Running a java program in Hadoop

1. Navigate to a Working Directory

```
cd ~
mkdir hadoop_projects
cd hadoop_projects
```

2. Create the Java File:

```
nano WordCount.java
```

3. Create a New File (e.g., input.txt):

```
nano input.txt
```

4. Paste the Java Code into WordCount.java and save the file:

```
import java.io.IOException;
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 {
         String[] tokens = value.toString().split("\\s+");
         for (String token : tokens) {
```

```
            word.set(token);
            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, "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);
}
}
```

5. Compile the Java Program:

```
javac -classpath $(hadoop classpath) -d . WordCount.java
```

6. Create a JAR File

```
jar cf wordcount.jar WordCount*.class
```

7. Start Hadoop Services

```
start-dfs.sh
start-yarn.sh
```

8. Create an Input Directory in HDFS:

```
hdfs dfs -mkdir -p /user/1910776142/input
```

9. Upload Your Input File to HDFS:

```
hdfs dfs -put input.txt  /user/1910776142/input
```

10. Run the Hadoop Job:

```
hadoop jar wordcount.jar WordCount /user/1910776142/input /user/1910776142/output
```

11. Check the Output:

```
hdfs dfs -cat /user/1910776142/output/part-r-00000
```

Note : In case of mapred-site.xml configuration error follow the steps below

1. Edit mapred-site.xml:

```
nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

2. Add the following properties if they are not already present, and ensure they are pointing to the correct Hadoop distribution directory.

```
<configuration>
  <property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=/path/to/your/hadoop/directory</value>
  </property>
  <property>
```

```
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=/path/to/your/hadoop/directory</value>
  </property>
  <property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=/path/to/your/hadoop/directory</value>
  </property>
</configuration>
```

3. Check Environment Variables:

```
echo $HADOOP_HOME
echo $HADOOP_MAPRED_HOME
```

4. If they are not set, add the following lines to your shell profile and reload it.

```
export HADOOP_HOME=/path/to/your/hadoop/directory
export HADOOP_MAPRED_HOME=$HADOOP_HOME
source ~/.bashrc
```

5. Restart Hadoop Services:

```
stop-dfs.sh
stop-yarn.sh
start-dfs.sh
start-yarn.sh
```

6. Run the Hadoop Job Again:

Hadoop Word Cound using Docker

Installation

1. Download and Install Docker Desktop (Personal):

   https://www.docker.com/products/docker-desktop/

2. Run Docker Desktop
3. Create a new directory anywhere (eg: /home/user/hadoop or E:\hadoop)
4. Open terminal from within the newly created directory.
5. **Run the command below to download the container:**

   ```
   docker run -p 9870:9870 -p 8088:8088 -v .:/home/hadoop/data -it
   --name=hadoop macio232/hadoop-pseudo-distributed-mode
   ```

6. Done.

Work

1. Start the container:

   ```
   docker start hadoop
   ```

2. Get inside the container:

   ```
   docker exec -it hadoop /bin/bash
   ```

Done.
Work:

1. Start the container:
   docker start hadoop

2. Get inside the container:
   docker exec -it hadoop /bin/bash

3. Start hadoop services:
   start-all.sh

4. Navigate to the Data Directory: Since you mounted the current directory to /home/hadoop/data (while installation), navigate there
   cd /home/hadoop/data

5. Create the WordCount.java File: Use a text editor like vi to create the Java file:
   vi WordCount.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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

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

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, "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);
}
}
```

Summary of Commands:

- ➔ **i**: Enter insert mode.
- ➔ **Esc**: Exit insert mode (back to normal mode).
- ➔ **:wq**: Save and quit.
- ➔ **:q!**: Quit without saving

6. Compile the Java Code:
   javac -classpath `hadoop classpath` -d . WordCount.java

7. Package the compiled classes into a JAR file:
   jar cf wordcount.jar WordCount*.class

8. Create a directory for the input data inside /home/hadoop/data:
   mkdir input

9. Create a sample text file:
   echo "Hello Hadoop Hello Docker" > input/file01.txt

10. Put the input data into HDFS
    hdfs dfs -mkdir -p /user/hadoop/input
    hdfs dfs -put ./input/* /user/hadoop/input/

11. Run the Hadoop job using:
    hadoop jar wordcount.jar WordCount /user/hadoop/input
    /user/hadoop/output

12. After the job completes, view the results:
    hdfs dfs -cat /user/hadoop/output/part-r-00000