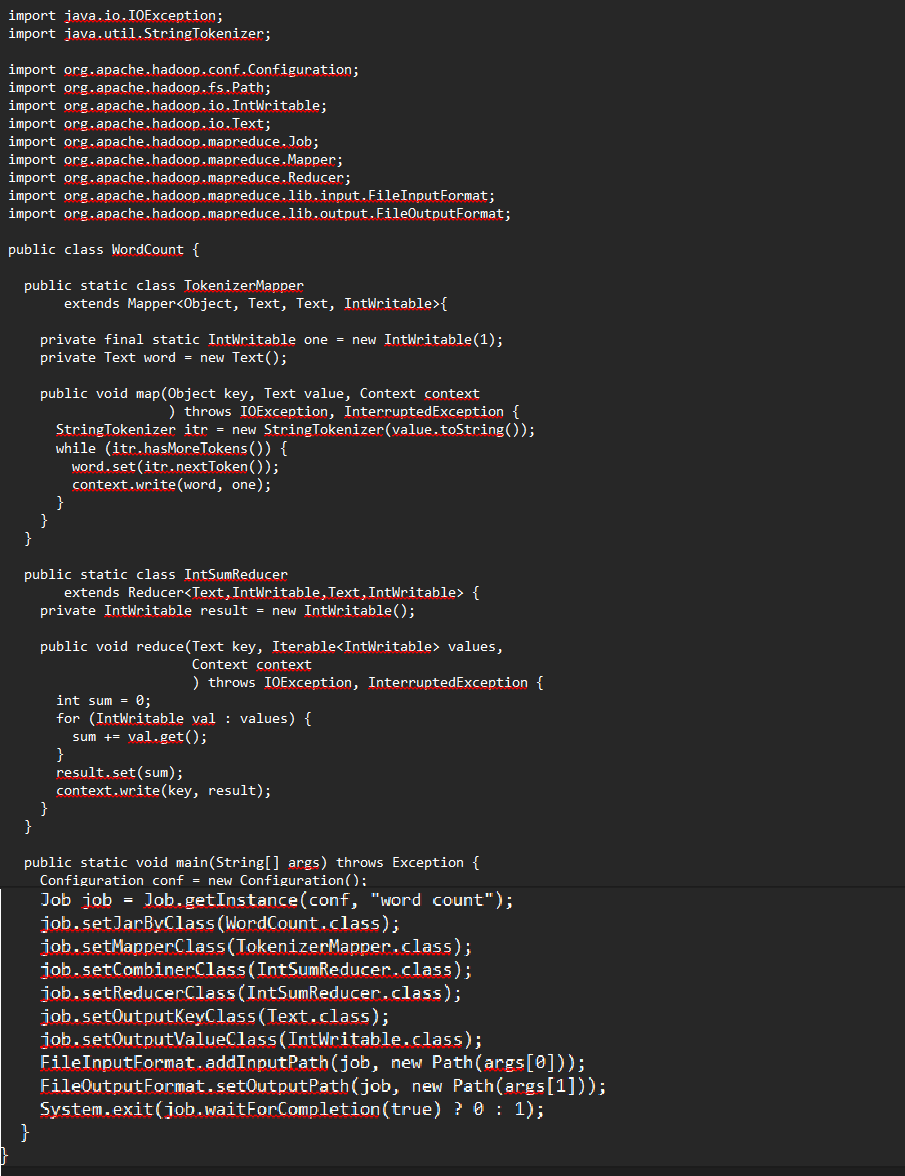
1. How do you ingest unstructured data using Flume?
2. What are the different Flume data flow channels?

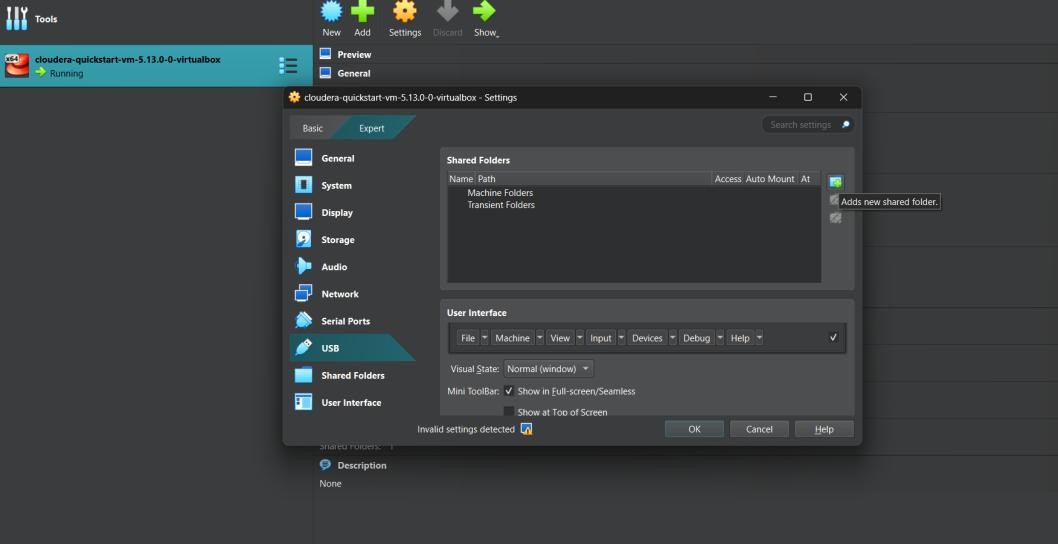
**Task-3:**

1. **Write a word count program to simulate the Map Reduce Paradigm.**

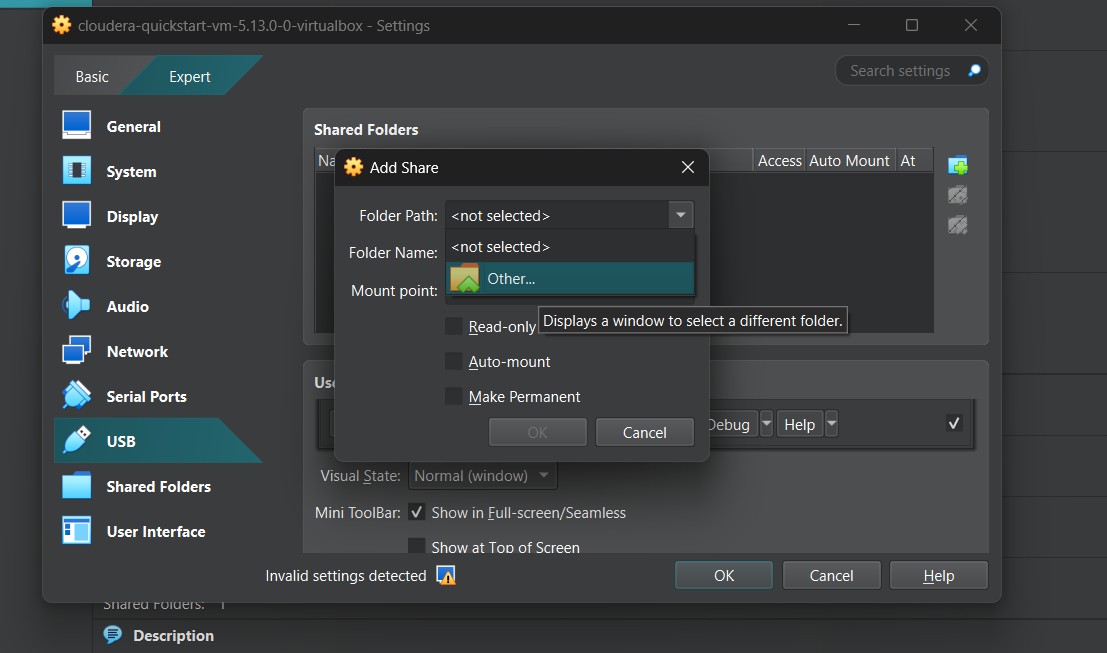
Step-1 : Create a folder “WC‟ on Desktop with wordcount.txt file which includes the java code for counting words.



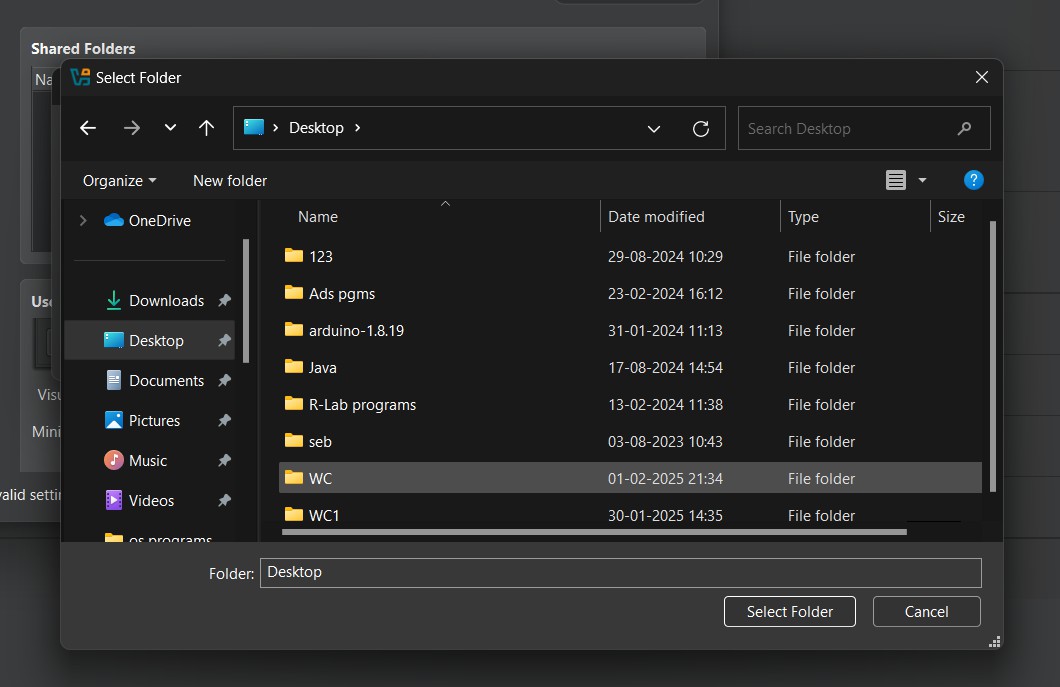
Step-2 : Open Oracle Virtual Box Manager-> click on Settings -> click on Shared Folders -> Select Machine Folders -> click on + symbol on right panel.



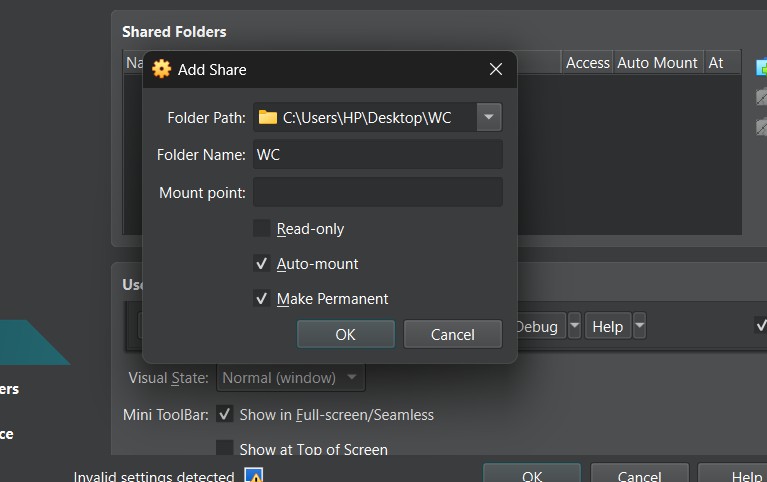
Step-3 : Select Folder path click on drop down in which “Other” option will be displayed.



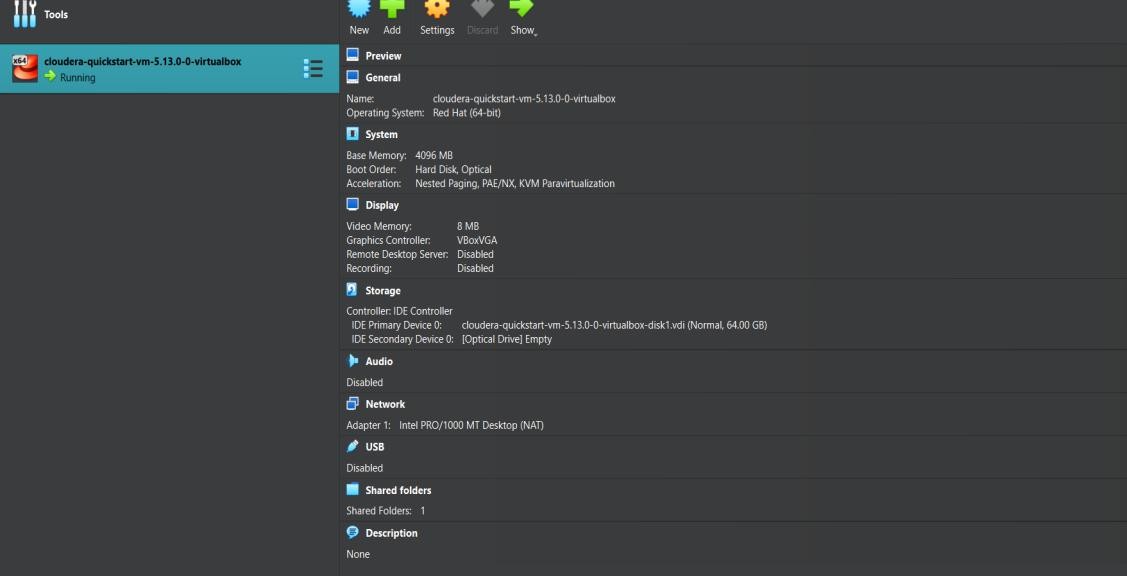
Step-4 : Now navigate to the path where the the folder “WC” mentioned in step-1.



Step-5: Select the folder WC and select checkboxes “Auto-mount” and “Make Permanent” -> click on OK -> click on OK.



Step-5: Now in Shared Folders , we can see “1” shared folder is added.

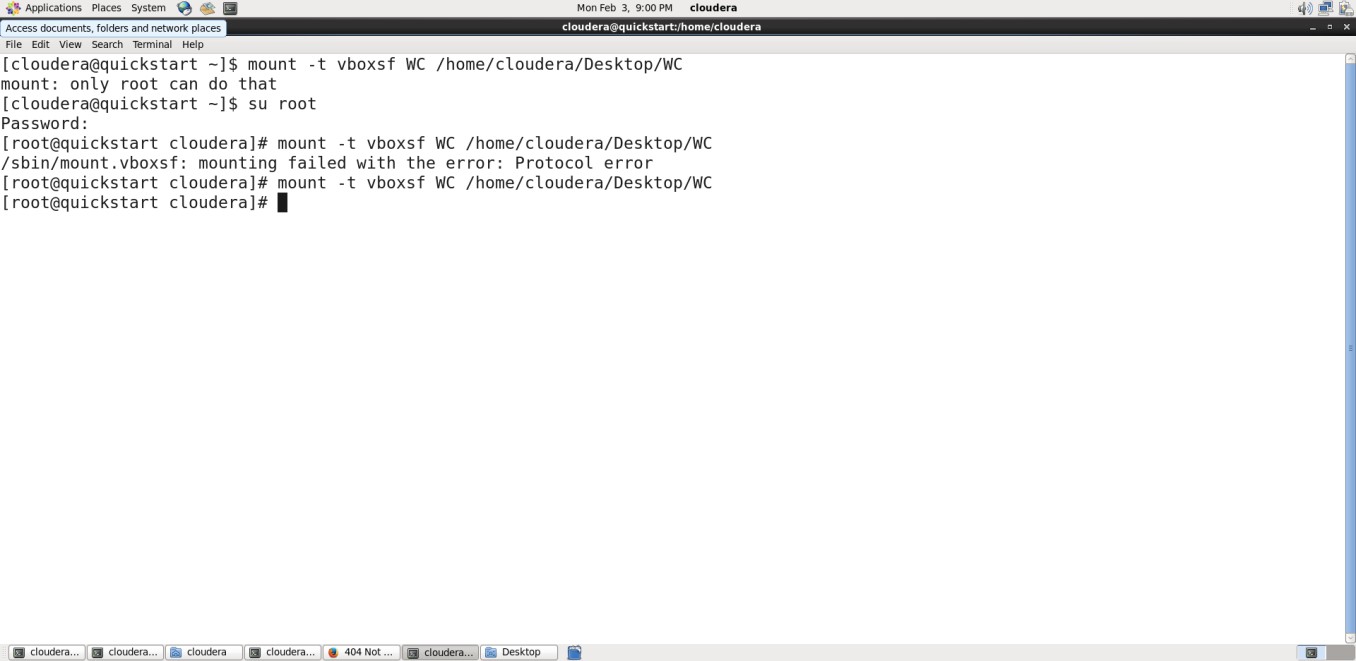


Step-6: Click on Cloudera terminal. Type su root and give password as cloudera.

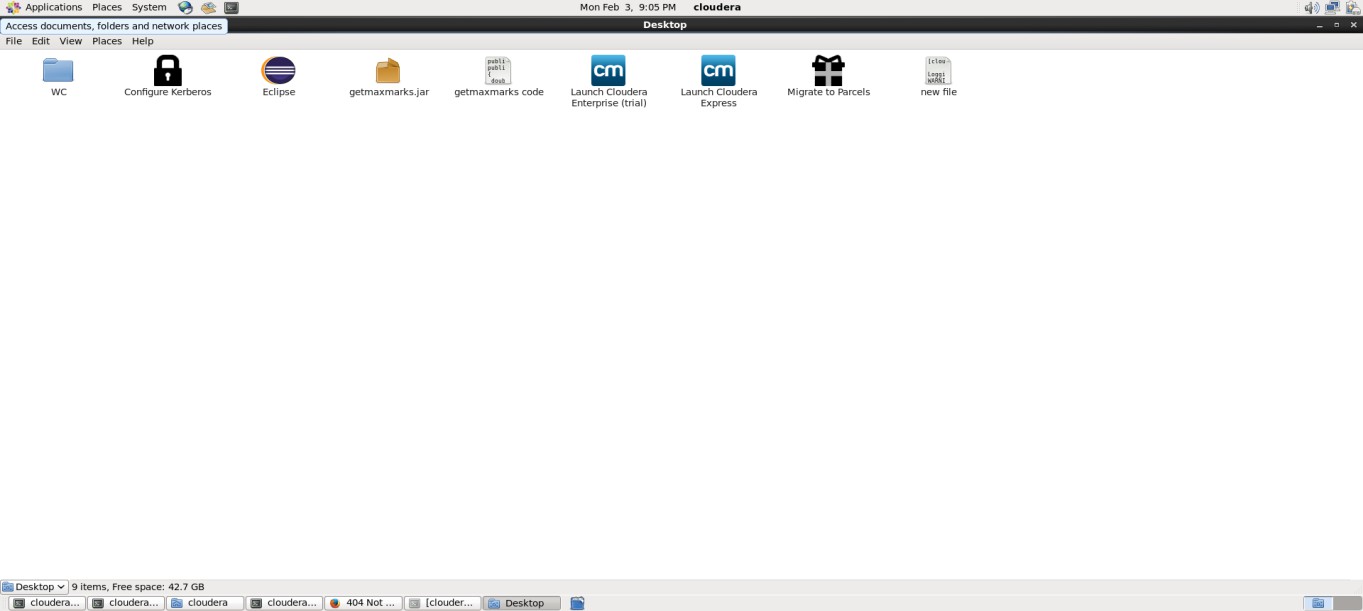
$ su root

Password : cloudera

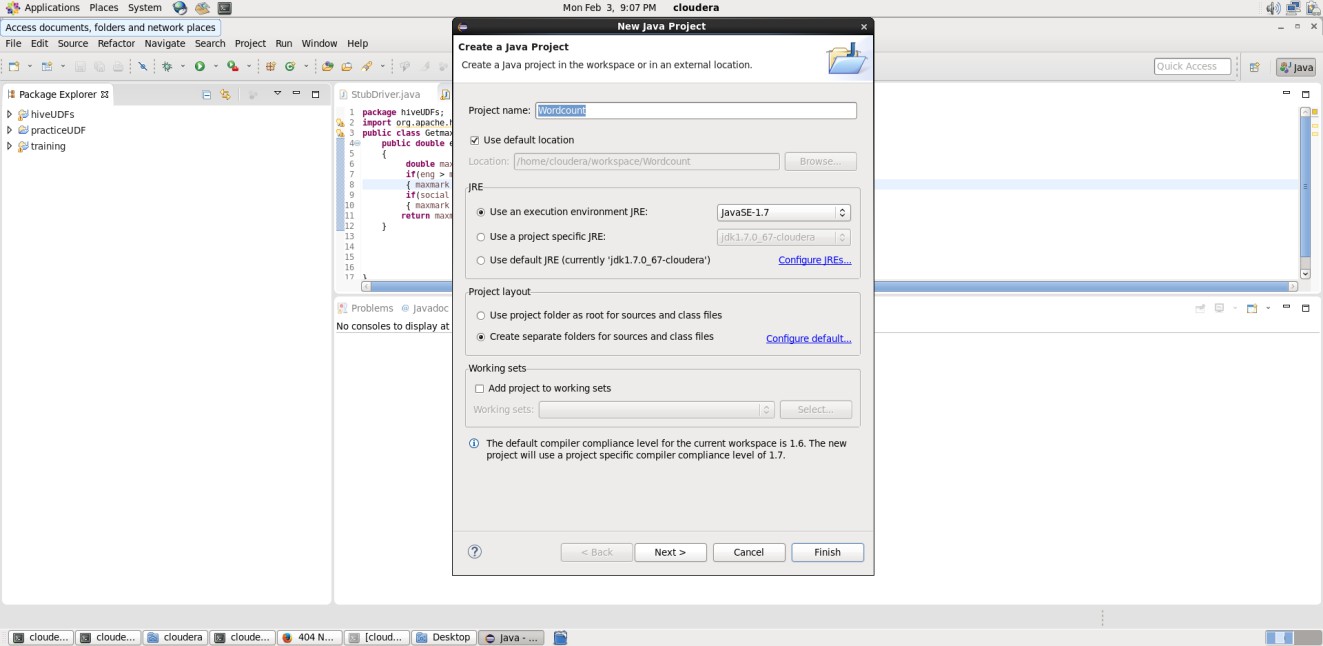
Now type mount -t vboxsf WC /home/cloudera/Desktop/WC



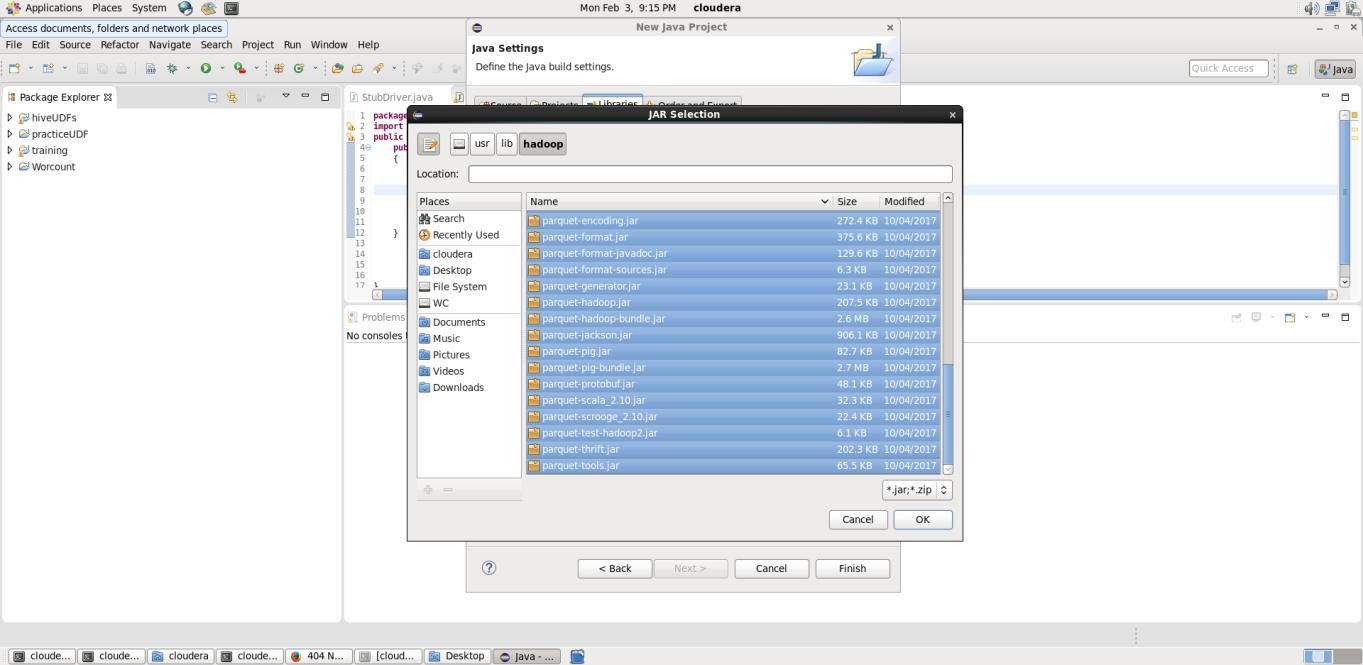
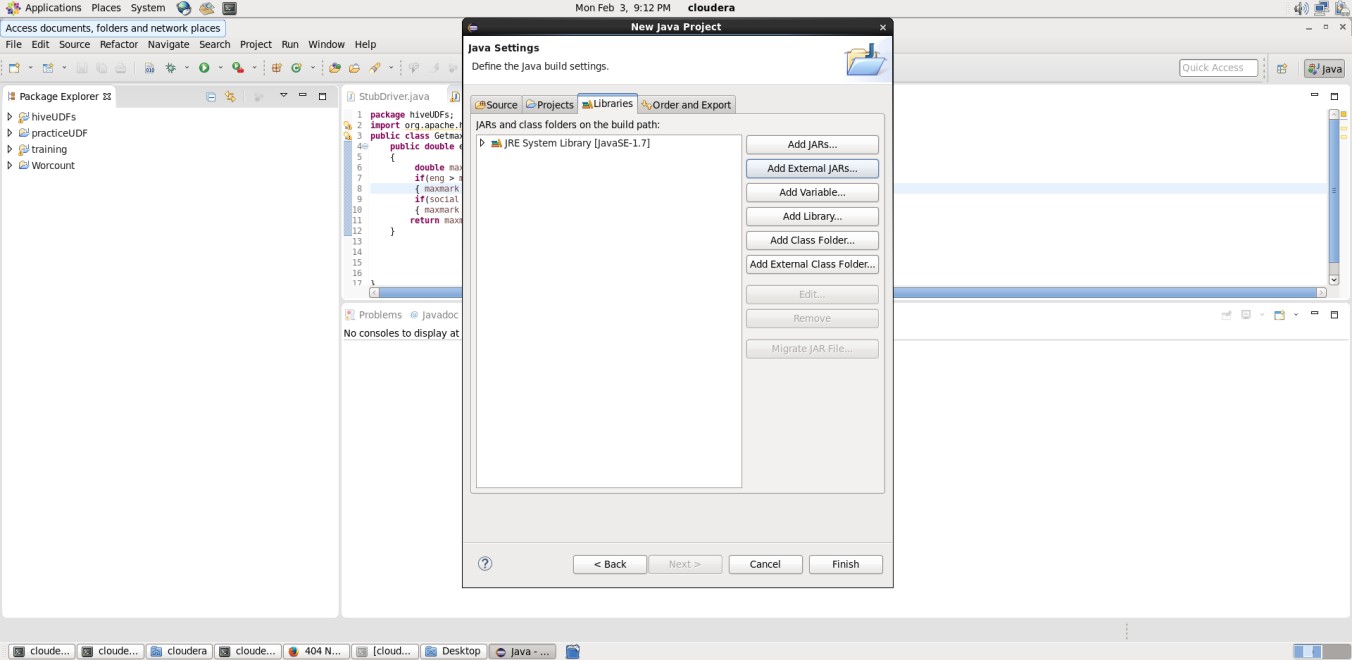
Step-7: Data from windows C:\Desktop\WC will be copied into cloudera desktop WC folder.



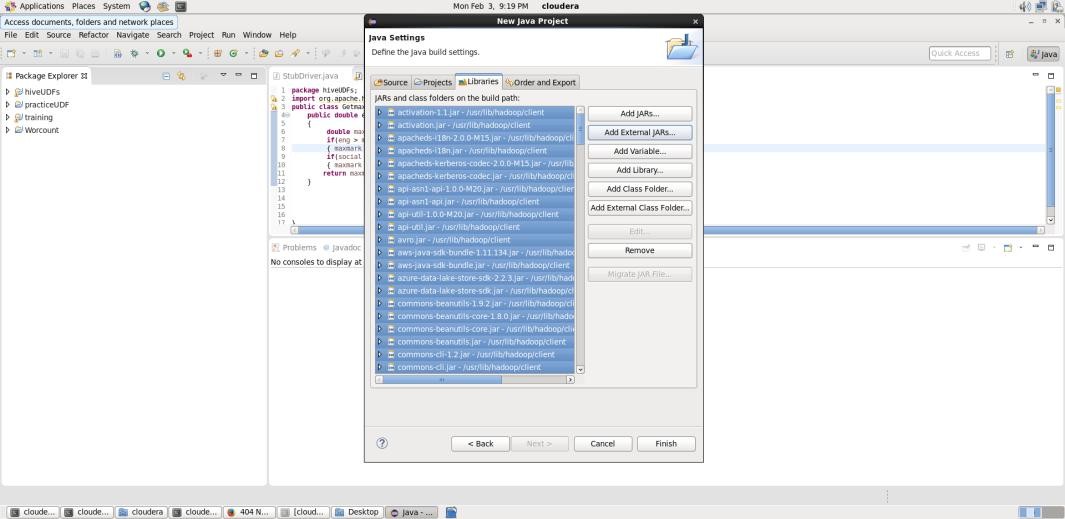
Step-8: Open eclipse in cloudera. Click on File -> New -> Java Project -> type Wordount -> click on Next button.



Step-9: Click on Libraries -> Add External JARs -> select File System -> usr -> lib -> hadoop -> select all Jar files -> click on OK.

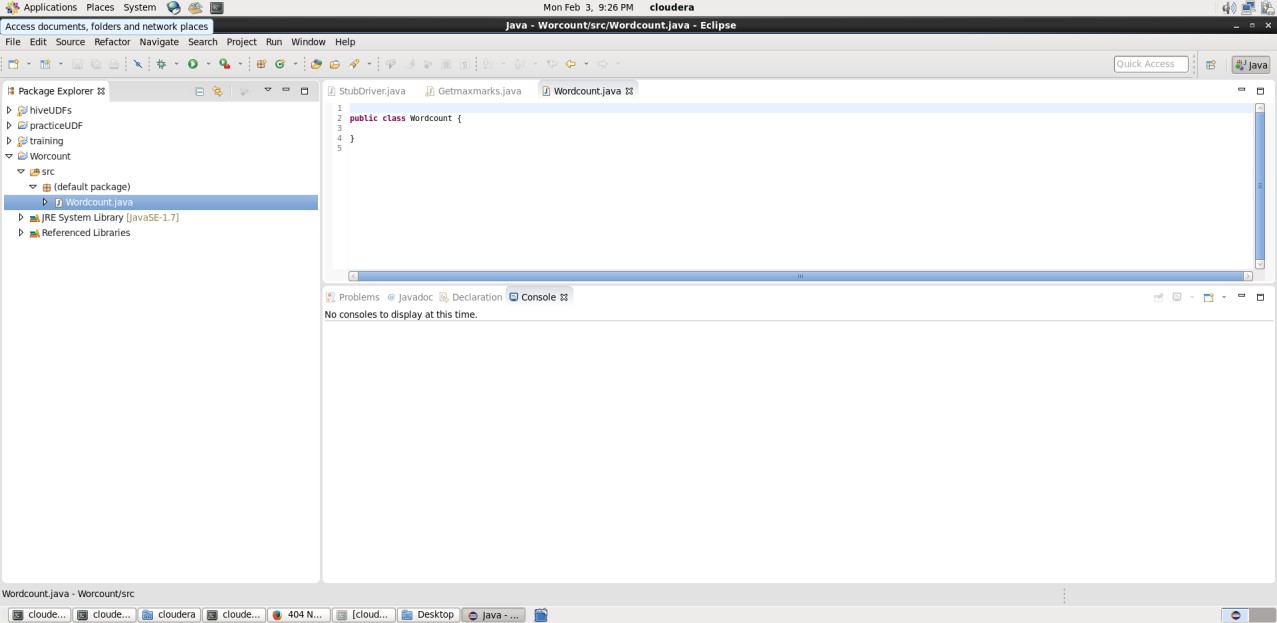


Step-10: Click on Add external JARs-> click on client -> select all Jar files -> click on OK -> click on Finish.

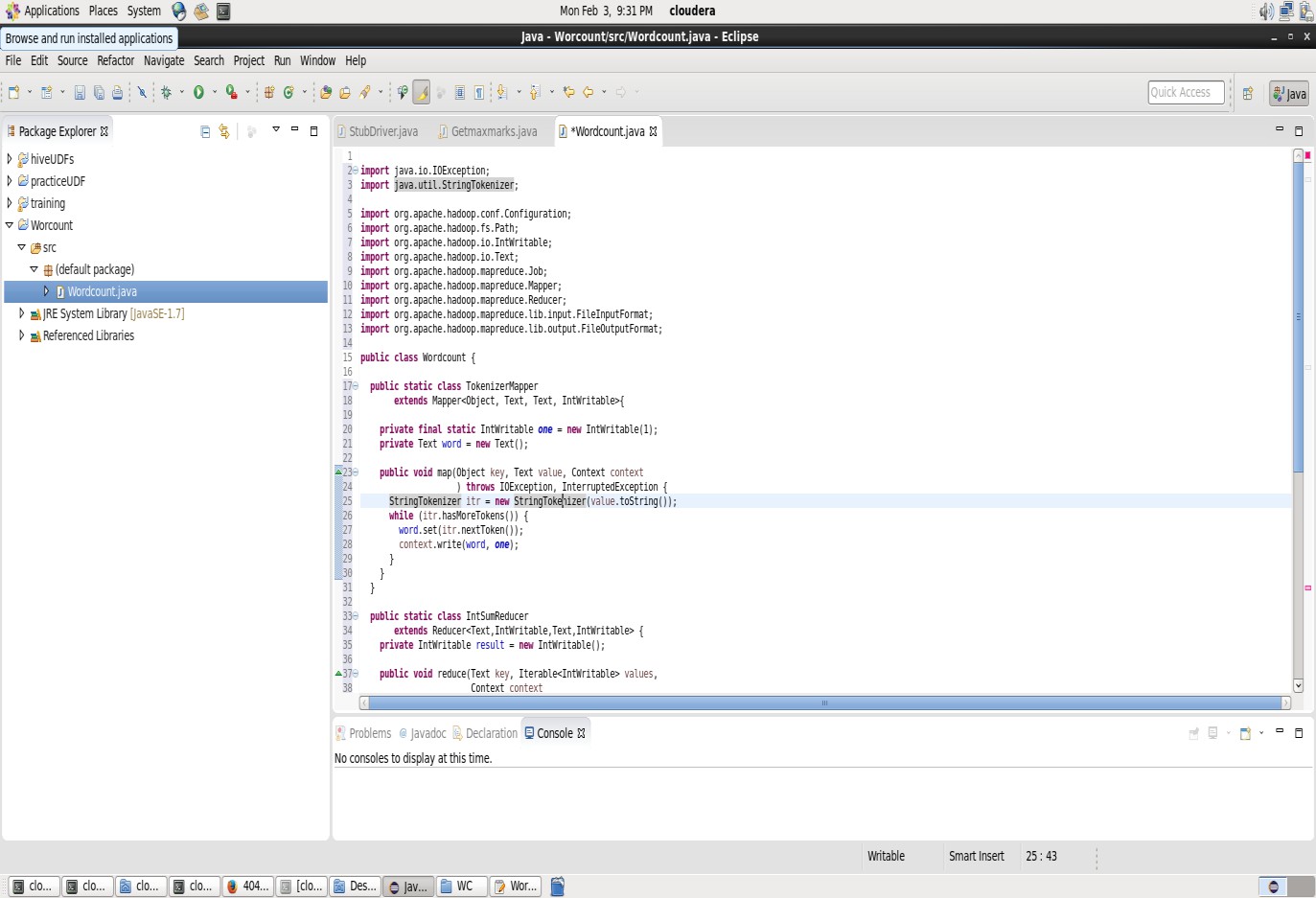


Step-11: -> Now Click on project Wordcount -> src -> right click on New -> class -> type the classname as “Wordcount” -> click on Finish.

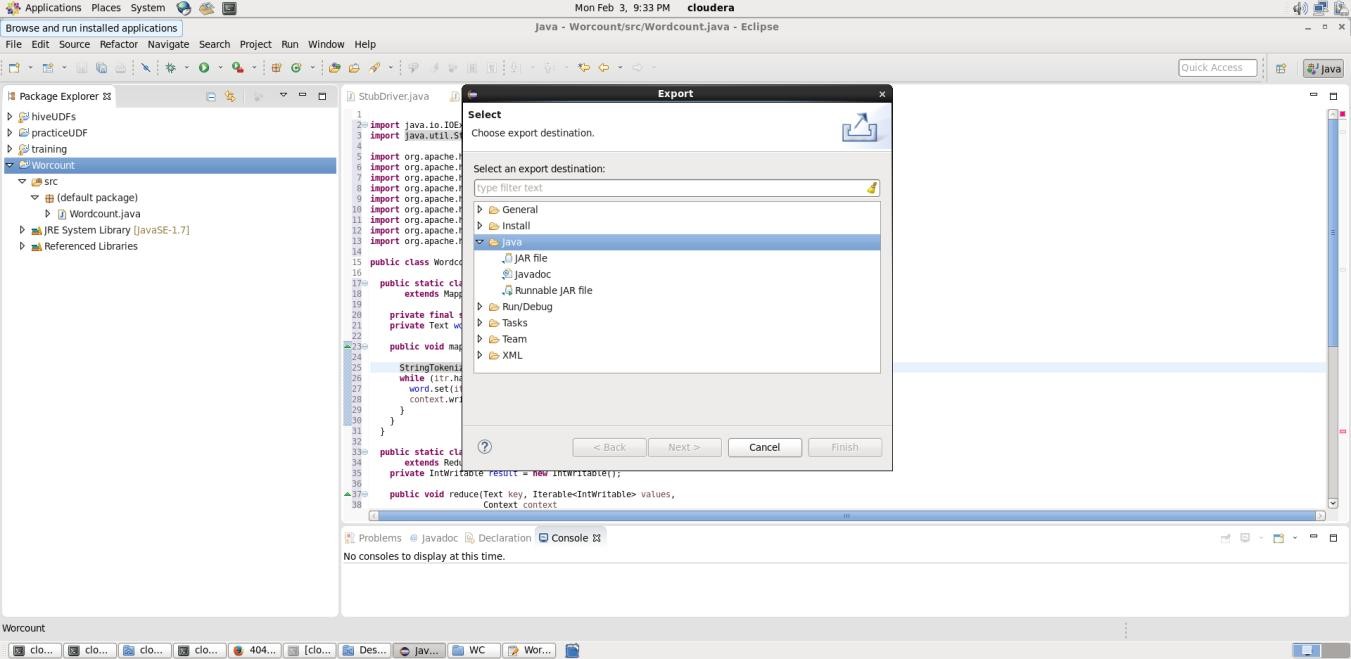
Wordcount.java class file will be created as shown.

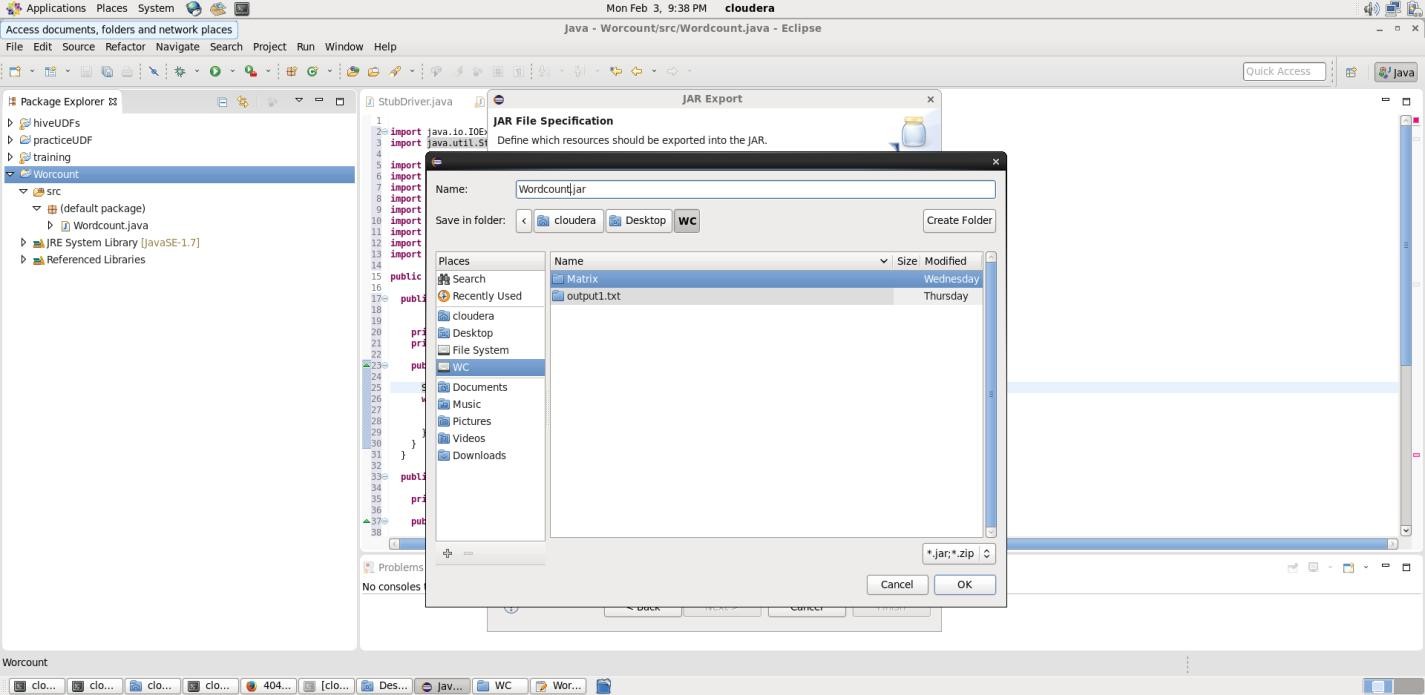


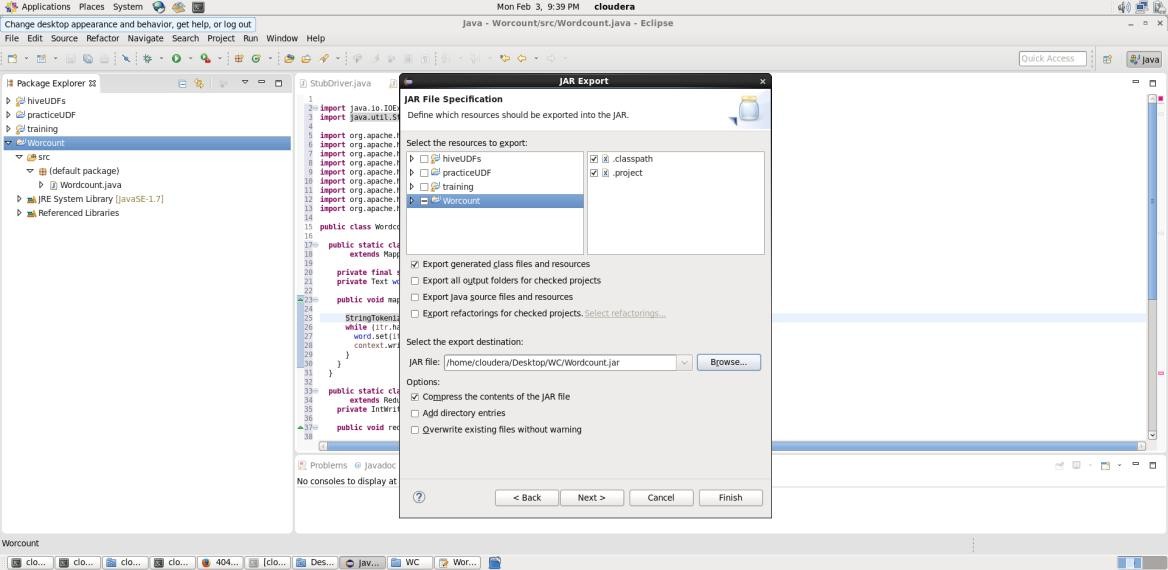
Step-12: In Wordcount.java class file , delete the contents. Copy the java code from Cloudera/Desktop/WC/wordcount.txt and paste into this class file.



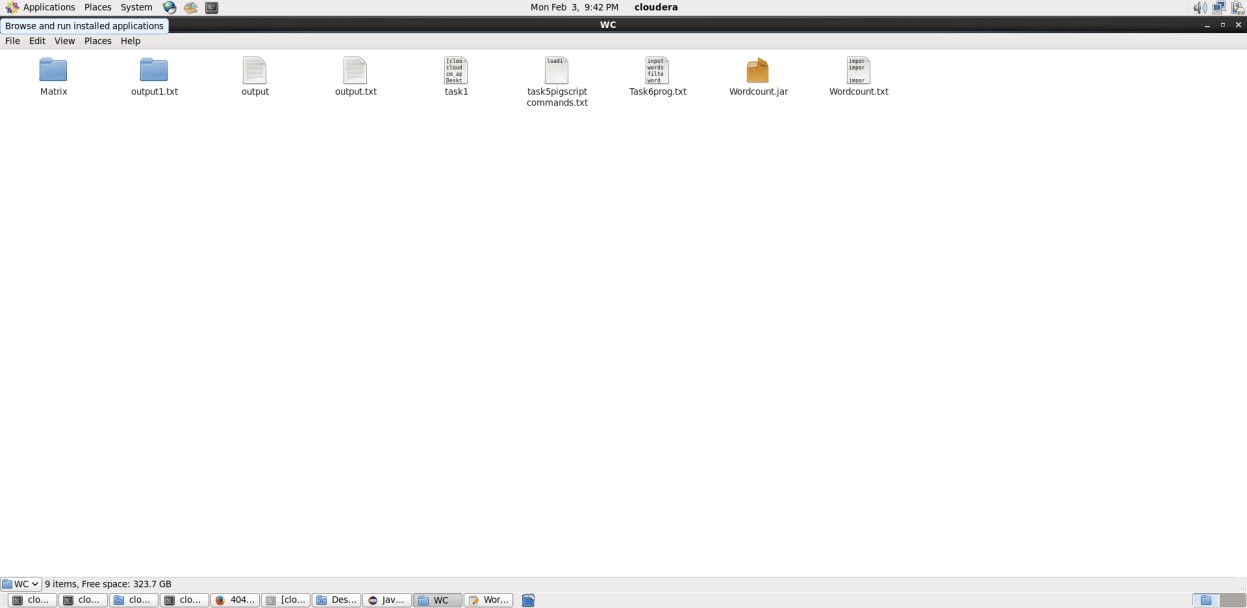
Step-13: -> Right click on Wordcount -> Export -> select Java -> JAR file -> click on „classpath‟ and „project‟ checkboxes -> select the export destination -> browse -> cloudera -> Desktop -> WC -> type the name of the jar file-> click on Finish.







Wordcount.Jar file will be displayed in cloudera/Desktop/WC folder



Step-14 : Click on cloudera terminal and type the following commands:

1. **$ cat > / home/cloudera/process1.txt (to create input file) big data analytics**

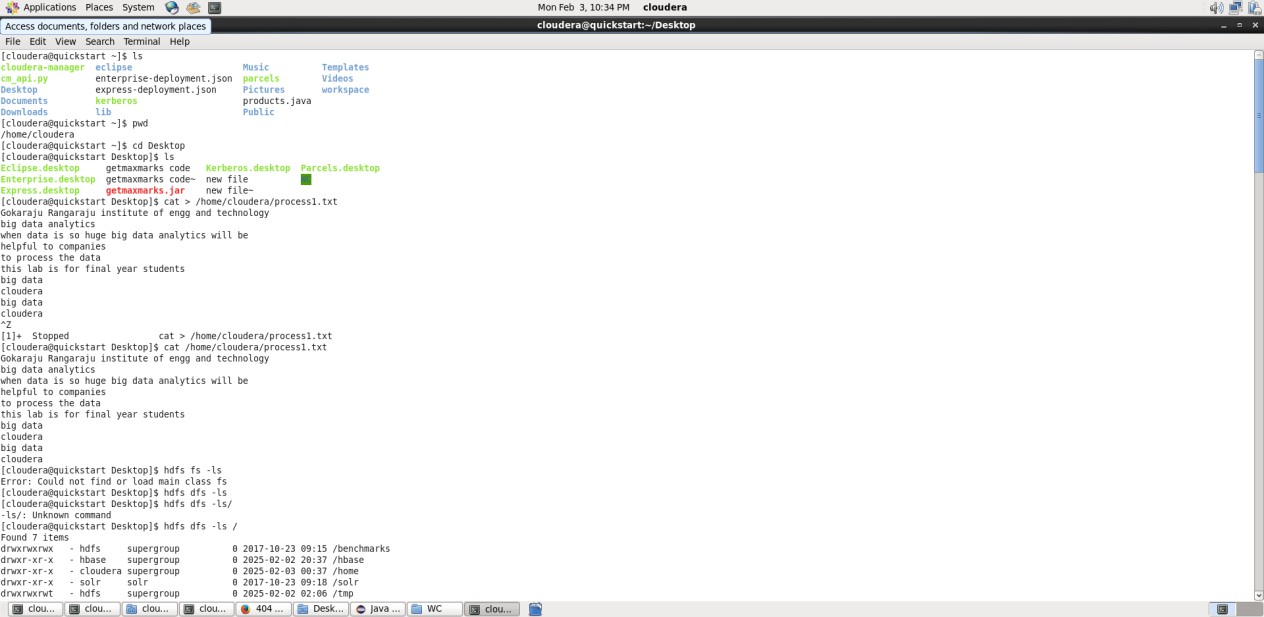
When data is so huge big data analytics will be Helpful to companies

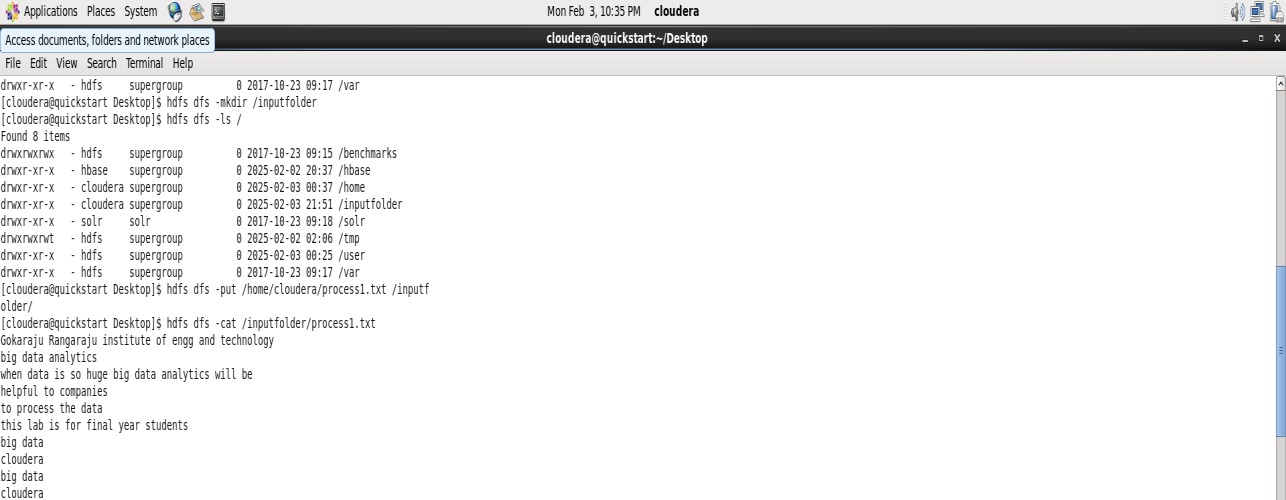
To process the data

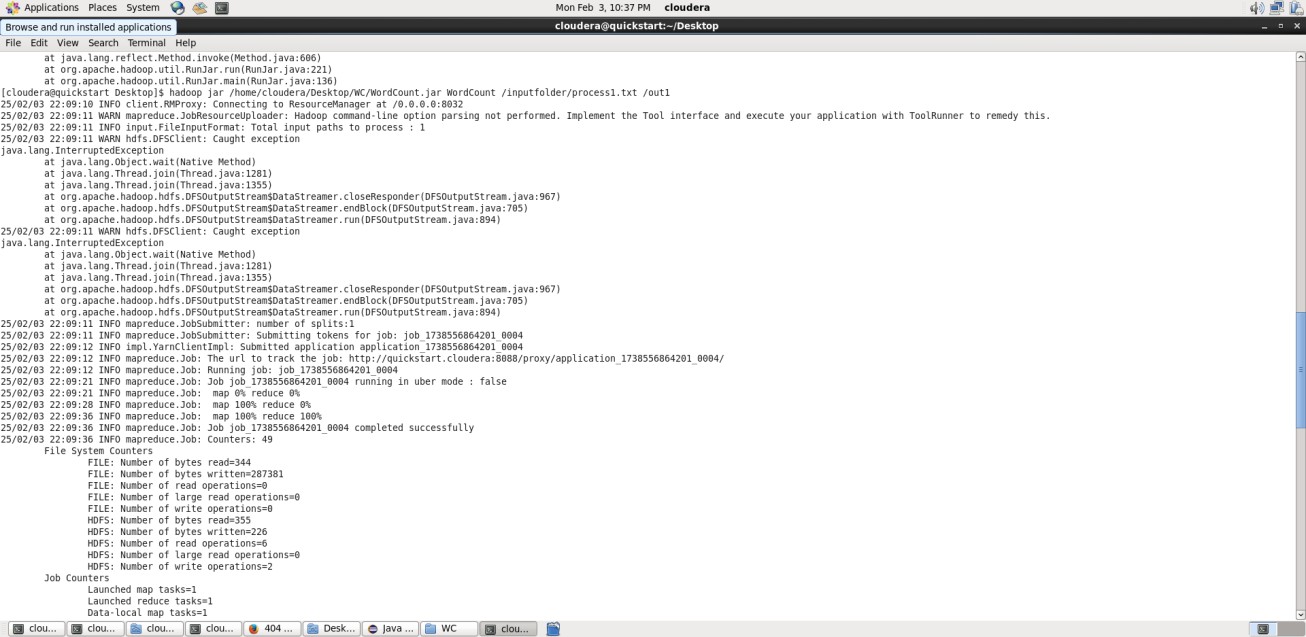
This lab is for final year students Big data

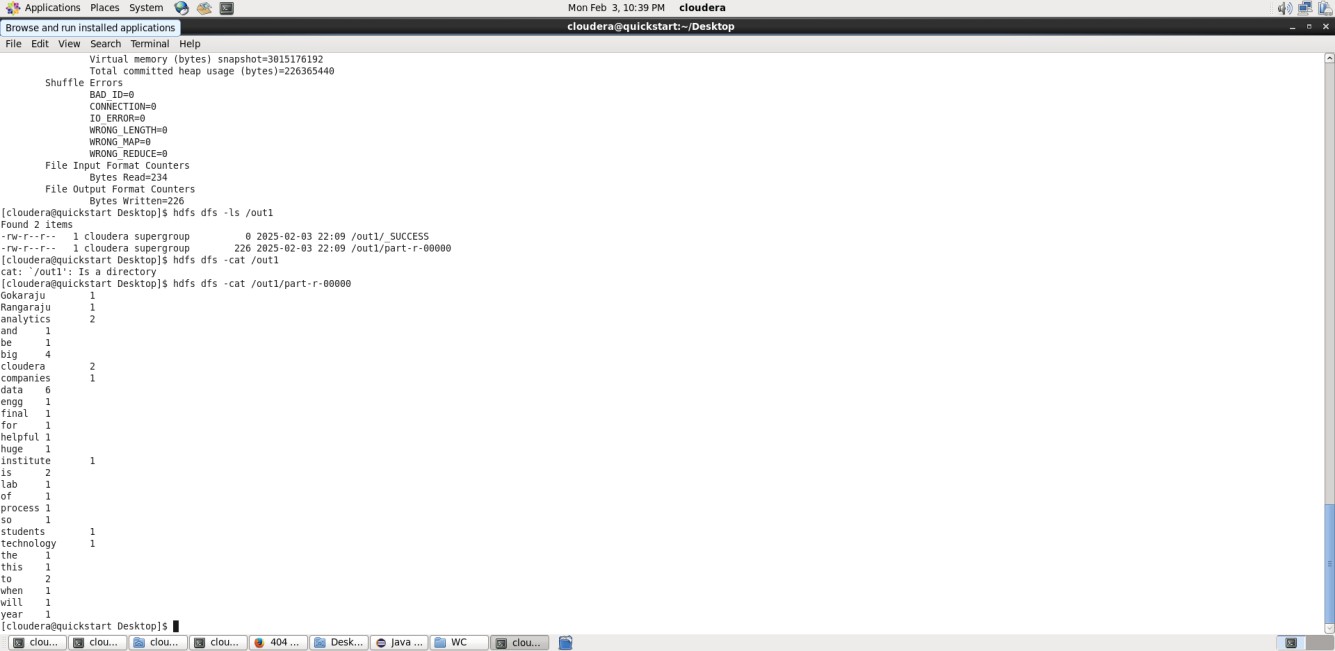
Cloudera Big data cloudera Type ctrl+z

1. **$ cat /home/cloudera/process1.txt displays the contents of the file.**
2. **$ hdfs dfs -ls /**
3. **$ hdfs dfs -put /home/cloudera/process1.txt /inputfolder/**
4. **$ hdfs dfs -cat/inputfolder/process1.txt**
5. **$ hadoop jar /home/cloudera/Desktop/WC/WordCount.jar Wordcount /inputfolder/process1.txt/out1**
6. **$ hdfs dfs -ls /out1**
7. **$ hdfs dfs -cat /out1/part-r-00000**









1. **Write a Map Reduce program to perform matrix multiplication.**

Matrix multiplication using Hadoop MapReduce involves splitting the matrix into key-value pairs, processing them in the mapper, and combining them in the reducer.

Steps for MapReduce

* + Mapper: Emits key-value pairs as (i, j) → partial product.
  + Reducer: Sums up the partial products to compute C(i, j).

Step-1: Create a Java Project in Eclipse

Open Eclipse → New Java Project → Name: MatrixMultiplication -> Click on Next -> Click on Libraries -> Add External JARs -> select File System -> usr -> lib -> hadoop -> select all Jar files -> click on OK.

Click on Libraires -> Add external JARs-> click on client -> select all Jar files -> click on OK -> click on Finish.

Step-2 :Now Click on project MatrixMultiplication -> src -> right click on

1. **New -> class -> type the classname as “MatrixMapper” -> click on Finish.**
2. **New -> class -> type the classname as “MatrixReducer” -> click on Finish.**
3. **New -> class -> type the classname as “MatrixDriver” -> click on Finish.**

Step-3: Click on MatrixMapper.Java file and include the following java code.

import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper; import java.io.IOException;

public class MatrixMapper extends Mapper<Object, Text, Text, IntWritable> {

public void map(Object key, Text value, Context context) throws IOException, InterruptedException { String[] tokens = value.toString().split(",");

int i = Integer.parseInt(tokens[0]); // Row index

int j = Integer.parseInt(tokens[1]); // Column index int val = Integer.parseInt(tokens[2]); // Matrix value

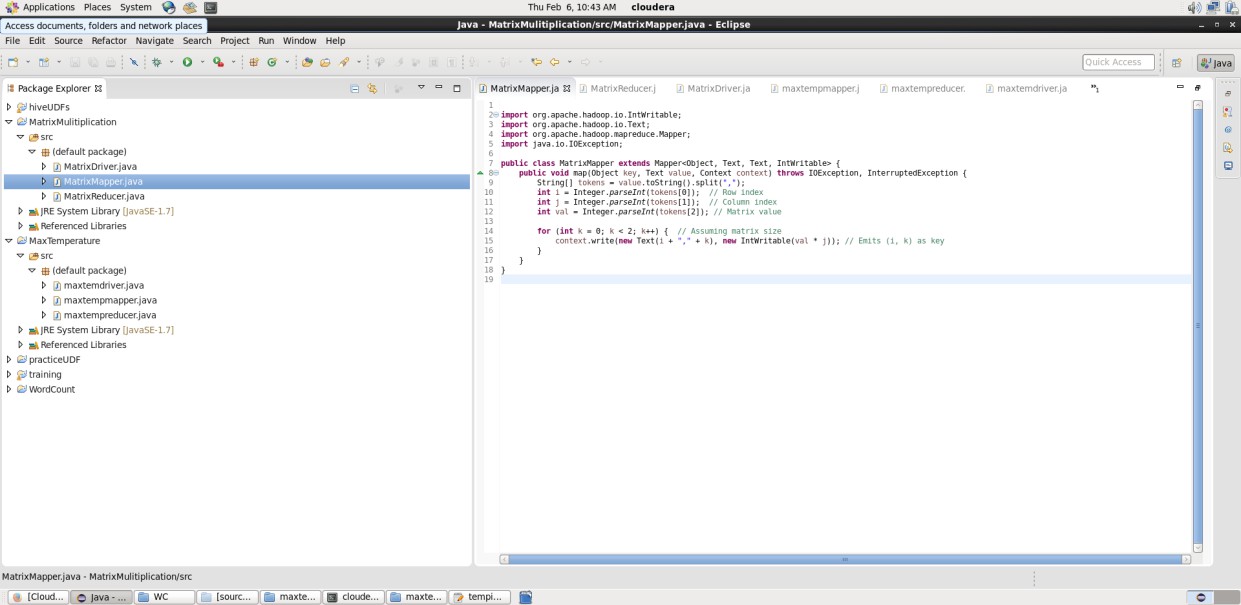
for (int k = 0; k < 2; k++) { // Assuming matrix size

context.write(new Text(i + "," + k), new IntWritable(val \* j)); // Emits (i, k) as key

}

}

}



Step-4: Click on MatrixReducer.Java file and include the following java code.

import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer; import java.io.IOException;

public class MatrixReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

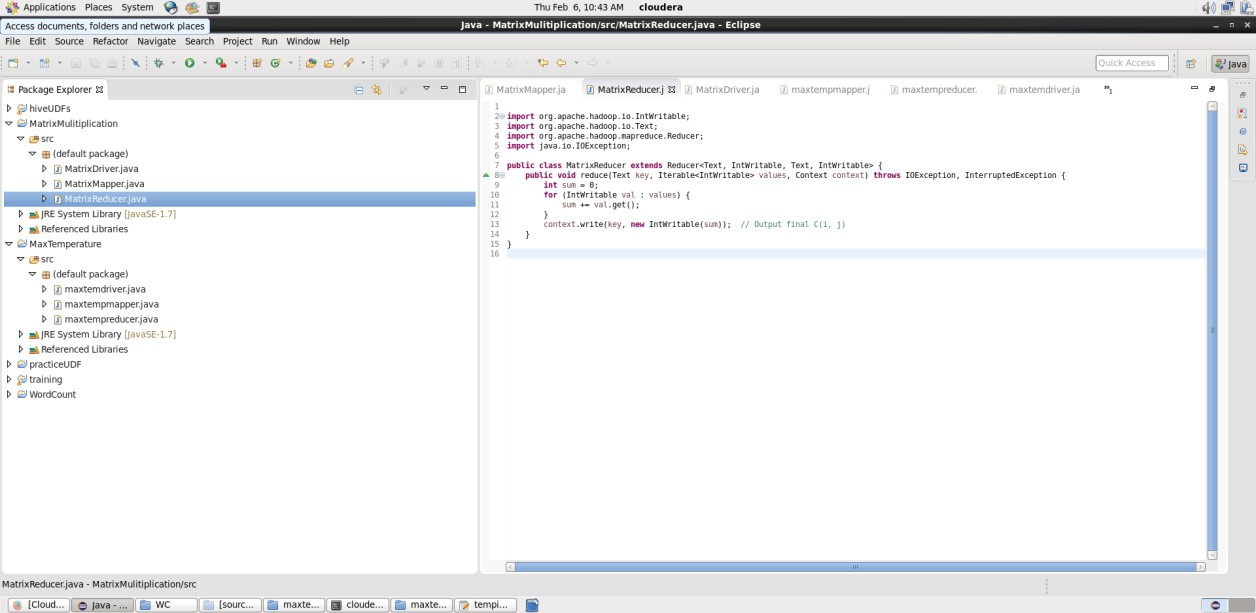
for (IntWritable val : values) { sum += val.get();

}

context.write(key, new IntWritable(sum)); // Output final C(i, j)

}

}



Step-5: Click on MatrixDriver.Java file and include the following java code.

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

public class MatrixDriver {

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

Job job = Job.getInstance(conf, "MatrixMultiplication"); // project name

job.setJarByClass(MatrixDriver.class); job.setMapperClass(MatrixMapper.class); job.setReducerClass(MatrixReducer.class);

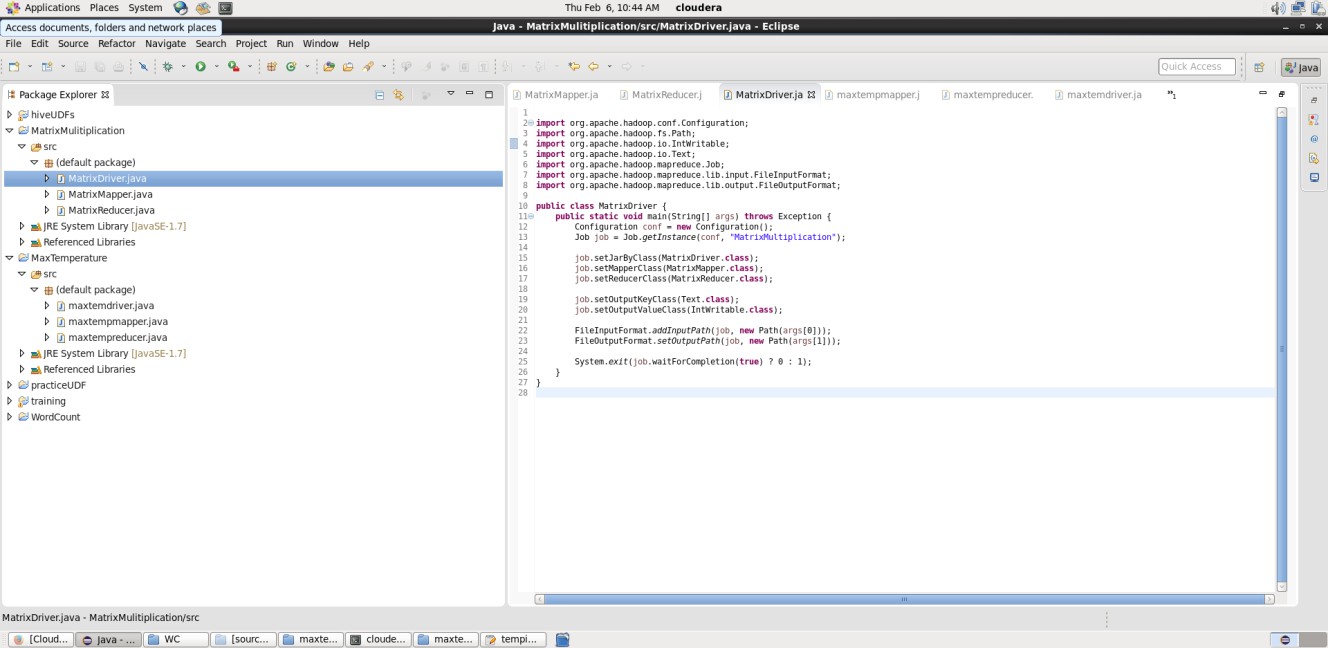
job.setOutputKeyClass(Text.class); job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0])); FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}



**Step-6: Prepare the input data** Input Matrix A (matrixA.txt) 0,0,1

0,1,2

1,0,3

1,1,4

Input Matrix B (matrixB.txt) 0,0,5

0,1,6

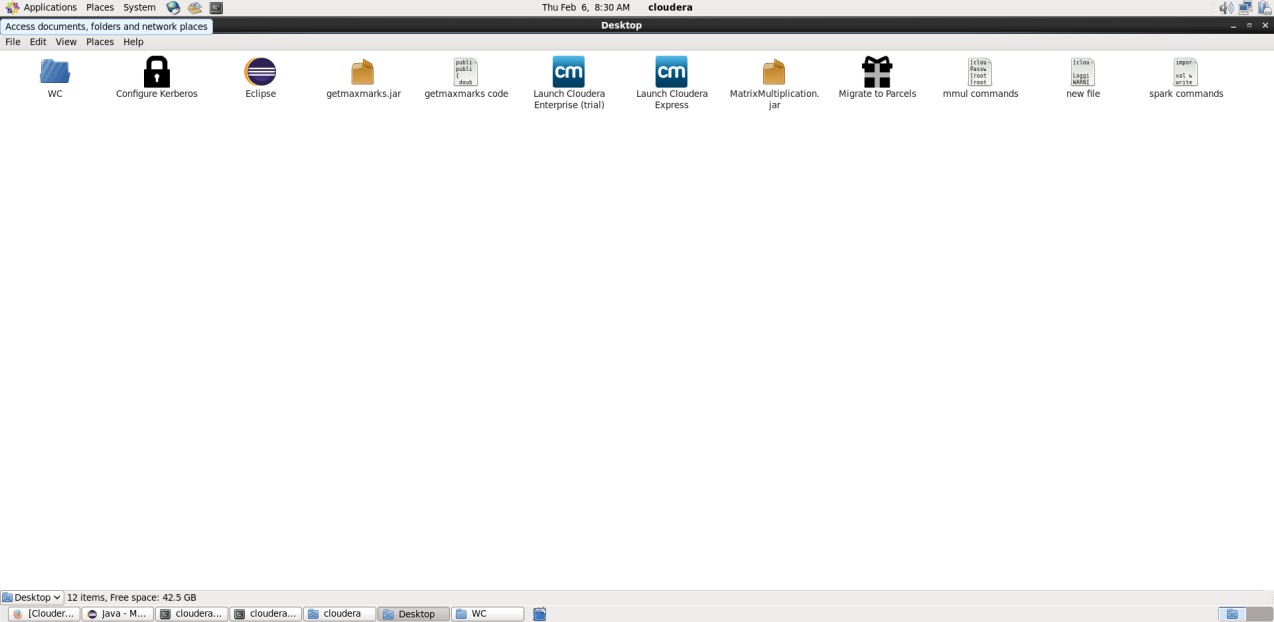
1,0,7

1,1,8

Store these two files in /home/cloudera/Desktop/WC/MatrixMul folder.

Step-7: In Eclipse, Right click on MatrixMultiplication -> Export -> select Java -> JAR file -> click on „classpath‟ and „project‟ checkboxes -> select the export destination -> browse -> cloudera -> Desktop -> type the name of the jar file as “MatrixMultiplication”-> click on Finish.

The Jar file will be stored in /home/cloudera/Desktop folder.



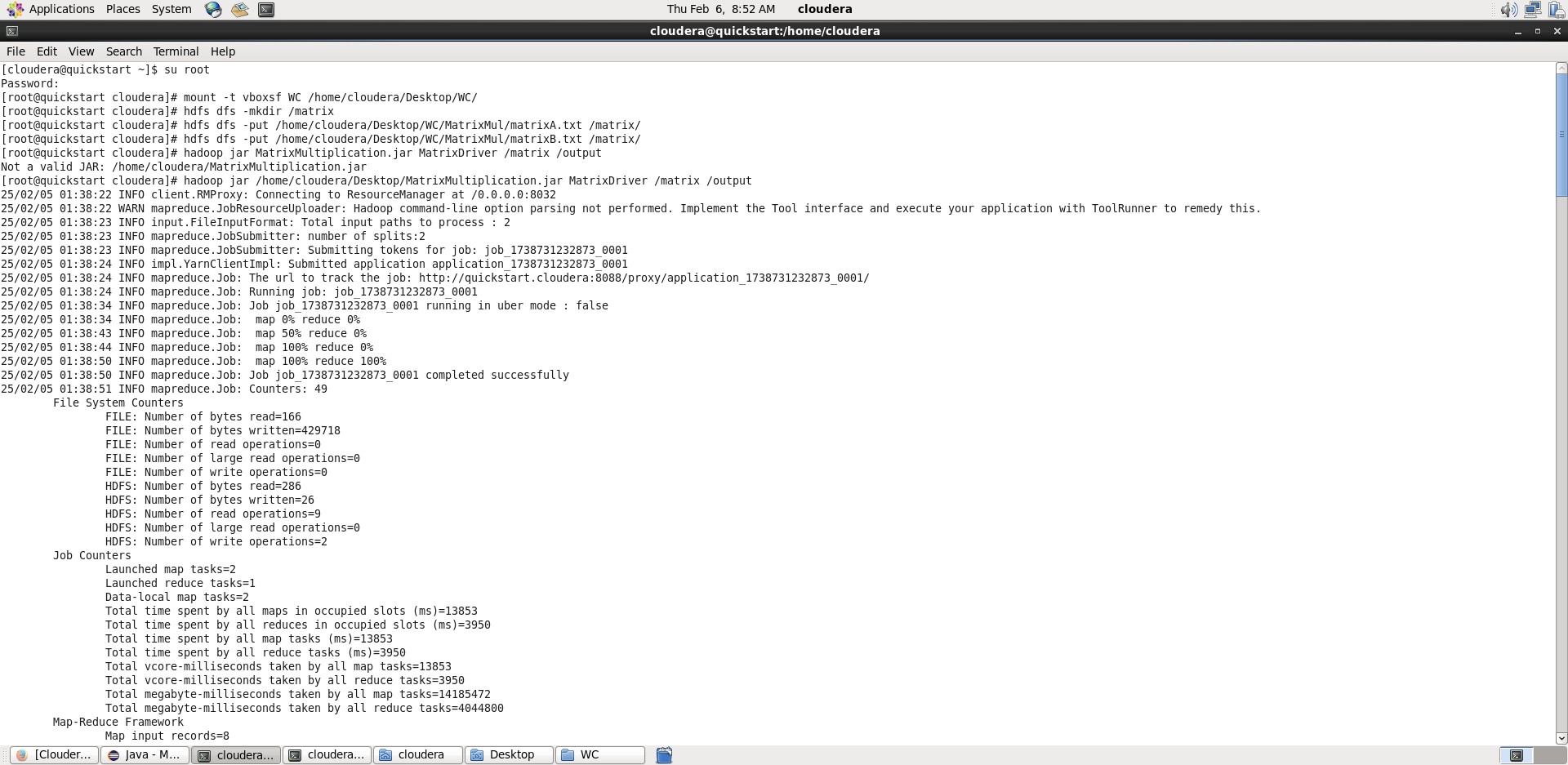
Step-8: Upload Input Data with the following commands.

hdfs dfs -mkdir /matrix

hdfs dfs -put /home/cloudera/Desktop/WC/MatrixMul/matrixA.txt /matrix/ hdfs dfs -put /home/cloudera/Desktop/WC/MatrixMul/matrixB.txt /matrix/

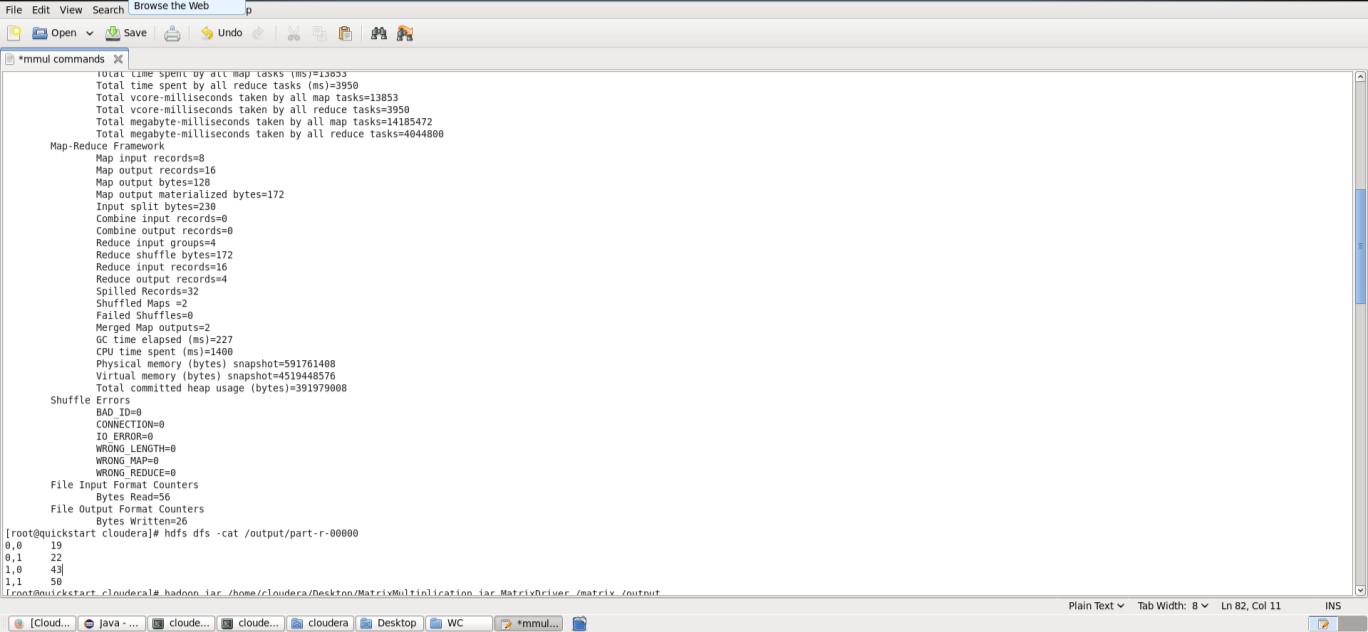
Step-9: Compile and Run

hadoop jar /home/cloudera/Desktop/matrixmultiplication.jar MatrixDriver /matrix /output



Step-10: Check Output

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



Viva Questions:

1. **Word Count Program:**
   1. What is the MapReduce paradigm?
   2. What is the role of the Mapper and the Reducer?
   3. Why is the Context.write() method used in MapReduce?
   4. How is data shuffled and sorted between the map and reduce phases?
   5. What happens if a reducer is not defined?
2. **Matrix Multiplication Program:**
   1. How do you represent a matrix in a MapReduce program?
   2. What are the input and output formats for a matrix multiplication job?
   3. How do you optimize a MapReduce job for matrix multiplication?
   4. Explain how you manage large matrix datasets in HDFS.
   5. What are the challenges in implementing matrix multiplication in MapReduce?