Big Data Hadoop and Spark Developer

Lesson-End Project Solution

Count the Number of Words Using MapReduce

Steps to Perform:

Step 1: Create a sample file with the name mapreducedemo.java
Step 2: Create two functions Mapper and Reduce with the main method
Mapper implementation:

```
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);
                }
        }
```

Reduce implementation:

```
                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 method:

```
package org.simplilearn.demo.mapreduce.wordcount;
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()) {
                        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 {
                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, "wordcount");
        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);
    }
}
```

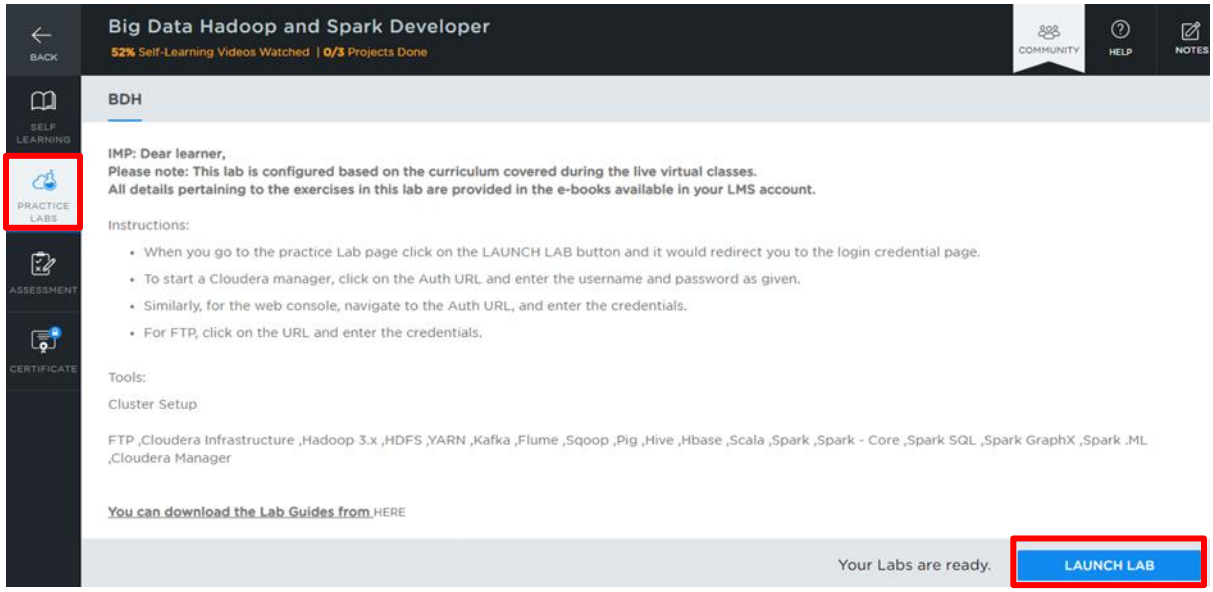
Step 3: Convert the JAVA files into JAR files and give it the name **“hadoop-mapreduce-example.jar”**

Note: “hadoop-mapreduce-example.jar” and “wordcount.txt” files are available in the Course Resources Section

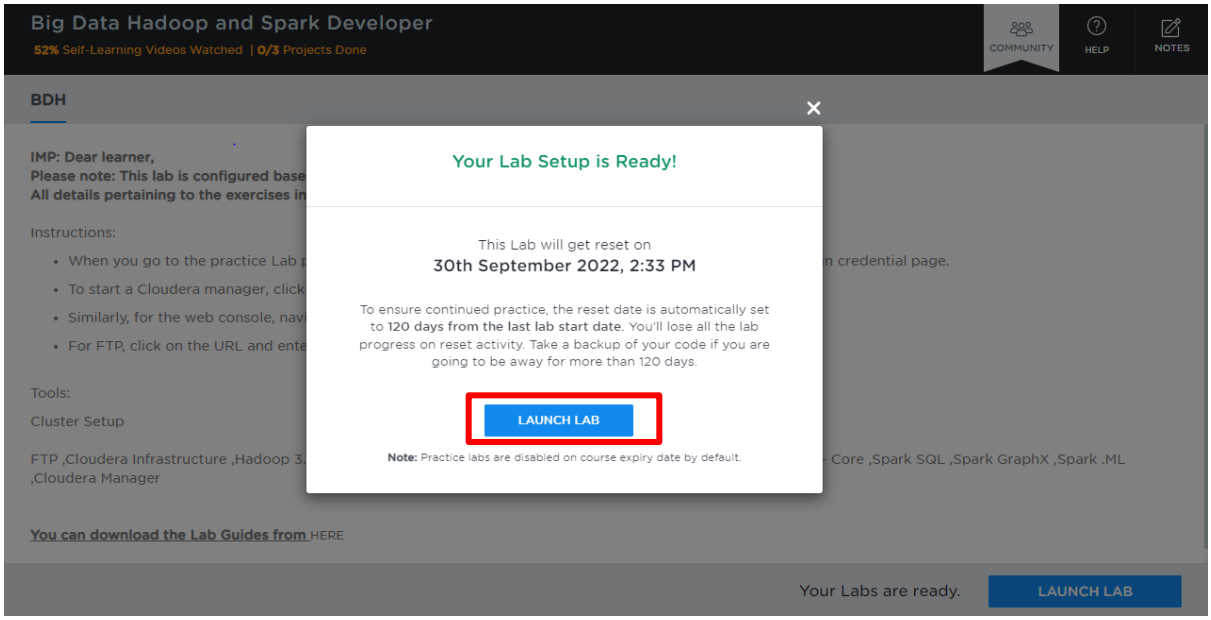
Step 4: Log in to your LMS account

Step 5: Open the course “**Big Data Hadoop and Spark Developer**”

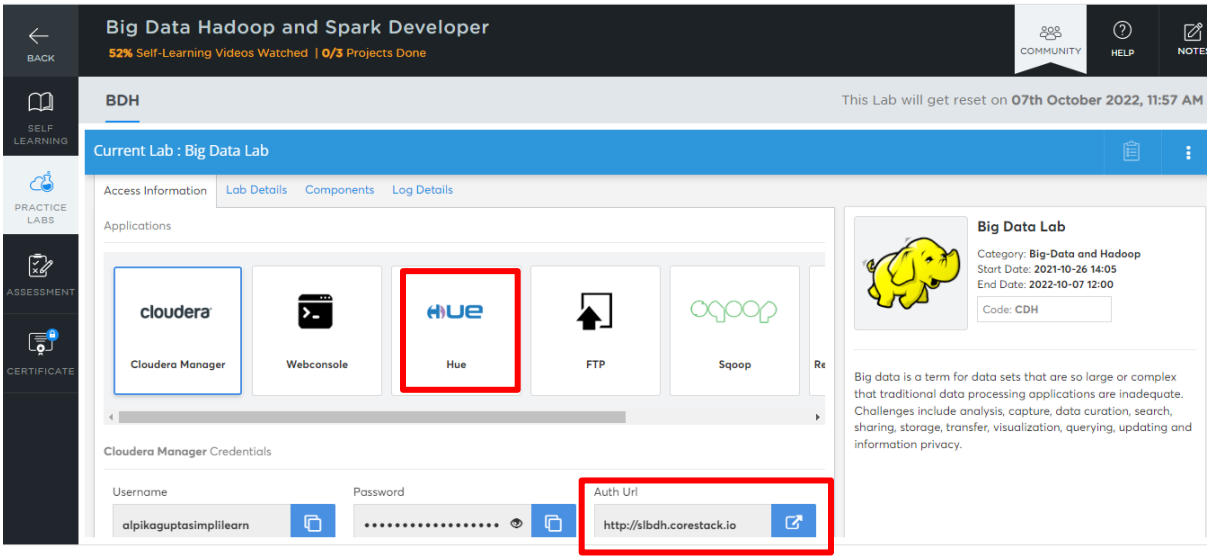
Step 6: Click on the “**PRACTICE LABS**” tab on the left side and select “**LAUNCH LAB**”



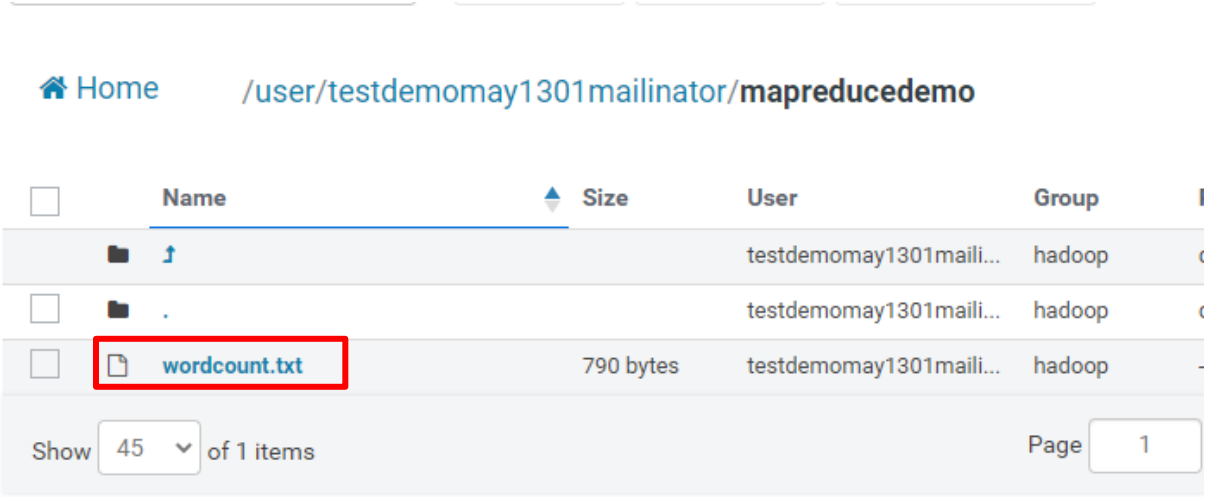
Step 7: Click on the "**LAUNCH LAB**" button



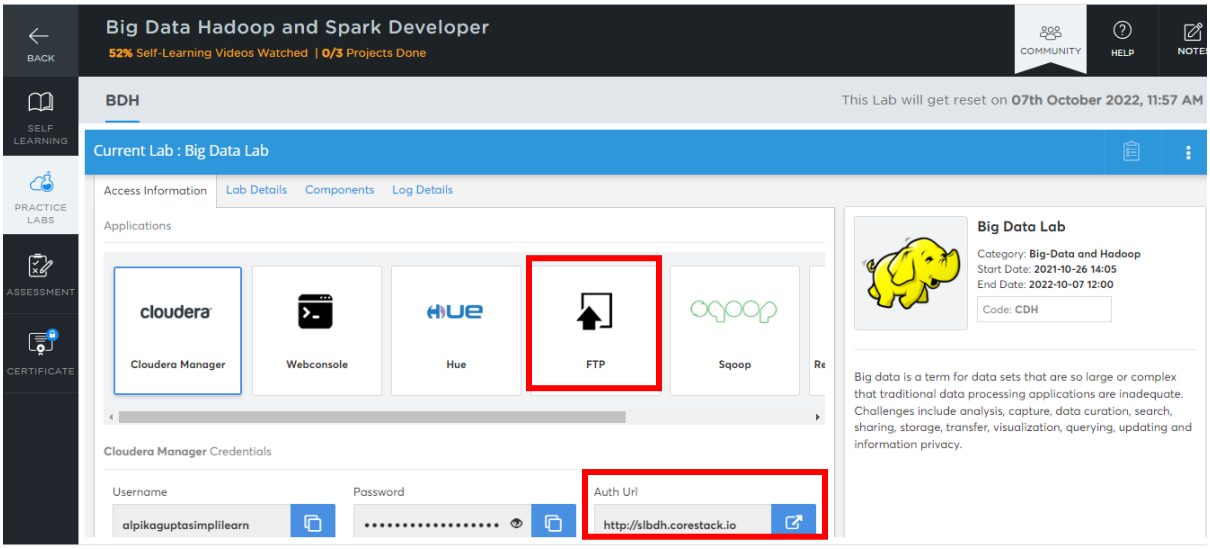
Step 8: Click on **"HUE"** to upload the datasets



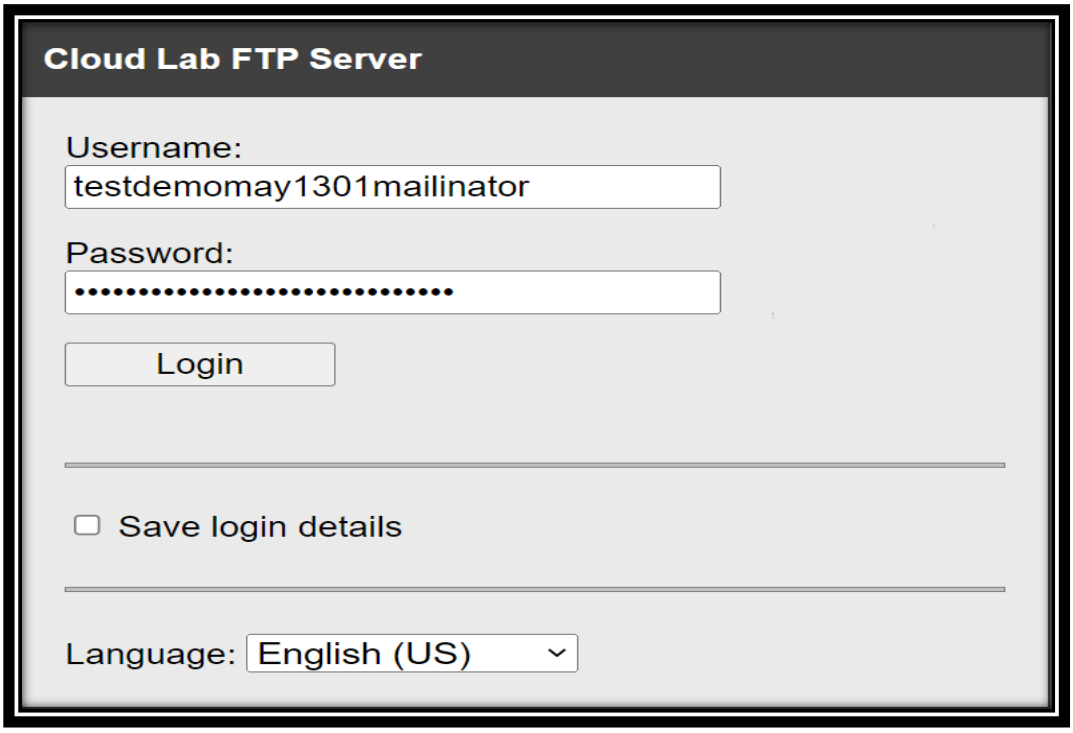
Step 9: Log in to "**HUE"** and click on create a directory named **"mapreducedemo"** and upload the **"wordcount.txt"** file into it



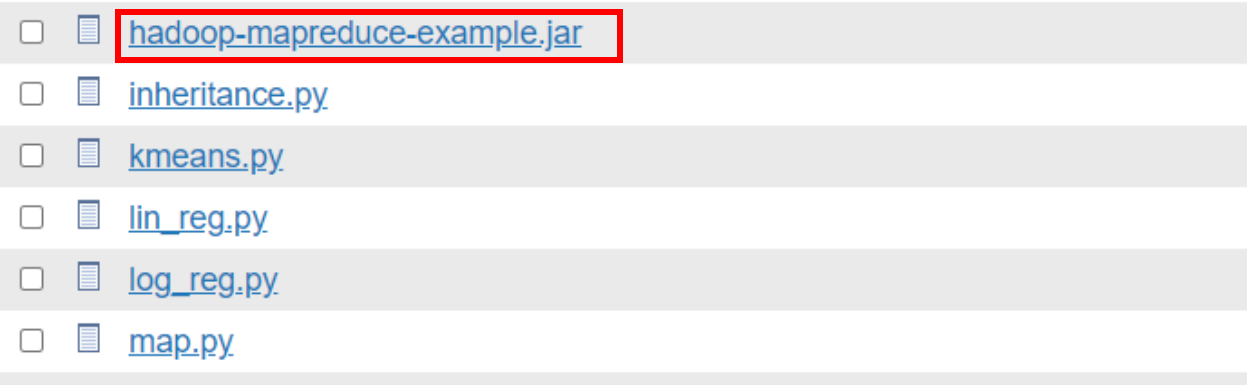
Step 10: Click on **"FTP"** and click on the **"Auth Url"** to upload the JAR file and copy the "**Username**" and "**Password**" provided to log in to **"FTP"**



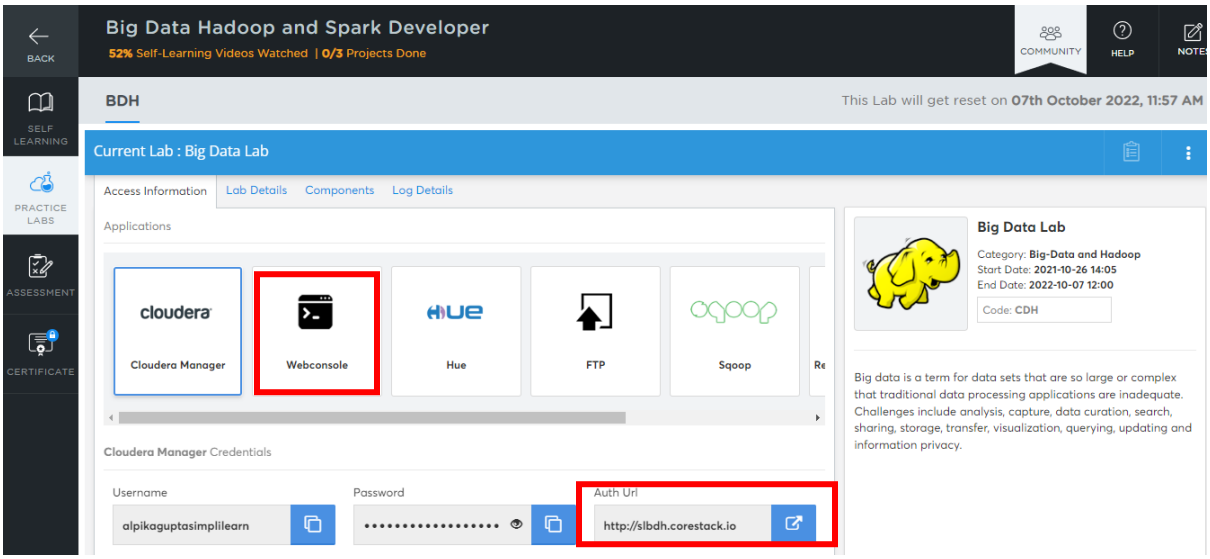
Step 11: Paste the **"Username"** and **"Password"** on the login window and click on **"Login"**



Step 12: Click on the **"Upload Files"** icon and upload the **"hadoop-mapreduce-example.jar"** file into FTP



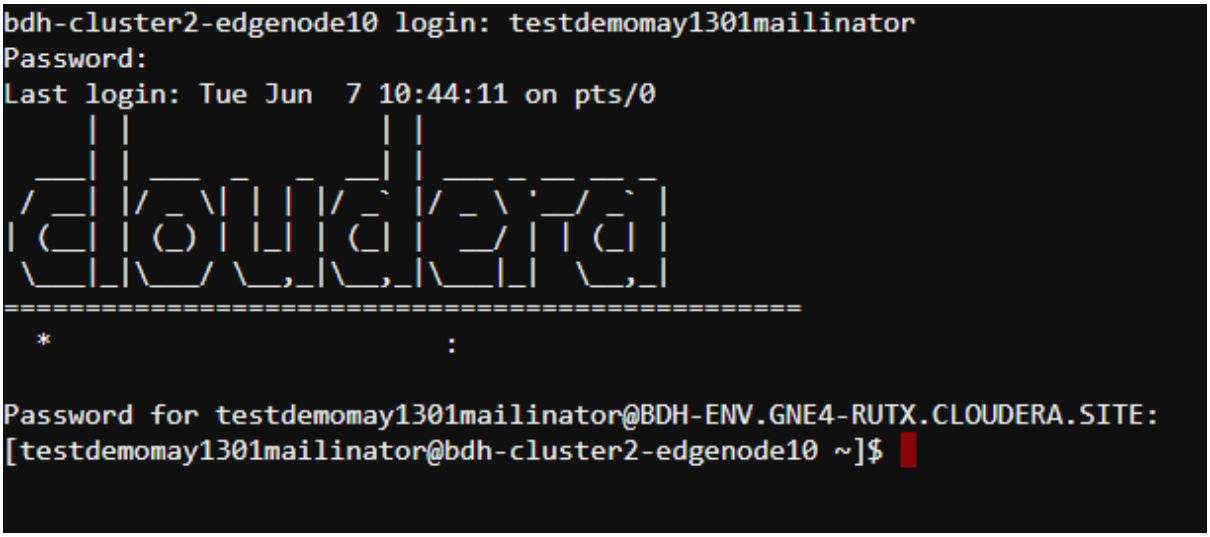
Step 13: Go back to the lab window and click on "**Webconsole**" and then on **"Auth Url"**



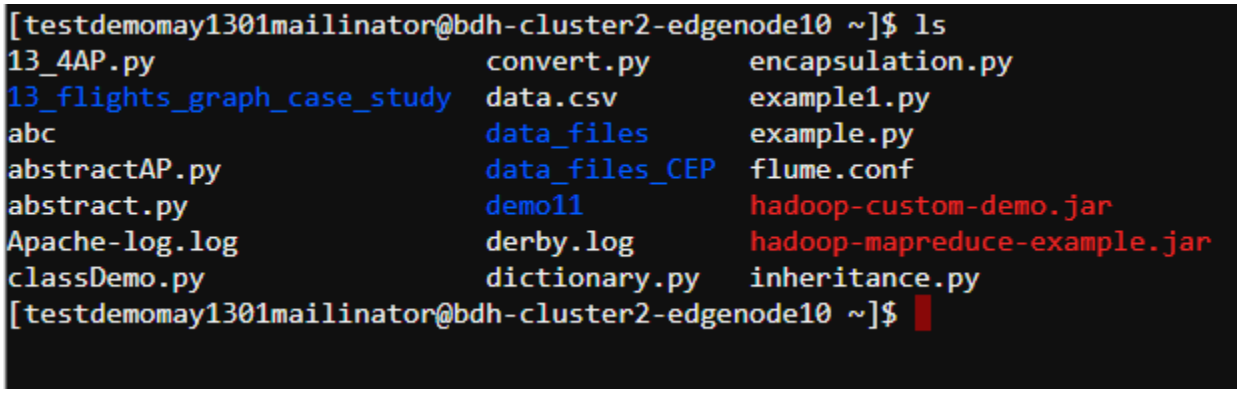
Step 14: Copy the "**Username**" and "**Password**" provided to log in to the **"Webconsole"**

Step 15: Paste the "**Username**" and "**Password**" on the console and click on enter

Note: The password will not be visible when pasted on the console.



Step 16: Make sure you have the **"hadoop-mapreduce-example.jar"** file present in FTP using the ls command

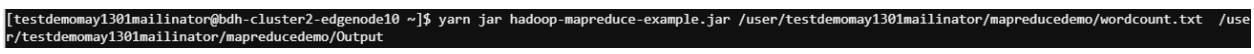


Step 17: Execute the below command and see if your job gets executed successfully

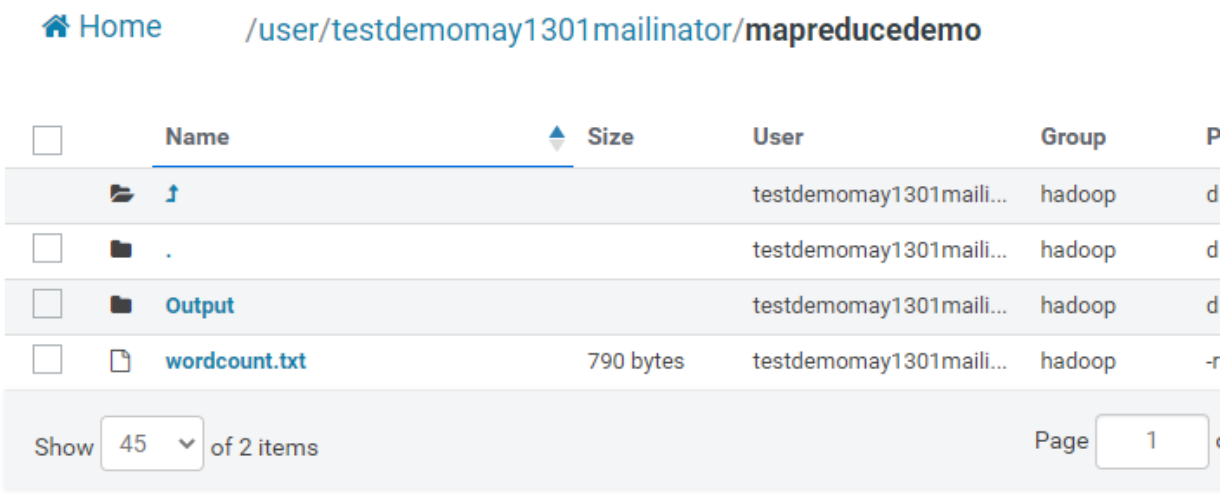
Note: Change the username of the Hadoop directory to "testdemomay1301mailinator" as assigned in your lab.

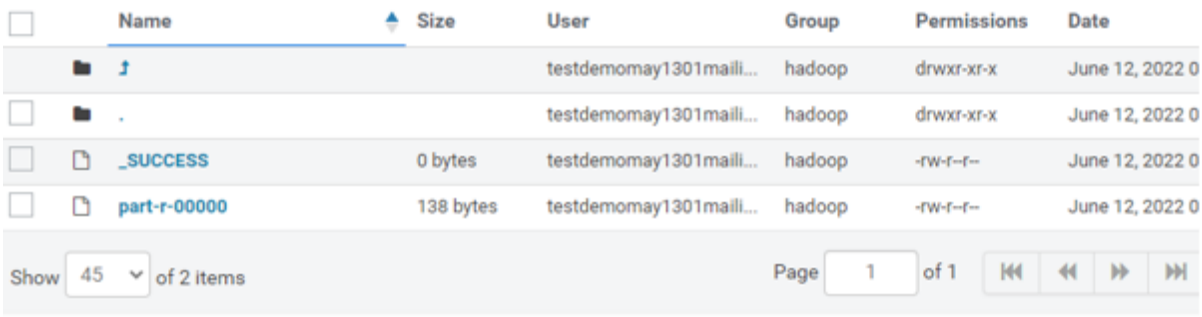
Command:

yarn jar hadoop-mapreduce-example.jar /user/testdemomay1301mailinator/mapreducedemo/wordcount.txt /user/testdemomay1301mailinator/mapreducedemo/Output



Step 18: Log in to **HUE** and open the **"mapreducedemo"** directory. You will see one more folder named **"Output"**





Note: You will be able to see the part files with the count of each word.

```
API     2
Dataset 2
Foundation,     1
Originally      1
RDD     3
architectural   1
as      2
codebase        1
data    2
dataset 1
deprecated.     1
developed       1
fault-tolerant  1
foundation      1
framework.      1
in      1
interface       2
it      1
later   1
parallelism     1
programming     2
```