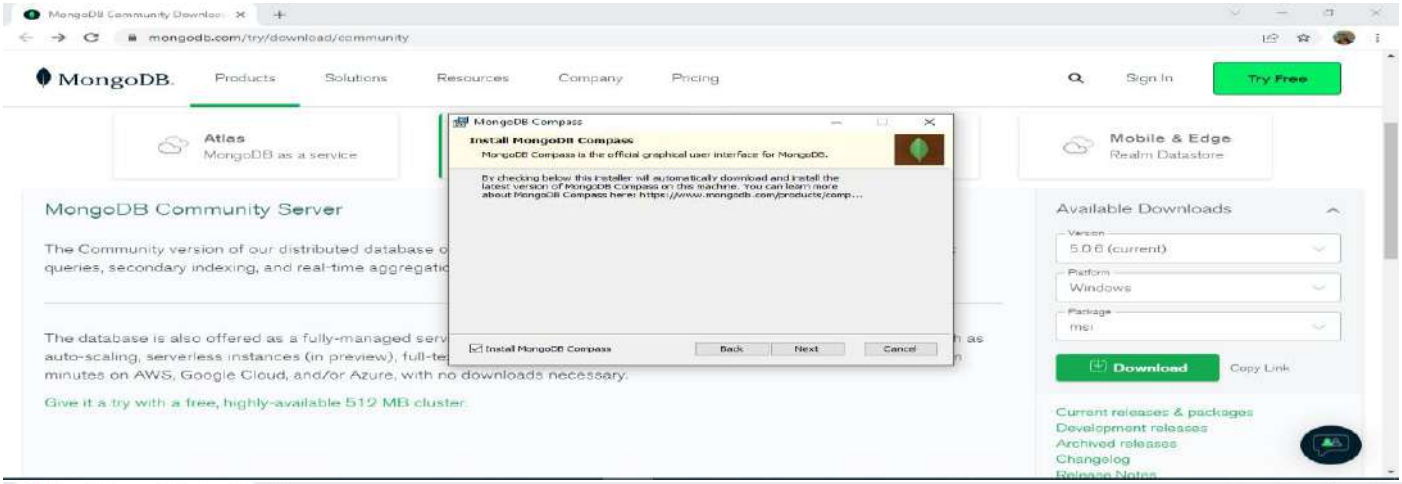
4) Click on the "**complete**" button to install all of the components. The custom option can be used to install selective components or if you want to change the location of the installation.

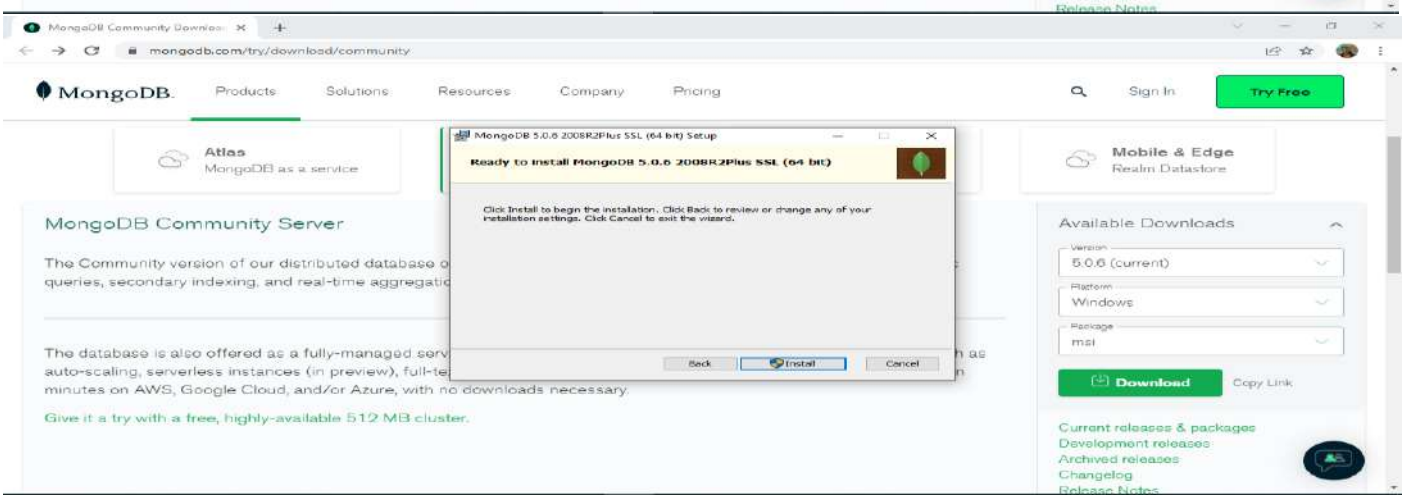


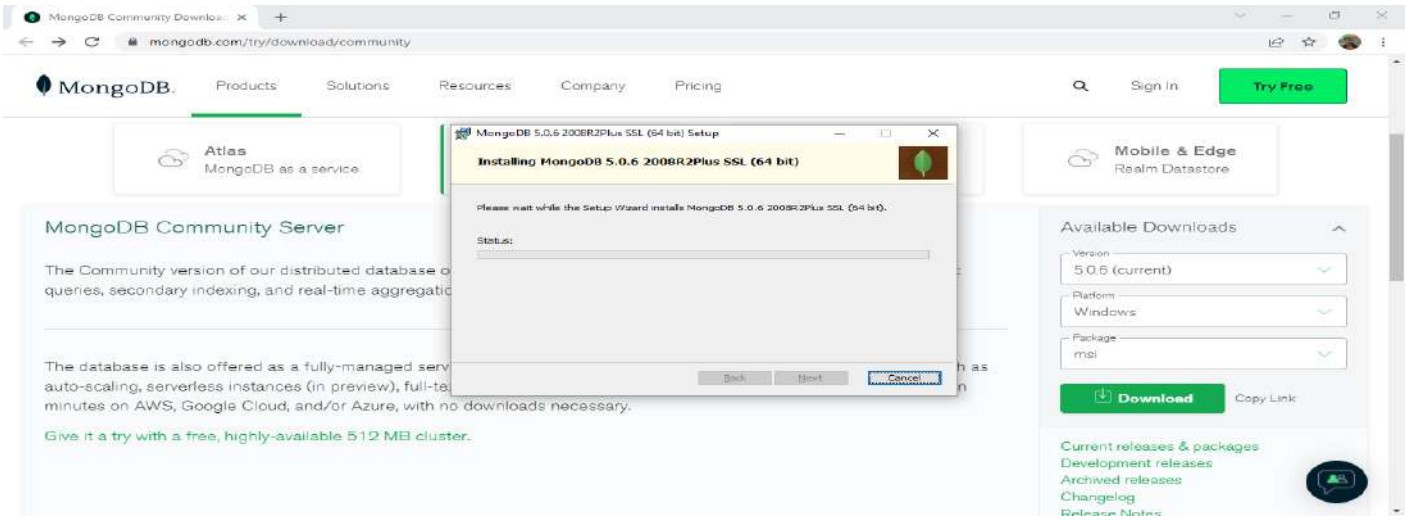
5) Select “Run service as Network Service user”. Make a note of the data directory, we’ll need this later and Click Next.



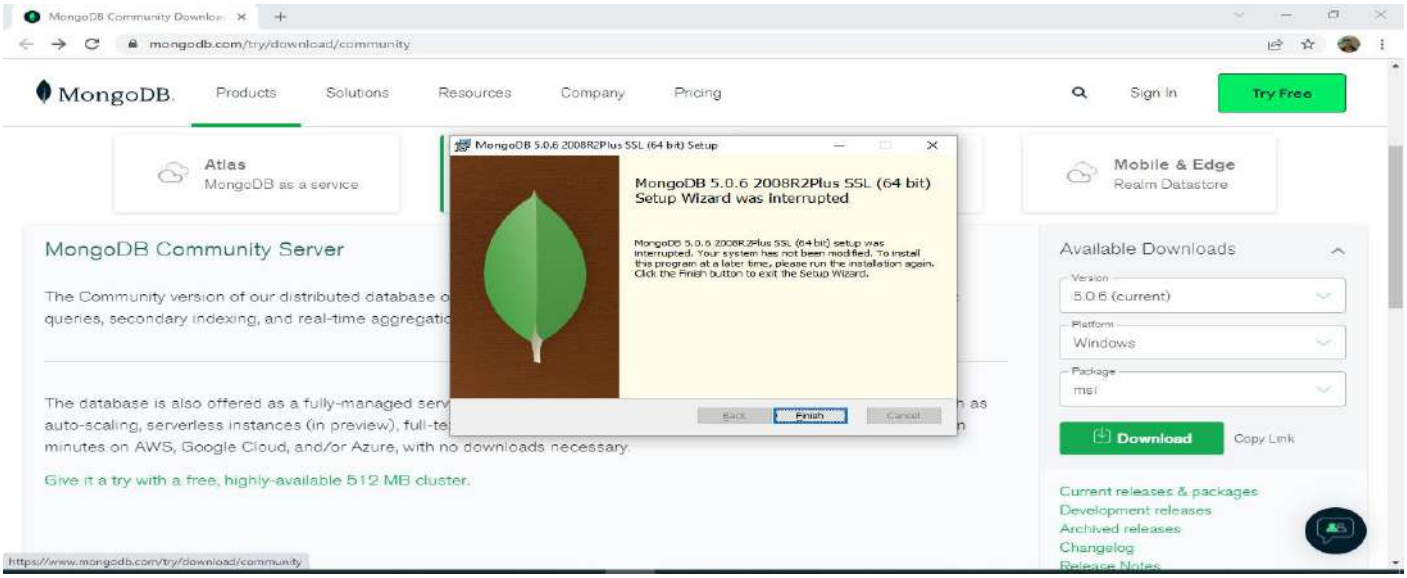
6) Click on the Install button to start the installation.

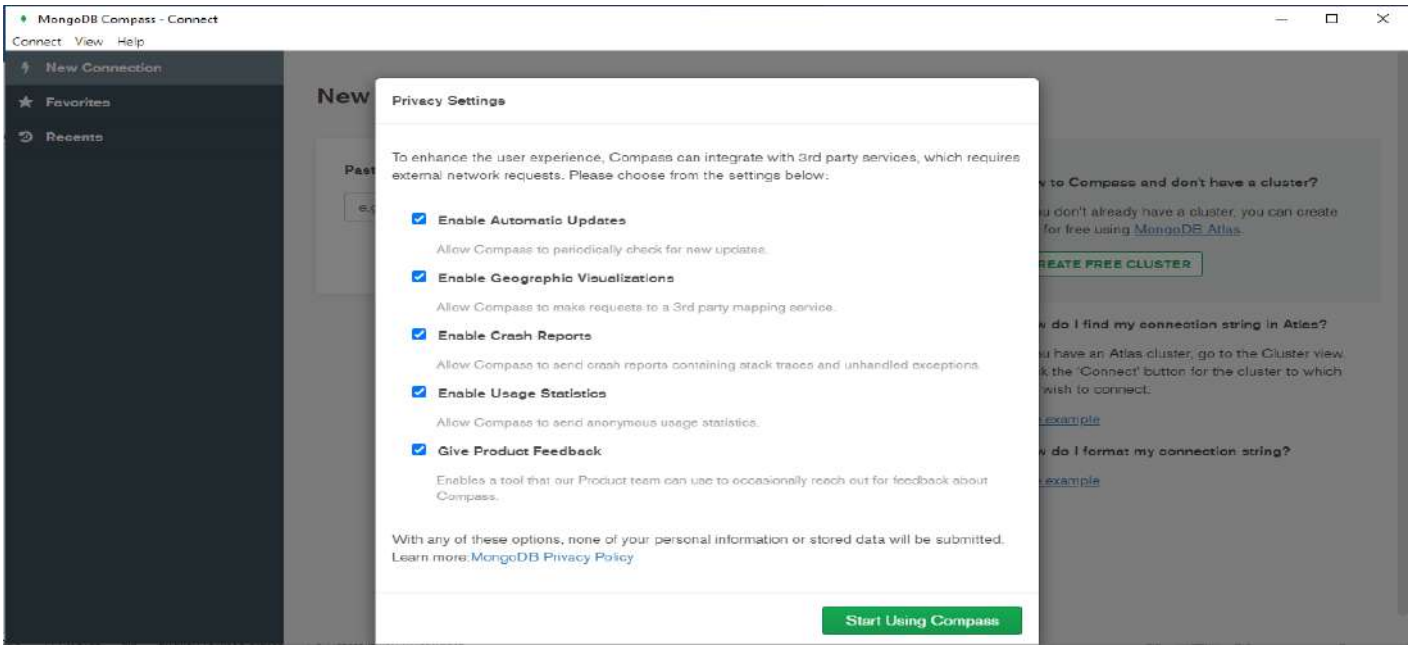




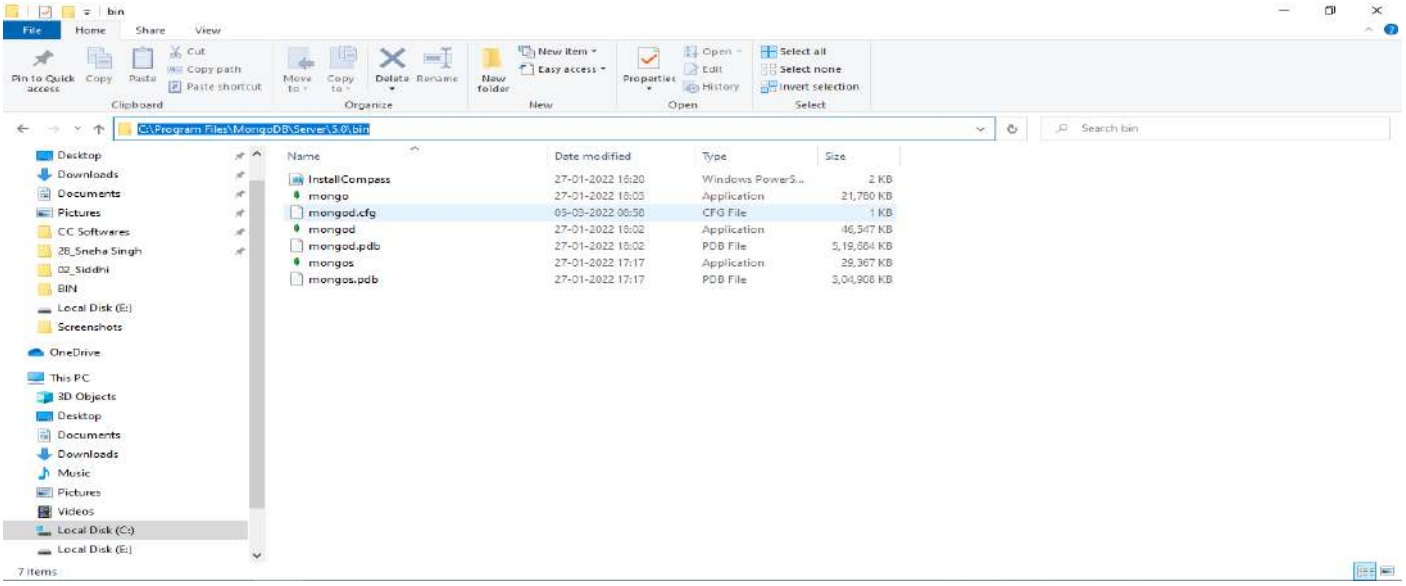


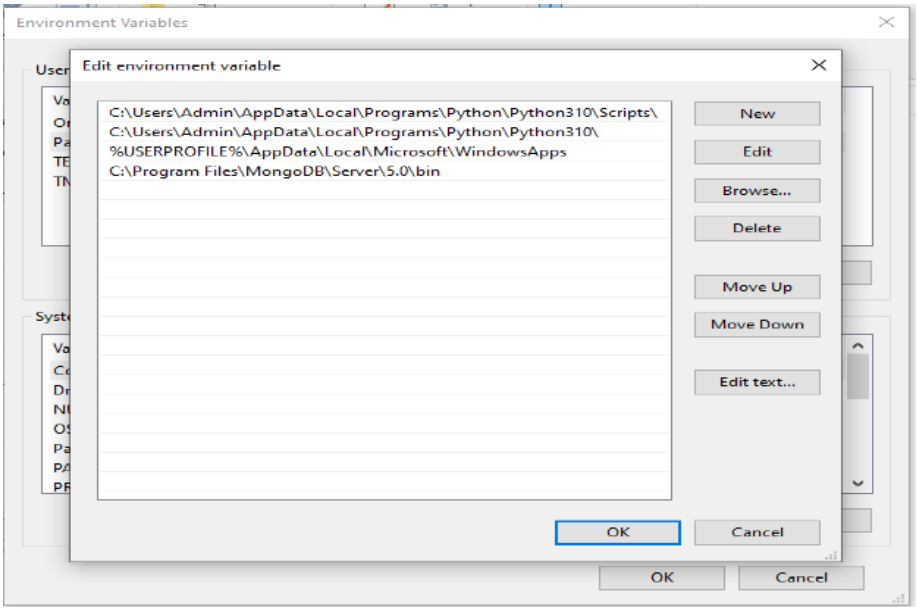
7) Click on the Finish button to complete the installation



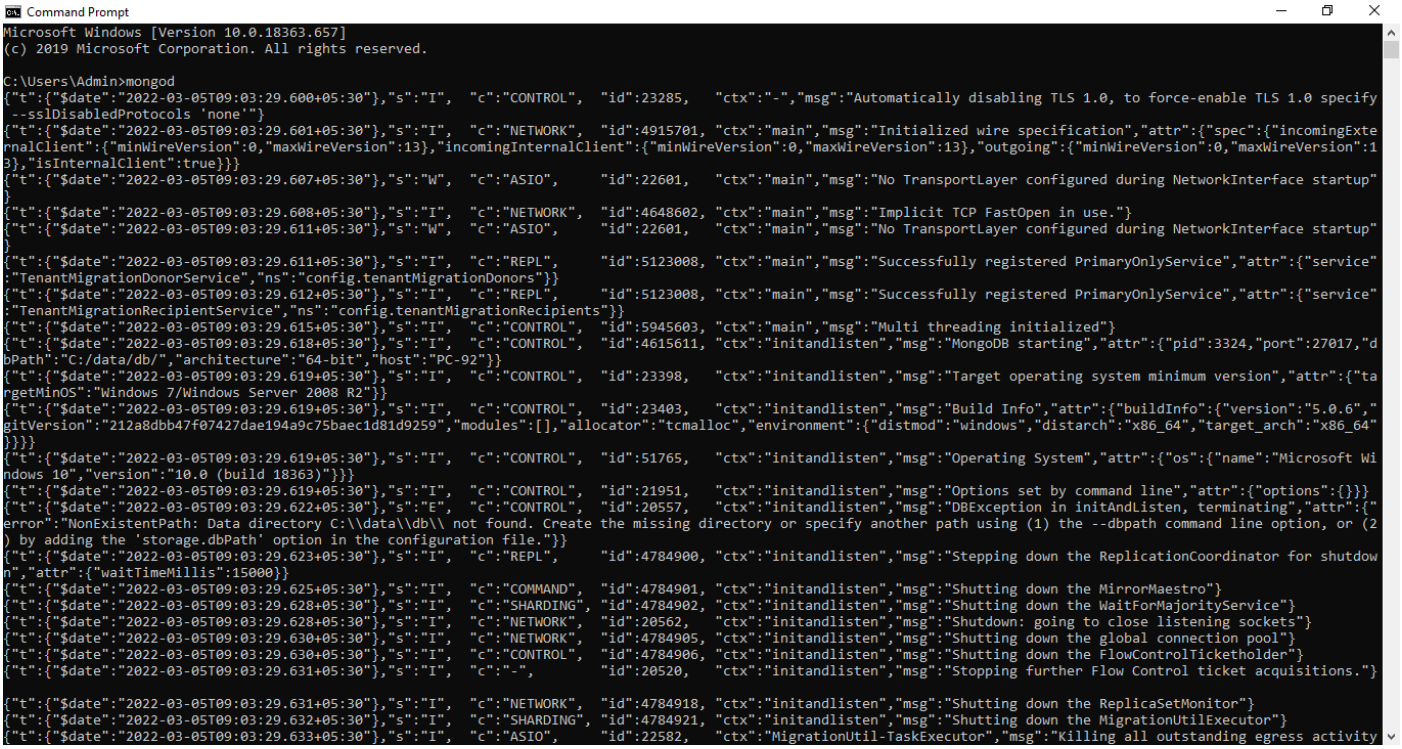


8) Go to folder " C:\Program Files\MongoDB\Server\4.0\bin" and copy the location.





9) Open the command prompt and run the cd command to change directory and run the command “mongod” to start the server and run the command “mongo” to work mongoDB.



Aim: B- Performing CRUD Operations for unstructured data

For storing data in a MongoDB, you need to create a database first. It will allow you to systematically organize your data so that it can be retrieved as per requirement. If you wish to delete a database, MongoDB also allows you to delete that.

What is CRUD in MongoDB?

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated.

CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

1. The Create operation is used to insert new documents in the MongoDB database.
2. The Read operation is used to query a document in the database.
3. The Update operation is used to modify existing documents in the database.
4. The Delete operation is used to remove documents in the database.

Creating a Database in MongoDB

MongoDB has no "create" command for creating a database. Also, it is essential to note that MongoDB does not provide any specific command for creating a database. This might seem a bit agitated if you are new to this subject and database tool or in case you have used that conventional SQL as your database where you are required to create a new database, which will contain table and, you will then have to use the INSERT INTO TABLE to insert values manually within your table.

In this MongoDB tool, you need not have to produce, or it is optional to create a database manually. This is because MongoDB has the feature of automatically creating it for the first time for you, once you save your value in that collection. So, explicitly, you do not need to mention or put a command to create a database; instead it will be created automatically once the collection is filled with values.

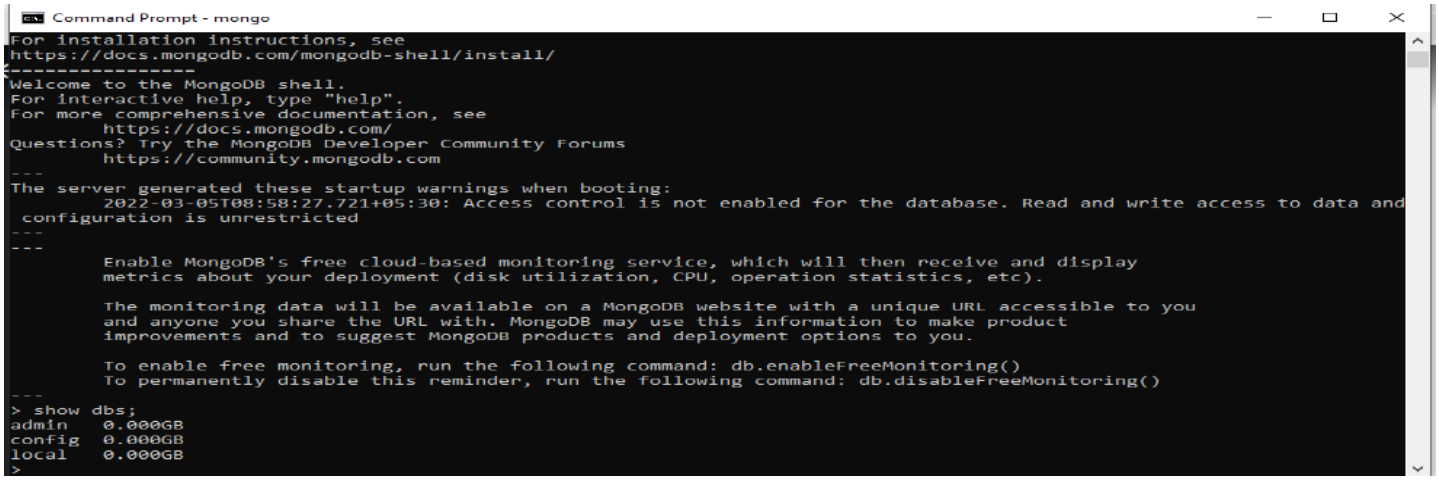
The "use" Command for Creating Database in MongoDB

You can make use of the "use" command followed by the database_name for creating a database. This command will tell the MongoDB client to create a database by this name if there is no database exists by this name. Otherwise, this command will return the existing database that has the name.

Show list of databases

First start the mongo Db using command **mongo**

1) show dbs;



2) use RetailBikeDb

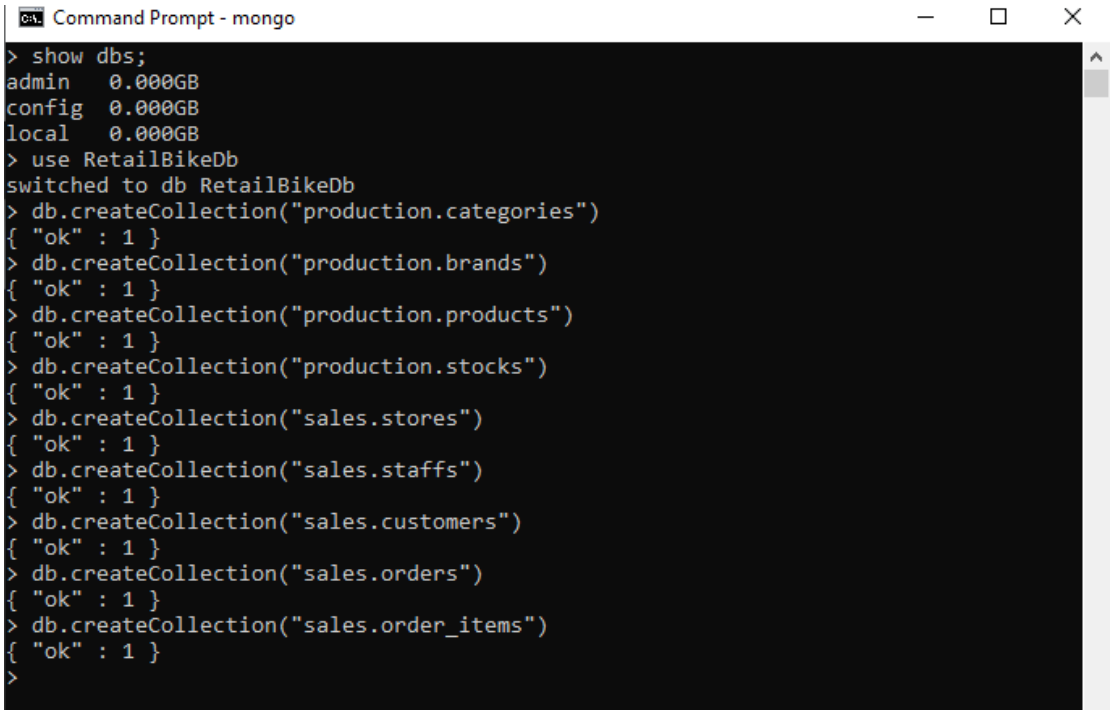
3) Creating and showing collections in MongoDB

Collections are like that of tables of RDBMS and are capable enough to store documents of diverse or dissimilar types. Creation and removal of collections in MongoDB can be done in specific ways. In this chapter, you will learn about the creation of collections in a database created using MongoDB.

Creation of collection can be done using

Syntax: db.createCollection(name)

db.createCollection("production.categories")
db.createCollection("production.brands")
db.createCollection("production.products")
db.createCollection("production.stocks")
db.createCollection("sales.stores")
db.createCollection("sales.staffs")
db.createCollection("sales.customers")
db.createCollection("sales.orders")
db.createCollection("sales.order_items")



4) We can show list of collection using "Show collections" commands in MongoDB.

show collections

5) CRUD operation

CRUD operation is one of the essential concepts of a database system. Inserting data in the database comes under one of the CRUD operations. If you do not insert data in your database, you will not be able to continue with other activities within your document.

1) Create Operations

For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.

MongoDB provides two different create operations that you can use to insert documents into a collection:

1. **db.collection.insertOne()**
2. **db.collection.insertMany()**

The insertOne() method

As the namesake, insertOne() allows you to insert one document into the collection. Example - db.movie.insertOne({ _id: 2, writername: "Stan Lee", name: "Aquaman" })

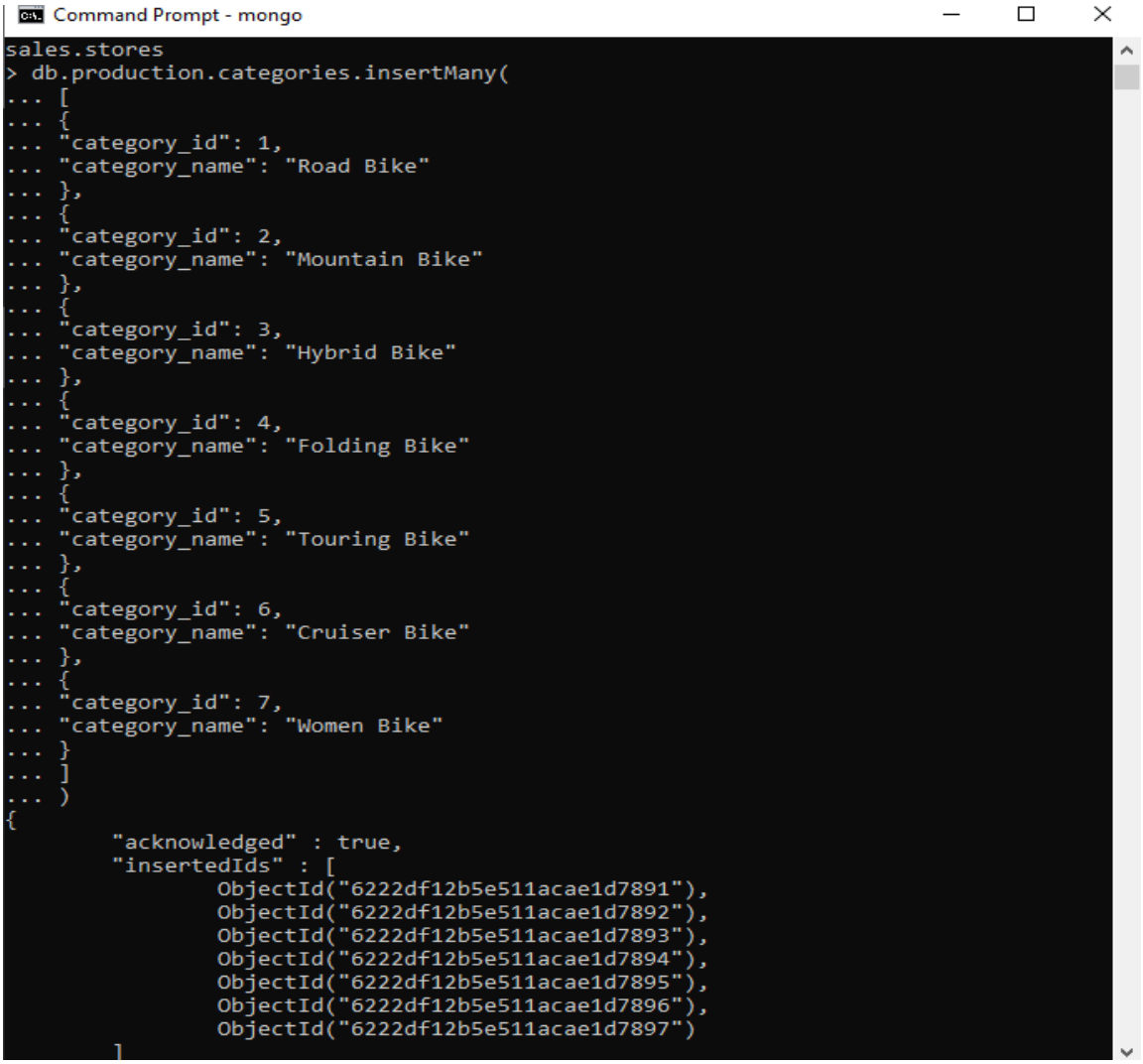
The insertMany() method

It's possible to insert multiple items at one time by calling the insertMany() method on the desired collection. In this case, we pass multiple items into our chosen collection (RetailDB) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

I have taken example of Retail Bike store database to demonstrate insert crud operations.

1. **production.categories**

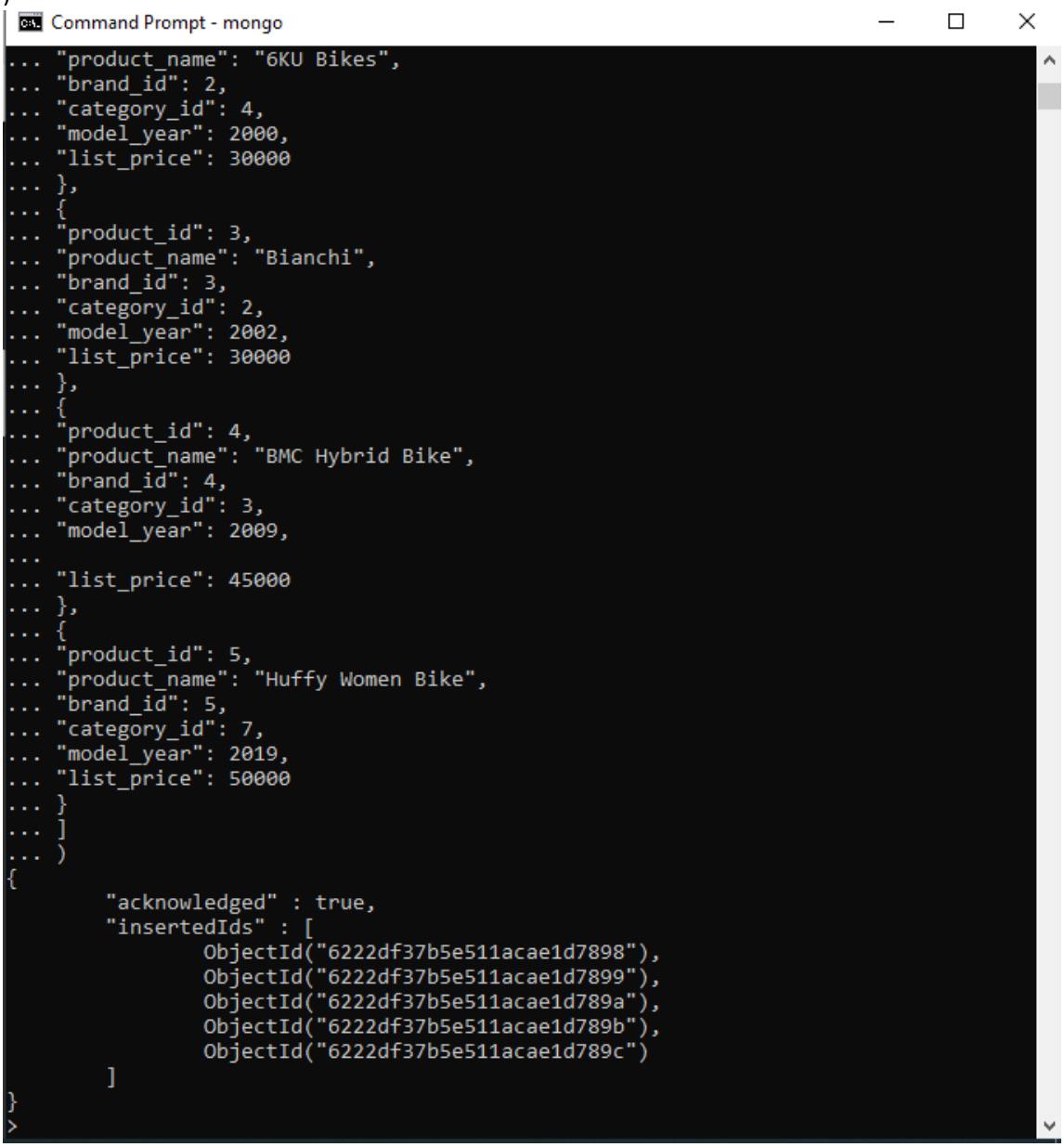
```
db.production.categories.insertMany(
[
{
"category_id": 1,
"category_name": "Road Bike"
},
{
"category_id": 2,
"category_name": "Mountain Bike"
},
{
"category_id": 3,
"category_name": "Hybrid Bike"
},
{
"category_id": 4,
"category_name": "Folding Bike"
},
{
"category_id": 5,
"category_name": "Touring Bike"
},
{
"category_id": 6,
"category_name": "Cruiser Bike"
},
{
"category_id": 7,
"category_name": "Women Bike"
}
]
)
```



2. **production.products**

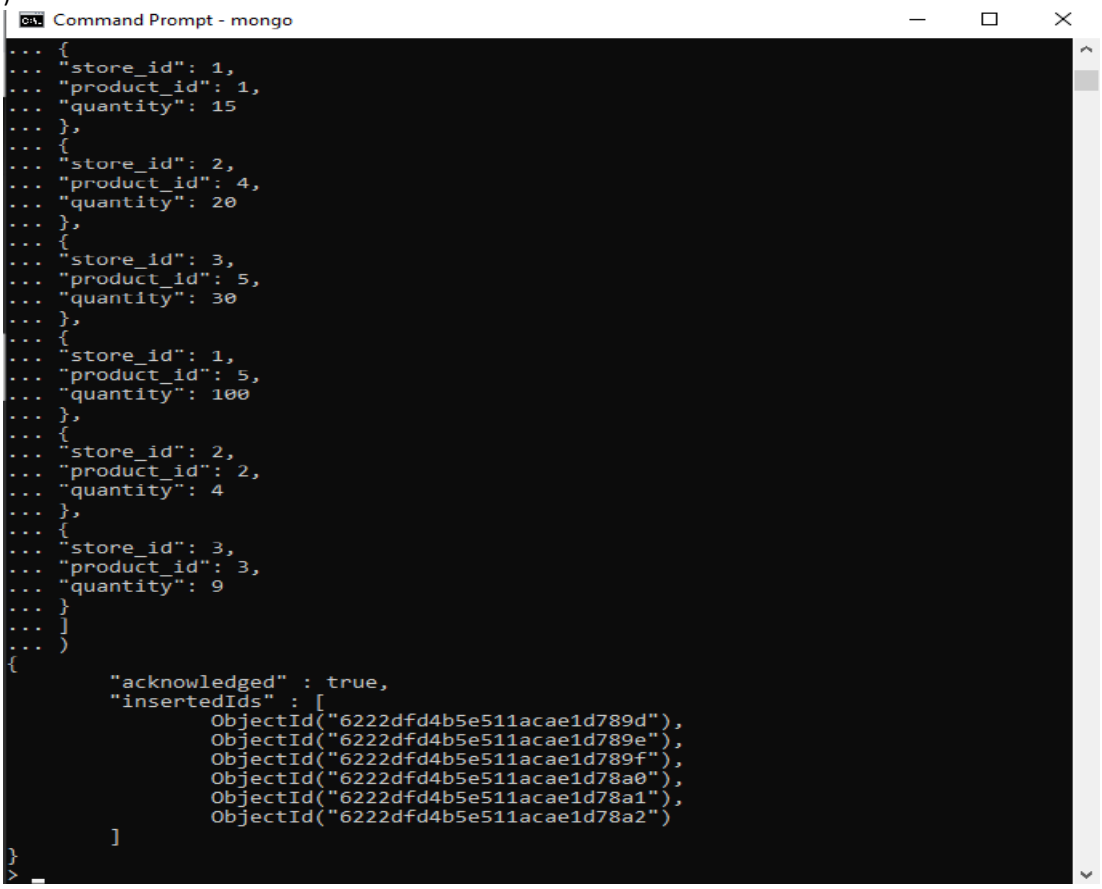
```
db.production.products.insertMany(
[
{
"product_id": 1,
"product_name": "Honda Superfast",
"brand_id": 1,
"category_id": 1,
"model_year": 1994,
"list_price": 25000
},
{
"product_id": 2,
"product_name": "6KU Bikes",
"brand_id": 2,
"category_id": 4,
"model_year": 2000,
"list_price": 30000
},
{
"product_id": 3,
"product_name": "Bianchi",
"brand_id": 3,
"category_id": 2,
"model_year": 2002,
"list_price": 30000
},
{
```

```
"product_id": 4,
"product_name": "BMC Hybrid Bike",
"brand_id": 4,
"category_id": 3,
"model_year": 2009,

"list_price": 45000
},
{
"product_id": 5,
"product_name": "Huffy Women Bike",
"brand_id": 5,
"category_id": 7,
"model_year": 2019,
"list_price": 50000
}
]
)
```



3. **production.stocks**

```
db.production.stocks.insertMany(
[
{
"store_id": 1,
"product_id": 1,
"quantity": 15
},
{
"store_id": 2,
"product_id": 4,
"quantity": 20
},
{
"store_id": 3,
"product_id": 5,
"quantity": 30
},
{
"store_id": 1,
"product_id": 5,
"quantity": 100
},
{
"store_id": 2,
"product_id": 2,
"quantity": 4
},
{
"store_id": 3,
"product_id": 3,
"quantity": 9
}
]
)
```



4. **sales.customers**

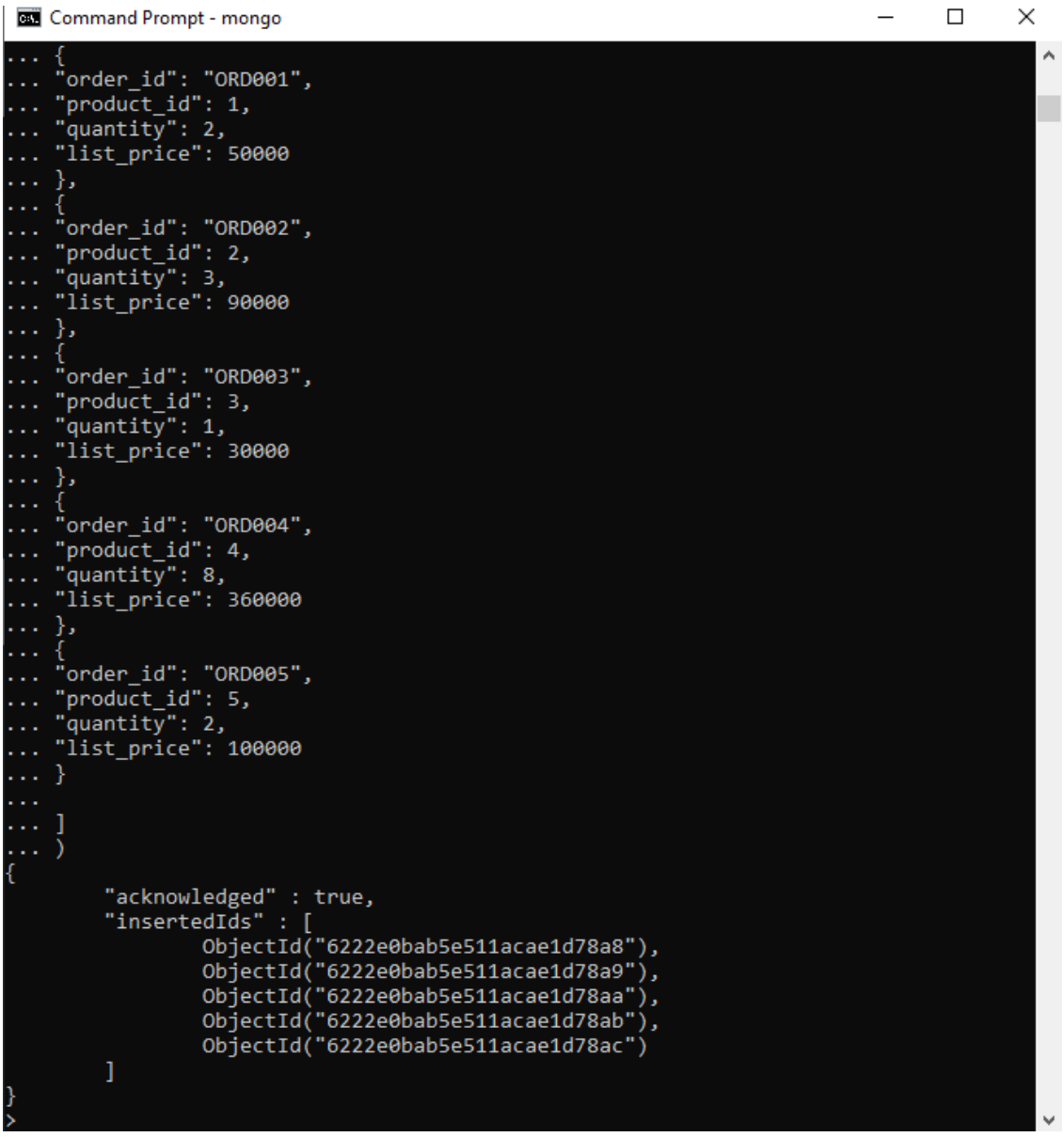
```
db.sales.customers.insertMany(
[
{
"customer_id": "Cus001",
"first_name": "Jay",
"last_name": "Mehta",
"phone": 1234567890,
"email": "jay@gmail.com",
"street": "T.P Road",
"city": "Mumbai",
"state": "Maharashtra",
"zip_code": 400072
},
{
"customer_id": "Cus002",
"first_name": "Ruhi",
"last_name": "Singh",
"phone": 7894561230,
"email": "ruhi@yahoo.com",
"street": "M.G Chauk",
"city": "Mumbai",
"state": "Maharashtra",
"zip_code": 400072
},
{
"customer_id": "Cus003",
"first_name": "Aria",
"last_name": "Josh",
"phone": 1245789630,
"email": "aria.josh@gmail.com",
"street": "JVM",
"city": "Gandhi Nagar",

"state": "Gujrat",
"zip_code": 401235
},
{
"customer_id": "Cus004",
"first_name": "Mahi",
"last_name": "Kaur",
"phone": 4567890123,
"email": "kaur.mahi@hotmail.com",
"street": "J.V.L.R",
"city": "Mumbai",
"state": "Maharashtra",
"zip_code": 400072
},
{
"customer_id": "Cus005",
"first_name": "Aditya",
"last_name": "Yadav",
"phone": 9638527410,
"email": "aditya@gmail.com",
"street": "Koliwada",
```

"city": "Pune",
"state": "Maharashtra",
"zip_code": 300075
}
]
)

5. **sales.order_items**

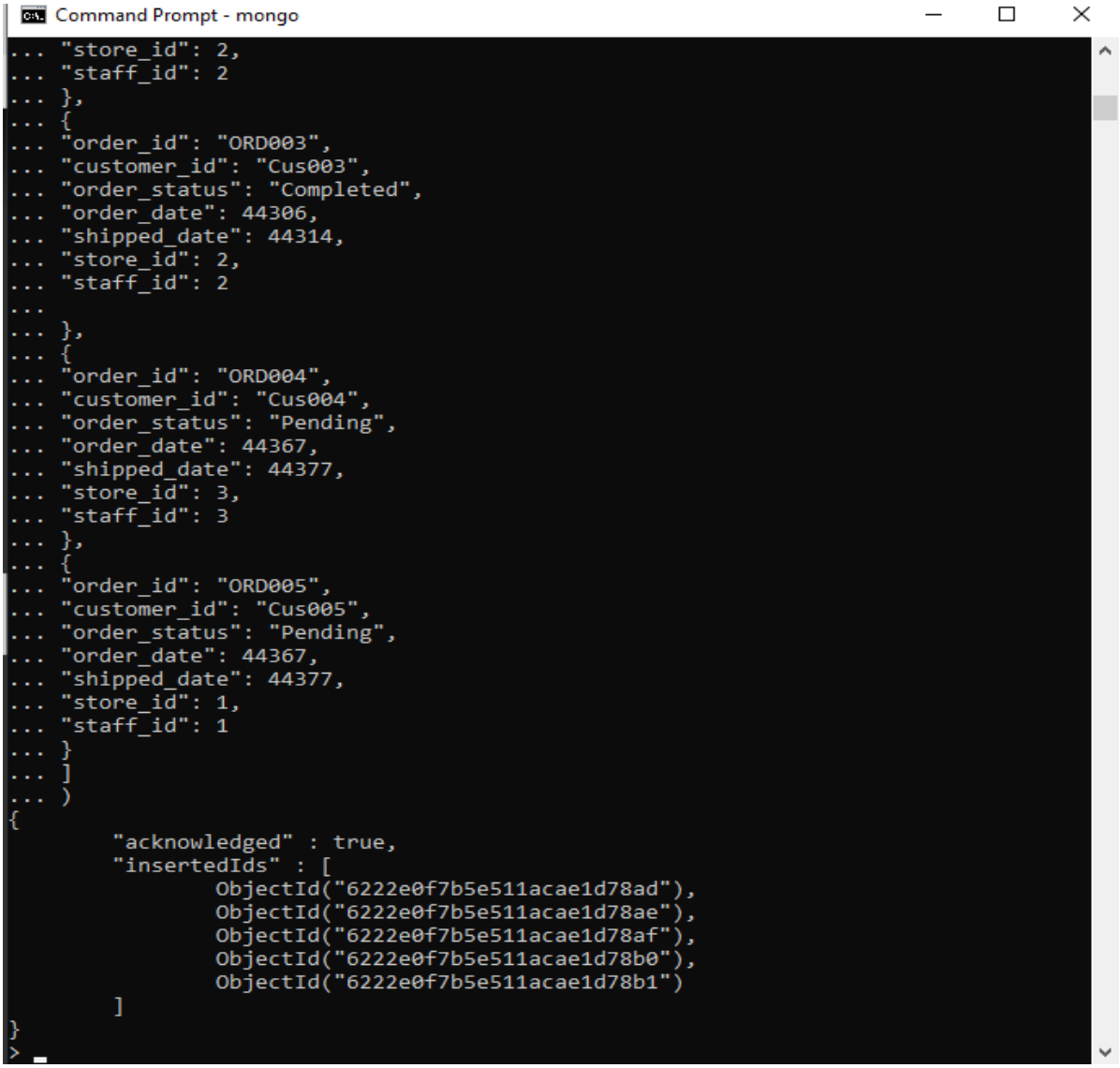
db.sales.order_items.insertMany(
[
{
"order_id": "ORD001",
"product_id": 1,
"quantity": 2,
"list_price": 50000
},
{
"order_id": "ORD002",
"product_id": 2,
"quantity": 3,
"list_price": 90000
},
{
"order_id": "ORD003",
"product_id": 3,

```
"quantity": 1,
"list_price": 30000
},
{
"order_id": "ORD004",
"product_id": 4,
"quantity": 8,
"list_price": 360000
},
{
"order_id": "ORD005",
"product_id": 5,
"quantity": 2,
"list_price": 100000
}

]
)
```



6. **sales.orders**

```
db.sales.orders.insertMany(
[
{
"order_id": "ORD001",
"customer_id": "Cus001",
"order_status": "Completed",
"order_date": 43992,
"shipped_date": 43994,
"store_id": 1,
"staff_id": 1
},
{
"order_id": "ORD002",
"customer_id": "Cus002",
"order_status": "Completed",
"order_date": 44221,
"shipped_date": 44227,
"store_id": 2,
"staff_id": 2
},
{
"order_id": "ORD003",
"customer_id": "Cus003",
"order_status": "Completed",
"order_date": 44306,
"shipped_date": 44314,
"store_id": 2,
"staff_id": 2

},
{
"order_id": "ORD004",
"customer_id": "Cus004",
"order_status": "Pending",
"order_date": 44367,
"shipped_date": 44377,
"store_id": 3,
"staff_id": 3
},
{
"order_id": "ORD005",
"customer_id": "Cus005",
"order_status": "Pending",
"order_date": 44367,
"shipped_date": 44377,
"store_id": 1,
"staff_id": 1
}
]
)
```



7. sales.staffs

```
db.sales.staffs.insertMany(
[
{
"staff_id": 1,
"first_name": "Pushpa",
"last_name": "Yadav",
"email": "pushpa@gmail.com",
"phone": 9999999999,
"active": "Yes",
"store_id": 1,
"manager_id": 1
},
{
"staff_id": 2,
"first_name": "Sadiksha",
"last_name": "Singh",
"email": "sadiksha@gmail.com",
"phone": 8888888888,
"active": "Yes",
"store_id": 2,
"manager_id": 1
},
{
"staff_id": 3,
"first_name": "Priya",
```

```
"last_name": "Nadar",
"email": "priya@gmail.com",
"phone": 7777777777,
"active": "Yes",
"store_id": 3,
"manager_id": 1

}
]
)
```

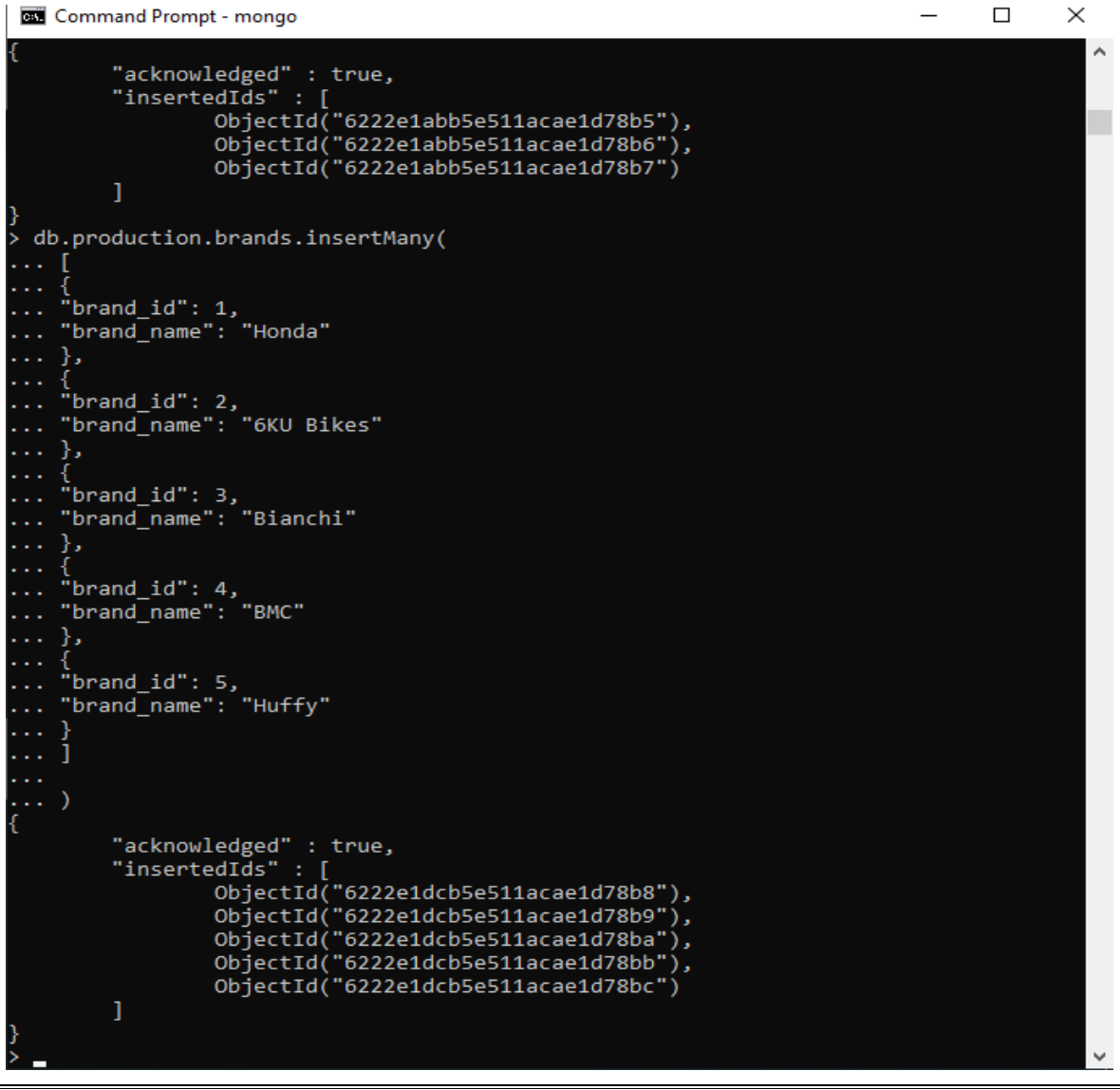
8. **sales.stores**

```
db.sales.stores.insertMany(
[
{
"store_id": 1,
"store_name": "Ambika Showroom",
"phone": 123456,
"email": "ambika@gmail.com",
"street": "GP",
"city": "Mumbai",
"state": "Maharashtra",
"zip_code": 400072
},
{
```

```
"store_id": 2,
"store_name": "Yash Bikes",
"phone": 789456,
"email": "yash.bikes@yahoo.com",
"street": "H.G",
"city": "Pune",
"state": "Maharashtra",
"zip_code": 300075
},
{
"store_id": 3,
"store_name": "Josh Automobiles",
"phone": 456983,
"email": "josh@gmail.com",

"street": "M.G",
"city": "Gandhi Nagar",
"state": "Gujrat",
"zip_code": 401235
}
]
)
```

9. **production.brands**

```
db.production.brands.insertMany(
[
{
"brand_id": 1,
"brand_name": "Honda"
},
{
"brand_id": 2,
"brand_name": "6KU Bikes"
},
{
"brand_id": 3,
"brand_name": "Bianchi"
},
{
"brand_id": 4,
"brand_name": "BMC"
},
{
"brand_id": 5,
"brand_name": "Huffy"
}
]

)
```



2) Read Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want. The MongoDB documentation contains more information on the available query filters. Query modifiers may also be used to change how many results are returned.

MongoDB has two methods of reading documents from a collection:

- **db.collection.find()**
- **db.collection.findOne()**

find()

In order to get all the documents from a collection, we can simply use the find() method on our chosen collection. Executing just the find() method with no arguments will return all records currently in the collection.

Example - **db.production.brands.find()**

Here we can see that every record has an assigned "ObjectId" mapped to the "_id" key.

```
> db.production.brands.find()
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b8"), "brand_id" : 1, "brand_name" : "Hon
da" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b9"), "brand_id" : 2, "brand_name" : "6KU
 Bikes" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78ba"), "brand_id" : 3, "brand_name" : "Bia
nchi" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bb"), "brand_id" : 4, "brand_name" : "BMC
" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bc"), "brand_id" : 5, "brand_name" : "Huf
fy" }
>
```

If you want to get more specific with a read operation and find a desired subsection of the records, you can use the previously mentioned filtering criteria to choose what results should be returned. One of the most common ways of filtering the results is to search by value.

Example: **db.production.brands.find({"brand_name" : "Honda"})**

```
> db.production.brands.find({"brand_name" : "Honda"})
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b8"), "brand_id" : 1, "brand_name" : "Hon
da" }
>
```

findOne()

In order to get one document that satisfies the search criteria, we can simply use the findOne() method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. If no documents satisfy the search criteria, the function returns null. The function takes the following form of syntax.

Syntax- db.{collection}.findOne({query}, {projection})

Example : **db.sales.order_items.findOne({"quantity": 2})**

```
> db.sales.order_items.findOne({"quantity": 2})
{
        "_id" : ObjectId("6222e0bab5e511acae1d78a8"),
        "order_id" : "ORD001",
        "product_id" : 1,
        "quantity" : 2,
        "list_price" : 50000
}
>
```

Query Documents

a. **Specify Equality Condition**

db.production.products.find() - to show all the documents

db.production.products.find({"list_price": 30000}) - to show only documents which have "list_price" as 30000

```
> db.sales.customers.find()
{ "_id" : ObjectId("6222e05db5e511acae1d78a3"), "customer_id" : "Cus001", "first_na
me" : "Jay", "last_name" : "Mehta", "phone" : 1234567890, "email" : "jay@gmail.com"
, "street" : "T.P Road", "city" : "Mumbai", "state" : "Maharashtra", "zip_code" : 4
00072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a4"), "customer_id" : "Cus002", "first_na
me" : "Ruhi", "last_name" : "Singh", "phone" : 7894561230, "email" : "ruhi@yahoo.co
m", "street" : "M.G Chauk", "city" : "Mumbai", "state" : "Maharashtra", "zip_code"
: 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a5"), "customer_id" : "Cus003", "first_na
me" : "Aria", "last_name" : "Josh", "phone" : 1245789630, "email" : "aria.josh@gmai
l.com", "street" : "JVM", "city" : "Gandhi Nagar", "state" : "Gujrat", "zip_code" :
 401235 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a6"), "customer_id" : "Cus004", "first_na
me" : "Mahi", "last_name" : "Kaur", "phone" : 4567890123, "email" : "kaur.mahi@hotm
ail.com", "street" : "J.V.L.R", "city" : "Mumbai", "state" : "Maharashtra", "zip_co
de" : 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a7"), "customer_id" : "Cus005", "first_na
me" : "Aditya", "last_name" : "Yadav", "phone" : 9638527410, "email" : "aditya@gmai
l.com", "street" : "Koliwada", "city" : "Pune", "state" : "Maharashtra", "zip_code"
 : 300075 }
> db.production.products.find({"list_price": 30000})
{ "_id" : ObjectId("6222df37b5e511acae1d7899"), "product_id" : 2, "product_name" :
"6KU Bikes", "brand_id" : 2, "category_id" : 4, "model_year" : 2000, "list_price" :
 30000 }
{ "_id" : ObjectId("6222df37b5e511acae1d789a"), "product_id" : 3, "product_name" :
"Bianchi", "brand_id" : 3, "category_id" : 2, "model_year" : 2002, "list_price" : 3
0000 }
>
```

b. Specify Conditions Using Query Operators

db.sales.customers.find() - to show all the documents

db.sales.customers.find({city: { $in: ["Mumbai", "Pune"] } }) - to show all the documents where "city" is either "Mumbai" or "Pune"

```
> db.sales.customers.find()
{ "_id" : ObjectId("6222e05db5e511acae1d78a3"), "customer_id" : "Cus001", "first_na
me" : "Jay", "last_name" : "Mehta", "phone" : 1234567890, "email" : "jay@gmail.com"
, "street" : "T.P Road", "city" : "Mumbai", "state" : "Maharashtra", "zip_code" : 4
00072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a4"), "customer_id" : "Cus002", "first_na
me" : "Ruhi", "last_name" : "Singh", "phone" : 7894561230, "email" : "ruhi@yahoo.co
m", "street" : "M.G Chauk", "city" : "Mumbai", "state" : "Maharashtra", "zip_code"
: 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a5"), "customer_id" : "Cus003", "first_na
me" : "Aria", "last_name" : "Josh", "phone" : 1245789630, "email" : "aria.josh@gmai
l.com", "street" : "JVM", "city" : "Gandhi Nagar", "state" : "Gujrat", "zip_code" :
 401235 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a6"), "customer_id" : "Cus004", "first_na
me" : "Mahi", "last_name" : "Kaur", "phone" : 4567890123, "email" : "kaur.mahi@hotm
ail.com", "street" : "J.V.L.R", "city" : "Mumbai", "state" : "Maharashtra", "zip_co
de" : 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a7"), "customer_id" : "Cus005", "first_na
me" : "Aditya", "last_name" : "Yadav", "phone" : 9638527410, "email" : "aditya@gmai
l.com", "street" : "Koliwada", "city" : "Pune", "state" : "Maharashtra", "zip_code"
 : 300075 }
> db.sales.customers.find({city: { $in: [ "Mumbai", "Pune" ] } })
{ "_id" : ObjectId("6222e05db5e511acae1d78a3"), "customer_id" : "Cus001", "first_na
me" : "Jay", "last_name" : "Mehta", "phone" : 1234567890, "email" : "jay@gmail.com"
, "street" : "T.P Road", "city" : "Mumbai", "state" : "Maharashtra", "zip_code" : 4
00072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a4"), "customer_id" : "Cus002", "first_na
me" : "Ruhi", "last_name" : "Singh", "phone" : 7894561230, "email" : "ruhi@yahoo.co
m", "street" : "M.G Chauk", "city" : "Mumbai", "state" : "Maharashtra", "zip_code"
: 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a6"), "customer_id" : "Cus004", "first_na
me" : "Mahi", "last_name" : "Kaur", "phone" : 4567890123, "email" : "kaur.mahi@hotm
ail.com", "street" : "J.V.L.R", "city" : "Mumbai", "state" : "Maharashtra", "zip_co
de" : 400072 }
{ "_id" : ObjectId("6222e05db5e511acae1d78a7"), "customer_id" : "Cus005", "first_na
me" : "Aditya", "last_name" : "Yadav", "phone" : 9638527410, "email" : "aditya@gmai
l.com", "street" : "Koliwada", "city" : "Pune", "state" : "Maharashtra", "zip_code"
 : 300075 }
>
```

c. Specify AND Conditions

db.sales.order_items.find() - to show all the documents

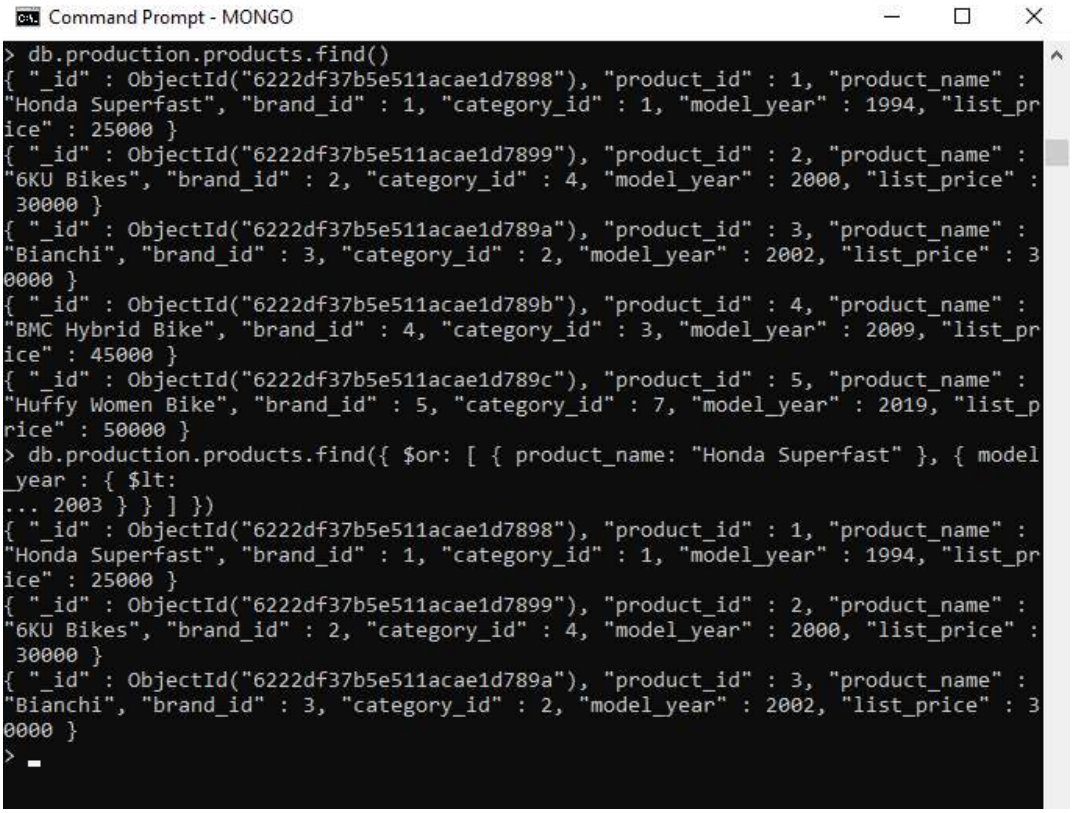
db.sales.order_items.find({quantity: 2, list_price : { $gt: 70000}}) - Here we are trying to find how sales order item documents for which quantity is 2 and list price is greater than 70000

```
> db.sales.order_items.find()
{ "_id" : ObjectId("6222e0bab5e511acae1d78a8"), "order_id" : "ORD001", "product_id"
 : 1, "quantity" : 2, "list_price" : 50000 }
{ "_id" : ObjectId("6222e0bab5e511acae1d78a9"), "order_id" : "ORD002", "product_id"
 : 2, "quantity" : 3, "list_price" : 90000 }
{ "_id" : ObjectId("6222e0bab5e511acae1d78aa"), "order_id" : "ORD003", "product_id"
 : 3, "quantity" : 1, "list_price" : 30000 }
{ "_id" : ObjectId("6222e0bab5e511acae1d78ab"), "order_id" : "ORD004", "product_id"
 : 4, "quantity" : 8, "list_price" : 360000 }
{ "_id" : ObjectId("6222e0bab5e511acae1d78ac"), "order_id" : "ORD005", "product_id"
 : 5, "quantity" : 2, "list_price" : 100000 }
> db.sales.order_items.find({quantity: 2, list_price : { $gt: 70000}})
{ "_id" : ObjectId("6222e0bab5e511acae1d78ac"), "order_id" : "ORD005", "product_id"
 : 5, "quantity" : 2, "list_price" : 100000 }
>
```

d. Specify OR Conditions

db.production.products.find() - to show all the documents

db.production.products.find({ $or: [{ product_name: "Honda Superfast" }, { model_year : { $lt: 2003 } }] }) - to show all the document which is either "product name" as "Honda Superfast" or "model year" is less than year 2003



3) Update Operations

Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

You should be careful when updating documents, as updates are permanent and can't be rolled back. This applies to delete operations as well.

For MongoDB CRUD, there are three different methods of updating documents:

- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

1. updateOne()

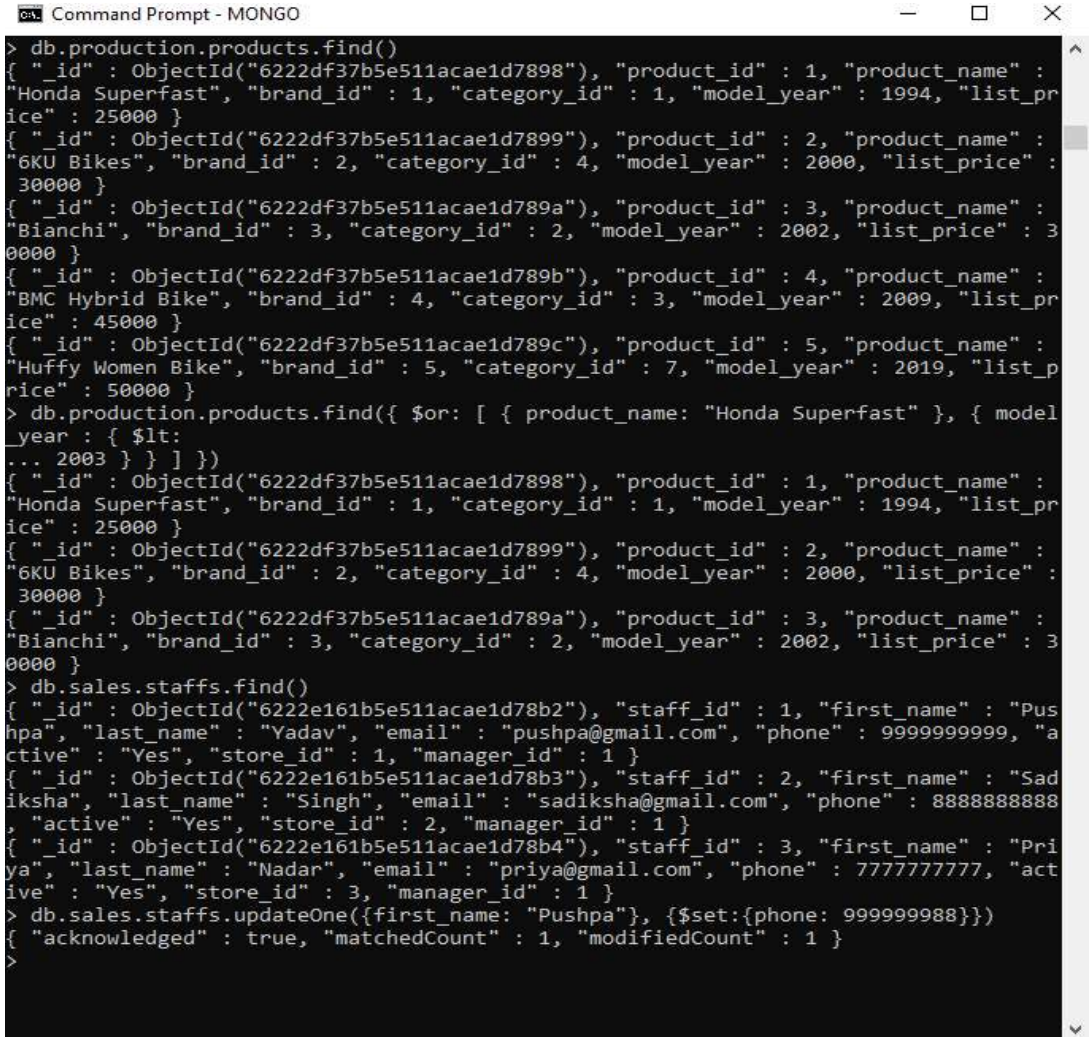
We can update a currently existing record and change a single document with an update operation. To do this, we use the updateOne() method on a chosen collection. To update a document, we provide the method with two arguments: an update filter and an update action.

The update filter defines which items we want to update, and the update action defines how to update those items. We first pass in the update filter. Then, we use the "$set" key and provide the fields we want to update as a value. This method will update the first record that matches the provided filter.

Example –

db.sales.staffs.find() - to show currect document in the system

db.sales.staffs.updateOne({first_name: "Pushpa"}, {$set:{phone: 999999988}}) - update mobile number from 9999999999 to 999999988 for document name where name is " first_name " in collection



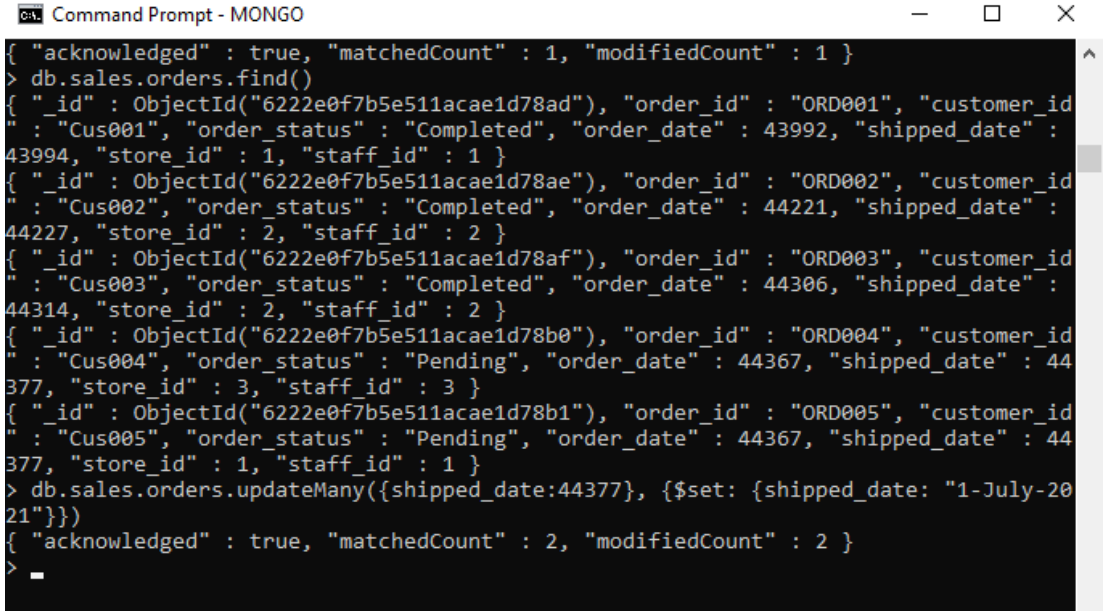
2. **updateMany()**

updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.

Example –

db.sales.orders.find() - to show current document in the system

db.sales.orders.updateMany({shipped_date:44377}, {$set: {shipped_date: "1-July-2021"}}) – with the help of this command we are updating “shipped_date” to "1-July-2021" where shipped_date is 44377



3. replaceOne()

The replaceOne() method is used to replace a single document in the specified collection. replaceOne() replaces the entire document, meaning fields in the old document not contained in the new will be lost.

4) Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.

MongoDB has two different methods of deleting records from a collection:

- db.collection.delete004Fne()
- db.collection.deleteMany()

1. deleteOne()

deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.

Example –

db.production.brands.find() - to show current documents in collection

db.production.brands.deleteOne({brand_id:6}) - delete the document where "brand_id" is 6

```
> db.production.brands.find()
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b8"), "brand_id" : 1, "brand_name" : "Hon
da" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b9"), "brand_id" : 2, "brand_name" : "6KU
 Bikes" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78ba"), "brand_id" : 3, "brand_name" : "Bia
nchi" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bb"), "brand_id" : 4, "brand_name" : "BMC
" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bc"), "brand_id" : 5, "brand_name" : "Huf
fy" }
> db.production.brands.deleteOne({brand_id:6})
{ "acknowledged" : true, "deletedCount" : 0 }
>
```

2. deleteMany()

deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in deleteOne().

Example –

db.production.brands.find() - to show current documents in collection

db.production.brands.deleteMany({brand_id: {$gt: 5}}) - delete documents for which brand id is greater than 5

Command Prompt - MONGO

```
> db.production.brands.deleteOne({brand_id:6})
{ "acknowledged" : true, "deletedCount" : 0 }
> db.production.brands.find()
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b8"), "brand_id" : 1, "brand_name" : "Hon
da" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78b9"), "brand_id" : 2, "brand_name" : "6KU
 Bikes" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78ba"), "brand_id" : 3, "brand_name" : "Bia
nchi" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bb"), "brand_id" : 4, "brand_name" : "BMC
" }
{ "_id" : ObjectId("6222e1dcb5e511acae1d78bc"), "brand_id" : 5, "brand_name" : "Huf
fy" }
> db.production.brands.deleteMany({brand_id: {$gt: 5}})
{ "acknowledged" : true, "deletedCount" : 0 }
>
```

Aim: Joins, Sorting, Subqueries using HiveQL

JOINS

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database.

There are different types of joins given as follows:

- **JOIN**
- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

➢ **JOIN**

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

➢ **LEFT OUTER JOIN**

The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

➢ **RIGHT OUTER JOIN**

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.
A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate

➢ **FULL OUTER JOIN**

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

SUB QUERIES:

A Query present within a Query is known as a sub query. The main query will depend on the values returned by the subqueries.

Subqueries can be classified into two types

- **Subqueries in FROM clause**
- **Subqueries in WHERE clause**

When to use:

- **To get a particular value combined from two column values from different tables**
- **Dependency of one table values on other tables**
- **Comparative checking of one column values from other tables**

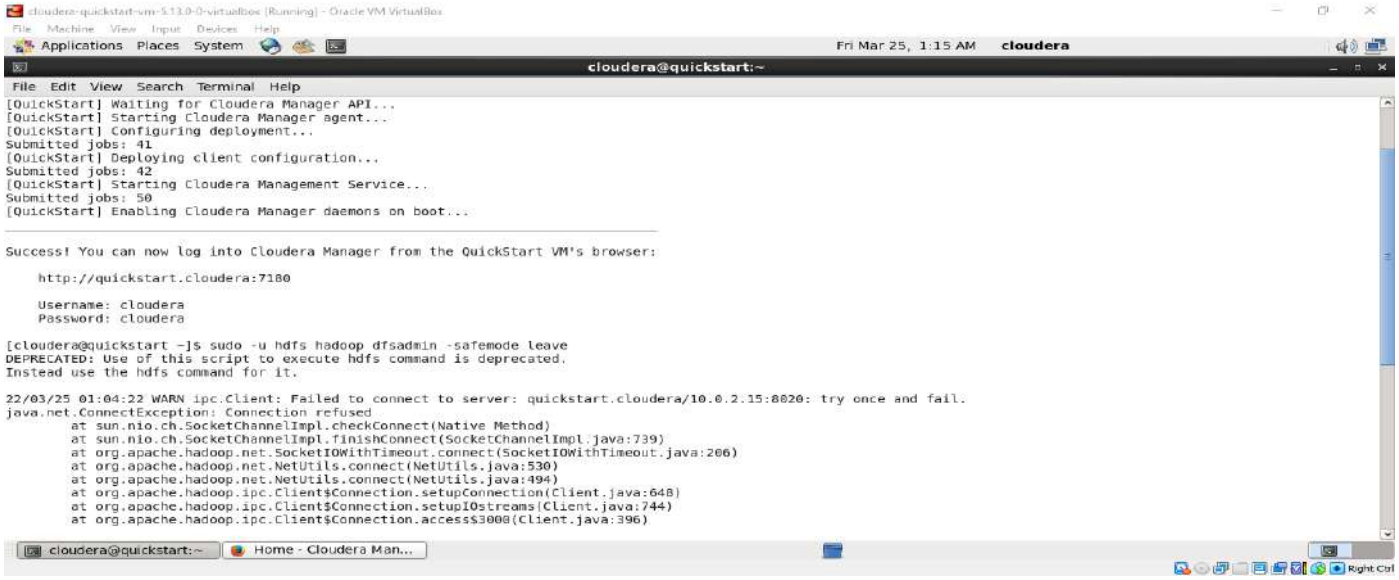
SORTING

The SORT BY syntax is similar to the syntax of ORDER BY in SQL language.

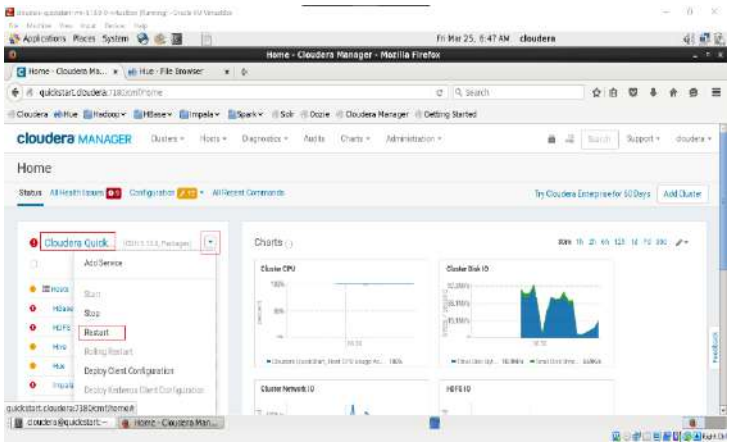
Hive supports SORT BY which sorts the data per reducer. The difference between "order by" and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, "sort by" may give partially ordered final results.
Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order

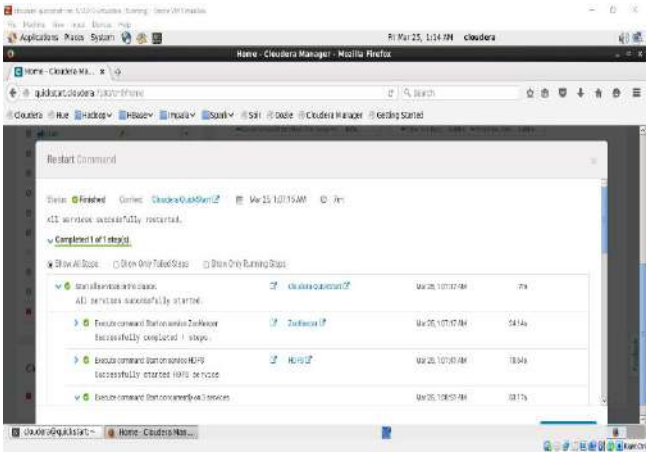
Before the perform the practical first perform 3 steps of the given following

1. **sudo /home/cloudera/cloudera-manager --force --express**

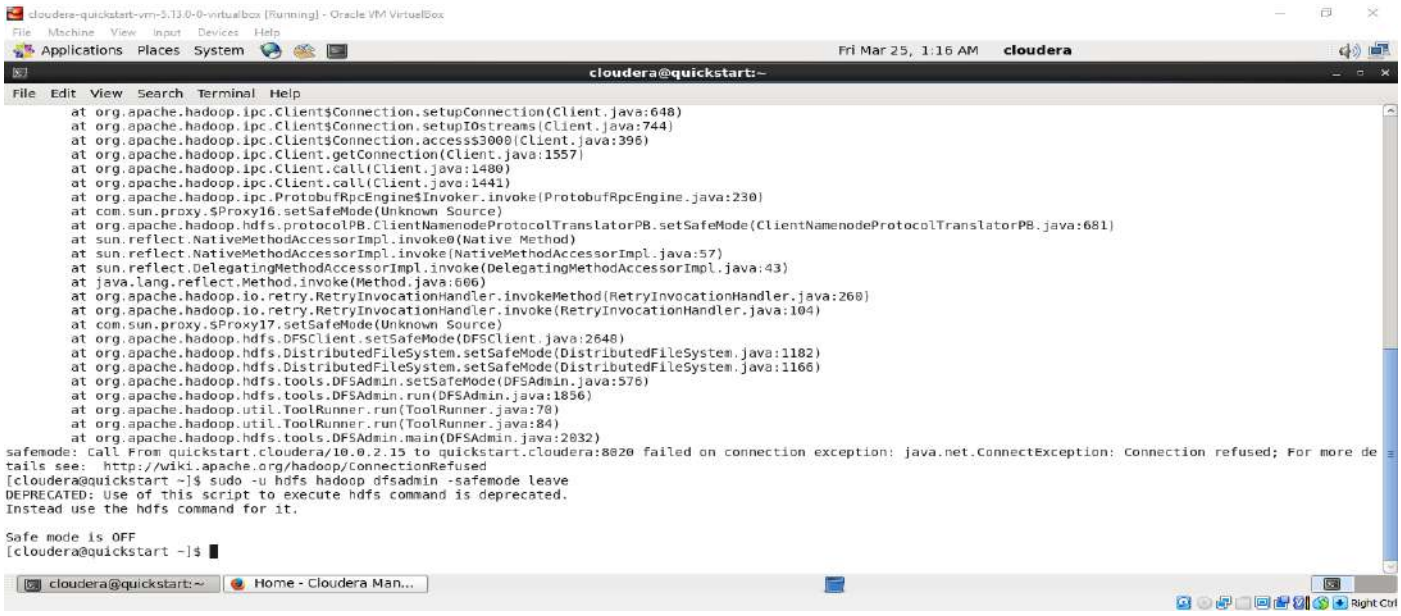


2. **Start all services**





3. **sudo -u hdfs hadoop dfsadmin -safemode leave**



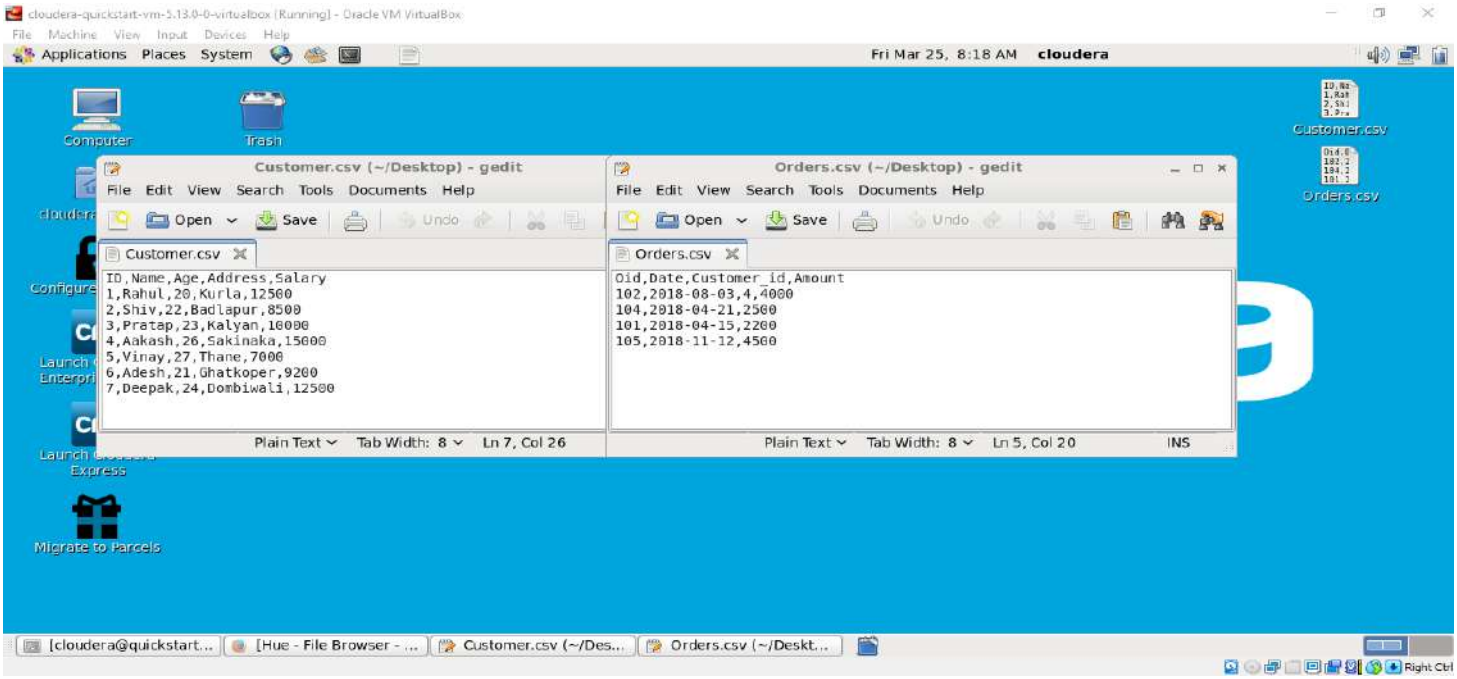
First we will create the **Customer.csv** and **Order.csv** file.

Customer.csv

ID,Name,Age,Address,Salary
1,Rahul,20,Kurla,12500
2,Shiv,22,Badlapur,8500
3,Pratap,23,Kalyan,10000
4,Aakash,26,Sakinaka,15000
5,Vinay,27,Thane,7000
6,Adesh,21,Ghatkoper,9200
7,Deepak,24,Dombiwali,12500

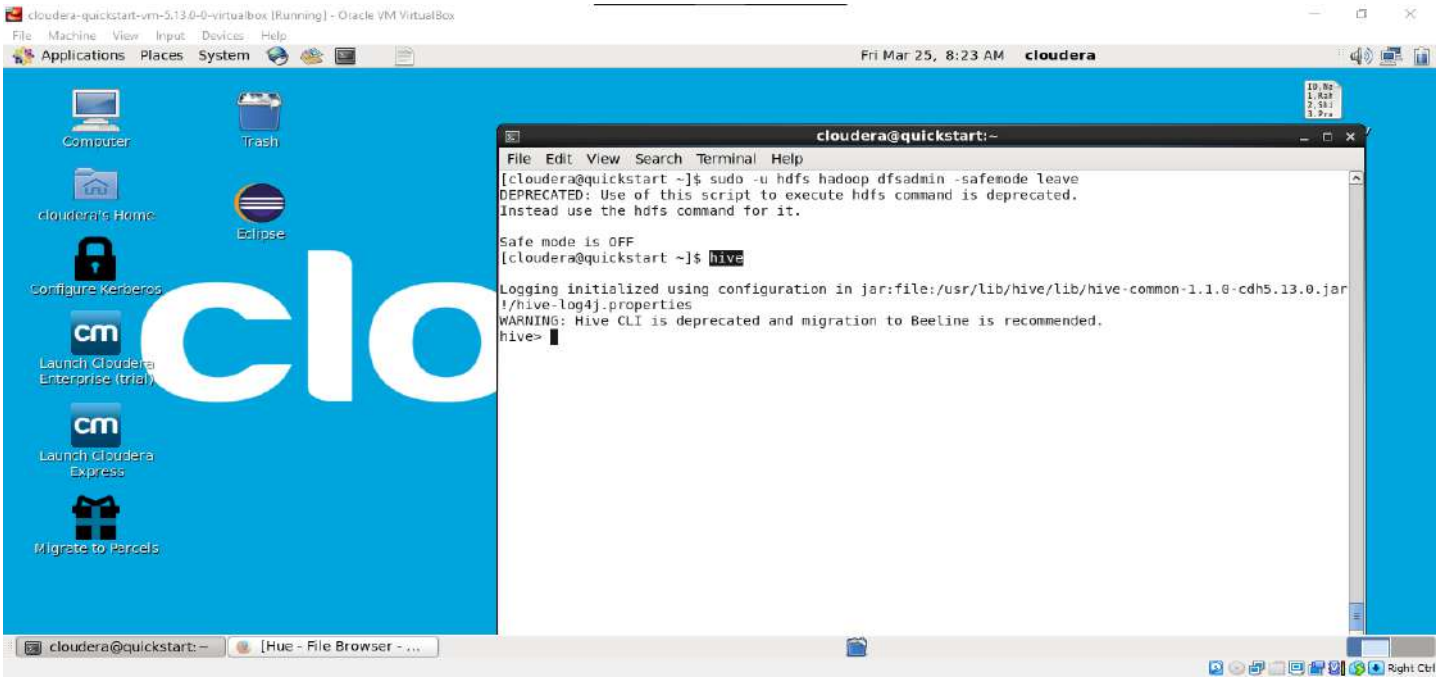
Orders.csv

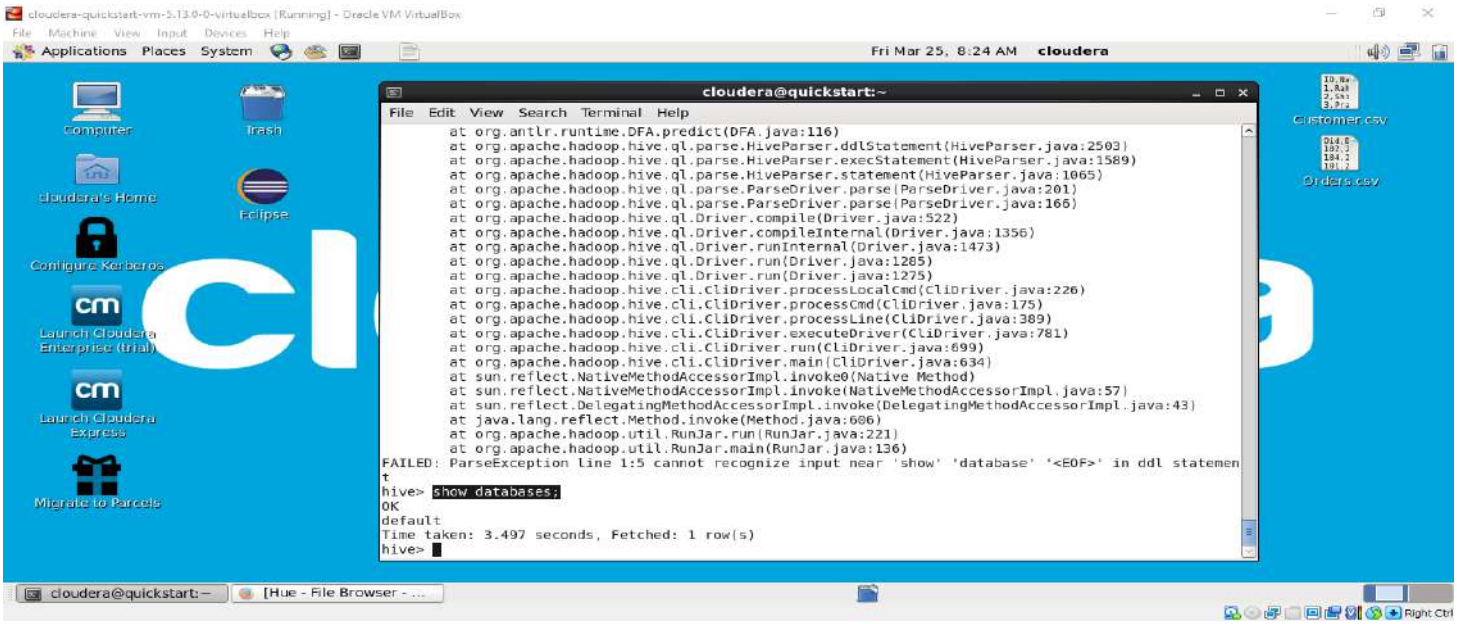
Oid,Date,Customer_id,Amount
102,2018-08-03,4,4000
104,2018-04-21,1,2500
101,2018-04-15,1,2200
105,2018-11-12,2,4500



Open the terminal, now we use **hive** command to enter the **hive shell prompt** and in hive shell we could execute all of the hive commands.

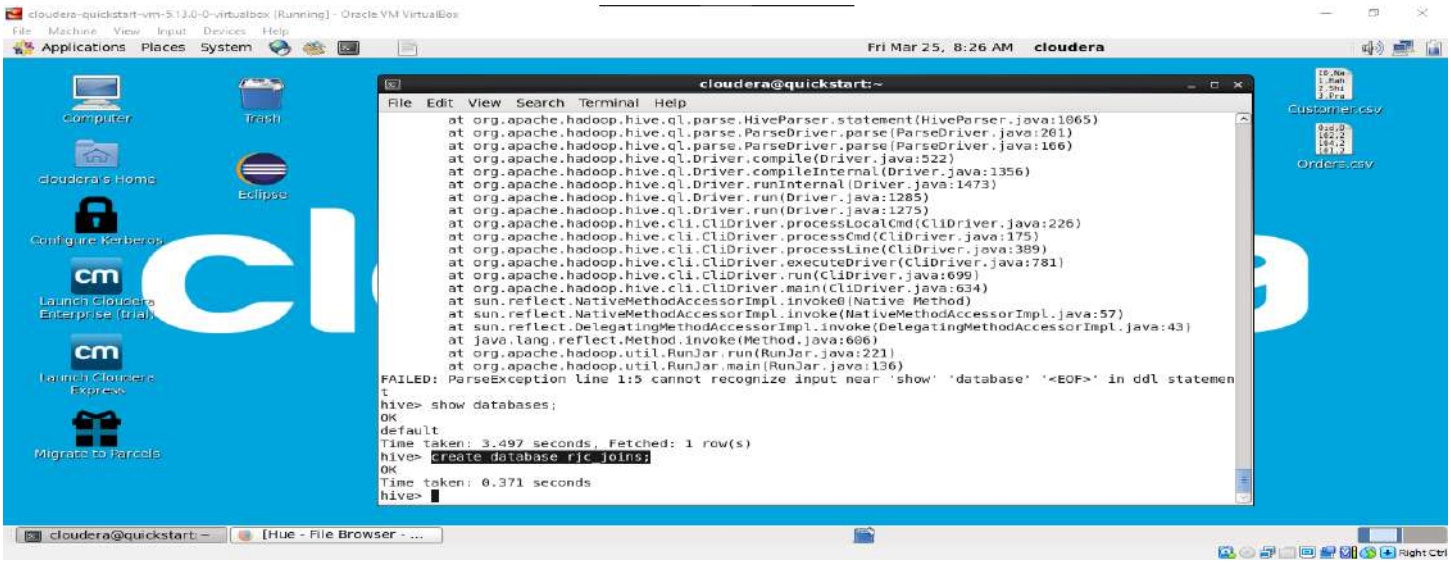
1. hive





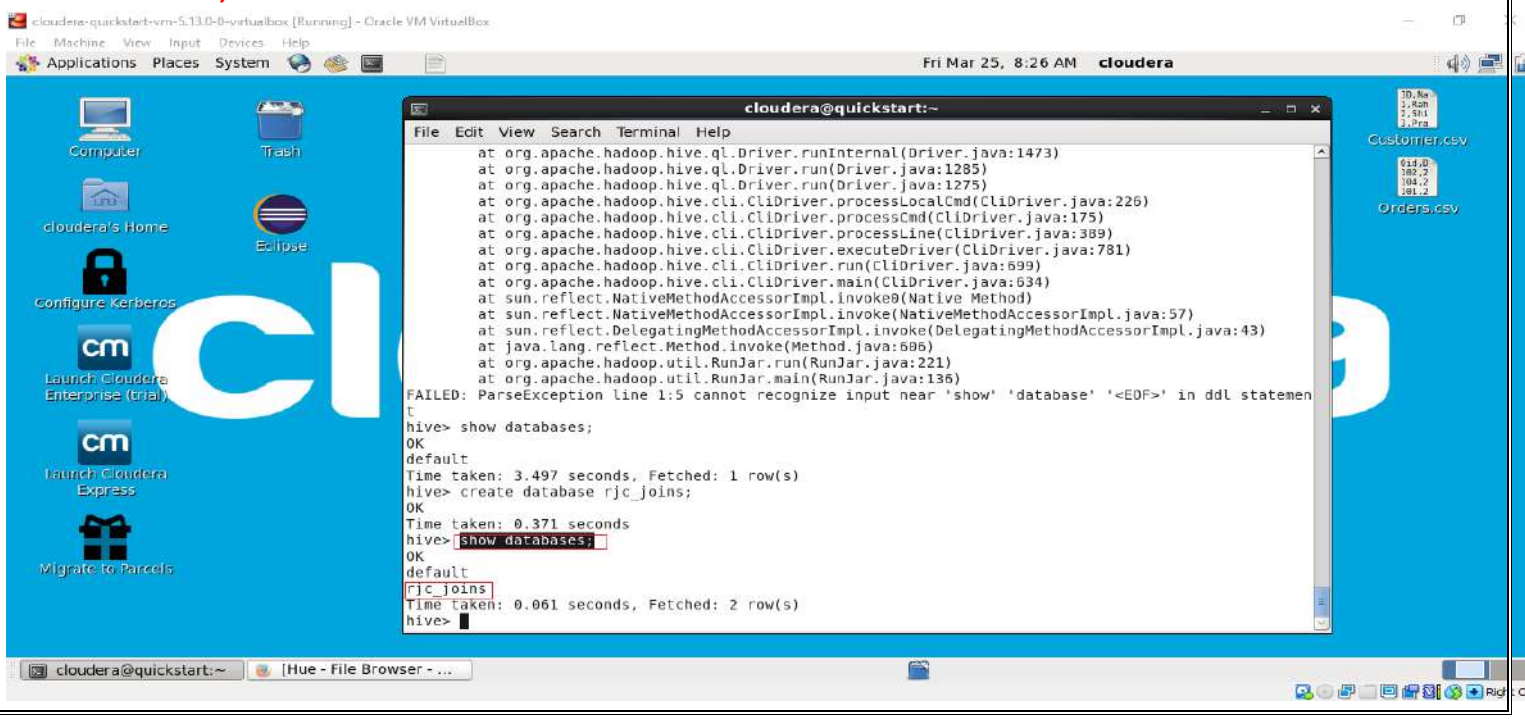
Now we will be creating a new database named as **rjc_joins** using below command,

2. create database rjc_joins;



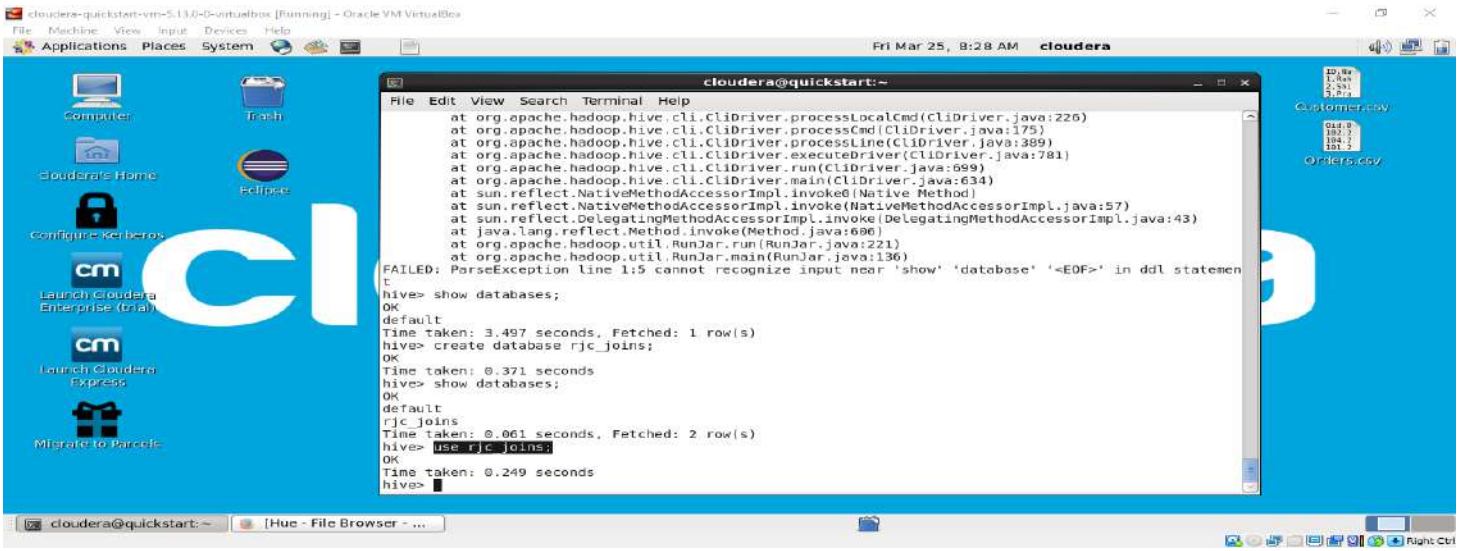
And then showing the databases.

show databases;



Now to work inside this database we use below command;

3. use rjc_joins;



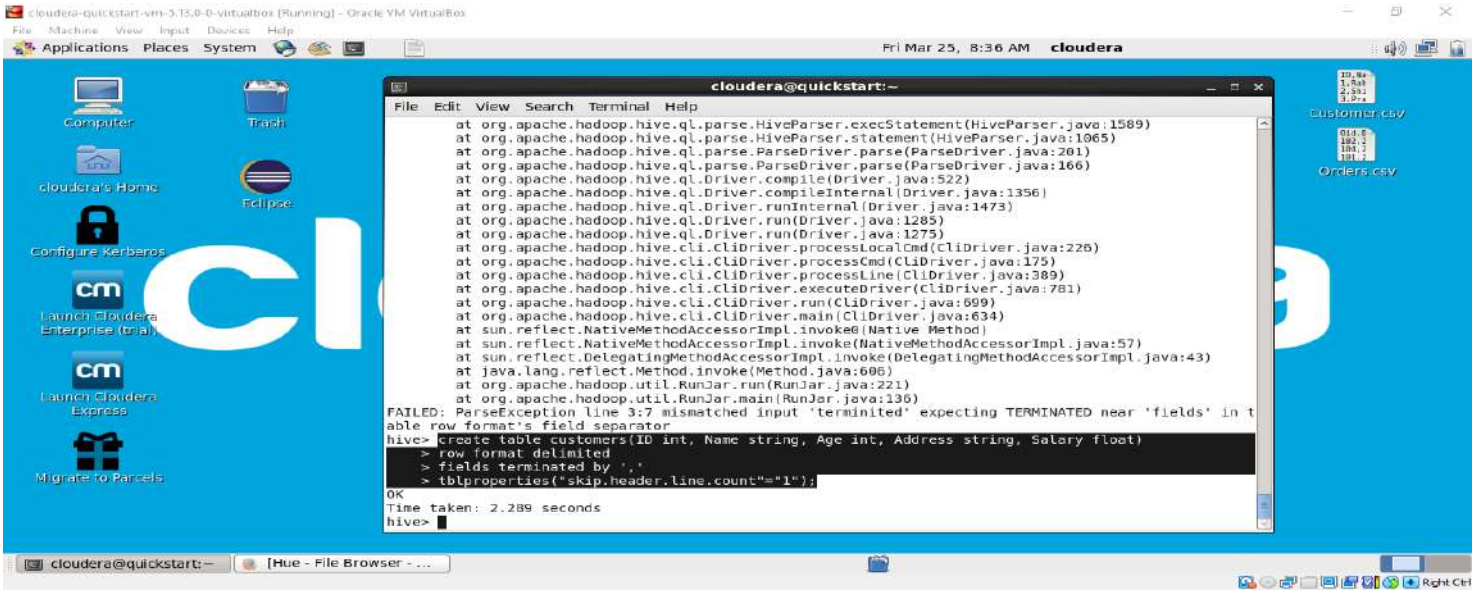
Now we will create <u>two tables</u> in one table we will load the **Customer.csv** file and in the other table we will load **Orders.csv** file.

4. create table customers(ID int, Name string, Age int, Address string, Salary float)

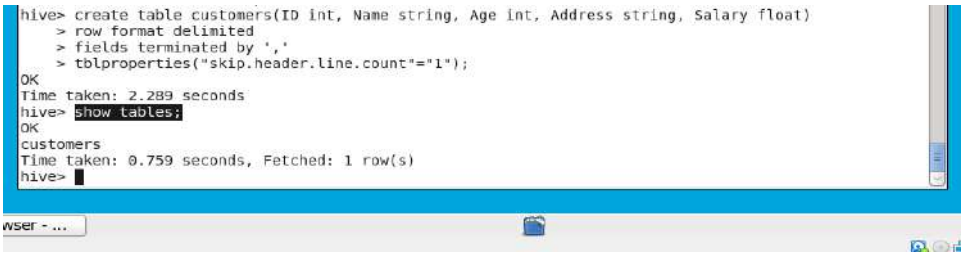
row format delimited

fields terminated by ','

tblproperties("skip.header.line.count"="1");

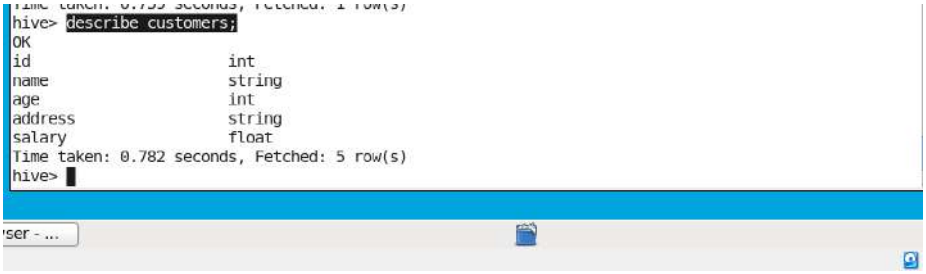


show tables;



Now we will see the schema of the table using describe command, **describe customers;**

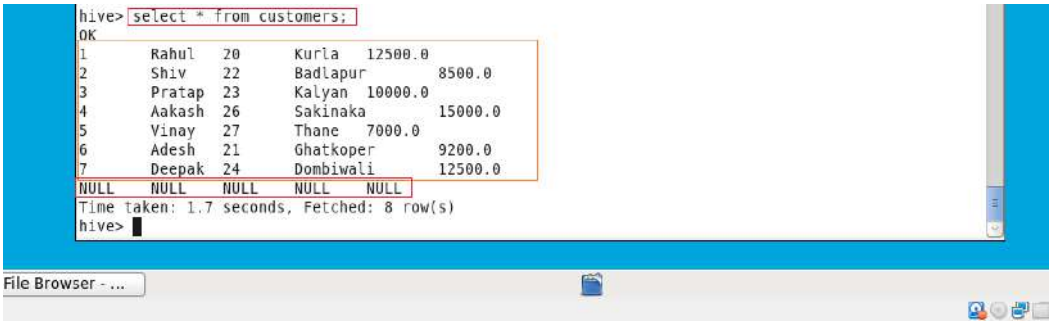
describe customers;



Now loading data in the **customers** table from **Customer.csv** file which present inside /home/cloudera/**<dir_name>** directory.

5. load data local inpath "/home/cloudera/Desktop/Customer.csv" into table customers;

select * from customers;



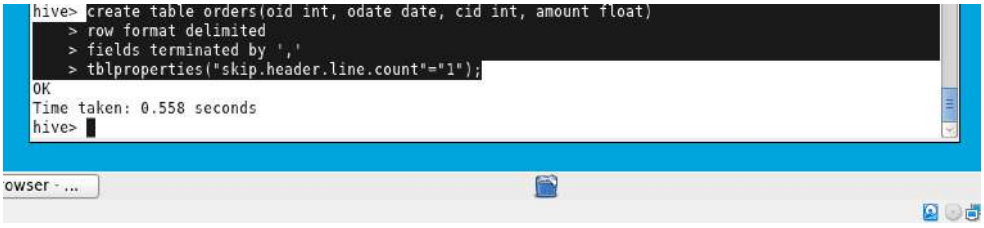
Creating a second table named as **orders** using below command,

6. create table orders(oid int, odate date, cid int, amount float)

row format delimited

fields terminated by ','

tblproperties("skip.header.line.count"="1");



Now we will see the schema of the table using **describe** command,

describe orders;

Now loading data in the **orders** table from **Orders.csv** file which present inside /home/cloudera/**<dir_name>** directory.

7. load data local inpath "/home/cloudera/Desktop/Orders.csv" into table orders;

```
hive> load data local inpath "/home/cloudera/Desktop/Orders.csv" into table orders
    > ;
Loading data to table rjc_joins.orders
Table rjc_joins.orders stats: [numFiles=1, totalSize=111]
OK
Time taken: 1.422 seconds
hive>
```

select * from orders;

```
hive> select * from orders;
OK
102     2018-08-03      4       4000.0
104     2018-04-21      2500    NULL
101     2018-04-15      2200    NULL
105     2018-11-12      4500    NULL
Time taken: 0.381 seconds, Fetched: 4 row(s)
hive> truncate table orders;
OK
Time taken: 1.605 seconds
hive> load data local inpath "/home/cloudera/Desktop/Orders.csv" into table orders;
Loading data to table rjc_joins.orders
Table rjc_joins.orders stats: [numFiles=1, numRows=0, totalSize=116, rawDataSize=0]
OK
Time taken: 3.16 seconds
hive> select * from orders;
OK
102     2018-08-03      4       4000.0
104     2018-04-21      1       2500.0
101     2018-04-15      1       2200.0
105     2018-11-12      2       4500.0
Time taken: 0.851 seconds, Fetched: 4 row(s)
hive>
```

JOIN

Now first we apply the normal joins on the two tables using below command, we want to retrieve customer id, name, age from customers table and amount from the orders table and join perform on id of the customers and orders table.

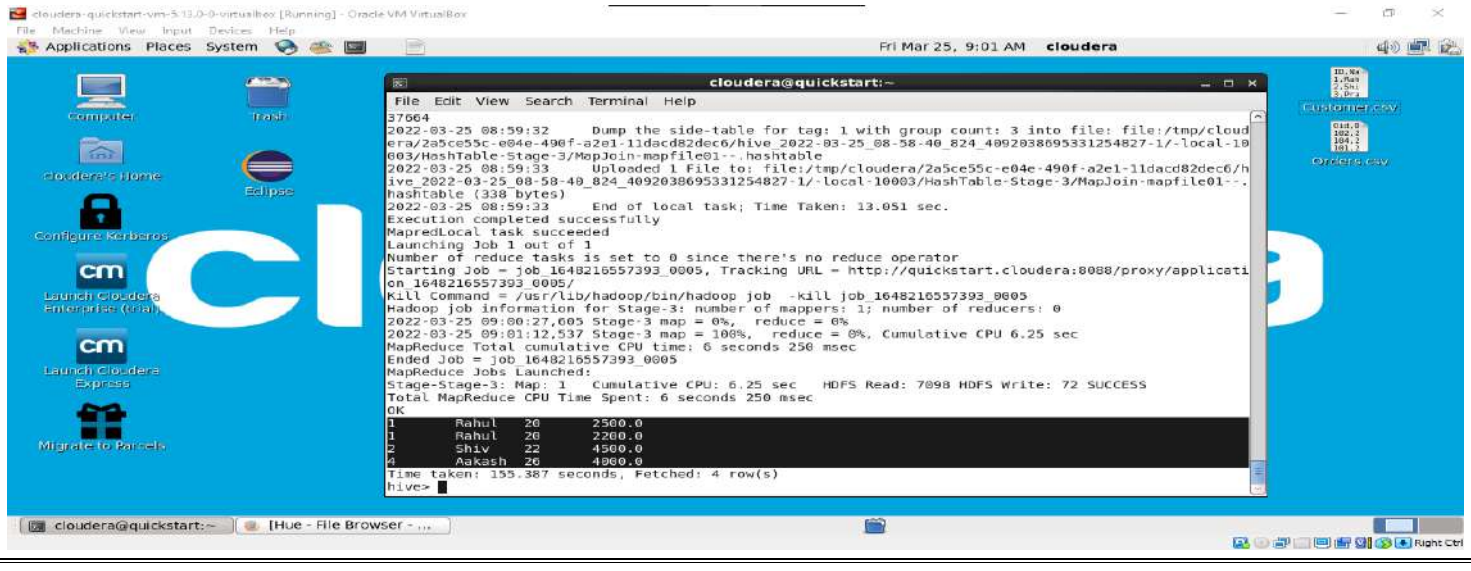
8. select c.id, c.name, c.age, o.amount

from customers c JOIN orders o

on (c.id = o.cid);

```
hive> select c.id, c.name, c.age, o.amount
    > from customers c JOIN orders o
    > on (c.id = o.cid);
Query ID = cloudera_20220325085858_6ac28064-2d67-4188-9d19-e62b7ee0c754
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20220325085858_6ac28064-2d67-4188-9d19-e62b7ee0c754.log
```

Mapreduce task is performed



LEFT OUTER JOIN

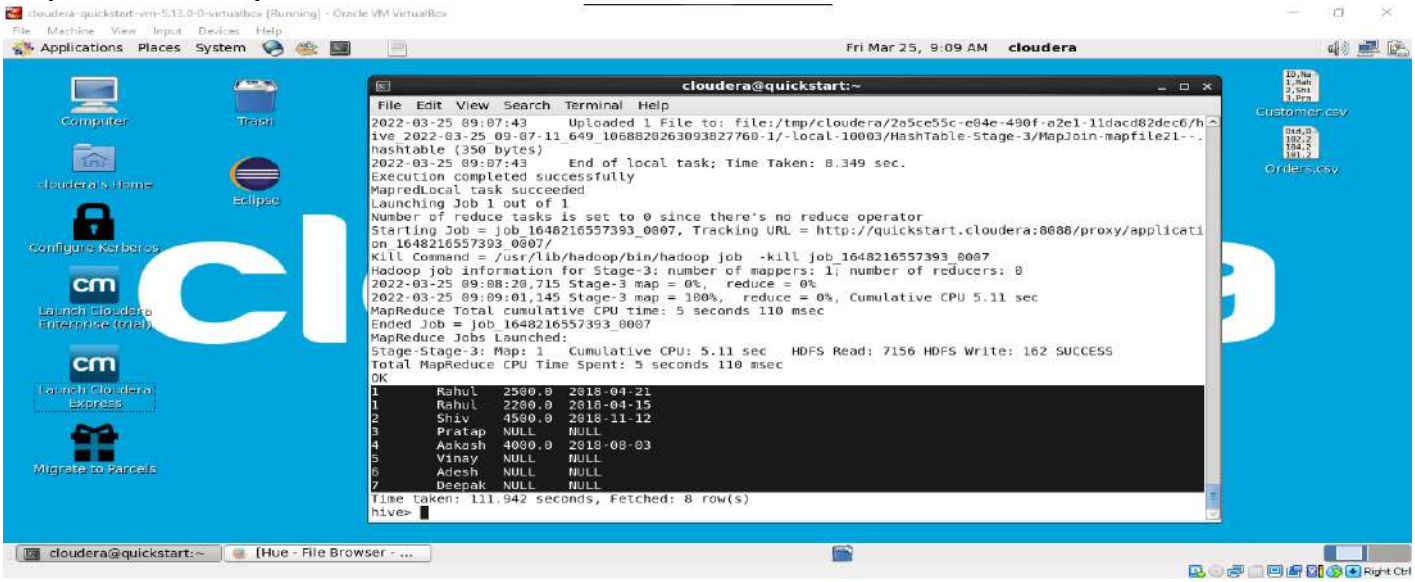
The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

9. select c.id, c.name, o.amount,o.odate
from customers c LEFT OUTER JOIN orders o
on (c.id = o.cid);

```
hive> select c.id, c.name, o.amount,o.odate
    > from customers c LEFT OUTER JOIN orders o
    > on (c.id = o.cid);
Query ID = cloudera_20220325090202_75101a32-531d-4e2c-9d17-1e716cf37995
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20220325090202_75101a32-531d-4e2c-9d17-1e716cf37995.log
2022-03-25 09:03:25     Starting to launch local task to process map join;      maximum memory = 1195
37664
```

Mapreduce task is performed



RIGHT OUTER JOIN

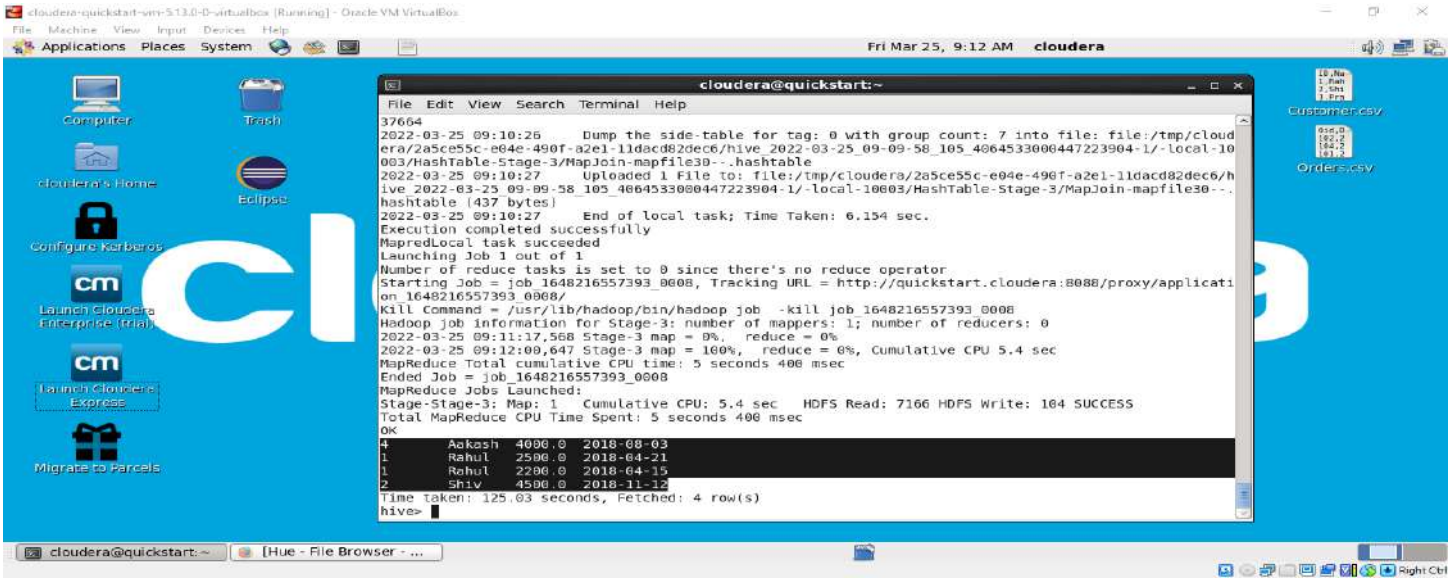
The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

10. select c.id, c.name, o.amount,o.odate
from customers c RIGHT OUTER JOIN orders o
on (c.id = o.cid);

```
hive> select c.id, c.name, o.amount,o.odate
    > from customers c RIGHT OUTER JOIN orders o
    > on (c.id = o.cid);
Query ID = cloudera_20220325090909_f1d30aa1-55a3-4835-ab4c-d7263d18f42d
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20220325090909_f1d30aa1-55a3-4835-ab4c-d7263d18f42d.log
2022-03-25 09:10:20     Starting to launch local task to process map join;      maximum memory = 1195
37664
2022-03-25 09:10:26     Dump the side-table for tag: 0 with group count: 7 into file: file:/tmp/cloud
era/2a5ce55c-e04e-490f-a2e1-11dacd82dec6/hive_2022-03-25_09-09-58_105_4064533000447223904-1/-local-10
003/HashTable-Stage-3/MapJoin-mapfile30--.hashtable
```

Mapreduce task is performed



Now we will be using the concept of **subqueries** for finding the second largest salary from the customers table.

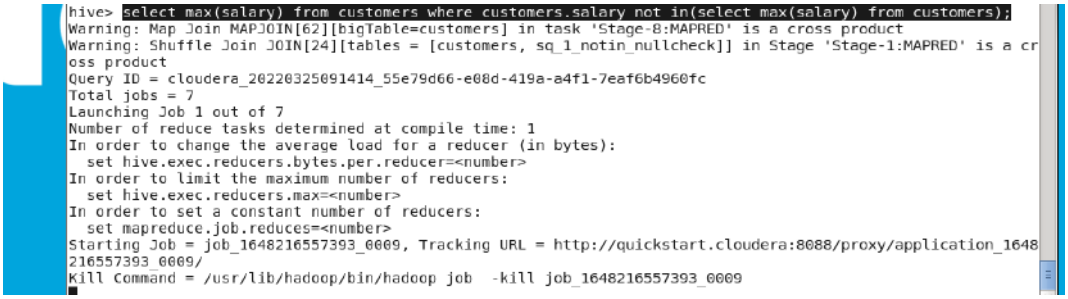
Sub Queries

A Query present within a Query is known as a sub query. The main query will depend on the values returned by the subqueries.

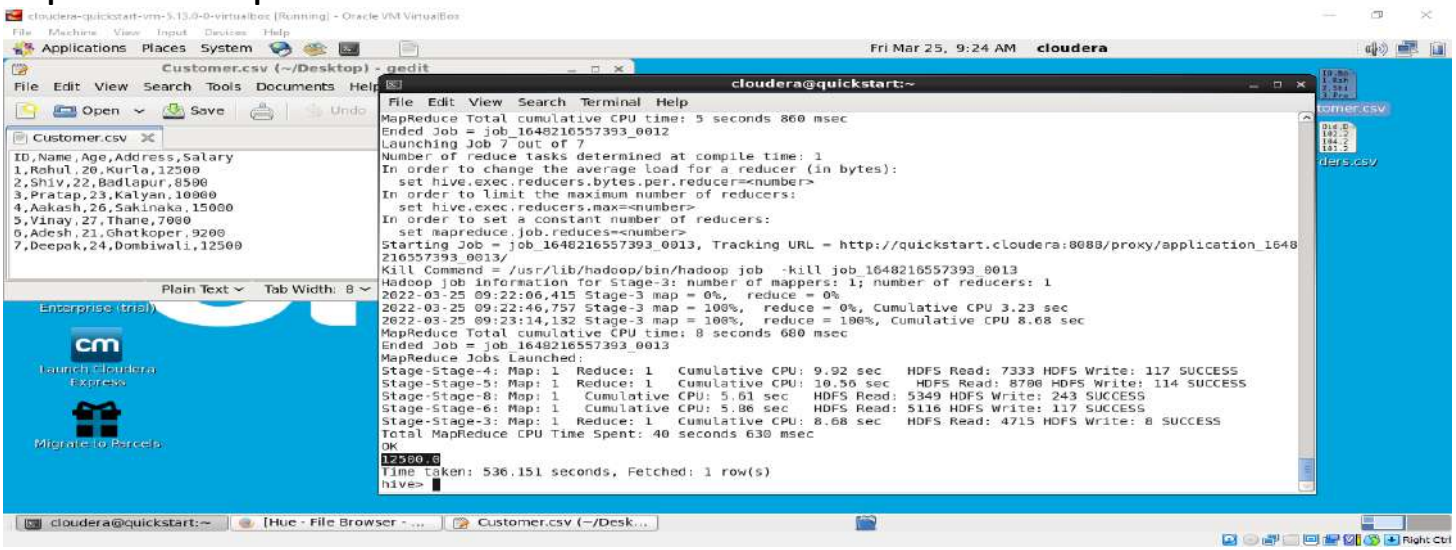
Subqueries can be classified into two types

- Subqueries in FROM clause
- Subqueries in WHERE clause

11. select max(salary) from customers where customers.salary not in(select max(salary) from customers);



Mapreduce task is performed



As we can see from the above output the **second largest salary** is **12500. 00.**

Sorting

The SORT BY syntax is similar to the syntax of **ORDER BY** in SQL language.

Hive supports **SORT BY** which sorts the data per reducer. The difference between "order by" and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, “sort by" may give partially ordered final results.

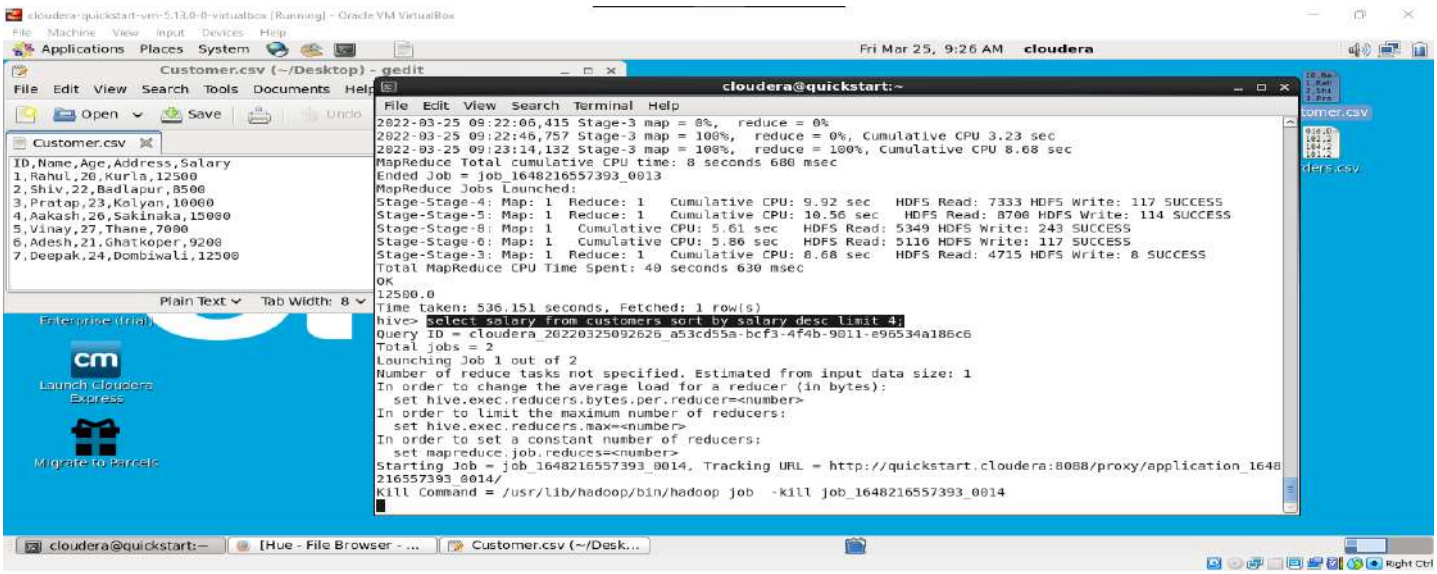
Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order.

LIMIT can be used to minimize sort time.

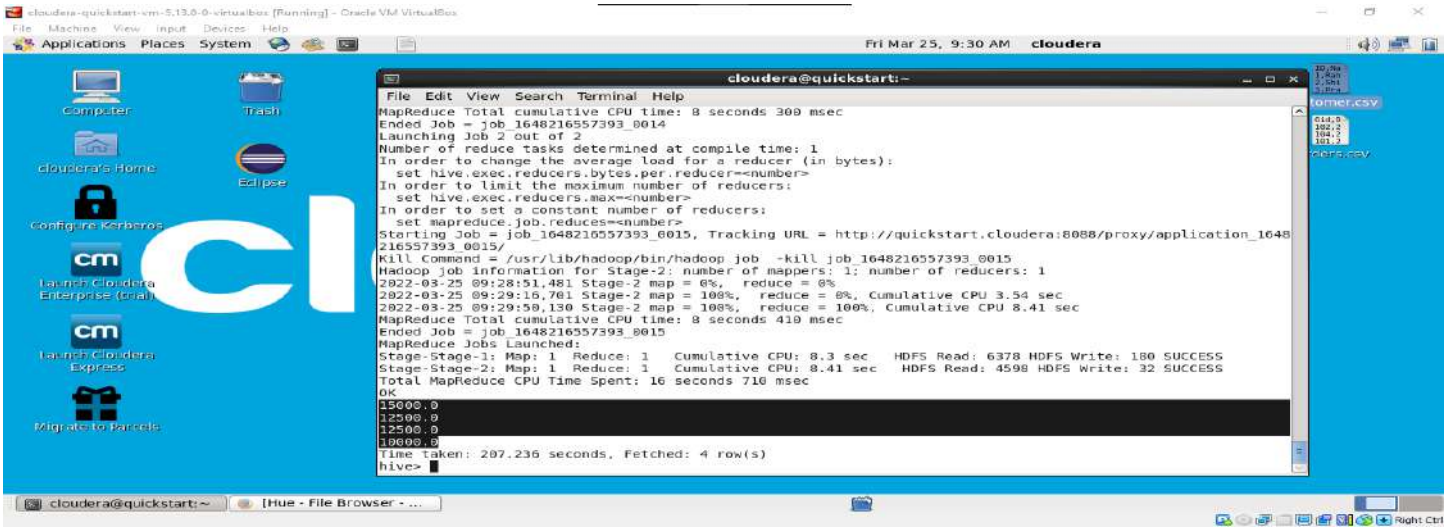
Now finding the **fourth largest salary** from the customers table using **Sort by** clause.

12. select salary from customers sort by salary desc limit 4;

It will give the only 4 records in the output after sorting them in descending order. This is not a complete syntax only we are showing what output it will give.



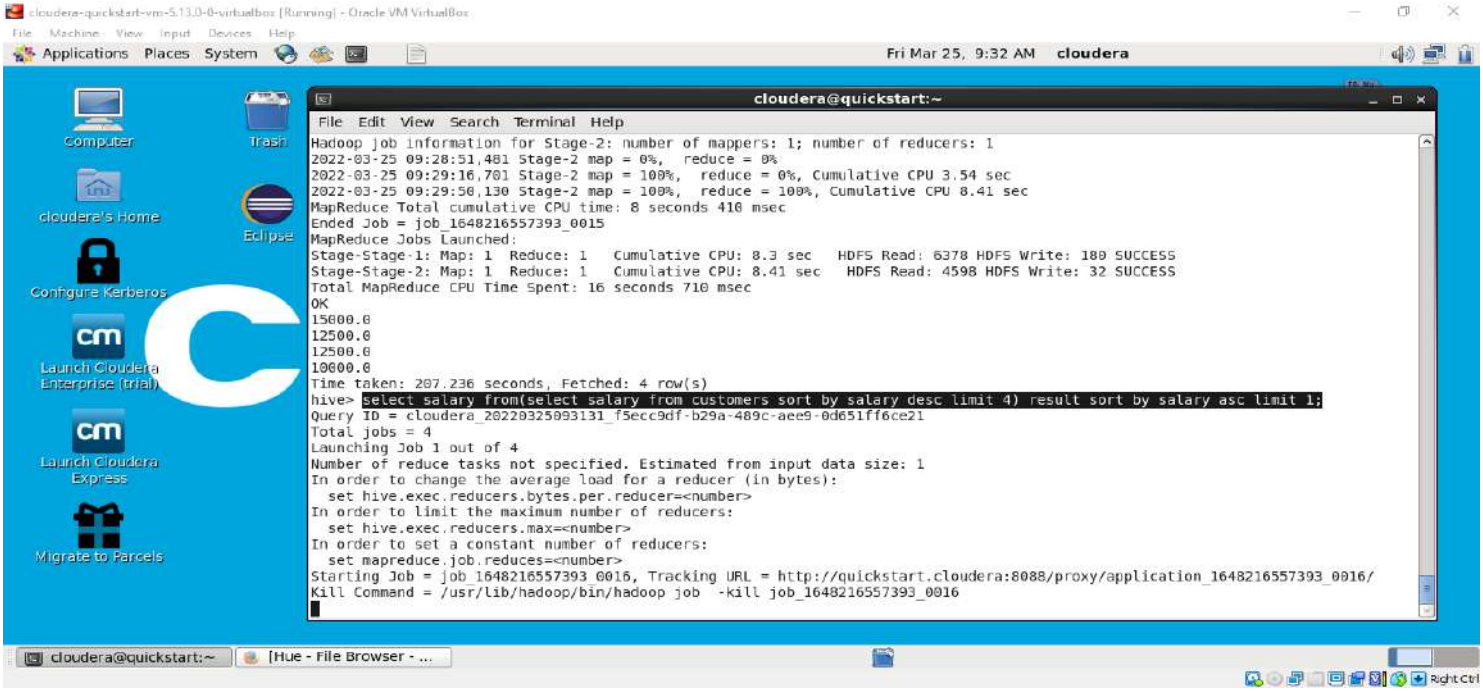
Mapreduce task is performed

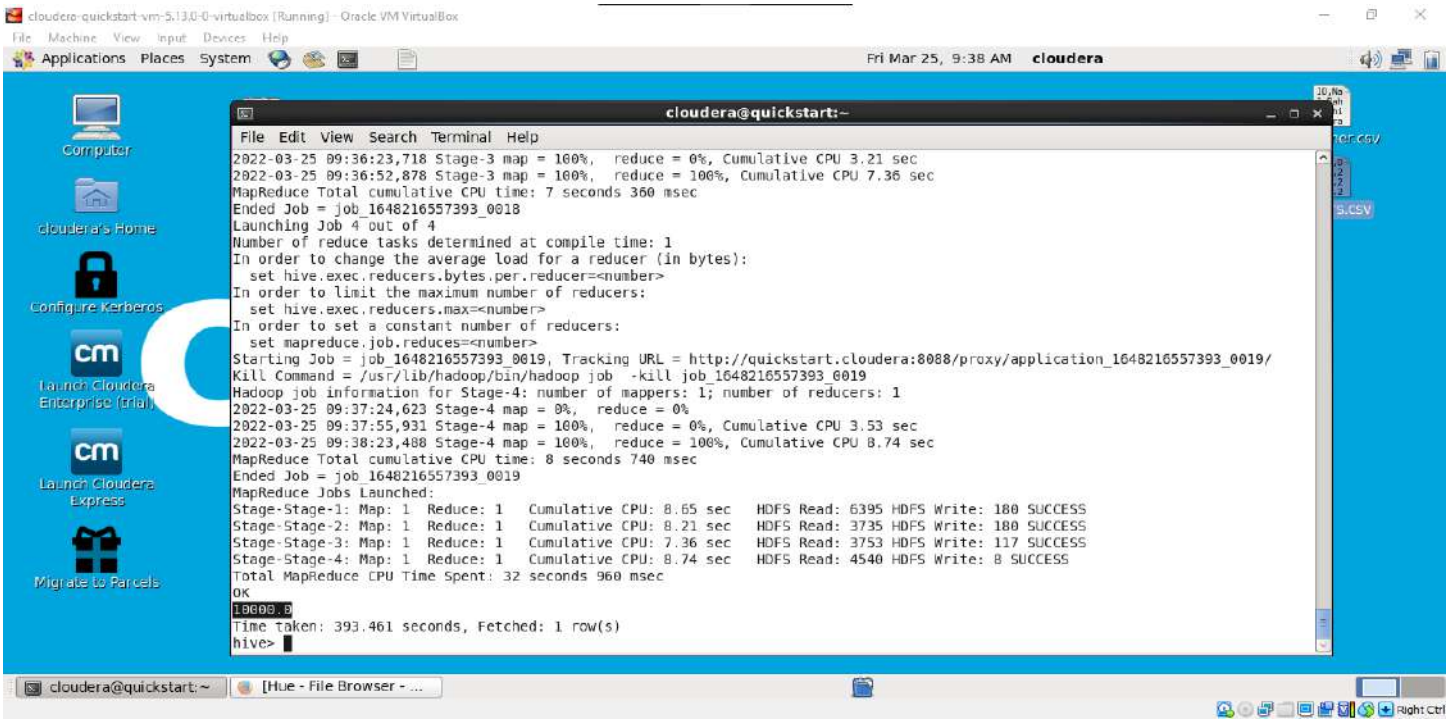


Now what records which we have got by executing the above queries now we will use this query as subqueries and we will now sort them in ascending order to find fourth largest salary of customer table.

13. select salary from(select salary from customers sort by salary desc limit 4) result sort by salary asc limit 1;

Now whatever result we get from subquery we will store them in result table and then it will sort the result table in ascending order and as we want fourth largest salary so we are limiting it to 1.





Now we got the fourth largest salary i.e. 10000.0 as an output.

Aim: Querying, Sorting, Aggregating data using HiveQL

What is HIVE:

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Features of Hive:

These are the following features of Hive:

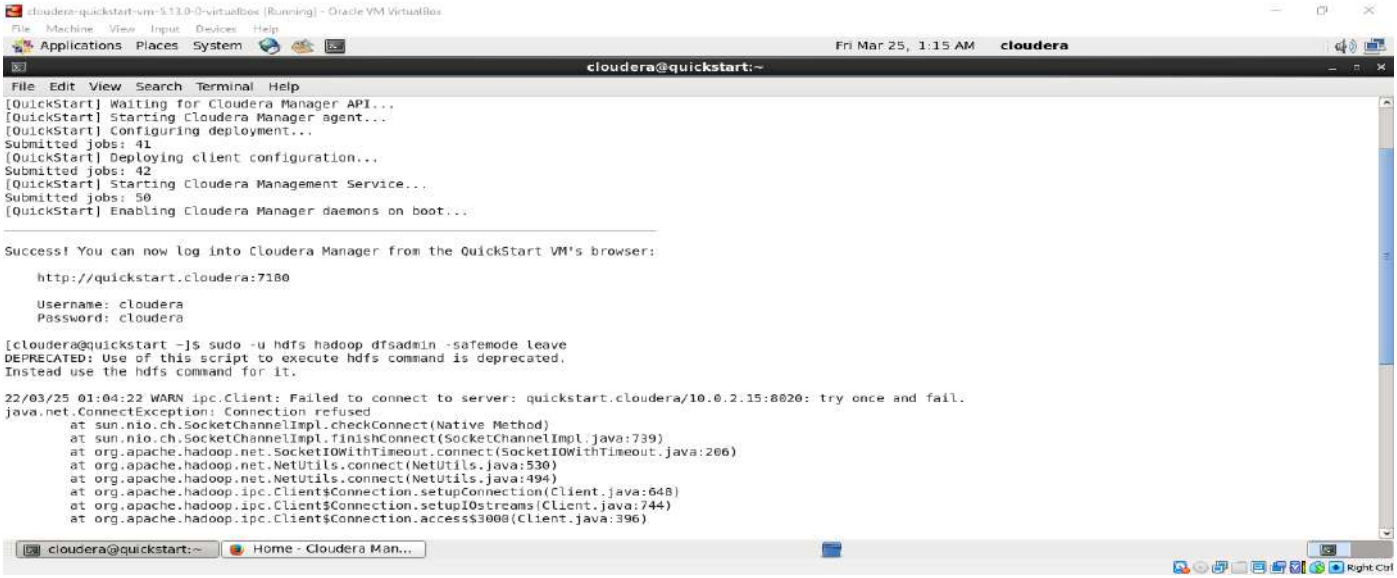
- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

Limitations of Hive

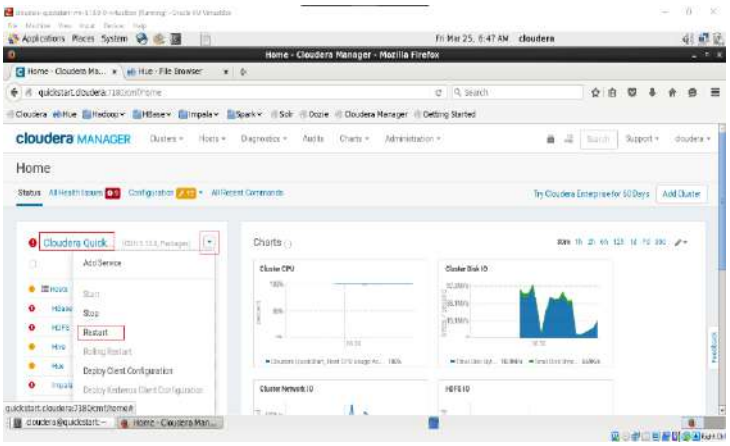
- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

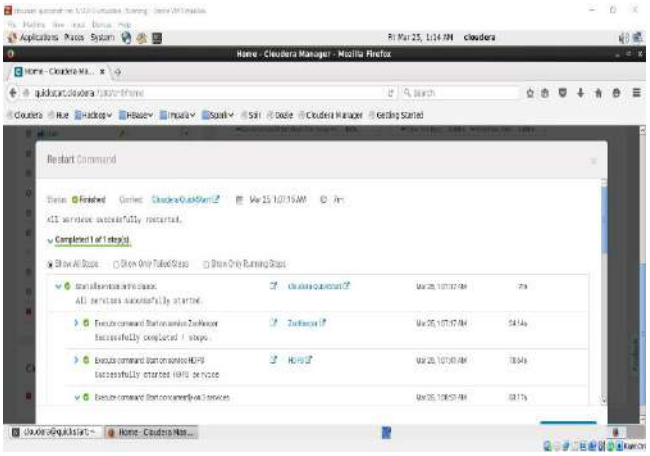
Before the perform the practical first perform <u>3 steps</u> of the given following

1. sudo /home/cloudera/cloudera-manager --force --express

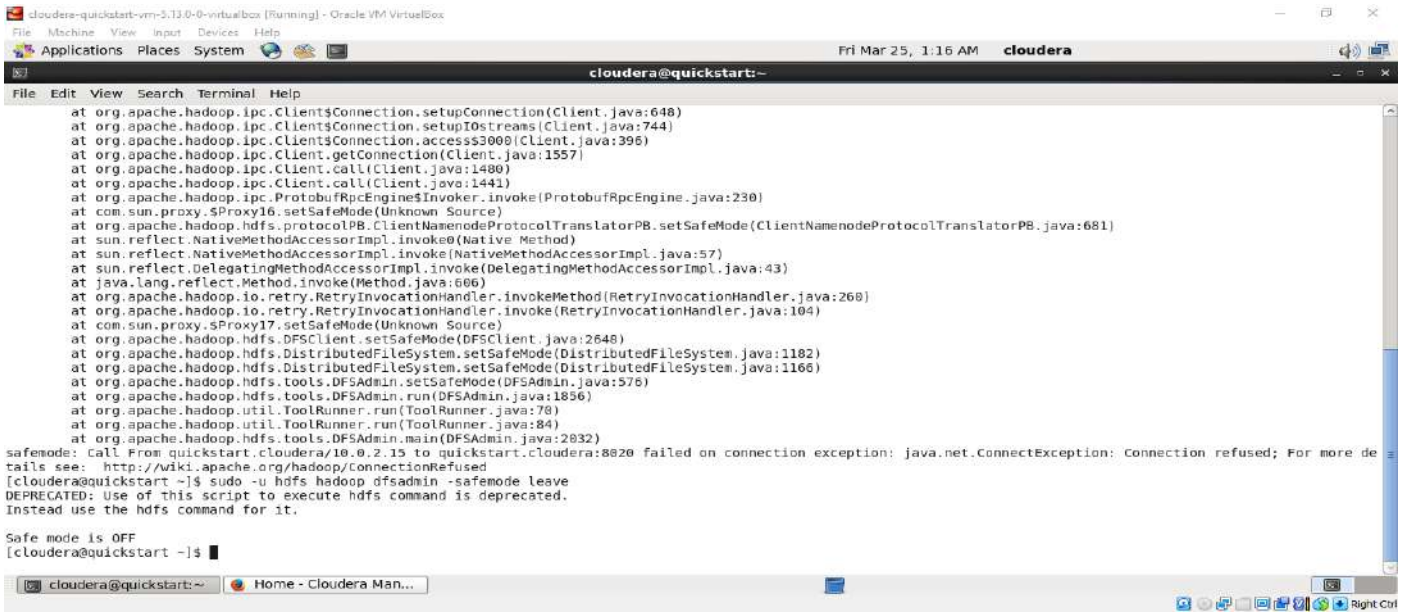


2. Start all services





3. sudo -u hdfs hadoop dfsadmin -safemode leave



1. **Open the terminal, Now we use hive command to enter the hive shell prompt and in hive shell we could execute all of the hive commands.**

hive

2. **Now we will see the databases which are already existing using below command.**

show databases;

Note: If remove database already present in hive then drop database <database_name> cascade;

3. **Creating a database name 'RJC' using below command.**

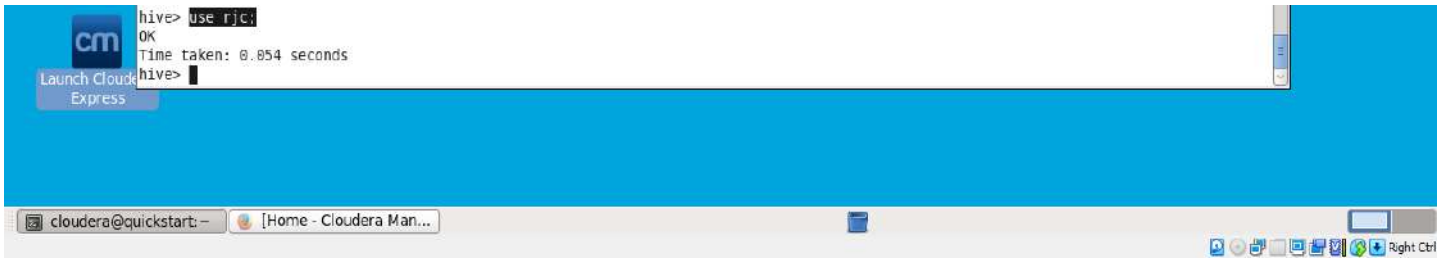
create database RJC;

4. **So if we want to see whether this RJC database is created or not we will use below command,**

show databases;

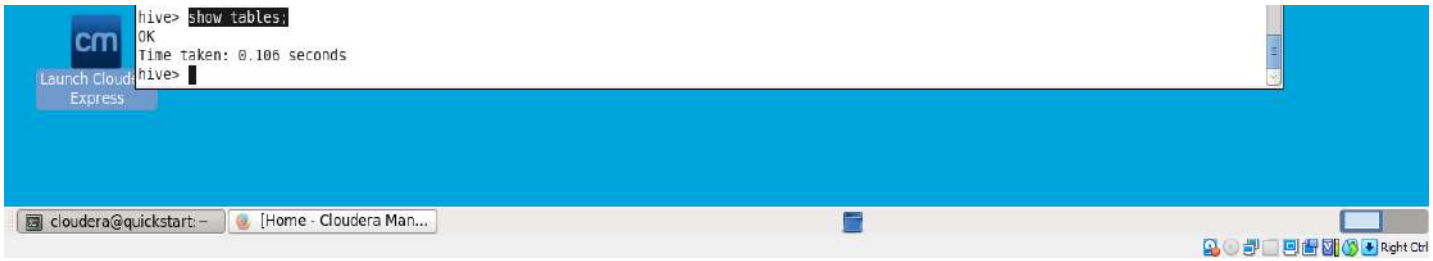
5. **Now we want to check whether we have any tables inside this rjc database or not. So first we will move to this databse rjc using below command,**

use rjc;



6. **Now we have moved inside this rjc database. Now we will check out which are the tables available using below command,**

show tables;



So as we can see from the above output it give us OK as output because there are no tables created inside this rjc database. So as there are no tables we did not get anything in the output we simply got as OK.

No will drop this 'rjc' database using below command we do not mention here cascade because there are no tables in this and so there are no data available or present inside this rjc database.

drop database rjc;

and to check whether the data base dropped or not using below command,

show databases;

7. **Creating a database "rjc".**

   create database rjc;

   Now let's move to this database using below command so now all the work we will do or perform it should be done within this rjc database.

   use rjc;

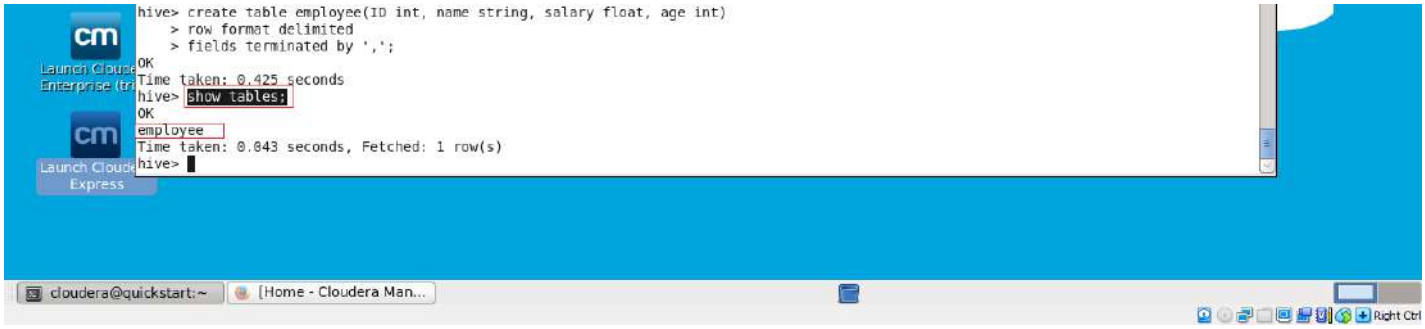
8. **Now we will create a table inside this rjc database named as employee using below command,**

   create table employee(ID int, name string, salary float, age int)

   Note: After this we will not put semicolon , When we will be loading the data from some existing csv file or maybe some other text files so we have to mention that how that data has to be loaded here. We are simply creating the schema of the table with some certain fields or attributes along with their datatypes and then I'm mentioning

   row format delimited

   fields terminated by ',';



row format delimited means , every record is present in one row and fields terminated by ',' and fields are terminated by comma. So as soon as it encounters one comma so that means that one is the value of some field and after comma it is encountering abc so that abc is the value of some another field. By default it is a "tab" character that means fields are separated by "tab".

9. **So now we will see the schema of the table using below command,**

   describe employee;

It will give different fields of table employee along with their respective datatypes.

10. **By default the internal table would store in the warehouse directory of hive. Whereas the external tables are available in the hdfs. And if we drop the internal table so then the table data and the metadata associated with that table will be deleted from the hdfs. Whereas when we drop the external table then only the metadata associated with that table will be deleted whereas the table data will be untouched by hive as it would be residing in the hdfs and it would be outside the warehouse directory of the hive. So Now we will check how the table which we will be created is internal table or external table using below command,**

describe formatted employee;

By default hive creates Internal table or Managed Table.

11. **Now we will create the external table using below command,**

Note: external

Do this

```
create external table employee2(ID int, name string, salary float, age int)
    row format delimited
    fields terminated by ','
    stored as textfile;
```

```
Time taken: 0.749 seconds
hive> create external table employee2(ID int, name string, salary float, age int)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;
OK
Time taken: 0.19 seconds
hive>
```

cloudera@quickstart:~ | Hue - Editor - Mozilla F...

Don't this

```
create table employee2(ID int, name string, salary float, age int)
    row format delimited
    fields terminated by ','
    stored as textfile;
```

```
hive> create table employee2(ID int, name string, salary float, age int)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;
OK
Time taken: 0.158 seconds
hive>
```

cloudera@quickstart:~ | [Home - Cloudera Man...

12. **Checking the schema of the table using below command,**

describe employee2;

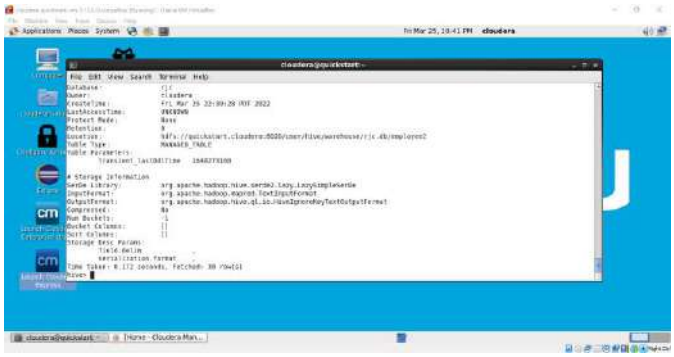
```
hive> describe employee2;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.147 seconds, Fetched: 4 row(s)
hive>
```

cloudera@quickstart:~ | [Home - Cloudera Man...

13. So Now we will check how the table which we will be created is internal table or external table using below command,

describe formatted employee2;

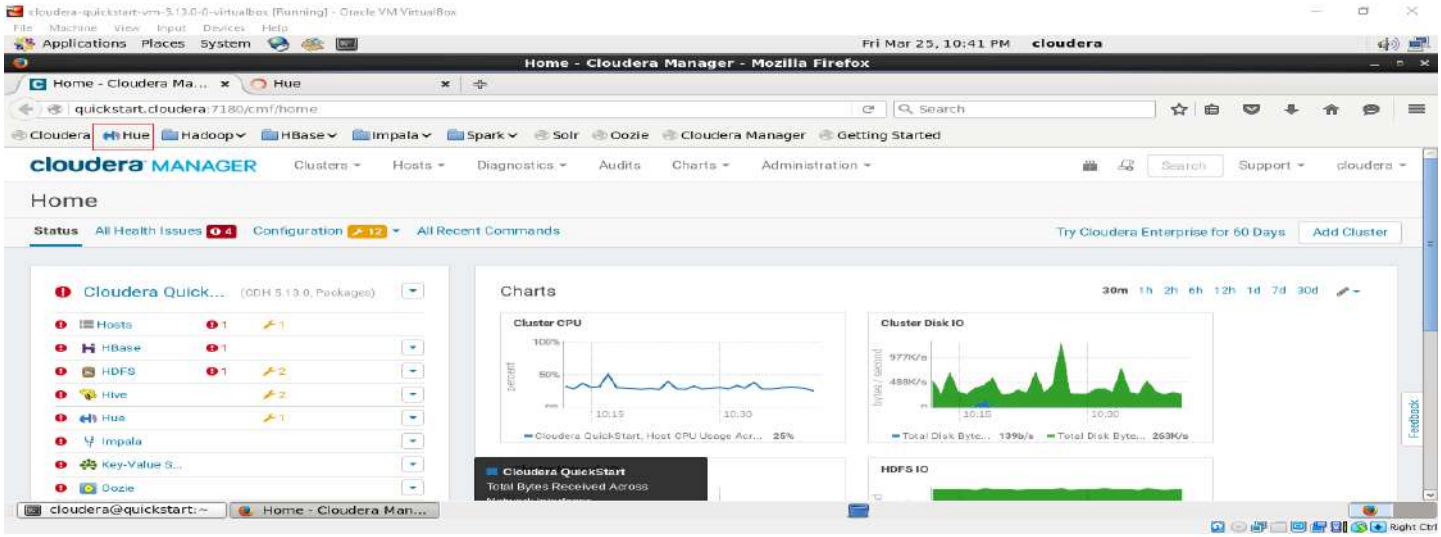
Don't **Do**



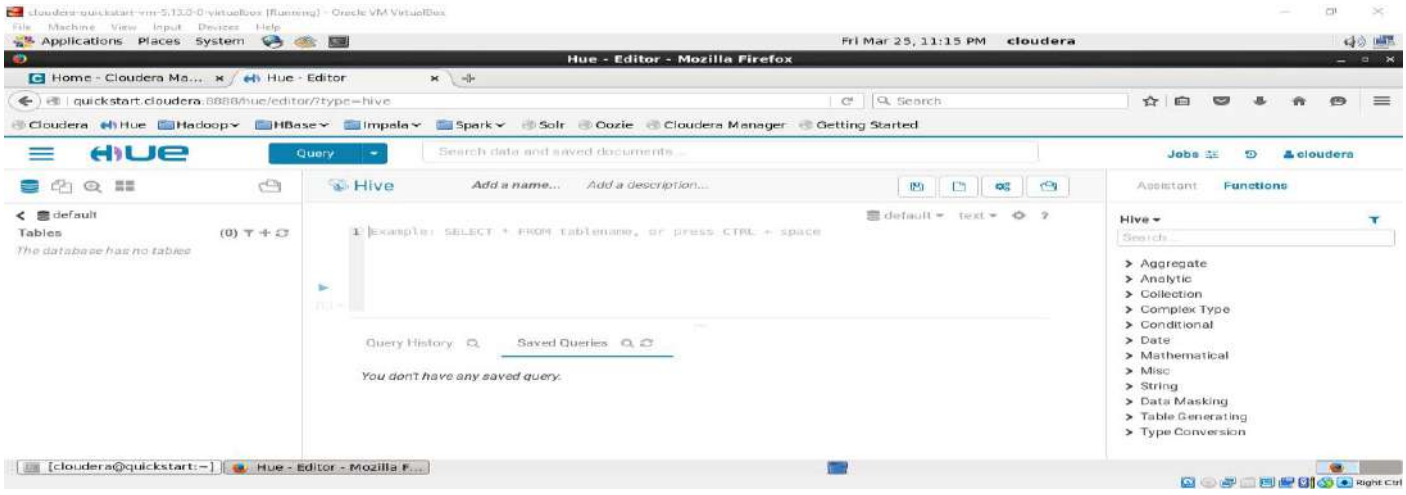
It is an Managed table. (If not write external) **It is an External table.**

14. So whatever we have done in terminal the same thing we can see in the browser as well.

Open the browser → click on Hue



Then click on Query->Editor->Hive

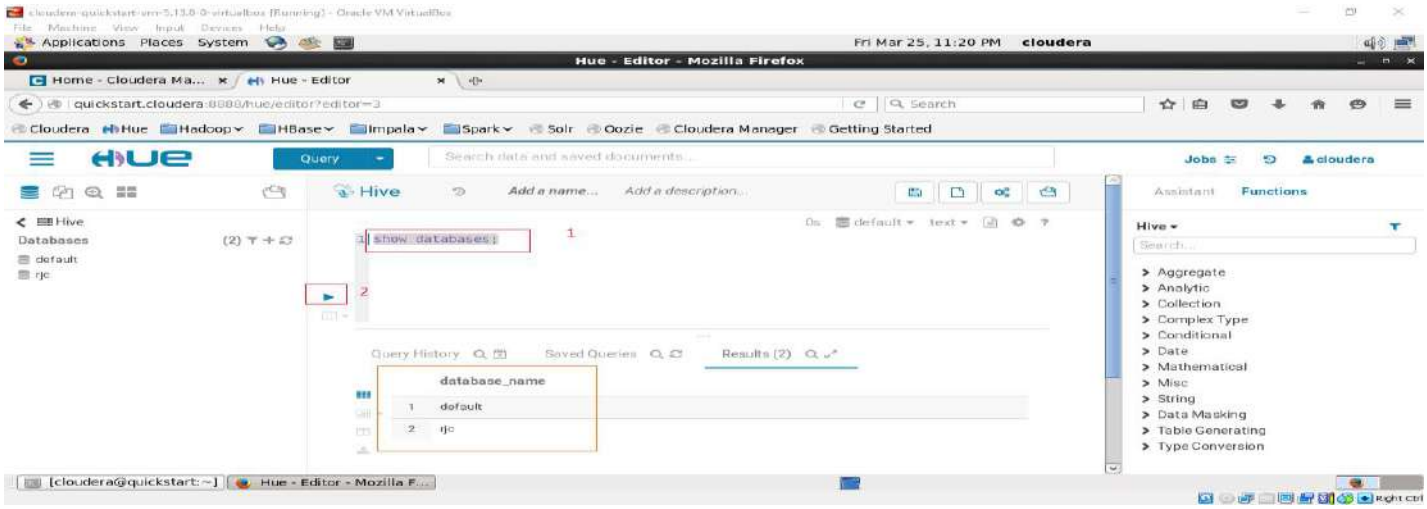


Write the query in query editor. Here we are showing the databases by using below command,

show databases;

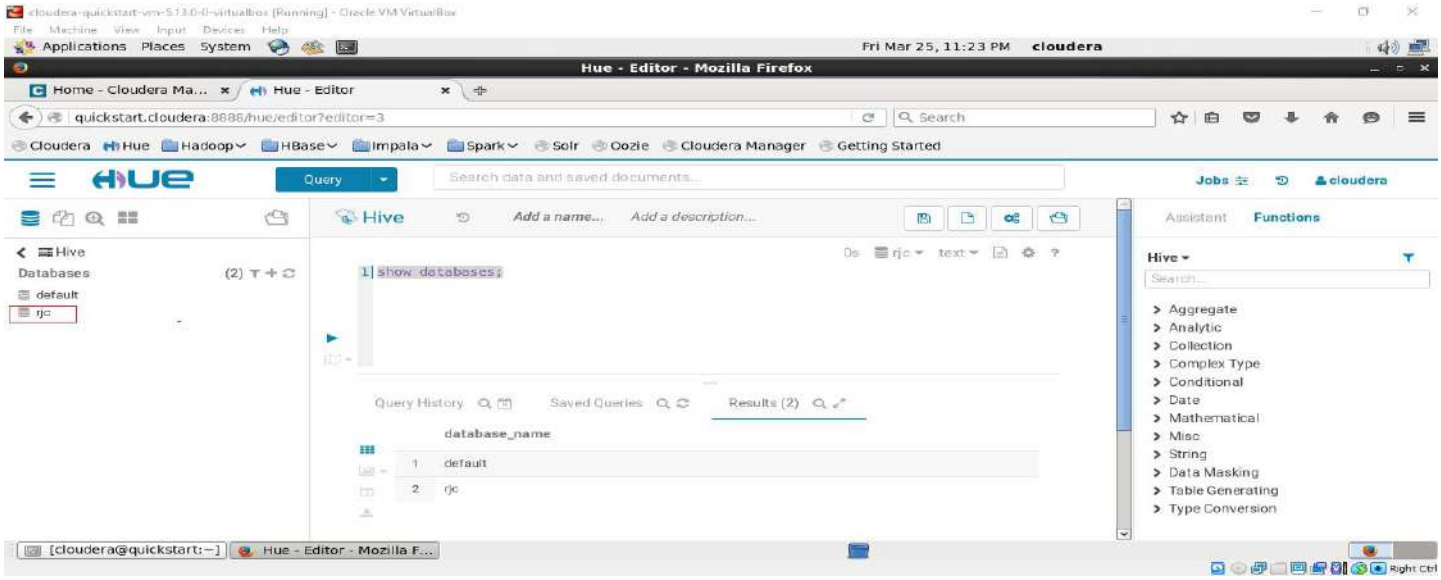
To execute -> CTRL + ENTER

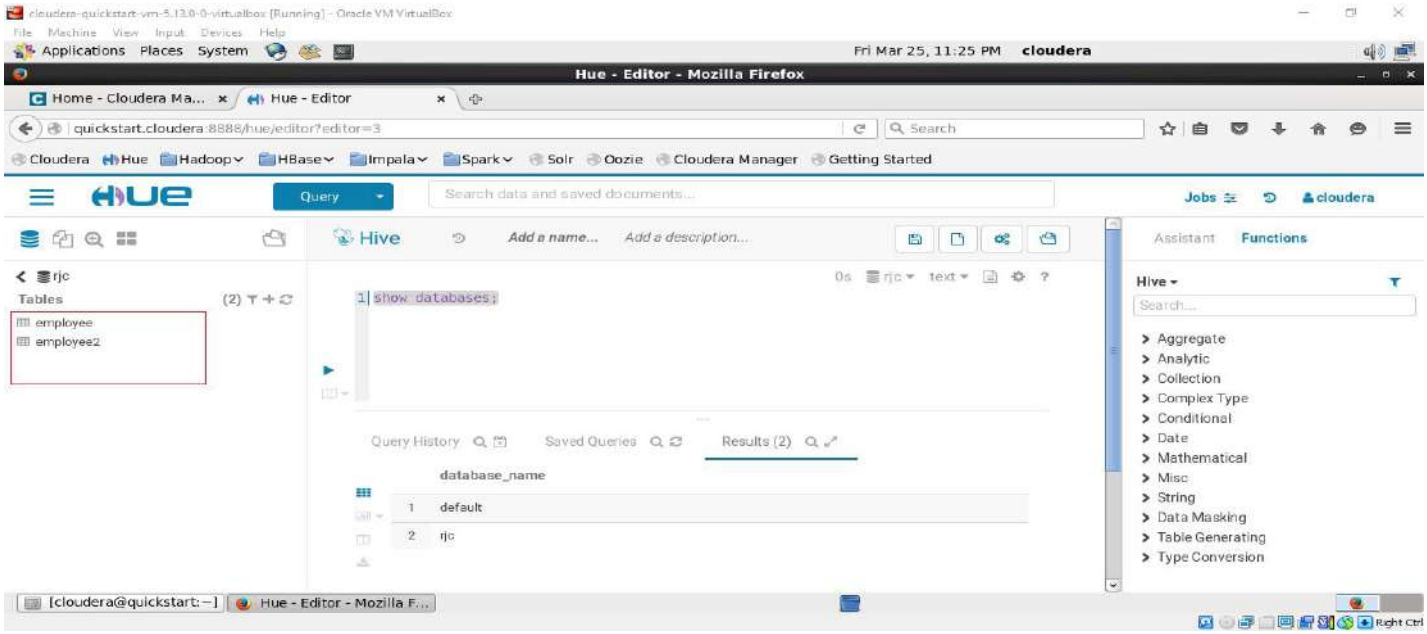
It will give the list of databases which are present.



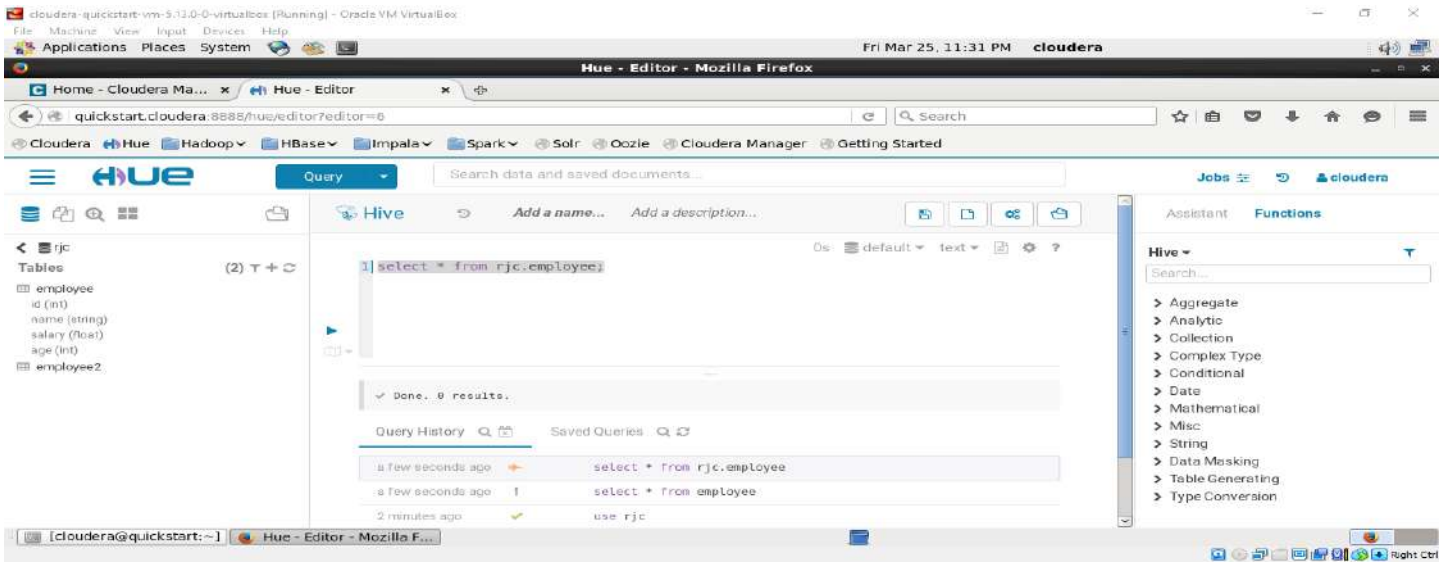
There are only two databases present i.e. default and rjc which we created earlier. We can also see the list of databases in left side corner. And after clicking on the database name here it is "rjc" it will give the list of all tables present inside "rjc" database.

Click on the rjc database

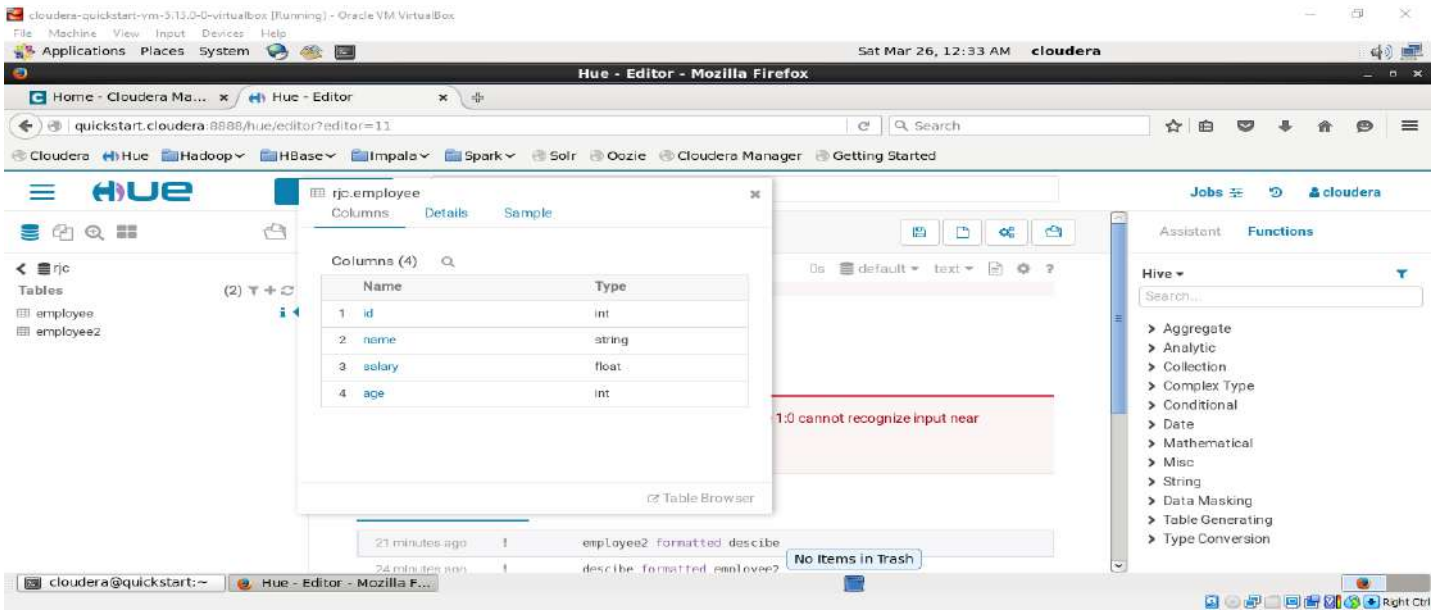


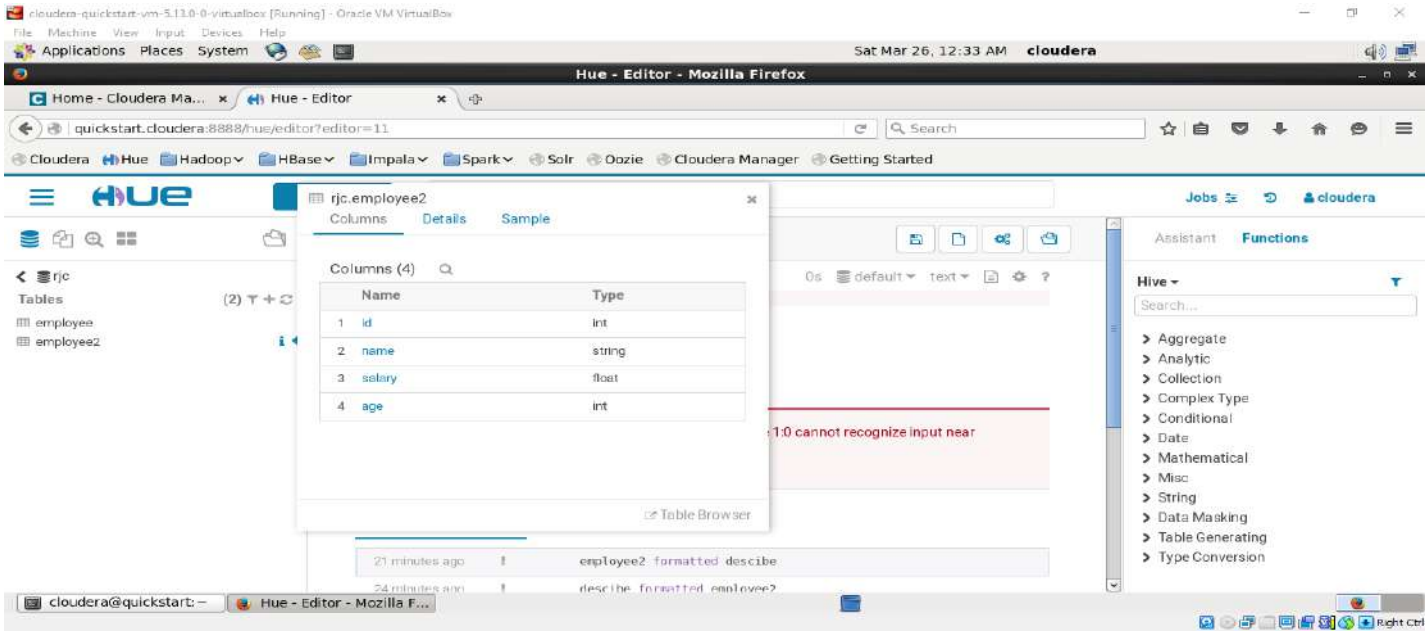


We can also Preview the sample data. Here it is showing blank because we have not inserted anything or data inside this employee table.



When we click on employee table it will give the fields of employee table.





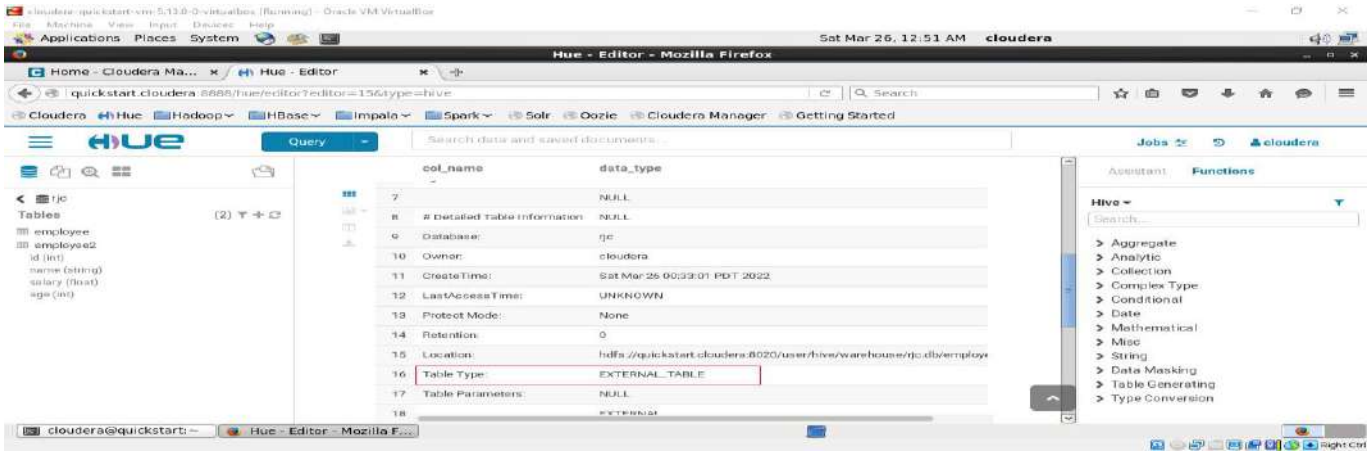
Above if not created external table then first drop the table

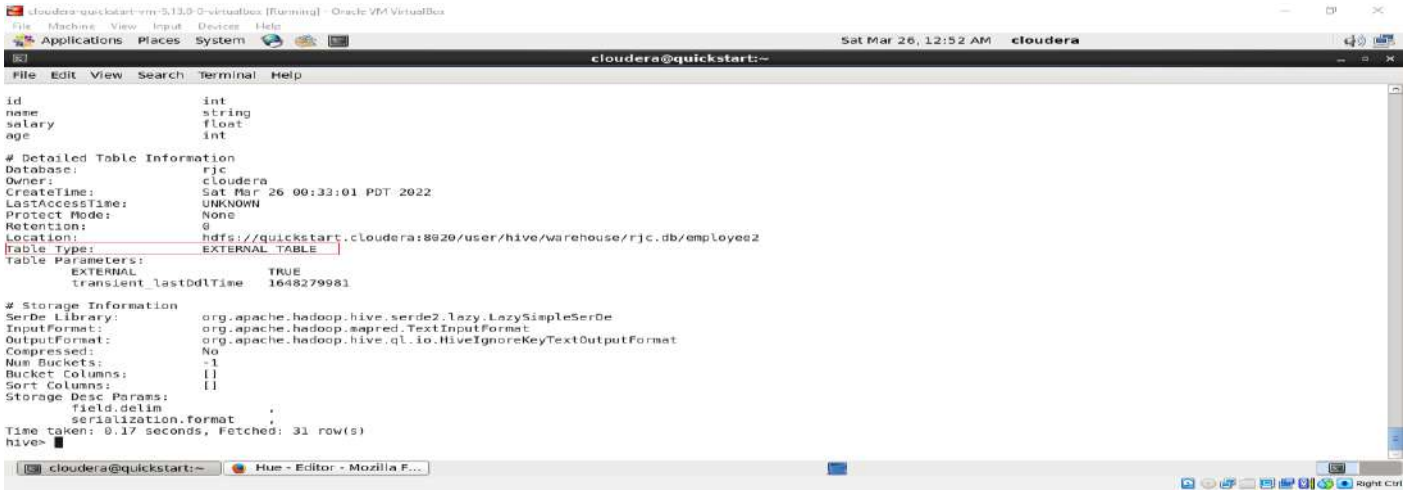
drop table employee2;

```
Time taken: 0.749 seconds
hive> create external table employee2(ID int, name string, salary float, age int)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;
OK
Time taken: 0.19 seconds
hive>
```

If view the properties describe command give the properties of employee2 table.

describe formatted employee2;

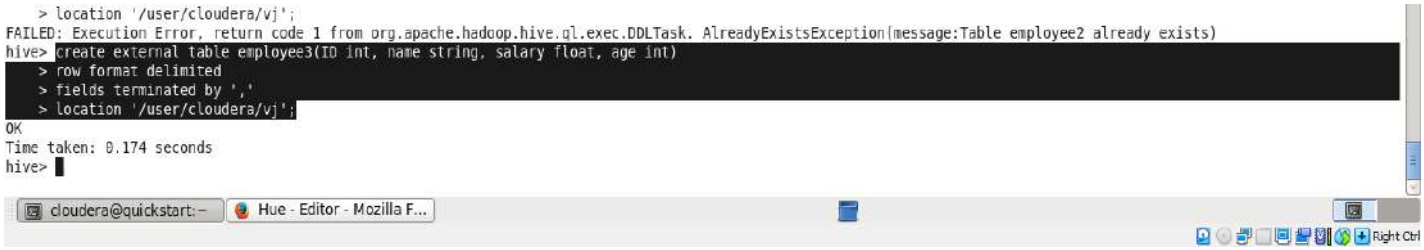




15. Creating a new external table named as employee3 in the specific location using below command,

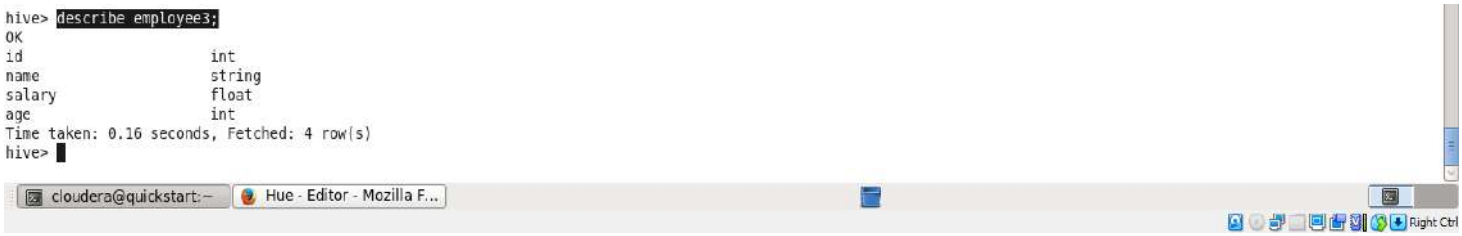
```
create external table employee3(ID int, name string, salary float, age int)
    row format delimited
    fields terminated by ','
    location '/user/cloudera/vj';
```

It will first create 'vj' directory inside the /user/cloudera and then inside 'vj' the employee3 table get stored.



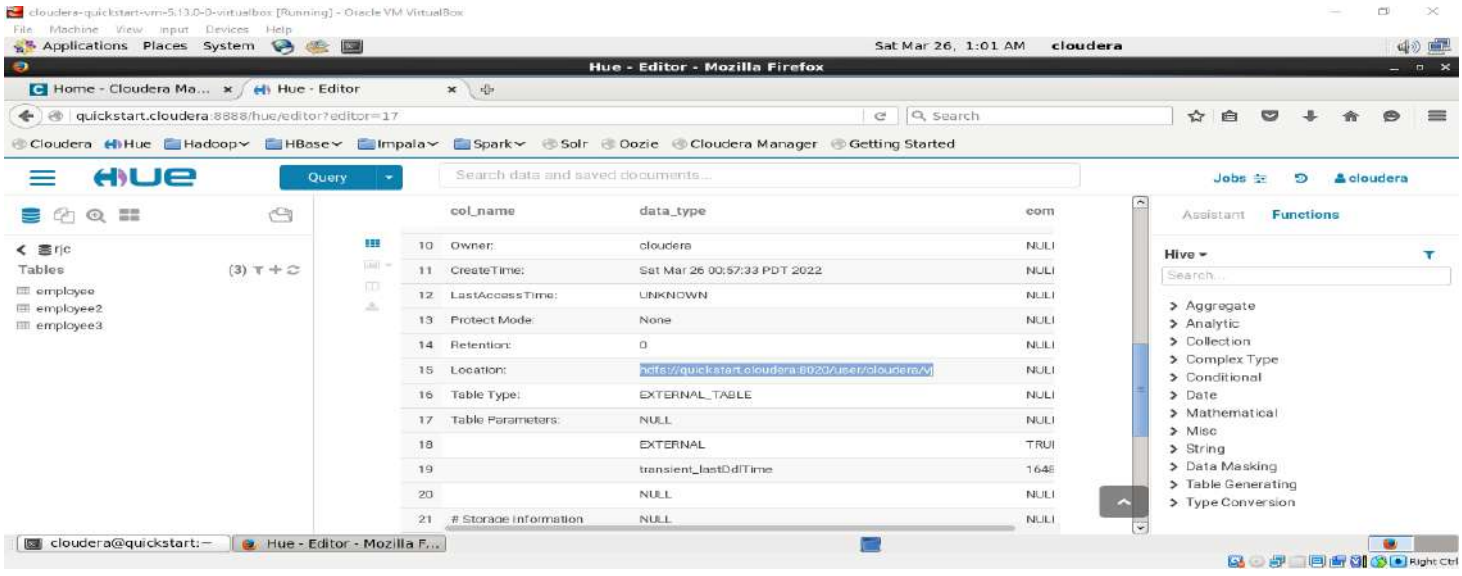
16. To see the schema of the employee3 table we use below command,

```
describe employee3;
```



17. Then switch to browser and Refresh and select the rjc database and refresh, now it will display the employee3 table along with other two tables which we have created earlier.

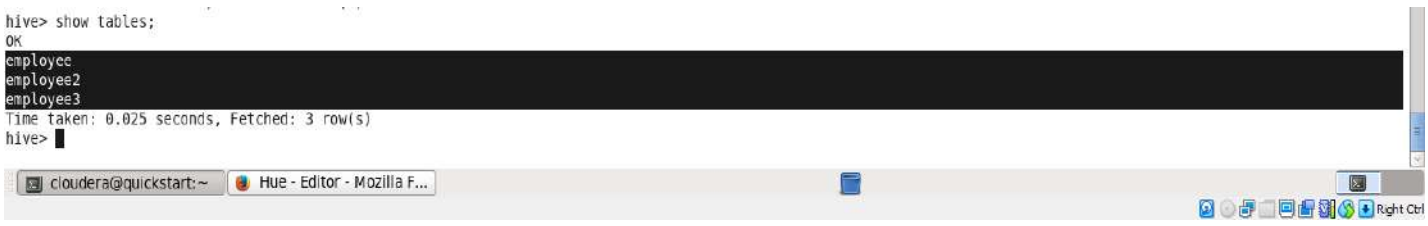
```
describe formatted employee3;
```



Here we will see the properties of employee3 table here we can also see the location of the table where it is stored.

18. Now move to terminal and listing out all the tables using below command;

```
show tables;
```



19. ALTER COMMANDS

If changing the name of the employee3 table to emptable using below command,

alter table employee3 RENAME TO emptable;

20. First we will see the fields of employee3 then we will add new column as surname in employee3 using below command,

alter table employee3 add columns(surname string);

```
hive> describe employee3;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.15 seconds, Fetched: 4 row(s)
hive> alter table employee3 add columns(surname string);
OK
Time taken: 0.265 seconds
hive> describe employee3;
OK
id                      int
name                    string
salary                  float
age                     int
surname                 string
Time taken: 0.186 seconds, Fetched: 5 row(s)
hive>
```

21. Now we will change field name of the emptable to first_name using alter command,

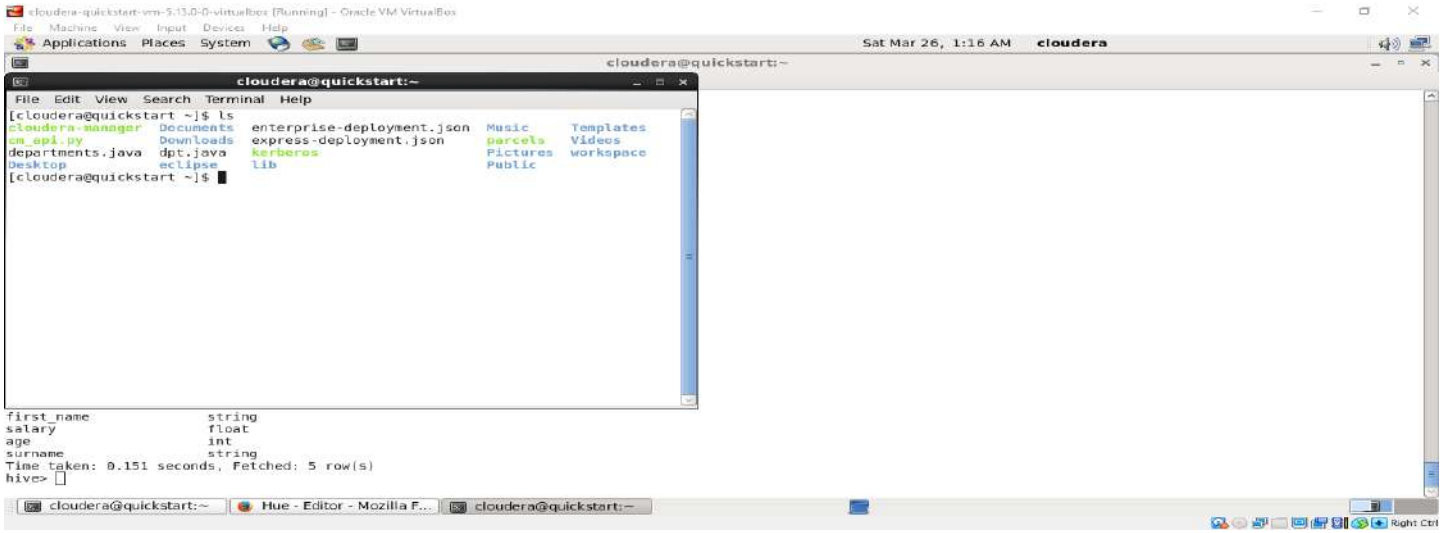
Syntax: alter table <table_name> change <old_coln_name> <new_coln_name> string;

alter table employee3 change name first_name string;

```
hive> alter table employee3 change name first_name string;
OK
Time taken: 0.572 seconds
hive> describe employee3;
OK
id                      int
first_name              string
salary                  float
age                     int
surname                 string
Time taken: 0.151 seconds, Fetched: 5 row(s)
hive>
```

Loading the data in the table

22. Before loading the data in the table we will first create the csv file. Now open the new terminal , using ls command list out all the directories --> change the directory to document directory --> use ls command to list all the files present inside the document folder or directory



23. Now creating new file as Student.csv using below command,

gedit Student.csv

As soon as we hit enter it will create a Student.csv file as it was not existing earlier.

ID,Name,Course,Age

1,Rahul,Hadoop,20

2,Shiv,Java,22

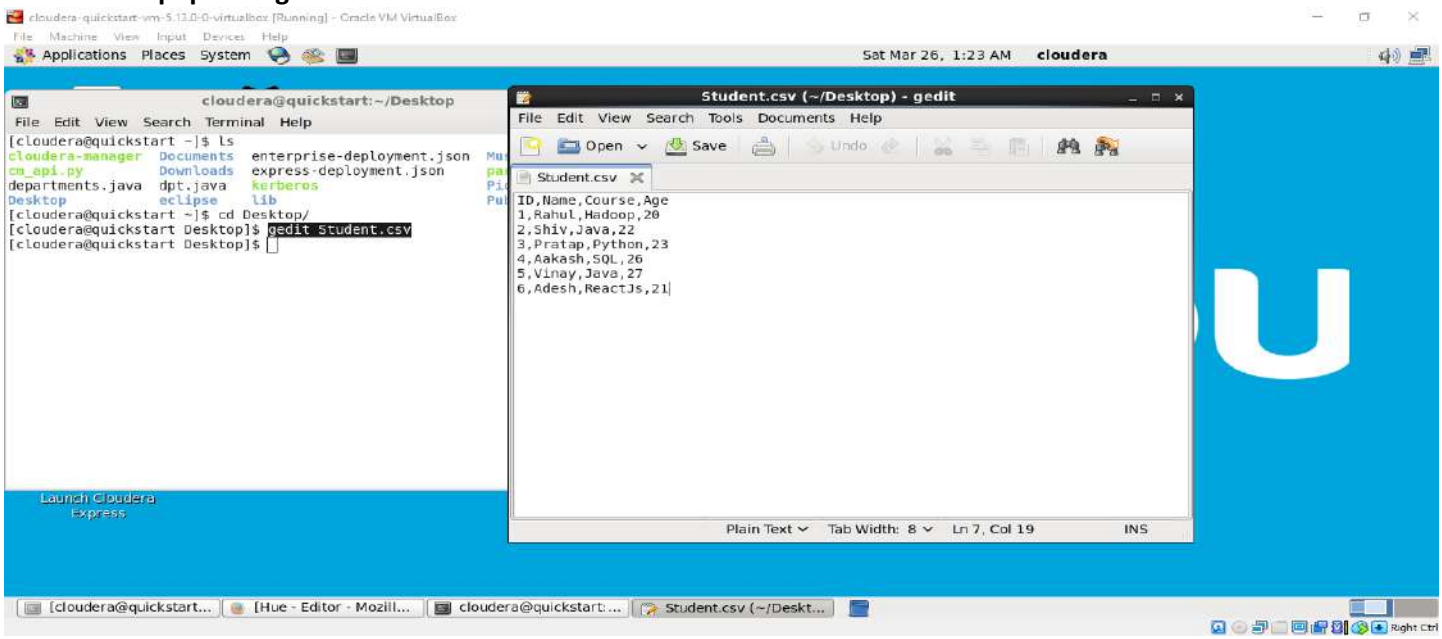
3,Pratap,Python,23

4,Aakash,SQL,26

5,Vinay,Java,27

6,Adesh,ReactJs,21

Now we are populating the Student.csv file with some data and save this file.

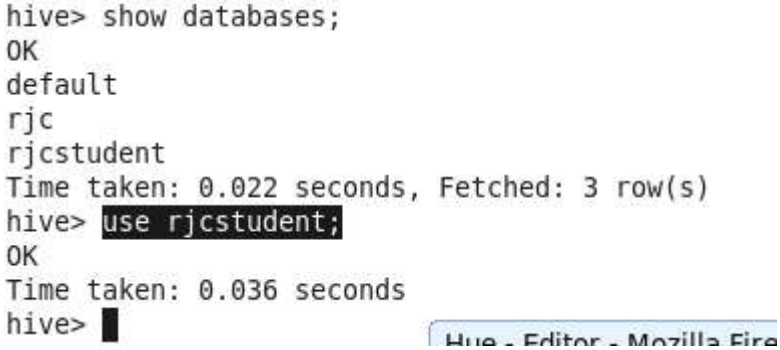


24. Creating a new database as rjcstudent.

create database rjcstudent;

25. Using rjcstudent database.

use rjcstudent;



26. Creating new table **student** inside **rjcstudent** database.

```
create table student(ID int, Name string, Age int)
    partitioned by(Course string)
    row format delimited
    fields terminated by ',';
```

```
hive> create table student(ID int, Name string, Age int)
    > partitioned by(Course string)
    > row format delimited
    > fields terminated by ',';
OK
Time taken: 0.516 seconds
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | cloudera@quickstart:...

27. To see the structure or schema of the table,

```
describe student;
```

```
hive> describe student;
OK
id                      int
name                    string
age                     int
course                  string

# Partition Information
# col_name              data_type               comment

course                  string
Time taken: 0.243 seconds, Fetched: 9 row(s)
```

28. Loading data in the student table from Student.csv file which we have created in document directory. Here we are partitioning based on course = 'Hadoop'.

```
load data local inpath "/home/cloudera/Desktop/Student.csv" into table student
partition(Course="Hadoop");
```

```
hive> load data local inpath "/home/cloudera/Desktop/Student.csv" into table student
    > partition(Course="Hadoop");
Loading data to table rjcstudent.student partition (course=Hadoop)
Partition rjcstudent.student{course=Hadoop} stats: [numFiles=1, numRows=0, totalSize=122, rawDataSize=0]
OK
Time taken: 2.081 seconds
hive>
```

cloudera@quickstart:~ | Hue - Editor - Mozilla F... | cloudera@quickstart:...

It is partitioning based on Hadoop.

```
select * from student;
```

```
hive> select * from student;
OK
NULL    Name    NULL    Hadoop
1       Rahul   NULL    Hadoop
2       Shiv    NULL    Hadoop
3       Pratap  NULL    Hadoop
4       Aakash  NULL    Hadoop
5       Vinay   NULL    Hadoop
6       Adesh   NULL    Hadoop
Time taken: 0.509 seconds, Fetched: 7 row(s)
hive>
```

cloudera@quickstart:~ | Hue - Editor - Mozilla F... | cloudera@quickstart:...

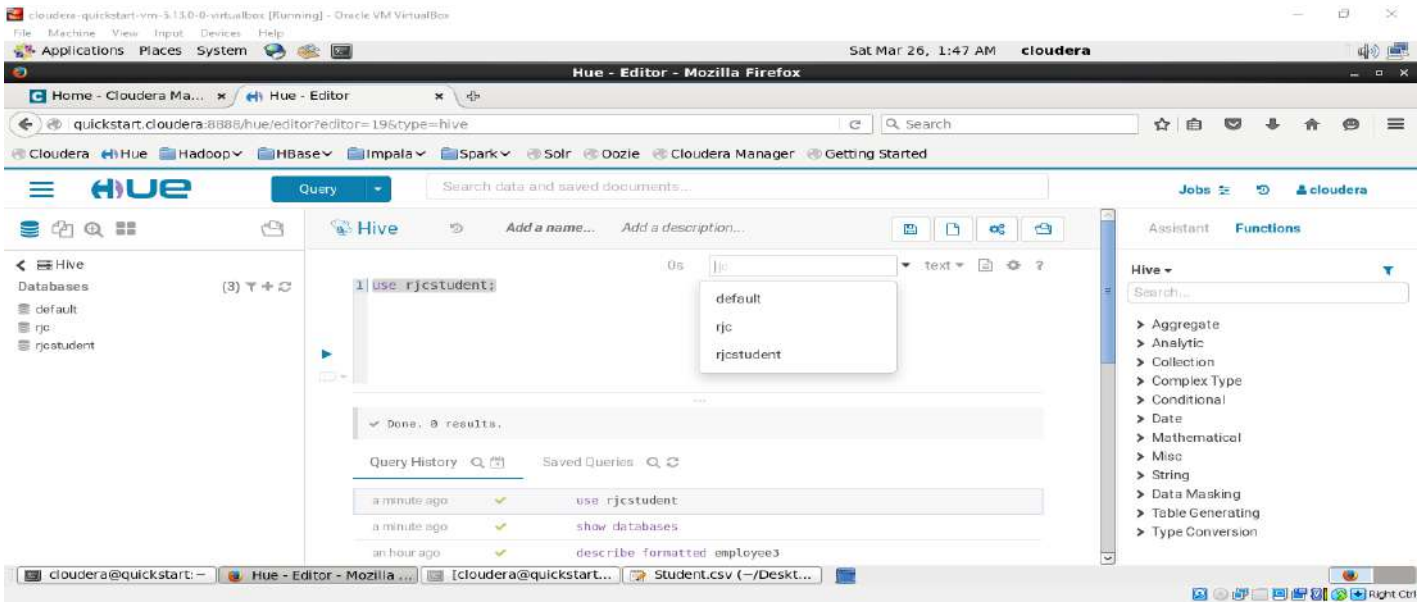
29. Now we similarly partition for course = Java and course = Python.

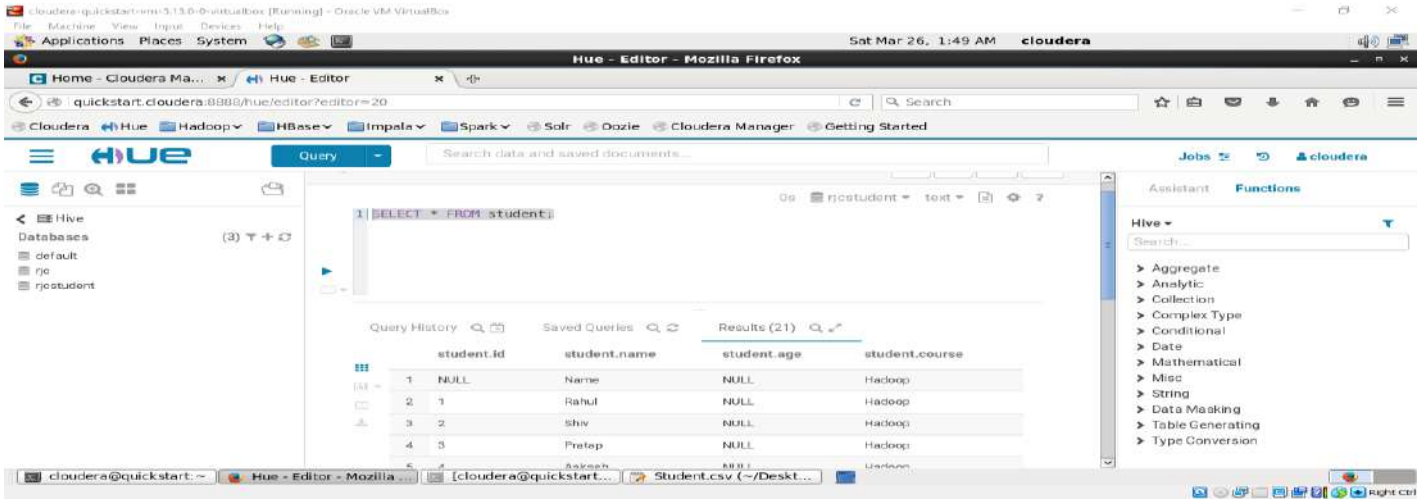
```
hive> load data local inpath "/home/cloudera/Desktop/Student.csv" into table student
    > partition(Course="Python");
Loading data to table rjcstudent.student partition (course=Python)
Partition rjcstudent.student{course=Python} stats: [numFiles=1, numRows=0, totalSize=122, rawDataSize=0]
OK
Time taken: 1.687 seconds
hive> load data local inpath "/home/cloudera/Desktop/Student.csv" into table student
    > partition(Course="Java");
Loading data to table rjcstudent.student partition (course=Java)
Partition rjcstudent.student{course=Java} stats: [numFiles=1, numRows=0, totalSize=122, rawDataSize=0]
OK
Time taken: 0.762 seconds
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ | [Hue - Editor - Mozill... | [cloudera@quickstart... | Student.csv (~/Deskt...

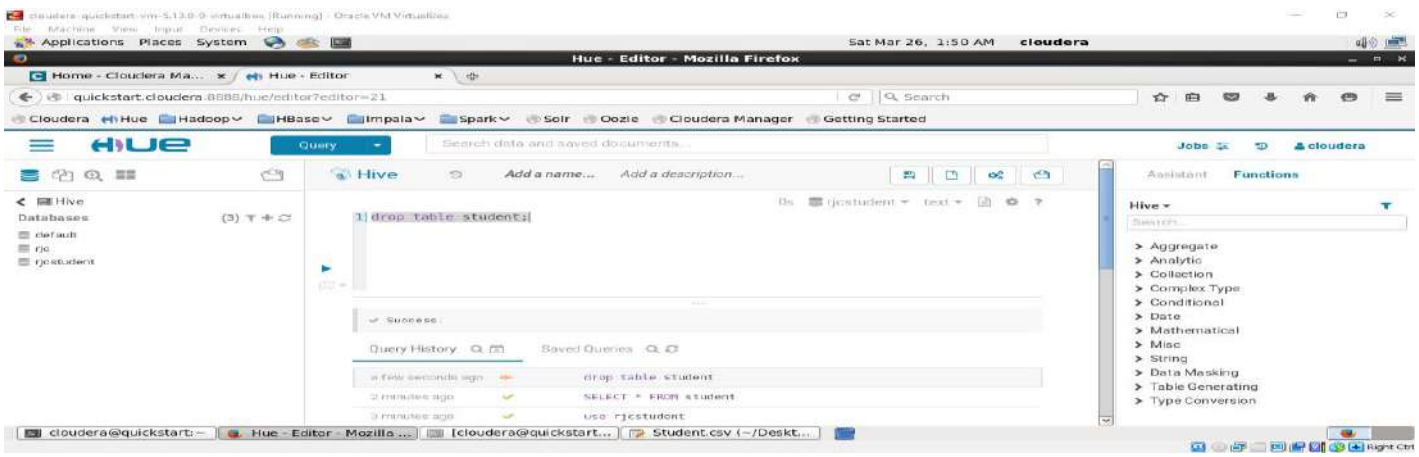
30. Now go to browser refresh the page and select database as rjcstudent and click in





31. Now dropping the table student and creating the student table again normally (i.e. without partitioning).

drop table student;



```
create table student(ID int, Name string, Course string, Age int)
    row format delimited
    fields terminated by ','
    tblproperties("skip.header.line.count"="1");
```

```
hive> create table student(ID int, Name string, Course string, Age int)
    > row format delimited
    > fields terminated by ','
    > tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.145 seconds
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ | Hue - Editor - Mozilla ... | [cloudera@quickstart... | Student.csv (~/Deskt...

32. Loading data in the student table from Student.csv file which present inside /home/cloudera/Desktop directory.

```
load data local inpath "/home/cloudera/Desktop/Student.csv" into table student;
```

```
hive> load data local inpath "/home/cloudera/Desktop/Student.csv" into table student;
Loading data to table rjcstudent.student
Table rjcstudent.student stats: [numFiles=1, totalSize=122]
OK
Time taken: 0.482 seconds
hive>
```

Student.csv (~/Desktop) - gedit

cloudera@quickstart:~ | [Hue - Editor - Mozill... | [cloudera@quickstart... | Student.csv (~/Deskt...

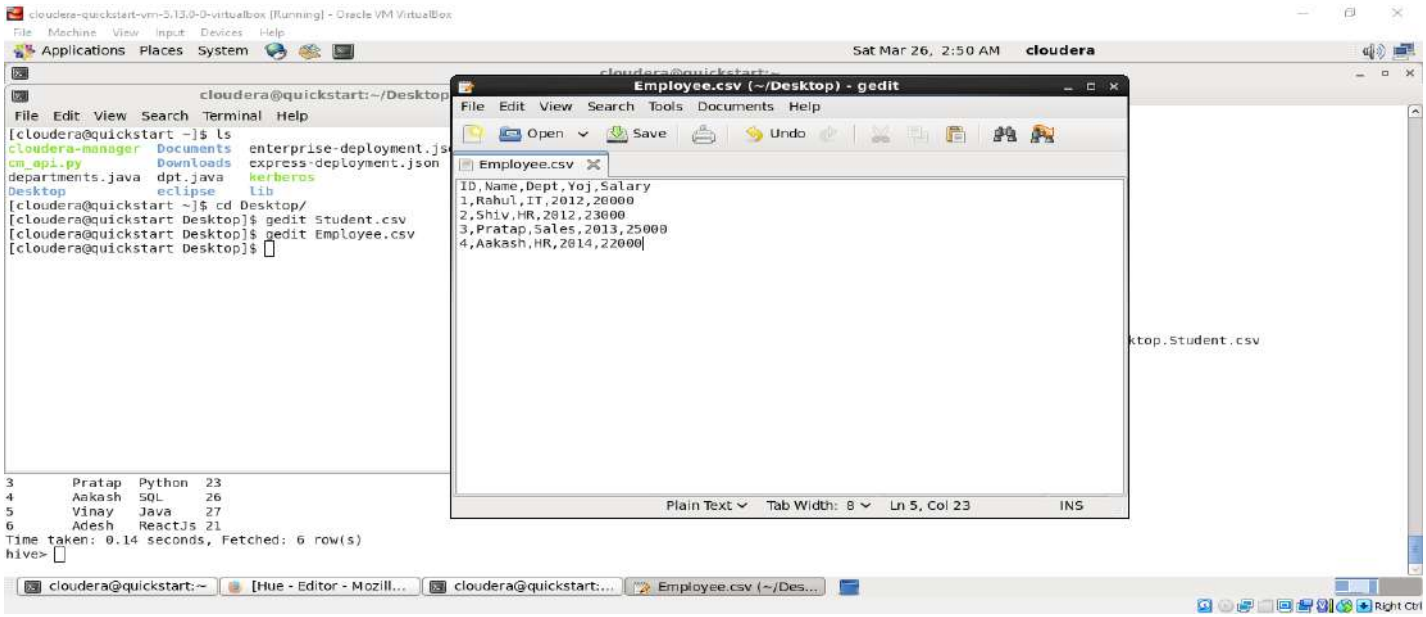
```
select * from student;
```

```
hive> select * from student;
OK
1       Rahul   Hadoop  20
2       Shiv    Java    22
3       Pratap  Python  23
4       Aakash  SQL     26
5       Vinay   Java    27
6       Adesh   ReactJs 21
Time taken: 0.14 seconds, Fetched: 6 row(s)
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozill... | [cloudera@quickstart... | Student.csv (~/Deskt...

33. Now creating employee.csv file.

```
gedit Employee.csv
```



Querying :

34. Creating new database for performing querying operations.

create database hiveql;

```
hive> create database hiveql;
OK
Time taken: 0.082 seconds
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

35. Using database hiveql and creating table employee inside the hiveql datanase.

create table employee(ID int, Name string, Department string, YOJ int, Salary float)

row format delimited

fields terminated by ','

tblproperties("skip.header.line.count"="1");

```
hive> create table employee(ID int, Name string, Department string, YOJ int, Salary float)
    > row format delimited
    > fields terminated by ','
    > tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.113 seconds
hive>
```

Employee.csv (~/Desktop) - gedit

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

Now we will see the schema of employee table using below command,

describe employee;

```
hive> describe employee;
OK
id                      int
name                    string
department              string
yoj                     int
salary                  float
Time taken: 0.184 seconds, Fetched: 5 row(s)
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

36. Loading the data into employee table from employee.csv file which we have created earlier and it is present in /home/cloudera/Documents directory.

load data local inpath "/home/cloudera/Desktop/Employee.csv" into table employee;

```
hive> load data local inpath "/home/cloudera/Desktop/Employee.csv" into table employee;
Loading data to table rjcstudent.employee
Table rjcstudent.employee stats: [numFiles=1, totalSize=116]
OK
Time taken: 0.609 seconds
hive>
```

Employee.csv (~/Desktop) - gedit

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

Displaying the table using below command,

select * from employee;

```
hive> select * from employee;
OK
1       Rahul   IT      2012    20000.0
2       Shiv    HR      2012    23000.0
3       Pratap  Sales   2013    25000.0
4       Aakash  HR      2014    22000.0
Time taken: 0.141 seconds, Fetched: 4 row(s)
hive>
```

Employee.csv (~/Desktop) - gedit

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

Now we Perform some operations on this employee table.

37. We are printing or displaying the rows or records of employees whose salary is greater than and equal to 25000.

select * from employee where salary >=25000;

```
hive> select * from employee where salary >=25000;
Query ID = cloudera_20220326025858_4b2362ef-bd75-4923-9285-d373dbb34163
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_1648262911818_0005, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0005/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 02:58:54,227 Stage-1 map = 0%,  reduce = 0%
2022-03-26 02:59:16,852 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.44 sec
MapReduce Total cumulative CPU time: 4 seconds 440 msec
Ended Job = job_1648262911818_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.44 sec   HDFS Read: 4735 HDFS Write: 28 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 440 msec
OK
3       Pratap  Sales   2013    25000.0
Time taken: 55.82 seconds, Fetched: 1 row(s)
hive>
```

38. Now we are printing or displaying the rows or records of employees whose salary is less than 25000.

select * from employee where salary <25000;

```
hive> select * from employee where salary <25000;
Query ID = cloudera_20220326030000_38afaa20-0321-413f-b867-18a4e55a7ab8
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_1648262911818_0006, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0006/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 03:00:44,357 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:01:09,379 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.62 sec
MapReduce Total cumulative CPU time: 4 seconds 620 msec
Ended Job = job_1648262911818_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.62 sec   HDFS Read: 4797 HDFS Write: 72 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 620 msec
OK
1       Rahul   IT      2012    20000.0
2       Shiv    HR      2012    23000.0
4       Aakash  HR      2014    22000.0
Time taken: 58.944 seconds, Fetched: 3 row(s)
hive>
```

Aggregating

39. Arithmetic operations:

We are **adding** 2000 in existing salary and displaying specific columns using below command,

select ID, Name, salary + 2000 from employee;

```
hive> select ID, Name, salary + 2000 from employee;
Query ID = cloudera_20220326030303_332505b8-4680-4457-aaeb-ad0c82963858
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_1648262911818_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 03:03:52,983 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:04:17,932 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.43 sec
MapReduce Total cumulative CPU time: 4 seconds 430 msec
Ended Job = job_1648262911818_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.43 sec   HDFS Read: 4881 HDFS Write: 65 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 430 msec
OK
1       Rahul   22000.0
2       Shiv    25000.0
3       Pratap  27000.0
4       Aakash  24000.0
Time taken: 64.708 seconds, Fetched: 4 row(s)
hive>
```

40. Displaying the **maximum salary** using below command,

select max(salary) from employee;

```
hive> select max(salary) from employee;
Query ID = cloudera_20220326030505_1f91277d-6abe-467d-8a66-c8e7e234dde1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0008/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0008
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:06:17,722 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:06:37,510 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.79 sec
2022-03-26 03:06:58,636 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.61 sec
MapReduce Total cumulative CPU time: 6 seconds 610 msec
Ended Job = job_1648262911818_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.61 sec   HDFS Read: 8055 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 610 msec
OK
25000.0
Time taken: 71.963 seconds, Fetched: 1 row(s)
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

Here we can see that **25000** is maximum salary.

41. Displaying the **minimum salary** using below command,

select min(salary) from employee;

```
hive> select min(salary) from employee;
Query ID = cloudera_20220326030707_6bbaa9ee-0051-4f2d-956a-197b44b53a46
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0009, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0009/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0009
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:08:30,966 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:08:51,622 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.84 sec
2022-03-26 03:09:13,784 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.81 sec
MapReduce Total cumulative CPU time: 6 seconds 810 msec
Ended Job = job_1648262911818_0009
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.81 sec   HDFS Read: 8055 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 810 msec
OK
20000.0
Time taken: 77.906 seconds, Fetched: 1 row(s)
hive>
```

No Items in Trash

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

Here we can see that **20000** is minimum salary.

42. Now finding the **square root** of salary using below command,

select ID, Name, sqrt(salary) from employee;

```
hive> select ID, Name, sqrt(salary) from employee;
Query ID = cloudera_20220326031010_0793d885-79ca-4f9c-819d-f2d7598ed1f7
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_1648262911818_0010, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 03:10:56,356 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:11:17,978 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.65 sec
MapReduce Total cumulative CPU time: 4 seconds 650 msec
Ended Job = job_1648262911818_0010
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.65 sec   HDFS Read: 4545 HDFS Write: 108 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 650 msec
OK
1       Rahul   141.4213562373095
2       Shiv    151.65750888103102
3       Pratap  158.11388300841898
4       Aakash  148.32396974191326
Time taken: 51.912 seconds, Fetched: 4 row(s)
hiv
```

cloudera@quickstart:~

cloudera@quickstart:~ [Hue - Editor - Mozill... [cloudera@quickstart... [Employee.csv (~/De...

43. Now we are showing the name column in **upper case** using below command,

select ID, upper(name) from employee;

```
hive> select ID, upper(name) from employee;
Query ID = cloudera_20220326031212_8d0193e4-eea2-4fac-bdca-39b329e488b8
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_1648262911818_0011, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0011/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 03:12:32,143 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:12:53,889 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.75 sec
MapReduce Total cumulative CPU time: 4 seconds 750 msec
Ended Job = job_1648262911818_0011
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.75 sec   HDFS Read: 4331 HDFS Write: 33 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 750 msec
OK
1       RAHUL
2       SHIV
3       PRATAP
4       AAKASH
Time taken: 52.859 seconds, Fetched: 4 row(s)
hive>
```

Hue - Editor - Mozilla Firefox

44. Now we are showing the name column in **lower case** using below command,

select ID, lower(name) from employee;

```
hive> select ID, lower(name) from employee;
Query ID = cloudera_20220326031313_fbbd273b-decf-40ac-be8f-86447f24bd5e
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_1648262911818_0012, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0012/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-26 03:14:14,234 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:14:39,884 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.53 sec
MapReduce Total cumulative CPU time: 4 seconds 530 msec
Ended Job = job_1648262911818_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.53 sec   HDFS Read: 4331 HDFS Write: 33 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 530 msec
OK
1       rahul
2       shiv
3       pratap
4       aakash
Time taken: 53.377 seconds, Fetched: 4 row(s)
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozill... | [cloudera@quickstart... | [Employee.csv (~/De...

45. Creating another **employee2.csv** file with the same data as employee.csv but adding one more column as **Country** to it.

46. Creating a new table as **empgroup**.

create table empgroup(ID int, Name string, Department string, YOJ int, Salary float, Country string)

row format delimited

fields terminated by ','

tblproperties("skip.header.line.count"="1");

```
hive> create table empgroup(ID int, Name string, Department string, YOJ int, Salary float, Country string)
    > row format delimited
    > fields terminated by ','
    > tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.206 seconds
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | [cloudera@quickstart:...

47. Loading the data into **empgroup** table from **employee2.csv** file which we have created and it is present in /home/cloudera/Dekstop directory.

load data local inpath "/home/cloudera/Desktop/Employee2.csv" into table empgroup;

```
hive> load data local inpath "/home/cloudera/Desktop/Employee2.csv" into table empgroup;
Loading data to table rjcstudent.empgroup
Table rjcstudent.empgroup stats: [numFiles=1, totalSize=138]
OK
Time taken: 0.868 seconds
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | [cloudera@quickstart:...

Displaying the table using below command,

Select * from empgroup;

```
hive> select * from empgroup;
OK
1       Rahul   IT      2012    20000.0 IND
2       Shiv    HR      2012    23000.0 UK
3       Pratap  Sales   2013    25000.0 UK
4       Aakash  HR      2014    22000.0 USA
Time taken: 0.129 seconds, Fetched: 4 row(s)
hive>
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | [cloudera@quickstart:...

48. Groupby clause

Now we display the **total sum of salary** of employees **country wise** using below command,

select Country, sum(Salary) from empgroup group by Country;

```
hive> select Country, sum(Salary) from empgroup group by Country;
Query ID = cloudera_20220326032121_506b17ce-b13e-4ae0-98c7-d4ff365af9ee
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0013, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0013/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0013
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:22:03,266 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:22:25,810 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.87 sec
2022-03-26 03:22:43,484 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.87 sec
MapReduce Total cumulative CPU time: 6 seconds 870 msec
Ended Job = job_1648262911818_0013
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.87 sec   HDFS Read: 8629 HDFS Write: 35 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 870 msec
OK
IND     20000.0
UK      48000.0
USA     22000.0
Time taken: 69.464 seconds, Fetched: 3 row(s)
hive>
```

Groupby clause along with the having clause

Taking the total sum of salary countrywise using groupby clause and from that selecting or displaying those country whose **total sum of salary** > **30000** using having clause.

select Country, sum(Salary) from empgroup group by Country having sum(Salary)>30000;

```
hive> select Country, sum(Salary) from empgroup group by Country having sum(Salary)>30000;
Query ID = cloudera_20220326032525_73a66c90-d2d6-4322-8a24-ab2fee5fe74e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0015, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0015/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0015
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:25:43,987 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:26:03,370 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.31 sec
2022-03-26 03:26:27,298 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 8.62 sec
MapReduce Total cumulative CPU time: 8 seconds 620 msec
Ended Job = job_1648262911818_0015
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 8.62 sec   HDFS Read: 9138 HDFS Write: 11 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 620 msec
OK
UK      48000.0
Time taken: 69.319 seconds, Fetched: 1 row(s)
hive>
```

Sorting : Order by

49. Now we are displaying the table by sorting the salary in **descending order**.

select * from empgroup order by Salary desc;

```
hive> select * from empgroup order by Salary desc;
Query ID = cloudera_20220326032828_d75db91e-d98a-4a2a-b3ac-4f8ddad78660
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0016, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0016/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0016
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:28:41,036 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:28:59,643 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.3 sec
2022-03-26 03:29:23,519 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.09 sec
MapReduce Total cumulative CPU time: 7 seconds 90 msec
Ended Job = job_1648262911818_0016
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.09 sec   HDFS Read: 8624 HDFS Write: 114 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 90 msec
OK
3       Pratap  Sales   2013    25000.0 UK
2       Shiv    HR      2012    23000.0 UK
4       Aakash  HR      2014    22000.0 USA
1       Rahul   IT      2012    20000.0 IND
Time taken: 70.666 seconds, Fetched: 4 row(s)
hive>
```

We can see that number of reducers: 1 for the order by.

50. Now we are displaying the table by **sorting** the salary in descending order.

select * from empgroup sort by Salary desc;

```
hive> select * from empgroup sort by Salary desc;
Query ID = cloudera_20220326033030_b0aa6056-13eb-48bb-aa8c-c1e56c95caec
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1648262911818_0017, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1648262911818_0017/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1648262911818_0017
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-26 03:30:55,515 Stage-1 map = 0%,  reduce = 0%
2022-03-26 03:31:20,041 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.83 sec
2022-03-26 03:31:43,167 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.78 sec
MapReduce Total cumulative CPU time: 6 seconds 780 msec
Ended Job = job_1648262911818_0017
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.78 sec   HDFS Read: 8624 HDFS Write: 114 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 780 msec
OK
3       Pratap  Sales   2013    25000.0 UK
2       Shiv    HR      2012    23000.0 UK
4       Aakash  HR      2014    22000.0 USA
1       Rahul   IT      2012    20000.0 IND
Time taken: 79.267 seconds, Fetched: 4 row(s)
hive>
```

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | [cloudera@quickstart:...

Now we can see the similar result as we got from order by and sort by so what is the difference between the two is that it depends on number of reducers in order by we got number of reducers is 1 and by using sort by here is also we got number of reducers is 1 so the difference between the two is that Order by will guarantee the total order in the output whereas sort by will only guarantee the ordering of the rows within the reducer.

Order by gives us completely sorted result whereas sort by give us partially sorted result.

quit

```
hive> quit
    > ;
WARN: The method class org.apache.commons.logging.impl.SLF4JLogFactory#release() was invoked.
WARN: Please see http://www.slf4j.org/codes.html#release for an explanation.
[cloudera@quickstart ~]$
```

cloudera@quickstart:~/Desktop

cloudera@quickstart:~ | [Hue - Editor - Mozilla ... | [cloudera@quickstart:...

Aim: To implement Word Count problem using Pig

What is Apache Pig

Apache Pig is a high-level data flow platform for executing MapReduce programs of Hadoop. The language used for Pig is Pig Latin.

The Pig scripts get internally converted to Map Reduce jobs and get executed on data stored in HDFS. Apart from that, Pig can also execute its job in Apache Tez or Apache Spark.

Pig can handle any type of data, i.e., structured, semi-structured or unstructured and stores the corresponding results into Hadoop Data File System. Every task which can be achieved using PIG can also be achieved using java used in MapReduce.

Features of Apache Pig

The various uses of Pig technology.

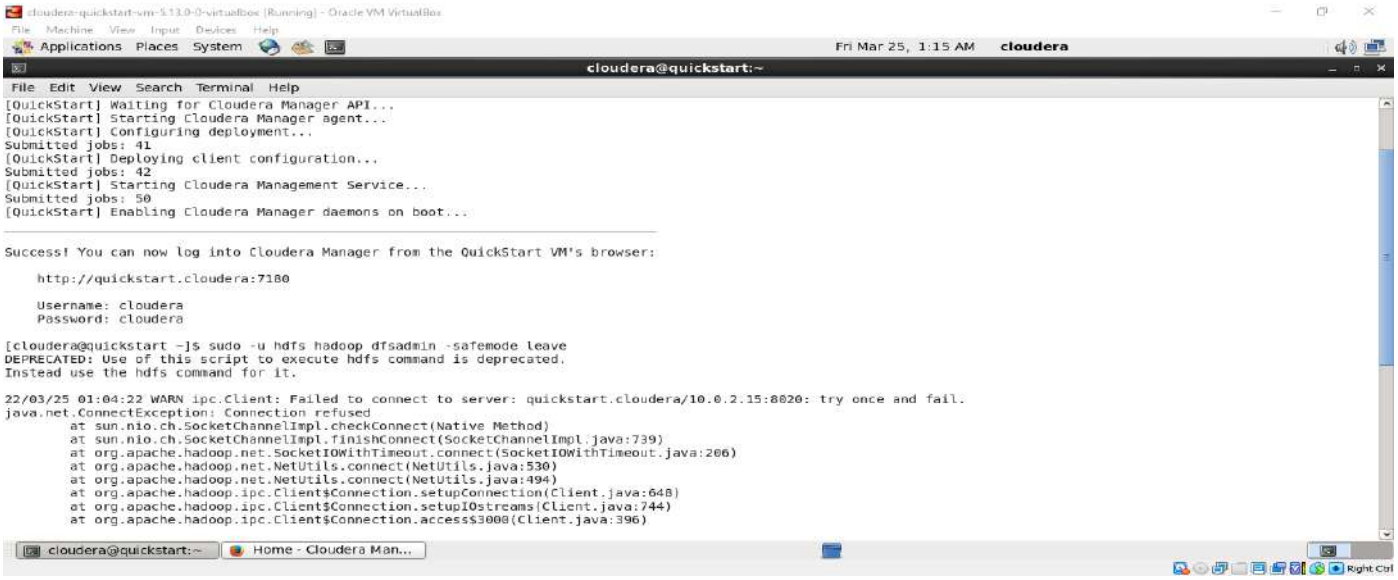
1) **Ease of programming**
   Writing complex java programs for map reduce is quite tough for non-programmers. Pig makes this process easy. In the Pig, the queries are converted to MapReduce internally.
2) **Optimization opportunities**
   It is how tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
3) **Extensibility**
   A user-defined function is written in which the user can write their logic to execute over the data set.
4) **Flexible**
   It can easily handle structured as well as unstructured data.
5) **In-built operators**
   It contains various type of operators such as sort, filter and joins.

Advantages of Apache Pig

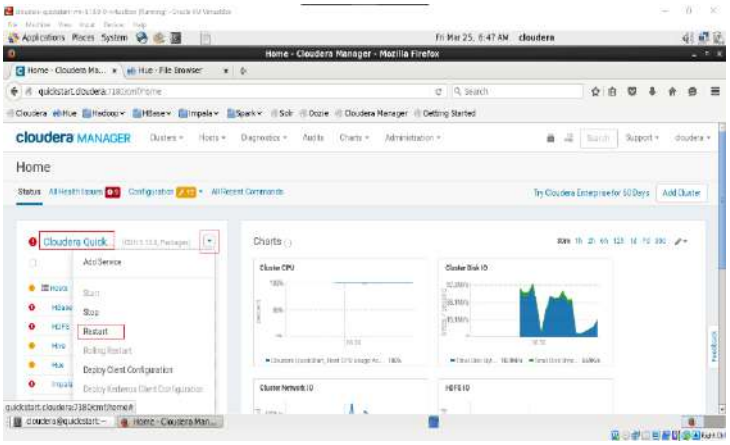
- **Less code - The Pig consumes less line of code to perform any operation.**
- **Reusability - The Pig code is flexible enough to reuse again.**
- **Nested data types - The Pig provides a useful concept of nested data types like tuple, bag, and**

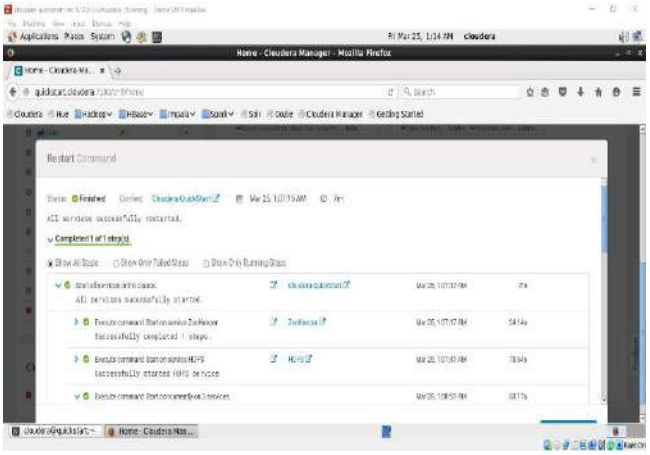
Before the perform the practical first perform 3 steps of the given following

1. **sudo /home/cloudera/cloudera-manager --force --express**

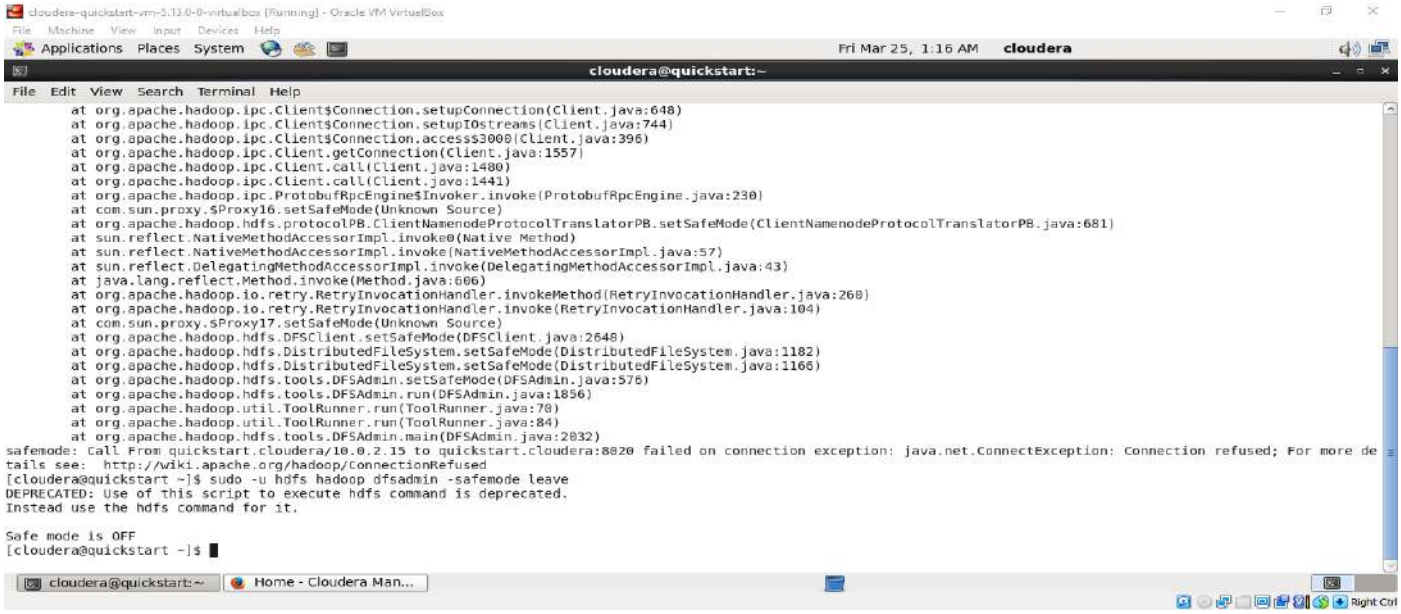


2. Start all services

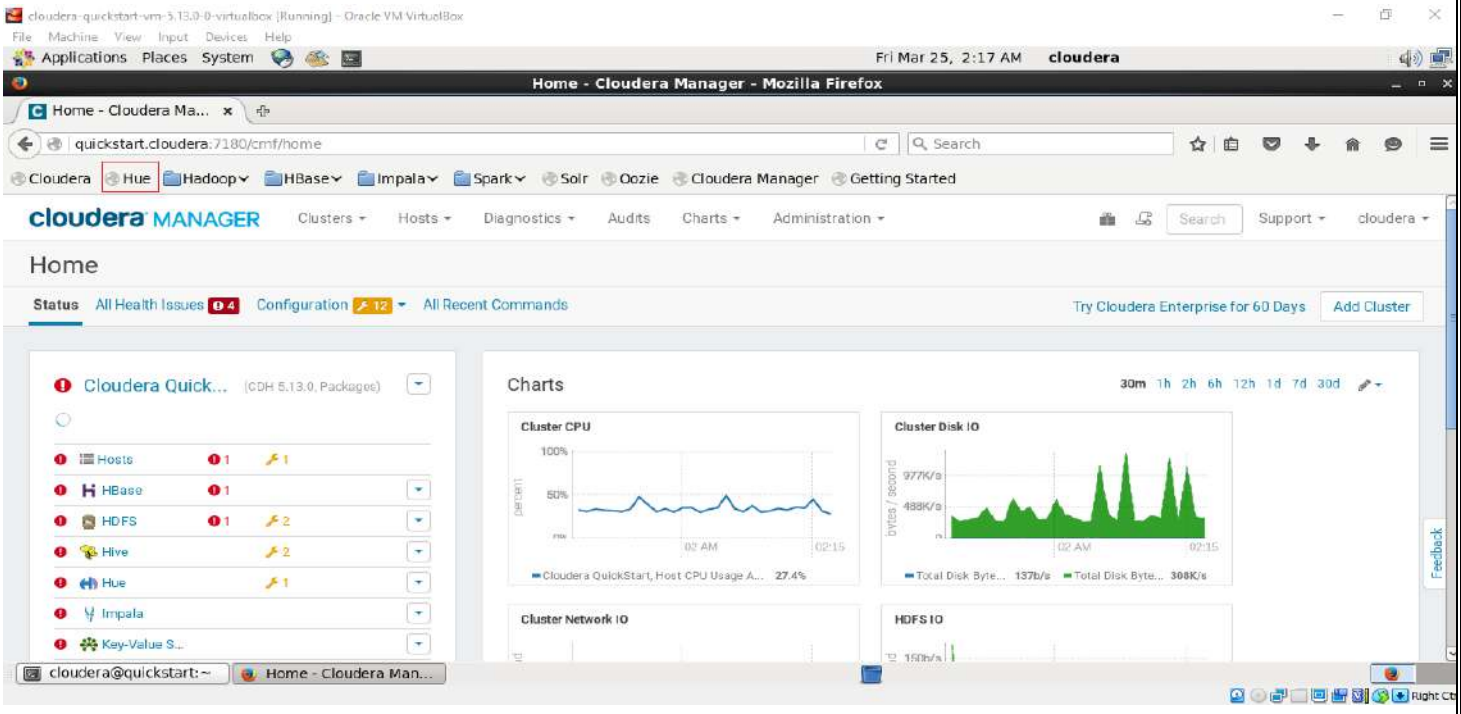


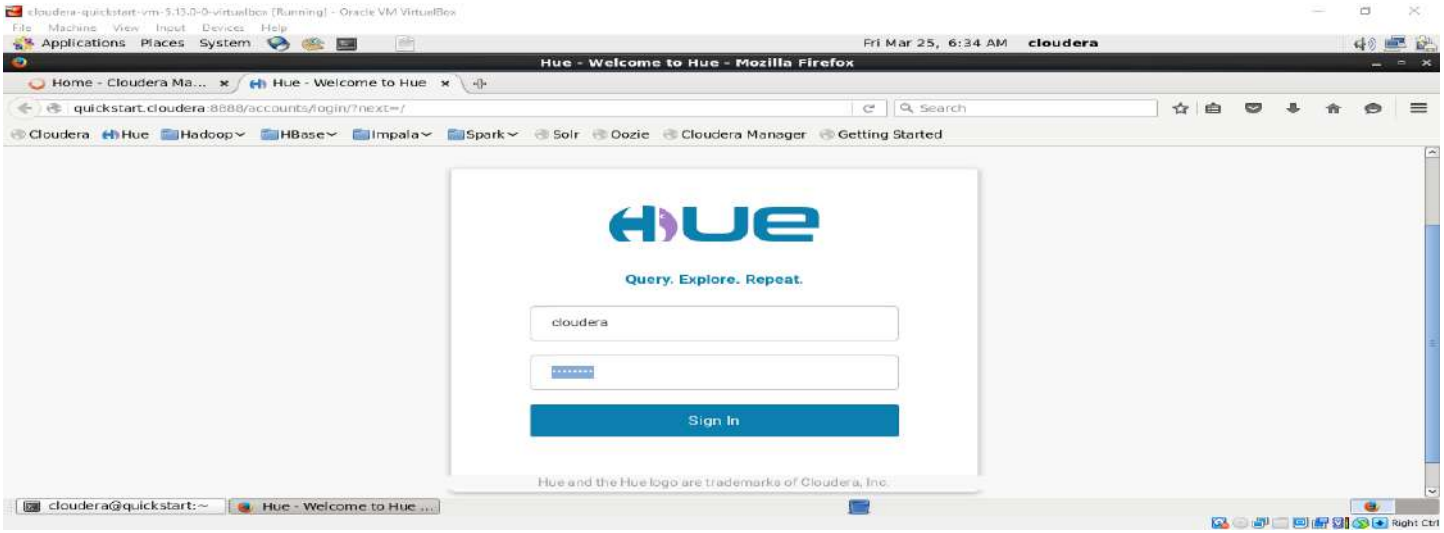


3. sudo -u hdfs hadoop dfsadmin -safemode leave

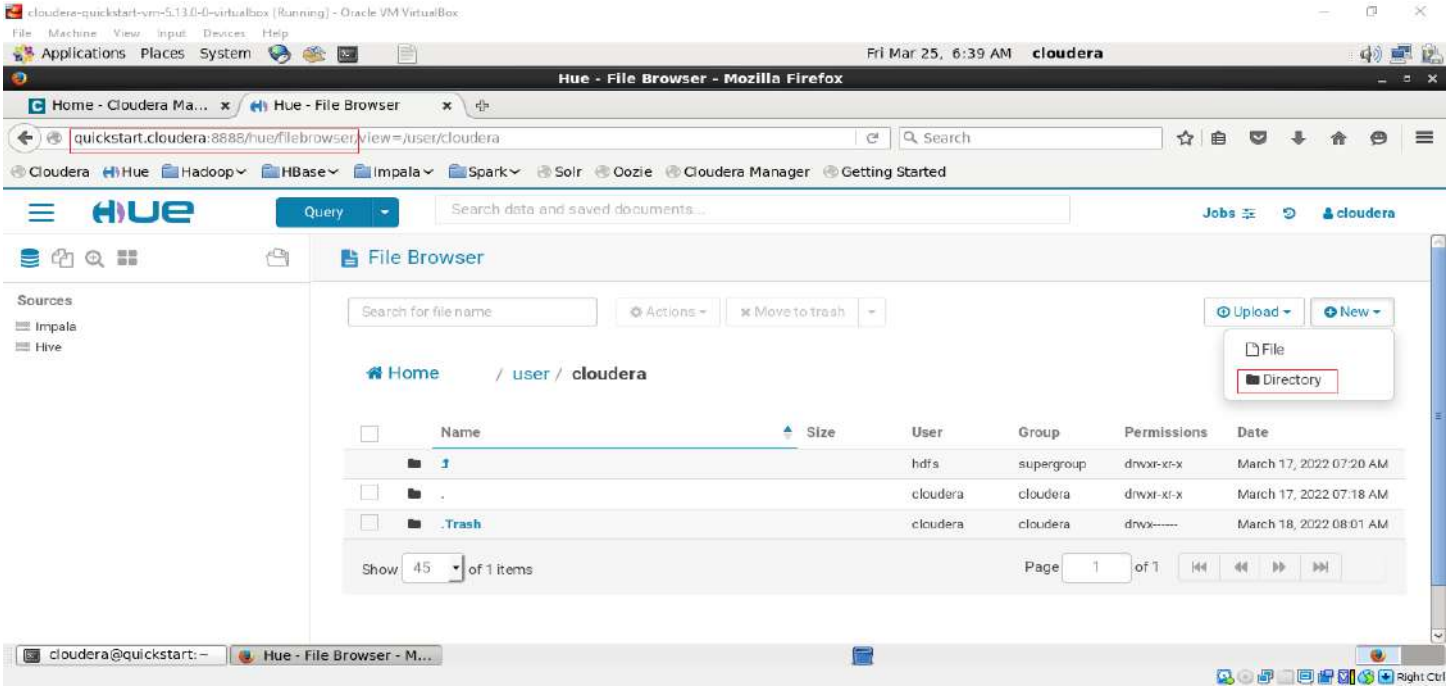


1. Open the browser. And then open Hue and login.

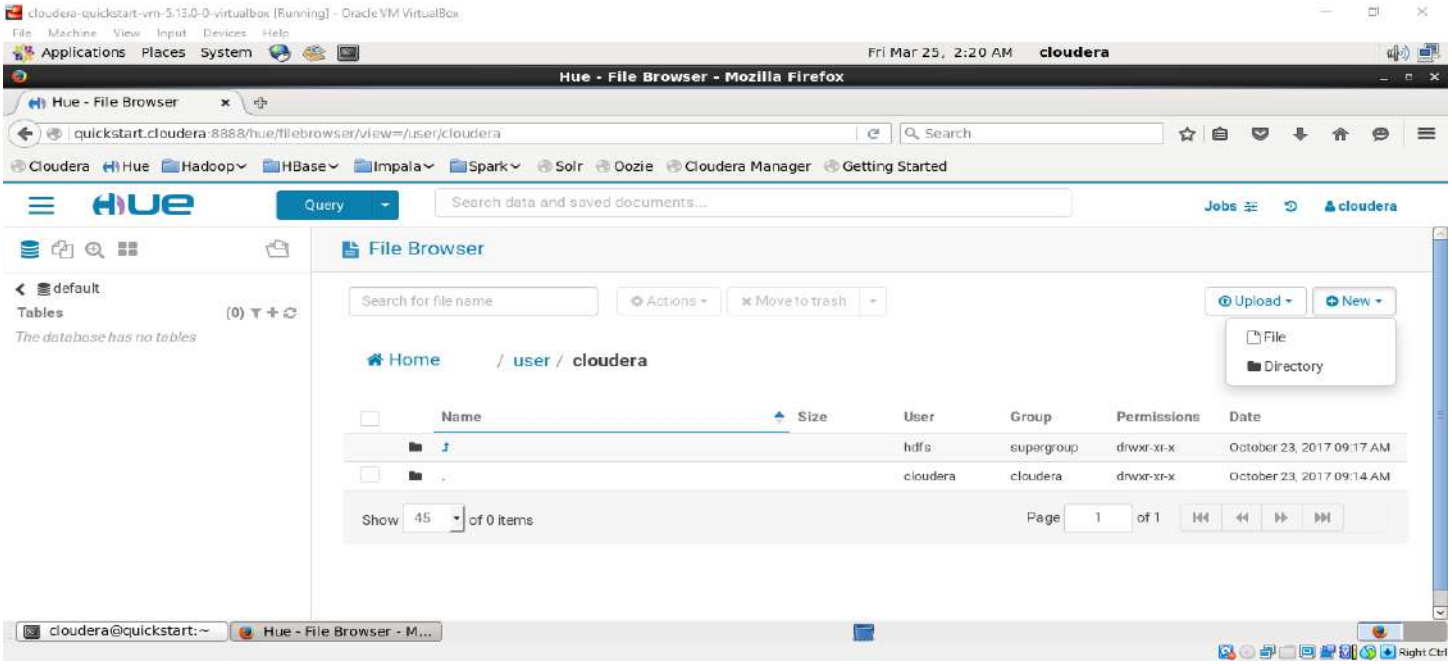


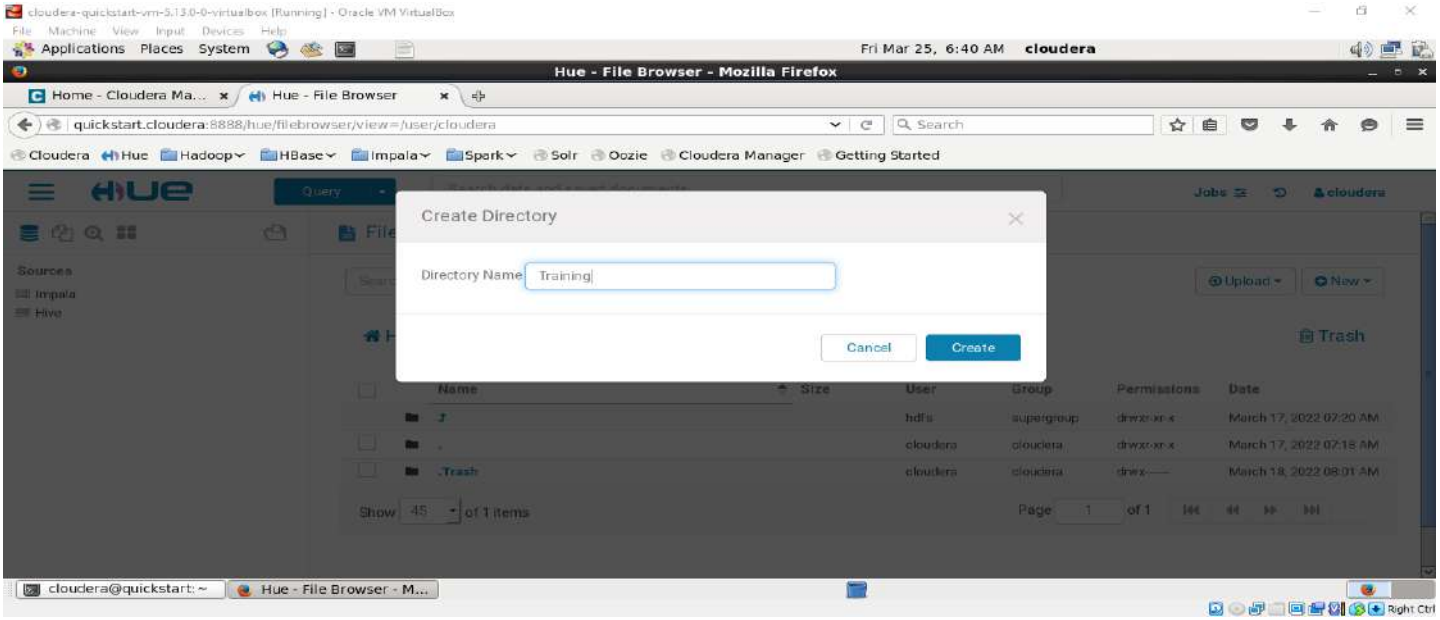


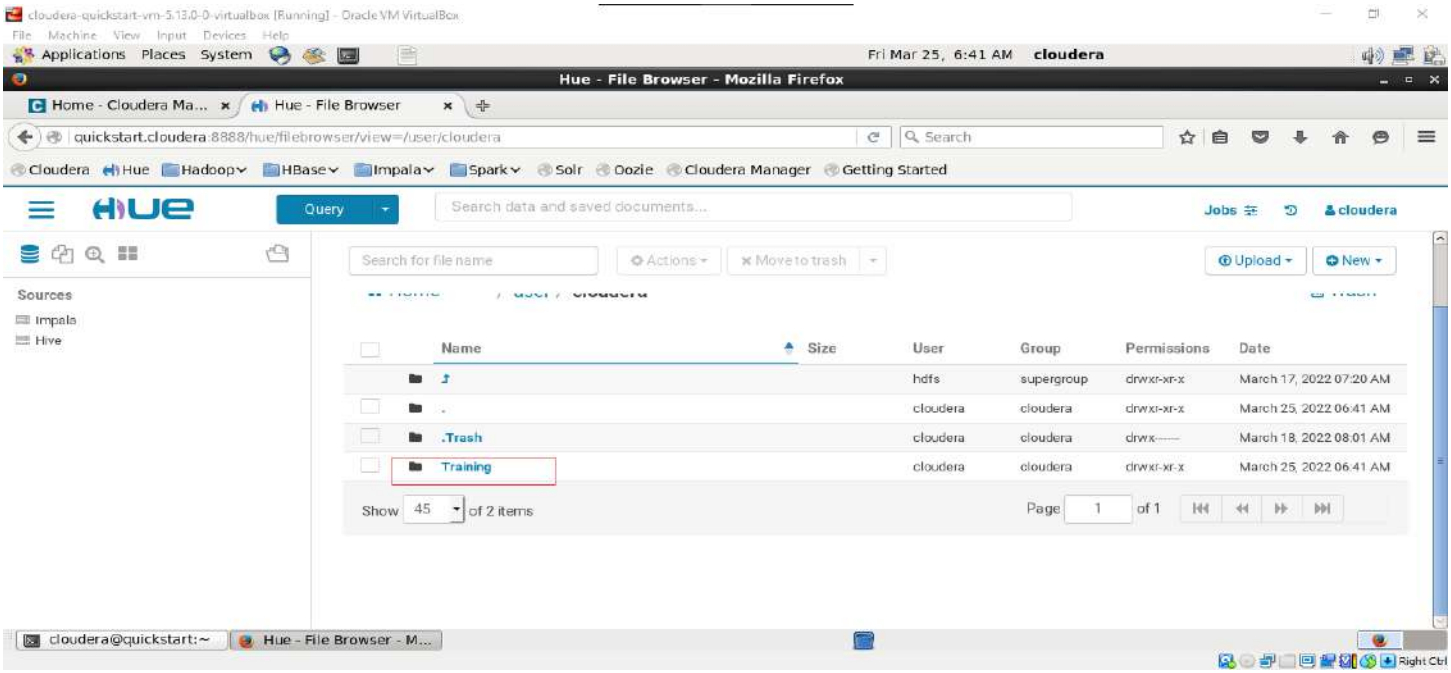
2. Now open the directory /user/cloudera
   Type /hue/**filebrowser**



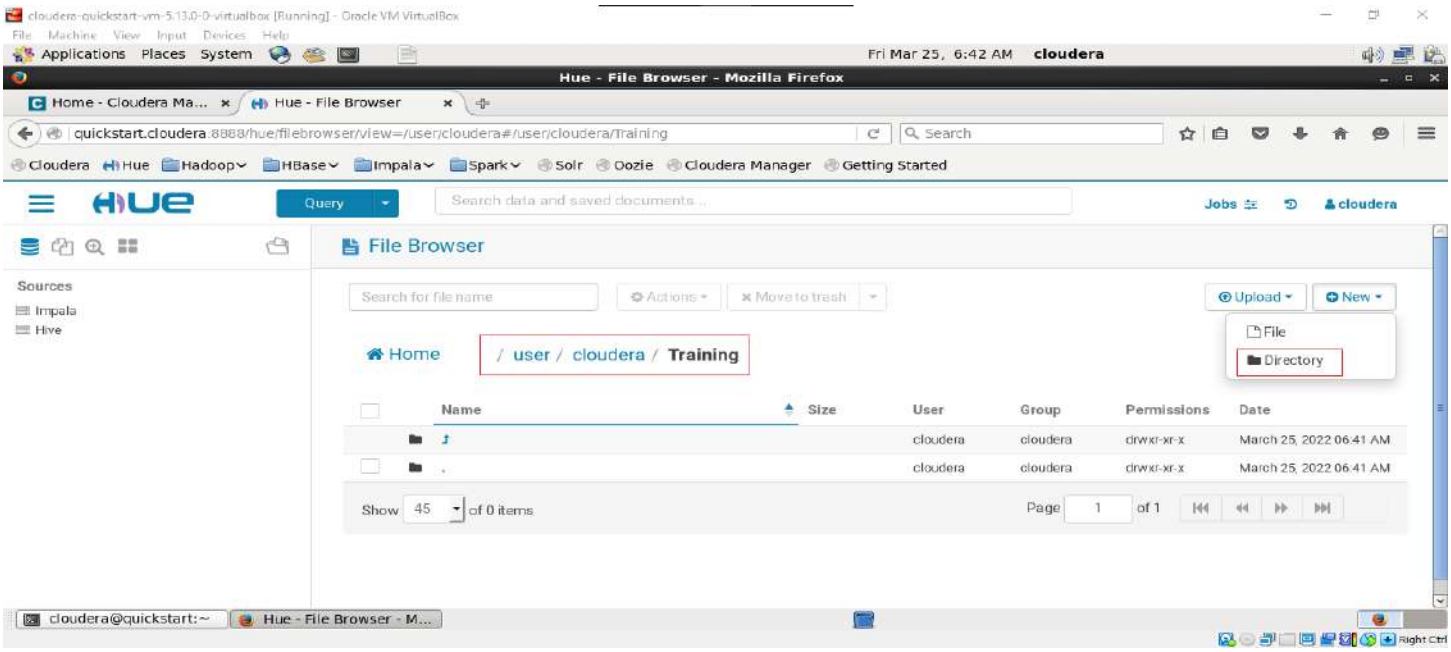
3. Now we are creating the directory as **Training** inside /user/cloudera

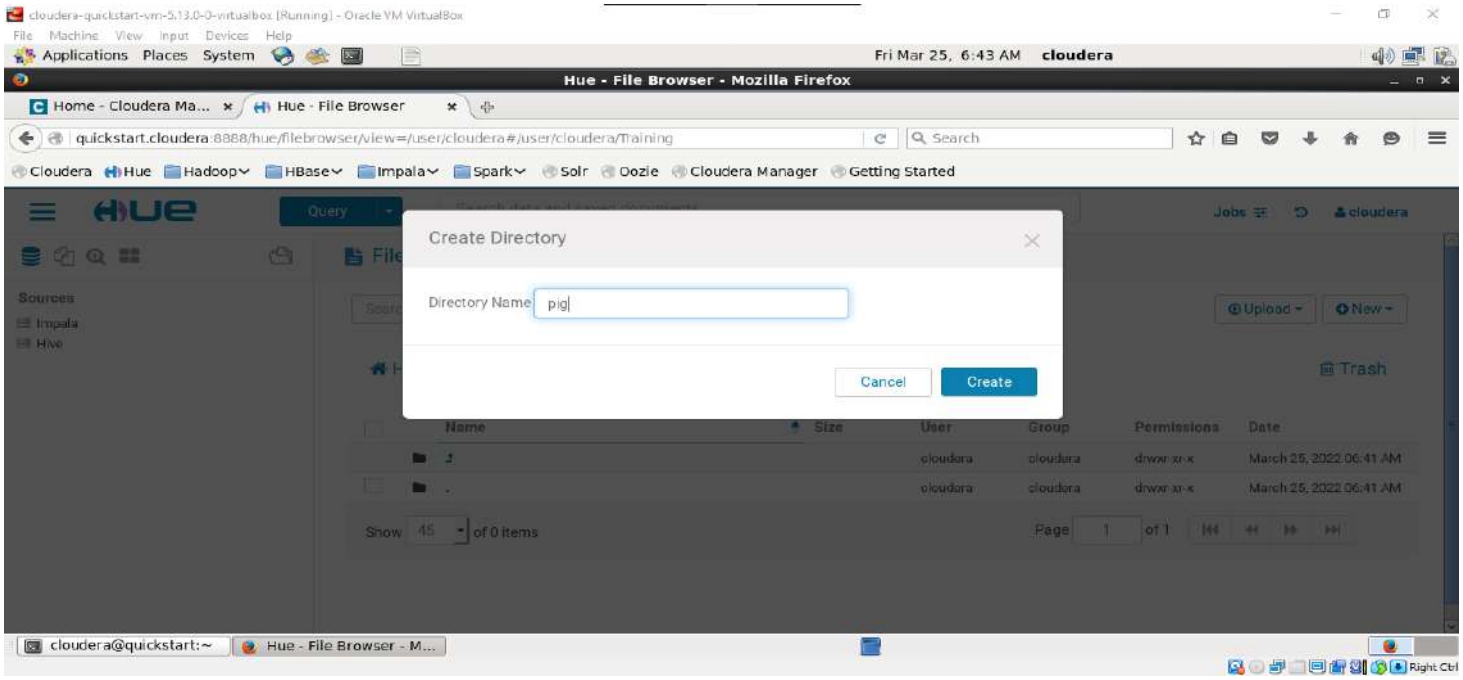




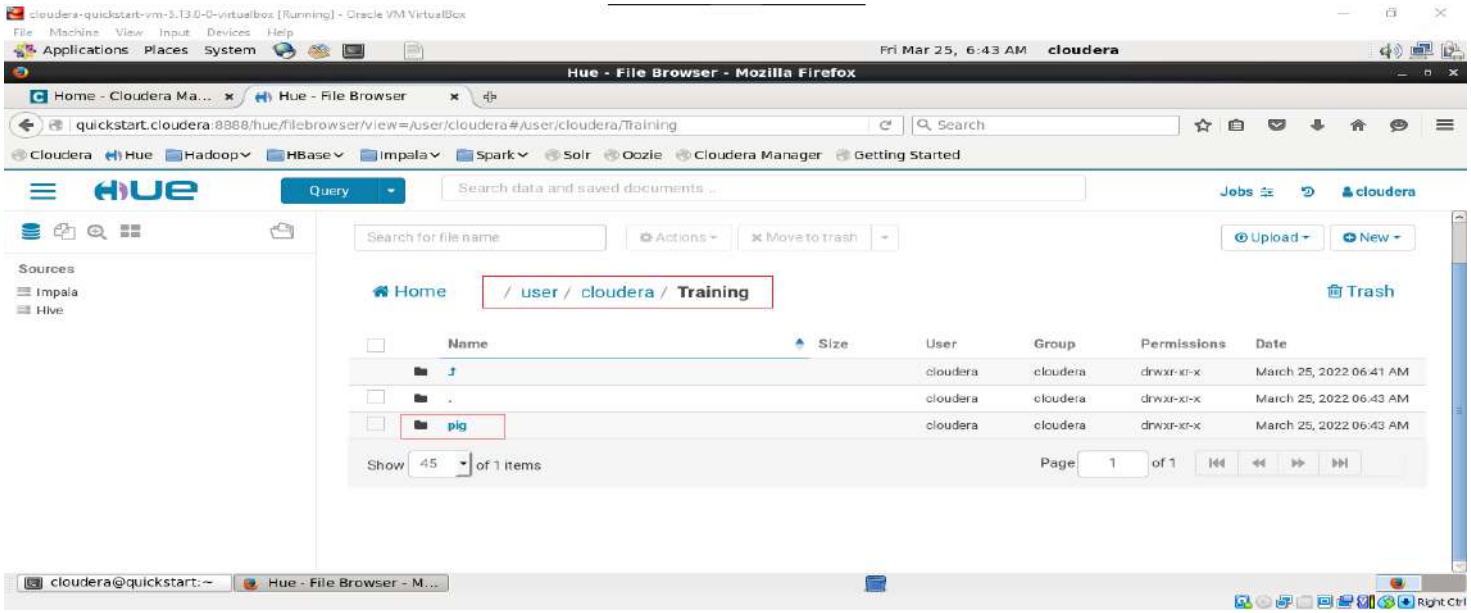


4. After creating Training directory now creating the **pig** directory inside /user/cloudera/**Training**

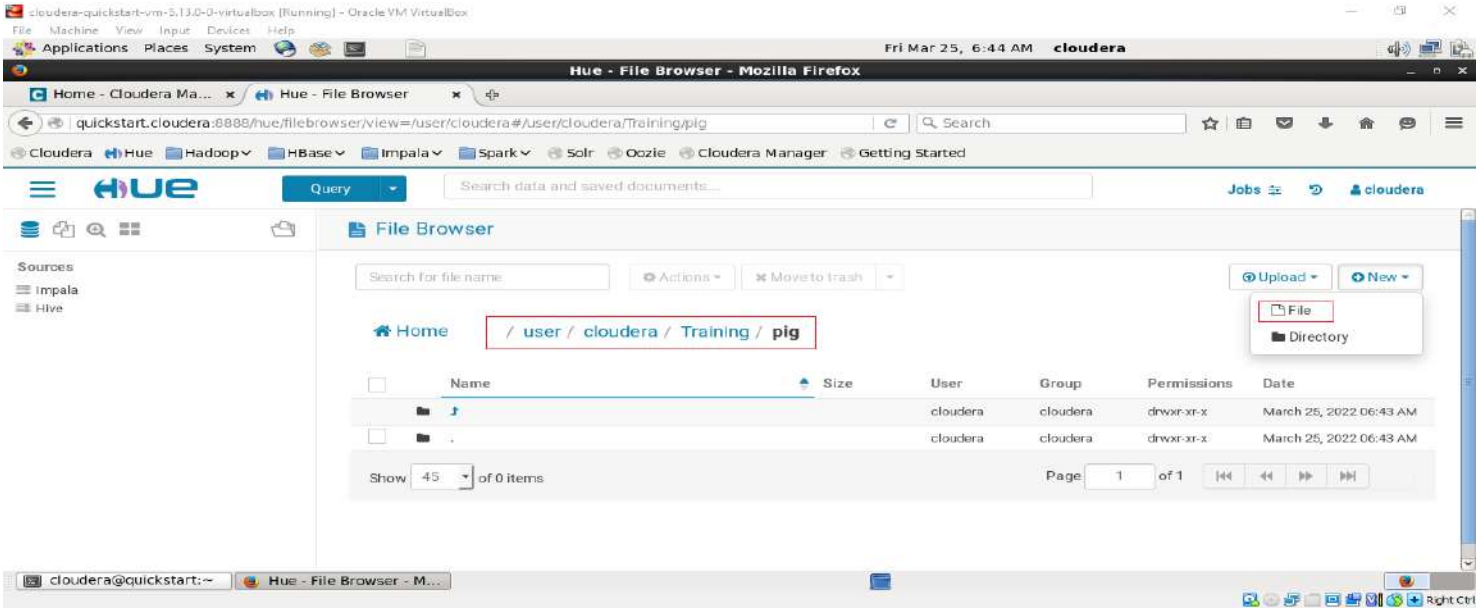


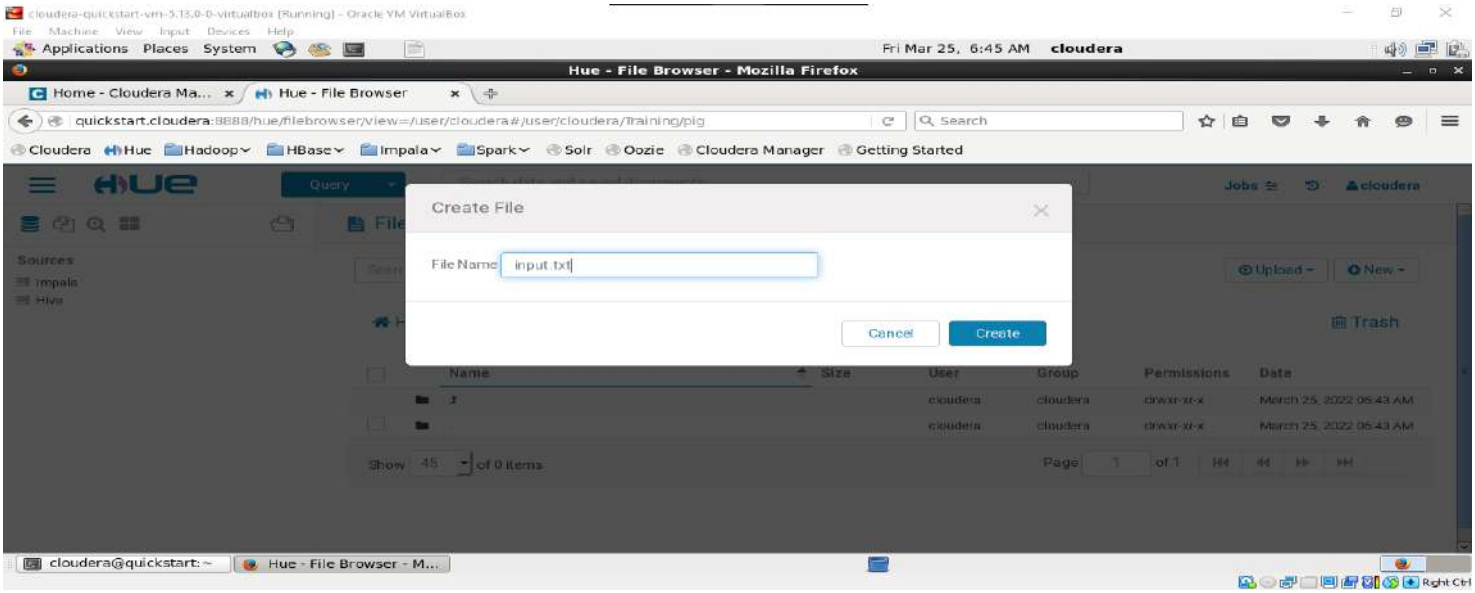


pig directory has been created inside /user/cloudera/Training

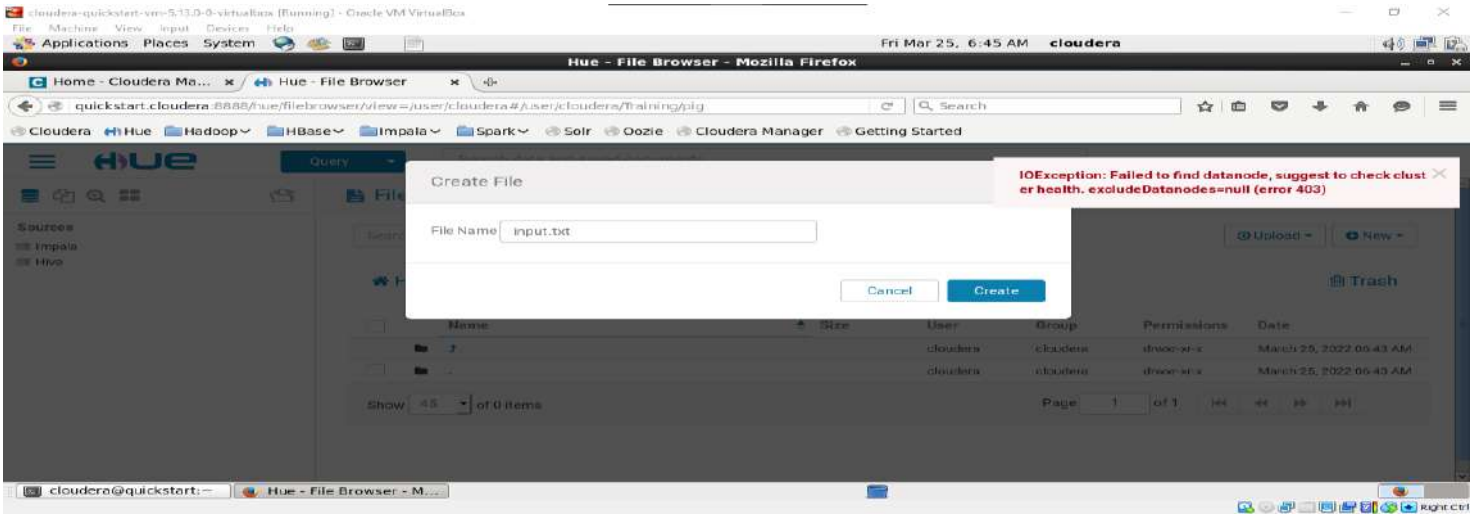


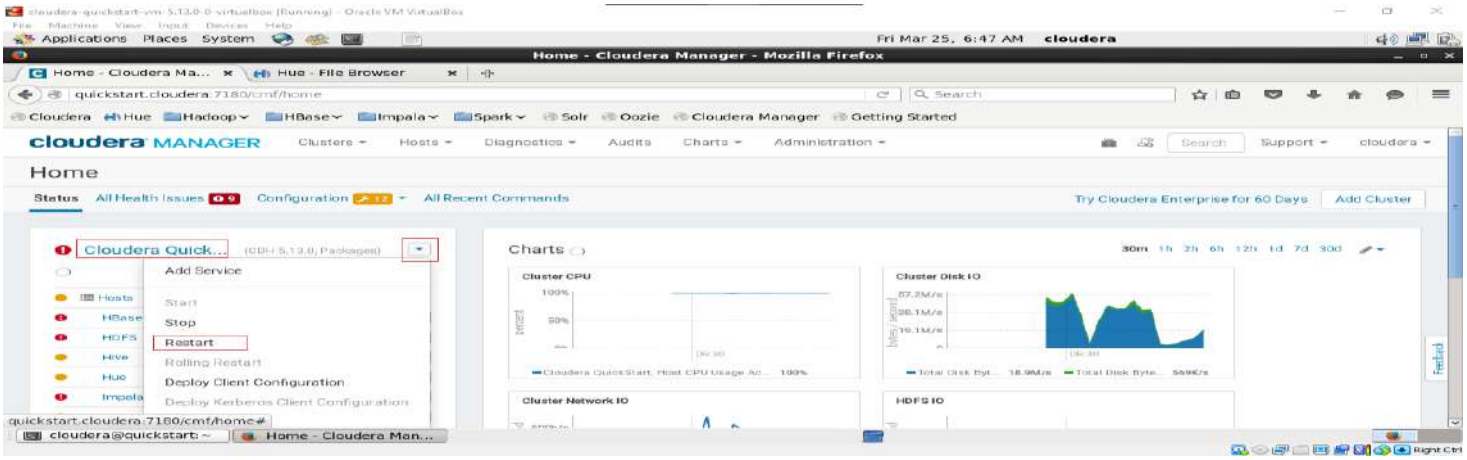
5. Creating **input.txt** file inside /usr/cloudera/training/**pig** directory



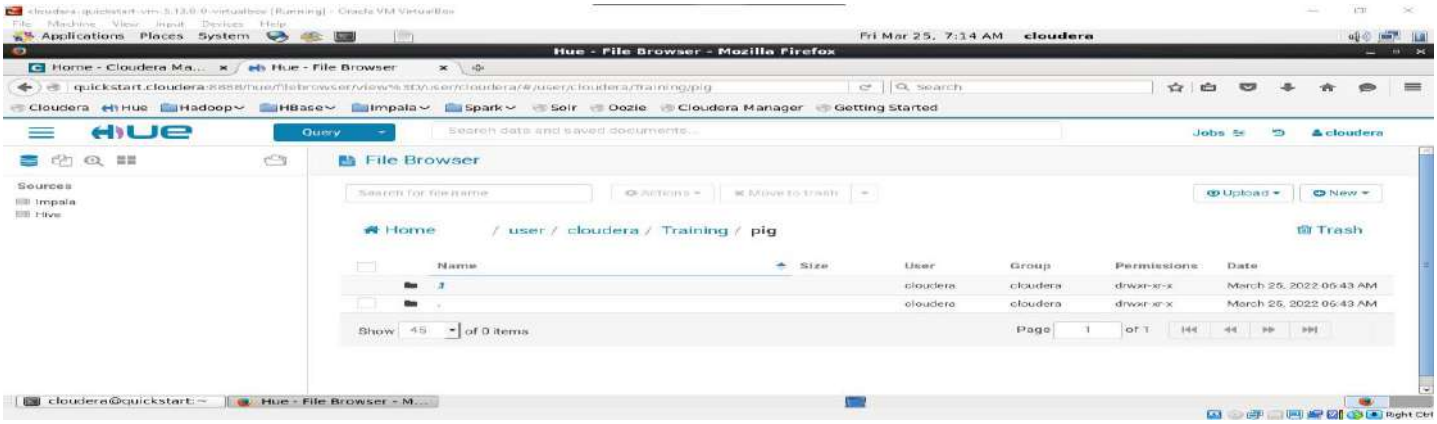


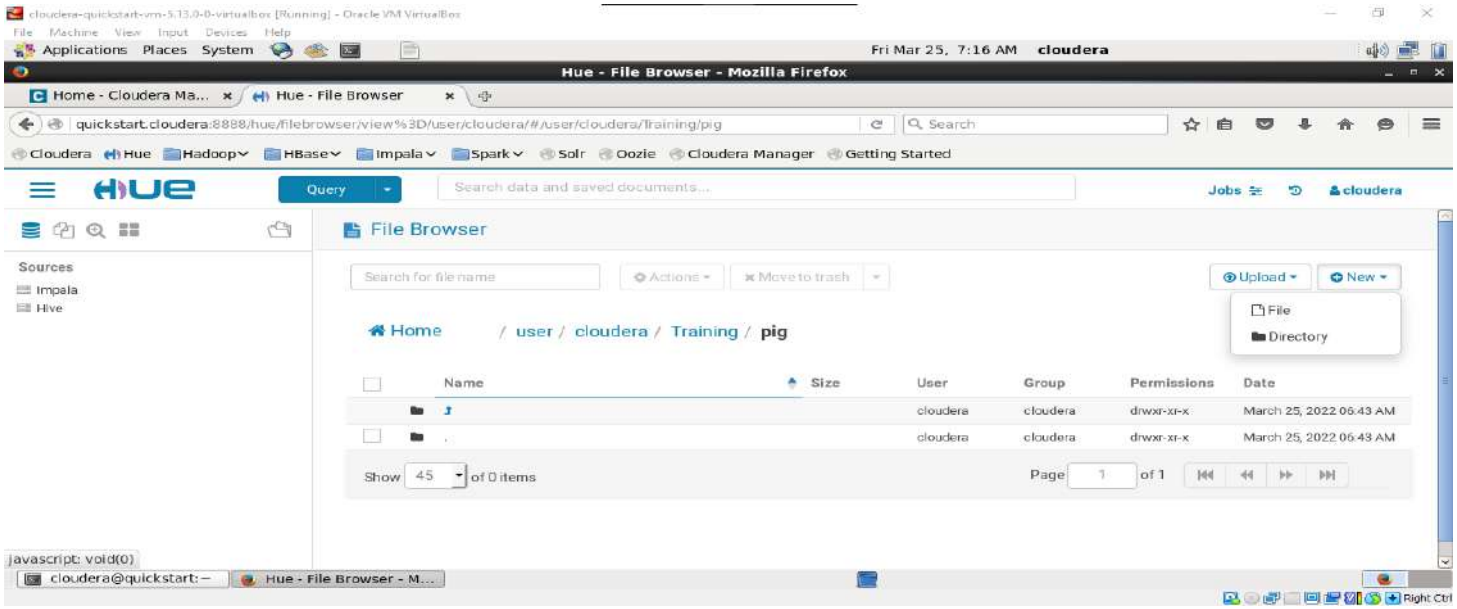
Note: If getting this type of error than start all services again

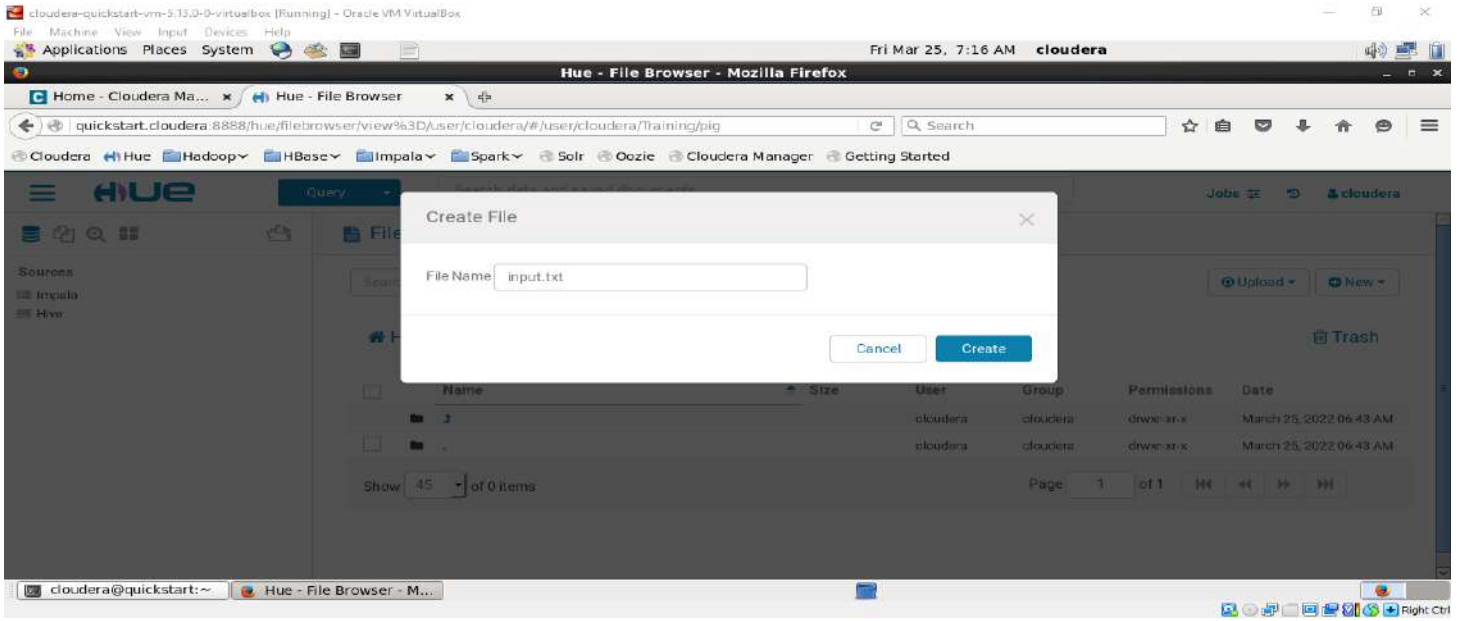




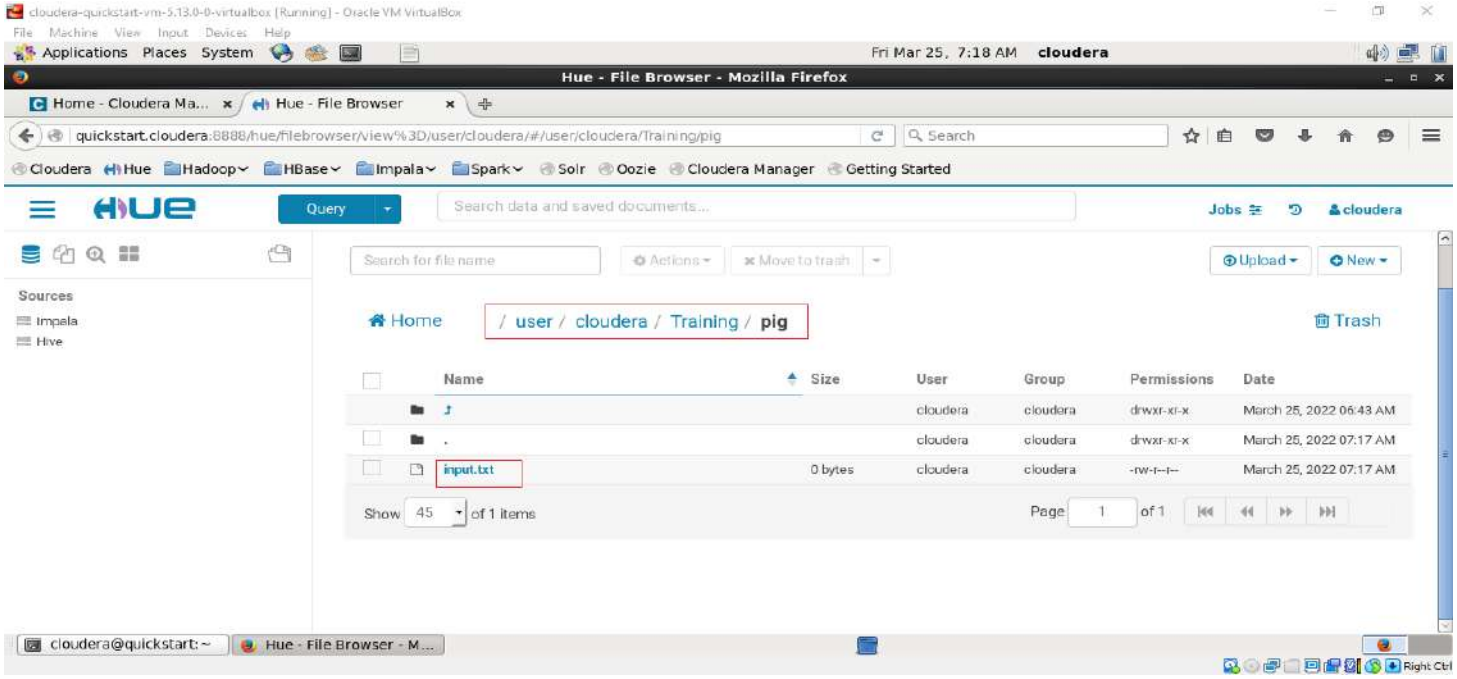
Refresh the Page



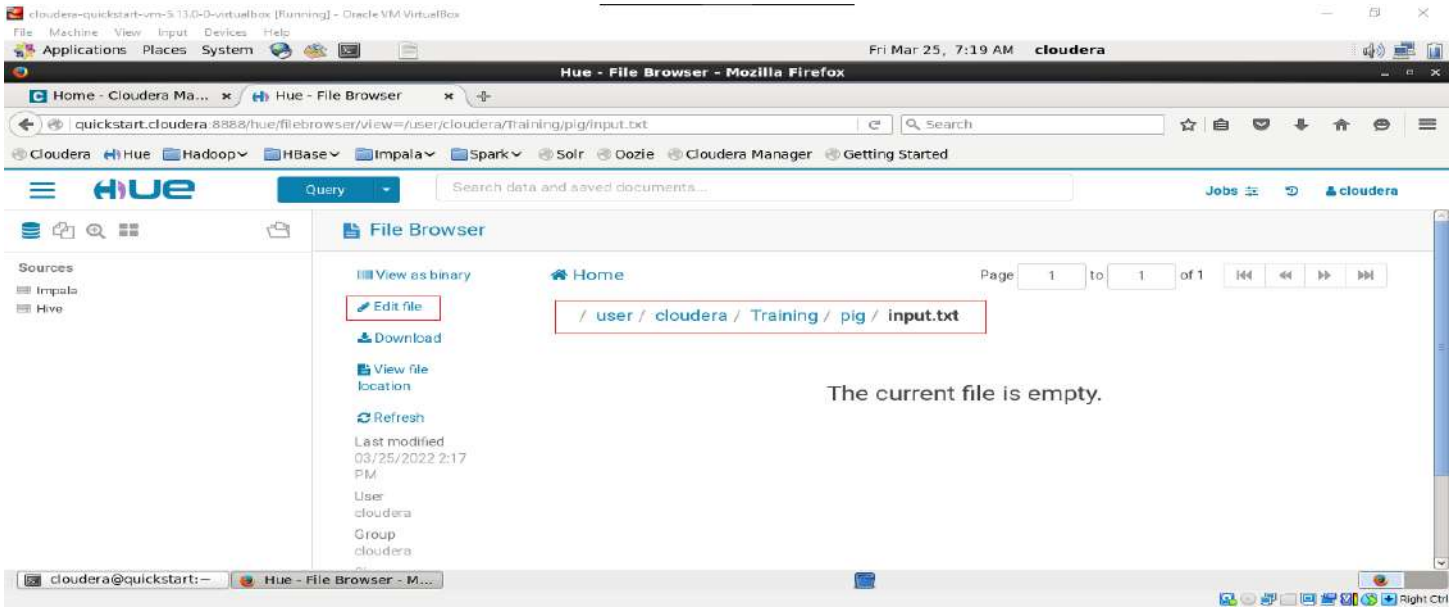




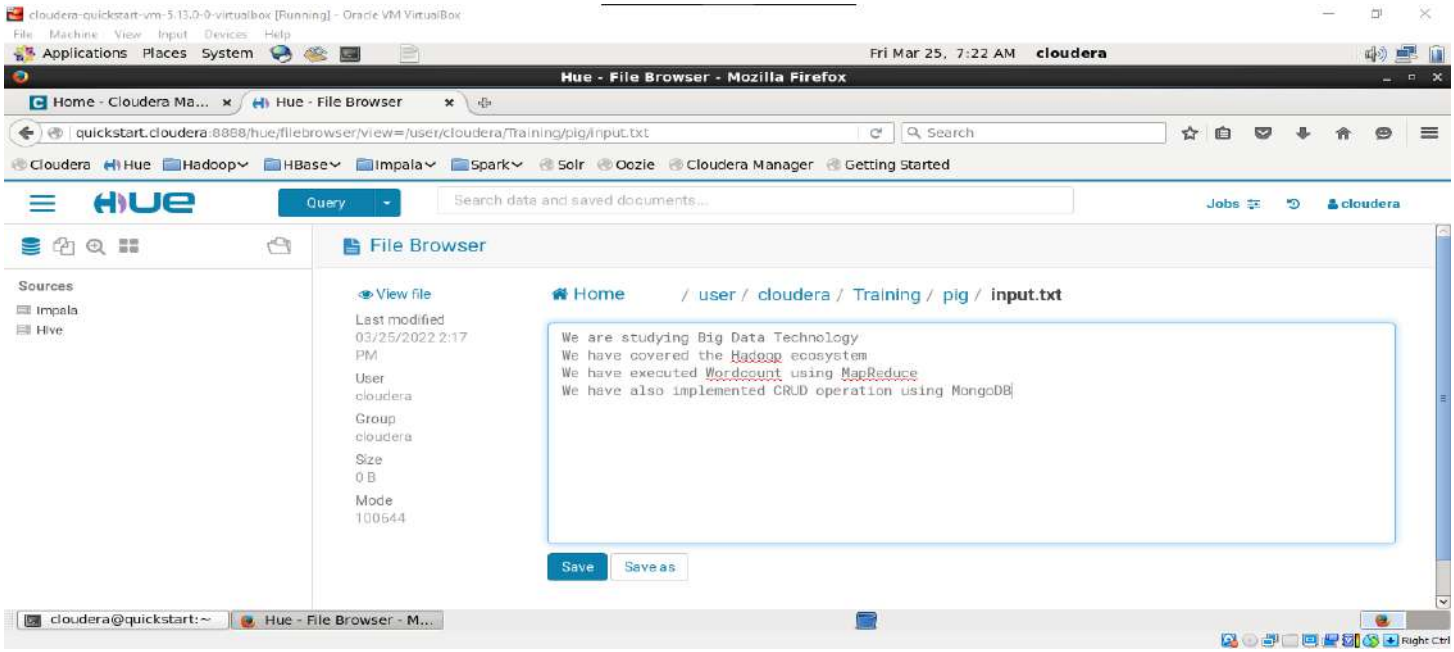
6. Adding some **contents** to this **input.txt** file.

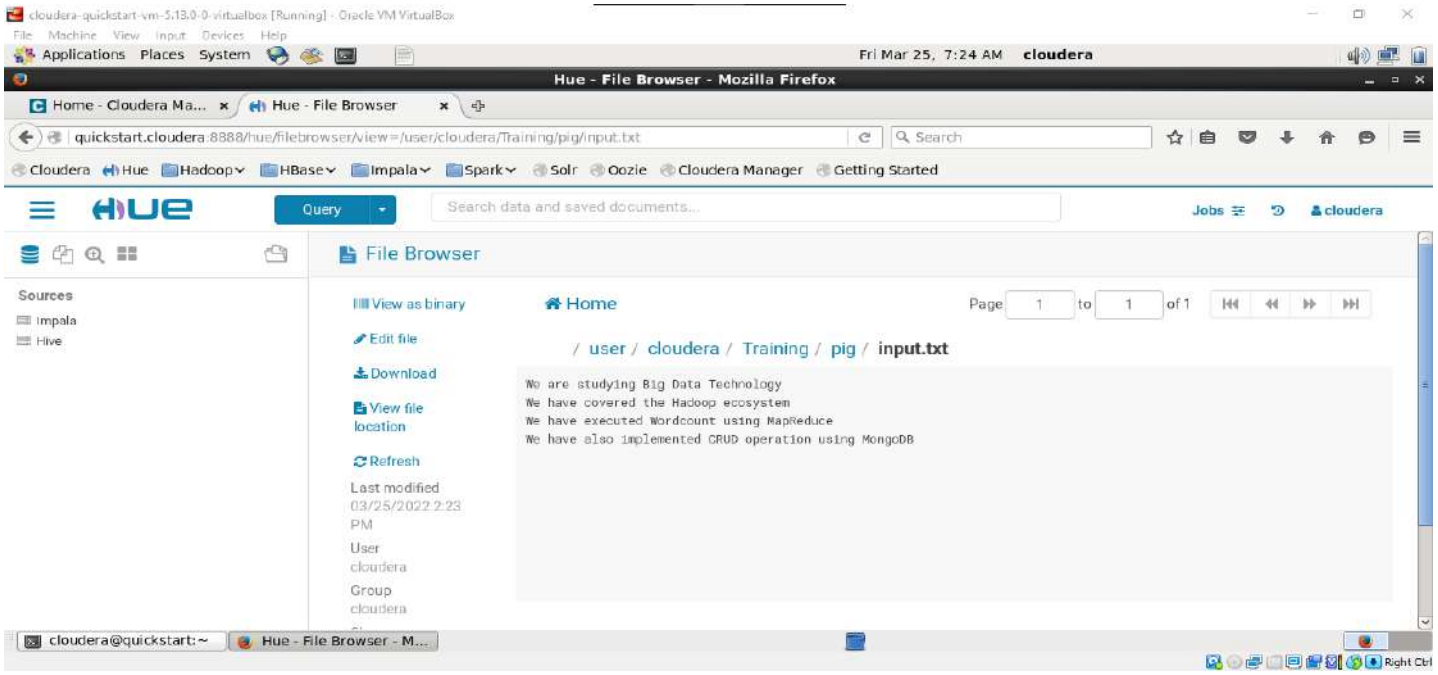


Click on **Edit file** option



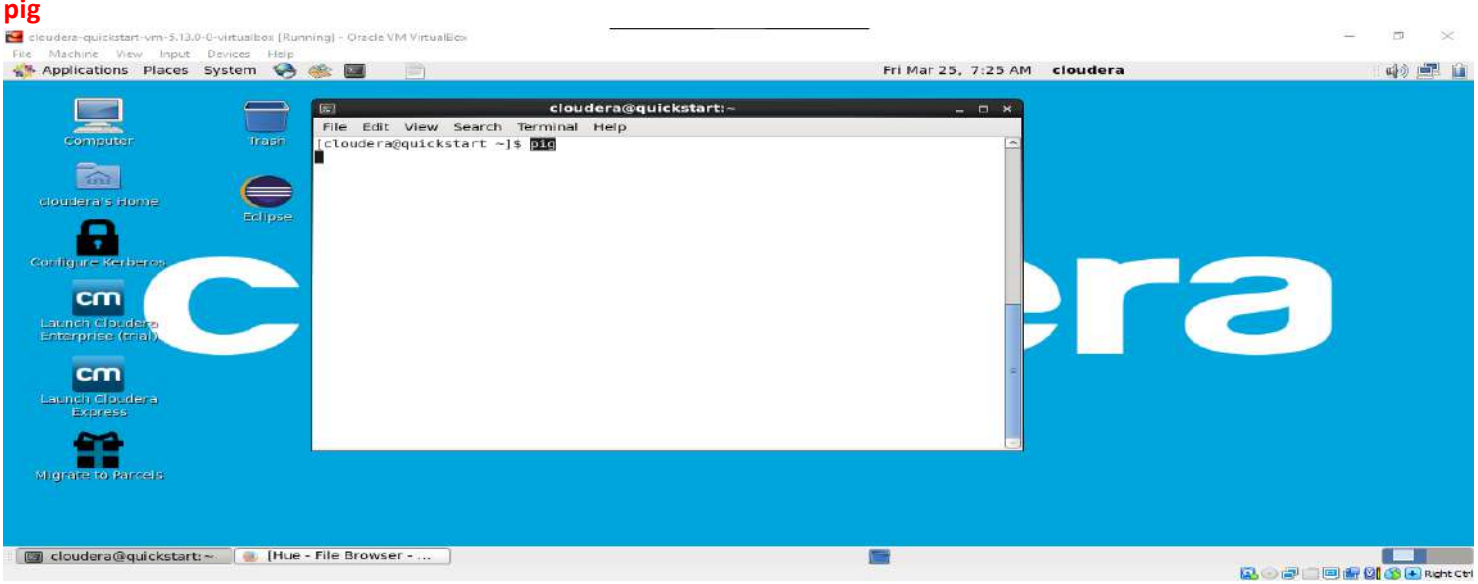
Save the input.txt file



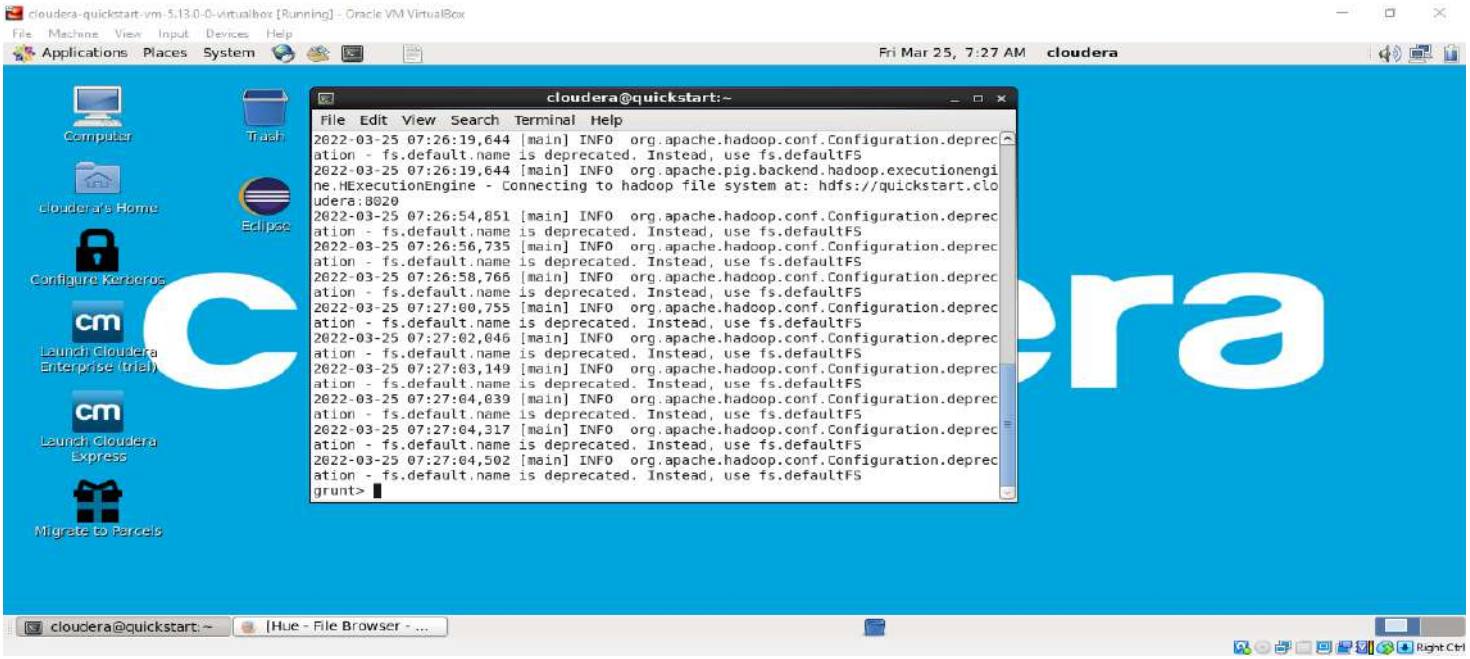


7. Now **Open** the terminal. And start **Pig** by typing pig on terminal.

pig

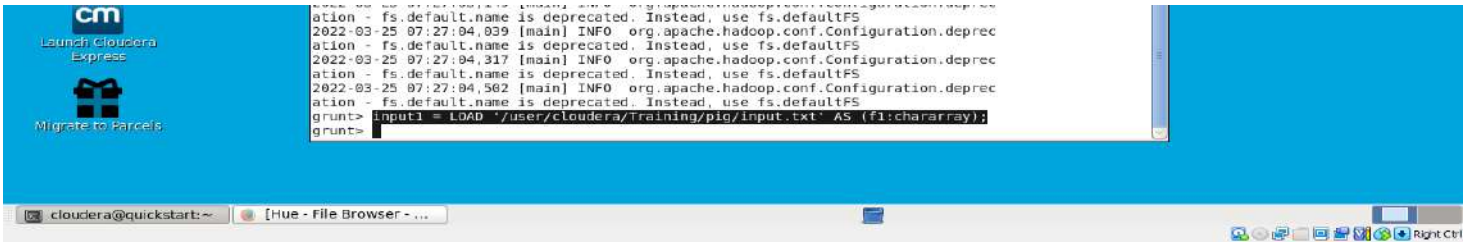


Now the pig started



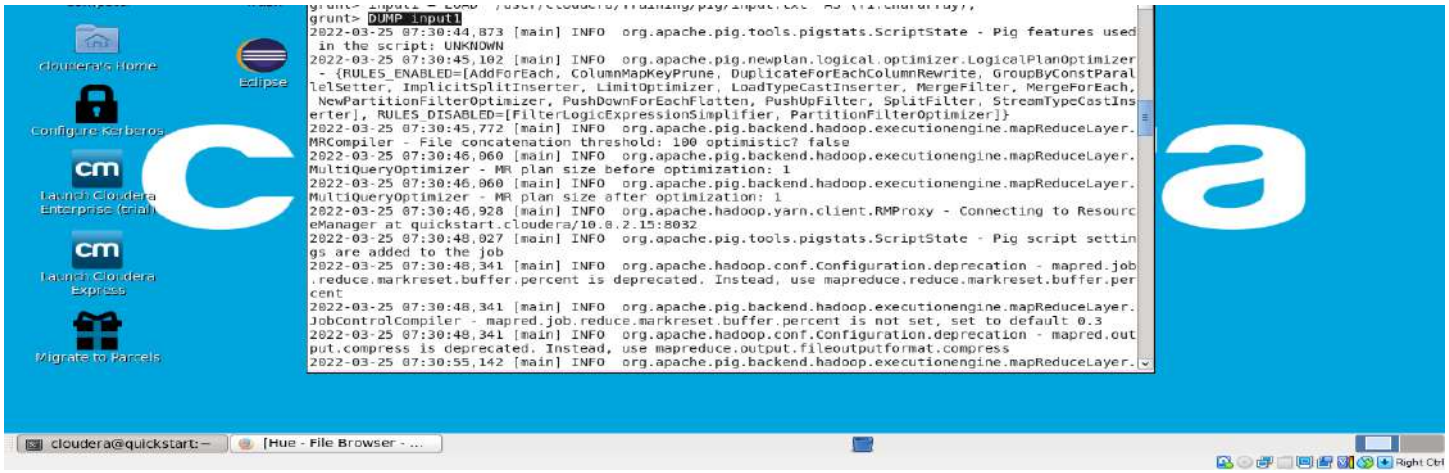
8. Now we have to load that input file where ever it is stored. By typing the command

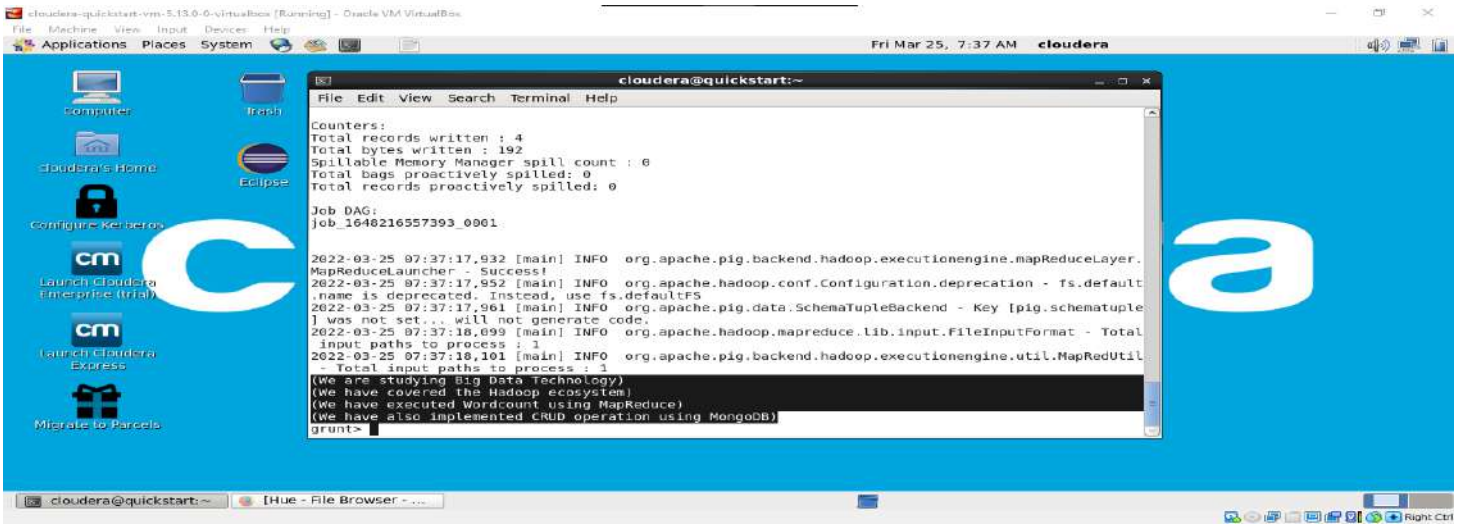
input1 = LOAD '/user/cloudera/Training/pig/input.txt' AS (f1:chararray);



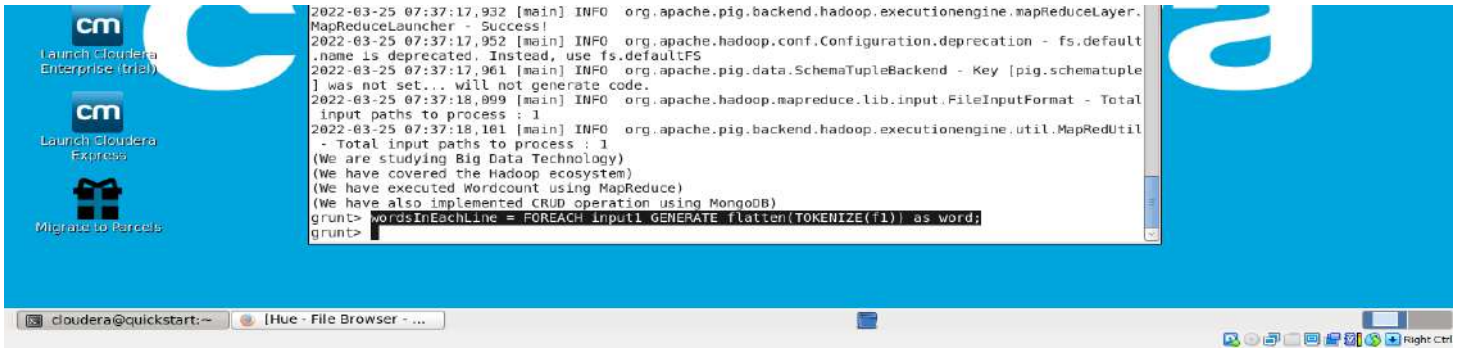
9. Now we are dumping the data. It will done the mapreduce task.

DUMP input1

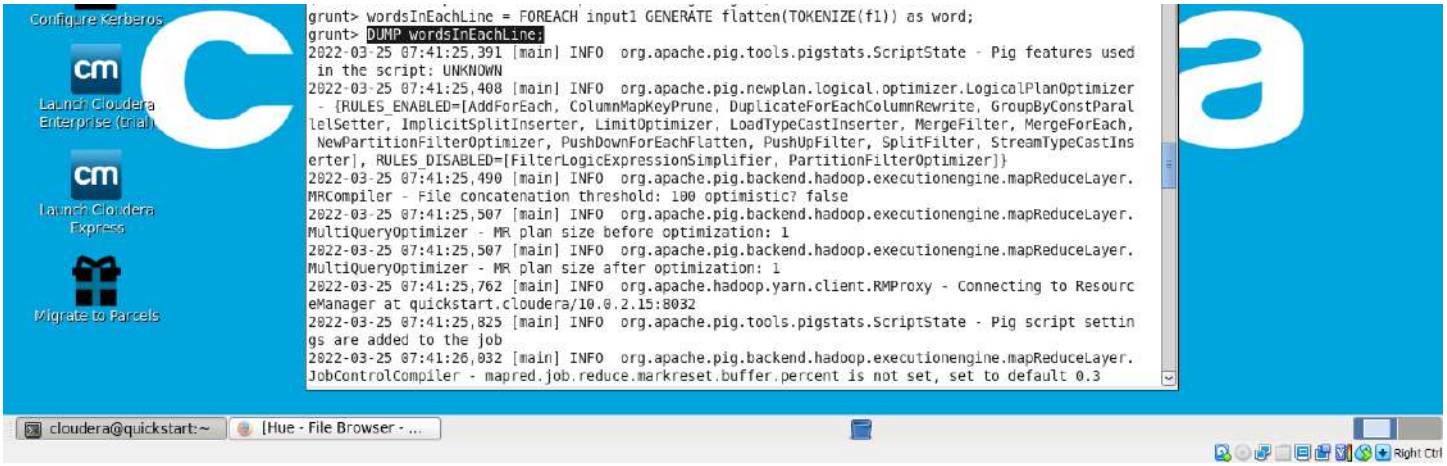


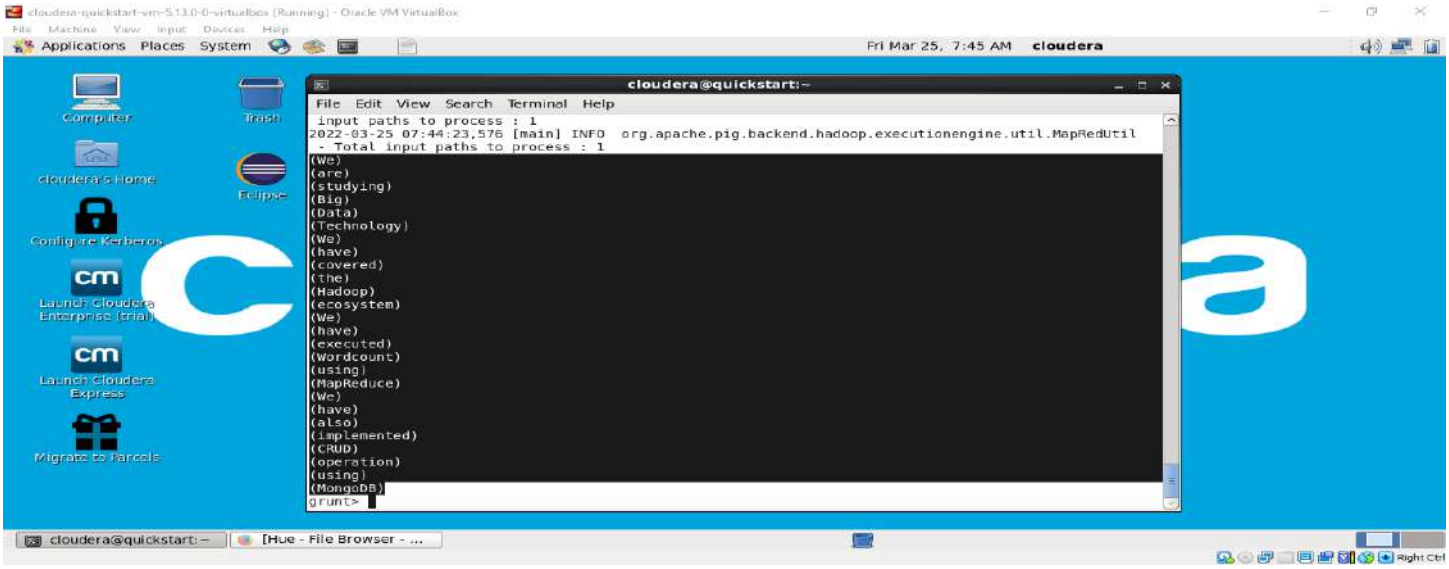


10. wordsInEachLine = FOREACH input1 GENERATE flatten(TOKENIZE(f1)) as word;



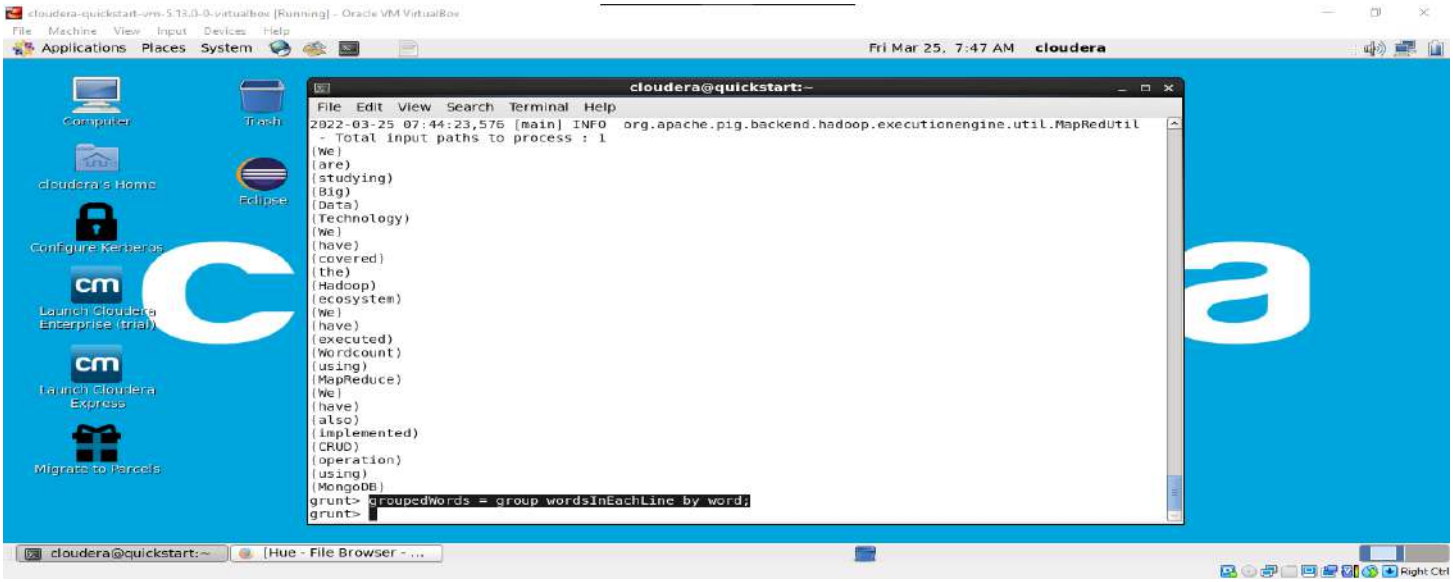
11. DUMP wordsInEachLine;



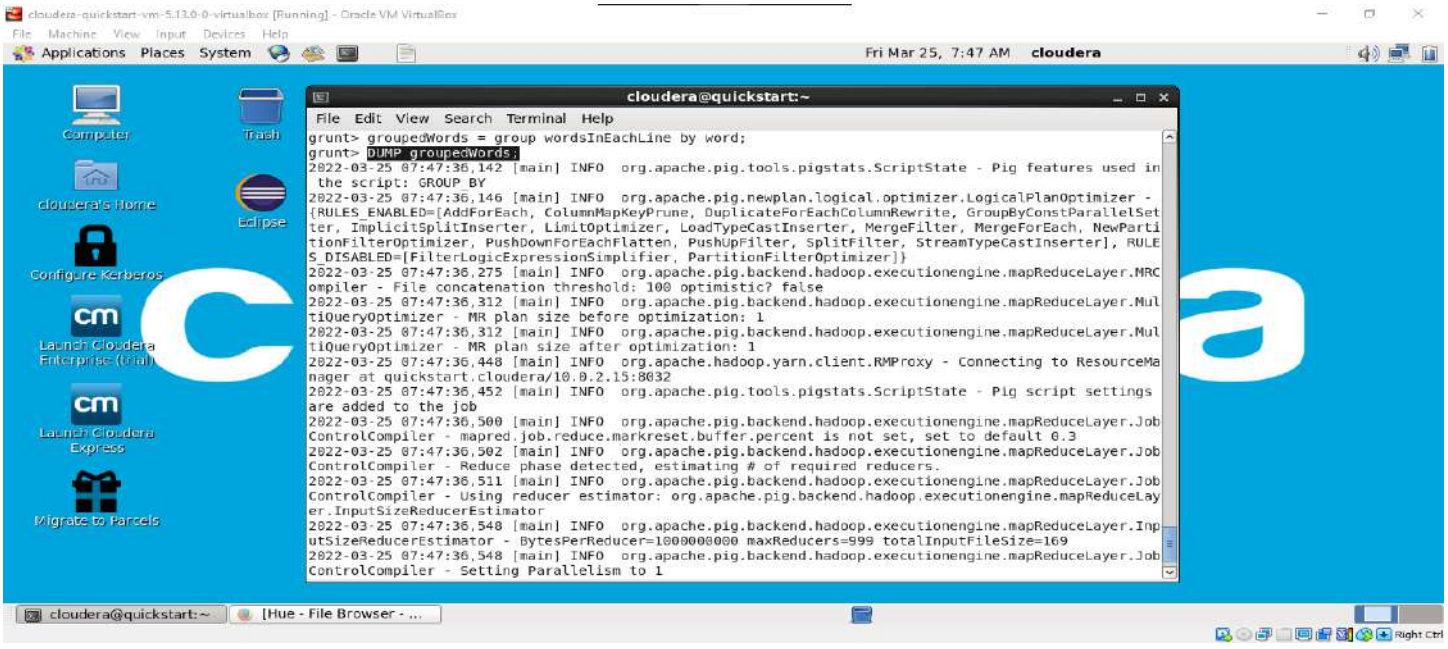


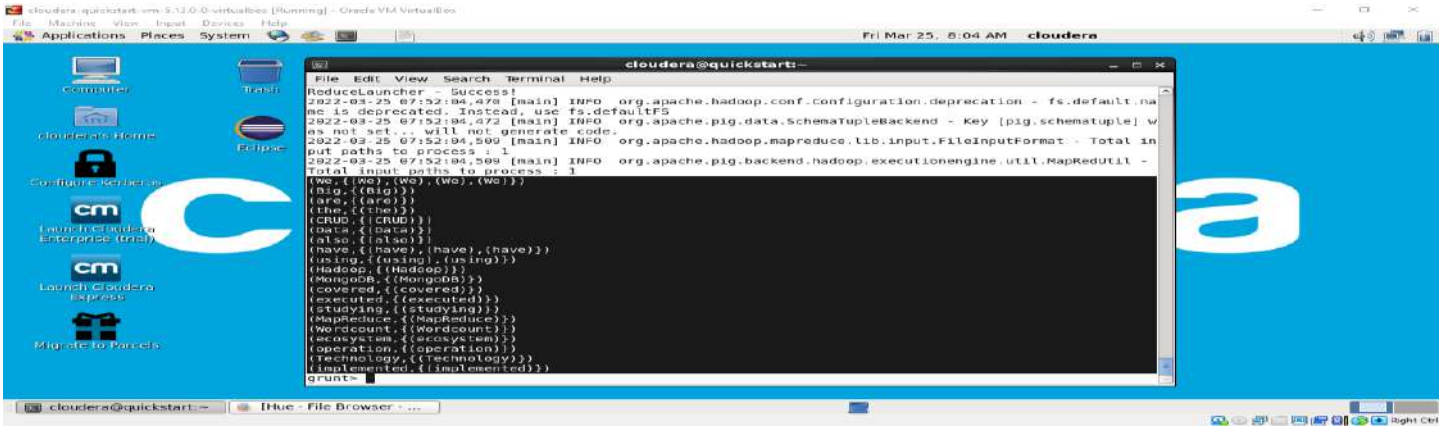
12. Now grouping the words present in each line.

groupedWords = group wordsInEachLine by word;



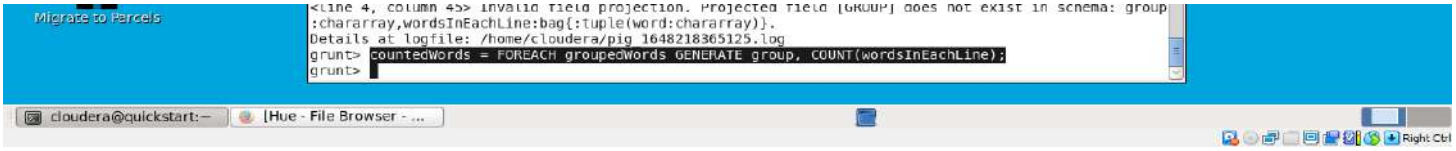
13. DUMP groupedWords;





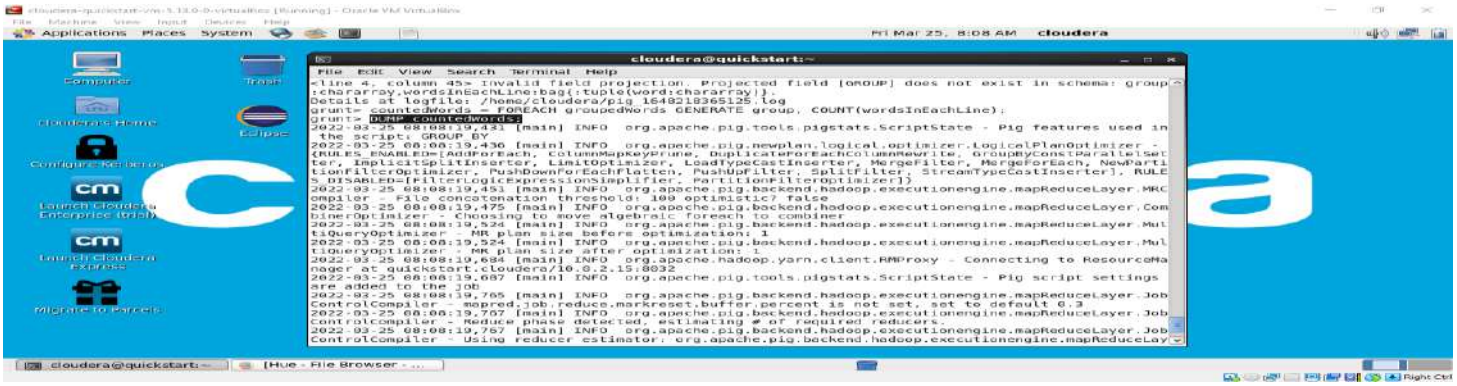
14. Now we count those words. For each group we count words in each line.

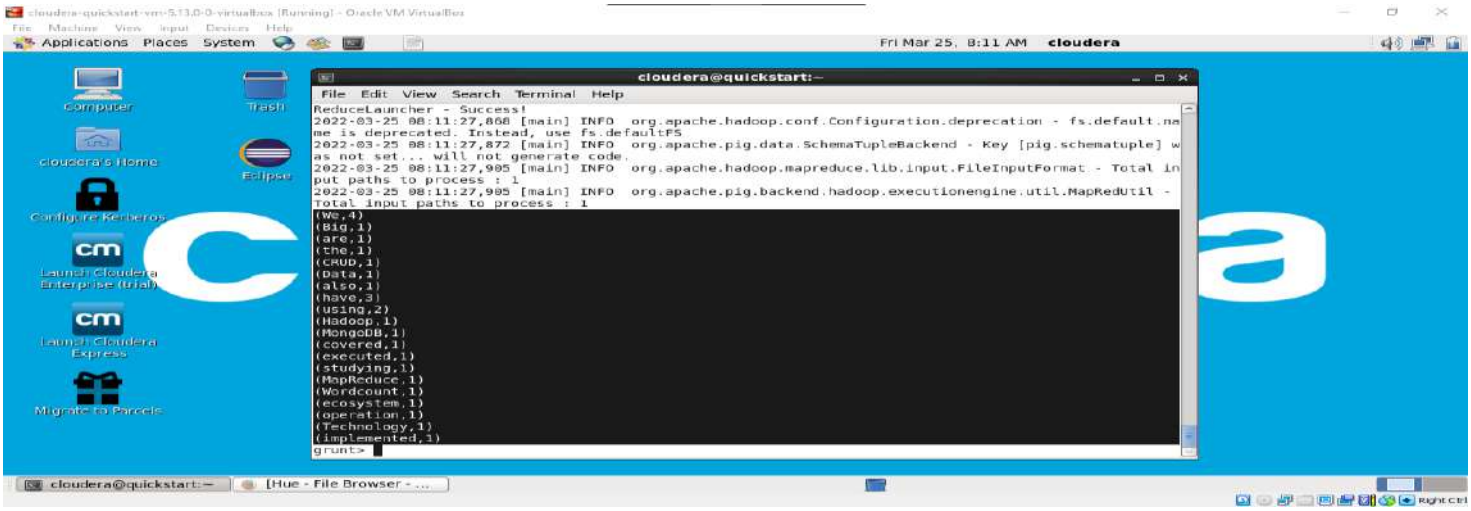
countedWords = FOREACH groupedWords GENERATE group, COUNT(wordsInEachLine);



15. DUMP countedWords;

Now the Final Output we are getting as word count for every word.





As we can see from above image the Word "We" start with capital W occurred four times, word "Big" occurred once, and so on.

Now Exit from the grunt shell using quit command.

16. quit

Aim: Demonstrate the use of Sqoop tool to transfer databetween Hadoop & relational database servers

Sqoop:

The traditional application management system, that is, the interaction of applications with relational database using RDBMS, is one of the sources that generate Big Data. Such Big Data,generated by RDBMS, is stored in Relational Database Servers in the relational database structure.

When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. ofthe Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. Here, Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop's HDFS.

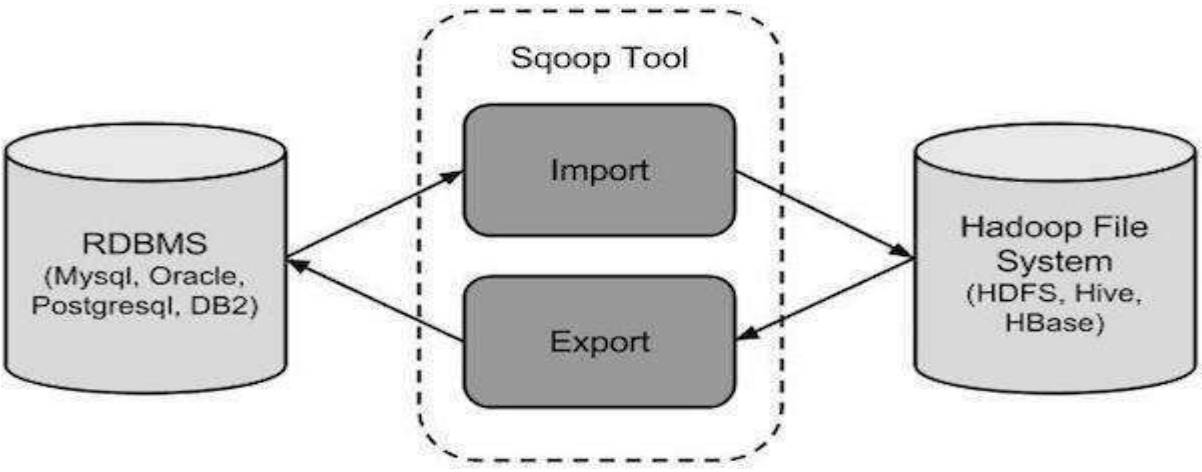
Sqoop – "SQL to Hadoop and Hadoop to SQL"

Sqoop is a tool designed to transfer data between Hadoop and external datastores suac asrelational databases and enterprise data warehouses.

It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, andexport from Hadoop file system to relational databases.

It imports data from external datastores into HDFS, Hive, and HBase.

Working of Sqoop:



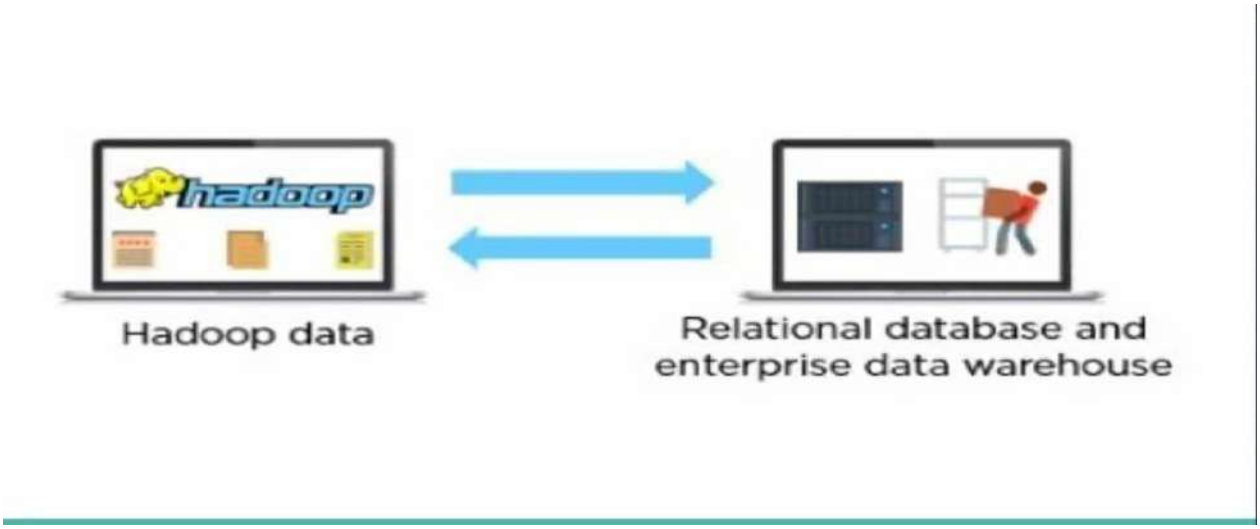
There are mainly two functions associated with Apache Sqoop tool which are Sqoop Import andSqoop export.

1. Sqoop Import

Sqoop Import This is an important function which executes the task of data importing from external sources (RDBMS) to HDFS. In HDFS, each row of a table is considered as a record. Theentire records are stored in a text format in the text files or as binary data in Sequence files.

2. Sqoop Export

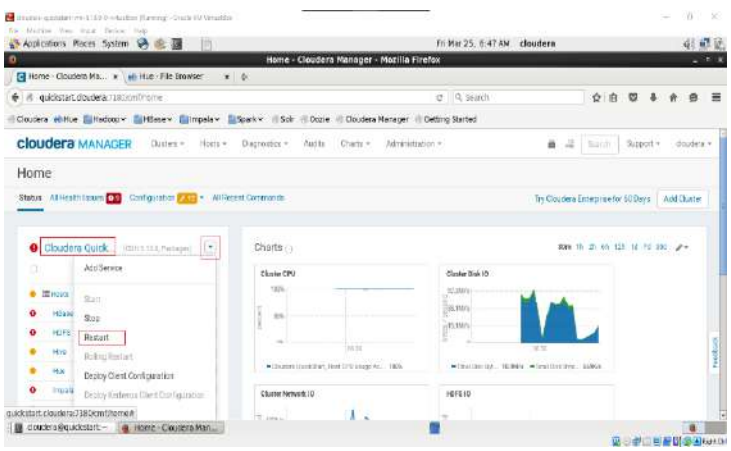
Sqoop Export This function performs bulk data exportation tasks from the HDFS to RDBMS. Once the modifications are done to the imported records you will get a result set and the nextprocess is to send back the data to the relational database (RDBMS). Sqoop export function reads a group of delimited text files from HDFS in parallel, divides the files into records, and stores them as new rows in a targeted database table.

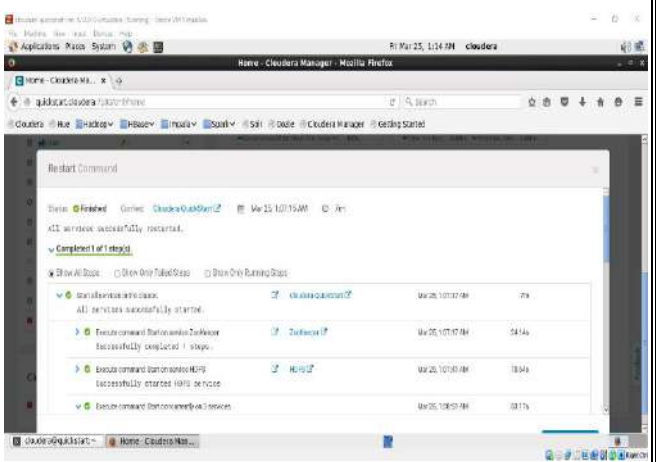


Before the perform the practical first perform <u>3 steps</u> of the given following

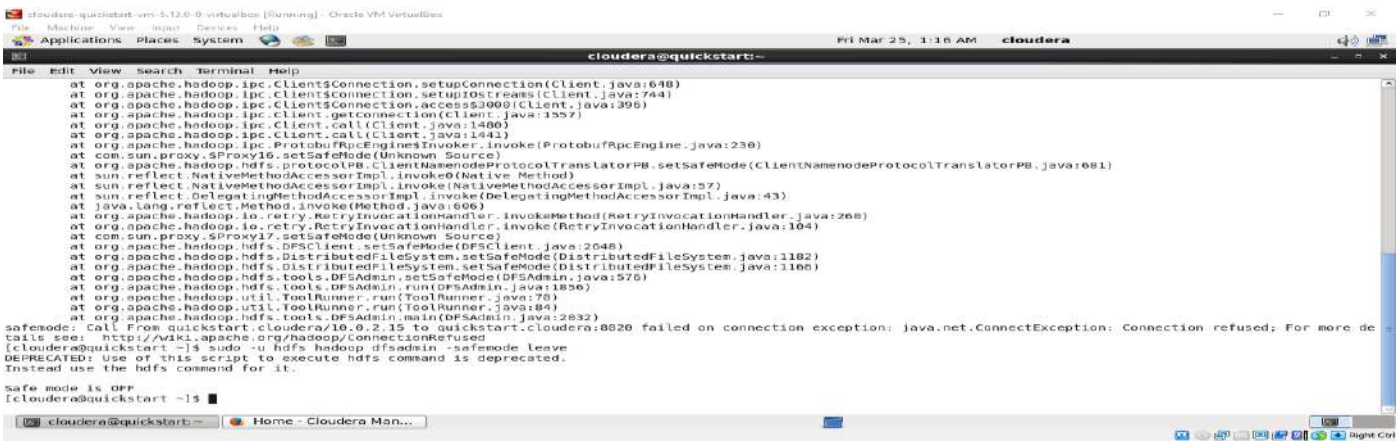
1. **sudo /home/cloudera/cloudera-manager --force --express**

2. **Start all services**





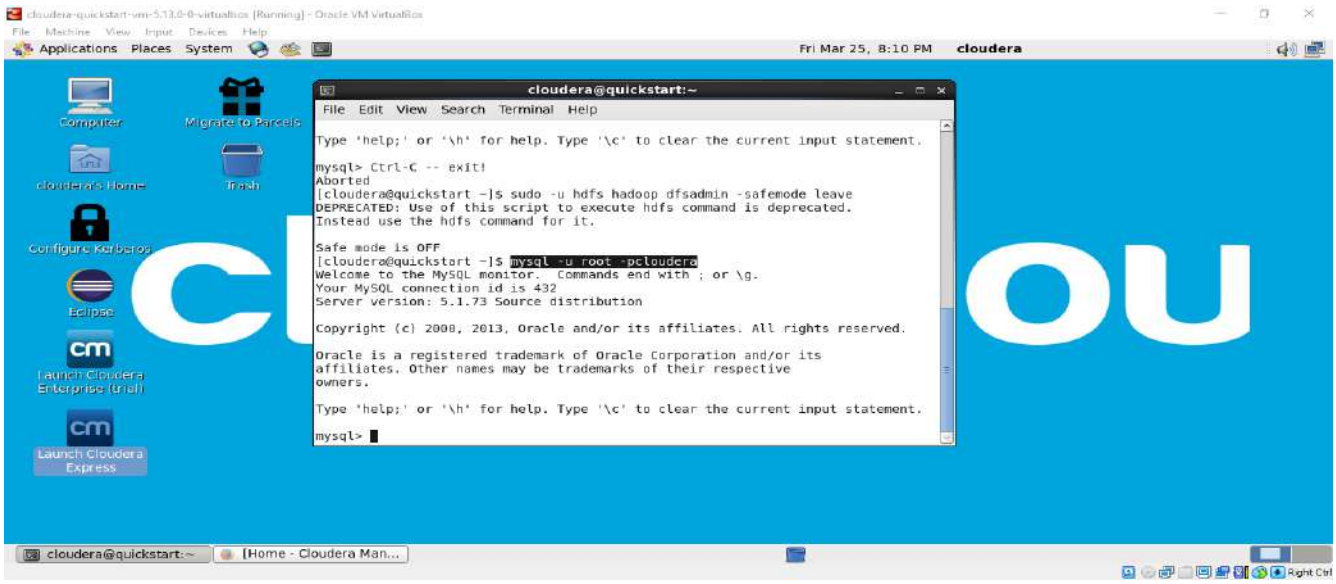
3. sudo -u hdfs hadoop dfsadmin -safemode leave



Steps: Demonstrate the use of Sqoop tool to transfer data between Hadoop &relational database servers

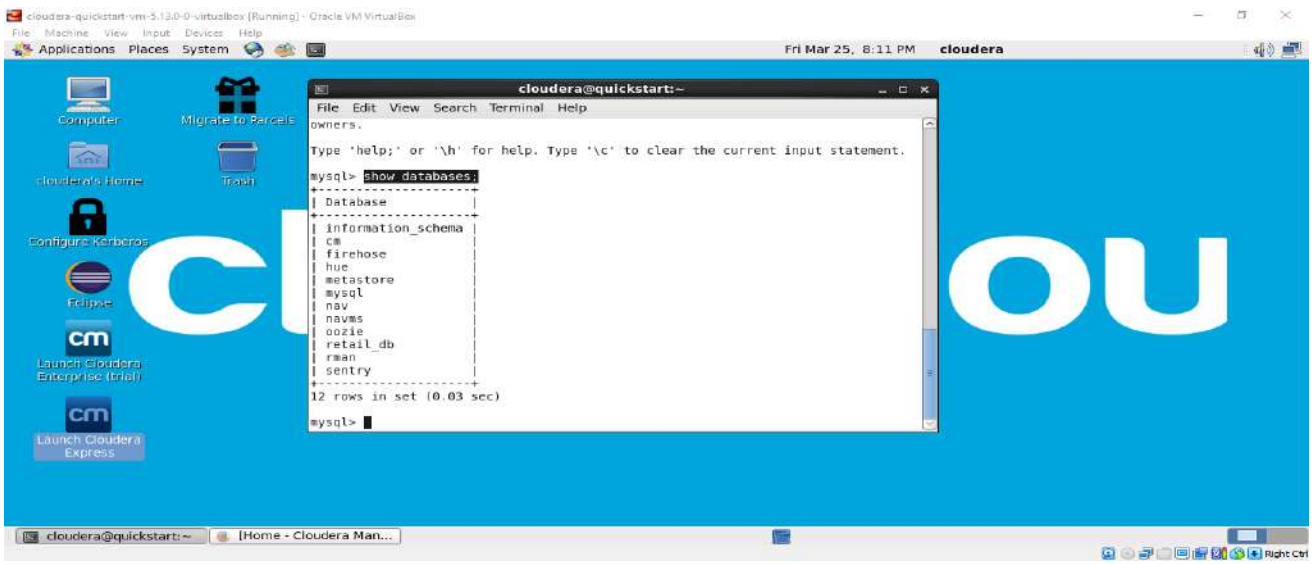
1. **Starting the mysql by giving username as root and password as cloudera.**

mysql -u root -pcloudera



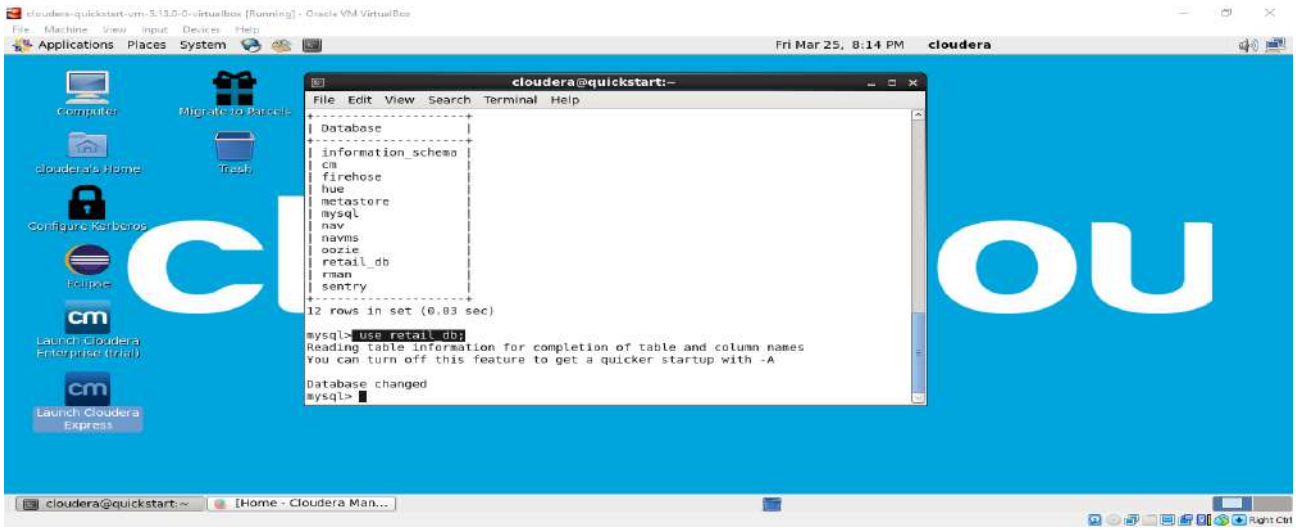
2. Now using below command it will displaying or give the list of databases which are alreadypresent or exist in mysql.

show databases;



3. Now we are using the existing database i.e. retail_db which are already present in mysql.

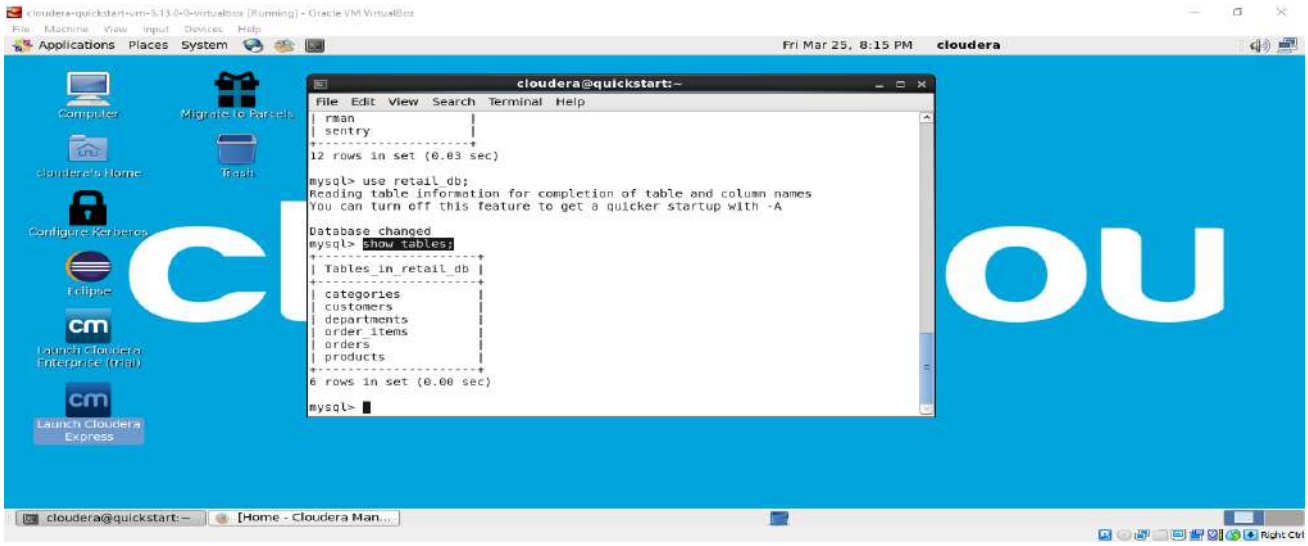
use retail_db;



So right now we are under retail_db database.

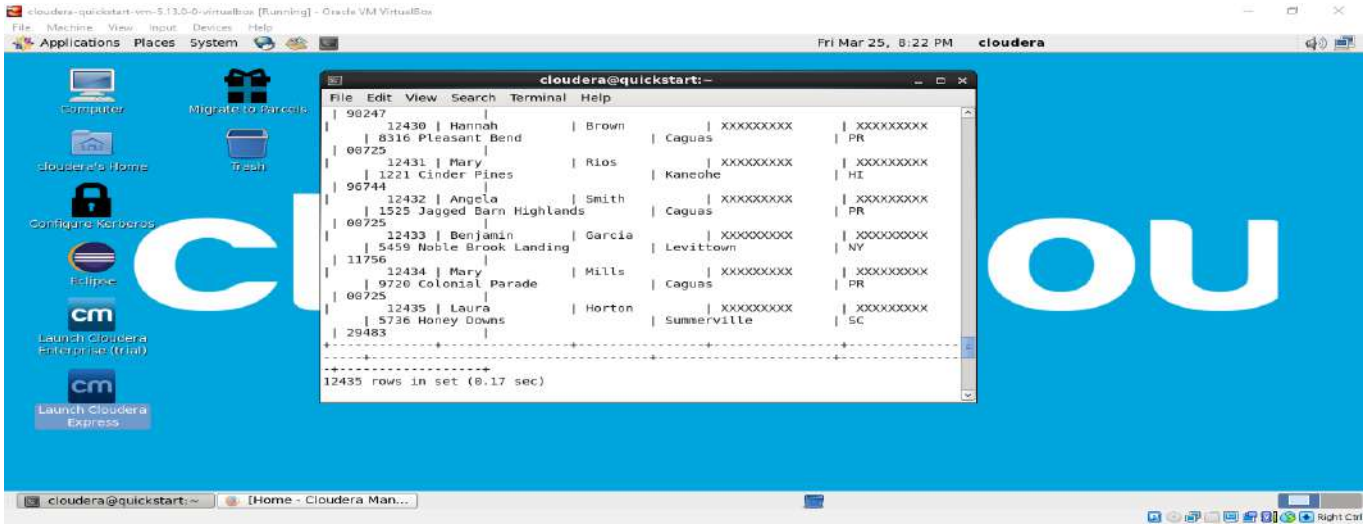
4. Now to see the tables under a specific database so we will be using the same commandwhich is used to display the databases.

show tables;



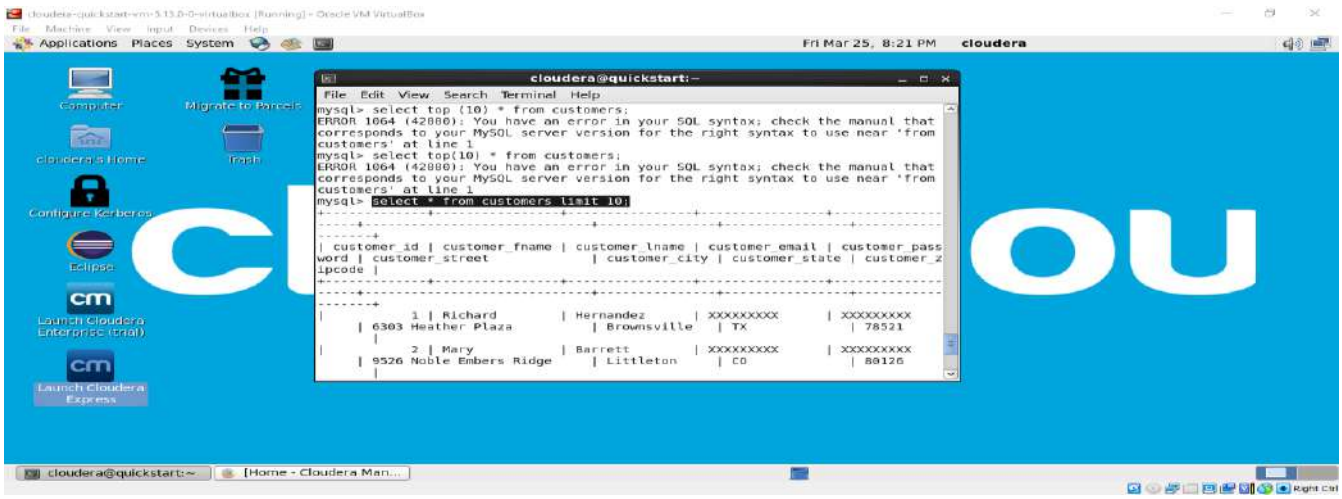
5. Here we are displaying all the records present in customers tables using below command.

select * from customers;



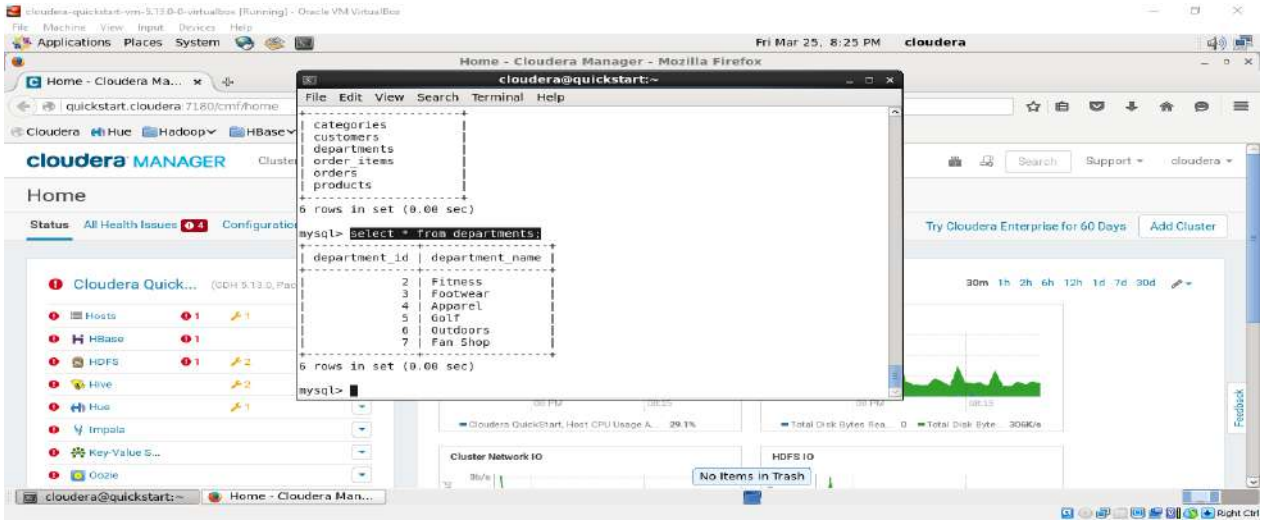
As it is a huge table. It contains total 12435 rows or records.

select * from customers limit 10;



6. Let see any other table. Here we are displaying department table it has 6 rows in it.

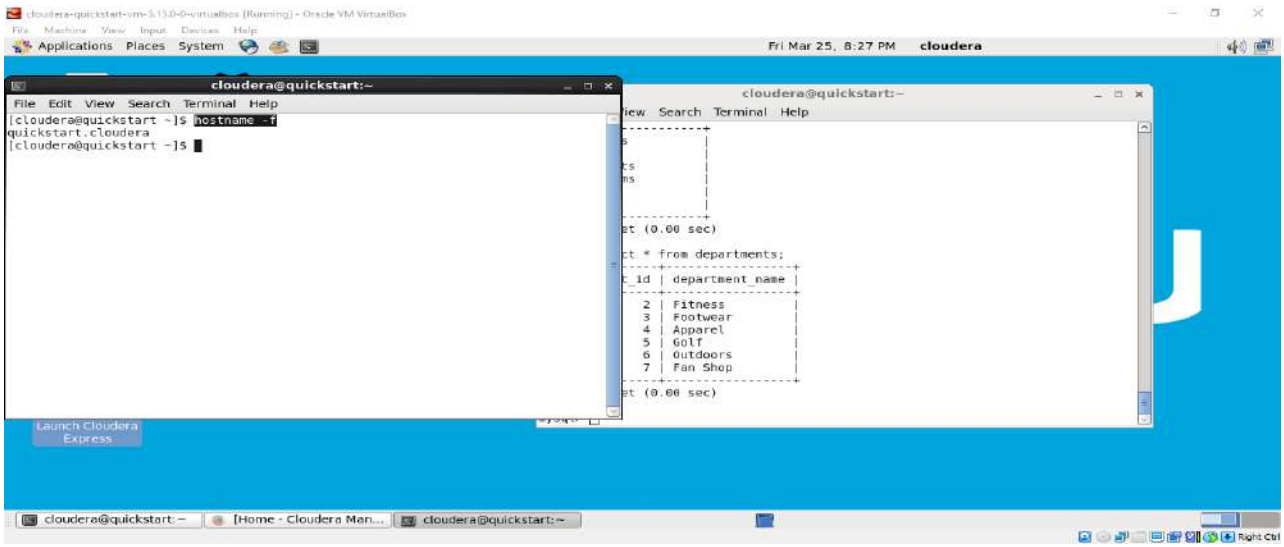
select * from departments;



These are the different departments i.e. department_name with their respectivedepartment_id.

7. Open the new terminal for running command for sqoop.
8. We will require hostname for this sqoop .

hostname -f

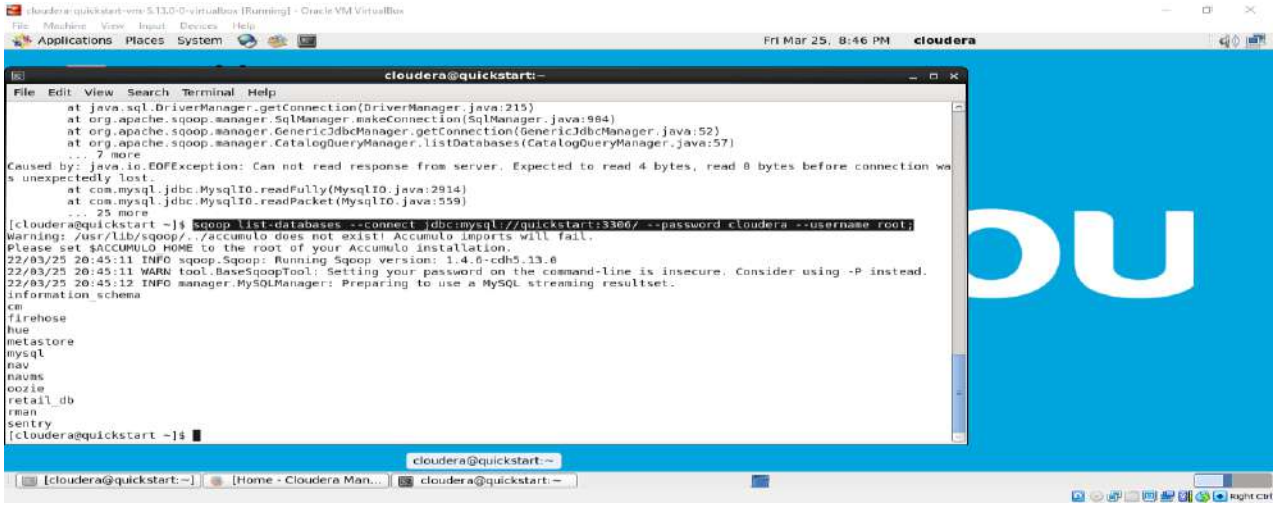


So it is giving as **quickstart.cloudera** as the name of the host that we are already connected to.

9. Then if we want to list down all the databases we will use below sqoop command.

sqoop list-databases --connect jdbc:mysql://quickstart:3306/ --password cloudera --username root;

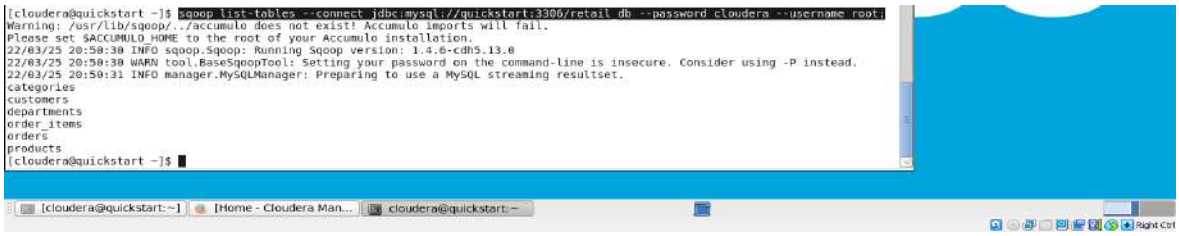
So we have studied in this sqoop that we can make use of jdbc or the odbc type driver. So the applications that support the jdbc will be connecting them with the jdbc driver. So here we areusing mysql which we are going to connect this with the jdbc . mysql running on **quickstart:3306** then we have to mention password which is **cloudera** and username i.e. **root**.



These are the lists of same databases which are present in mysql and the information alsopresent here. So we are connecting the sqoop with mysql with the help of jdbc.

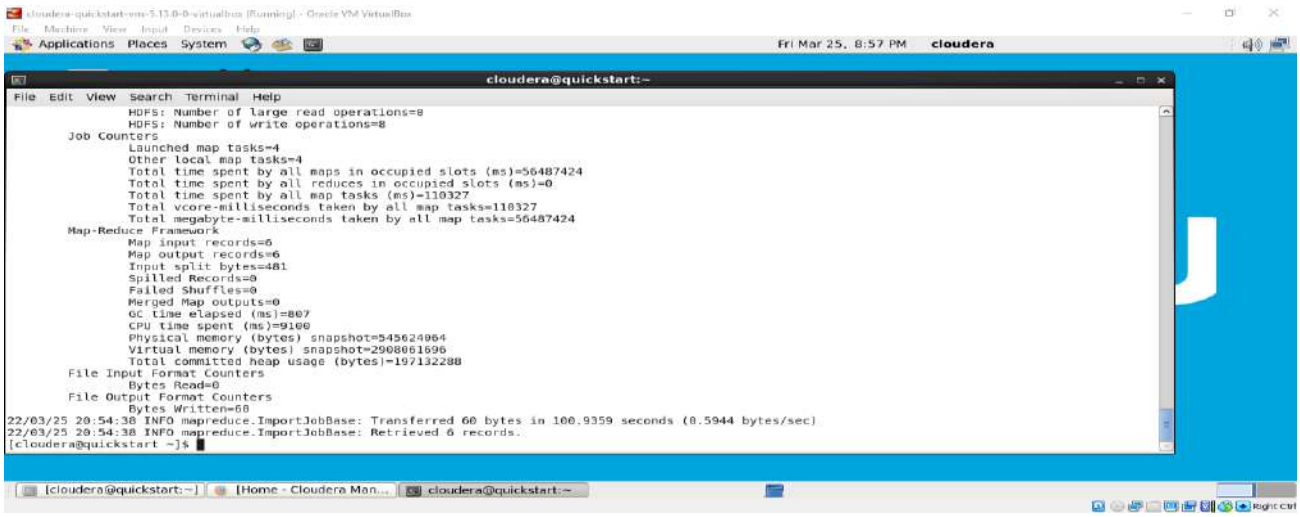
10. Now we will list out all the tables using below command.

sqoop list-tables --connect jdbc:mysql://quickstart:3306/retail_db --password cloudera --username root;



11. Now we will start with the import and export tools of the Hadoop. We wantto Import table "departments" from reatail_db database which are present inside in mysql.

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --password cloudera --username root --table departments;



12. Now we will see whether all departments table successfully imported from mysql in Hadoop (hdfs) or not using below command.

hadoop fs -ls

Departments table is now successfully imported in hdfs.

13. Now we will see what inside this deparment using below command.

hadoop fs -ls departments;

As we can see there are once SUCCESS file and four part-m files which has got the outputpresent inside the departments.

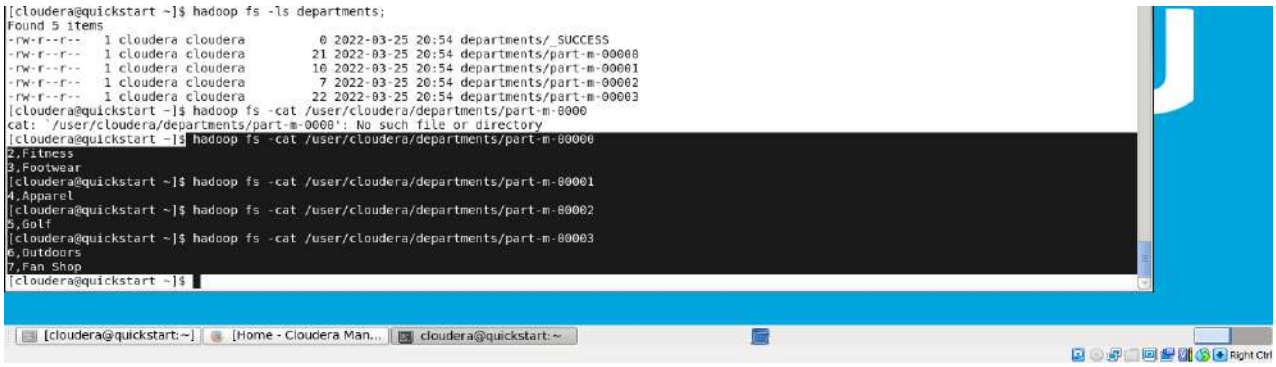
14. Now we will see what there inside this some of the part m files so that can be done with the help of below commands.

hadoop fs -cat /user/cloudera/departments/part-m-00000

hadoop fs -cat /user/cloudera/departments/part-m-00001

hadoop fs -cat /user/cloudera/departments/part-m-00002

hadoop fs -cat /user/cloudera/departments/part-m-00003



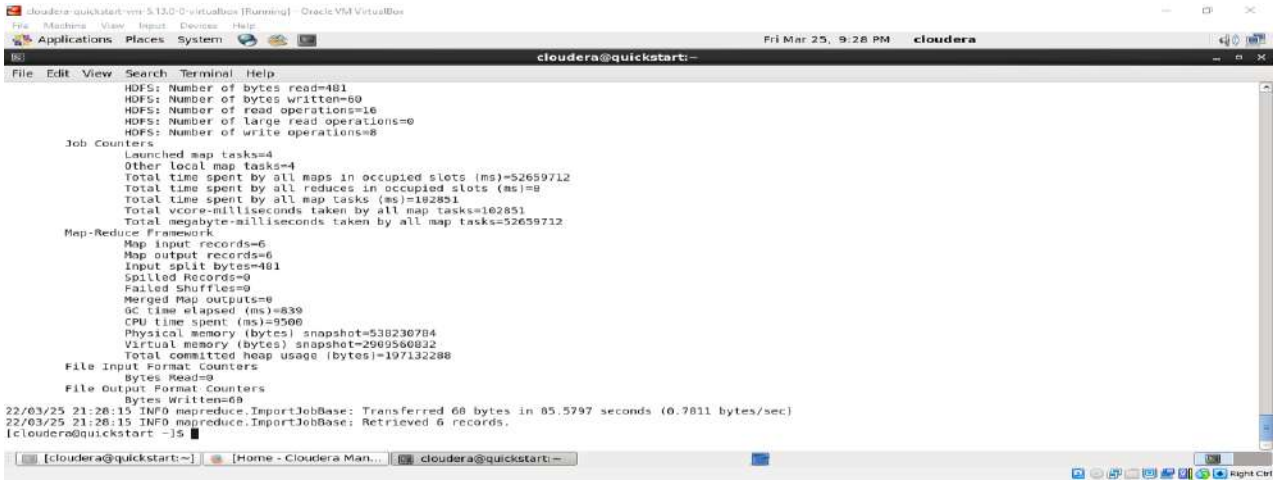
15. If want to display output of all part-m files together we will use below command.

hadoop fs -cat /user/cloudera/departments/part*

16. If we want to mention that where should we have our this output in the hdfs so for that we have to mention the target directory.

    We will want to import my department table in –target directory as department1.

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --password cloudera --username root --table departments --target-dir /user/cloudera/department1

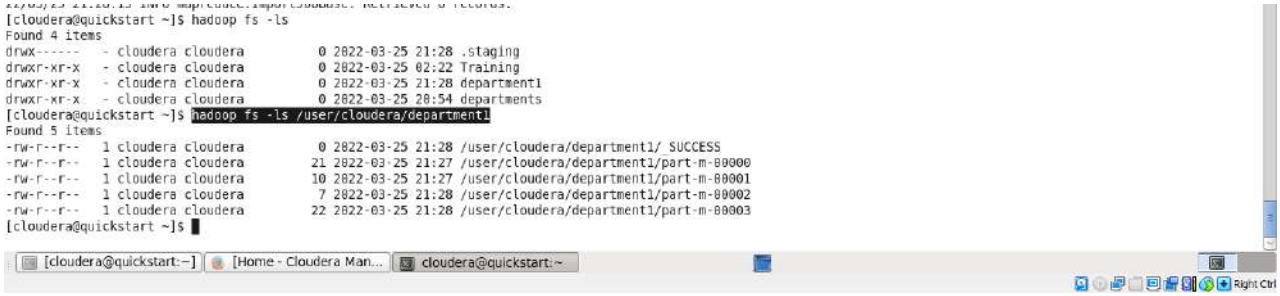


As we can see 6 records are retrived successfully.

17. Now let's check it using below command.
    hadoop fs -ls

So here we have department1 directory.

18. Now we will check what is present inside this department1 directory using belowcommand.
    hadoop fs -ls /user/cloudera/department1



So it has different part-m files.

19. Now we will read the content of these part-m files using below command.
    hadoop fs -cat /user/cloudera/department1/part*

So we have got the output.

20. Now we will filter out some or specific rows only from the departments table and have it inhdfs but

before we will apply some conditions on the rows of the departments table and whichever rows will satisfy the condition only those rows are would be stored in the hdfs.

We want to fetch only those departments where department_id is greater than 4.

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --password cloudera --username root --table departments --where "department_id>4" --target-dir /user/cloudera/department2

As we can see 3 records are retrieved successfully.

21. Now we will check it using below command.

hadoop fs -ls /user/cloudera/department2

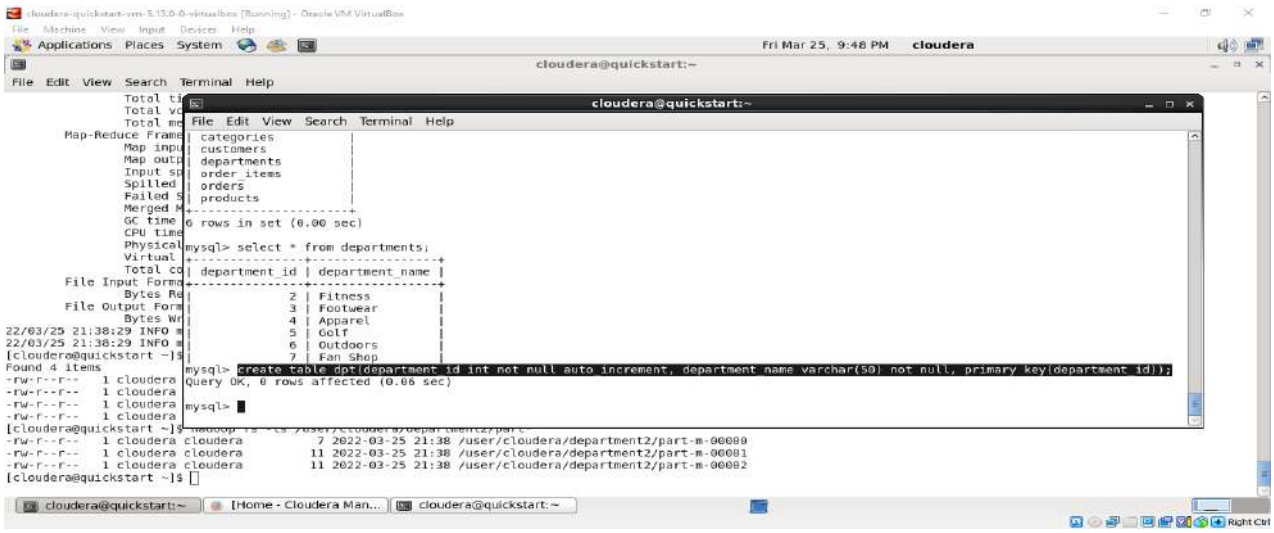
22. Now will read the content of these part-m files using cat command.

hadoop fs -ls /user/cloudera/department2/part*

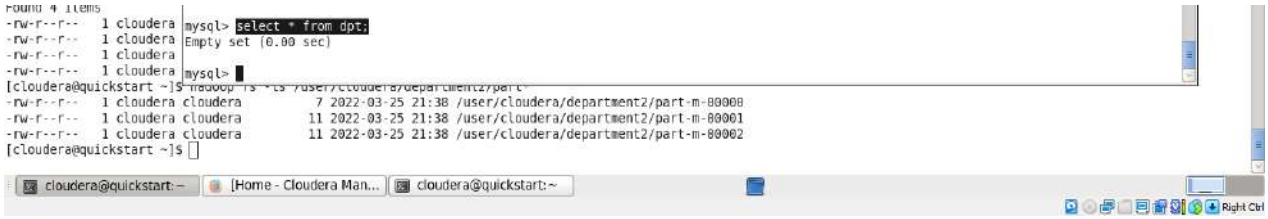
23. Now we will see the Export command. So what the export tool does is it willexport the data from our hdfs to the RDBMS. So for that we need to have some table in mysql with some records so for that we will now move to mysql.

24. So here we will create the table “dpt” and it will be having two attributes as “department_id” and “ department_name”.

create table dpt(department_id int not null auto_increment, department_name varchar(50) not null, primary key(department_id));



25. Now we want to check what we have inside this dpt table.
    select * from dpt;



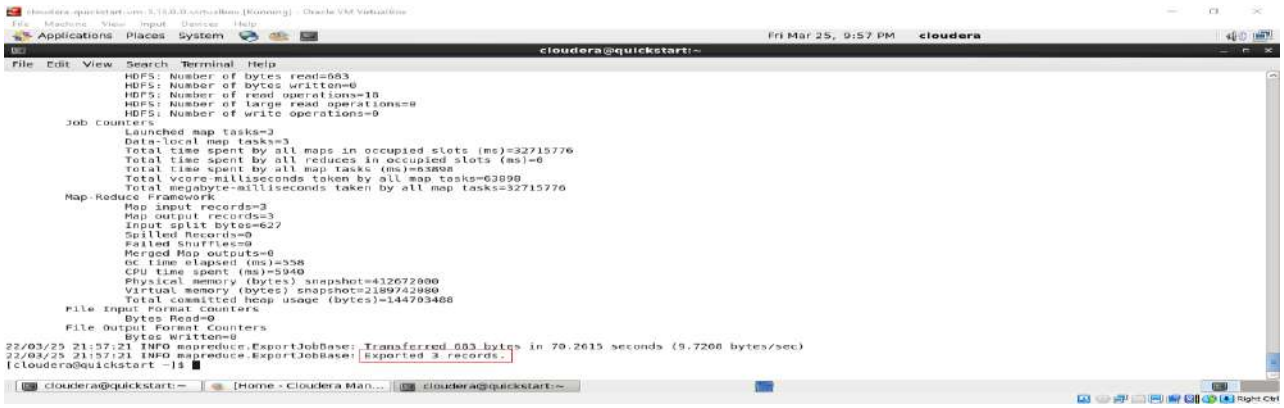
    As we have not inserted any records inside the dpt table so that's why it is showing as Empty set.

26. Now we will exporting the data from the hdfs to dpt table of mysql. Now we will move tothe sqoop terminal.

27. **So now we are performing export operation using below command.**

    we are trying to export department2 which are present in cloudera to inside our dpt tablewhich are present inside mysql.

    sqoop export --connect jdbc:mysql://quickstart:3306/retail_db --password cloudera --username root --table dpt --export-dir /user/cloudera/department2



    Now we have successfully exported 3 records.

28. Now we will see whether the records are successfully exported in dpt table which arepresent inside mysql using below command.
    select * from dpt;

    As we can see these 3 records which are present in department2 table are successfullyexported inside the dpt table of mysql.

    quit

Aim - Schema design using HBaseWhat is HBase?

HBase is a column-oriented non-relational database management system that runs on topof Hadoop Distributed File System (HDFS). HBase provides a fault-tolerant way of storing sparse data sets, which are common in many big data use cases. It is well suited for real- time data processing or random read/write access to large volumes of data.

Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java™ much like a typical Apache MapReduce application. HBase does support writing applicationsin Apache Avro, REST and Thrift.

An HBase system is designed to scale linearly. It comprises a set of standard tables with rows and columns, much like a traditional database. Each table must have an element defined as a primary key, and all access attempts to HBase tables must use this primary key.

Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations andrecords. A sort order can also be defined for the data.

HBase relies on ZooKeeper for high-performance coordination. ZooKeeper is built into HBase, but if you're running a production cluster, it's suggested that you have a dedicatedZooKeeper cluster that's integrated with your HBase cluster.

HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications.

An example of HBase

An HBase column represents an attribute of an object; if the table is storing diagnostic logsfrom servers in your environment, each row might be a log record, and a typical column could be the timestamp of when the log record was written, or the server name where therecord originated.

HBase allows for many attributes to be grouped together into column families, such that theelements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together. With HBase you must predefine the table schema and specify the column families. However, new columns can be added to families at any time, making the schema flexible and able to adapt to changing application requirements.

Just as HDFS has a NameNode and slave nodes, and MapReduce has JobTracker and TaskTracker slaves, HBase is built on similar concepts. In HBase a master node manages thecluster and region servers store portions of the tables and perform the work on the data. Inthe same way HDFS has some enterprise concerns due to the availability of the NameNode HBase is also sensitive to the loss of its master node.

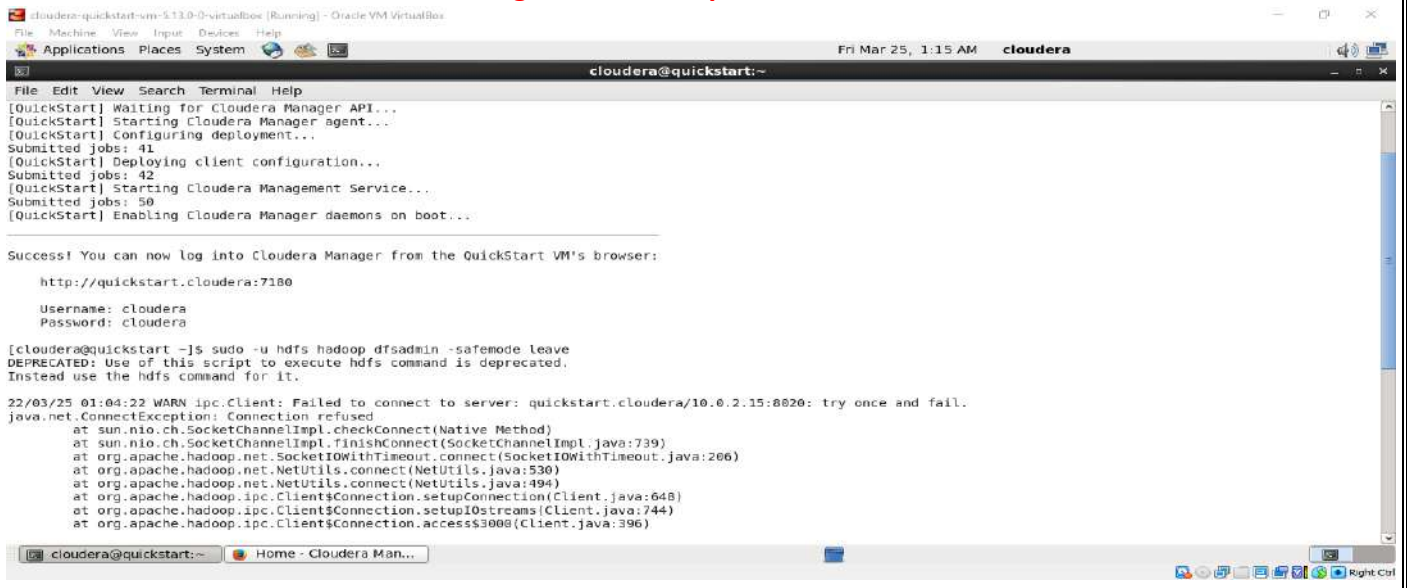
HBase Shell

HBase contains a shell using which you can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and storedin region servers.

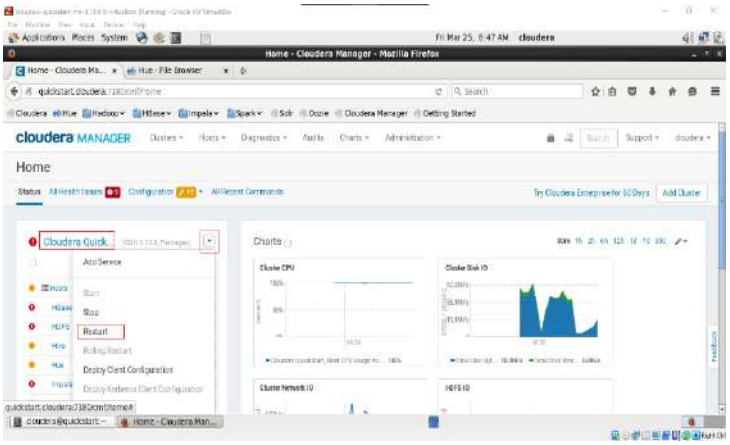
The master server manages these region servers and all these tasks take place on HDFS.Given below are some of the commands supported by HBase Shell.

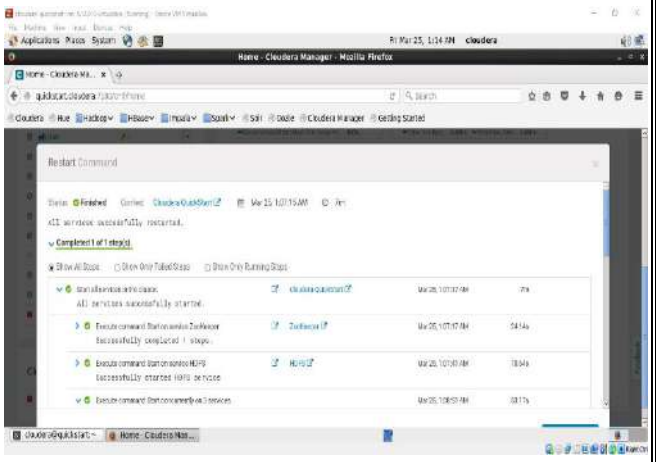
Before the perform the practical first perform <u>3 steps</u> of the given following

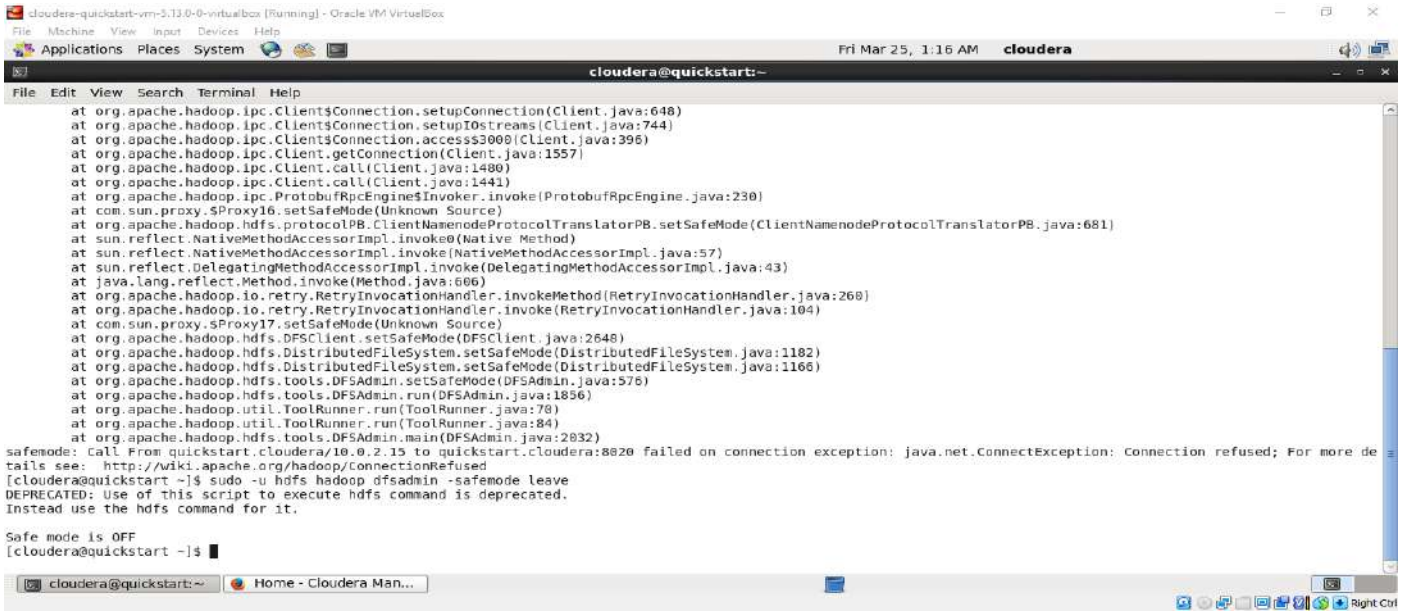
1. **sudo /home/cloudera/cloudera-manager --force --express**



2. **Start all services**



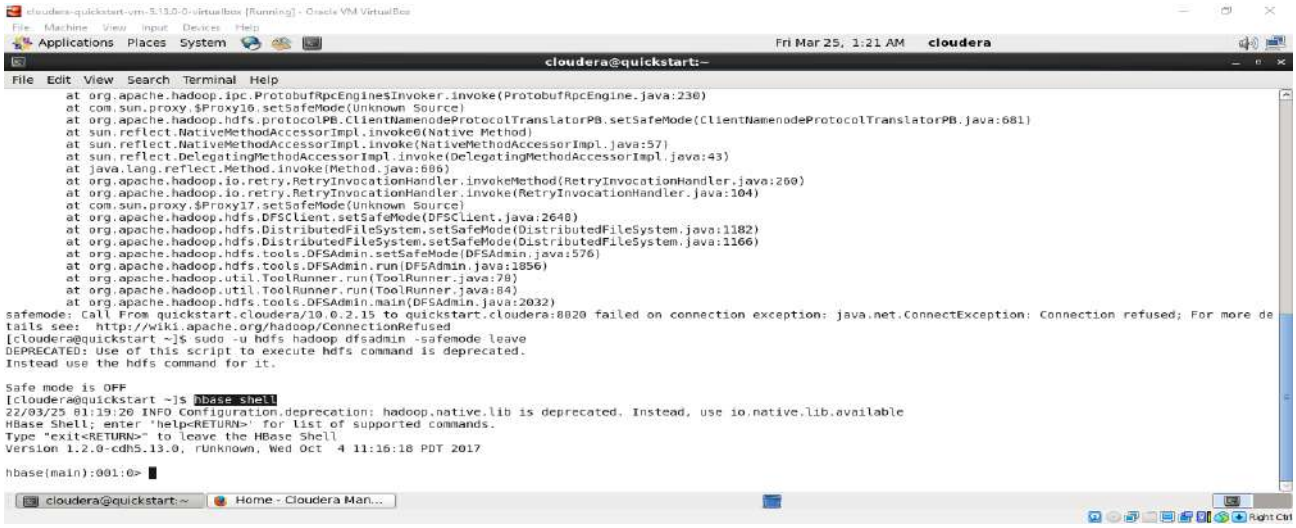




3. **sudo -u hdfs hadoop dfsadmin -safemode leave**

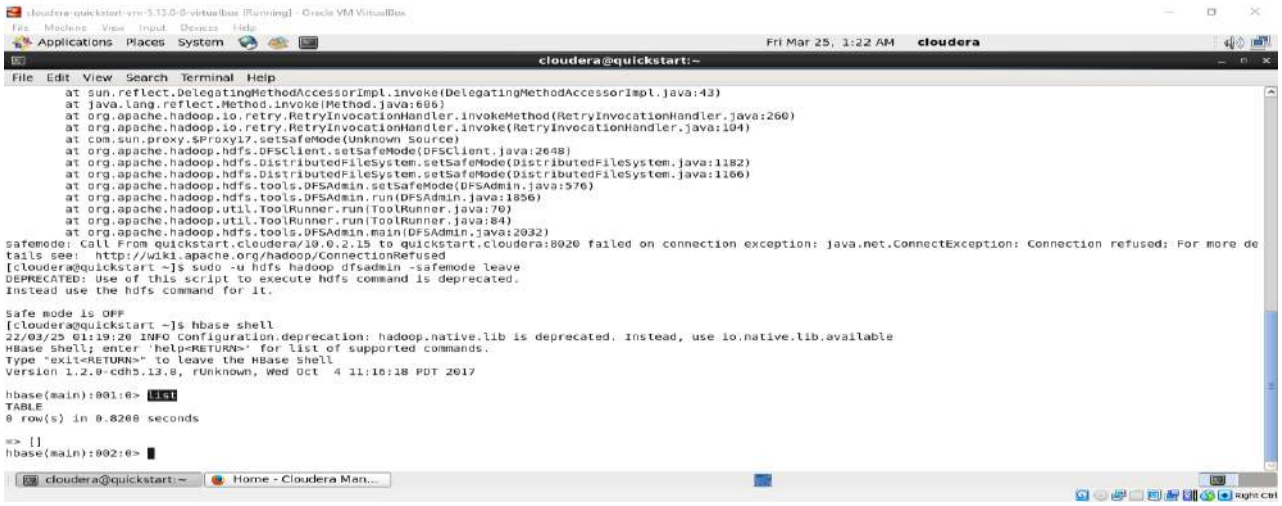
We can start the HBase interactive shell using “hbase shell” command as shown below.

hbase shell



Check the shell functioning before proceeding further. Use the list command for this purpose. List is a command used to get the list of all the tables in HBase. It lists all the tablesin HBase.

List



Restart HBase services if this is not running on terminal

$ sudo su – This commands is to become super user

$ service hbase-master restart – This commands is to restart hbase-master services

$ service hbase-regionserver restart – This commands is to restart hbase-regionserverservices

Once these commands run successfully then Open the browser and refresh the page andsee all the HBase serveries will be restarted.

Schema Design using HBase

The HBase schema design is very different compared to the relation database schema design. Below are some of general concept that should be followed while designing schemain Hbase:

Row key: Each table in HBase table is indexed on row key. Data is sorted lexicographically bythis row key. There are no secondary indices available on HBase table.

Automaticity: Avoid designing table that requires atomacity across all rows. All operationson HBase rows are atomic at row level.

Even distribution: Read and write should uniformly distributed across all nodes available in cluster. Design row key in such a way that, related entities should be stored in adjacent rowsto increase read efficacy.

HBase Schema Row key, Column family, Column qualifier, individual and Row value SizeLimit

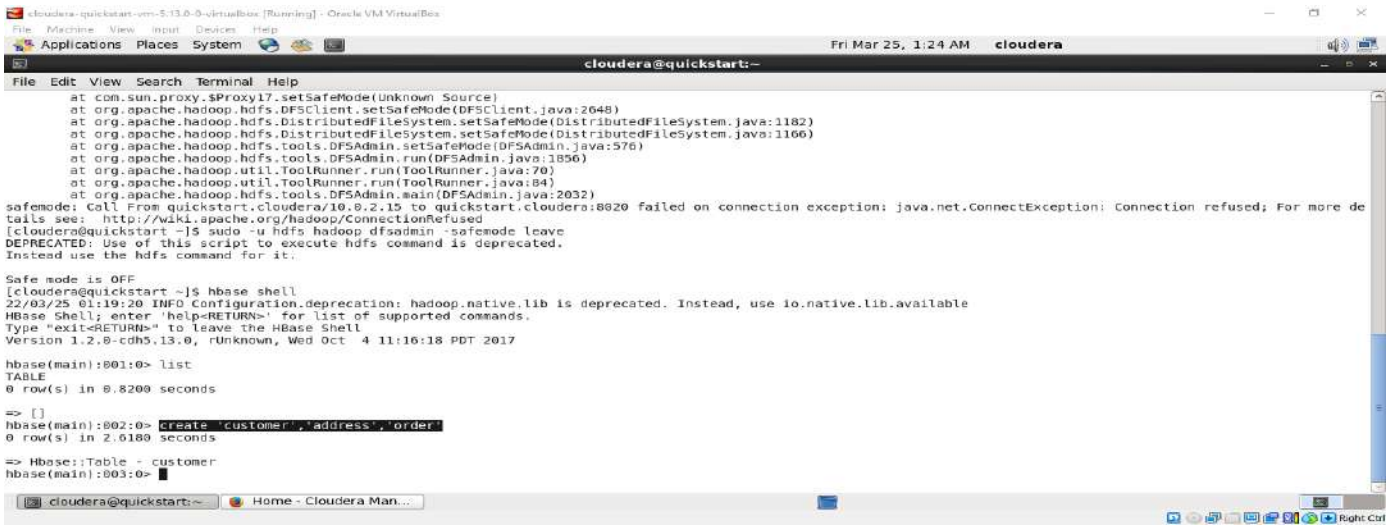
1. Creating a Table using HBase Shell

We can create a table using the create command, here you must specify the table name andthe Column Family name. The syntax to create a table in HBase shell is shown below.

create '<table name>','<column family>'

create 'customer','address','order'

customer -> table ; address & order -> column_name



list

2. Put: Inserts a new record into the table with row identified by 'row..'

This command is used for following things

- It will put a cell 'value' at defined or specified table or row or column.
- It will optionally coordinate time stamp.

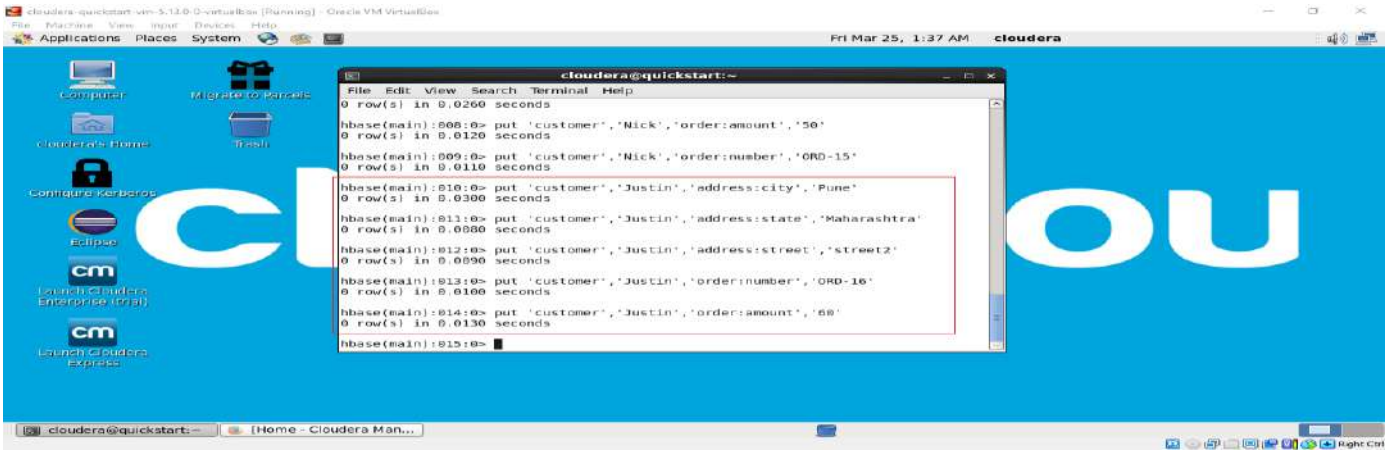
Syntax: **put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>**

Example – with the help of put commands we have inserted new records in “customer” table for address and order family. Here name “**Nick**” is Row key

put 'customer','Nick','address:city','Mumbai'
put 'customer','Nick','address:state','Maharashtra'
put 'customer','Nick','address:street','Steet1'
put 'customer','Nick','order:number','ORD-15'
put 'customer','Nick','order:amount','50'
put 'customer','Nick','address:state','Maharashtra'

Adding **one more record** of customer name “**Justin**”. Here name “Justin” is Row key

put 'customer','Justin','address:city','Pune'
put 'customer','Justin,'address:state','Maharashtra'
put 'customer','Justin','order:number','ORD-16'
put 'customer','Justin','order:amount','60'

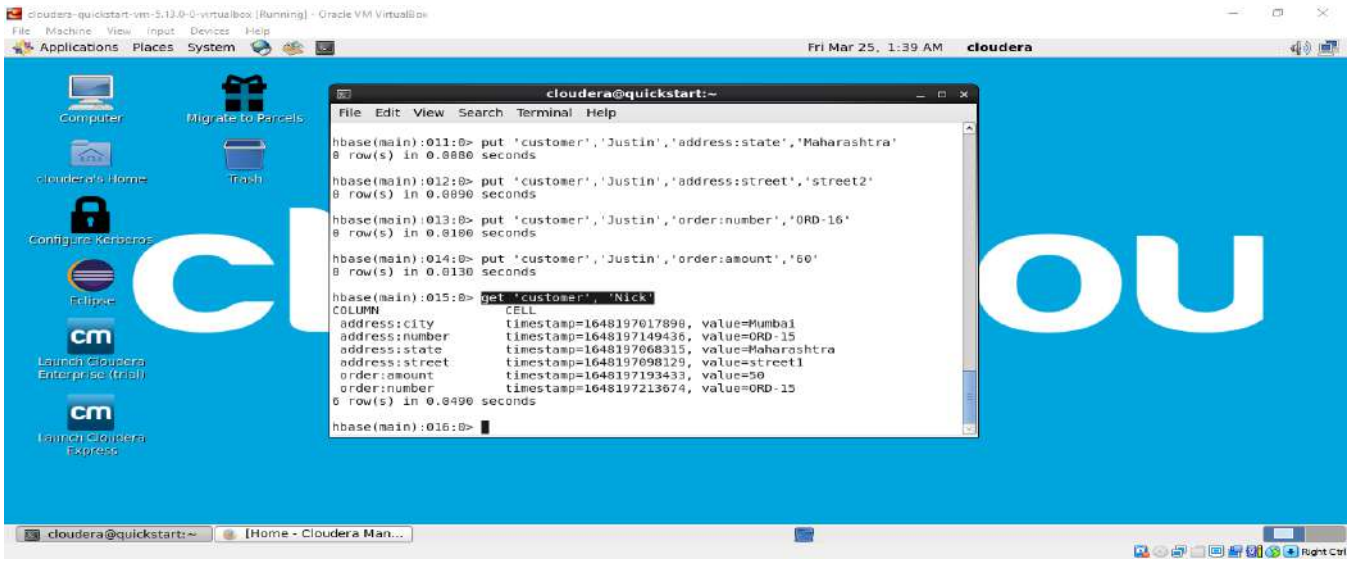


3. Get: Returns the records matching the row identifier provided in the table

By using this command, you will get a row or cell contents present in the table. In additionto that you can also add additional parameters to it like TIMESTAMP, TIMERANGE,VERSIONS, FILTERS, etc. to get a particular row or cell content.

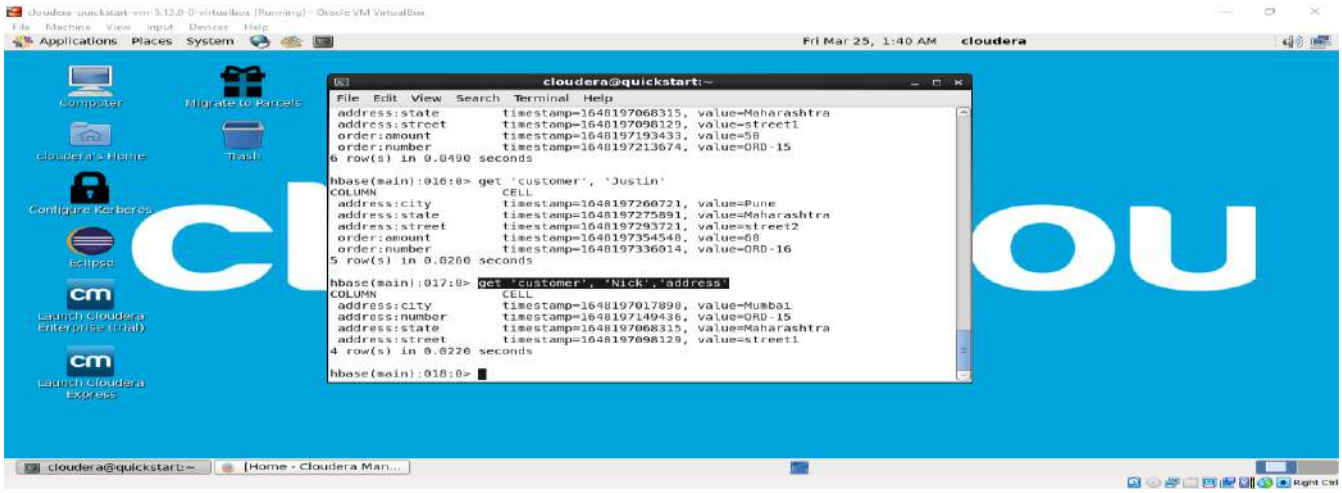
Syntax: **get <'tablename'>, <'rowname'>, {< Additional parameters>}**

a) get 'customer', 'Nick'



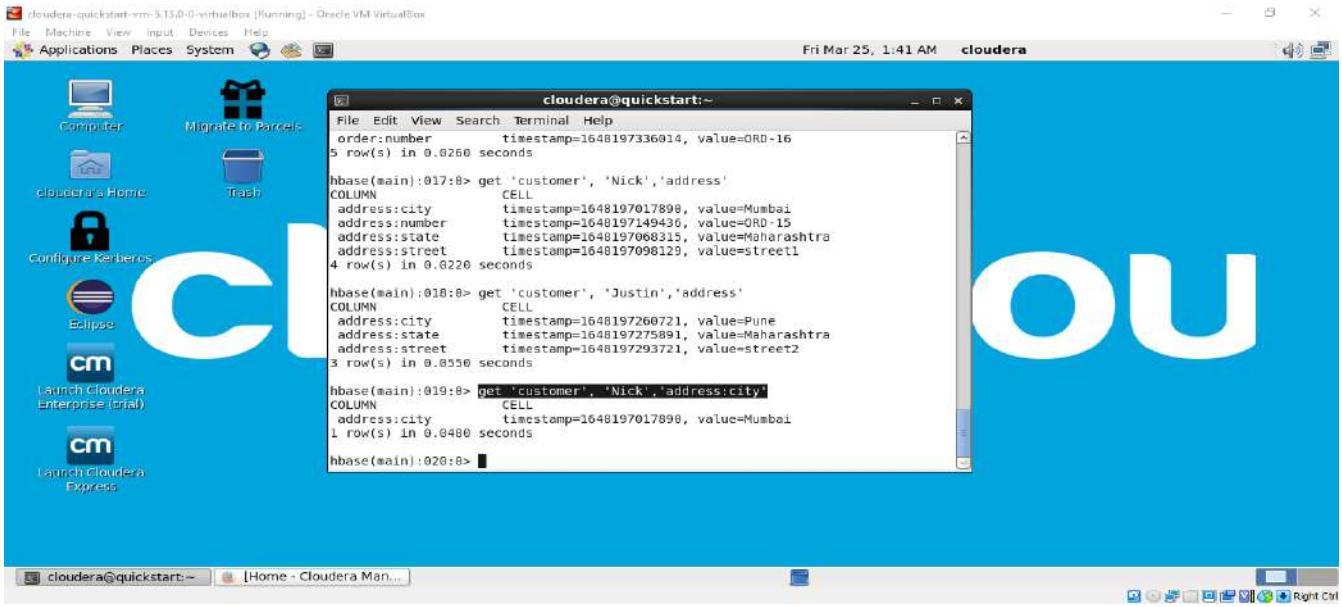
b) Additional parameters to get only address details

get 'customer', 'Nick','address'



c) Additional parameters to get only city details

get 'customer', 'Nick','address:city'



4. Scan: The scan command is used to view the data in HTable.Using the scan command, you can get the table data.

- This command scans entire table and displays the table contents.
- We can pass several optional specifications to this scan command to get moreinformation about the tables present in the system.
- Scanner specifications may include one or more of the following attributes.
  - These are TIMERANGE, FILTER, TIMESTAMP, LIMIT, MAXLENGTH,COLUMNS, CACHE, STARTROW and STOPROW.

Its syntax is as follows:

Syntax: **scan <'tablename'>, {Optional parameters}**

scan 'customer'

When we execute above commands in HBase then we will be getting all the table "customer" contents along with additional parameters like timestamp as show in belowscreenshot.

5. Delete -Using the delete command, you can delete a specific cell in a table.

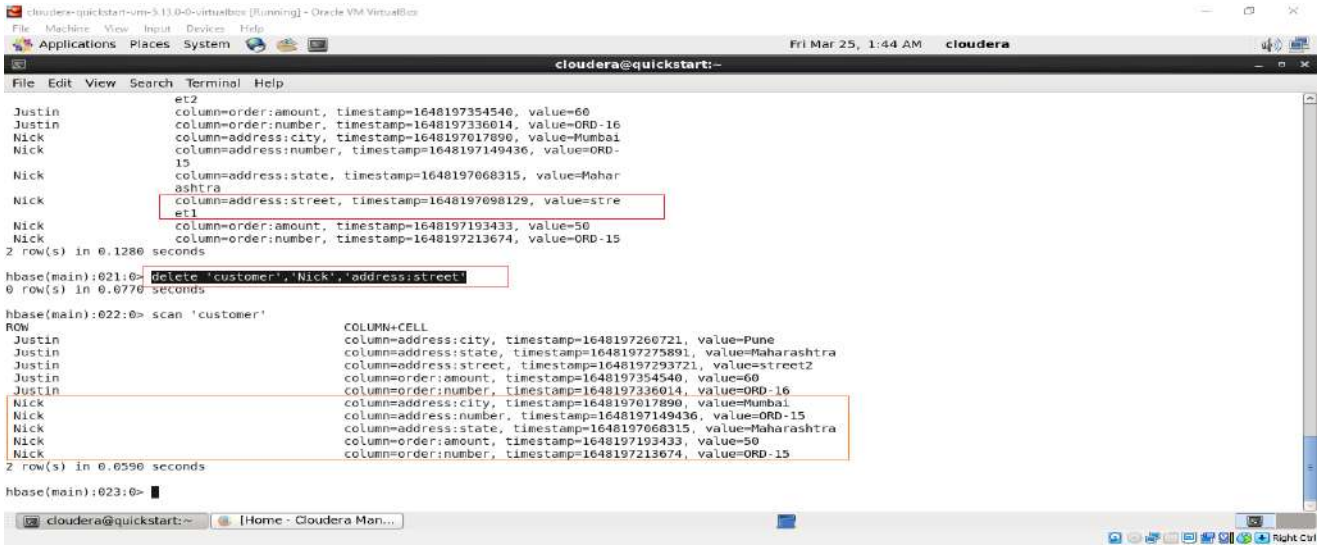
- This command will delete cell value at defined table of row or column.
- Delete must and should match the deleted cells coordinates exactly.
- When scanning, delete cell suppresses older versions of values.The syntax of delete command is as follows:

Syntax:**delete <'tablename'>,<'row name'>,<'column name'>**

delete 'customer','Nick','address:street'

The above command below delete street from address family for row key "Nick" from "customer" table.

Use scan command to see if street is deleted for customer "Nick". As we can see "street"information is deleted from customer "Nick" from below screenshot.



6. Alter - This command alters the column family schema. To understandwhat exactly it does, we have explained it here with an example.

Alter commands are useful for below cases -

a. Altering single, multiple column family names

b. Deleting column family names from table

c. Several other operations using scope attributes with table

Syntax: **alter <tablename>, NAME=><column familyname>, VERSIONS=>5**

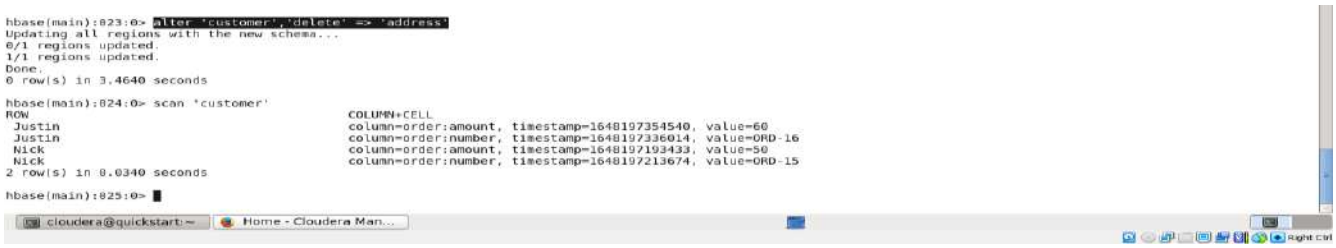
We can delete specific column family by using alter commands

alter 'customer','delete' => 'address'

After deleting “address” family from “customer” table. Let’s again check customer tableusing “scan” commands as follow

scan ‘customer’

As you can see from below screenshot we only now have order:amount and order:number.



7. Describe - This command describes the named table.

a. It will give more information about column families present in the mentioned table
b. In our case, it gives the description about table "customer."
c. It will give information about table name with column families, associated filters,versions and some more details.

Syntax:**describe <table name>**

desc 'customer'

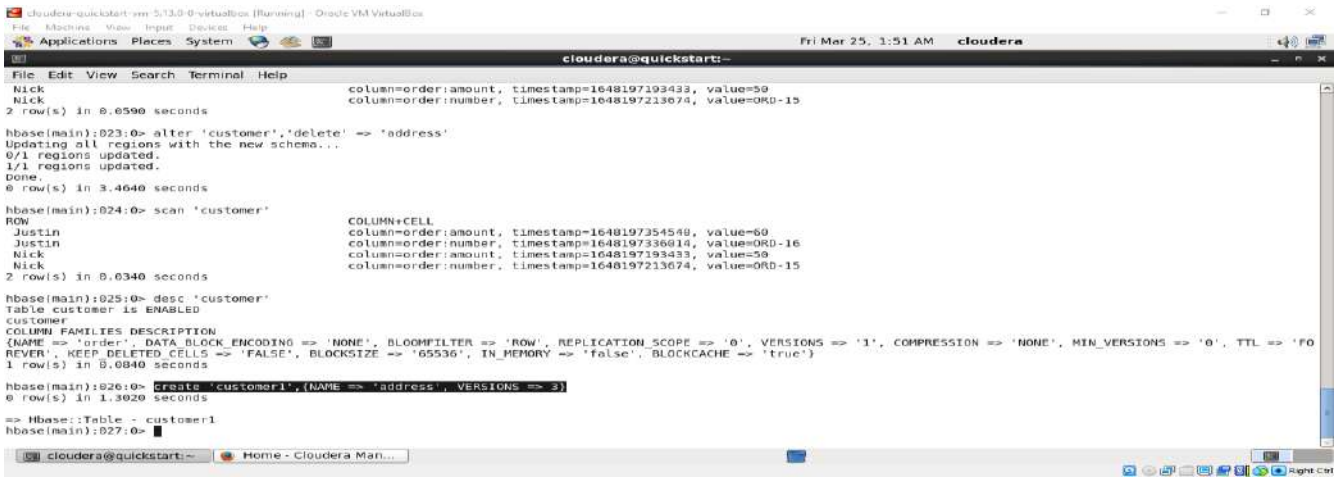
8. Versions –

A {row, column, version} tuple exactly specifies a cell in HBase. In the Apache HBase you can have many cells where row and columns are same but differs only in version values. A version is a timestamp values is written alongside each value. By default, the timestamp values represent the time on the RegionServer when the data was written, but you can change the default HBase setting and specify a different timestamp value when you put datainto the cell.

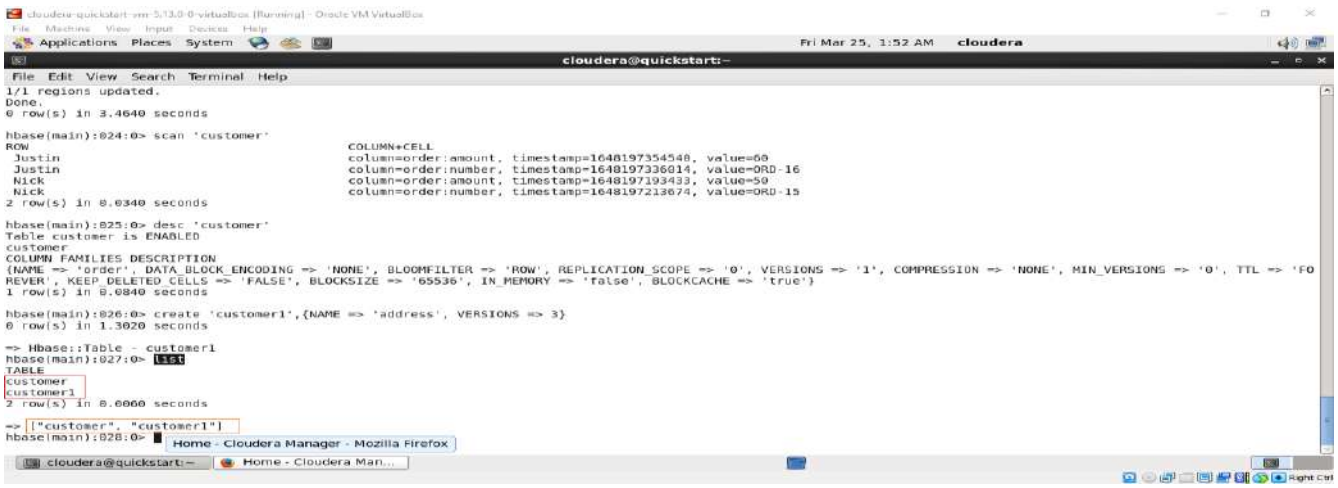
In HBase, rows and column keys are expressed as bytes, the version is specified using a longinteger. The HBase version dimension is stored in decreasing order, so that when reading from a store file, the most recent values are found first.

create 'customer1',{NAME => 'address', VERSIONS => 3}

With the help of above commands we are creating 3 versions

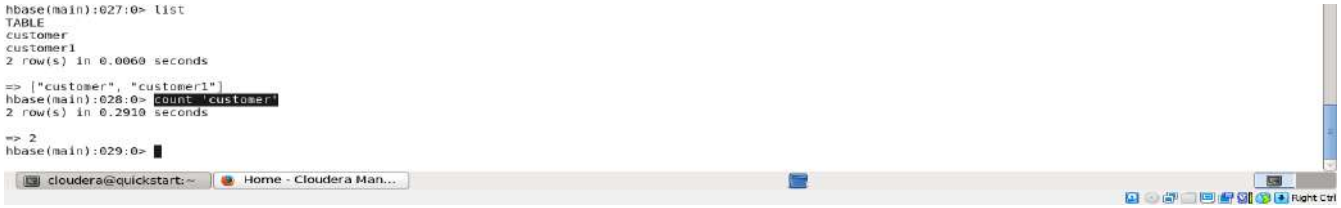


Verifying if “customer1” is created after executing above commands with the help of listcommands as shown in screenshot below.



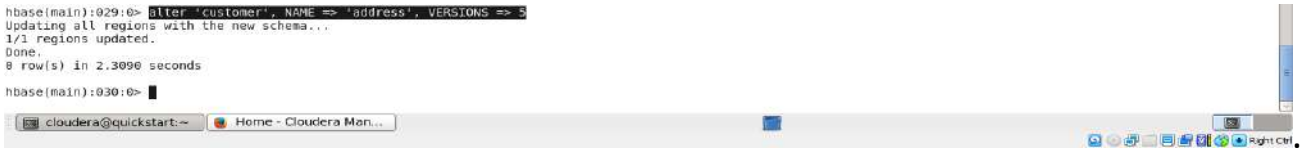
9. Count - You can count the number of rows of a table using the countcommand. Its syntax is as follows:

count 'customer'



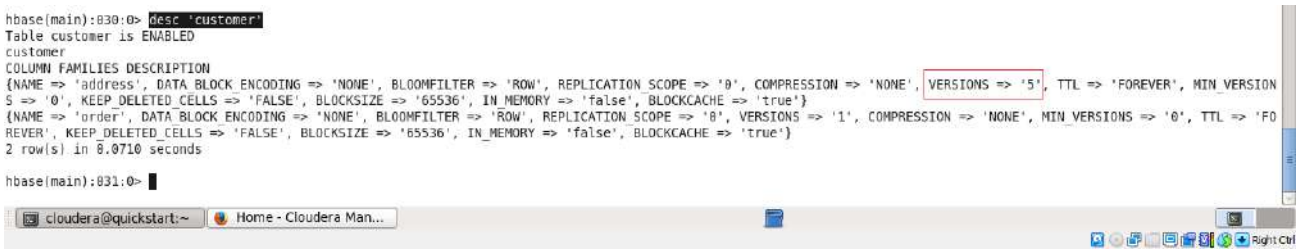
10. Alter -Update the version number for already existing columns family

alter 'customer', NAME => 'address', VERSIONS => 5

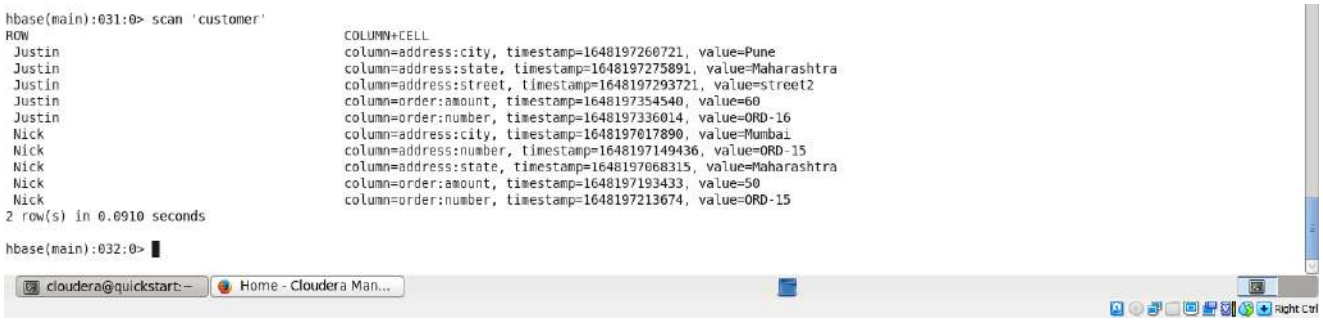


Again using describe command to see version is updated

desc 'customer'



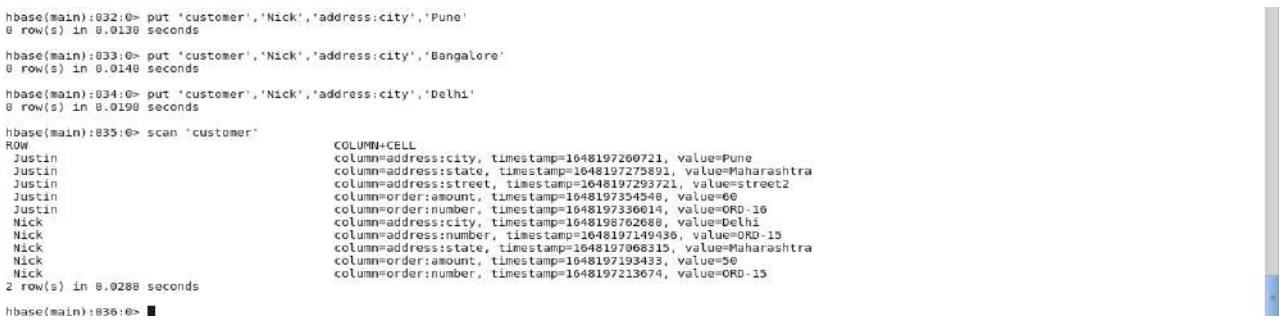
scan 'customer'



put 'customer', 'Nick' , 'address:city', 'Pune'

put 'customer', 'Nick' , 'address:city', 'Bangalore'

put 'customer', 'Nick' , 'address:city', 'Delhi'



Below are example of adding version of address:city

Currently below is the data in 'customer'. We will be adding more information in it one byone.

scan 'customer', {COLUMN => 'address:city', VERSIONS => 2}

```
hbase(main):036:0> scan 'customer', {COLUMN => 'address:city', VERSIONS => 2}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Nick                                 column=address:city, timestamp=1648198762680, value=Delhi
 Nick                                 column=address:city, timestamp=1648198755614, value=Bangalore
2 row(s) in 0.0260 seconds

hbase(main):037:0>
```

scan 'customer', {COLUMN=> 'address:city', VERSIONS => 3}

scan 'customer', {COLUMN=> 'address:city', VERSIONS => 4}

After executing above commands its giving us 3 and 4 version of address:city records asshown in below screenshot.

```
hbase(main):038:0> scan 'customer', {COLUMN => 'address:city', VERSIONS => 1}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Nick                                 column=address:city, timestamp=1648198762680, value=Delhi
2 row(s) in 0.0220 seconds

hbase(main):039:0> scan 'customer', {COLUMN => 'address:city', VERSIONS => 0}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Nick                                 column=address:city, timestamp=1648198762680, value=Delhi
2 row(s) in 0.0290 seconds

hbase(main):040:0> scan 'customer', {COLUMN => 'address:city', VERSIONS => 4}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Nick                                 column=address:city, timestamp=1648198762680, value=Delhi
 Nick                                 column=address:city, timestamp=1648198755614, value=Bangalore
 Nick                                 column=address:city, timestamp=1648198734145, value=Pune
 Nick                                 column=address:city, timestamp=1648197017890, value=Mumbai
2 row(s) in 0.0540 seconds

hbase(main):041:0> scan 'customer', {COLUMN => 'address:city', VERSIONS => 5}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Nick                                 column=address:city, timestamp=1648198762680, value=Delhi
 Nick                                 column=address:city, timestamp=1648198755614, value=Bangalore
 Nick                                 column=address:city, timestamp=1648198734145, value=Pune
 Nick                                 column=address:city, timestamp=1648197017890, value=Mumbai
```

scan 'customer', {VERSIONS => 5}

```
hbase(main):042:0> scan 'customer', {VERSIONS => 5}
ROW                                   COLUMN+CELL
 Justin                               column=address:city, timestamp=1648197260721, value=Pune
 Justin                               column=address:state, timestamp=1648197275891, value=Maharashtra
 Justin                               column=address:street, timestamp=1648197293721, value=street2
 Justin                               column=order:amount, timestamp=1648197354548, value=60
 Justin                               column=order:number, timestamp=1648197336014, value=ORD-16
 Nick                                 column=address:city, timestamp=1648198762688, value=Delhi
 Nick                                 column=address:city, timestamp=1648198755614, value=Bangalore
 Nick                                 column=address:city, timestamp=1648198734145, value=Pune
 Nick                                 column=address:city, timestamp=1648197017898, value=Mumbai
 Nick                                 column=address:number, timestamp=1648197149436, value=ORD-15
 Nick                                 column=address:state, timestamp=1648197068315, value=Maharashtra
 Nick                                 column=order:amount, timestamp=1648197193433, value=50
 Nick                                 column=order:number, timestamp=1648197213674, value=ORD-15
2 row(s) in 0.0760 seconds

hbase(main):043:0>
```

11. Disable -This command will start disabling the named table

If table needs to be deleted or dropped, it has to disable first

Syntax: **disable <table_name>**

disable 'customer'

```
=> ["customer", "customer1"]
hbase(main):047:0> disable 'customer'
0 row(s) in 0.0720 seconds
```

12. **Drop– It drops a table from HBase. Drop means complete deletion oftable. For this first disable the table then drop it.**

    a. To delete the table present in HBase, first we have to disable it

    b. To drop the table present in HBase, first we have to disable it

    c. So either table to drop or delete first the table should be disable using disablecommand

Syntax:**drop <table_name>**

drop 'customer'