

Complete Step-by-Step Guide to Running a Java Program on Hadoop

This guide will walk you through everything from setting up your Hadoop environment to running a Java MapReduce program on it.

◆ Step 1: Set Up Hadoop

Before running a Java program, you need to ensure that Hadoop is correctly installed and configured.

1.1 Start Hadoop Services

Run the following commands in your terminal:

```
start-dfs.sh
```

```
start-yarn.sh
```

This will start:

- **HDFS Services** (NameNode, DataNode, Secondary NameNode)
- **YARN Services** (ResourceManager, NodeManager)

1.2 Verify Services

Check if all Hadoop daemons are running:

```
jps
```

expected output:

NameNode

DataNode

SecondaryNameNode

ResourceManager

NodeManager

◆ Step 2: Prepare Your Java Program

Now, let's create a simple **WordCount** MapReduce program.

2.1 Create a New Java File

Open a terminal and create a new file:

```
nano WordCount.java
```

1. Copy and paste this code:

```
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> {
        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));
        }
    }

    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);
    }
}
```

1. Save the file (CTRL + X, then Y, then Enter).
-

◆ Step 3: Compile the Java Program

3.1 Set Up the Classpath

Since Hadoop has multiple JAR files, you need to specify them when compiling.

```
export HADOOP_CLASSPATH=$(hadoop classpath)
```

3.2 Compile the Java Code

```
javac -classpath $HADOOP_CLASSPATH -d . WordCount.java
```

3.3 Package it into a JAR File

```
jar -cvf wordcount.jar *.class
```

Now, your Java program is ready to run on Hadoop.

◆ Step 4: Prepare Input Data

4.1 Create an Input File

```
nano input.txt
```

Add the following sample text:

Hadoop is fun

Hadoop is powerful

I love Hadoop

Save (CTRL + X, Y, Enter).

Upload input file to hdfs

```
hdfs dfs -mkdir /input
```

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

Verify the file

```
hdfs dfs -ls /input
```

Output

Found 1 items

```
-rw-r--r--  1 username supergroup  34 2025-02-12  input/input.txt
```

Step 5: Run the Hadoop Job

Now, execute the MapReduce program using:

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

✦ Step 6: Check Output

6.1 View the Output Files

```
hdfs dfs -ls /output
```

6.2 Read the Output

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

Expected output:

Hadoop 3

I 1

is 2

love 1

powerful 1

fun 1

◆ Step 7: Web UI Monitoring

7.1 HDFS UI

- Open your browser and go to:
`http://localhost:9870`
- Click on "**Utilities**" → "**Browse the file system**"
- Navigate to `/input/` and `/output/` to see files.

7.2 YARN UI

- Open: `http://localhost:8088`
- Click "**Applications**" to monitor running jobs.

◆ Step 8: Clean Up (Optional)

If you want to rerun the job, delete the previous output:

```
hdfs dfs -rm -r /output
```