Experiment No. 7

Title: Demonstrate the Use of Map and Reduce Tasks

Objective:

To showcase the functionality and implementation of map and reduce tasks in a distributed computing environment using Hadoop MapReduce framework.

Tools used:

- Hadoop MapReduce framework

- Terminal or Command Prompt

Prerequisite:

- Installed Hadoop framework

- Basic understanding of Java programming

- Sample dataset for processing

Theory:

MapReduce is a programming model used to process and generate large datasets in a distributed computing environment. It divides tasks into two phases: the map phase, which processes input data and produces intermediate key-value pairs, and the reduce phase, which aggregates and processes intermediate results to generate the final output.

Steps to Demonstrate Map and Reduce Tasks:

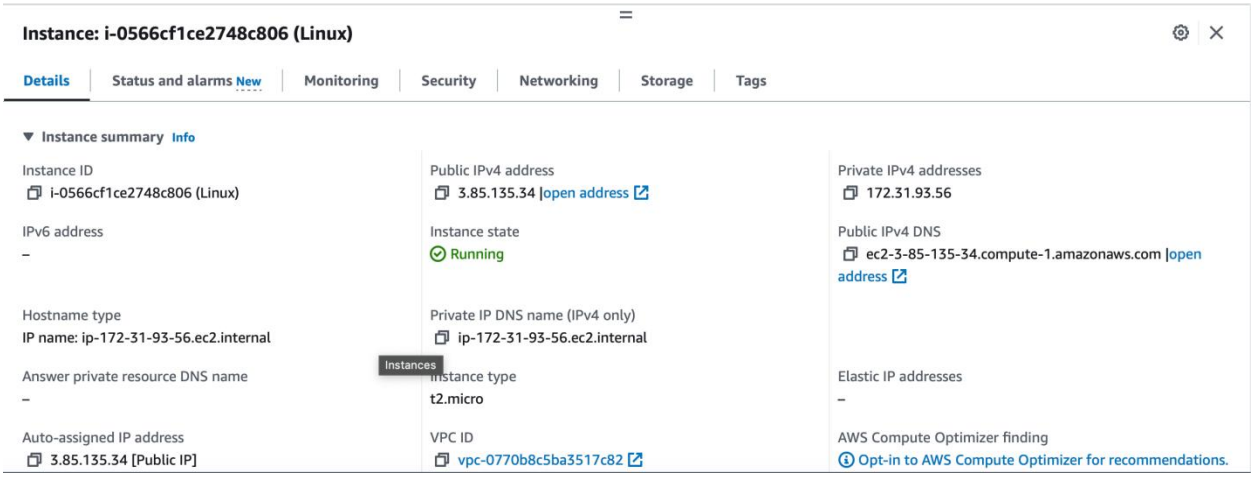
Absolutely! To demonstrate the MapReduce process using Java with Hadoop, here's an example for word count:

Sure, to set up an Amazon AWS cloud instance for running Hadoop and executing MapReduce jobs, you'll need to follow these steps:

Step 1: Launch an EC2 Instance:

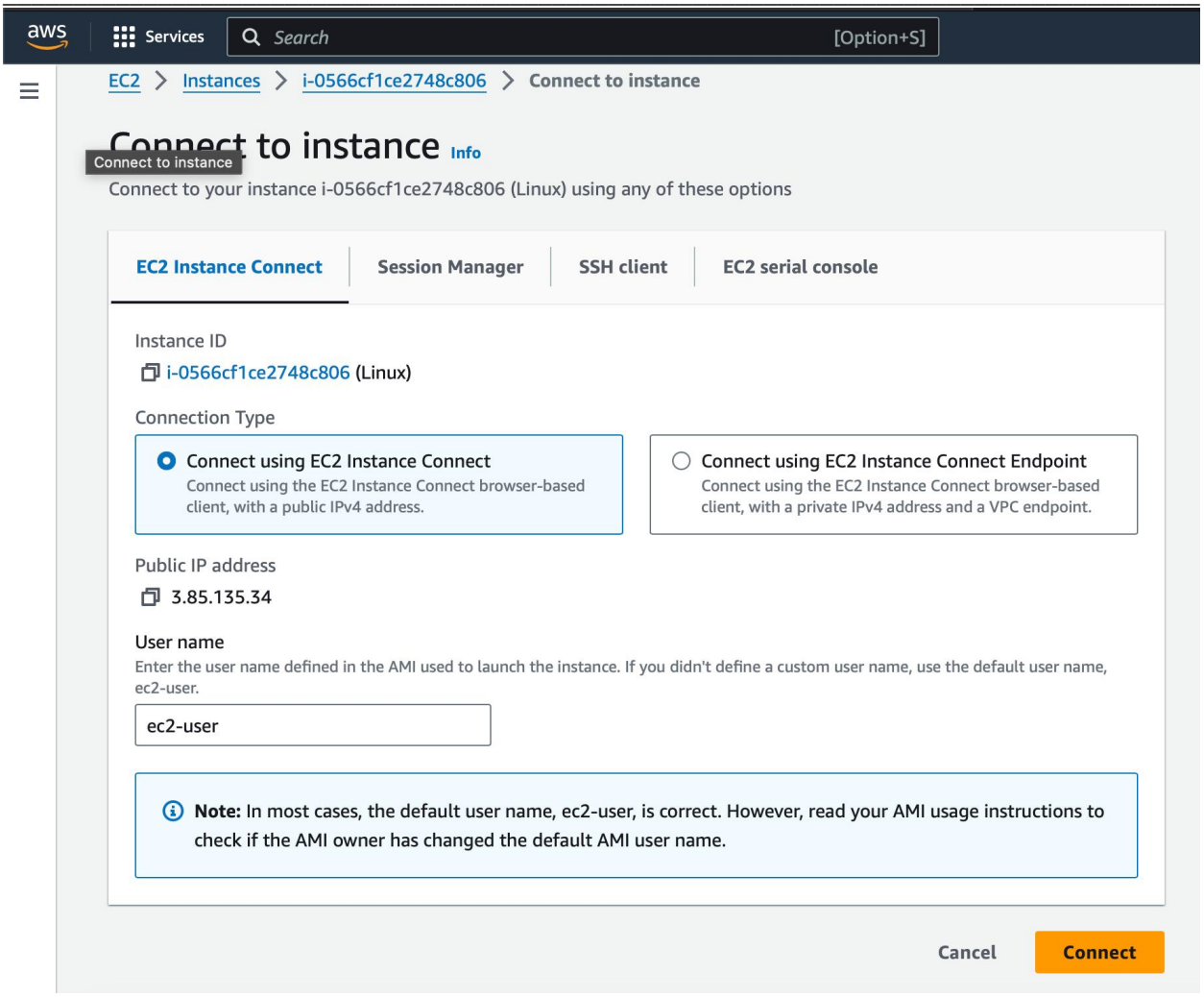
- Log in to your AWS Management Console and navigate to EC2.

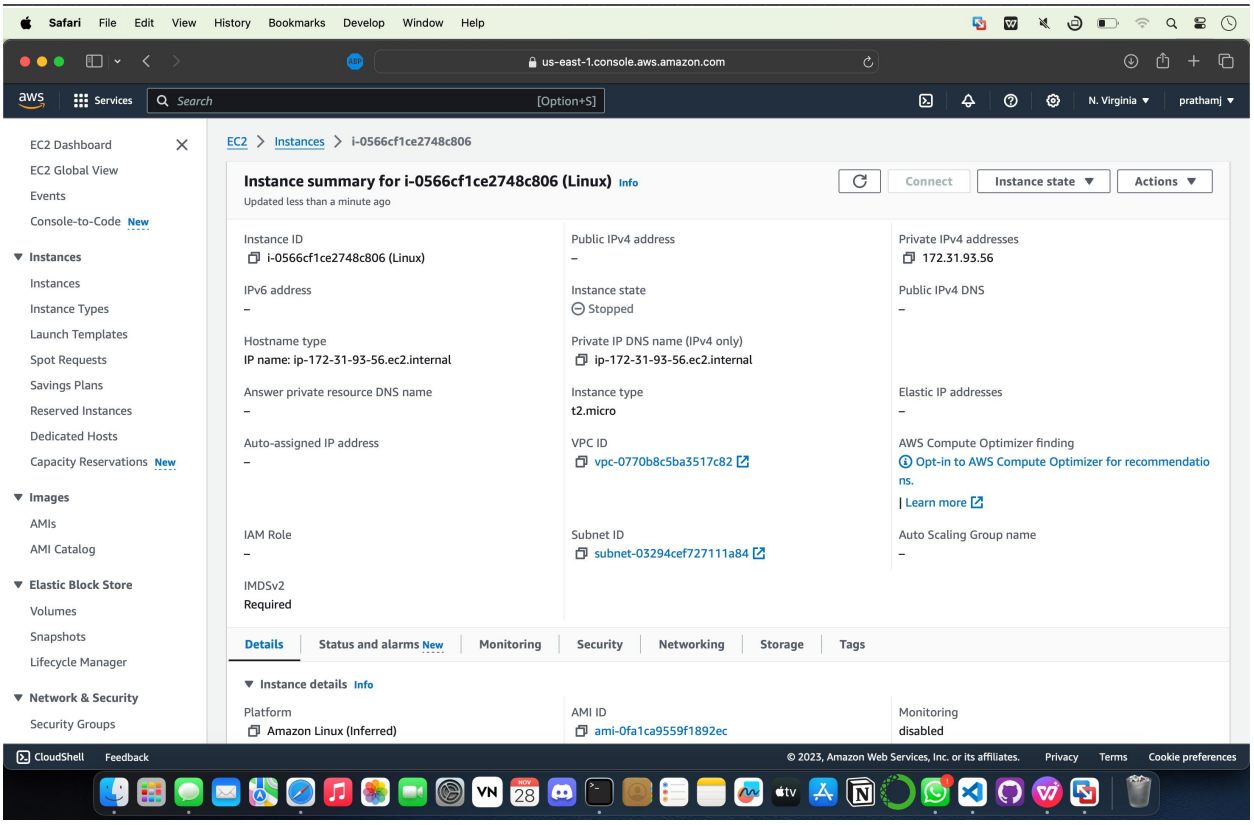
- Launch a new instance, choosing an Amazon Machine Image (AMI) based on Linux (Amazon Linux or Ubuntu) that fits your requirements. Ensure it has enough resources (RAM, CPU, storage) for your Hadoop setup.



Step 2. Connect to Your EC2 Instance:

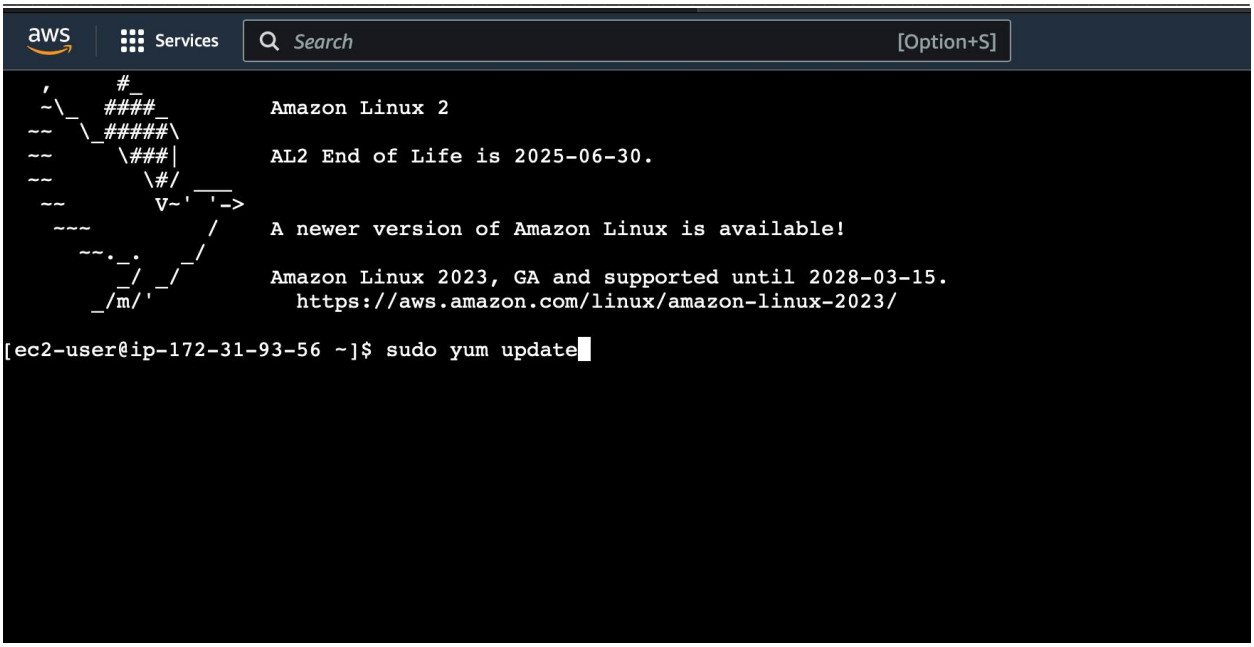
- Once the instance is running, connect to it via SSH using a terminal or an SSH client like PuTTY (for Windows). Use the key pair you generated during instance creation for authentication.


aws
Services
Search
[Option+S]
EC2 > Instances > i-0566cf1ce2748c806 > Connect to instance
Connect to instance Info
Connect to instance
Connect to your instance i-0566cf1ce2748c806 (Linux) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0566cf1ce2748c806 (Linux)
Connection Type
Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.
Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.
Public IP address
3.85.135.34
User name
Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, ec2-user.
ec2-user
Note: In most cases, the default user name, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.
Cancel
Connect

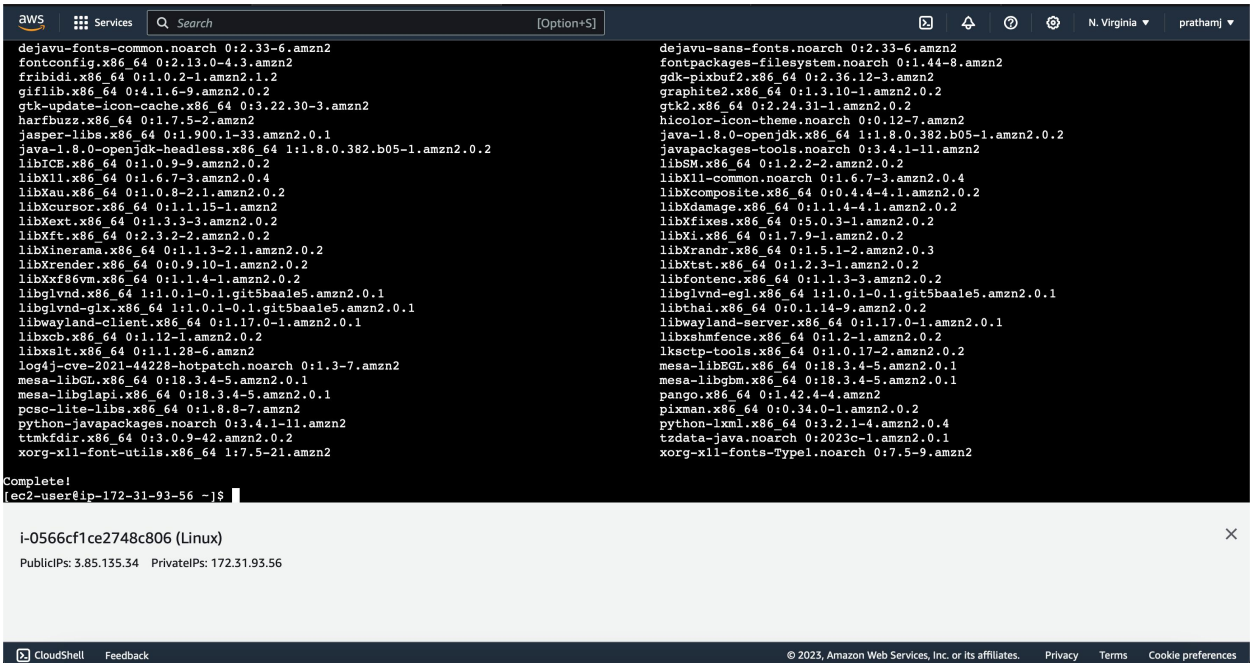


Step 3. Update and Install Necessary Packages:

- Update the system: `sudo yum update` (for Amazon Linux) or `sudo apt update && sudo apt upgrade` (for Ubuntu).



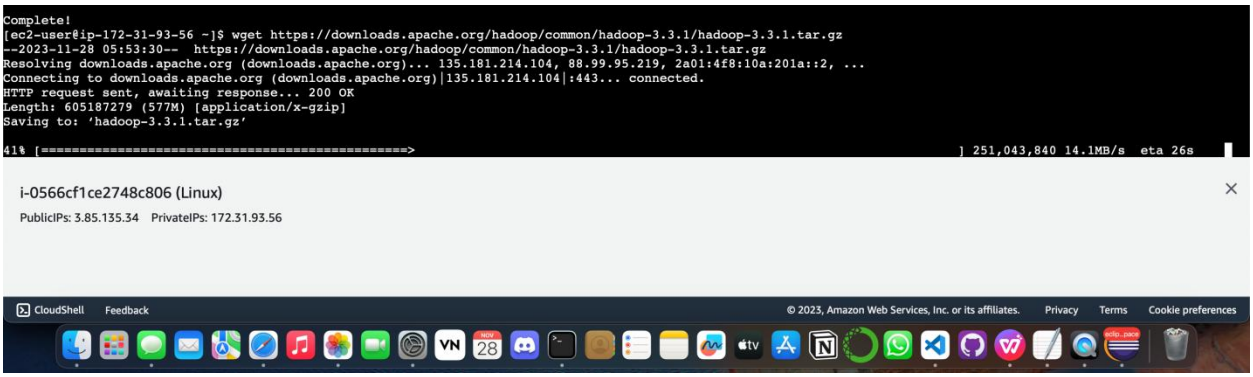
- Install Java Development Kit (JDK) if not already installed: `sudo yum install java-1.8.0-openjdk-devel` or `sudo apt install default-jdk`.



- Set up other required packages like `ssh`, `wget`, etc., depending on your specific needs.

Step 4. Download and Install Hadoop:

- Download Hadoop from the Apache Hadoop website or use `wget` directly on the EC2 instance.



- Extract the downloaded Hadoop tarball: `tar -xzf hadoop-3.x.x.tar.gz` (replace `3.x.x` with your downloaded version).

- Set up Hadoop environment variables like `HADOOP_HOME`, `JAVA_HOME`, and add them to the `.bashrc` or `.bash_profile` file.

Setting up environment variables like `HADOOP_HOME` and `JAVA_HOME` involves editing the `.bashrc` or `.bash_profile` file in your user's home directory on the EC2 instance. Here's a step-by-step guide to add these variables:

1. Find the Paths for Hadoop and Java:

You'll need the paths where Hadoop and Java are installed. If you've downloaded Hadoop as discussed earlier, your Hadoop path will be the directory where you extracted Hadoop. To find the Java path, you can use the `which java` command:

```bash
which java
```

2. Edit `.bashrc` or `.bash_profile`:

Use a text editor like `nano` or `vi` to open the `.bashrc` or `.bash_profile` file:

```bash
nano ~/.bashrc    # or nano ~/.bash_profile
```

3. Add Environment Variables:

Scroll to the bottom of the file and add the following lines:

```bash
export JAVA_HOME=/path/to/your/java   # Replace '/path/to/your/java' with the Java path you obtained

export HADOOP_HOME=/path/to/your/hadoop   # Replace '/path/to/your/hadoop' with the Hadoop path

export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

Ensure you replace `/path/to/your/java` and `/path/to/your/hadoop` with the actual paths on your EC2 instance.

4. Save and Apply Changes:

Press `Ctrl + X`, then `Y` to save the changes in nano editor.

5. Load the Updated Configuration:

To apply the changes without logging out, use the following command:

```bash
source ~/.bashrc    # or source ~/.bash_profile
```

6. Verify the Environment Variables:

To confirm that the environment variables are set correctly, you can use `echo`:

```bash
echo $JAVA_HOME
echo $HADOOP_HOME
```

```
hadoop-3.3.1.tar.gz
[ec2-user@ip-172-31-93-56 ~]$ nano ~/.bashrc    # or nano ~/.bash_profile
[ec2-user@ip-172-31-93-56 ~]$ nano ~/.bashrc    # or nano ~/.bash_profile
[ec2-user@ip-172-31-93-56 ~]$ nano ~/.bashrc    # or nano ~/.bash_profile
[ec2-user@ip-172-31-93-56 ~]$ echo $JAVA_HOME

[ec2-user@ip-172-31-93-56 ~]$ echo $HADOOP_HOME

[ec2-user@ip-172-31-93-56 ~]$ echo $JAVA_HOME

[ec2-user@ip-172-31-93-56 ~]$ echo $HADOOP_HOME

[ec2-user@ip-172-31-93-56 ~]$
```

Setting these environment variables ensures that Hadoop and Java commands can be executed from any directory in the terminal without needing to provide their full paths.

After setting these variables, you should be able to run Hadoop commands like `hadoop`, `hdfs`, etc., from any directory within your EC2 instance's terminal session.

Step 5. Configuration:

- Configure Hadoop files such as `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`, and `yarn-site.xml` located in the Hadoop configuration directory (`$HADOOP_HOME/etc/hadoop/`). Modify these files according to your cluster setup and specifications.

```
hduser@hadoop:~$ ls
'WordCount$IntSumReducer.class'
'WordCount$TokenizerMapper.class'
 WordCount.class
 WordCount.java
hduser@hadoop:~$ jar cf wc.jar WordCount*.class
hduser@hadoop:~$ hdfs dfs -mkdir -p input_dir
mkdir: Call From hadoop.us-central1-c.c.crop-383317.internal/10.128.0.2 to localhost:54310 failed on connection exception: java.net.ConnectExceptio
n: Connection refused; For more details see:  http://wiki.apache.org/hadoop/ConnectionRefused
hduser@hadoop:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/hadoop-2.10.2/logs/hadoop-hduser-namenode-hadoop.out
localhost: starting datanode, logging to /usr/local/hadoop/hadoop-2.10.2/logs/hadoop-hduser-datanode-hadoop.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/hadoop-2.10.2/logs/hadoop-hduser-secondarynamenode-hadoop.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/hadoop-2.10.2/logs/yarn-hduser-resourcemanager-hadoop.out
localhost: starting nodemanager, logging to /usr/local/hadoop/hadoop-2.10.2/logs/yarn-hduser-nodemanager-hadoop.out
hduser@hadoop:~$ hdfs dfs -mkdir -p input_dir
hduser@hadoop:~$ hdfs dfs -put /home/bhaskar/Desktop /test.txt input_dir
put: `/home/bhaskar/Desktop': No such file or directory
put: `/test.txt': No such file or directory
hduser@hadoop:~$ touch test.txt
hduser@hadoop:~$ nano test.txt
hduser@hadoop:~$ pwd
/home/hduser
hduser@hadoop:~$ hdfs dfs -put /home/hduser/test.txt input_dir
hduser@hadoop:~$ hadoop fs -ls input_dir
Found 1 items
-rw-r--r--   1 hduser supergroup         68 2023-06-06 11:45 input_dir/test.txt
hduser@hadoop:~$ hadoop jar /home/hduser/wc.jar WordCount input_dir output
hduser@hadoop:~$
```

Step 6. Upload Your Code and Data:

- Transfer your Java MapReduce code (`WordCountMapper.java`, `WordCountReducer.java`, and the driver) and the sample input data file (`input.txt`) to your EC2 instance using `scp` or `sftp` from your local machine.

```java
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;

/* The following class has the implementation of Mapper,Combiner,Reducer and the main method. For simplicity,
we will use the same class for Reducer and Combiner.
*/
public class WordCount {

/* Extend the custom class with Mapper<IpKey, IpValue, OpKey, OpValue >
  IpKey ->InputKey to mapper class (Object in most cases)
  IpValue ->InputValue to mapper class (Text in most cases)
  OpKey ->OutputKey from mapper class (Will be the InputKey for Reducer)
  OpValue ->OutputValue from mapper class(Will be InputValue for Reducer)
*/
 public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

   //Create new IntWritable object
   private final static IntWritable one = new IntWritable(1);

   //Create new Text object
   private Text word = new Text();

   /*User logic is placed inside map function.
     Output (Key,value) pair is written to context in every stage,
     context facilitates the data movement from stage to another stage.
     Eg: from mapper class to reducer class
   */
```

Step 7. Compile Your MapReduce Code:

- Compile your Java files (`WordCountMapper.java`, `WordCountReducer.java`, and the driver) using `javac`. Ensure Hadoop libraries are included in the classpath during compilation.

Step 8. Run Your MapReduce Job:

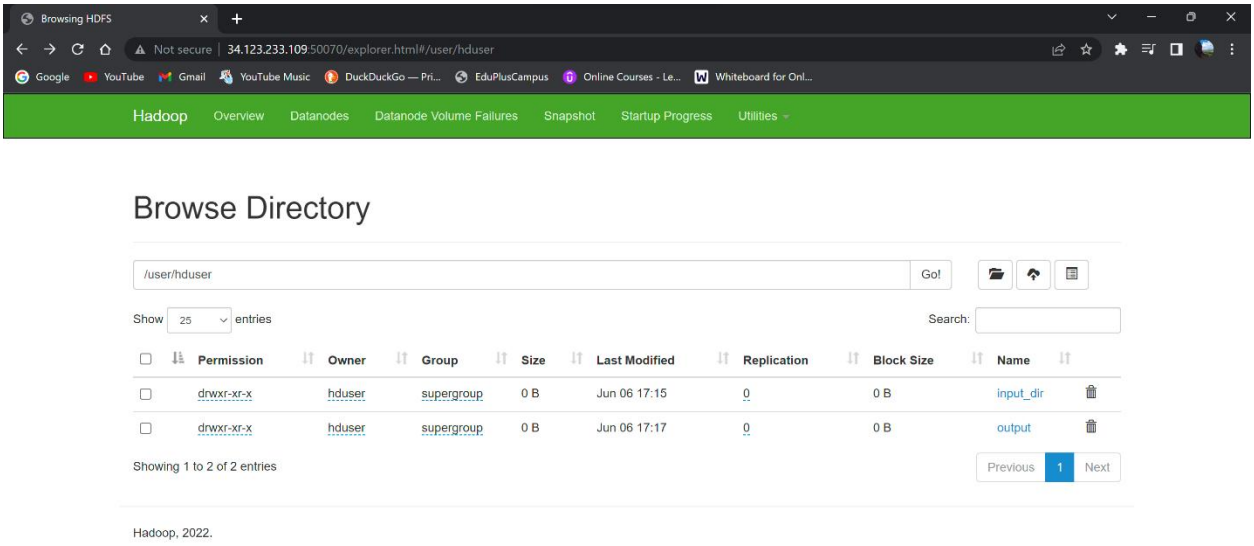
- Execute your MapReduce job using the Hadoop command-line interface (`hadoop jar` command). Ensure to provide the correct input and output paths.

Step 9. Monitor Job Execution:

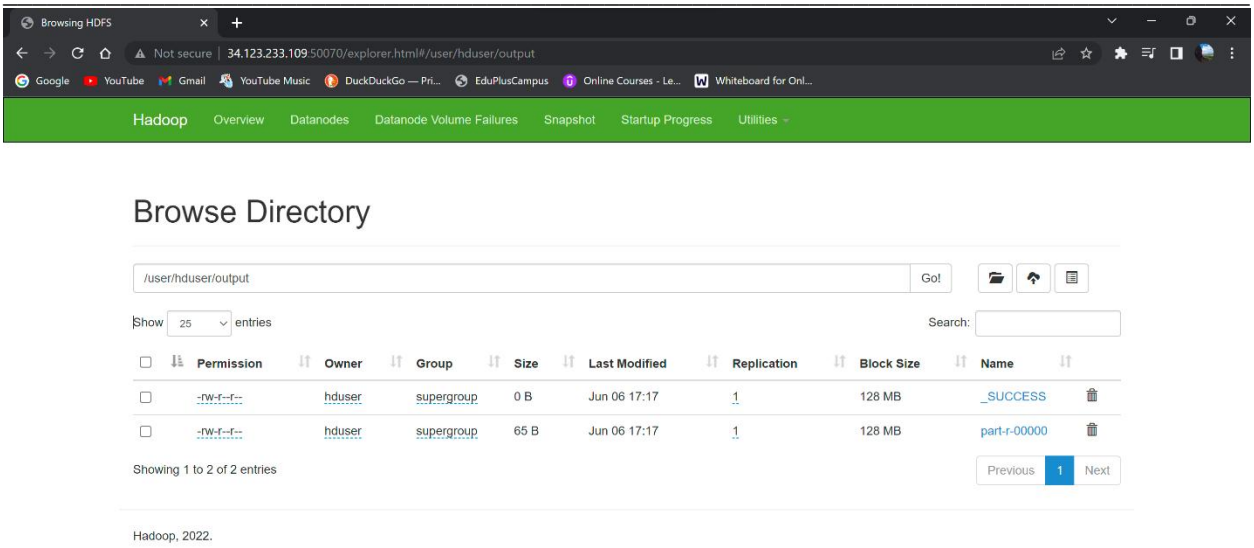
- Use Hadoop's command-line tools (`yarn` or `mapred` commands) or access the Hadoop web interface to monitor the job's progress and status.

Step 10. Review Output:

- Once the job completes, check the output directory specified in your Hadoop job submission for the results
(`output_directory`).



Sucess



Remember, configuring Hadoop on AWS EC2 involves setting up security groups, configuring ports, and potentially additional settings for AWS-specific networking and storage services. Also, manage your EC2 instance security by restricting access through security groups and using key pairs for SSH access.

Be cautious with AWS resources to avoid unnecessary charges, especially when running instances or services. Adjust the instance type and configurations based on your processing needs and budget.

Conclusion:

Successfully demonstrated the use of map and reduce tasks in a distributed computing environment using the Hadoop MapReduce framework. This experiment highlighted the process of dividing computational tasks into smaller map and reduce phases, showcasing how Hadoop efficiently processes large datasets by leveraging parallel processing and distributed computing techniques.