Maharishi University of Management

CS522 – Big Data

Prof. Premchand Nair

Set Up a Single Node Cluster

January 30th, 2017

Author	ID	Date
Sovichea Cheth	985421	01/30/2017

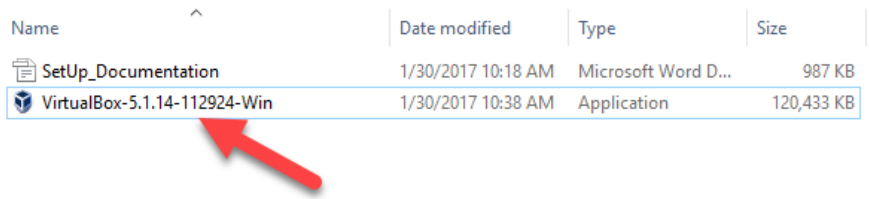
In this manual, we're going to see how to set up a single node cluster and optionally an eclipse development environment to create and test your programs on Microsoft Windows.

I. Install Oracle Virtual Box

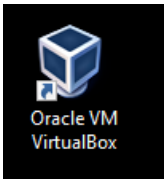
First, go to https://www.virtualbox.org/wiki/Downloads

Under VirtualBox 5.1.14 platform packages, click on Windows hosts to download setup file for Windows.

Then after download, double click on setup file to perform the VirtualBox installation



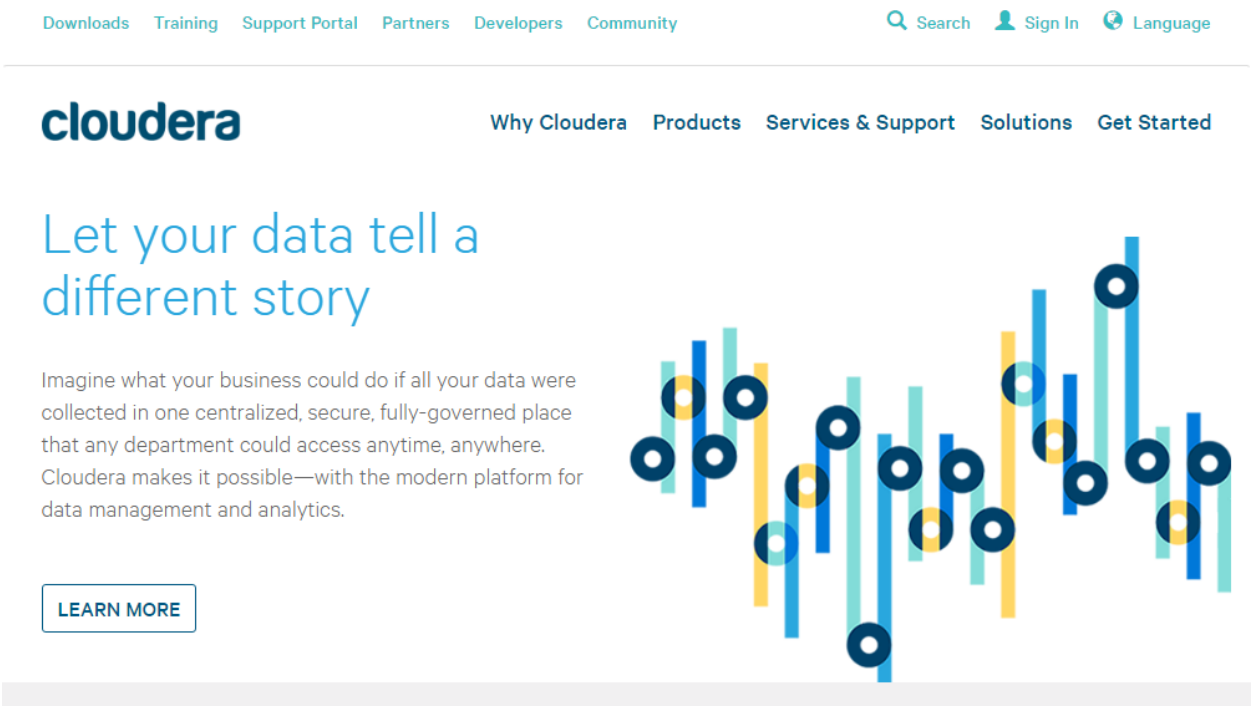
After finished installation, you will able to run the program.



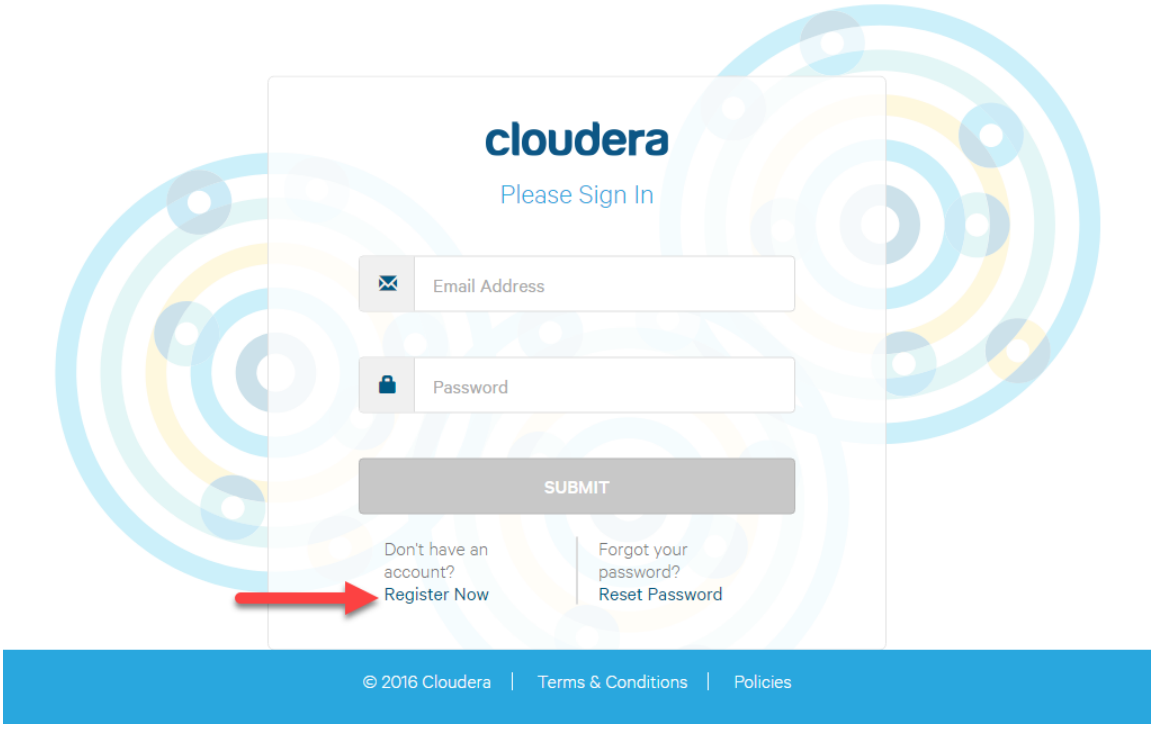
II. Download & Install Cloudera

a. Get Cloudera

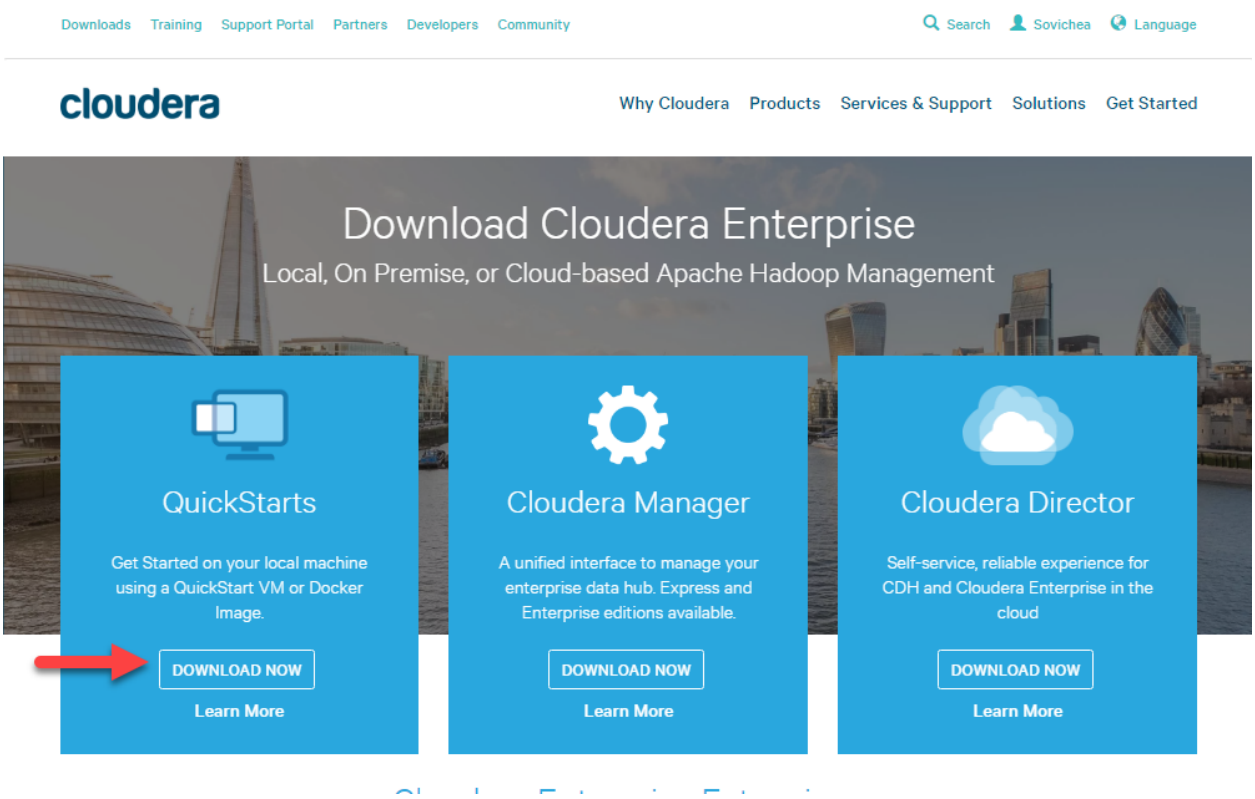
First, go to http://www.cloudera.com/



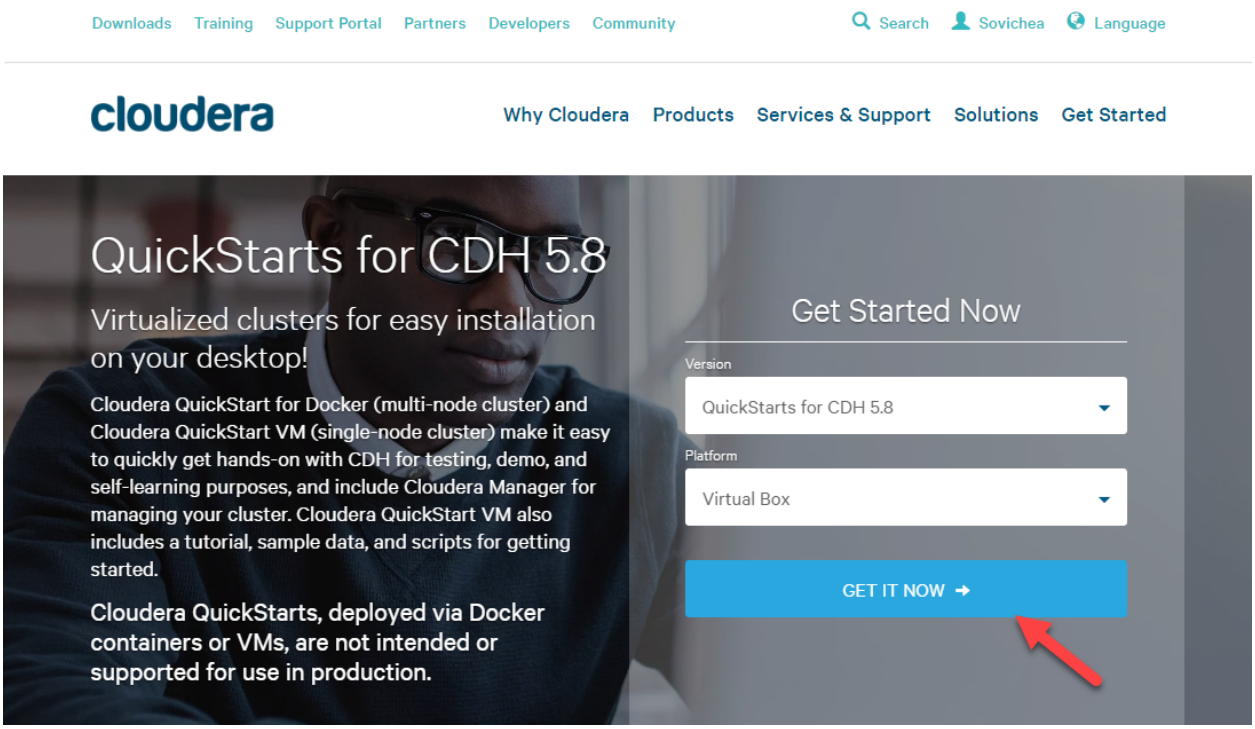
Click on **Sign In**, you will need an account to download Cloudera, if you don't have an account yet, click on **Register Now**.



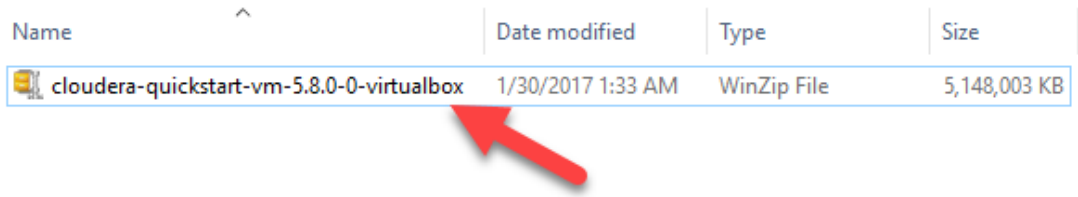
After finished Register or Sign In, go to http://www.cloudera.com/downloads.html to download Cloudera. Under Quick Start, click on **Download Now**



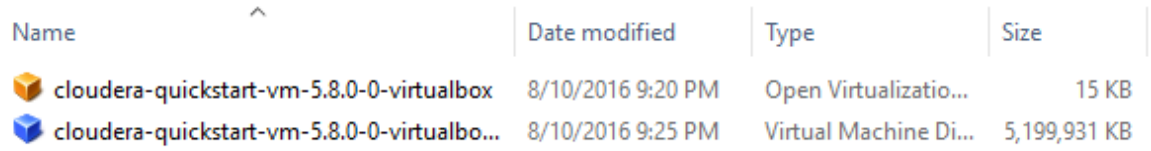
Select Version and Platform: Virtual Box, then click **Get It Now**



Download will start, you will get the zip file after finished

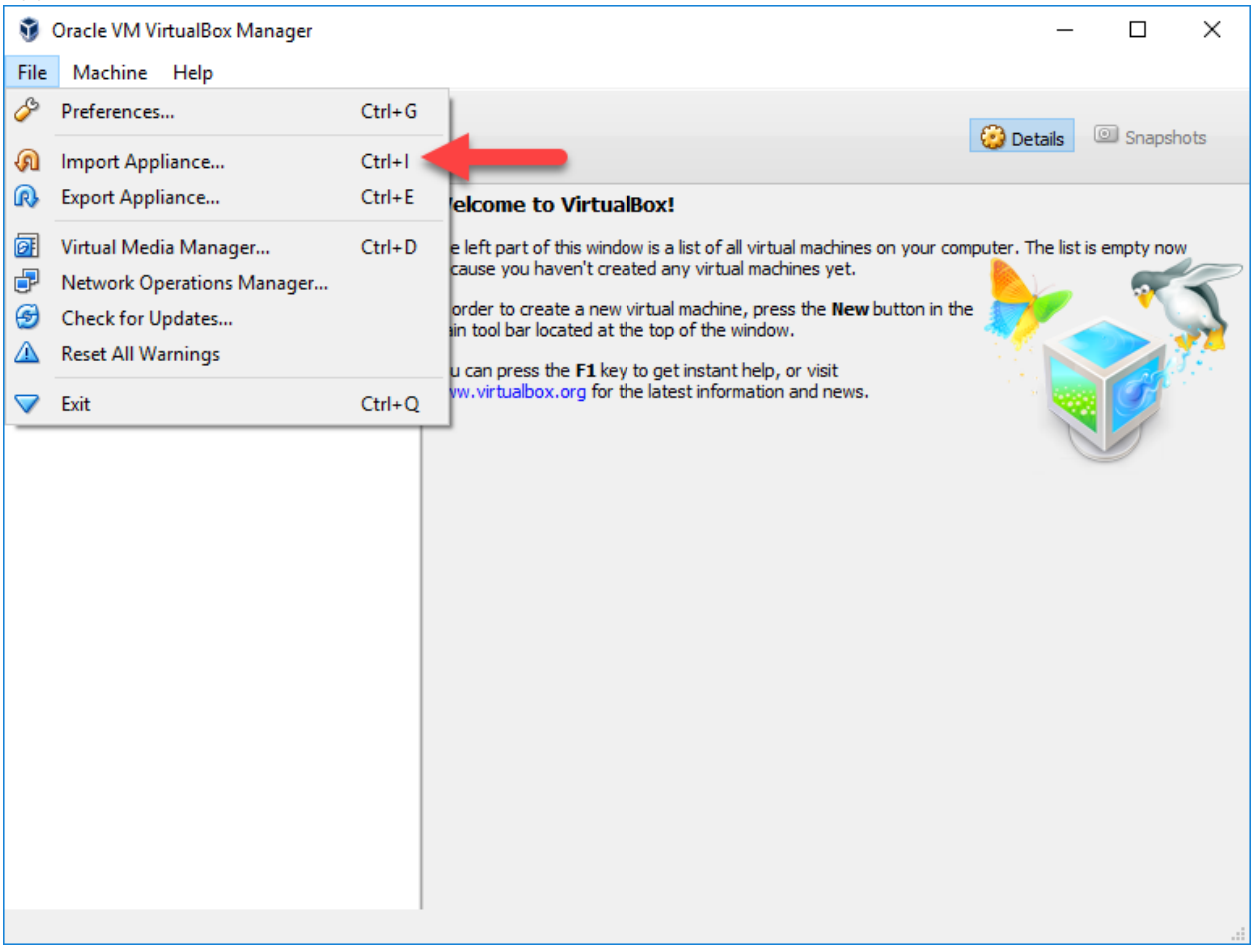


Extract the zip file, you will get VirtualBox image file

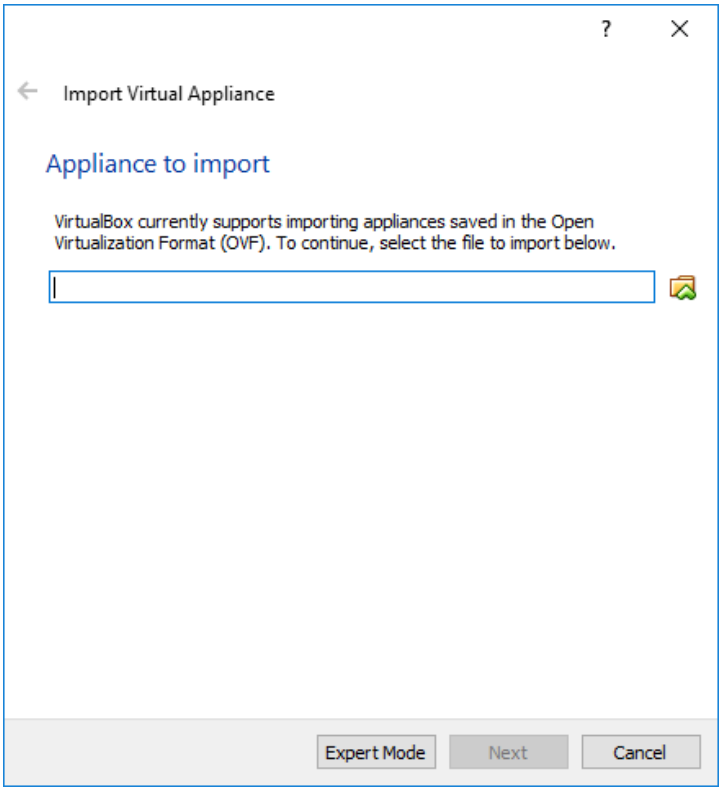


b. Install Cloudera

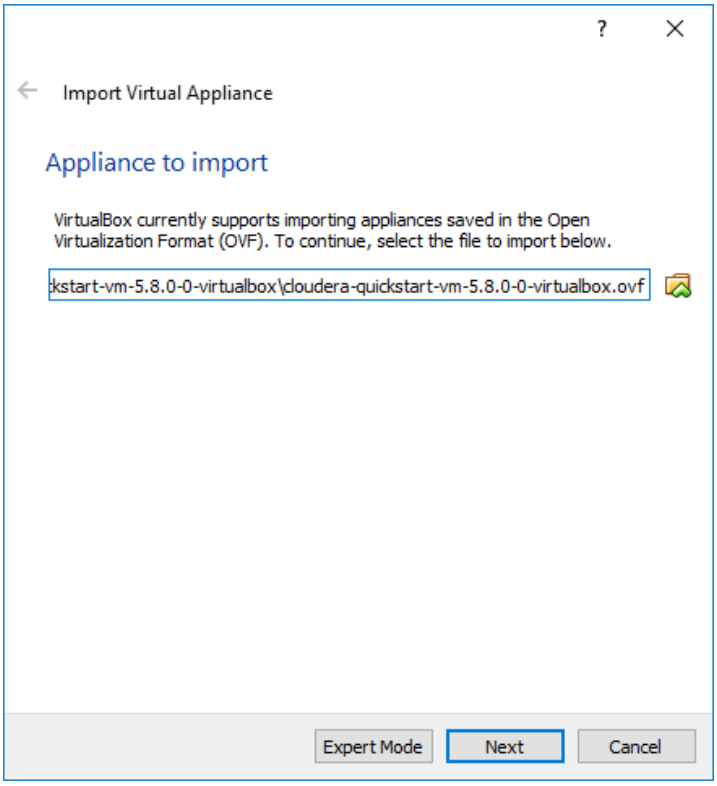
After you VirtualBox image file, you can import it to VirtualBox. Run VirtualBox, then click on Import Appliance



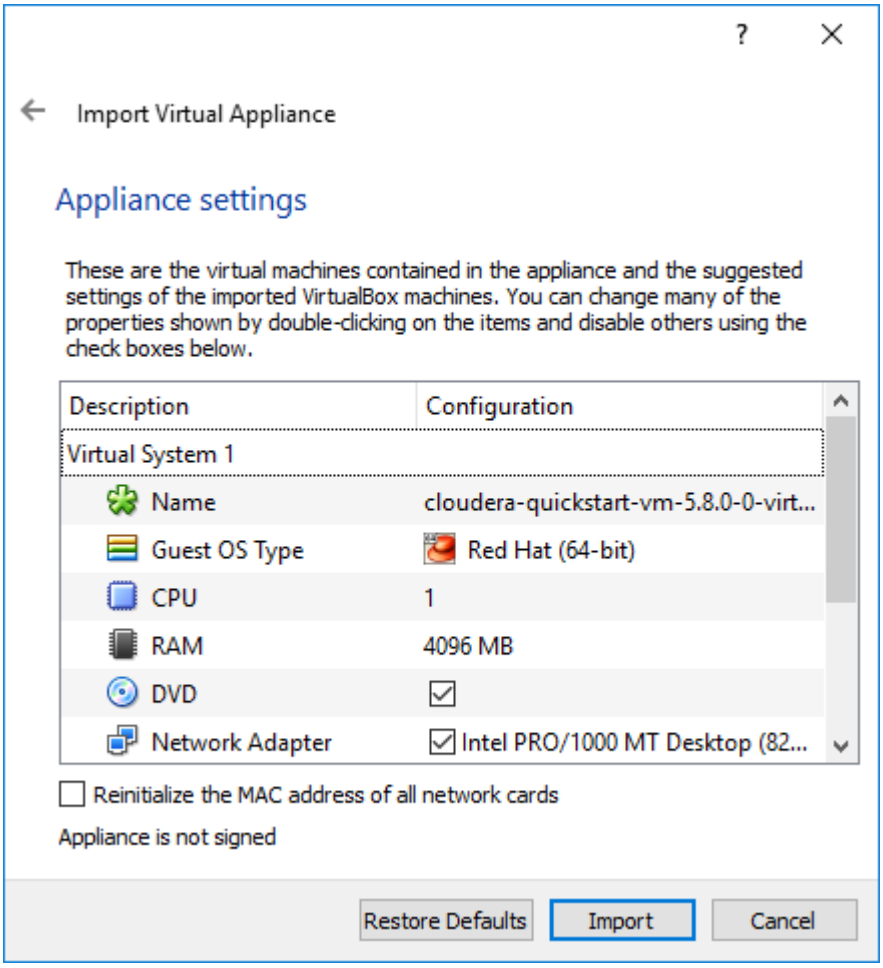
Select your VirtualBox image file location



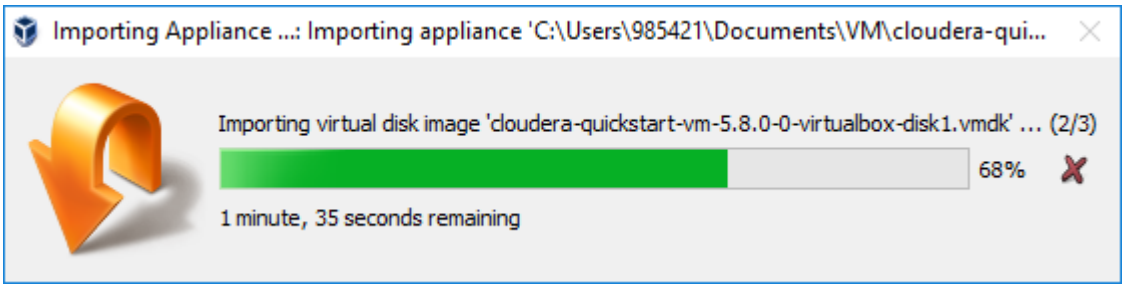
After that click Next



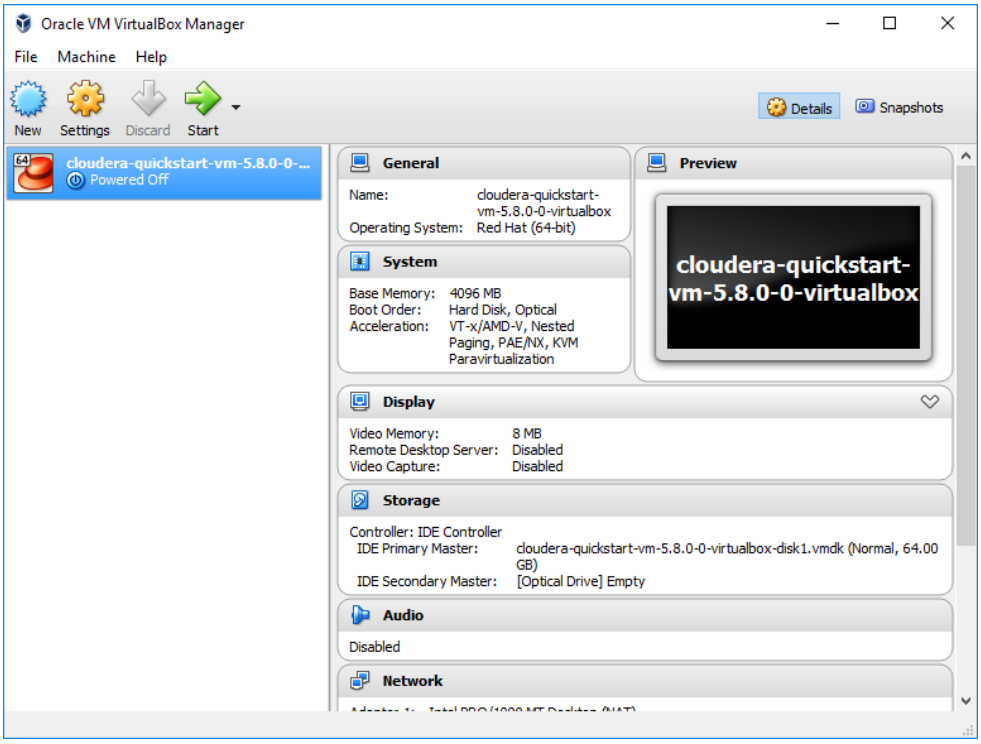
Configure your setting. Minimum RAM requirement for this VM is 4GB. Then click on Import.



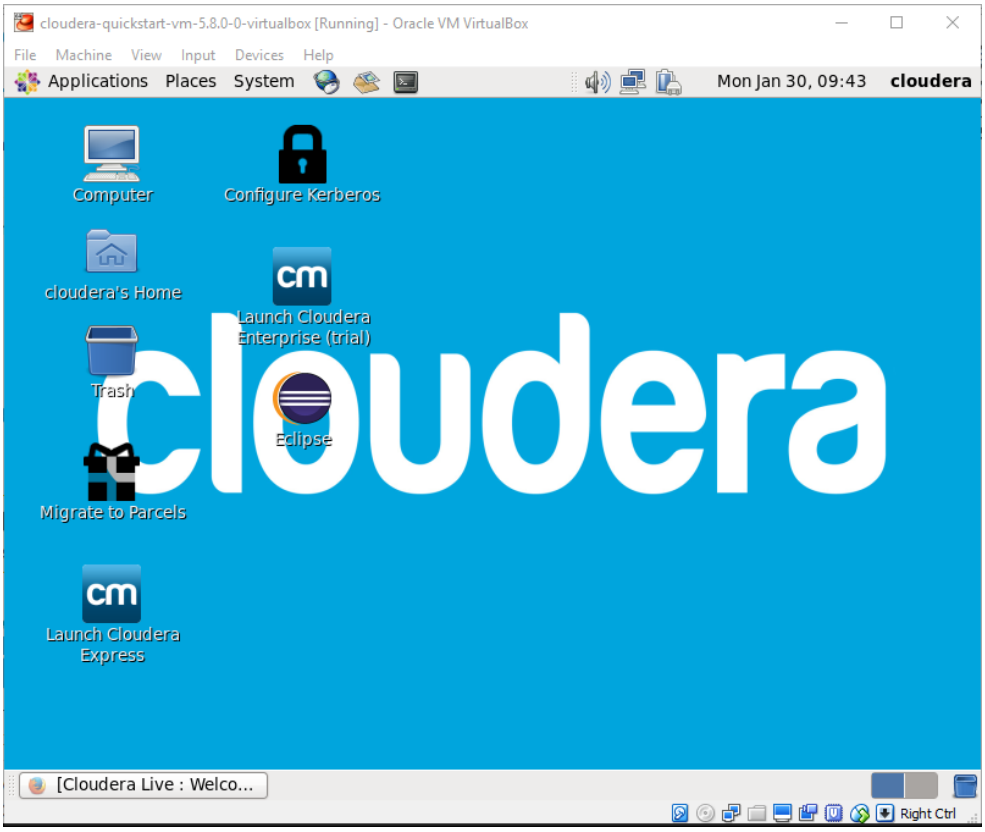
VirtualBox will start importing the appliance.



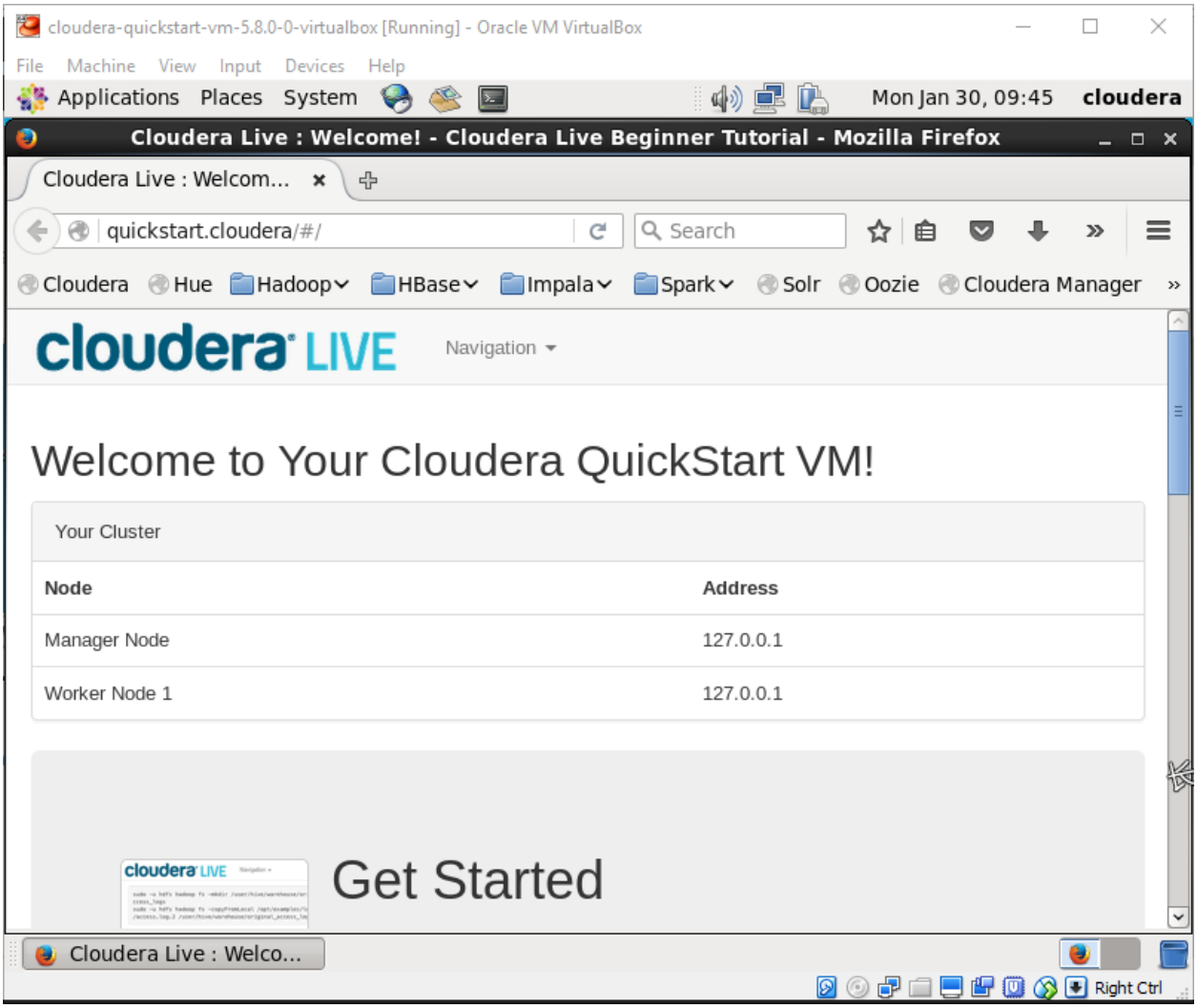
After finished import, you will able to see your Virtual Machine. Click Start to start the VM.



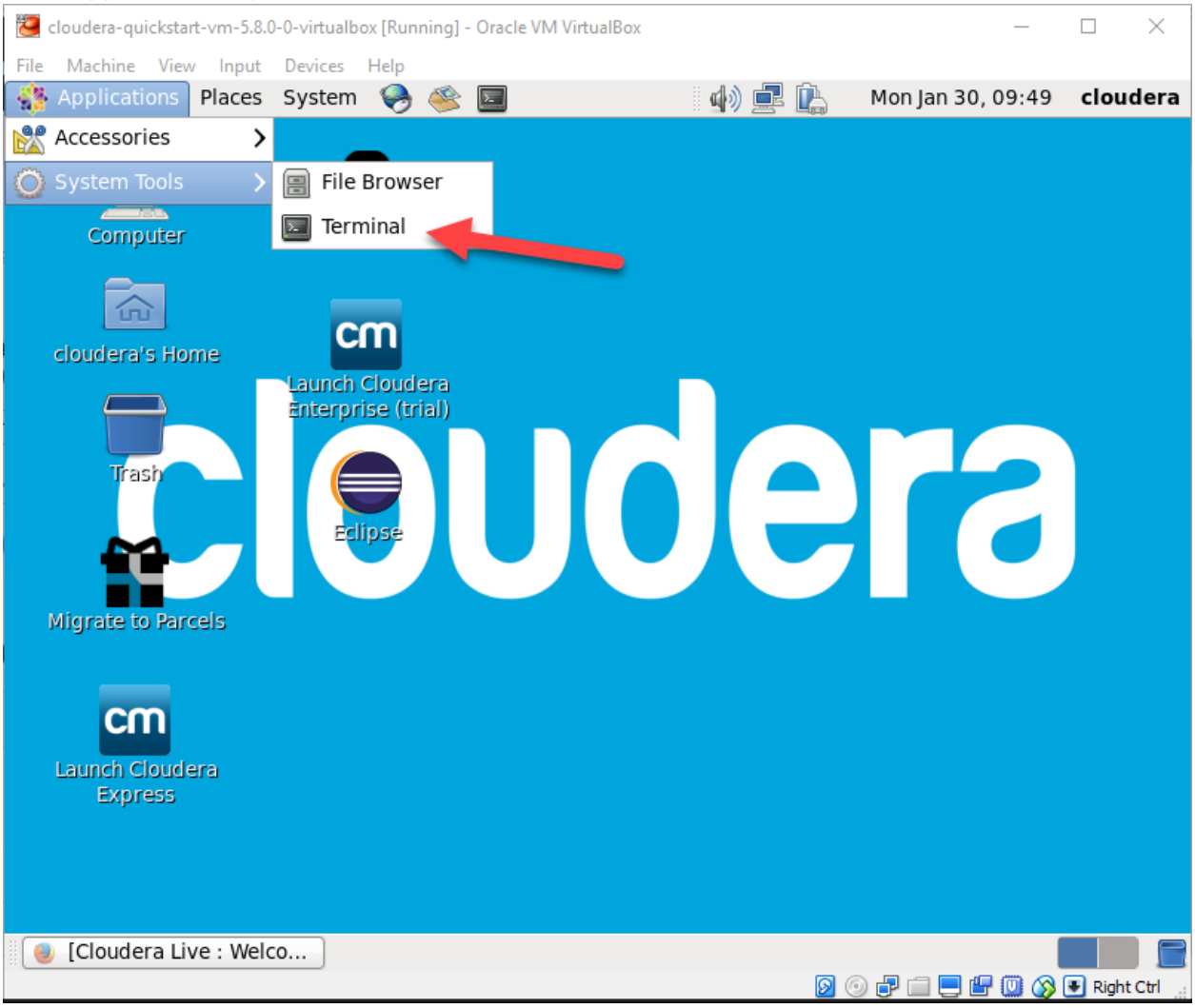
You will able to get to Red Hat CentOS after finish running



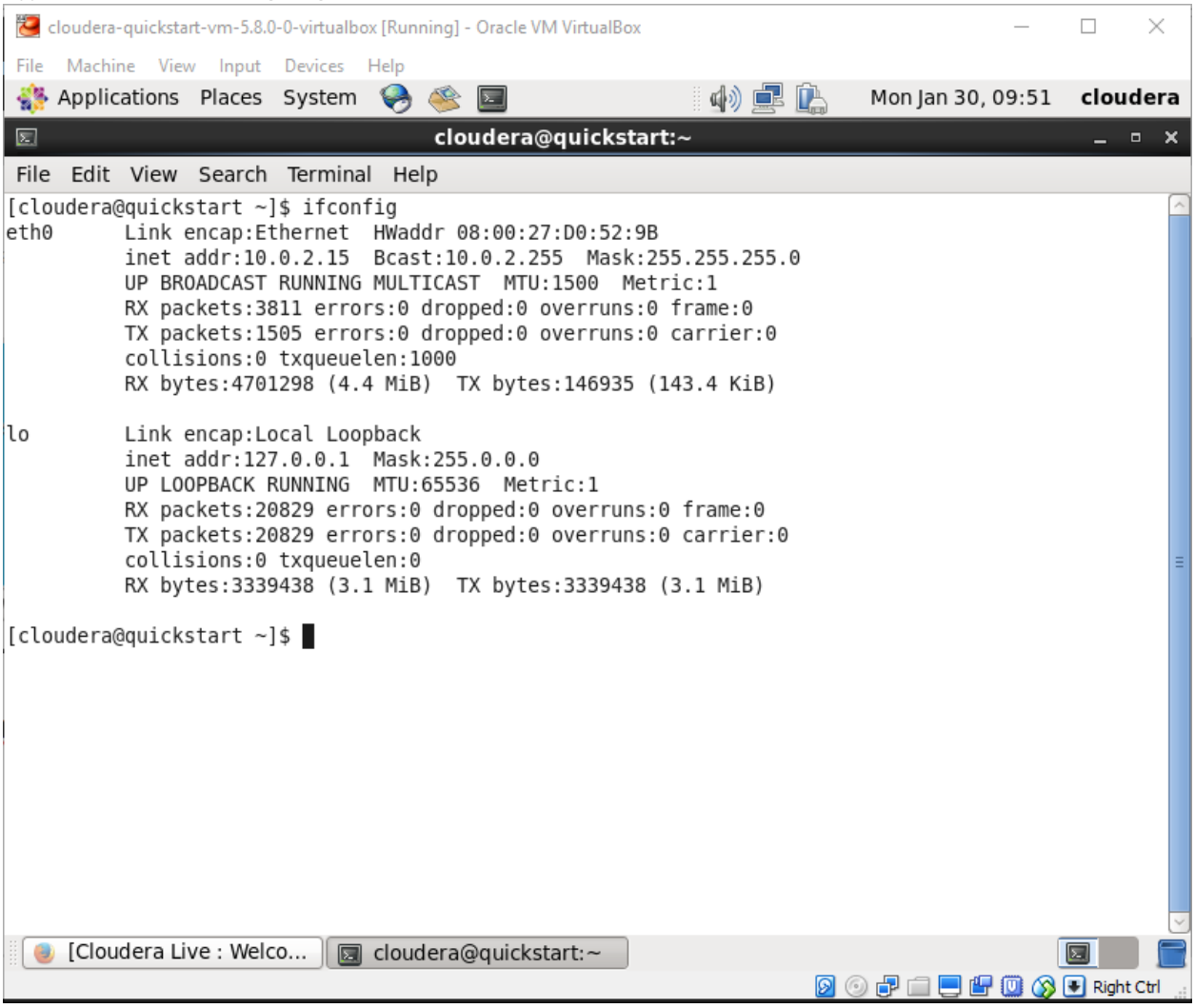
Cloudera Welcome page will pop up automatically.



Go to Application -> System Tools -> Terminal



Type command: **ifconfig** to get the IP and network information



III. Test Run WordCount

a. Download Hadoop Libraries

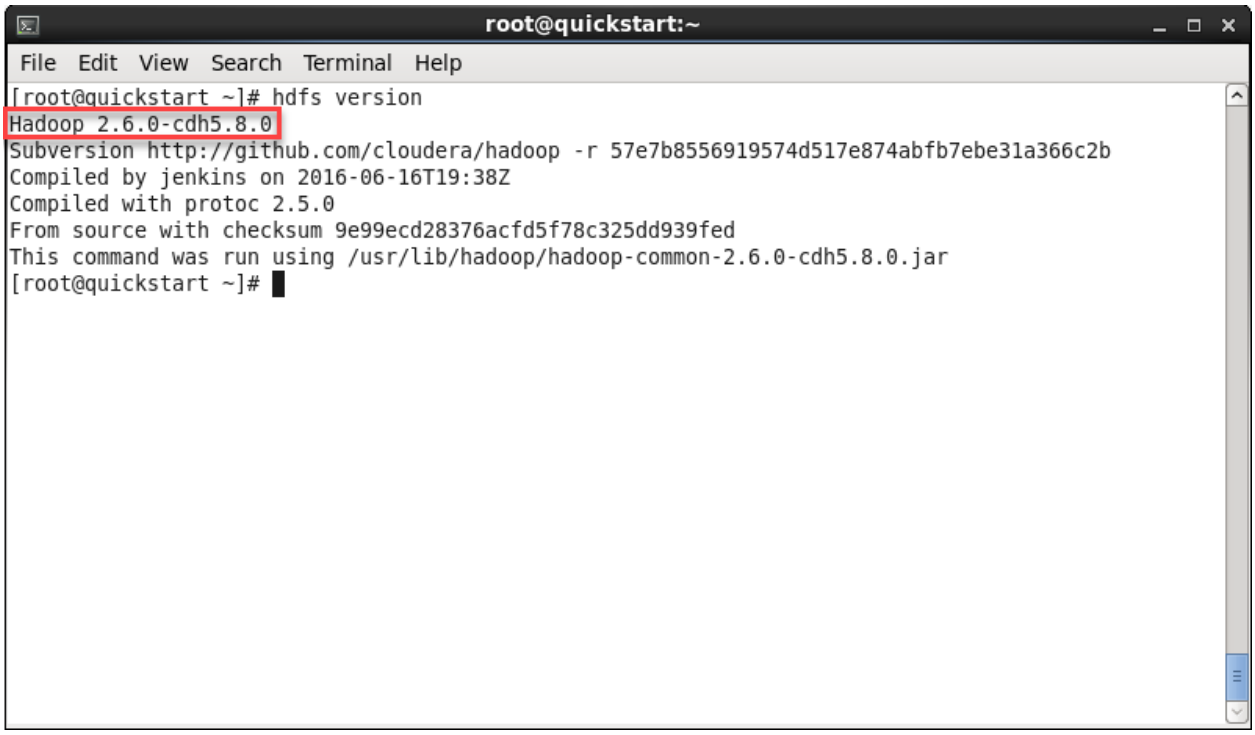
You will require hadoop libraries to run WordCount.

First, check your Hadoop version. Go to Terminal -> run command

hdfs version

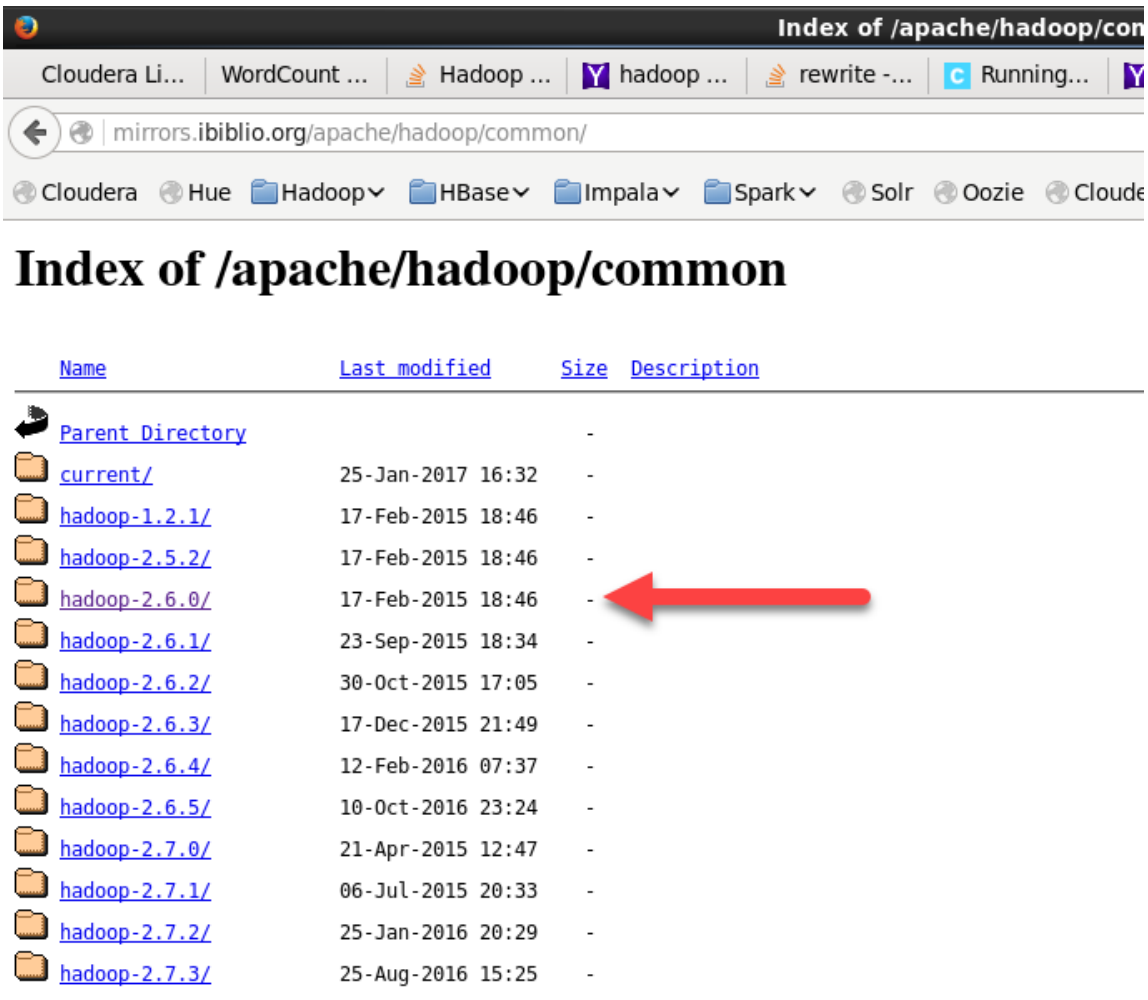
Or

hadoop version

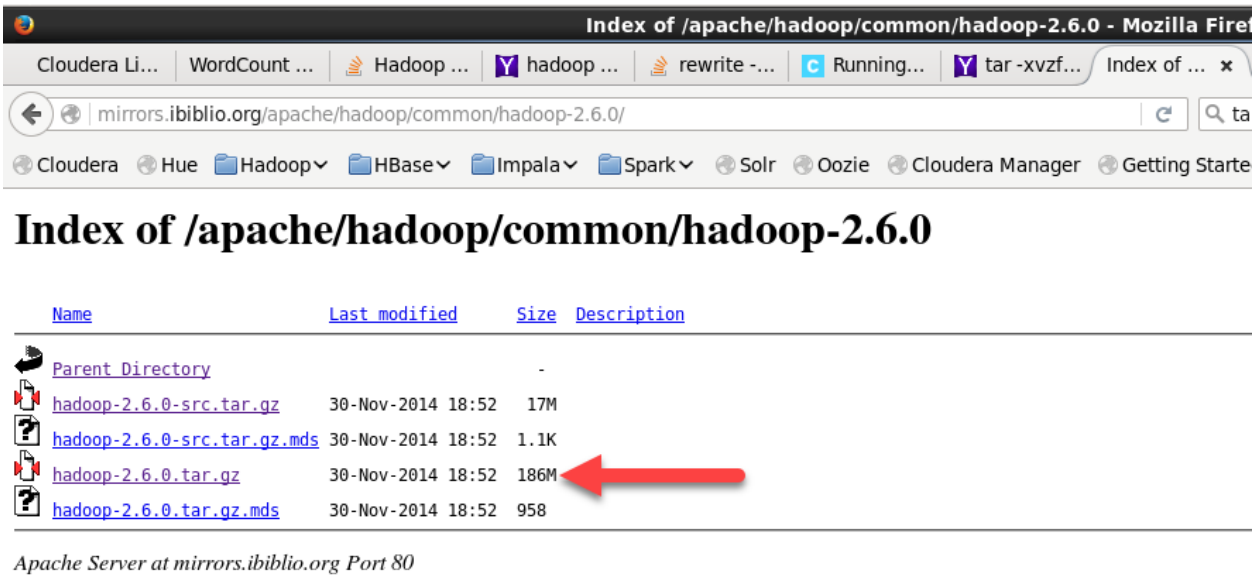


In this case, Hadoop version is 2.6.0, so we will need to download libraries exactly as this version.

You can download Hadoop library from this website
http://mirrors.ibiblio.org/apache/hadoop/common/



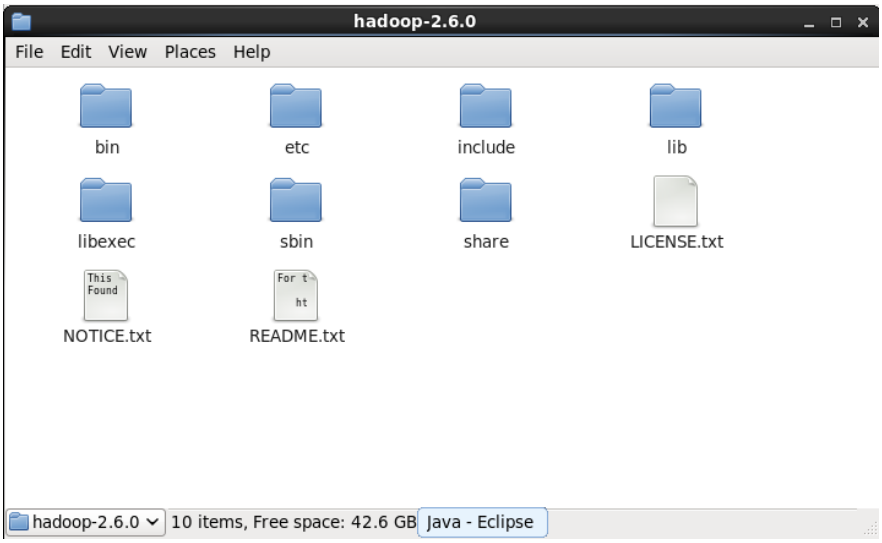
Choose the correct version and file



After that, extract the file with this command

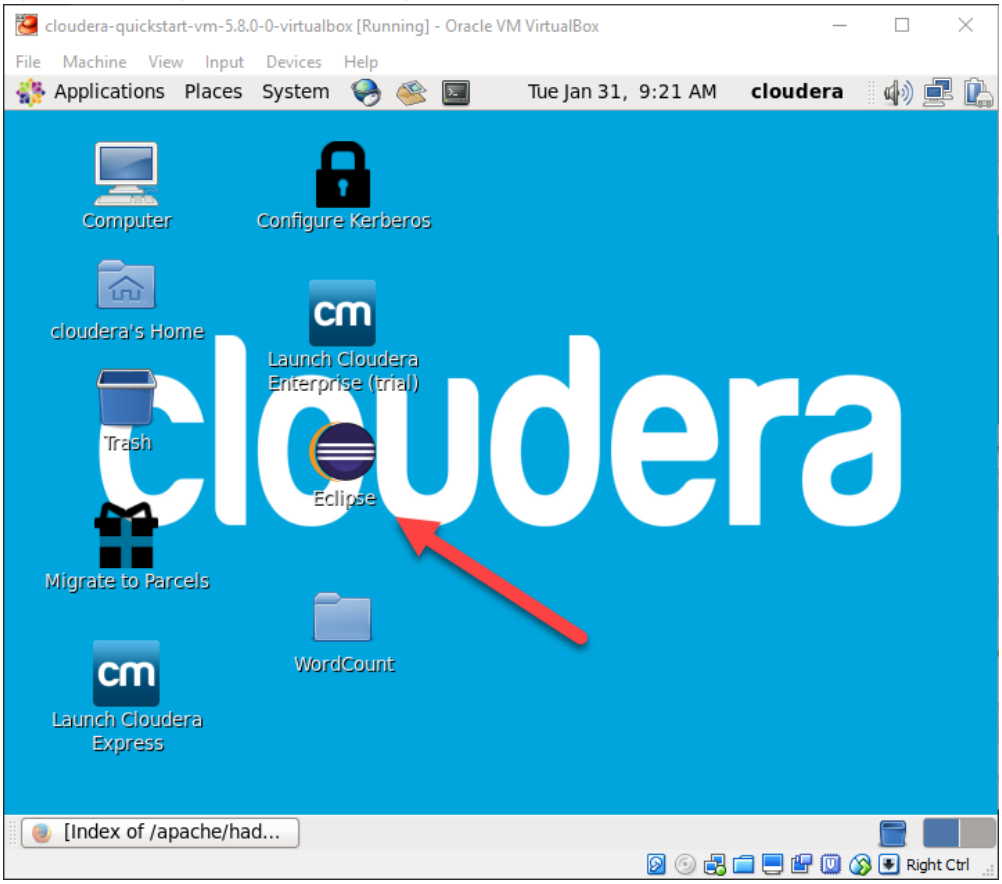
tar –xvzf filename.tar.gz

You will get the file as below, and you will need it to import to Eclipse later

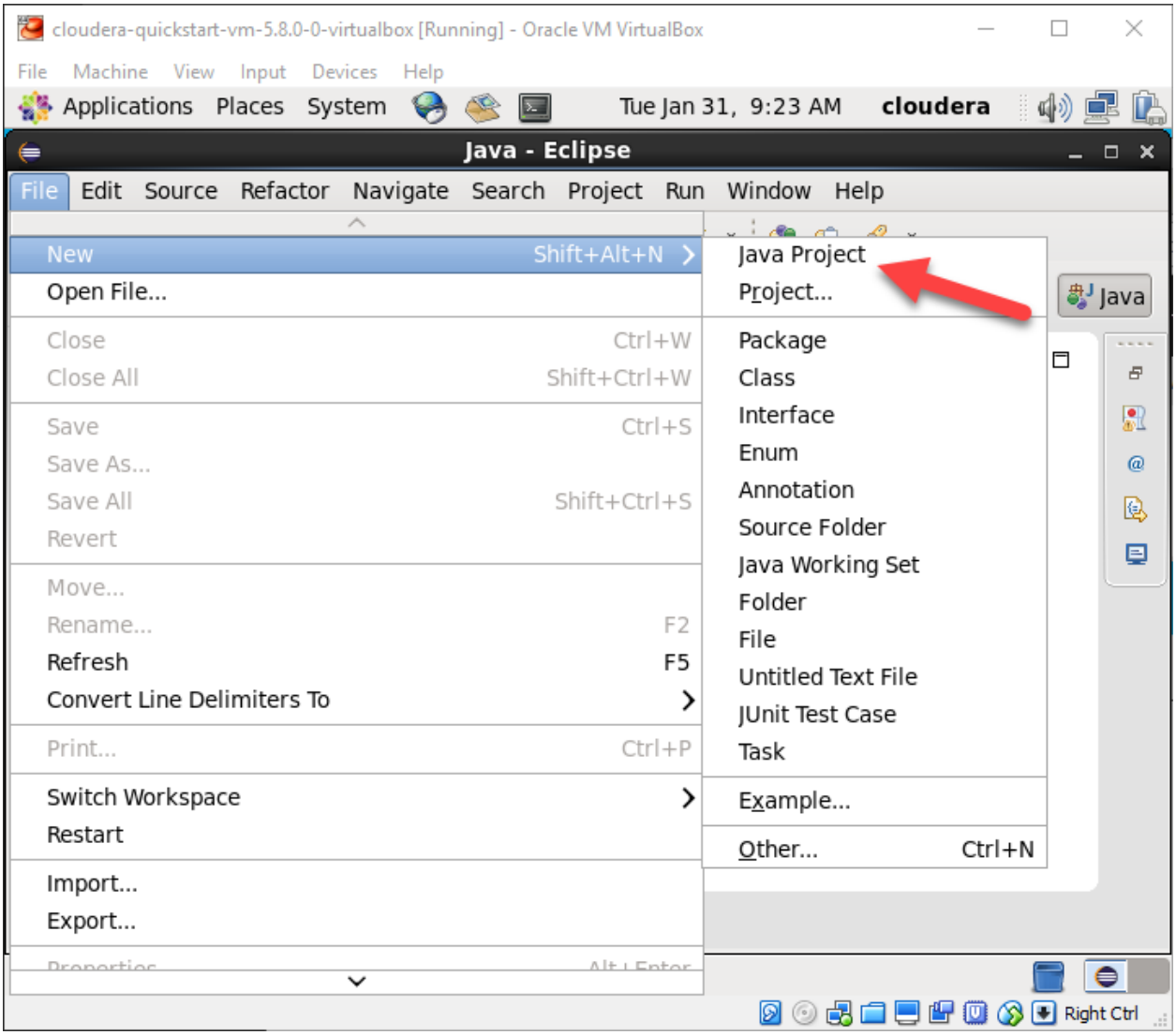


b. Eclipse Project Setup

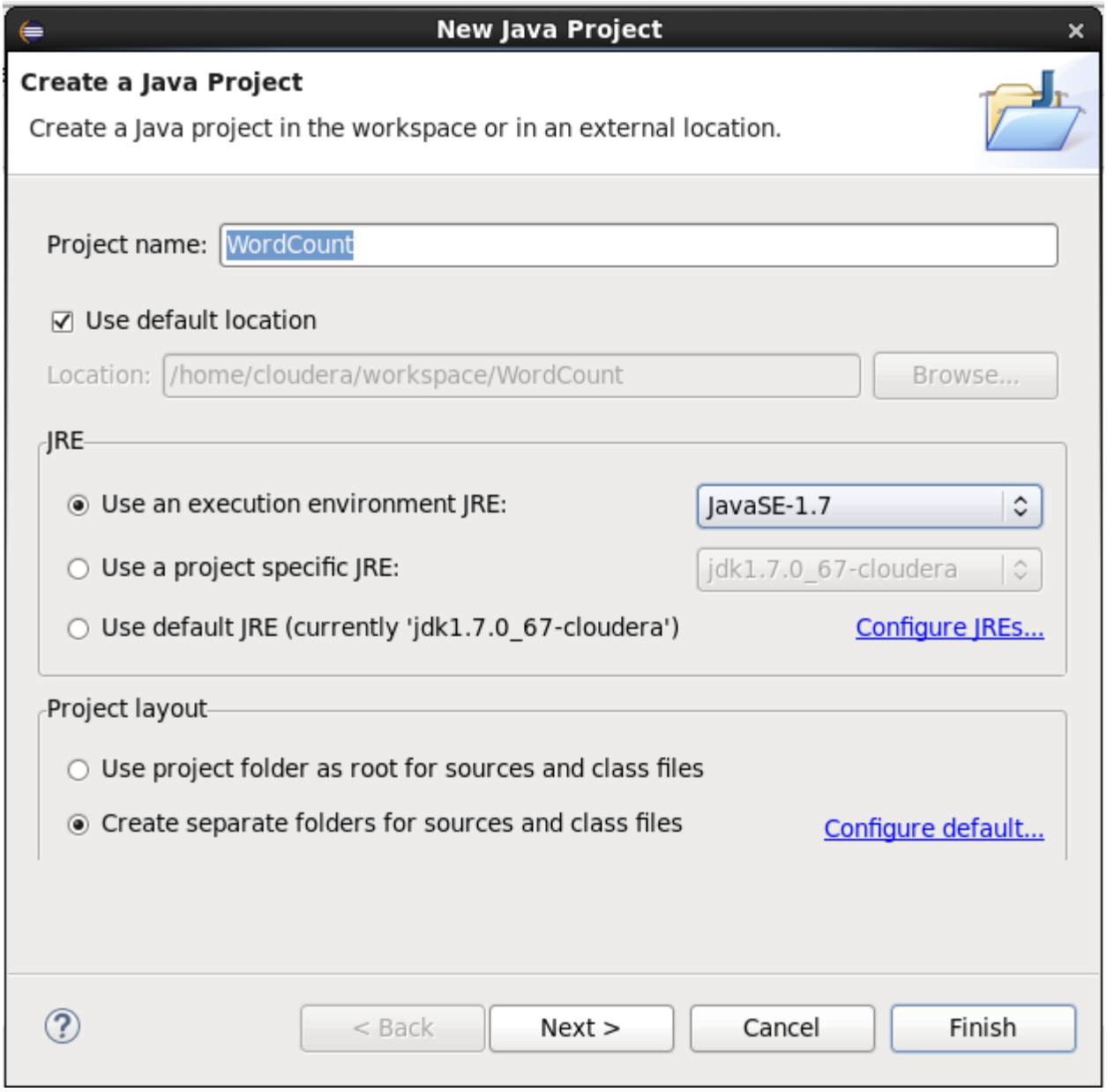
By default, Eclipse is available on your VM machine



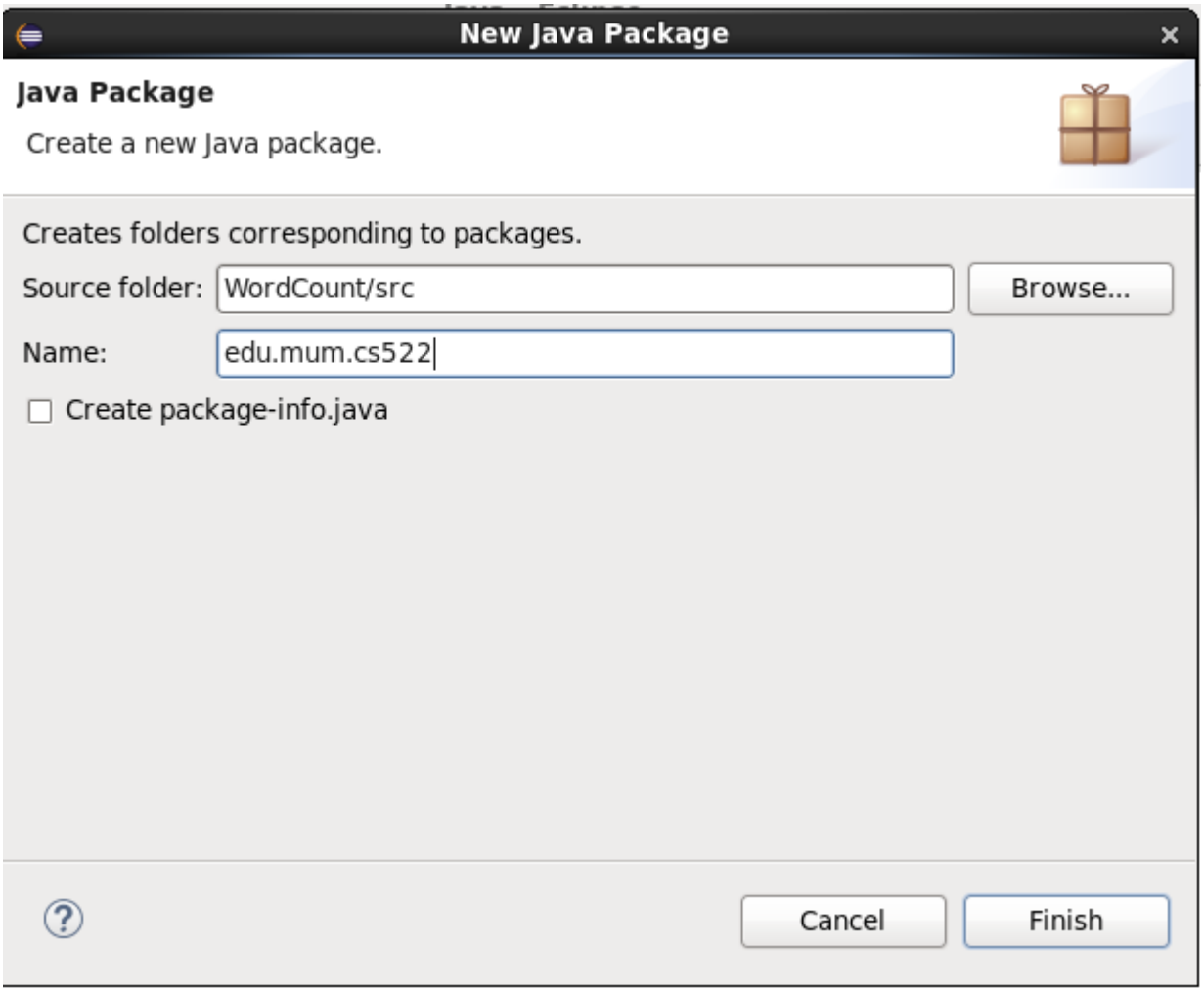
Run the Eclipse IDE and then create Java Project



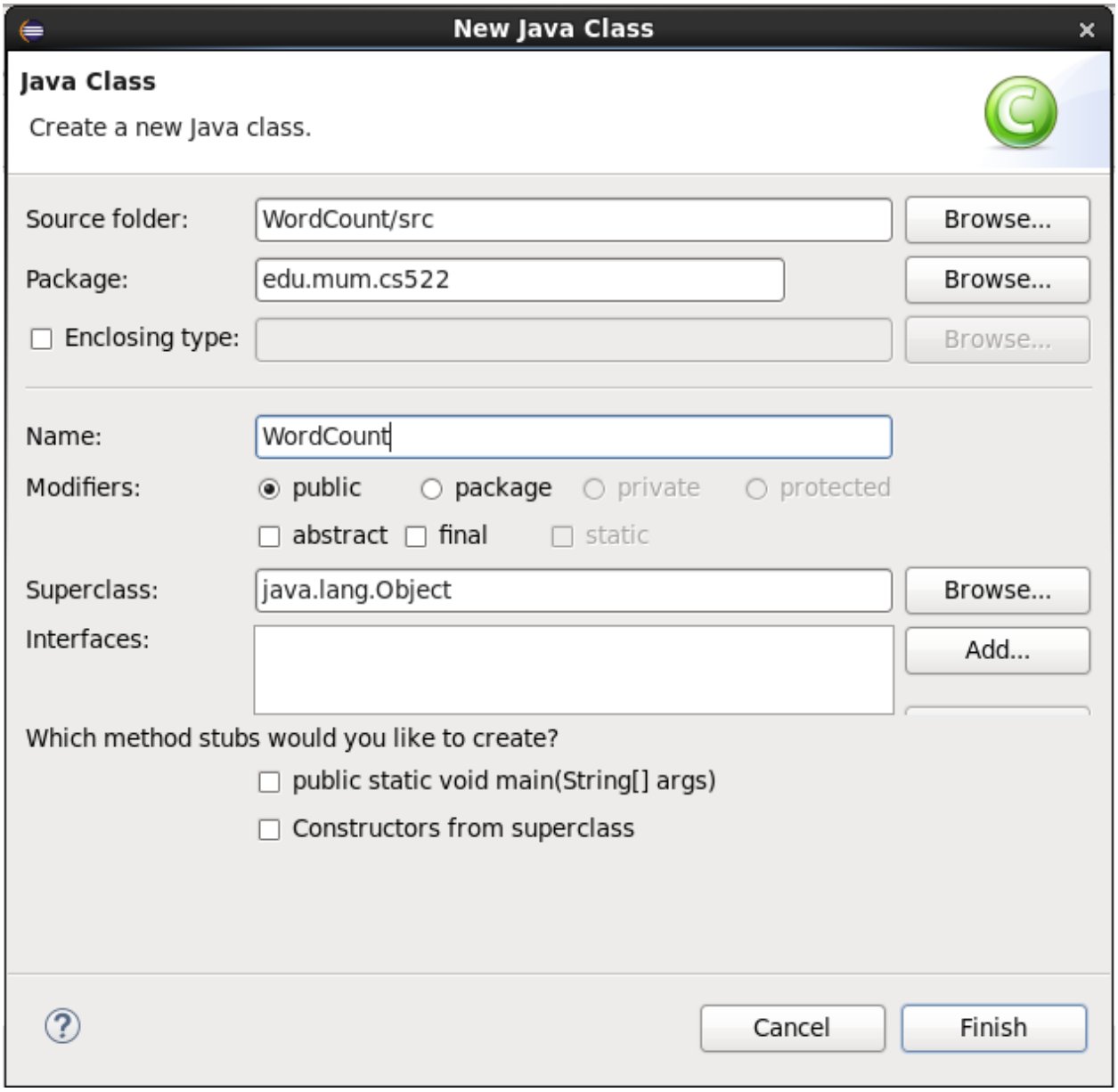
Setup project name and JRE



Create Package name



Create WordCount class



c. WordCount Source Code and Library Setup

WordCount maps (extract) words from an input source and reduces the result, return a count of each word. You can find source code below or on the internet

```java
package edu.mum.cs522;

import java.io.IOException;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(WordCount.class);

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new WordCount(), args);
    System.exit(res);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "wordcount");
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is used
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private long numRecords = 0;
    private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");

    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      Text currentWord = new Text();
      for (String word : WORD_BOUNDARY.split(line)) {
        if (word.isEmpty()) {
            continue;
        }
            currentWord = new Text(word);
```

```
            context.write(currentWord,one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
        sum += count.get();
      }
      context.write(word, new IntWritable(sum));
    }
  }
}
```

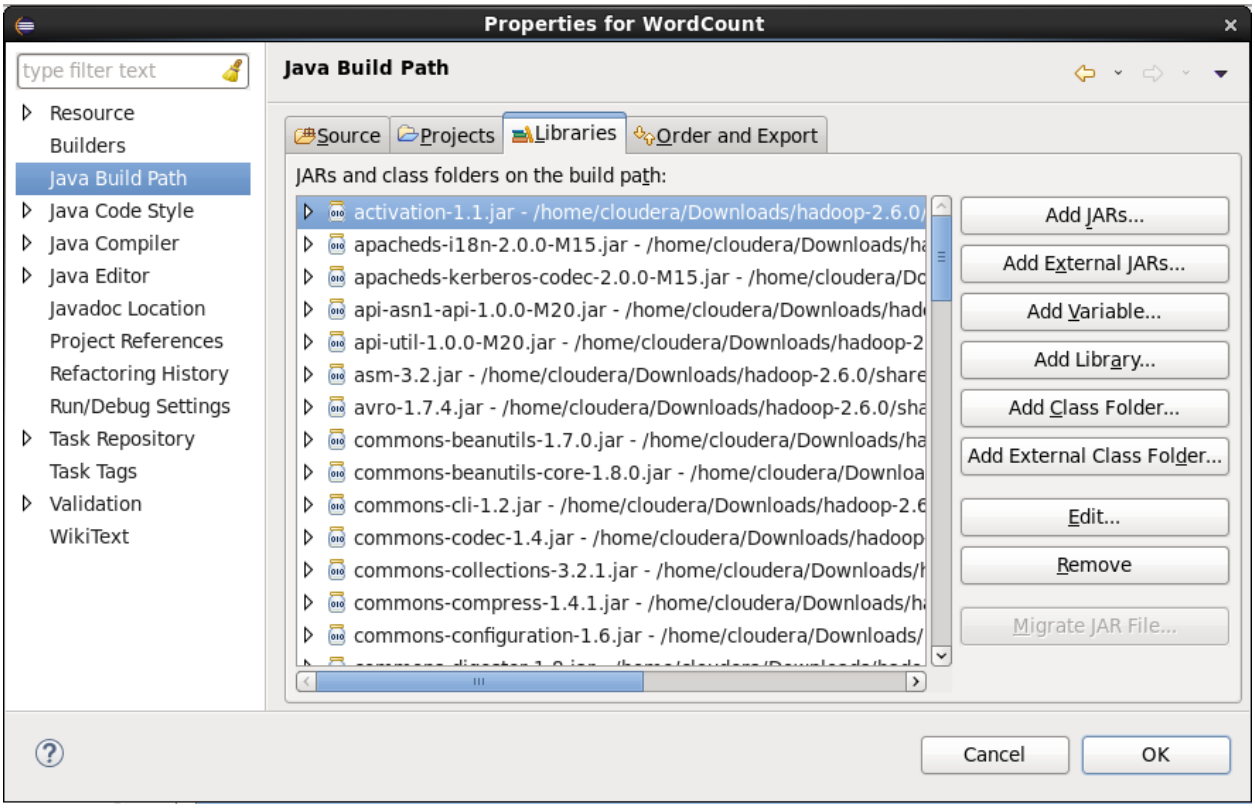
Add all required libraries as below

`share/hadoop/common/hadoop-common-*.jar`

`share/hadoop/mapreduce/hadoop-mapreduce-client-core-*.jar`

`share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-*.jar`

`share/hadoop/common/hadoop-common-lib-*.jar (all files)`



d. Running WordCount

Before you run, you must create input and output locations in HDFS. Use the following commands to create input directory /user/cloudera/wordcount/input in HDFS:

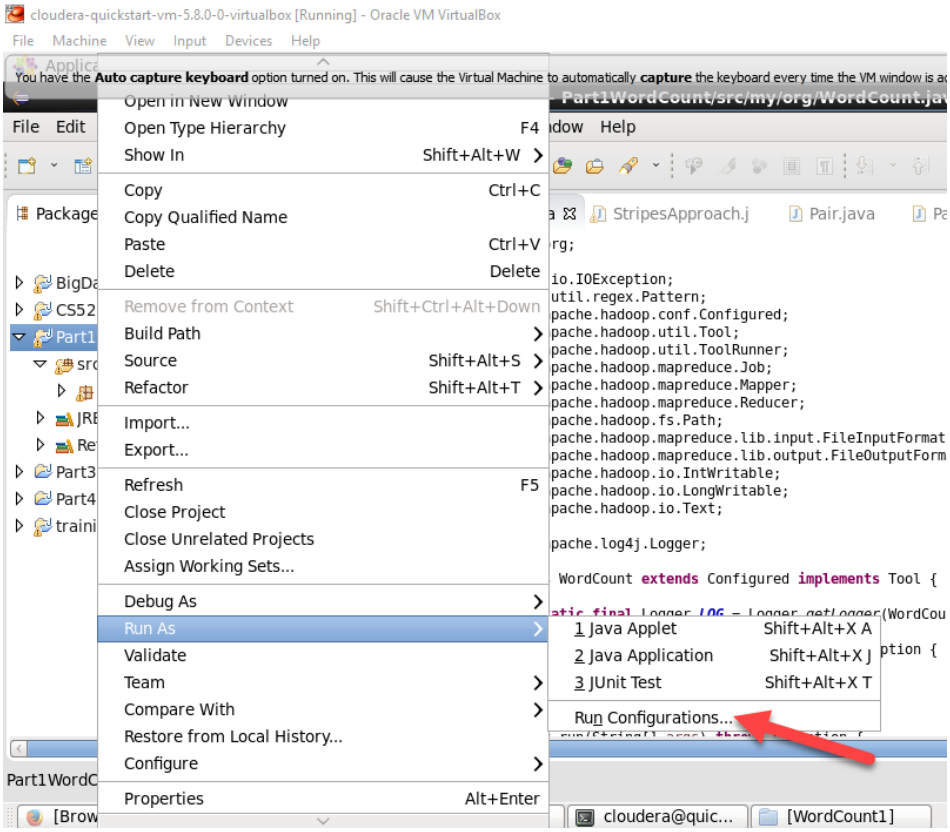
```
$ sudo su hdfs

$ hadoop fs -mkdir /user/cloudera

$ hadoop fs -chown cloudera /user/cloudera

$ exit$ sudo su cloudera

$ hadoop fs -mkdir /user/cloudera/wordcount /user/cloudera/wordcount/input
```

Create sample text files to use as input, and move them to the /user/cloudera/wordcount/input directory in HDFS. You can use any files you choose; for convenience, the following shell commands create a few small input files for illustrative purposes.

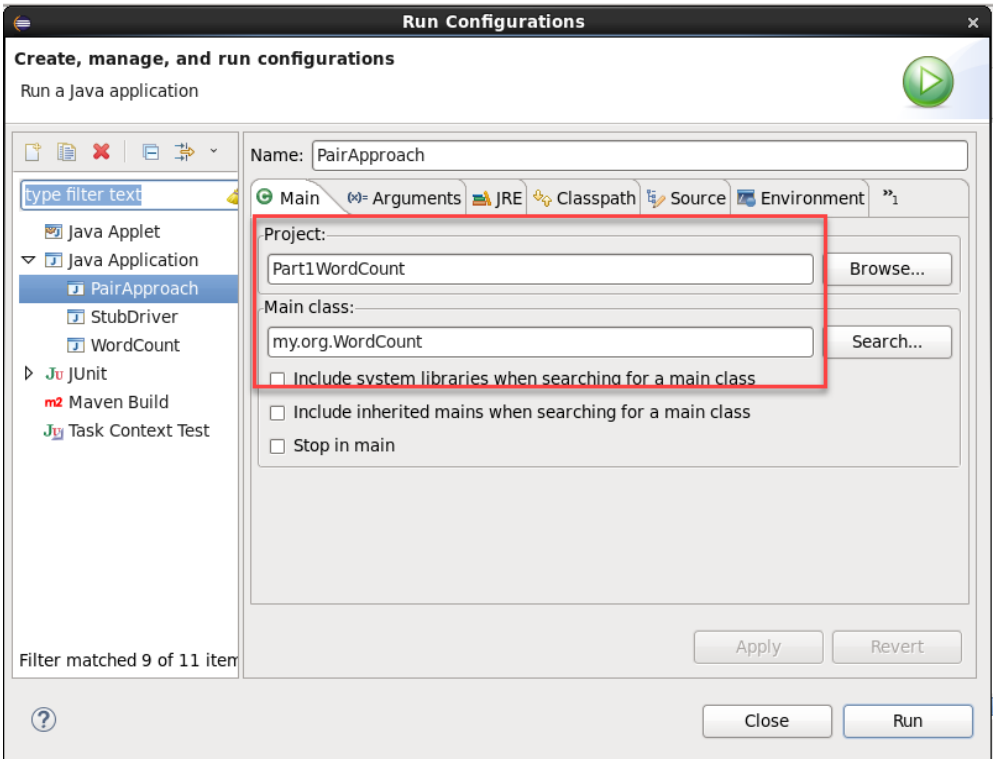
```
$ echo "Hadoop is an elephant" > file0

$ echo "Hadoop is as yellow as can be" > file1

$ echo "Oh what a yellow fellow is Hadoop" > file2

$ hadoop fs -put file* /user/cloudera/wordcount/input
```

Compile WordCount class.

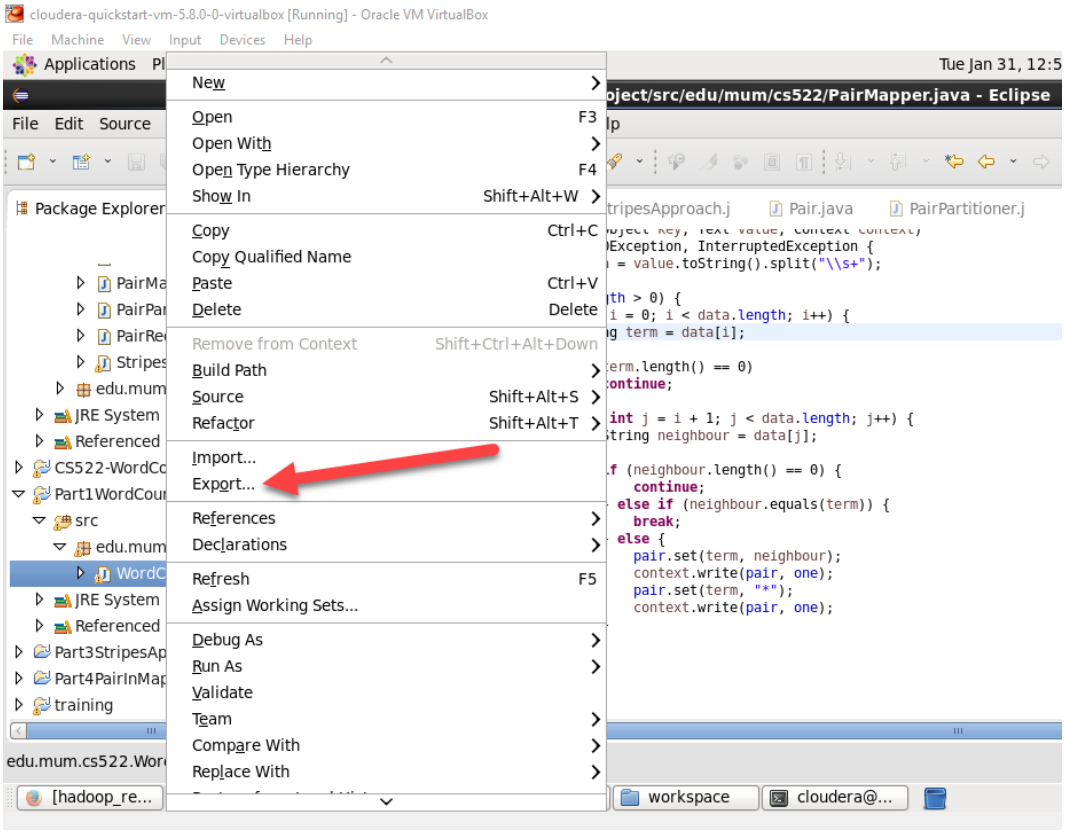
Check your project Run Configuration



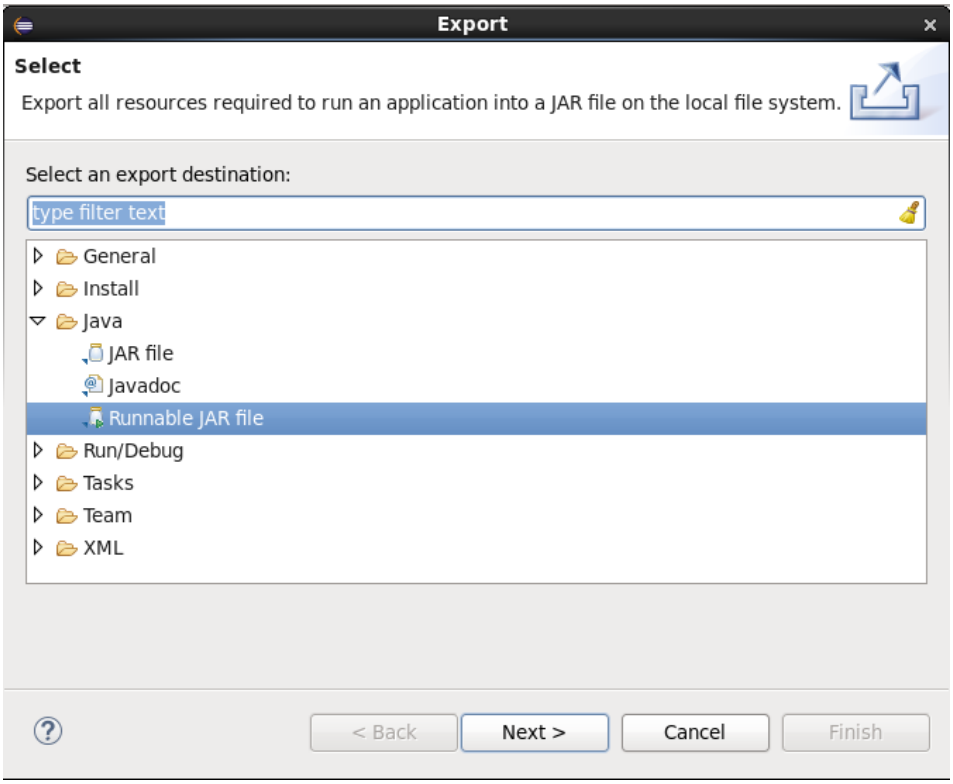
Make sure you check your project name and main class



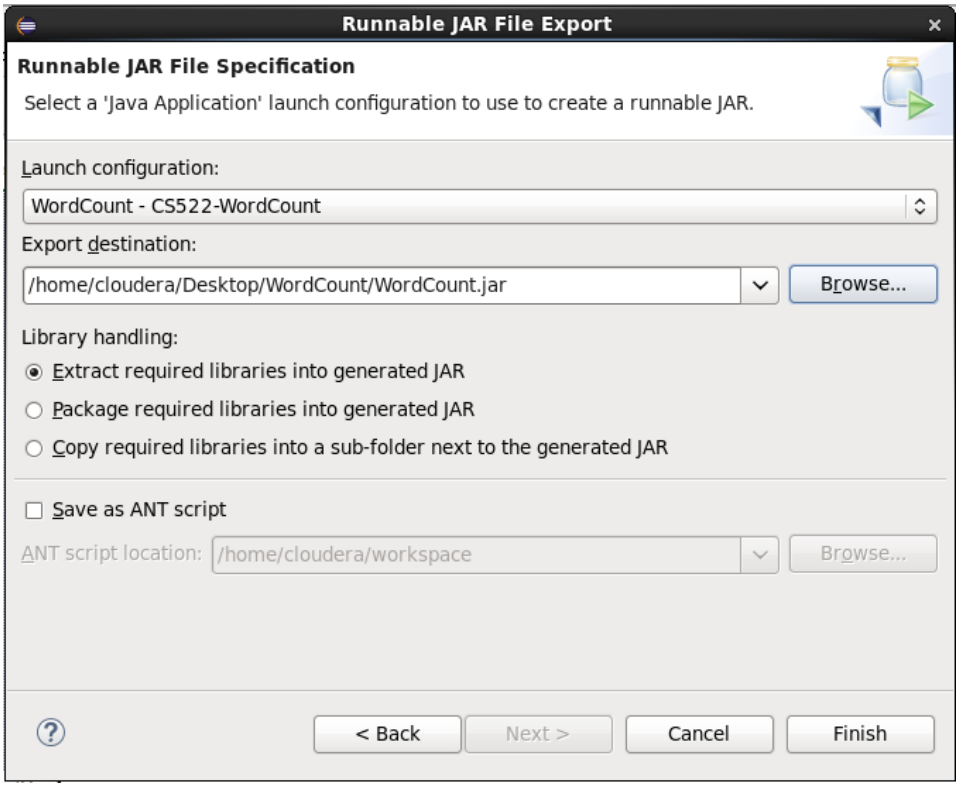
In Eclipse, right click on your class file -> Choose Export



Select Java -> Runnable JAR file



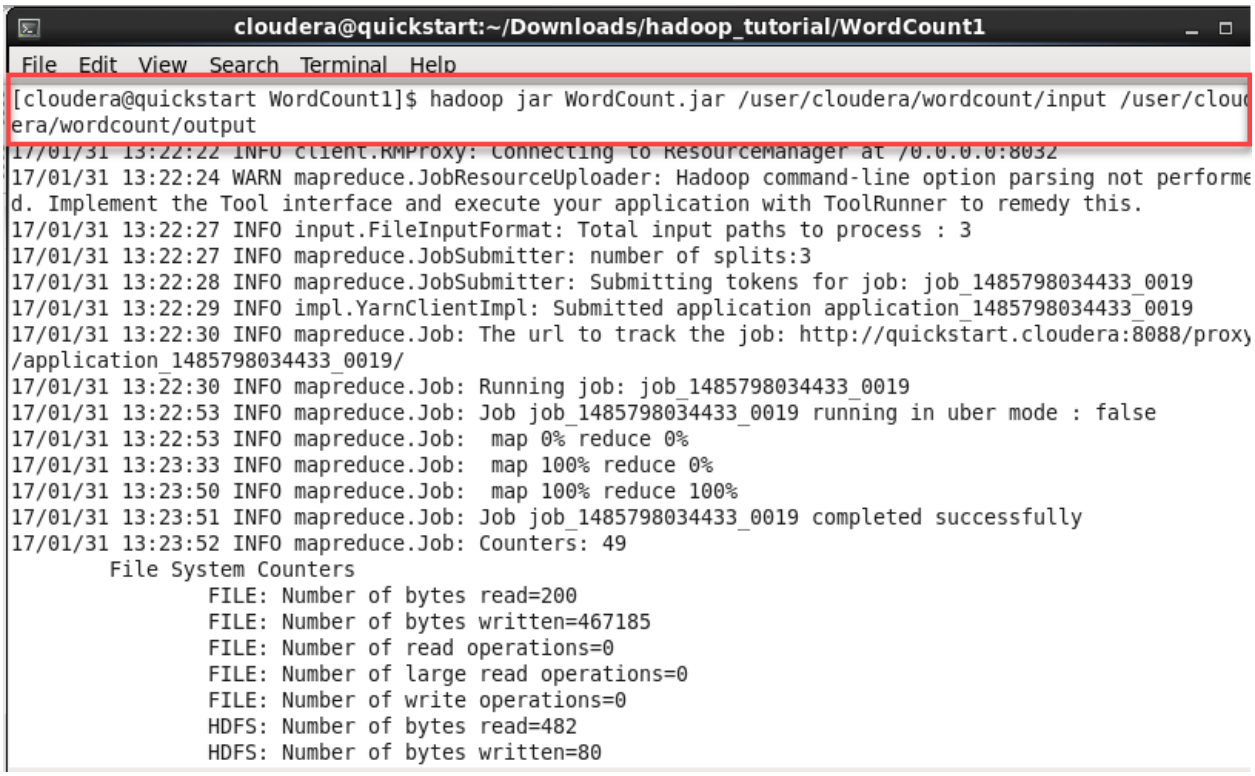
Select your project configuration and export destination. After that, click Finish.



To run the WordCount application from JAR file, passing the paths to the input and output directories in HDFS.

Run command

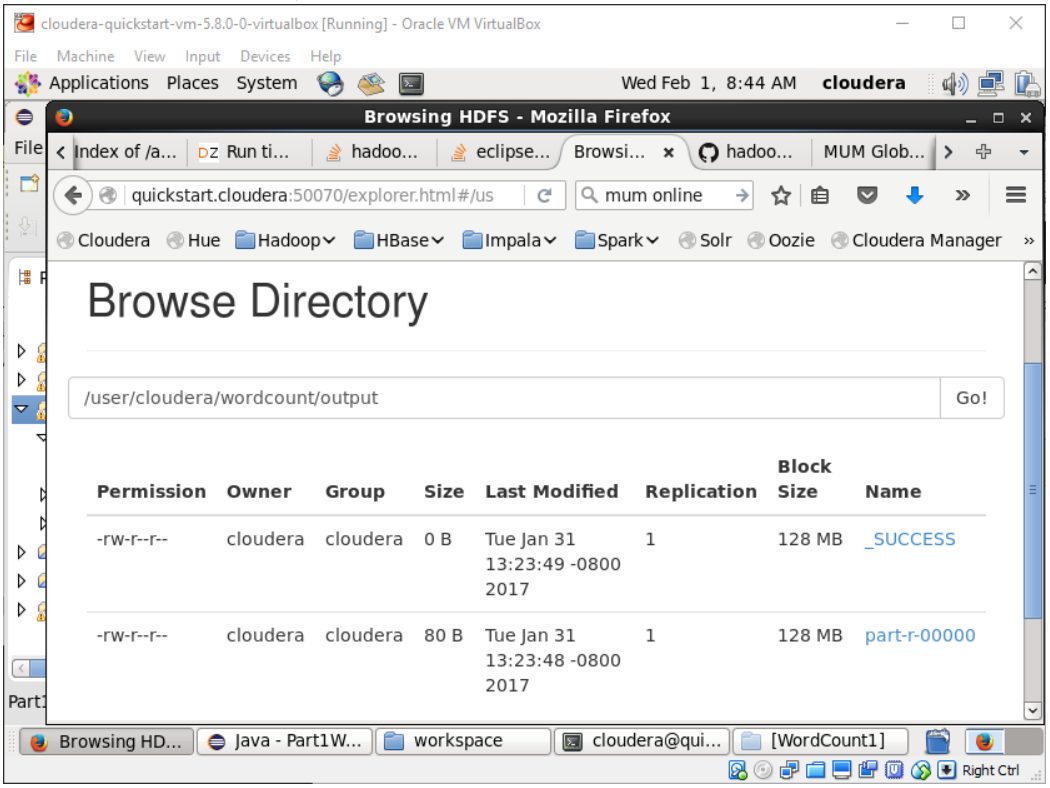
$ hadoop jar WordCount.jar /user/cloudera/wordcount/input /user/cloudera/wordcount/output



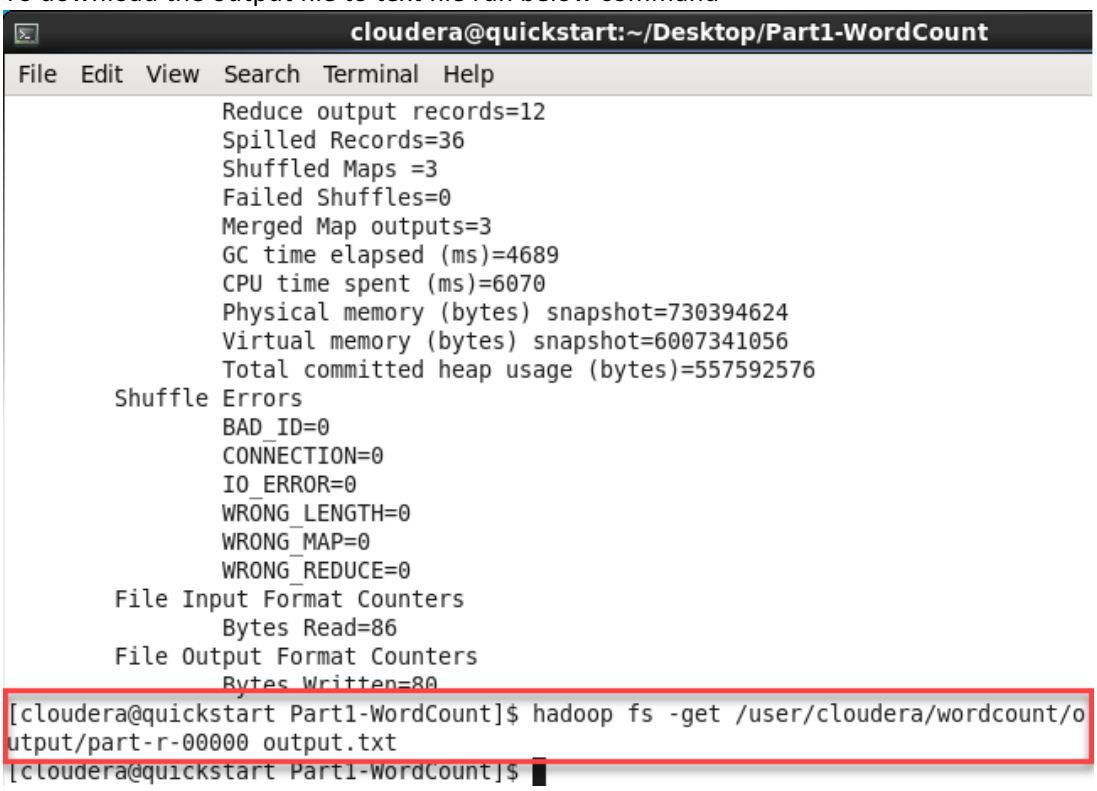
You can see some output result as below

```
cloudera@quickstart:~/Downloads/hadoop_tutorial/WordCount1
File  Edit  View  Search  Terminal  Help
17/01/31 13:23:51 INFO mapreduce.Job: Job job_1485798034433_0019 completed successfully
17/01/31 13:23:52 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=200
                FILE: Number of bytes written=467185
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=482
                HDFS: Number of bytes written=80
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=3
                Launched reduce tasks=1
                Data-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=110973
                Total time spent by all reduces in occupied slots (ms)=14377
                Total time spent by all map tasks (ms)=110973
                Total time spent by all reduce tasks (ms)=14377
                Total vcore-seconds taken by all map tasks=110973
                Total vcore-seconds taken by all reduce tasks=14377
                Total megabyte-seconds taken by all map tasks=113636352
                Total megabyte-seconds taken by all reduce tasks=14722048
        Map-Reduce Framework
```

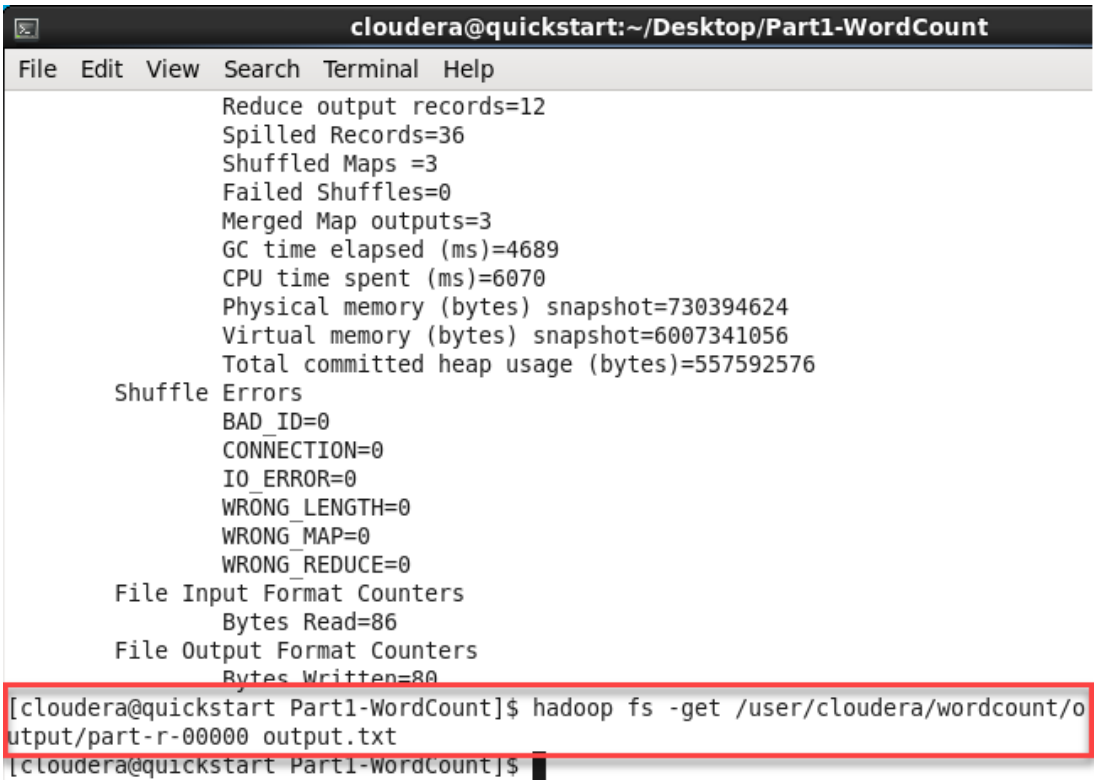
You also can browse output file on the web browser



To download the output file to text file run below command



Output.txt



To clear the output directory run below command

```
[cloudera@quickstart Part1-WordCount]$ hadoop fs -rm -r /user/cloudera/wordcount
/output
Deleted /user/cloudera/wordcount/output
```