# INDEX

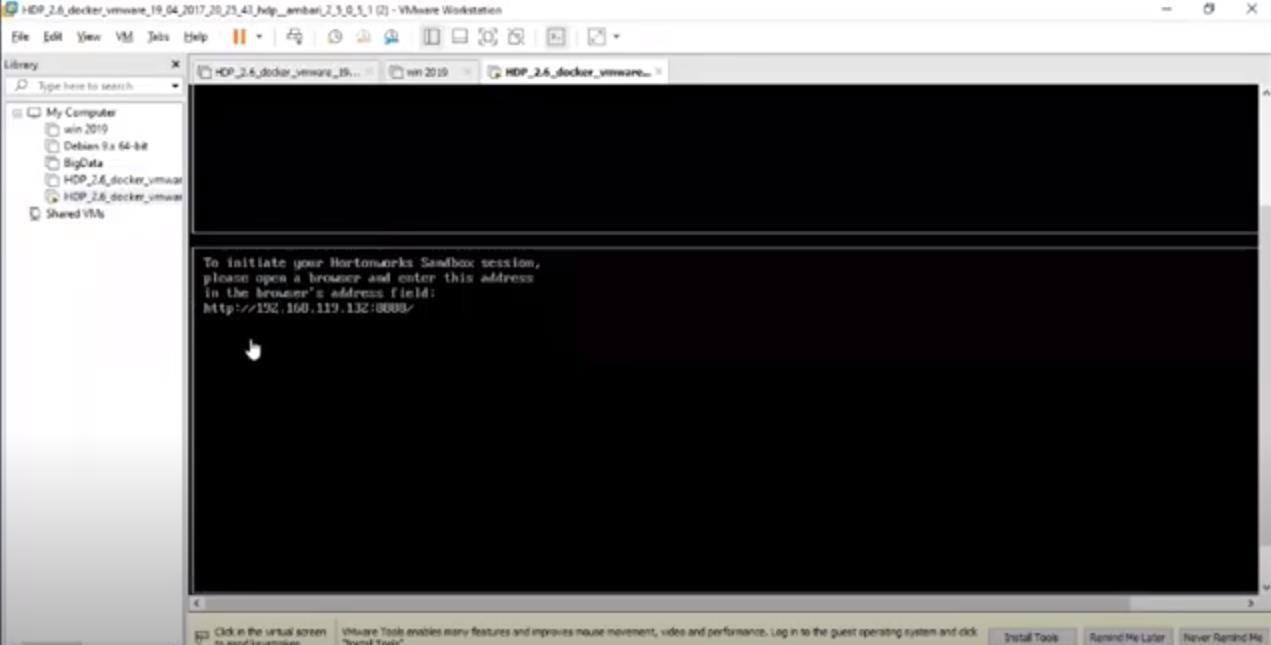
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Practical** | **Date** | **Sign** |
| **1** | **Install, configure and run Hadoop and**  **HDFS ad explore HDFS.** |  |  |
| **2** | **Implement word count / frequency**  **programs using MapReduce** |  |  |
| **3** | **Implement an MapReduce program**  **that processes a weather dataset.** |  |  |
| **4** | **Implement the program using Pig.** |  |  |
| **5** | **Implement the application in Hive.** |  |  |
| **6** | **Implement an application that stores**  **big data in Hbase/ Python** |  |  |
| **7** | **Implement Decision tree classification techniques** |  |  |
| **8** | **Implement SVM classification techniques** |  |  |

##### Practical 1

Install, configure and run Hadoop and HDFS ad explore HDFS.

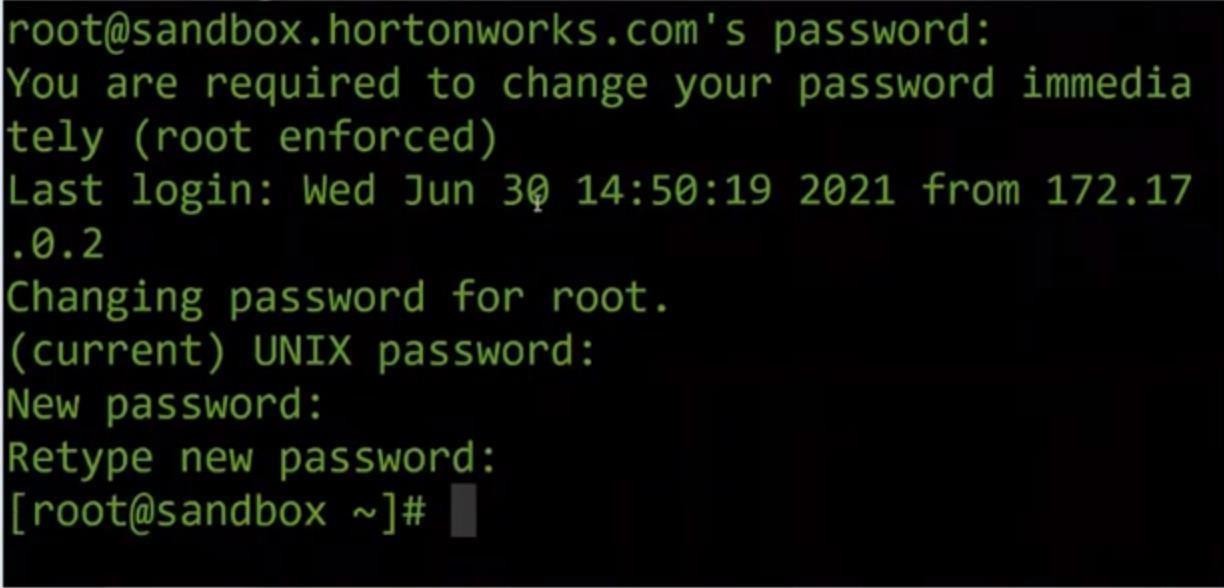
Download Virtual machine setup i.e. VMware setup (in which Hadoop is configured.

Step 1: Load the server on VM ware workstation



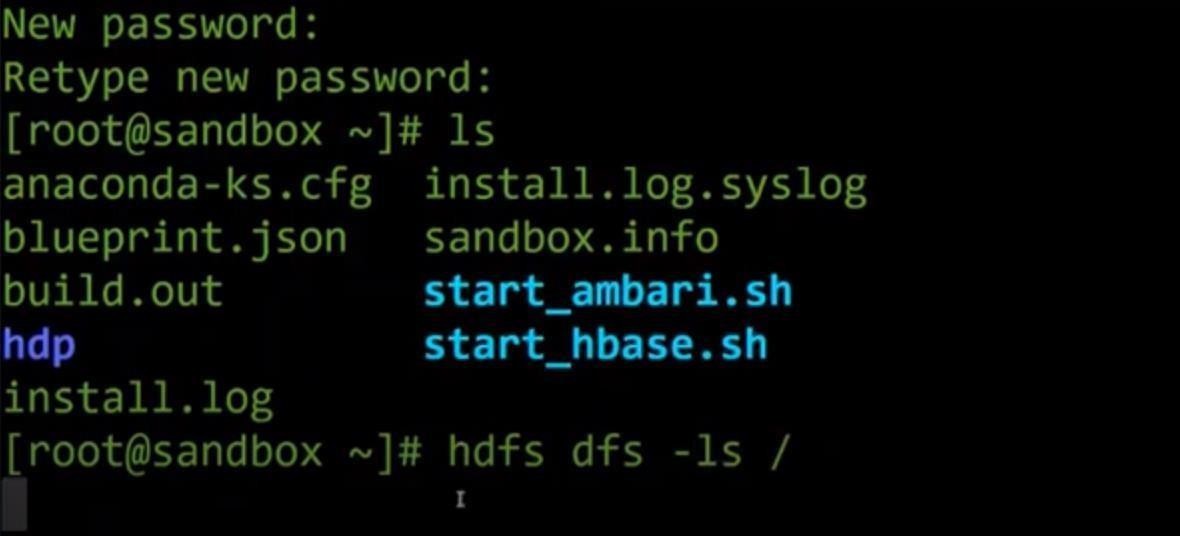
Step 2: To enable admin login open shell and reset root login. Open Terminal 192.168.119.132:4200

In Sandbox login enter root And Password is Hadoop And reset the password

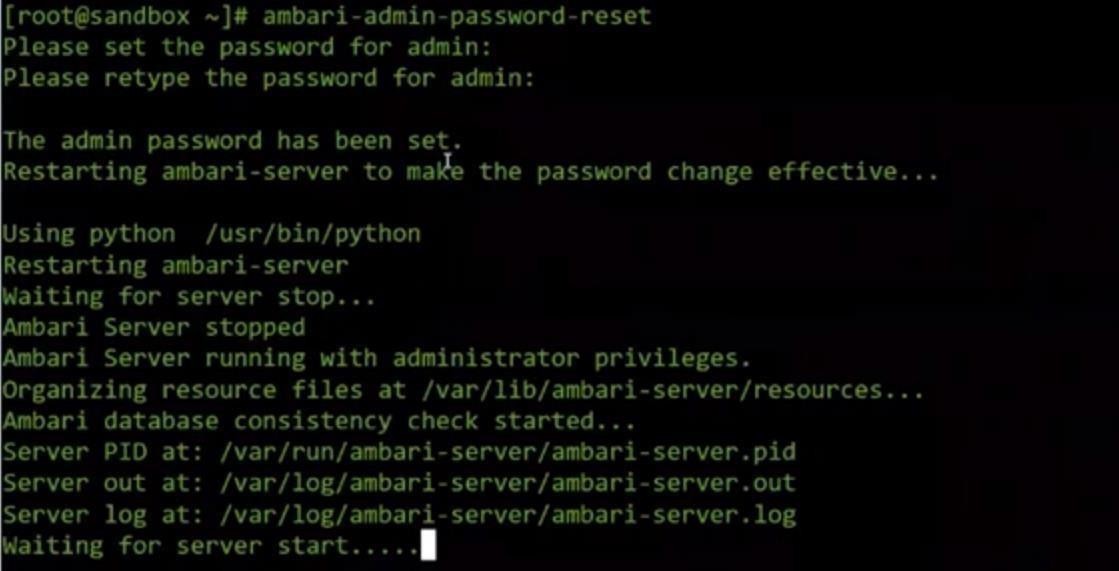


Windows linux system and Hadoop system are different When we type **ls** command it is executed in local system

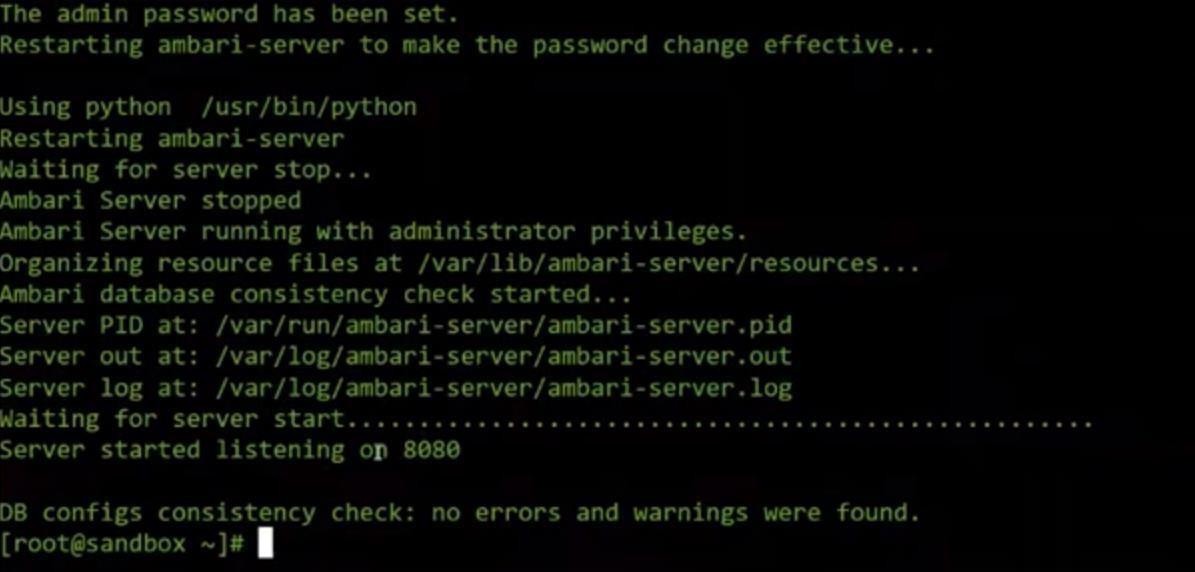
When we type **hdfs dfs -ls** it will execute in Hadoop system directory



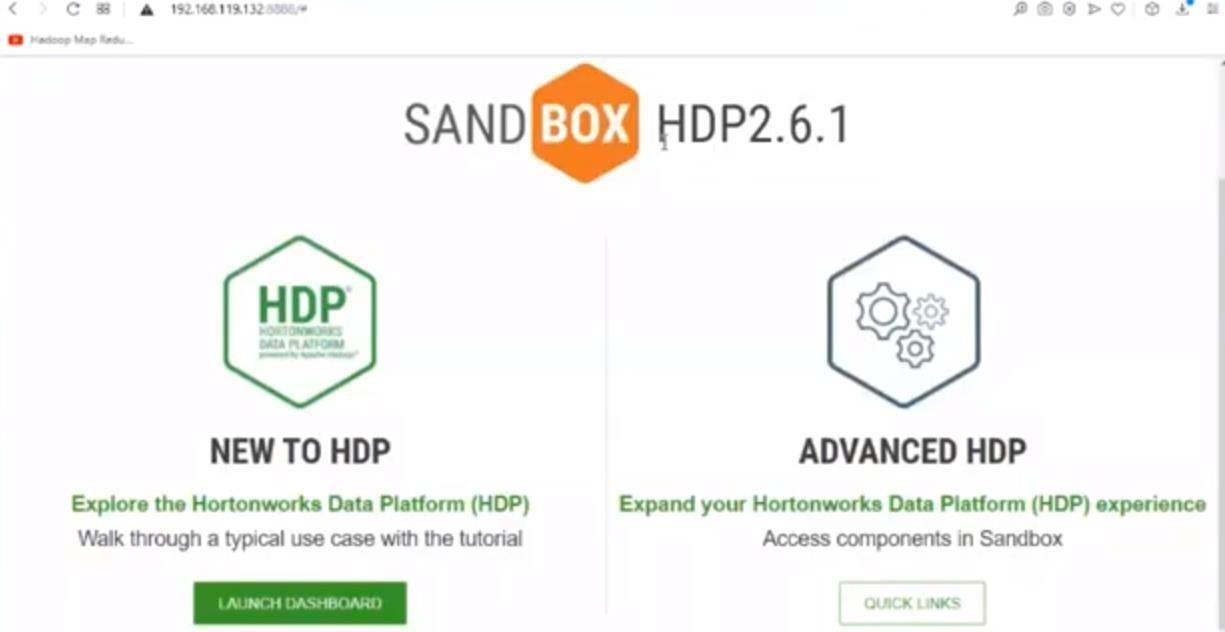
Step 3: Reset Admin account Password



Server listening on 8080 and shell login is complete.

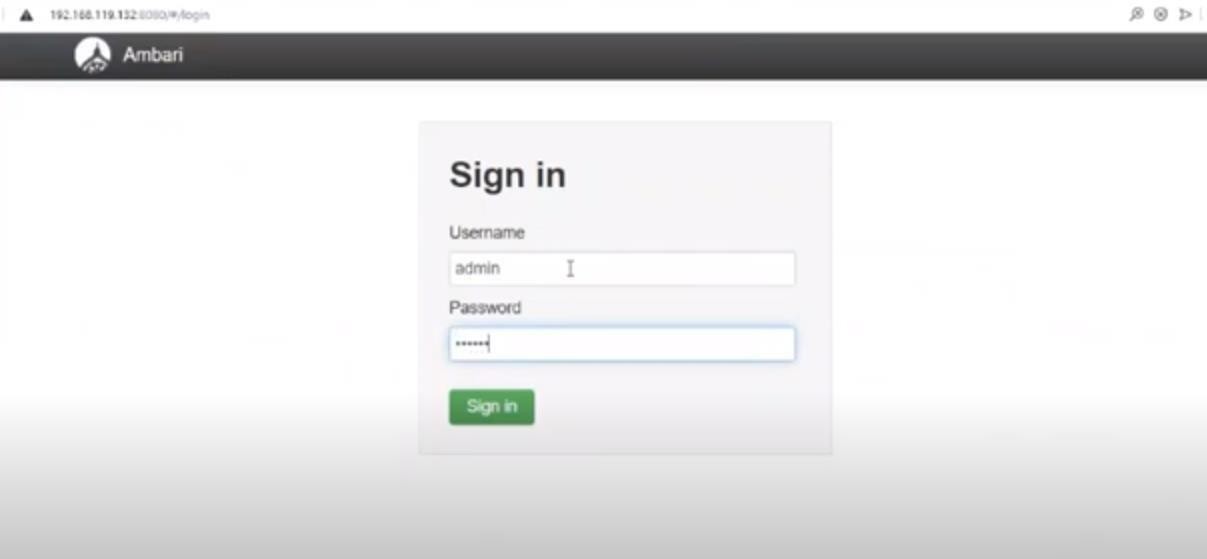


To use graphical user interface login to 192.168.119.132:4200

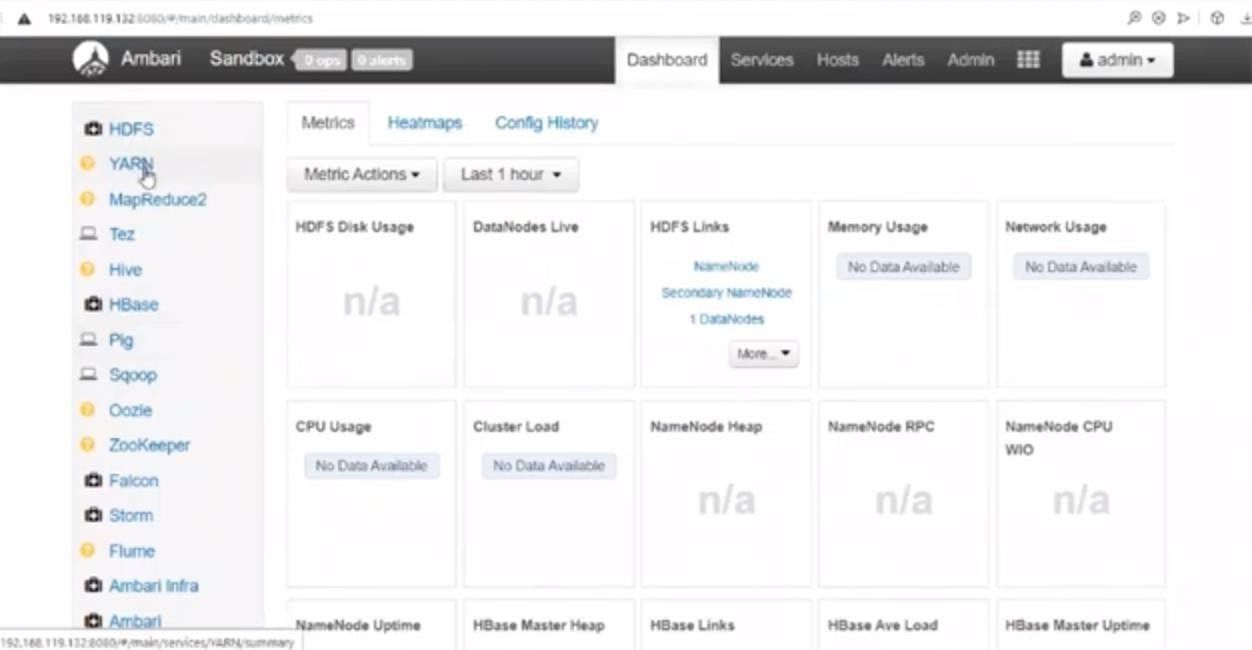


Click on Launch Dashboard

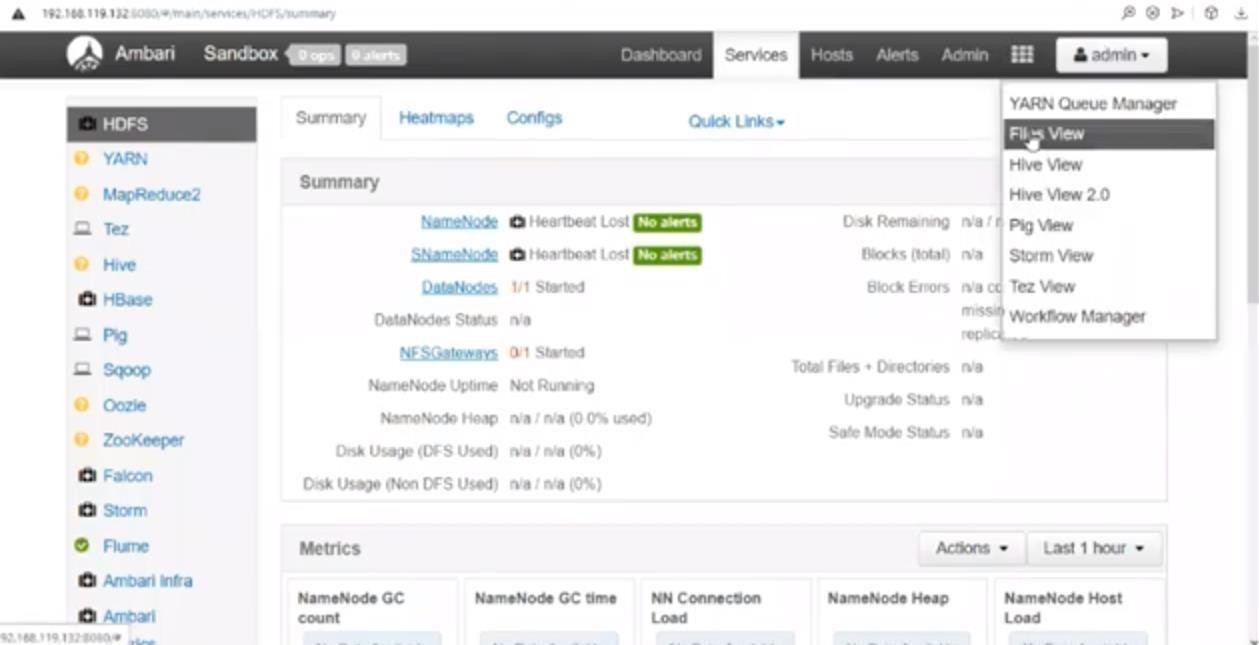
Enter the username and password for admin login.



Below is the Hadoop server.



To view file in HDFS click on HDFS and click on File view.



Commands:

1. To view root folder file from terminal use command hdfs dfs -ls /user and press enter. It will display all the files in the root user that we see in UI(Screenshot 2).

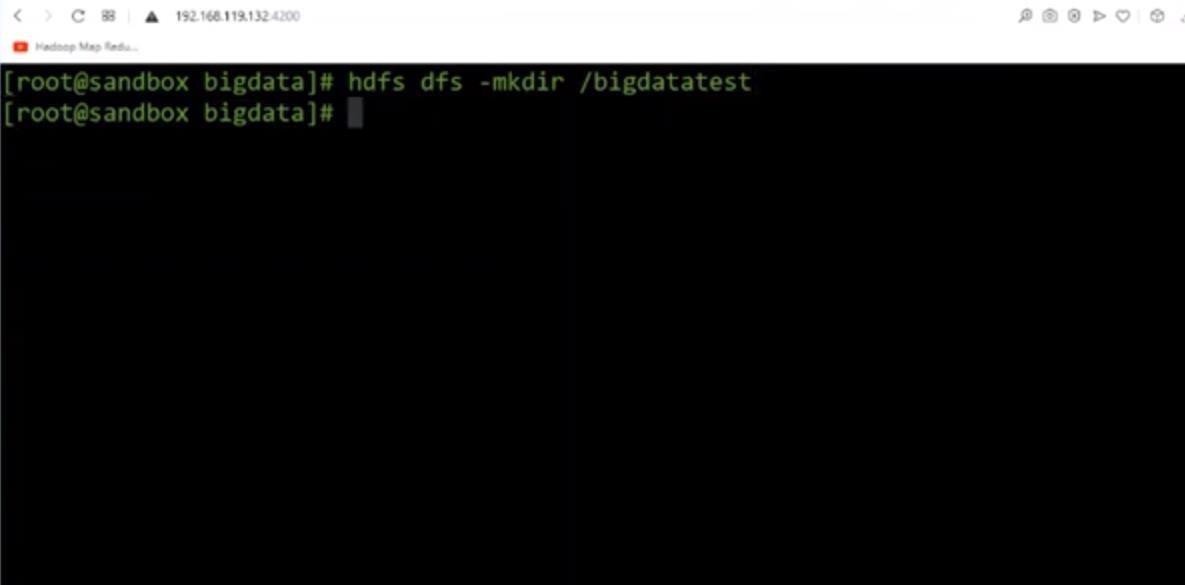
**ls:** This command is used to list all the files

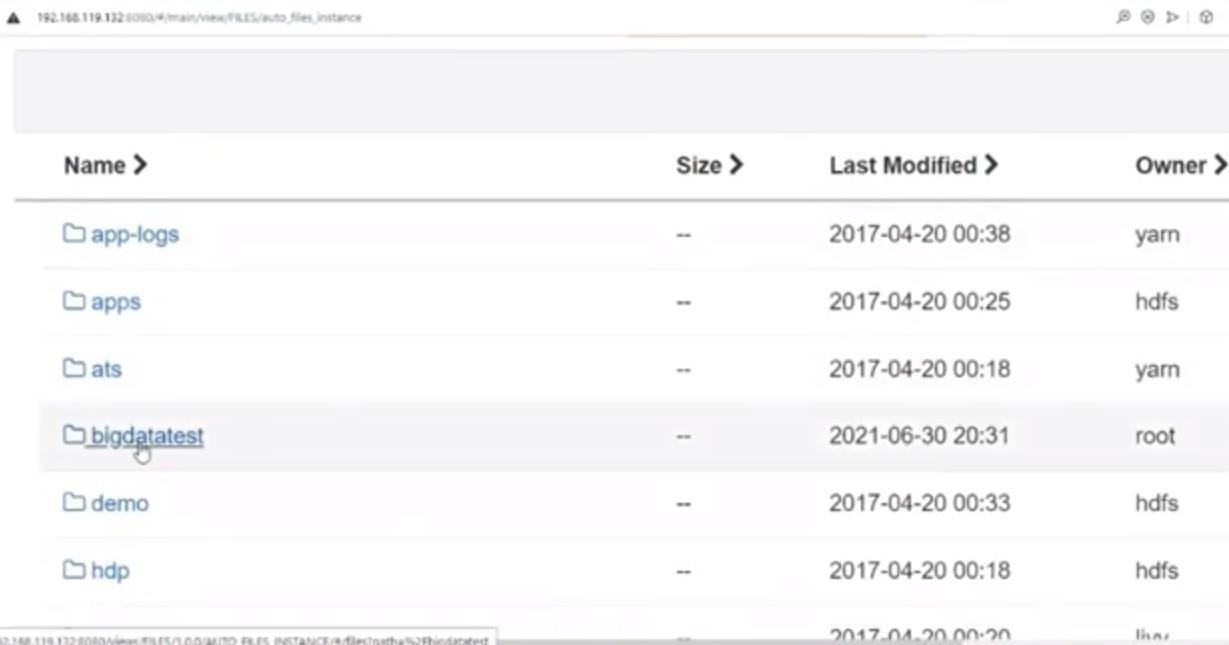




1. **mkdir**: To create a directory.

Create a folder in Hadoop directory. Type command hdfs dfs -mkdir /bigdatatest and enter. After it execute the command, we will see whether it is created folder in UI.

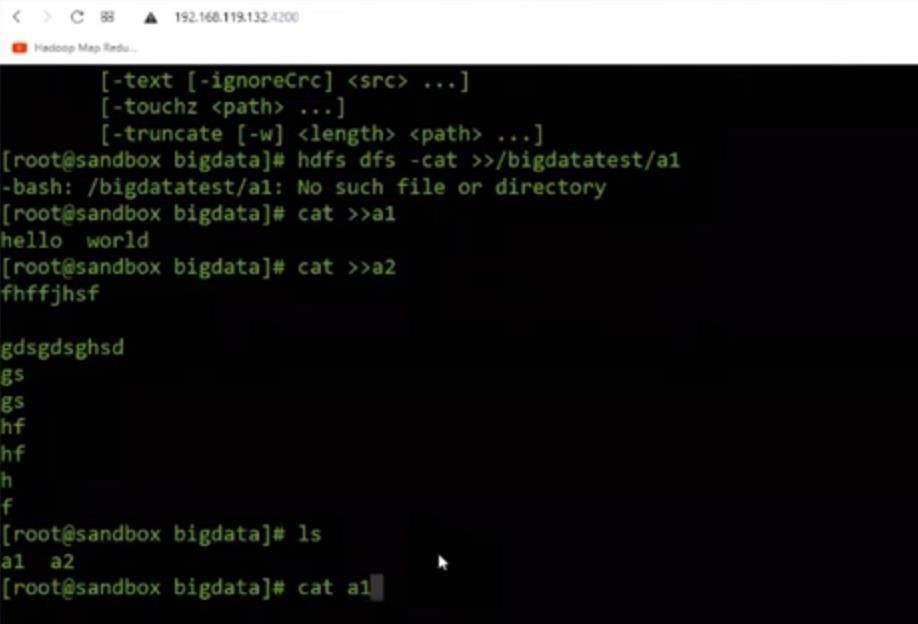




1. Create a file in local directory

**Cat:** Createa file. Cat>>

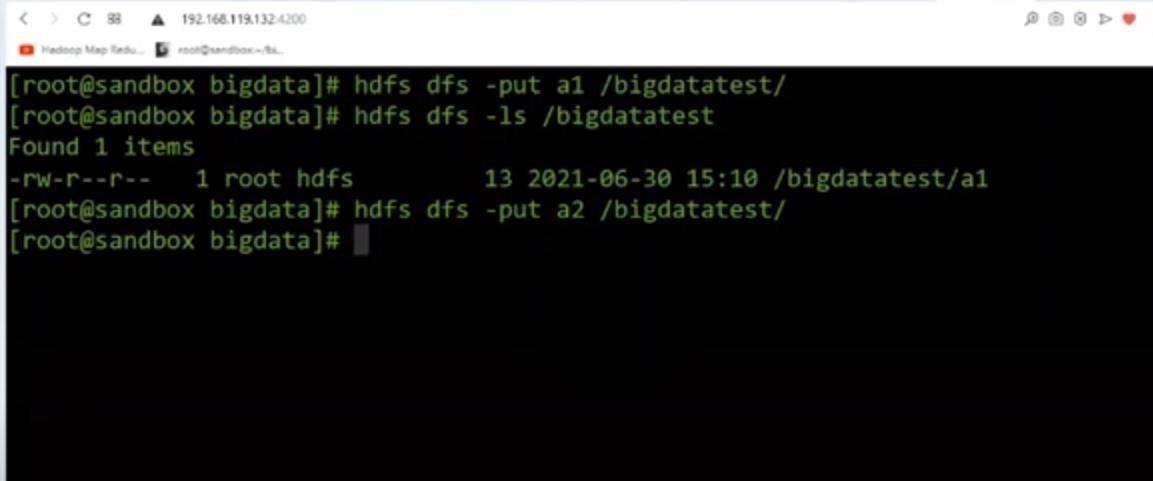
To terminate press ctrl+d



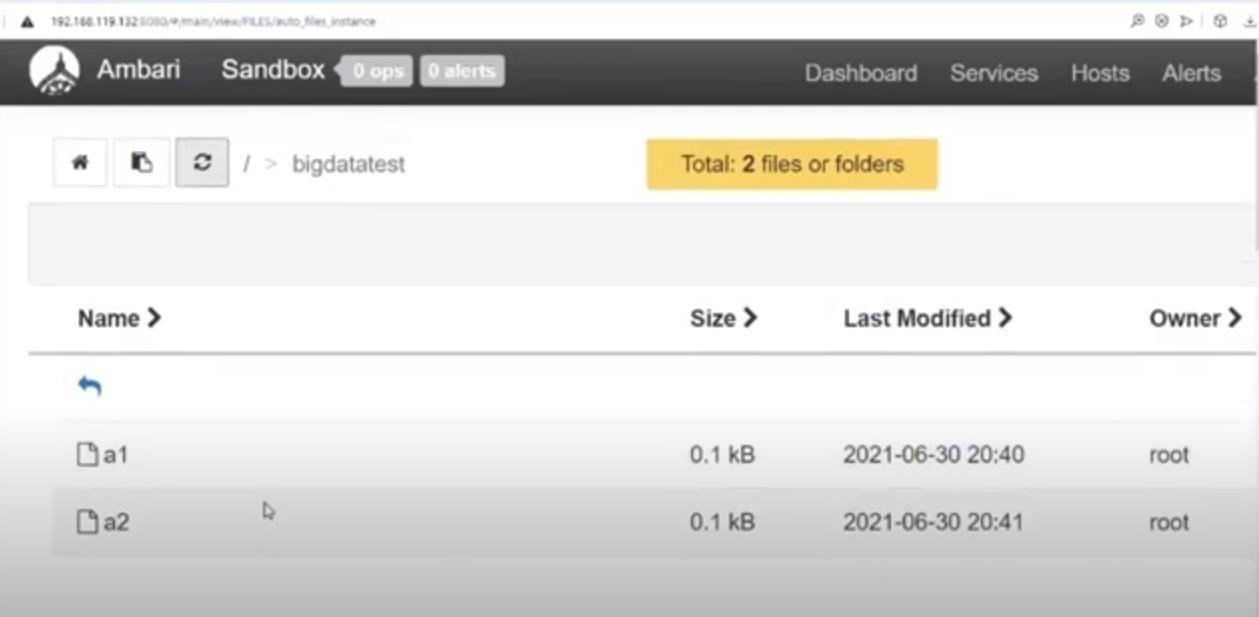
1. To upload files/directory to from local to HDFS

**Put:** to move a local file or directories into the distributed file system

Command: hdfs dfs -put a1 /bigdatatest/ and hdfs dfs -put a2 /bigdatatest/ will upload both the files.



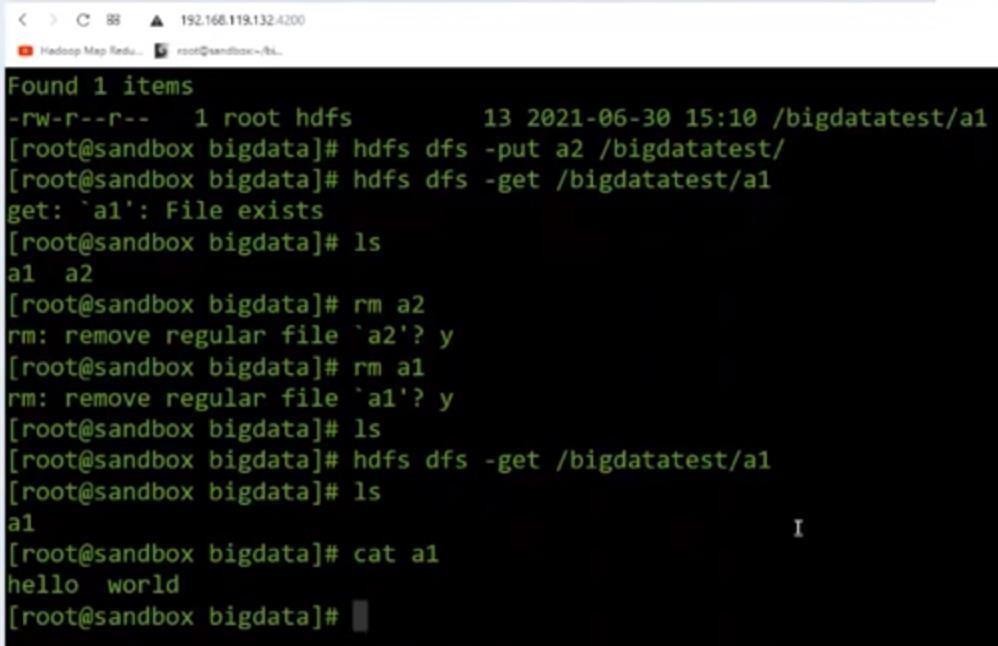
Refresh the user interface we can see both the files.



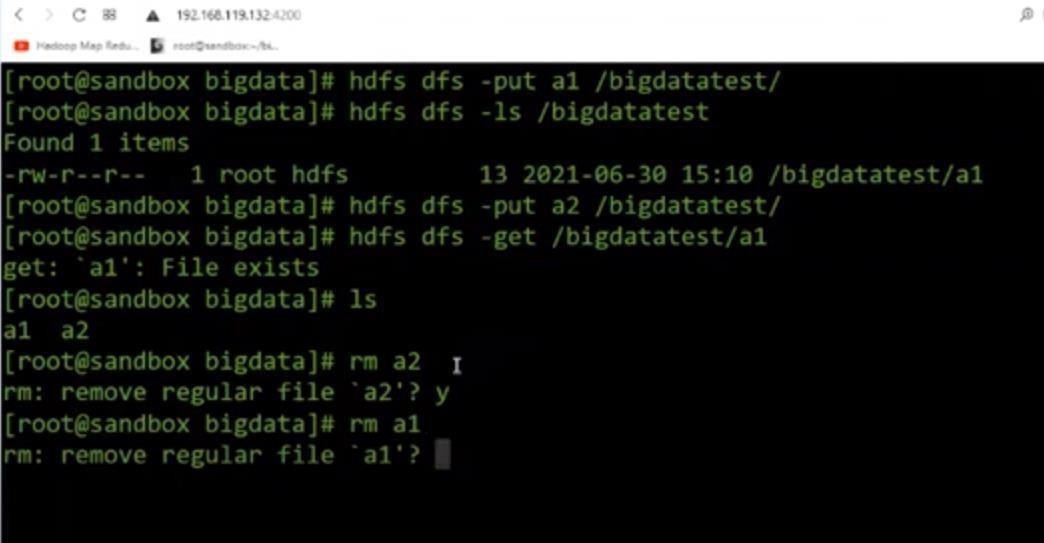
1. To download files/directories to from hdfs to local

**Get:** To copy files/folders from hdfs store to local file system.

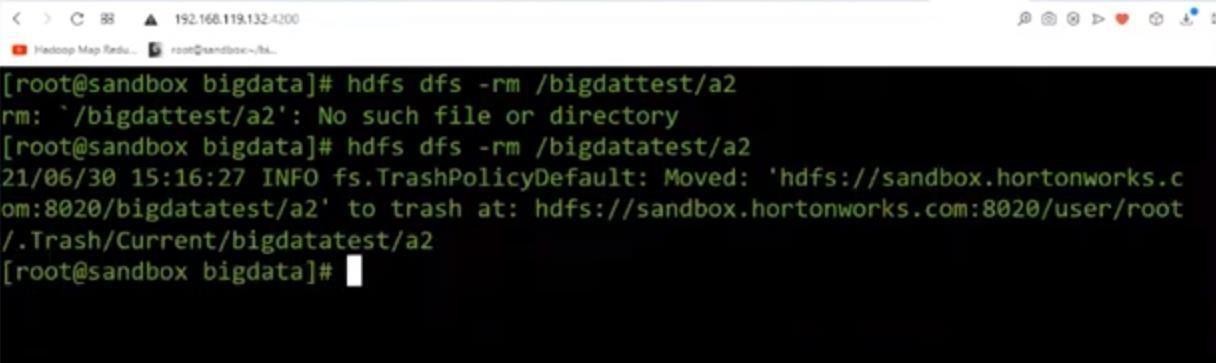
Command: hdfs dfs -get /bigdatatest/a1 and hdfs dfs -get /bigdatatest/ a2 will upload both the files.



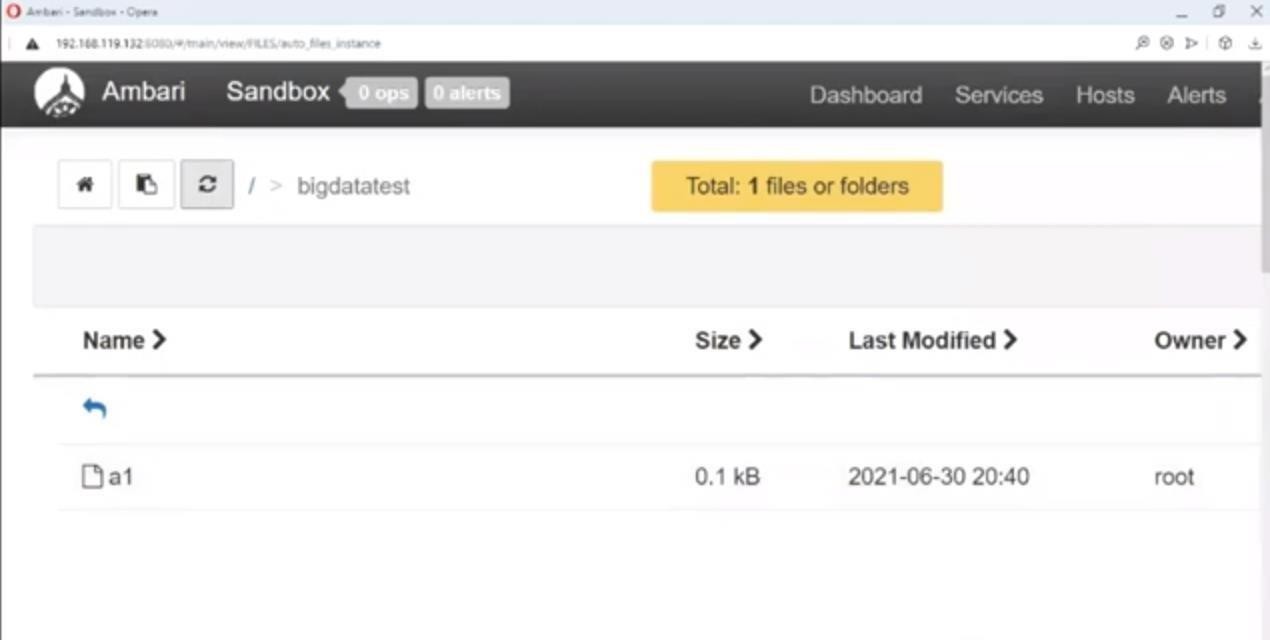
1. To remove file from local use rm command



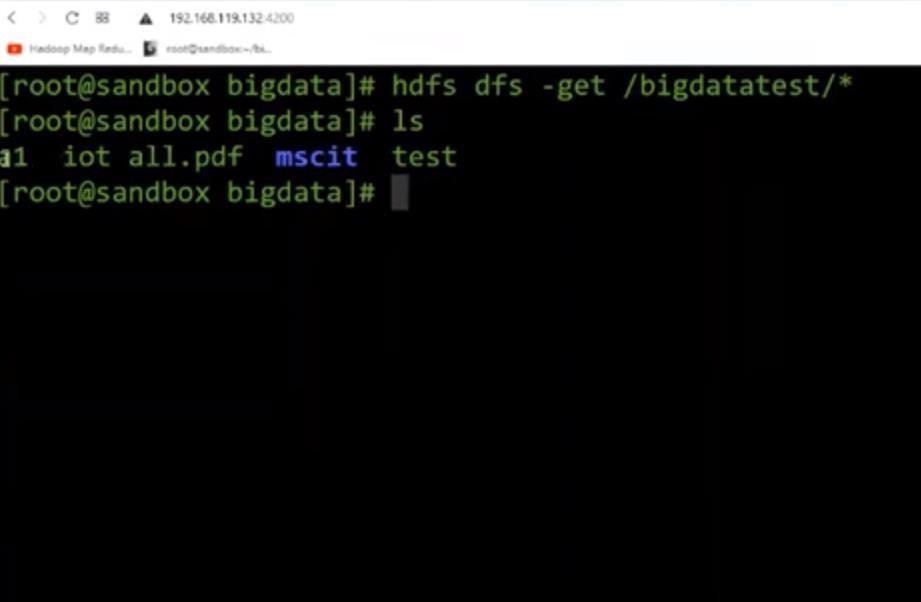
1. To remove file from Hadoop directory Command: hdfs dfs -rm a2 /gibdatatest/a2



Now refresh the UI and the file will be deleted.

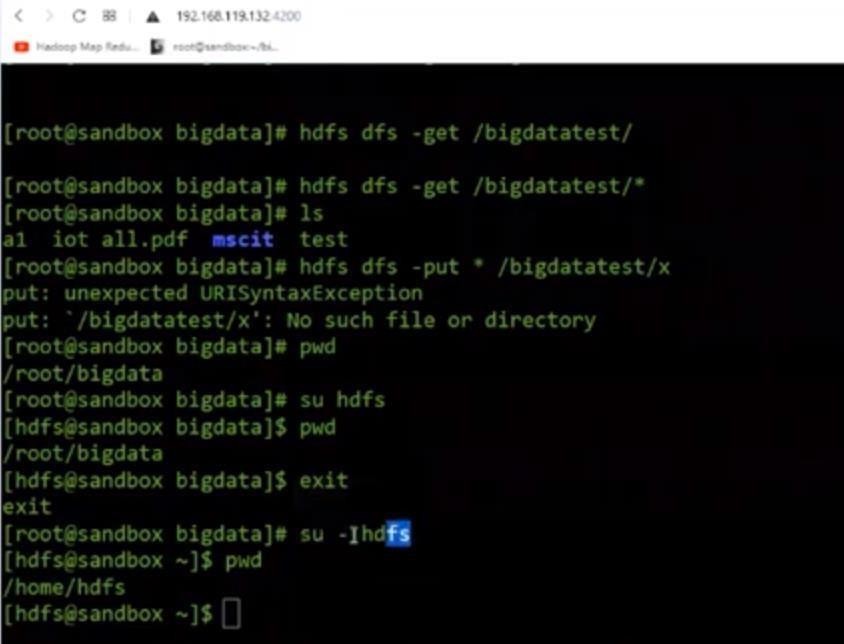


1. To download all the files from hdfs to local Command: hdfs dfs -get /bigdatatest/\*



1. Change user and directory and change user only Command: su – hdfs (Change user and directory)

Su hdfs (change user only)



##### Practical 2

Implement word count / frequency programs using MapReduce

Map Reduce as two component Map and Reduce.

**Java program:**

write program save as WordCount.java

//////////////////////// 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 {

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()) {//"This is the output is the" word.set(itr.nextToken());

context.write(word, one);

}

}

}

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

{//is,3

int sum = 0;

for (IntWritable val : values) { sum += val.get();

}

result.set(sum); context.write(key, result);

}

}

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);

}

}

////////////////////////

**Text File:**

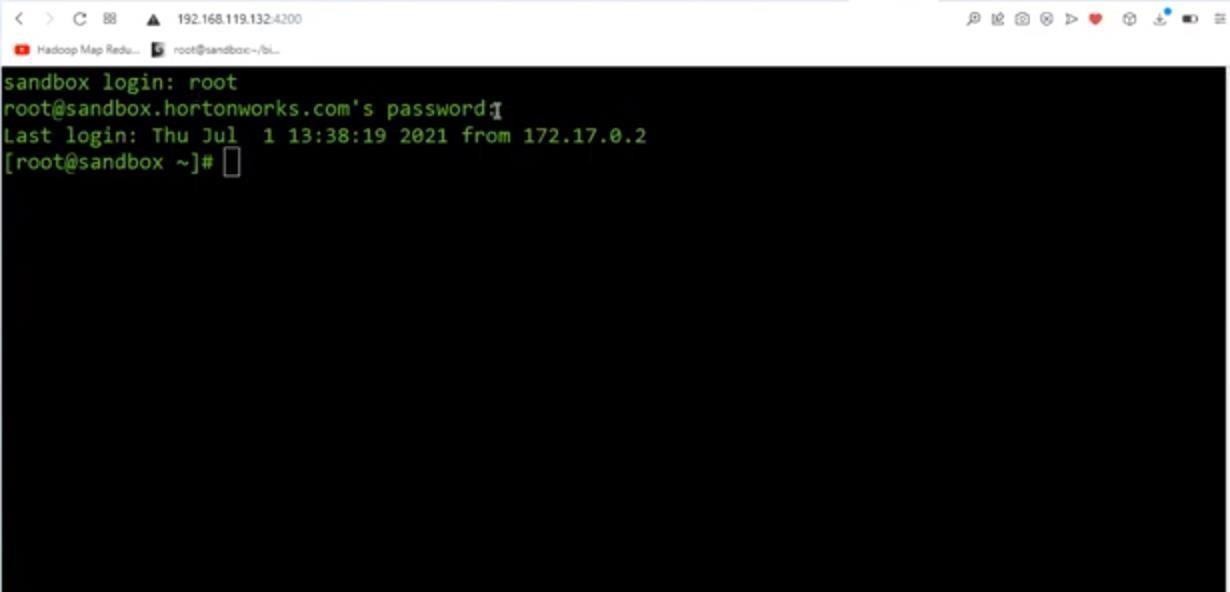
Hello World

This is the output is the

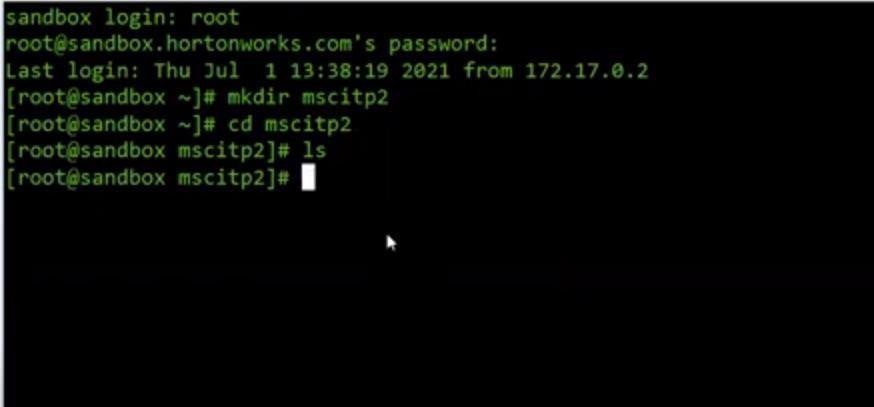
Start the server (Horton Sandbox)



Open the terminal with 192.168.119.132/4200 Enter the login: root and the password and enter



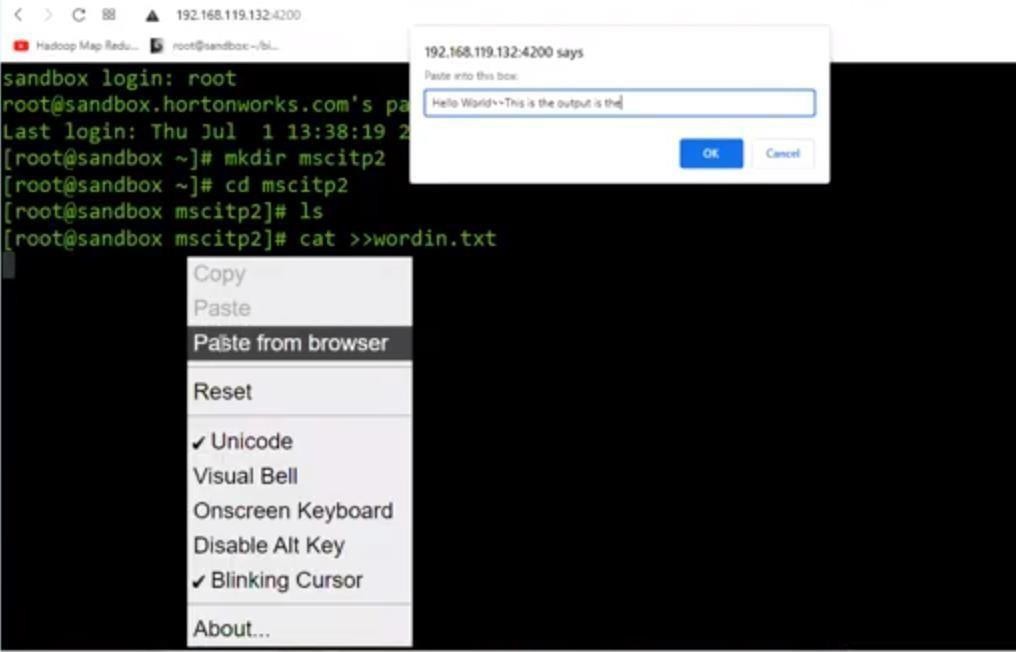
Create a folder in local directory. Command: mkdir mscitp2 Change the directory cd mscitp2

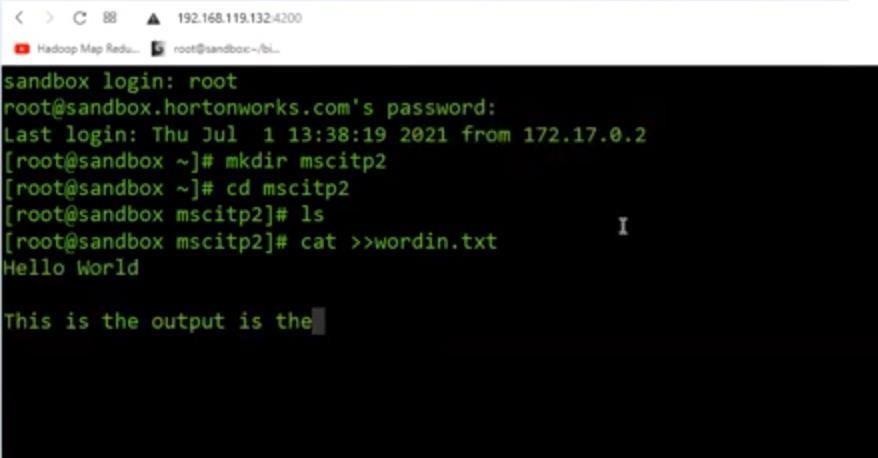


Now create input file Command: cat >> wordin.txt

Paste the text by right clicking on terminal Hello World

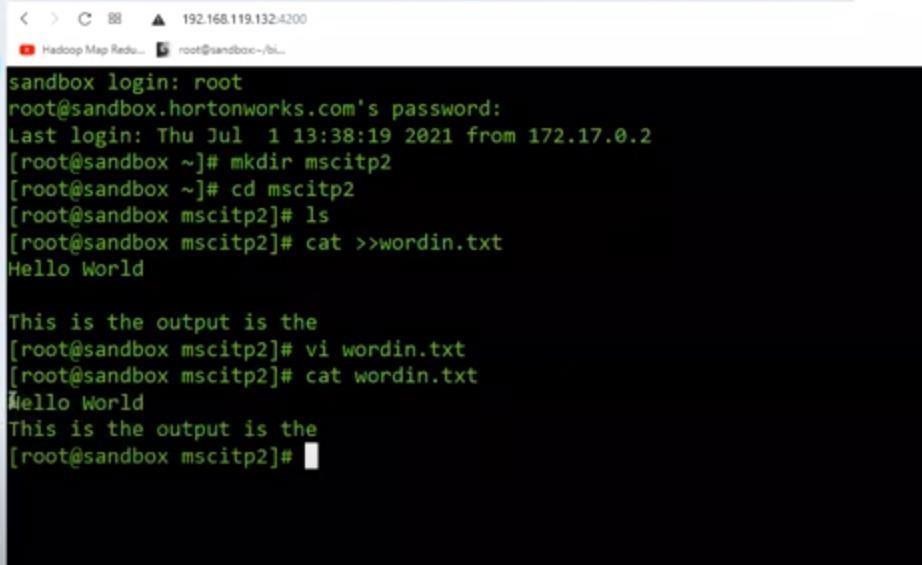
This is the output is the



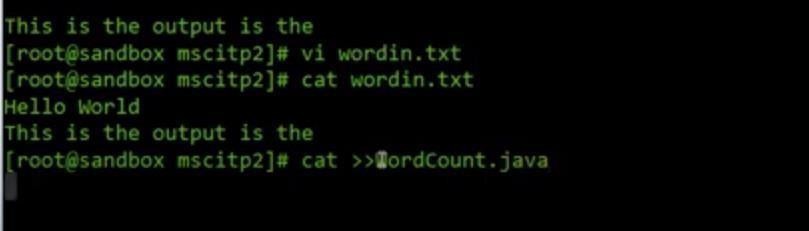


To remove the extra space type command vi wordin.txt

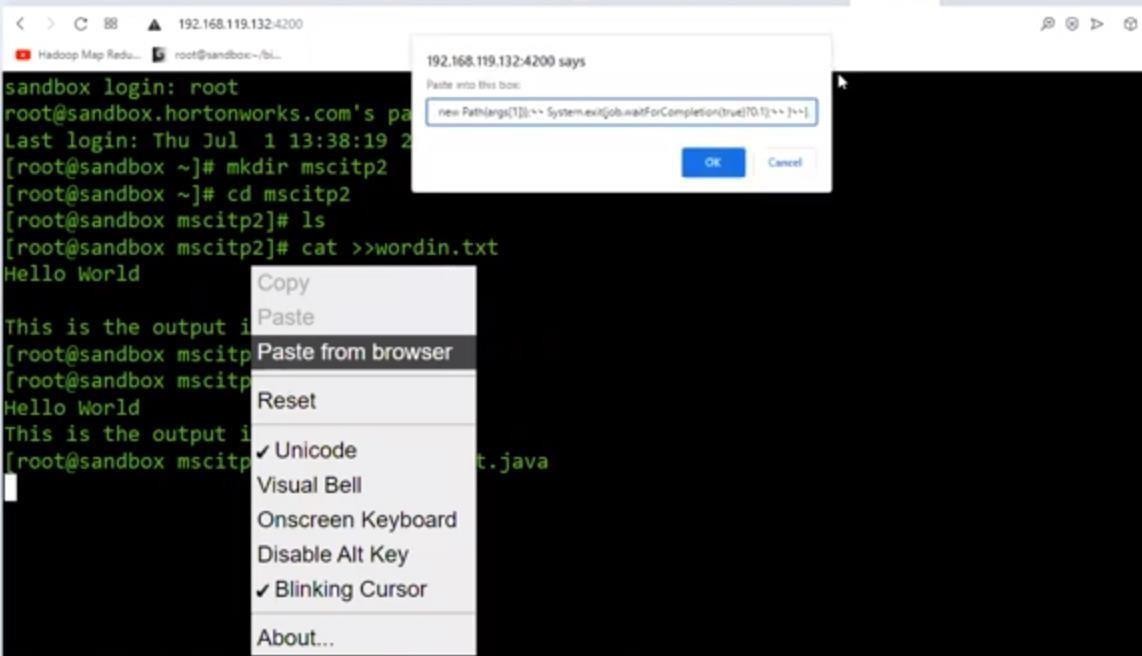
After removing the extra space check the content of the file cat wordin.txt



Create another file wordcount.java

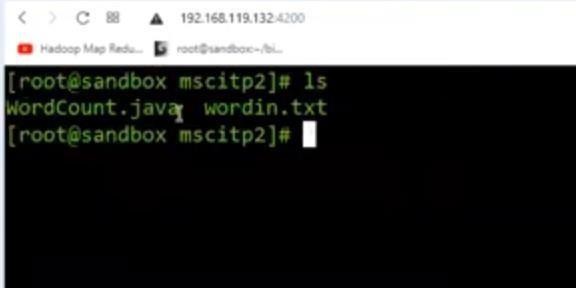


Paste the java code.



Press control d to save the file

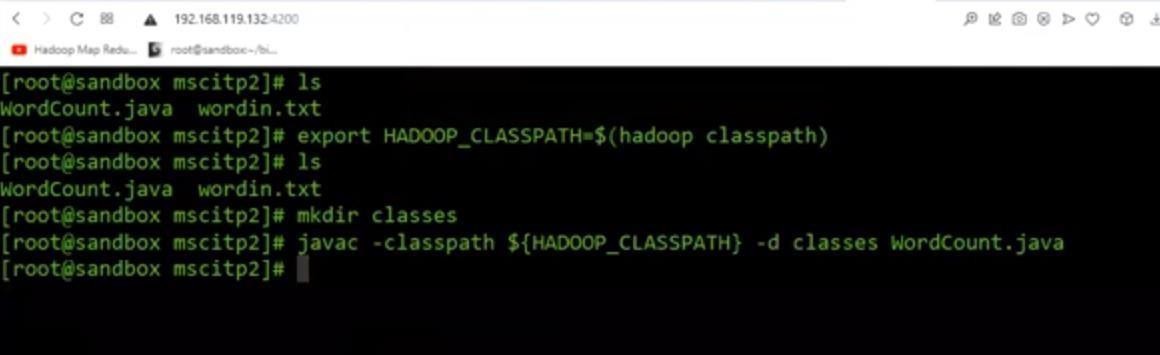
Check both the files create with command ls



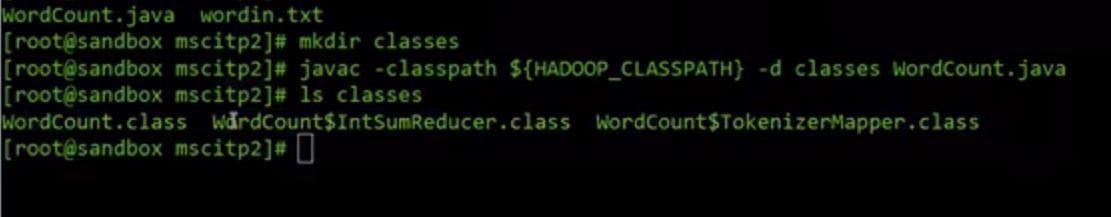
Now, to compile the java file

export HADOOP\_CLASSPATH=$(hadoop classpath) mkdir classes (To keep the compile files)

javac -classpath ${HADOOP\_CLASSPATH} -d classes WordCount.java

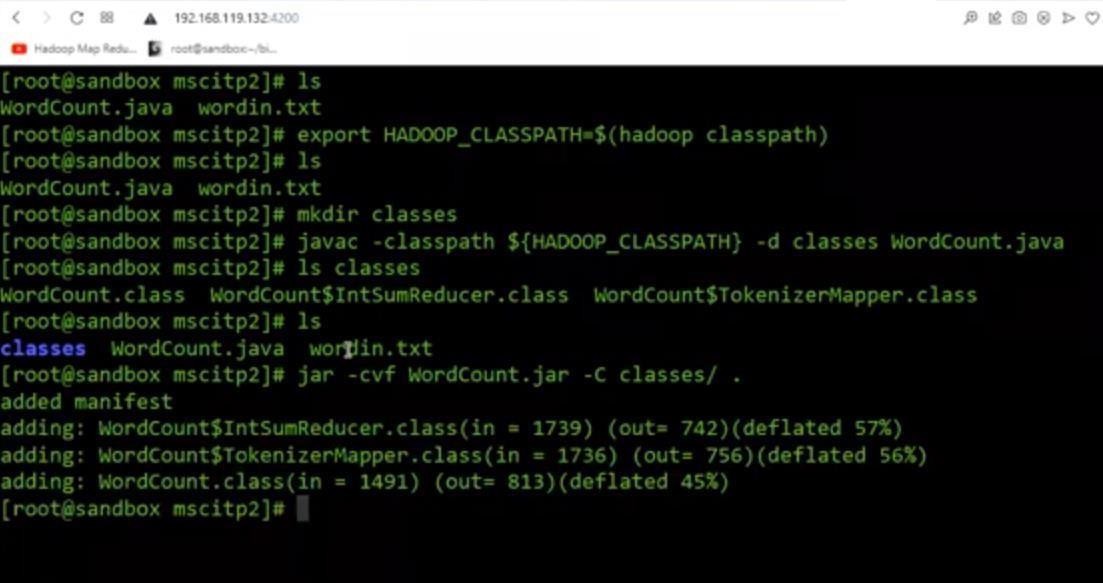


Check class files are created with command ls classes

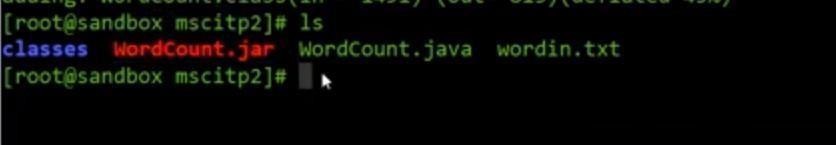


Now we have to bind all the class into single jar file with below command

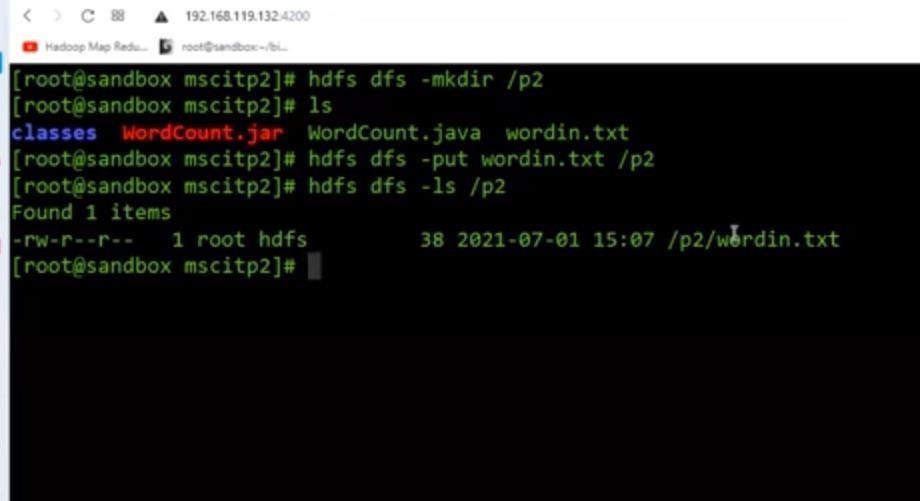
jar -cvf WordCount.jar -C classes/ .



Run ls command we can see jar file is created.

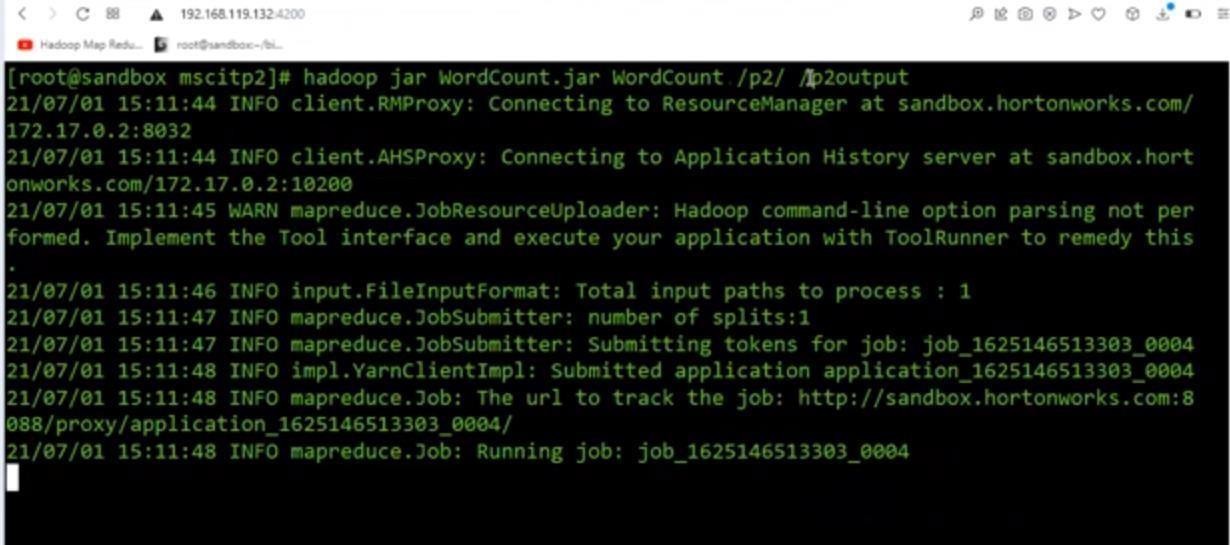


wordin.txt should be present in word directory of hdfs. So we need to upload wordin.txt file.

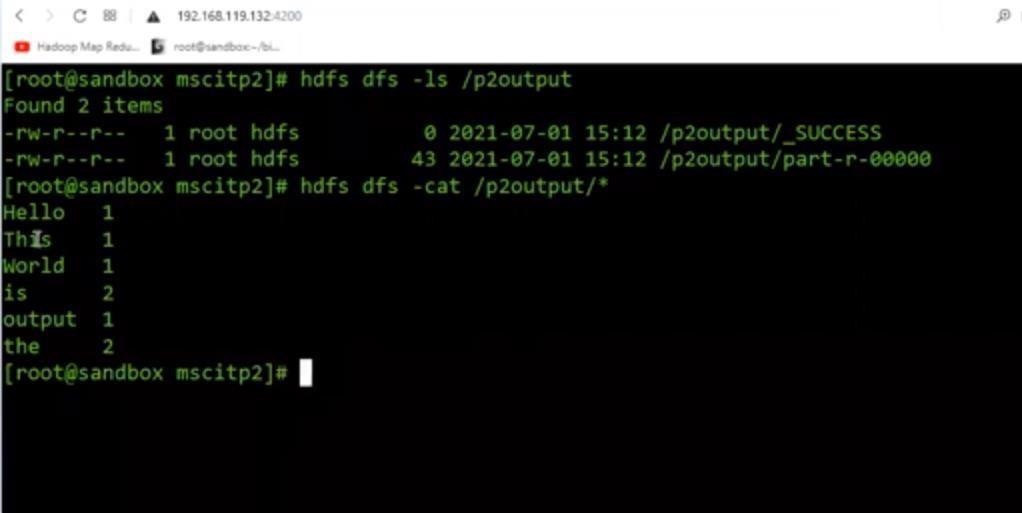


We need to put the final output p2output.

hadoop jar WordCount.jar WordCount /p2/ /p2output



Print the content of the output file Command: hdfs dfs -cat /p2ouput/\*



Ctrl + l to clear the screen.

vi filename.txt= this command will create/ open filename.txt two modes of vi editor

1. Insert mode – i (press i key)
2. Command mode – esc key

:wq is to save and exit

##### Practical 3

Implement an MapReduce program that processes a weather dataset.

**Java program:**

MyMaxMin.java

///////////////////////////

// importing Libraries importjava.io.IOException; import java.util.Iterator;

import org.apache.hadoop.fs.Path;

importorg.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; 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 MyMaxMin {

// Mapper

/\*MaxTemperatureMapper class is static

* and extends Mapper abstract class
* having four Hadoop generics type
* LongWritable, Text, Text, Text.

\*/

public static class MaxTemperatureMapper extends

Mapper<LongWritable, Text, Text, Text> {

/\*\*

* + @method map
  + This method takes the input as a text data type.
  + Now leaving the first five tokens, it takes
  + 6th token is taken as temp\_max and
  + 7th token is taken as temp\_min. Now
  + temp\_max > 30 and temp\_min < 15 are
  + passed to the reducer.

\*/

// the data in our data set with

// this value is inconsistent data public static final int MISSING = 9999;

@Override

public void map(LongWritable arg0, Text Value, Context context) throws IOException, InterruptedException {

// Convert the single row(Record) to

// String and store it in String

// variable name line

String line = Value.toString();

// Check for the empty line if (!(line.length() == 0)) {

// from character 6 to 14 we have

// the date in our dataset

String date = line.substring(6, 14);

// similarly we have taken the maximum

// temperature from 39 to 45 characters

float temp\_Max = Float.parseFloat(line.substring(39, 45).trim());

// similarly we have taken the minimum

// temperature from 47 to 53 characters

float temp\_Min = Float.parseFloat(line.substring(47, 53).trim());

// if maximum temperature is

// greater than 30, it is a hot day if (temp\_Max > 30.0) {

// Hot day

context.write(new Text("The Day is Hot Day :" + date),

new

Text(String.valueOf(temp\_Max)));

}

// if the minimum temperature is

// less than 15, it is a cold day if (temp\_Min < 15) {

// Cold day

context.write(new Text("The Day is Cold Day :" + date), new Text(String.valueOf(temp\_Min)));

}

}

}

}

// Reducer

/\*MaxTemperatureReducer class is static and extends Reducer abstract class having four Hadoop generics type

Text, Text, Text, Text.

\*/

//The Day is Cold Day :20150101 ,-21.8

public static class MaxTemperatureReducer extends

Reducer<Text, Text, Text, Text> {

/\*\*

* + @method reduce
  + This method takes the input as key and
  + list of values pair from the mapper,
  + it does aggregation based on keys and
  + produces the final context.

\*/

public void reduce(Text Key, Iterator<Text> Values, Context context) throws IOException, InterruptedException {

// putting all the values in

// temperature variable of type String

String temperature =Values.next().toString(); context.write(Key, new Text(temperature));

}

}

/\*\*

* @method main
* This method is used for setting
* all the configuration properties.
* It acts as a driver for map-reduce
* code.

\*/

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

// reads the default configuration of the

// cluster from the configuration XML files Configuration conf = new Configuration();

// Initializing the job with the

// default configuration of the cluster

Job job = new Job(conf, "weather example");

// Assigning the driver class name job.setJarByClass(MyMaxMin.class);

// Key type coming out of mapper job.setMapOutputKeyClass(Text.class);

// value type coming out of mapper

job.setMapOutputValueClass(Text.class);

// Defining the mapper class name job.setMapperClass(MaxTemperatureMapper.class);

// Defining the reducer class name job.setReducerClass(MaxTemperatureReducer.class);

// Defining input Format class which is

// responsible to parse the dataset

// into a key value pair job.setInputFormatClass(TextInputFormat.class);

// Defining output Format class which is

// responsible to parse the dataset

// into a key value pair job.setOutputFormatClass(TextOutputFormat.class);

// setting the second argument

// as a path in a path variable

Path OutputPath = new Path(args[1]);

// Configuring the input path

// from the filesystem into the job FileInputFormat.addInputPath(job, new Path(args[0]));

// Configuring the output path from

// the filesystem into the job FileOutputFormat.setOutputPath(job, new Path(args[1]));

// deleting the context path automatically

// from hdfs so that we don't have

// to delete it explicitly OutputPath.getFileSystem(conf).delete(OutputPath);

// exiting the job only if the

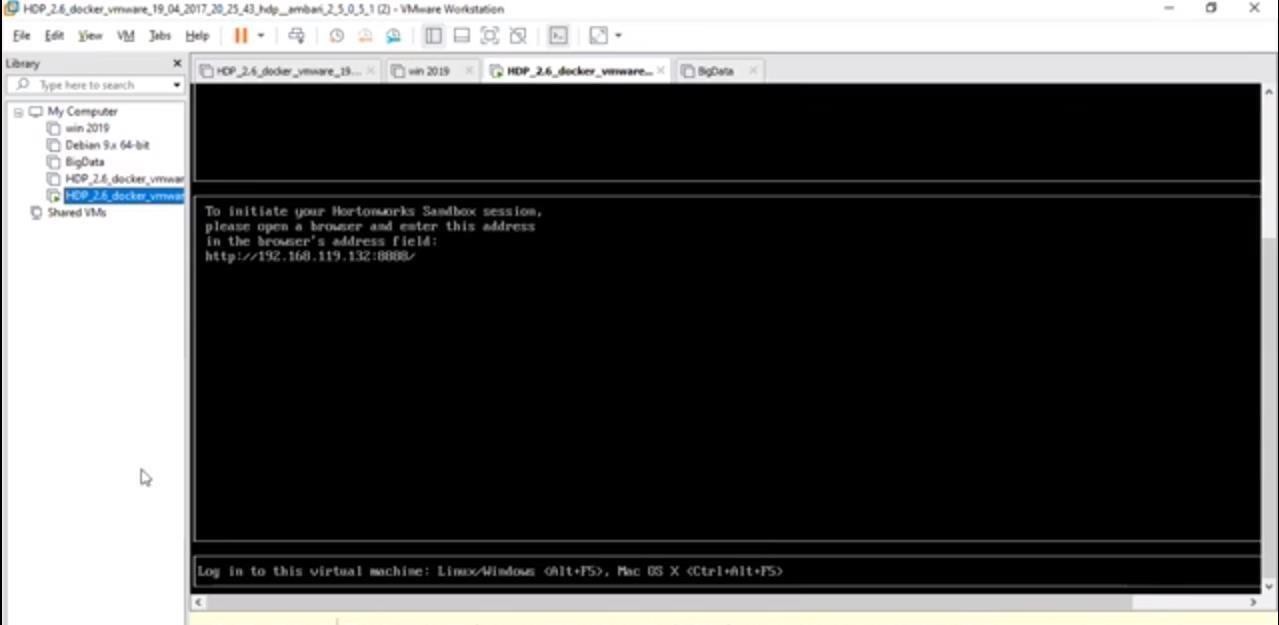
// flag value becomes false System.exit(job.waitForCompletion(true) ? 0 : 1);

}

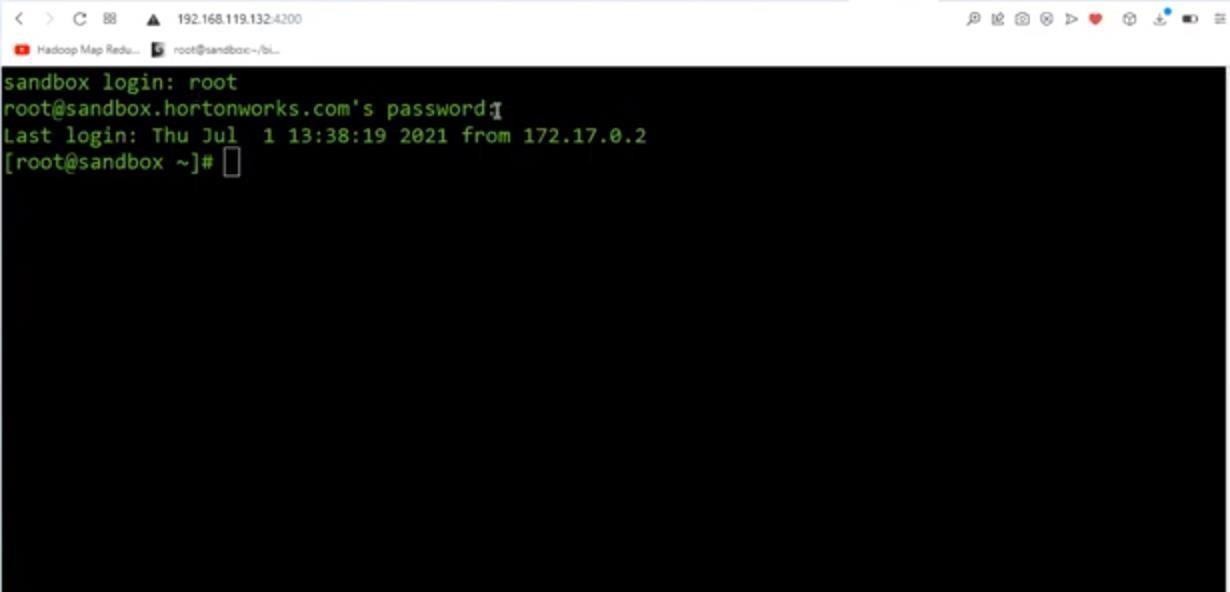
}

/////////////////////

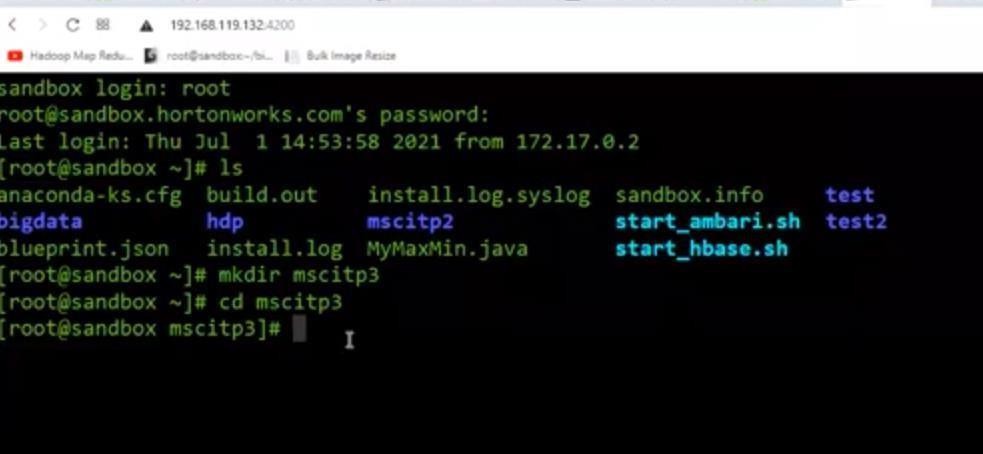
Start the server



Open the terminal with 192.168.119.132/4200 Enter the login: root and the password and enter



Create a folder in local directory. Command: mkdir mscitp3 Change the directory cd mscitp3



Now create input file

Command: cat >> weatherin2.txt

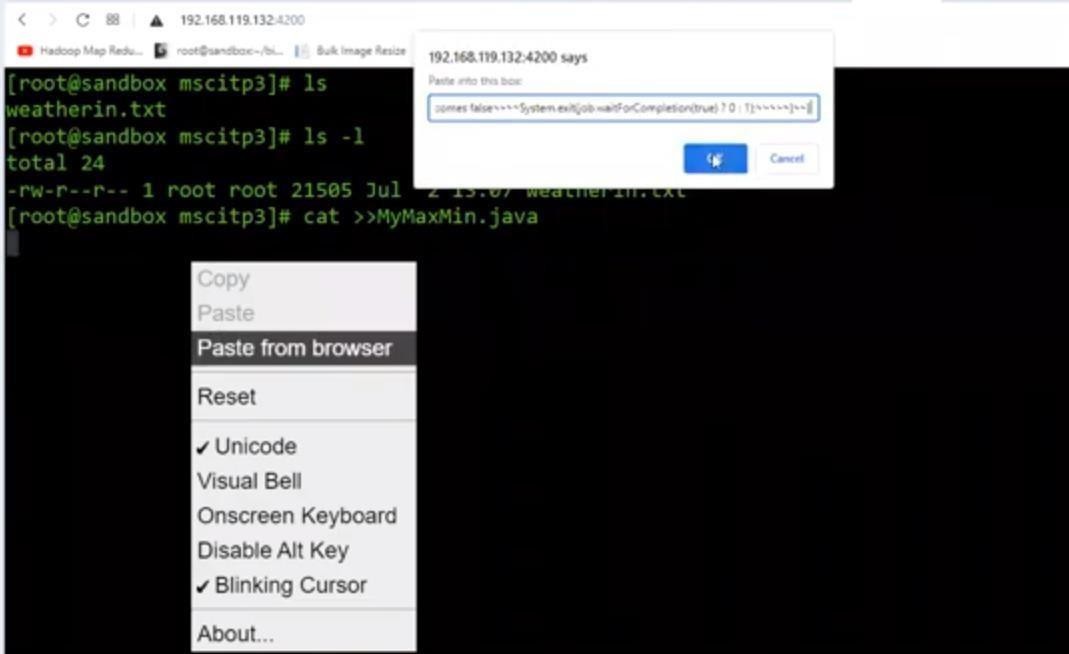
Paste the weather dataset by right clicking on terminal Ctrl d will save the file

Run command ls to see the file.

Create java file

Command: cat >>MyMaxMin.java

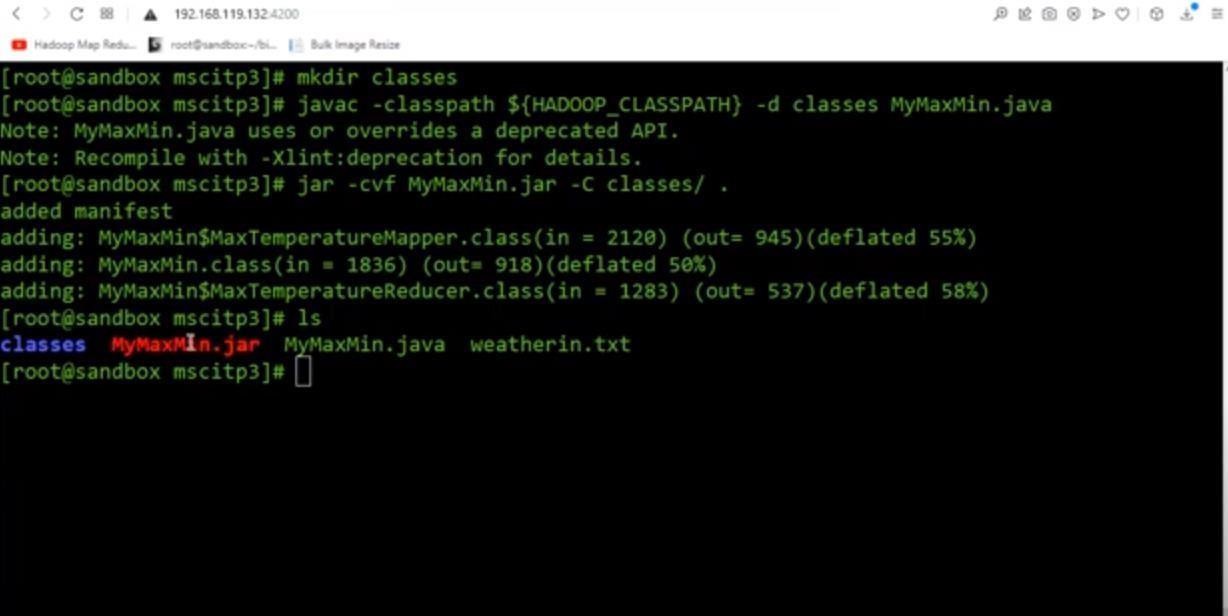
Paste the java code and ctrl d to save the file



export HADOOP\_CLASSPATH=$(hadoop classpath) ////compile and to create jar file mkdir classes

javac -classpath${HADOOP\_CLASSPATH} -d classes MyMaxMin.java After compile need to create a jar file

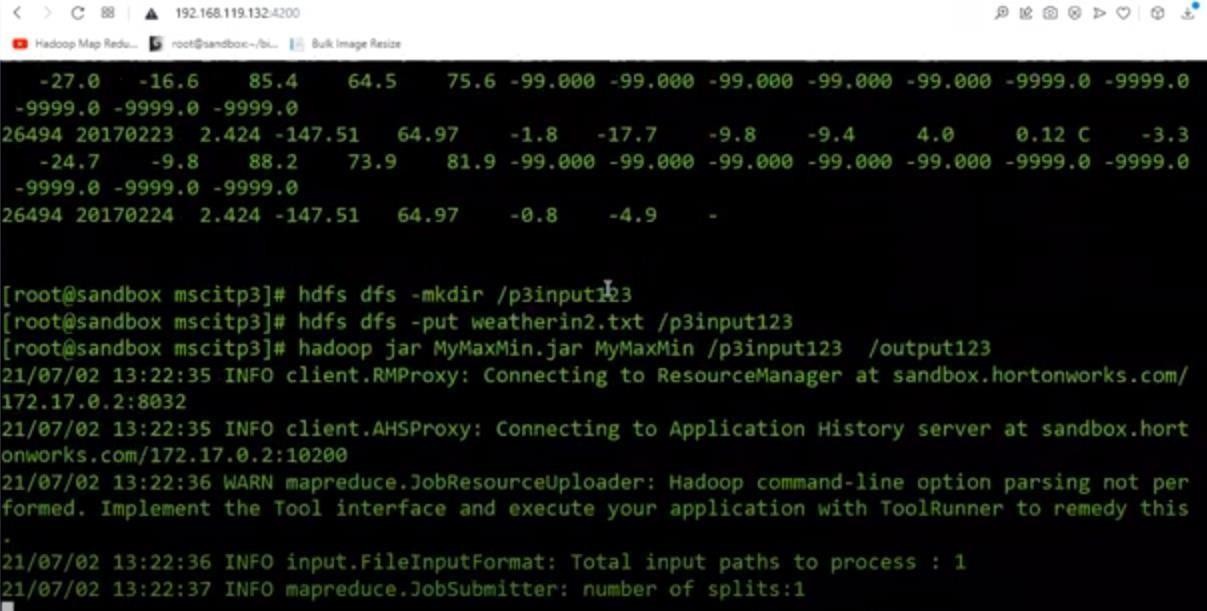
jar -cvf MyMaxMin.jar -C classes/ .



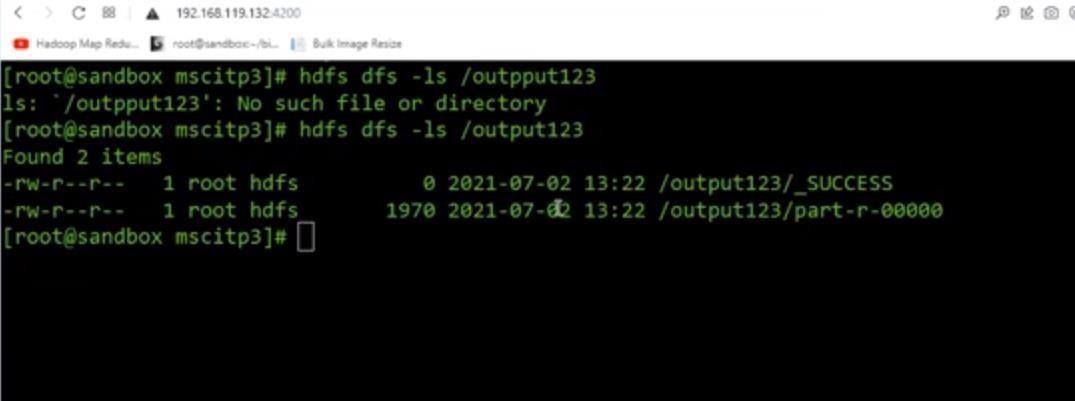
Now, put weatherin.txt in hdfs Before that create a folder

Command: hdfs dfs -mkdir /p3input123

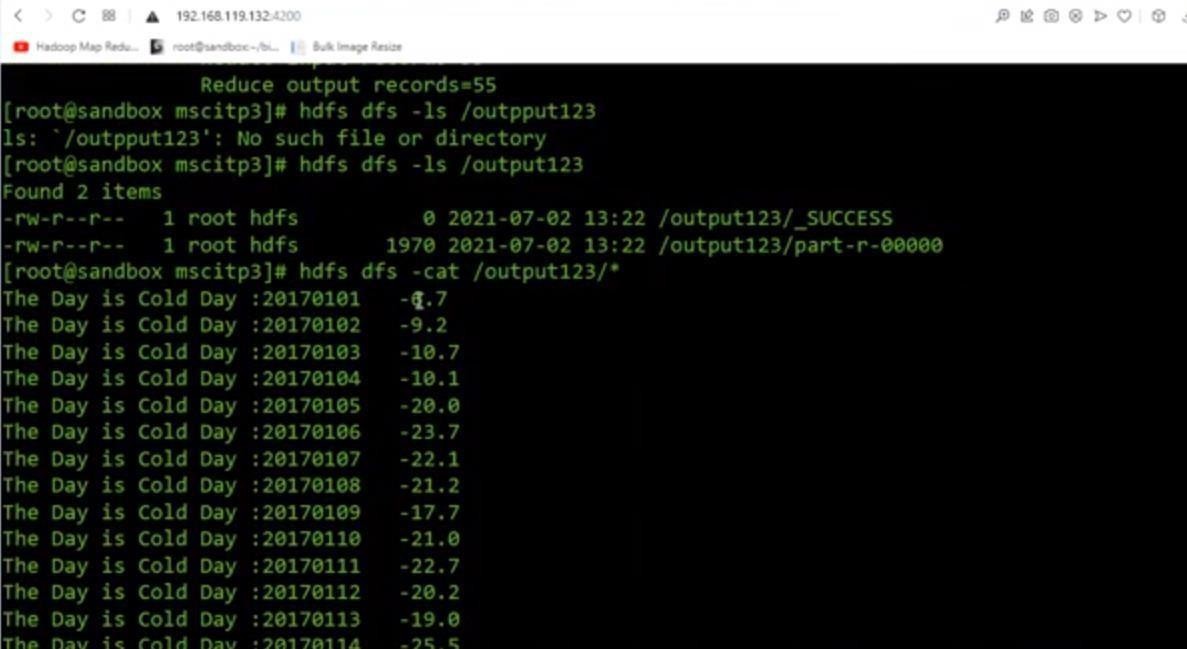
Then run command: hdfs dfs -put weatherin2.txt /p3input123 hadoop jar MyMaxMin.jar MyMaxMin /p3inputw /output123



Check outfile is created Command: hdfs dfs -ls /output123



hdfs dfs -cat /output123/\*



##### Practical 4

Implement the program using Pig.

Dataset:

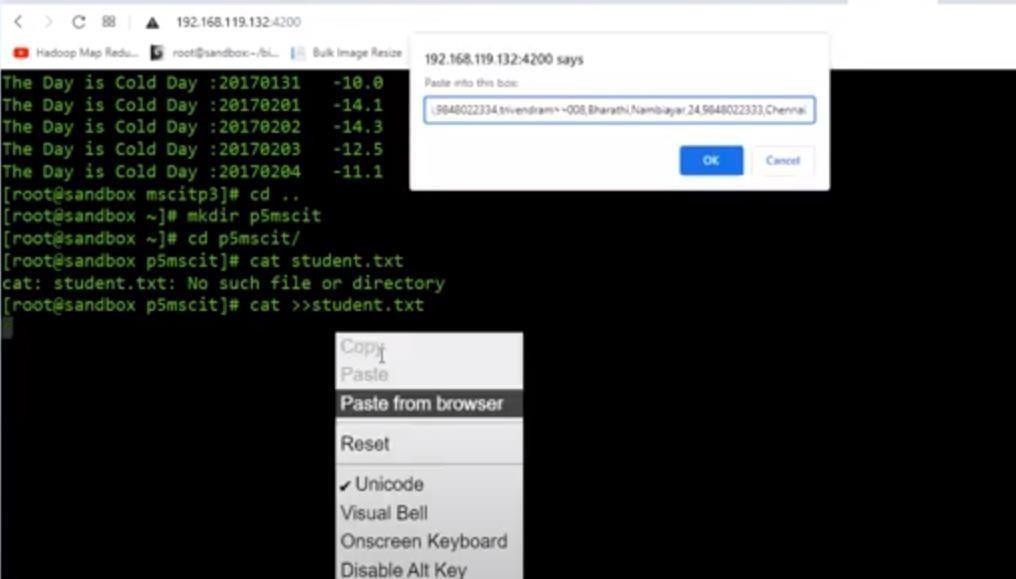
001,Rajiv,Reddy,21,9848022337,Hyderabad 002,siddarth,Battacharya,22,9848022338,Kolkata 003,Rajesh,Khanna,22,9848022339,Delhi 004,Preethi,Agarwal,21,9848022330,Pune 005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar 006,Archana,Mishra,23,9848022335,Chennai 007,Komal,Nayak,24,9848022334,trivendram 008,Bharathi,Nambiayar,24,9848022333,Chennai #student.txt

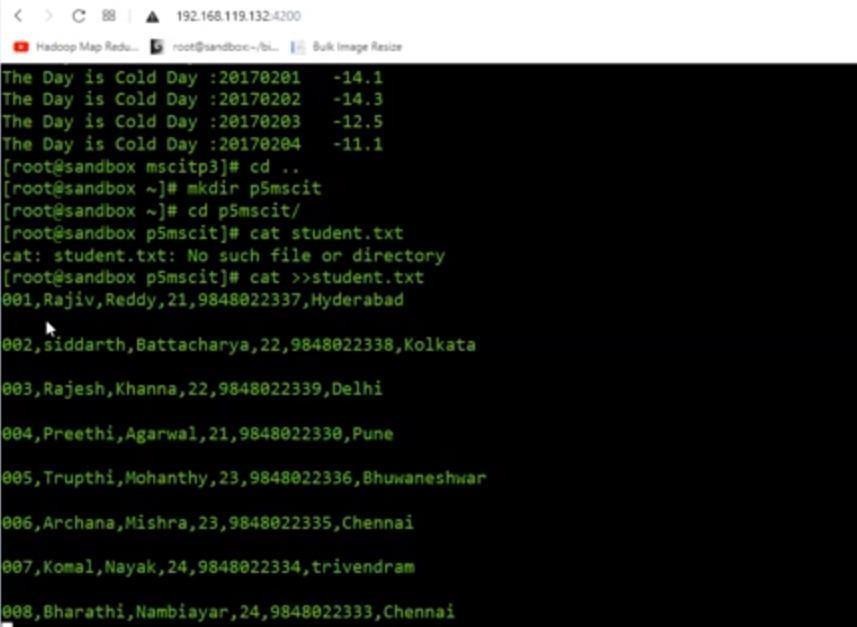
create a directory and get into that directory Command: mkdir p5mscit



Create a file

Command: cat >>student.txt Right click and paste the text





Remove the space with vi editor

Command: vi student.txt and press i for insert mode After editing: wq and enter

Print the content and see the text



Create a program file

/////////////////////////script start

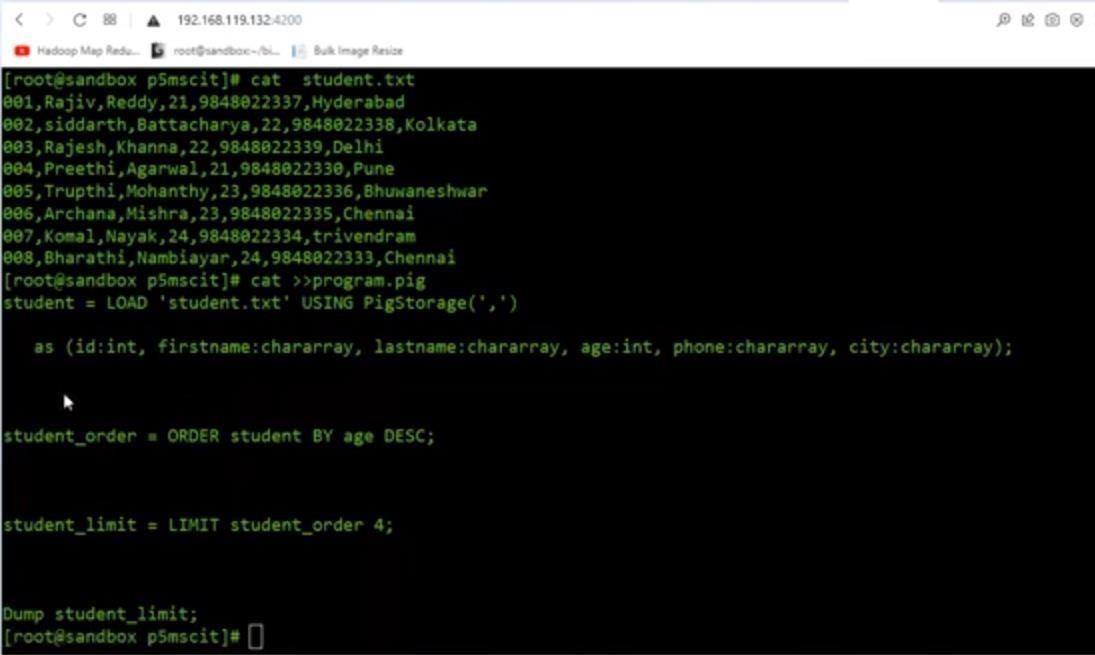
student = LOAD 'student.txt' USING PigStorage(',')

as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

student\_order = ORDER student BY age DESC;

student\_limit = LIMIT student\_order 4; Dump student\_limit;

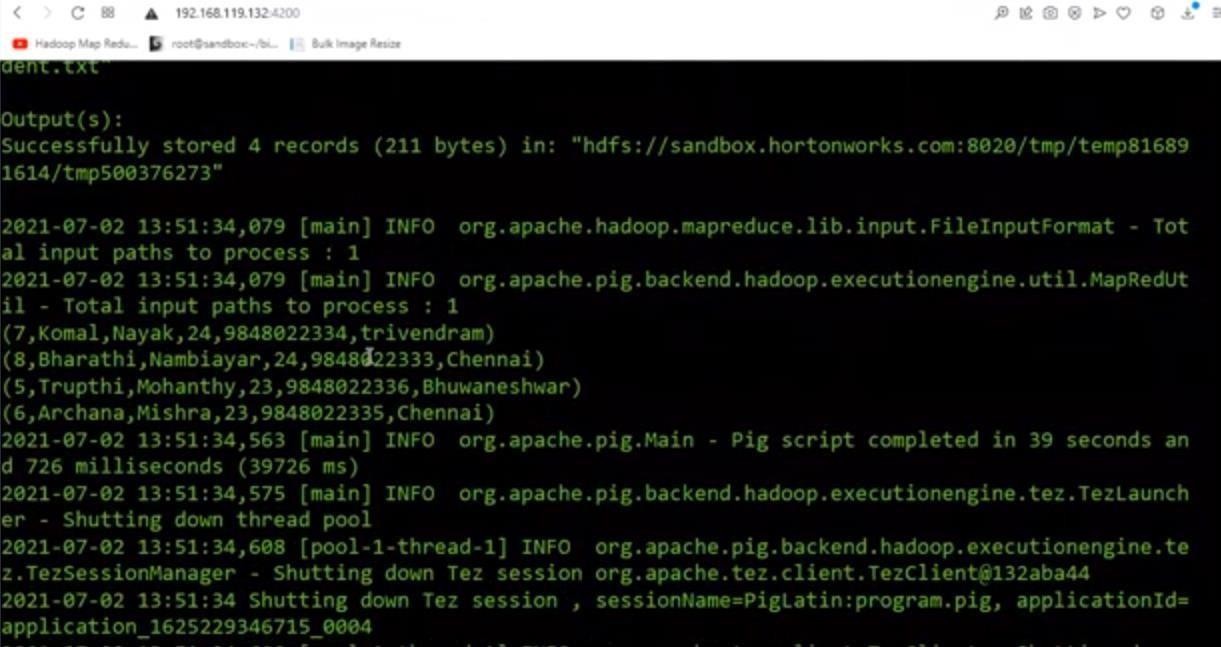
////////script end



Upload student on hdfs

Command: hdfs dfs -put student.txt /user/root/ Run the pig program

Output:



##### Practical 5

Implement the application in Hive.

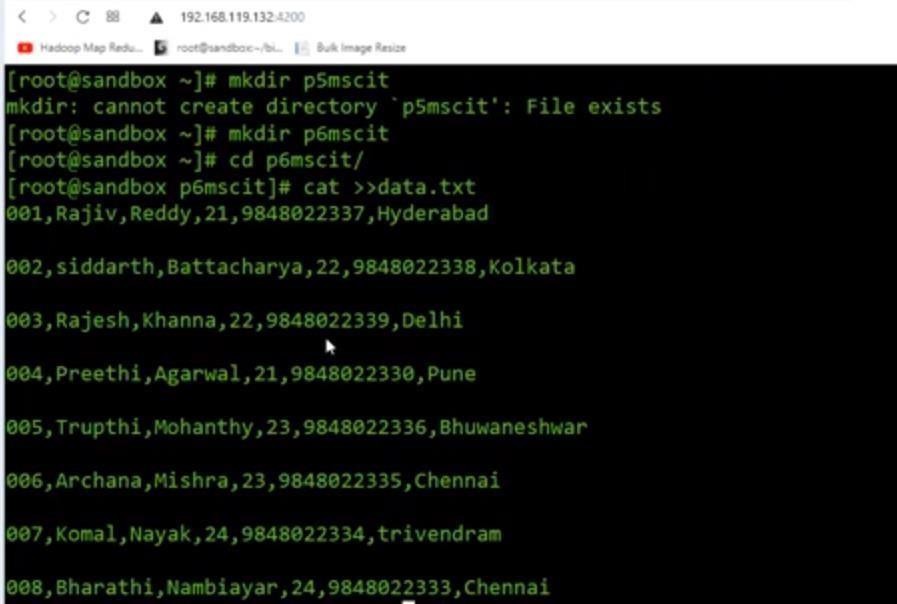
Dataset:

001,Rajiv,Reddy,21,9848022337,Hyderabad 002,siddarth,Battacharya,22,9848022338,Kolkata 003,Rajesh,Khanna,22,9848022339,Delhi 004,Preethi,Agarwal,21,9848022330,Pune 005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar 006,Archana,Mishra,23,9848022335,Chennai 007,Komal,Nayak,24,9848022334,trivendram 008,Bharathi,Nambiayar,24,9848022333,Chennai #student.txt

create a directory and get into that directory Command: mkdir p6mscit

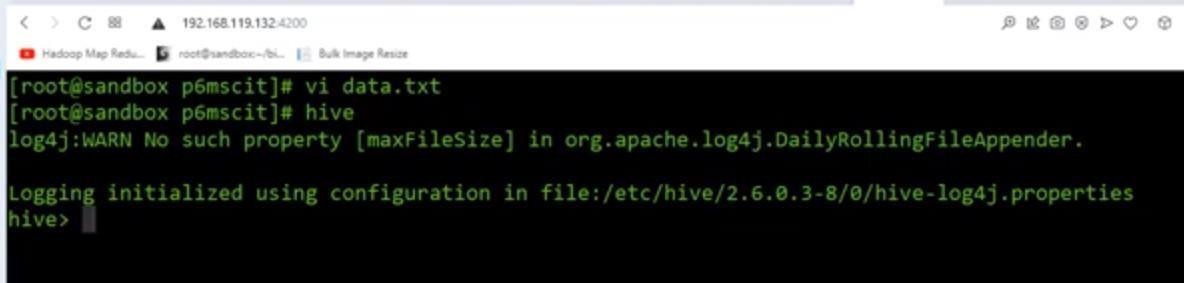
Create a file

Command: cat >>data.txt Right click and paste the text



Remove the space with vi editor

Command: vi student.txt and press i for insert mode After editing: wq and enter

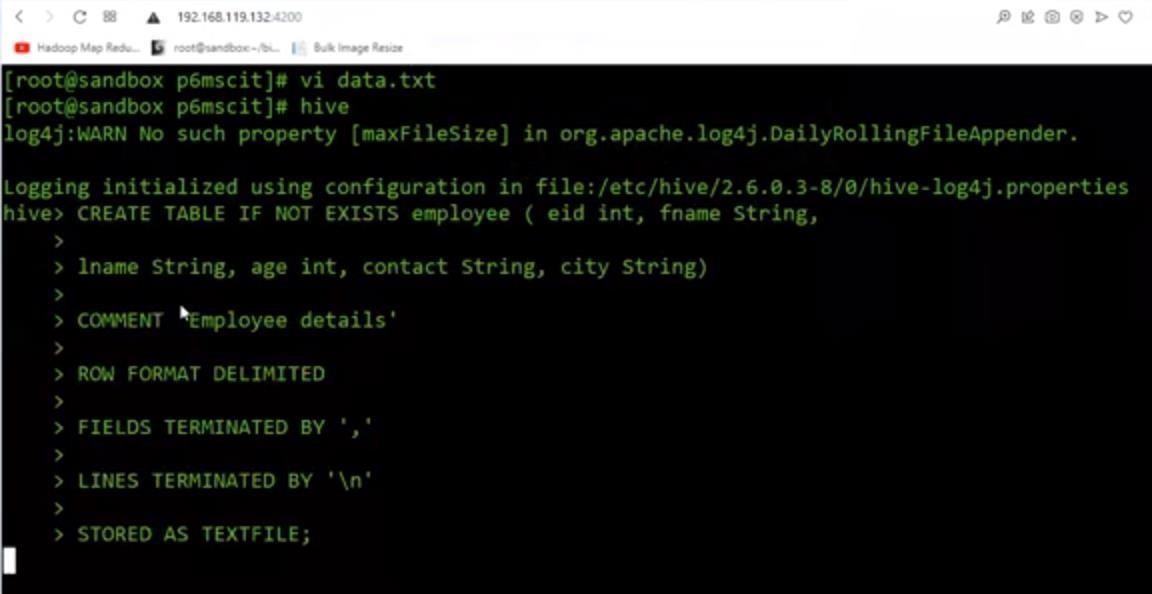
Print the content and see the text Now start the hive terminal Command: hive

Copy paste below command on hive and enter

create table

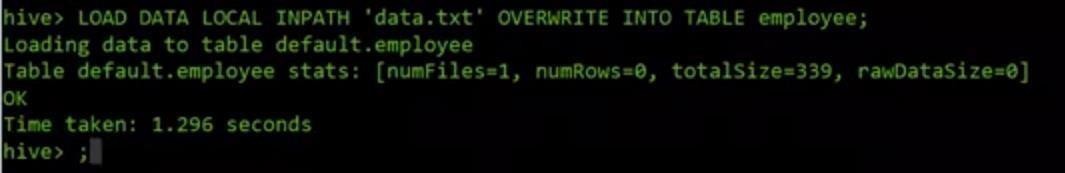
CREATE TABLE IF NOT EXISTS employee ( eid int, fname String, lname String, age int, contact String, city String)

COMMENT 'Employee details' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;

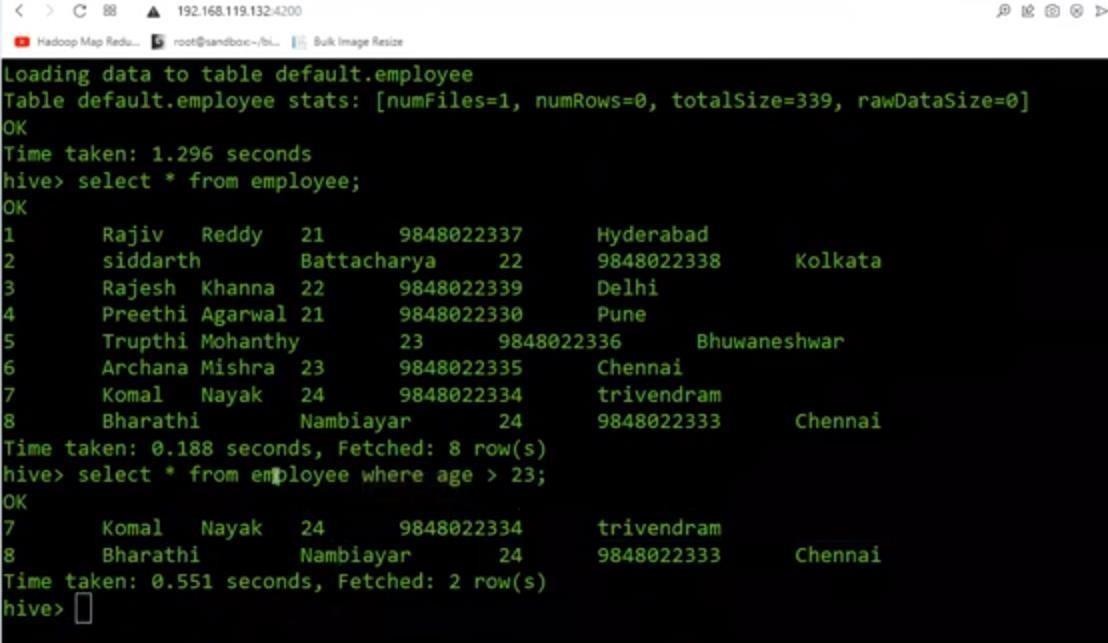


Run command: LOAD DATA LOCAL INPATH 'data.txt' OVERWRITE INTO TABLE

employee;



Run the command like select \* from employee;



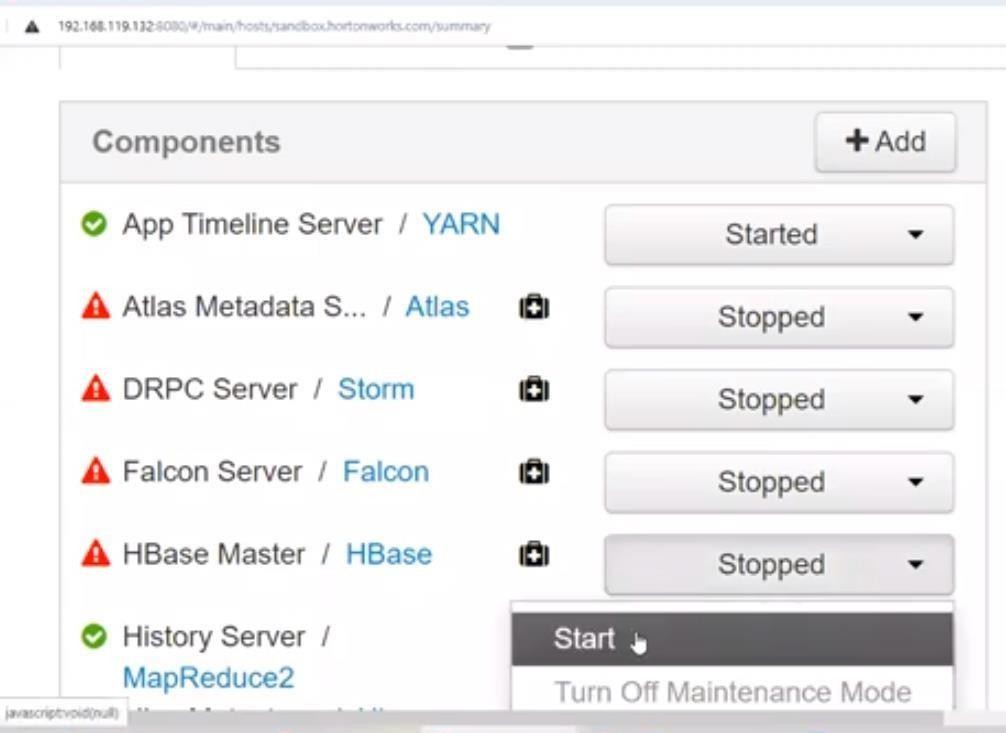
##### Practical 6

Implement an application that stores big data in Hbase/ Python

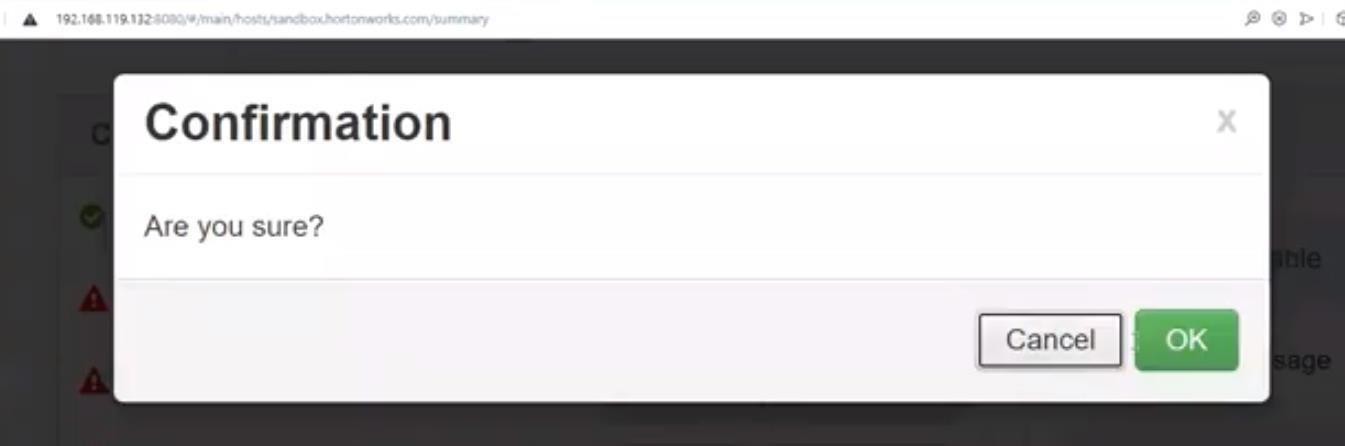
What is HBase?

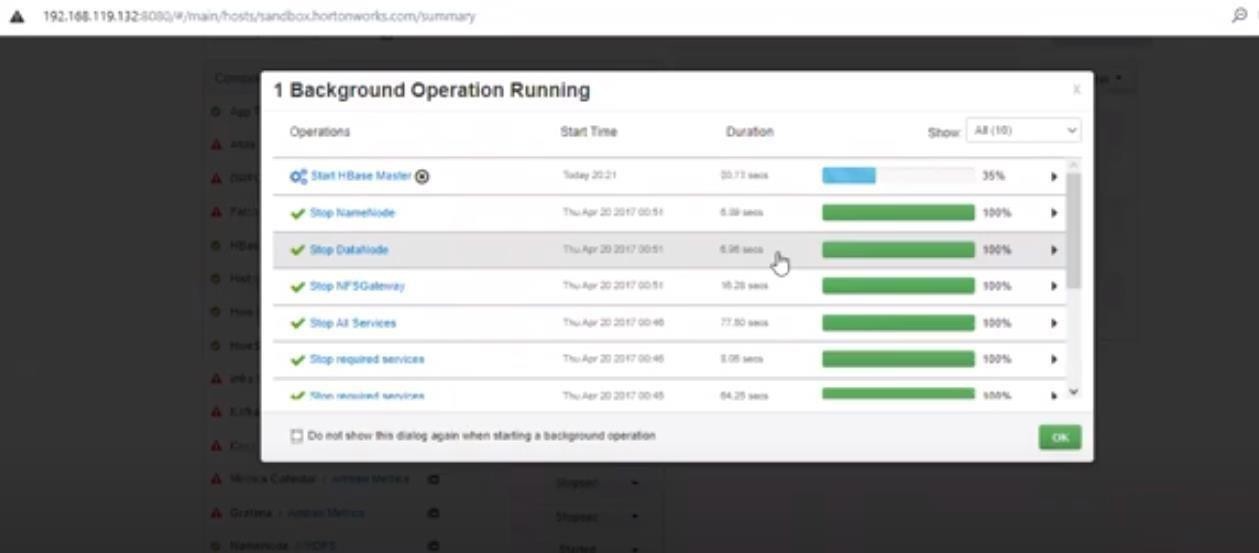
HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open- source project and is horizontally scalable. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

Go to GUI page and start the hbase service.

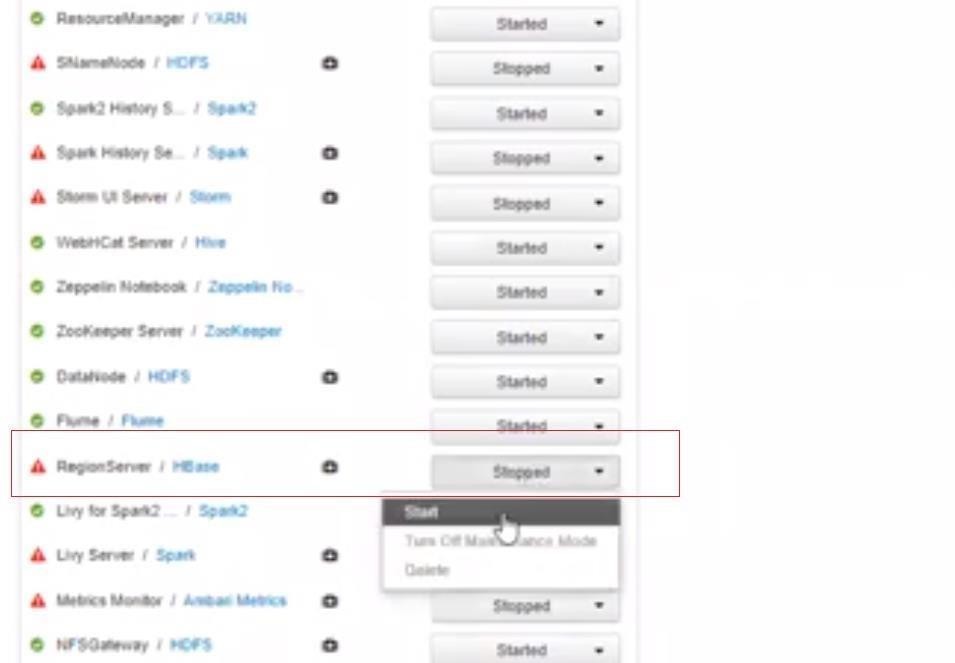


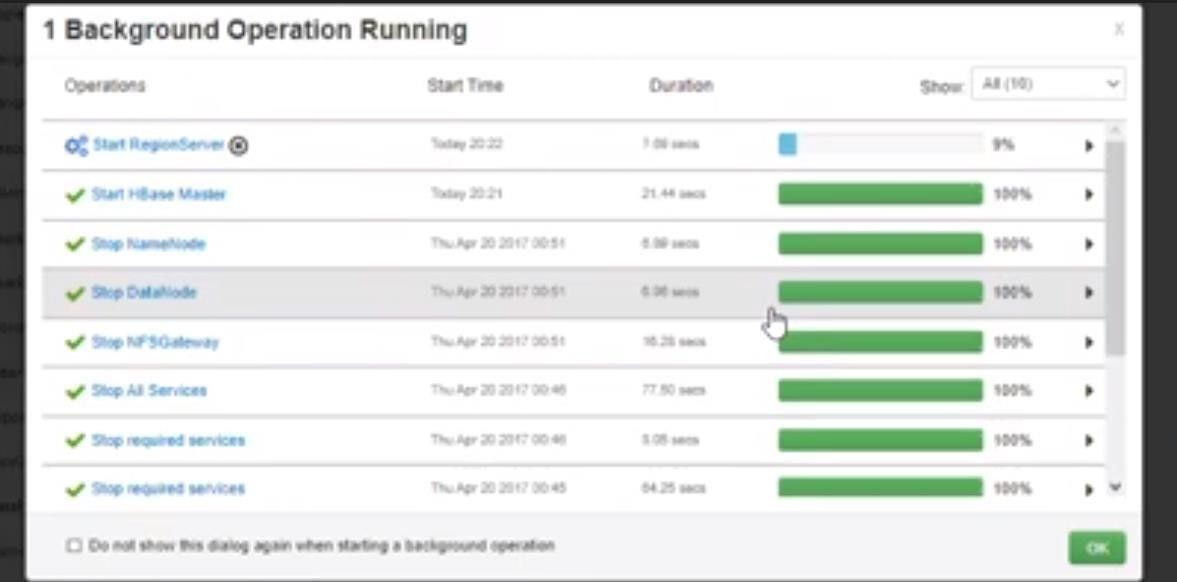
Click on OK to start the service.



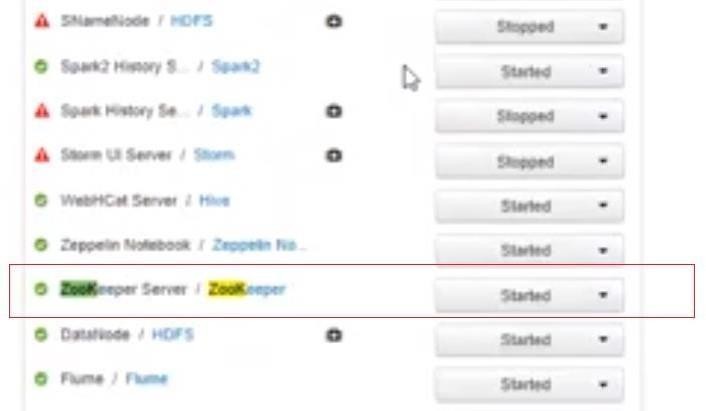


Now we must start region server.

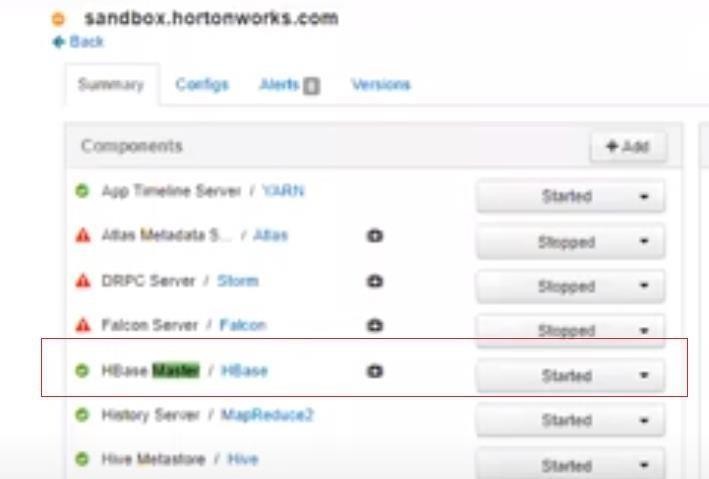




Check zooperkeeper server is started.



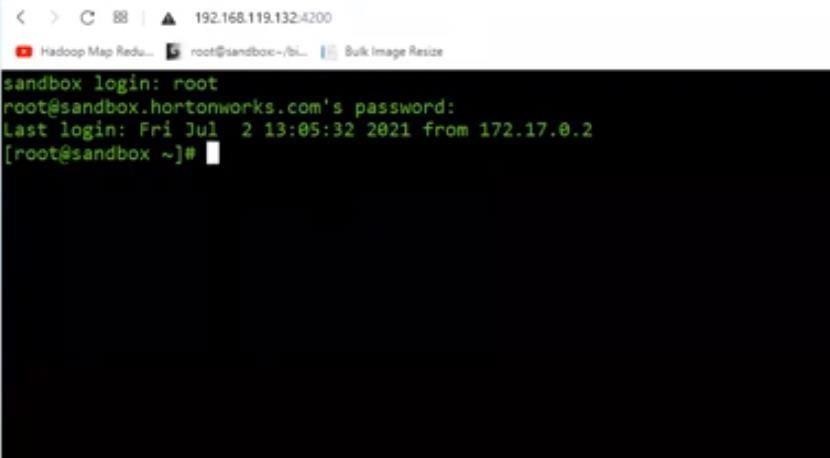
Check hbase and region server are started.





Command: **which application-name** gives directory in which application-name is installed.

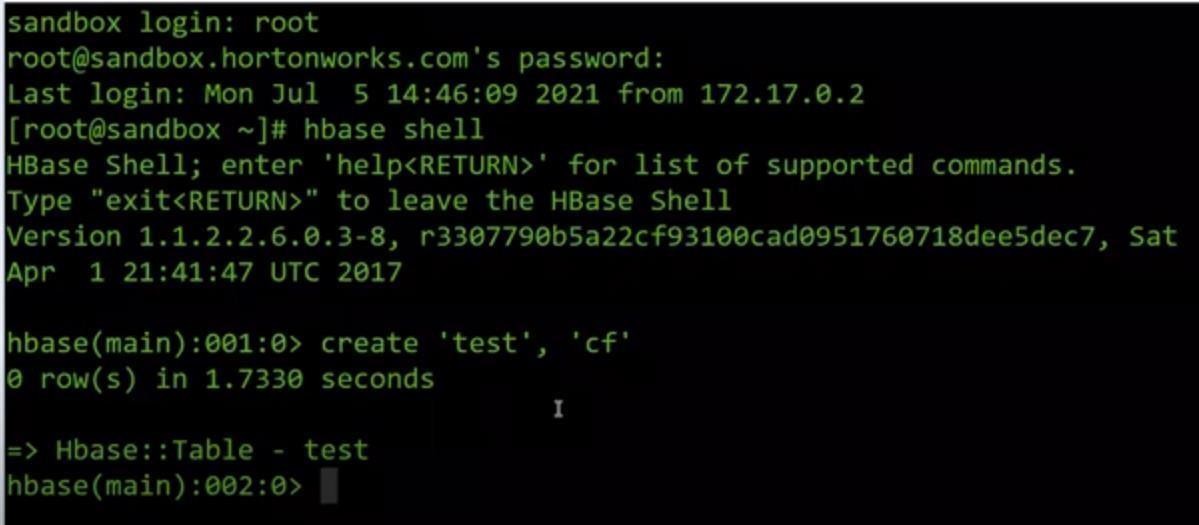
Open the shell 192.168.119.132:4200



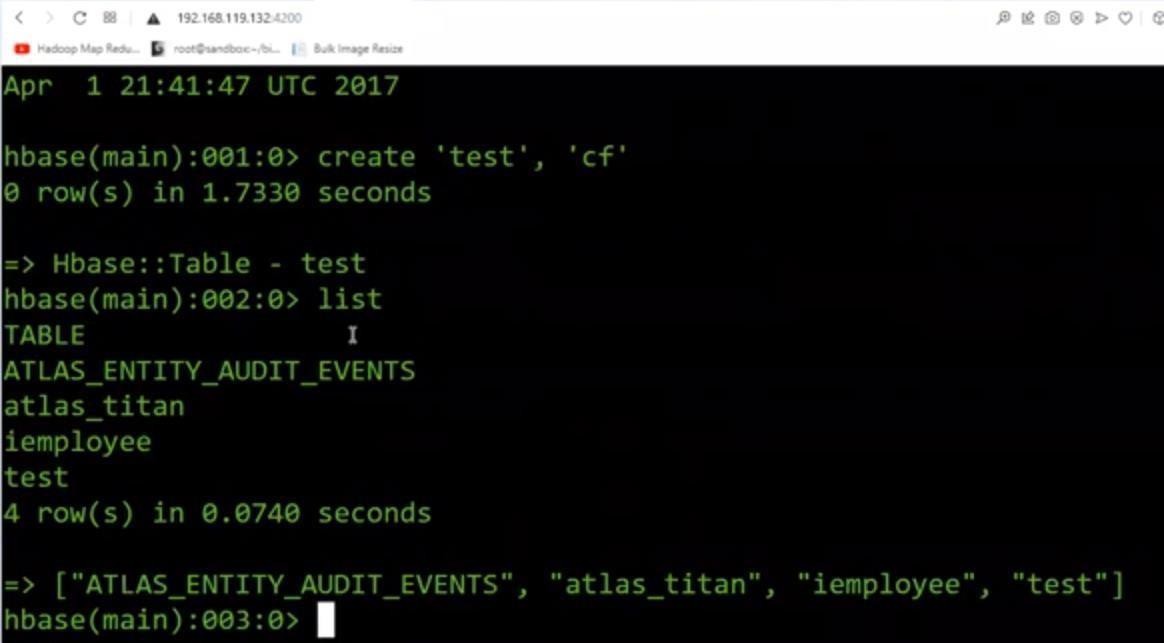
Command: hbaseshell It will start the server

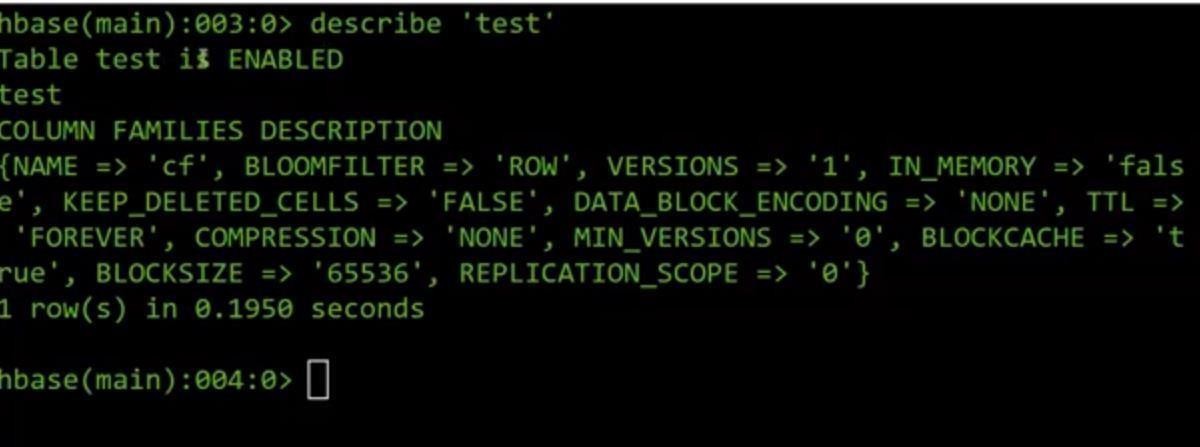


Enter the command create ‘test’, ‘cf’ and it will create the table



Check the table is created with command List- It will list all the tables created.



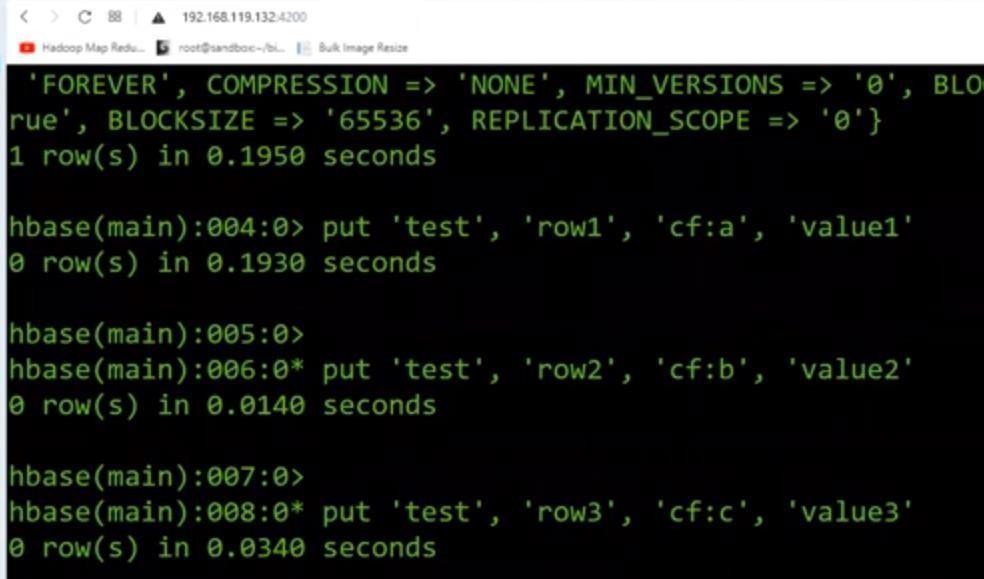
If we want to see column description of a table. Command- describe tablename

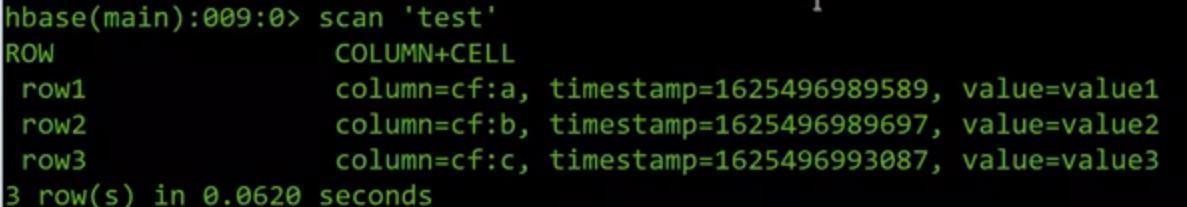
Now, we have to put the values in table Values:

put ‘test’, ‘row1’, ‘cf:a’, ‘value1’

put ‘test’, ‘row2’, ‘cf:b’, ‘value2’

put ‘test’, ‘row3’, ‘cf:c’, ‘value3’ copy paste the data in shell.

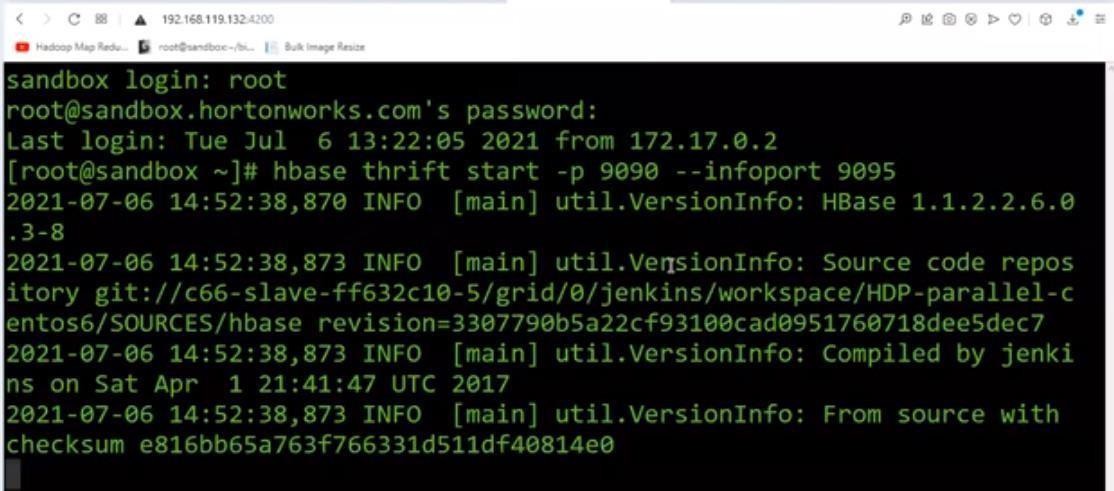


We to display the records of table Command: scan ‘test’

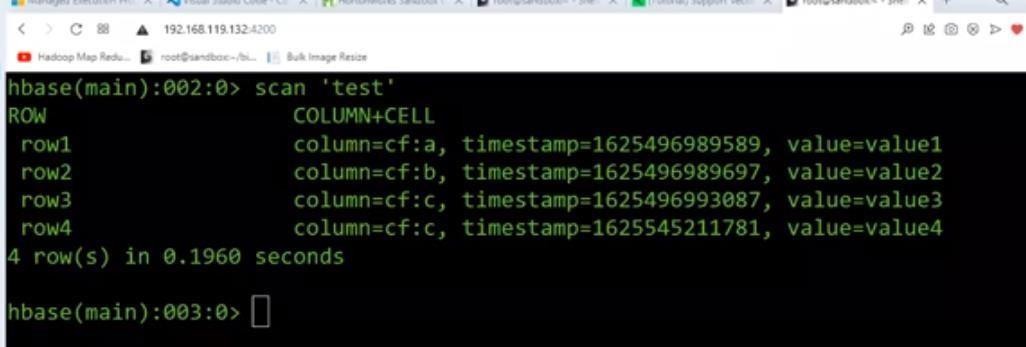
**Python: storage/retrieval**

Start the service with command

Hbase thrift start -p 9090 –inforport 9095



Create the table the way we did it in hbase and see the records using scan command



Create a program file Import happybase as hb

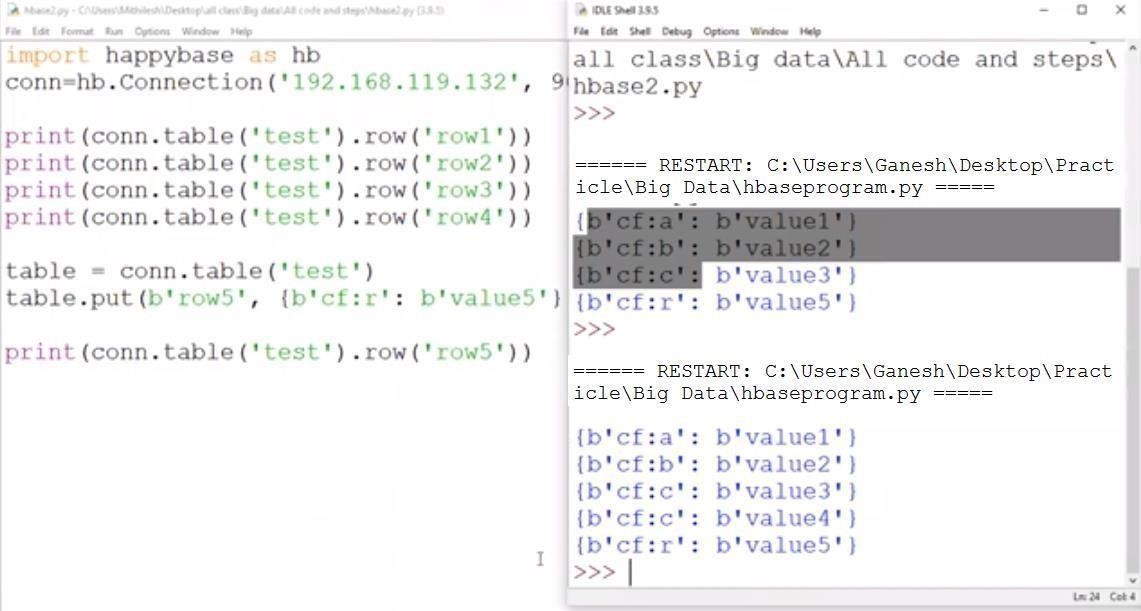
conn=hb.connection(‘192.168.119.132’, 9090) print(conn.table(‘test’).row(‘row1’)

print(conn.table(‘test’).row(‘row2’)

print(conn.table(‘test’).row(‘row3’)

print(conn.table(‘test’).row(‘row4’) table = conn.table(‘test’) table.put(b’row5’, {b’cf:r’: b’value5’})

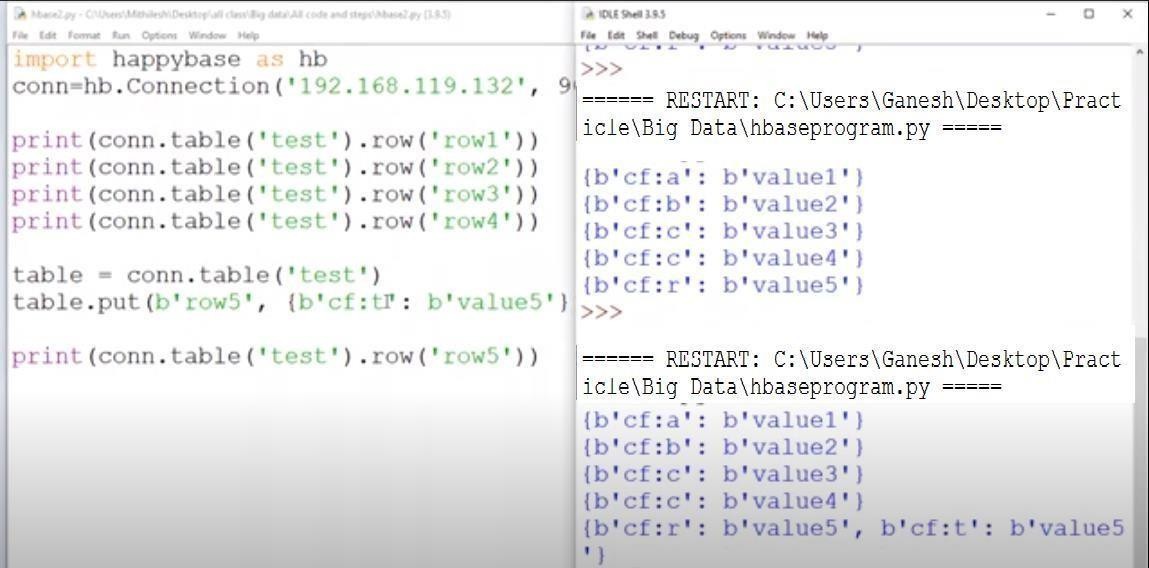
print(conn.table(‘test’).row(‘row5’)



Run a scan command on shell to display the values

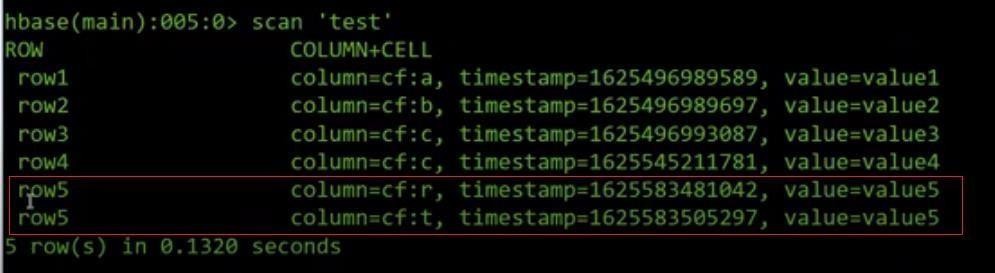


Now, try with duplicate value at row 5 say value t



Run a scan command on shell to display the values

When there is unique value, it will create a record. If duplicate value it will not create a record



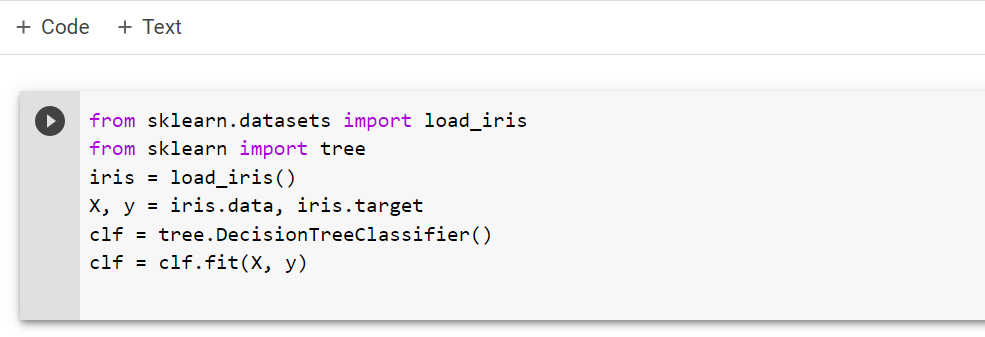
##### Practical 7

Implement Decision tree classification techniques

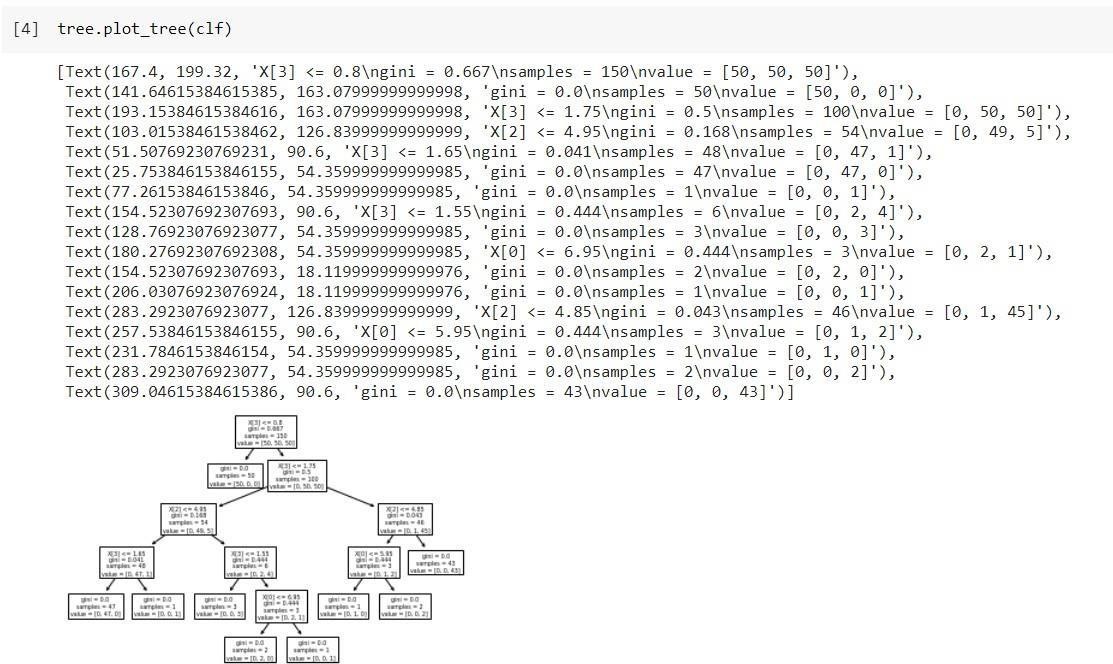
**Decision Trees (DTs)** are a non-parametric supervised learning method used

for [classification](https://scikit-learn.org/stable/modules/tree.html#tree-classification) and [regression.](https://scikit-learn.org/stable/modules/tree.html#tree-regression) The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Using the Iris dataset, we can construct a tree as follows:



Once trained, we can plot the tree with the [**plot\_tree**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html#sklearn.tree.plot_tree)function:



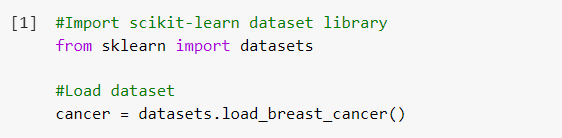
##### Practical 8

Implement SVM classification techniques

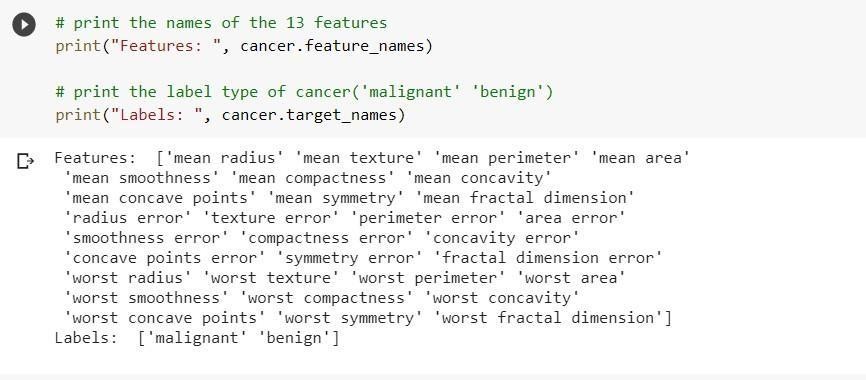
**Support Vector Machines**

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane (MMH) that best divides the dataset into classes.

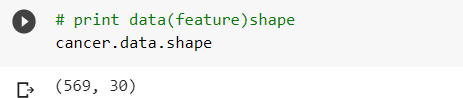
**Loading data:**

****

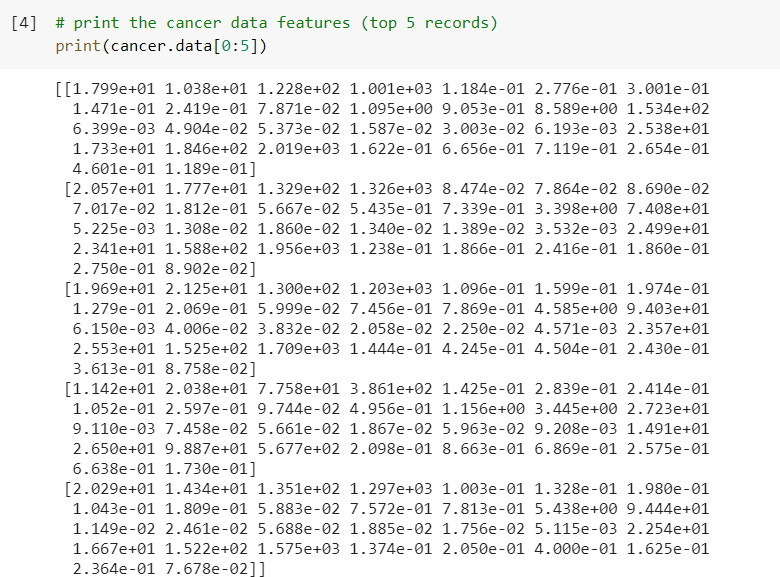
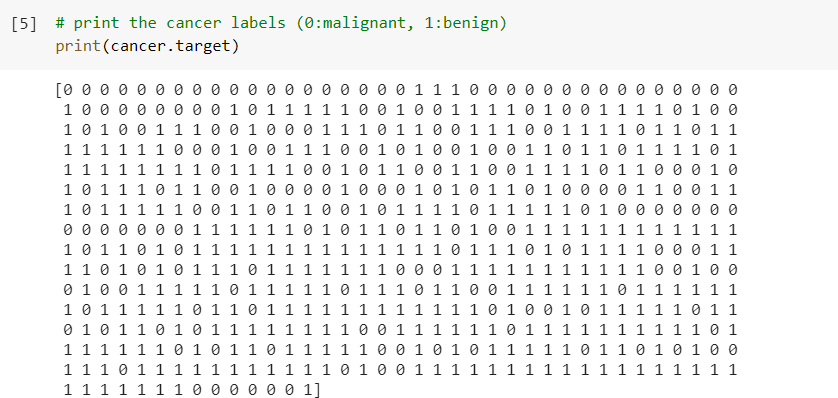
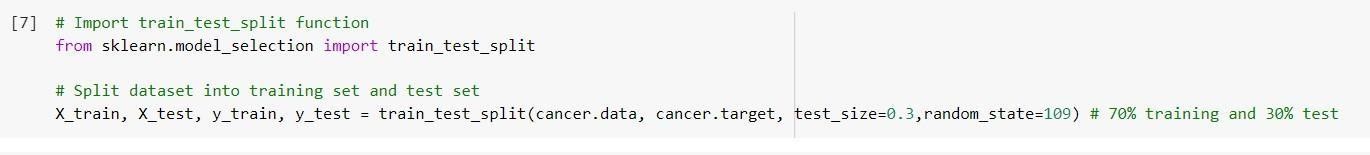
**Exploring data:**

****

Check the shape of the dataset using shape.



Check top 5 records of the feature set.



**Target set:**

**Splitting Data:**

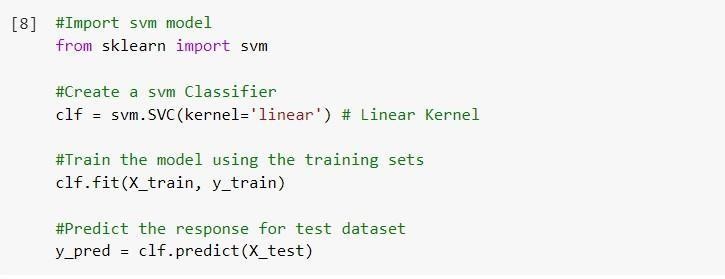
To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Split the dataset by using the function train\_test\_split(). you need to pass 3 parameters features, target, and test\_set size. Additionally, you can use random\_state to select records randomly.

**Generate Model:**

Let's build support vector machine model. First, import the SVM module and create support vector classifier object by passing argument kernel as the linear kernel in SVC() function.

Then, fit your model on train set using fit() and perform prediction on the test set using predict().



**Evaluating the Model:**

Let's estimate how accurately the classifier or model can predict the breast cancer of patients. Accuracy can be computed by comparing actual test set values and predicted values.

