VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING

AND INFORMATION SYSTEMS

FALL SEMESTER 2024-2025

PMCA506L – CLOUD COMPUTING

DIGITAL ASSIGNMENT – 2

SUBMITTED ON: 23 – NOV - 2024

SUBMITTED BY-

AKASH KUMAR BANIK

PROGRAM: MCA

REGISTER No.: 24MCA0242

INSTALLATION AND CONFIGURATION OF JAVA 8 AND HADOOP 3.3.6 ON XUBUNTU (VIRTUALBOX)

STEP 1: INSTALLING JAVA 8

1.1 UPDATING THE SYSTEM

I began by ensuring my package manager was up to date. I opened the terminal in my xubuntu (light-weight version of ubuntu) virtual system and ran the command:

sudo apt update

This command refreshed the package repository information, allowing me to install the latest version of Java available in the repository.

1.2 INSTALLING JAVA

Next, I proceeded to install OpenJDK 8, which is the open-source implementation of the Java Platform. I executed the command:

sudo apt install openjdk-8-jdk

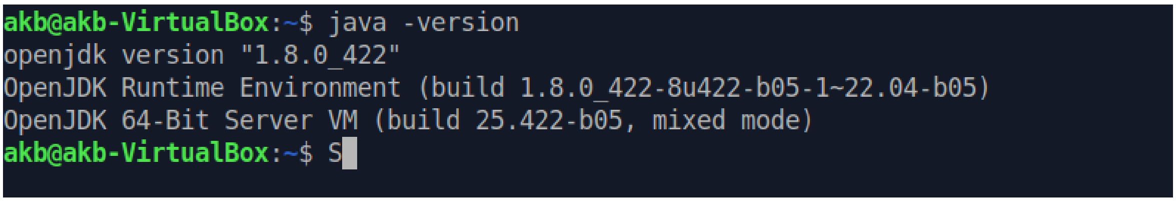
This command downloaded and installed the Java Development Kit (JDK) version 8. I could see the installation process in the terminal, where it fetched the required packages and set everything up.

When prompted, I pressed Y to allow the installation to proceed.

After the installation was complete, I wanted to confirm that Java was installed correctly. I checked the installed version by running the command:

java -version

The output displayed like this:



```
akb@akb-VirtualBox:~$ java -version
openjdk version "1.8.0_422"
OpenJDK Runtime Environment (build 1.8.0_422-8u422-b05-1~22.04-b05)
OpenJDK 64-Bit Server VM (build 25.422-b05, mixed mode)
akb@akb-VirtualBox:~$ S
```

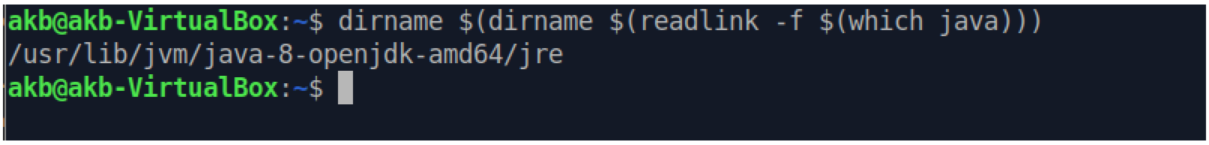
This output indicated that Java 8 was successfully installed on my virtual system.

1.3 SETTING UP JAVA_HOME

Hadoop needs to know where Java is installed, so I had to set the JAVA_HOME environment variable. To find where Java was installed, I used:

dirname $(dirname $(readlink -f $(which java)))

This gave me the path /usr/lib/jvm/java-8-openjdk-amd64/jre.



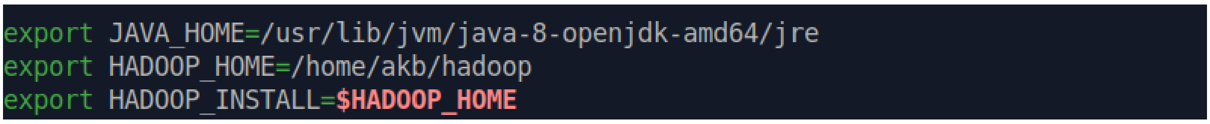
```
akb@akb-VirtualBox:~$ dirname $(dirname $(readlink -f $(which java)))
/usr/lib/jvm/java-8-openjdk-amd64/jre
akb@akb-VirtualBox:~$ 
```

I then added this path to the .bashrc file. To do that, I opened .bashrc using:

nano ~/.bashrc

At the bottom of the file, I added:

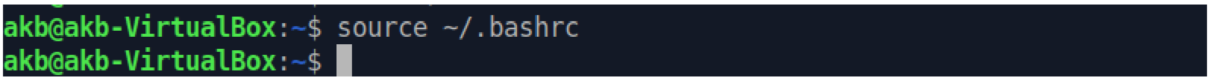
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64



```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
export HADOOP_HOME=/home/akb/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
```

After saving the file, I reloaded the settings with the command:

source ~/.bashrc



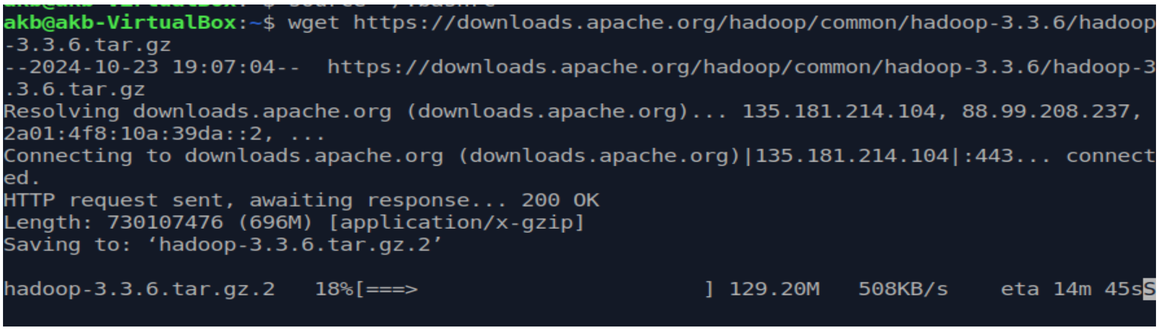
```
akb@akb-VirtualBox:~$ source ~/.bashrc
akb@akb-VirtualBox:~$ 
```

STEP 2: INSTALLING HADOOP 3.3.6

2.1 DOWNLOADING HADOOP

Next, I moved on to downloading Hadoop. I went to the Hadoop website and copied the download link for **Hadoop 3.3.6**. Then, in the terminal, I used wget to download hadoop:

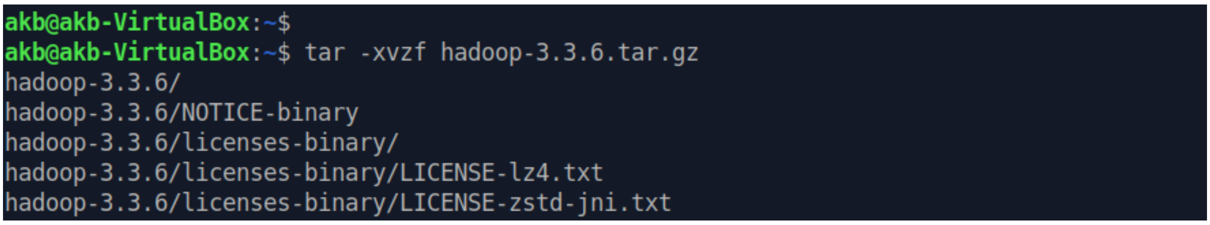
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz



```
akb@akb-VirtualBox:~$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop
-3.3.6.tar.gz
--2024-10-23 19:07:04--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3
.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 135.181.214.104, 88.99.208.237,
2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|135.181.214.104|:443... connect
ed.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: ‘hadoop-3.3.6.tar.gz.2’

hadoop-3.3.6.tar.gz.2   18%[===>                      ] 129.20M   508KB/s    eta 14m 45s
```

The download took some time, but after it was done, I unzipped the file using:

tar -xvzf hadoop-3.3.6.tar.gz



```
akb@akb-VirtualBox:~$
akb@akb-VirtualBox:~$ tar -xvzf hadoop-3.3.6.tar.gz
hadoop-3.3.6/
hadoop-3.3.6/NOTICE-binary
hadoop-3.3.6/licenses-binary/
hadoop-3.3.6/licenses-binary/LICENSE-lz4.txt
hadoop-3.3.6/licenses-binary/LICENSE-zstd-jni.txt
```

2.2 RENAMING THE HADOOP DIRECTORY

To make things easier to navigate, I renamed the folder from hadoop-3.3.6 to just hadoop:

mv hadoop-3.3.6 hadoop

2.3 SETTING UP HADOOP ENVIRONMENT VARIABLES

Now, I needed to set up environment variables for Hadoop. So, I edited the .bashrc file again using:

nano ~/.bashrc

At the bottom, I added the following lines to set up the Hadoop paths:

export HADOOP_HOME=~/hadoop

export HADOOP_INSTALL=$HADOOP_HOME

export HADOOP_MAPRED_HOME=$HADOOP_HOME

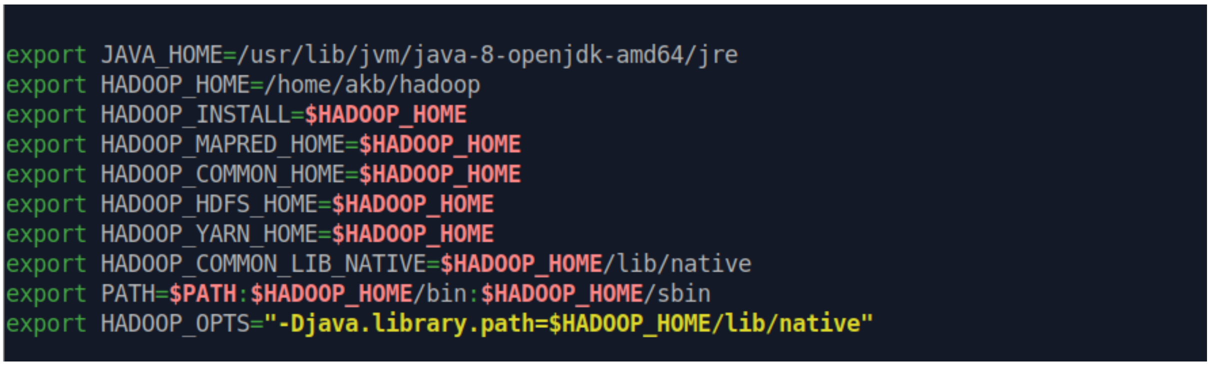
export HADOOP_COMMON_HOME=$HADOOP_HOME

export HADOOP_HDFS_HOME=$HADOOP_HOME

export HADOOP_YARN_HOME=$HADOOP_HOME

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native

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



```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
export HADOOP_HOME=/home/akb/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

I saved the file and ran the command: *source ~/.bashrc*

This applied the new environment settings for Hadoop.

STEP 3: CONFIGURING HADOOP

Now that Hadoop was installed, I needed to configure several important files located in the ~/hadoop/etc/hadoop/ directory for the functioning of MapReduce properly.

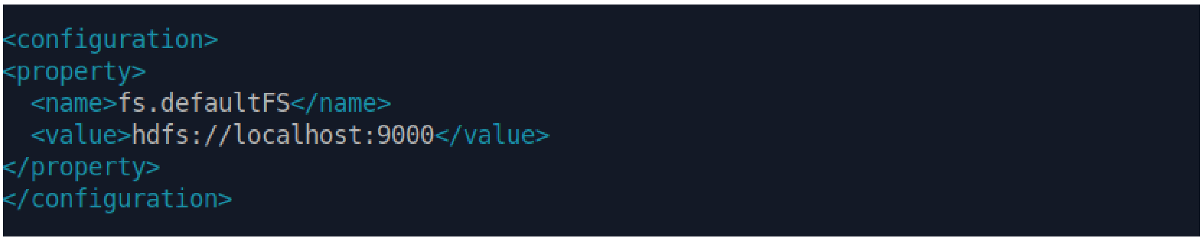
3.1 CONFIGURING CORE-SITE.XML

First, I configured the **core-site.xml** file, which defines the default file system:

nano ~/hadoop/etc/hadoop/core-site.xml

Inside this file, I added:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```



```
<configuration>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

This ensures that Hadoop knows to use the localhost as the default file system for its operations.

3.2 CONFIGURING HDFS-SITE.XML

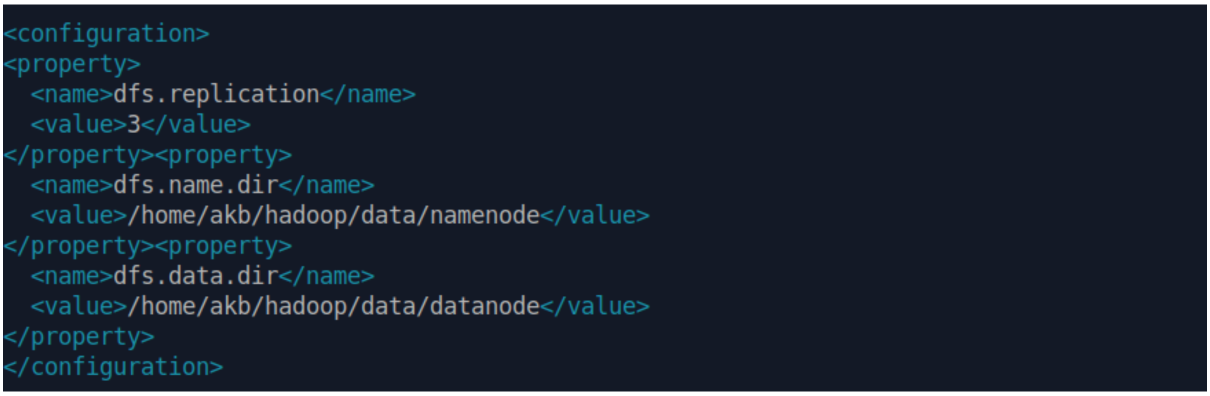
Next, I configured the **hdfs-site.xml** file to set up the HDFS (Hadoop Distributed File System) inside the Hadoop directory:

nano hdfs-site.xml

I added these properties:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
```

```
</property>
<property>
  <name>dfs.name.dir</name>
  <value>/home/akb/hadoop/data/namenode</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/home/akb/hadoop/data/datanode</value>
</property>
```



```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property><property>
  <name>dfs.name.dir</name>
  <value>/home/akb/hadoop/data/namenode</value>
</property><property>
  <name>dfs.data.dir</name>
  <value>/home/akb/hadoop/data/datanode</value>
</property>
</configuration>
```

I created the necessary directories for the NameNode and DataNode:

```
mkdir -p ~/hadoop/data/namenode
mkdir -p ~/hadoop/data/datanode
```

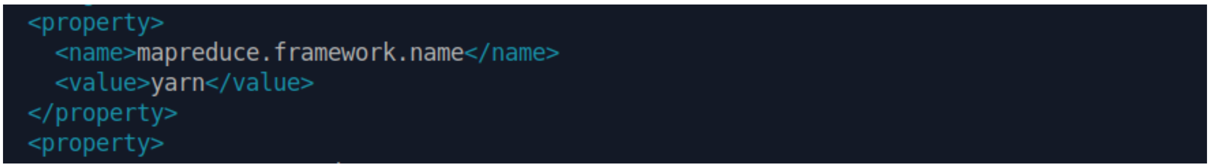
3.3 CONFIGURING MAPRED-SITE.XML

I then configured the MapReduce framework by configuring the mapred-site.xml file in the hadoop directory using the command:

```
nano mapred-site.xml
```

Inside, I added the properties:

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```



```
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
```

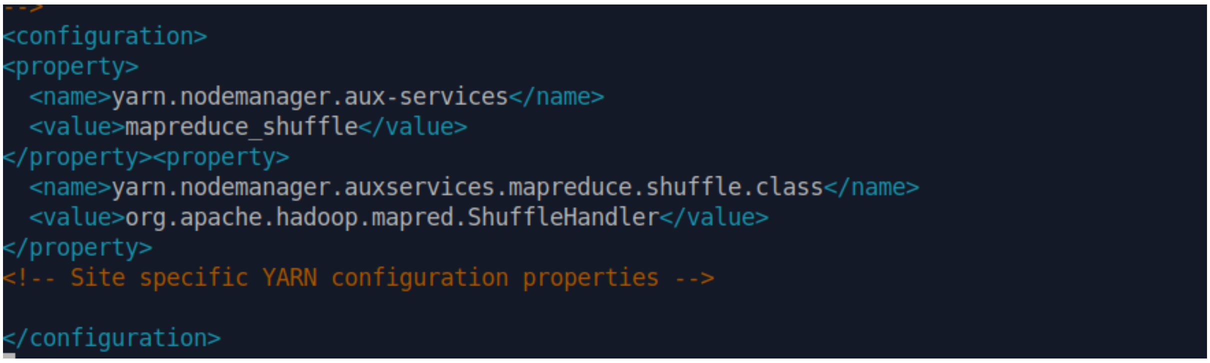
3.4 Configuring yarn-site.xml

Finally, I configured **YARN**, Hadoop's resource manager inside the Hadoop directory using:

nano yarn-site.xml

I added these properties:

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```



```
-->
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property><property>
  <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<!-- Site specific YARN configuration properties -->

</configuration>
```

STEP 4: SETTING UP SSH

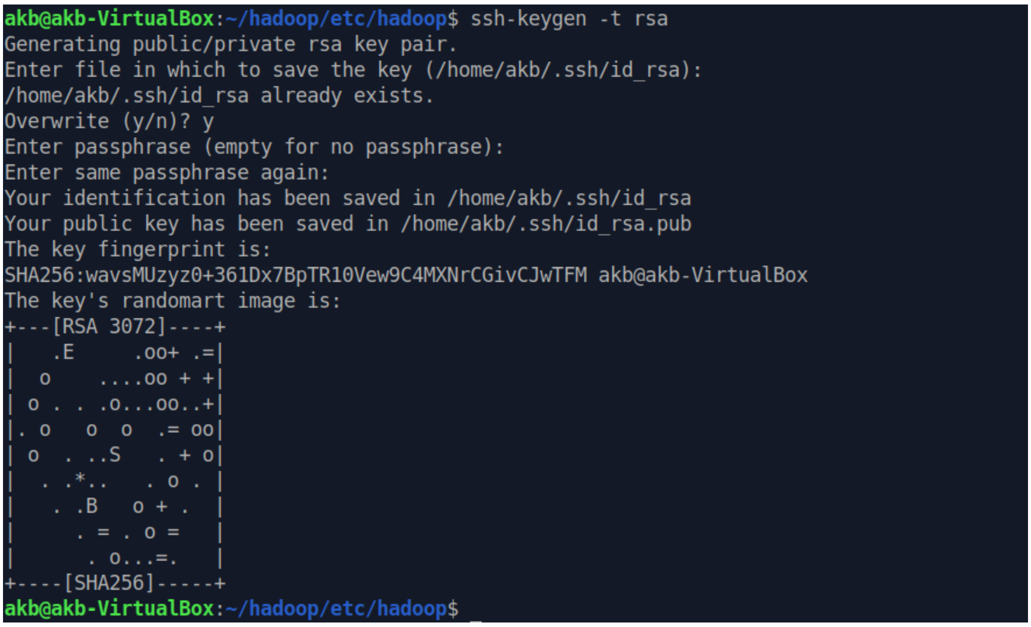
Hadoop requires SSH to communicate between different nodes. I set up SSH on my virtual machine as follows:

4.1 GENERATING SSH KEY

I created an SSH key without a password by running:

ssh-keygen -t rsa

And pressed Enter on each prompted asked, until the ssh key is generated.



```
akb@akb-VirtualBox:~/hadoop/etc/hadoop$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/akb/.ssh/id_rsa):
/home/akb/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/akb/.ssh/id_rsa
Your public key has been saved in /home/akb/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:wavsMUzyz0+361Dx7BpTR10Vew9C4MXNrCGivCJwTFM akb@akb-VirtualBox
The key's randomart image is:
+---[RSA 3072]----+
|   .E     .oo+ .=|
|  o    ....oo + +|
| o . . .o...oo..+|
|. o   o  o  .= oo|
| o  . ..S   . + o|
|  . .*..   . o . |
|   . .B   o + .  |
|     . = . o =   |
|      . o...=.   |
+----[SHA256]-----+
akb@akb-VirtualBox:~/hadoop/etc/hadoop$ _
```

4.2 AUTHORIZING SSH KEY

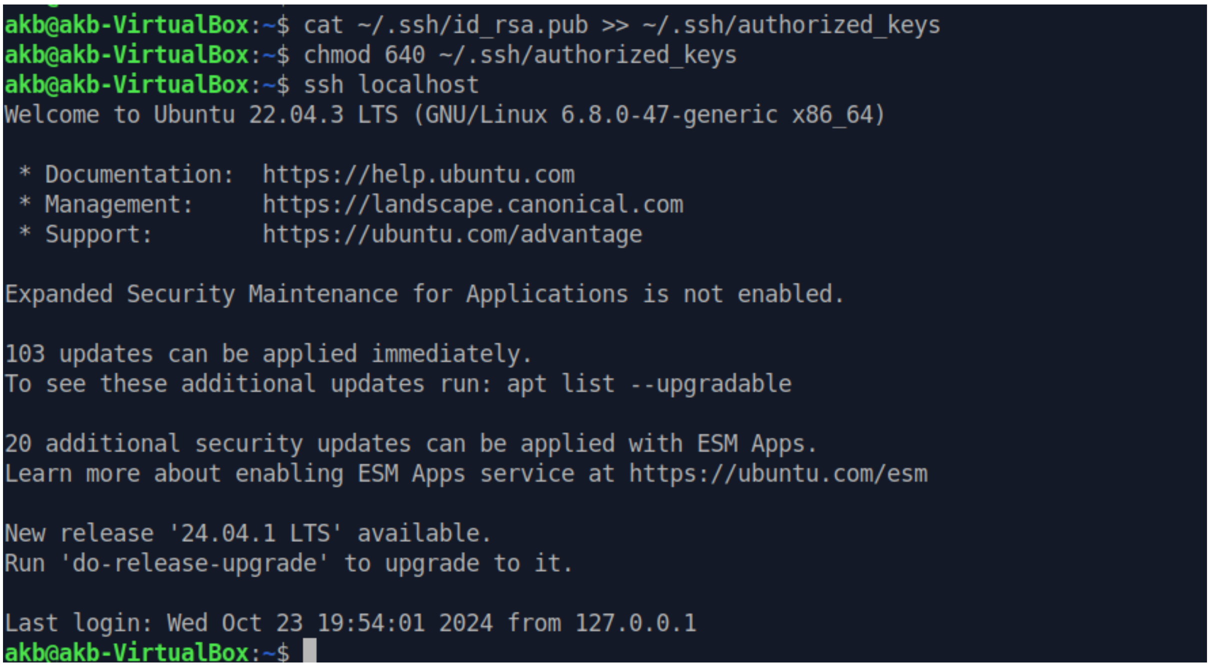
Then, I authorized the SSH key using the commands:

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

chmod 640 ~/.ssh/authorized_keys

4.3 TESTING SSH

I tested my SSH setup by running the command: *ssh localhost*



```
akb@akb-VirtualBox:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
akb@akb-VirtualBox:~$ chmod 640 ~/.ssh/authorized_keys
akb@akb-VirtualBox:~$ ssh localhost
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.8.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.

103 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

20 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '24.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Oct 23 19:54:01 2024 from 127.0.0.1
akb@akb-VirtualBox:~$ 
```

STEP 5: STARTING HADOOP

Now that everything was set up, I proceeded to start Hadoop.

5.1 FORMATTING NAMENODE

Before starting Hadoop, I needed to format the NameNode. I did this using the command:

hdfs namenode -format

5.2 STARTING HDFS

To start the Hadoop Distributed File System, I ran the command:

start-dfs.sh

5.3 STARTING YARN

To start the YARN resource manager, I ran the coomand:

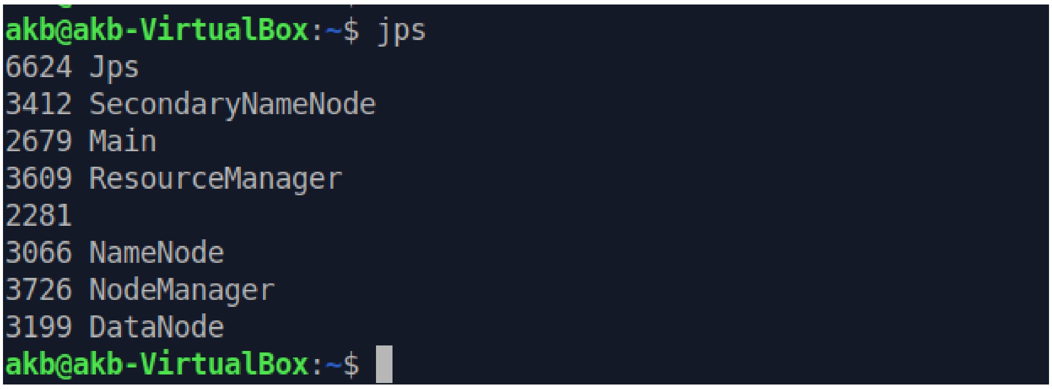
start-yarn.sh

5.4 CHECKING THE SERVICES

To make sure everything was running, I used the jps command in the terminal:

jps

This command listed running Hadoop services like NameNode, DataNode, ResourceManager, and NodeManager.



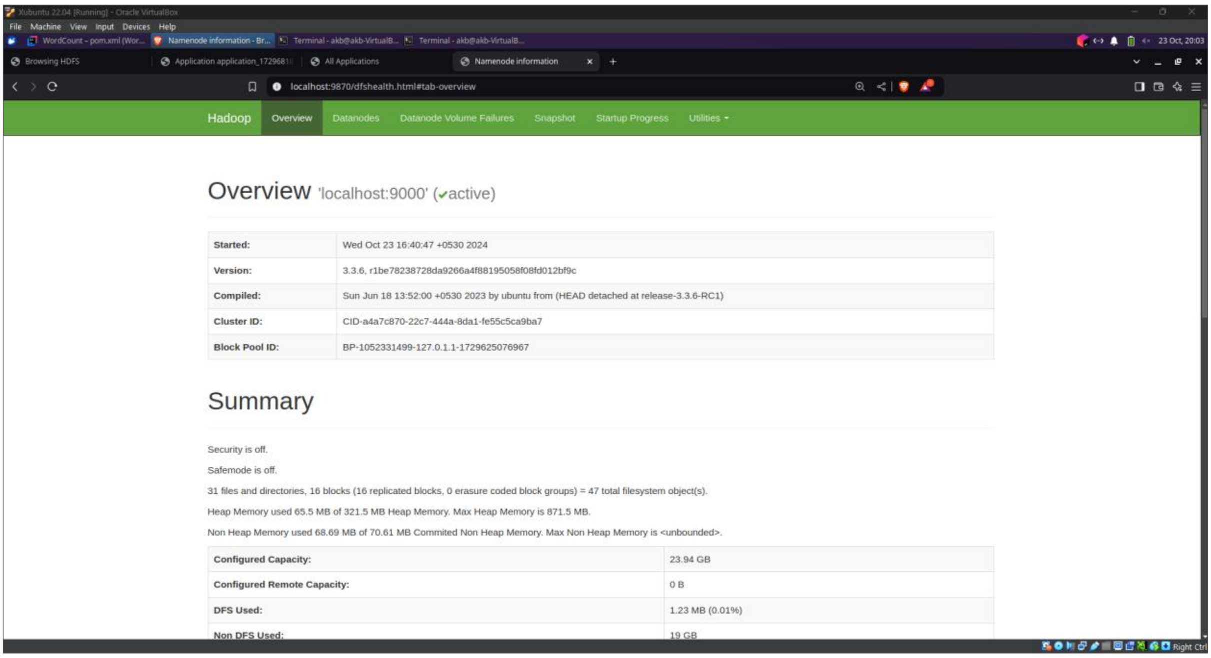
```
akb@akb-VirtualBox:~$ jps
6624 Jps
3412 SecondaryNameNode
2679 Main
3609 ResourceManager
2281
3066 NameNode
3726 NodeManager
3199 DataNode
akb@akb-VirtualBox:~$
```

STEP 6: ACCESSING HADOOP

Finally, to access the Hadoop web interface and check the status of my cluster, I opened the web browser and went to:

http://localhost:9870

This brought up the Hadoop cluster summary page, confirming that everything was working as expected.


Xubuntu 22.04 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
WordCount – pom.xml (Wor...
Namenode information - Br...
Terminal - akb@akb-VirtualB...
Terminal - akb@akb-VirtualB...
23 Oct, 20:03
Browsing HDFS
Application application_1729681
All Applications
Namenode information
localhost:9870/dfshealth.html#tab-overview
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Right Ctrl

Overview 'localhost:9000' (✓active)

Started:	Wed Oct 23 16:40:47 +0530 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 13:52:00 +0530 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-a4a7c870-22c7-444a-8da1-fe55c5ca9ba7
Block Pool ID:	BP-1052331499-127.0.1.1-1729625076967

Summary

Security is off.

Safemode is off.

31 files and directories, 16 blocks (16 replicated blocks, 0 erasure coded block groups) = 47 total filesystem object(s).

Heap Memory used 65.5 MB of 321.5 MB Heap Memory. Max Heap Memory is 871.5 MB.

Non Heap Memory used 68.69 MB of 70.61 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	23.94 GB
Configured Remote Capacity:	0 B
DFS Used:	1.23 MB (0.01%)
Non DFS Used:	19 GB

ISSUES ENCOUNTERED AND SOLUTIONS

SSH Key Issue: Initially, I had permission issues with the SSH key. I fixed this by setting the right permissions:

chmod 640 ~/.ssh/authorized_keys

Connection Refused Error: When starting Hadoop, I encountered a "localhost connection refused" error. This was resolved by ensuring that SSH was installed and properly configured:

sudo apt install openssh-server

By following these steps, I was able to successfully install and configure Java 8 and Hadoop 3.3.6 on my Xubuntu virtual system, and now my system is ready to run Hadoop-based applications.

WORDCOUNT HADOOP PROJECT

In this project, I implemented a simple **WordCount** example using **Hadoop** and **Java**. The WordCount program is one of the most basic and widely used examples to demonstrate how Hadoop's MapReduce framework works. Its purpose is to count the occurrences of each word in a given text file by breaking down the process into two phases: **map** and **reduce**. This distributes the workload across multiple nodes for parallel processing, making it scalable for large datasets.

SETTING UP THE ENVIRONMENT

Before diving into the code, I needed to set up the environment, which involved installing and configuring **Maven**, **Hadoop**, and **IntelliJ IDEA**.

1. Setting Up IntelliJ IDEA and Maven

I used **IntelliJ IDEA Community Edition** to build this project. To start, I created a new **Maven** project.

- Opened IntelliJ and clicked on **New Project**.
- Selected **Maven** as the project type and ensured that **Java JDK** is installed.
- I named the project as WordCount, and set the group and artifact IDs. I set the groupId as **org.akb** and left the artifact ID as default.

After the project was created, I deleted the default main class, as it wasn't needed for this project.

The next step was to add the necessary **dependencies** to the pom.xml file.

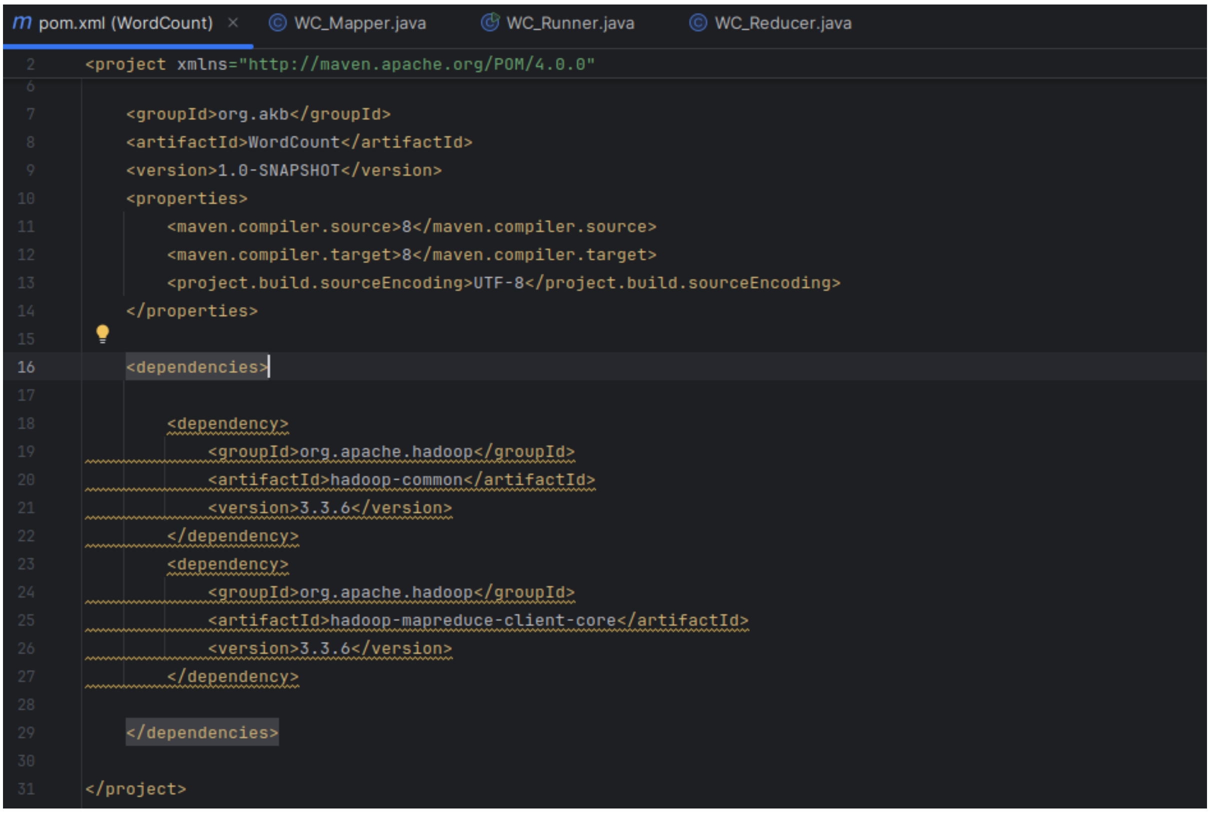
2. Adding Dependencies

In the pom.xml file, I added dependencies for **Hadoop Common** and **Hadoop MapReduce Client Core** to ensure that my project could use the required Hadoop libraries.

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>3.3.6</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>3.3.6</version>
        </dependency>
    </dependencies>
```

I reloaded the Maven project so that all dependencies were downloaded and added to the classpath.



```
<project xmlns="http://maven.apache.org/POM/4.0.0"

    <groupId>org.akb</groupId>
    <artifactId>WordCount</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>3.3.6</version>
        </dependency>

    </dependencies>

</project>
```

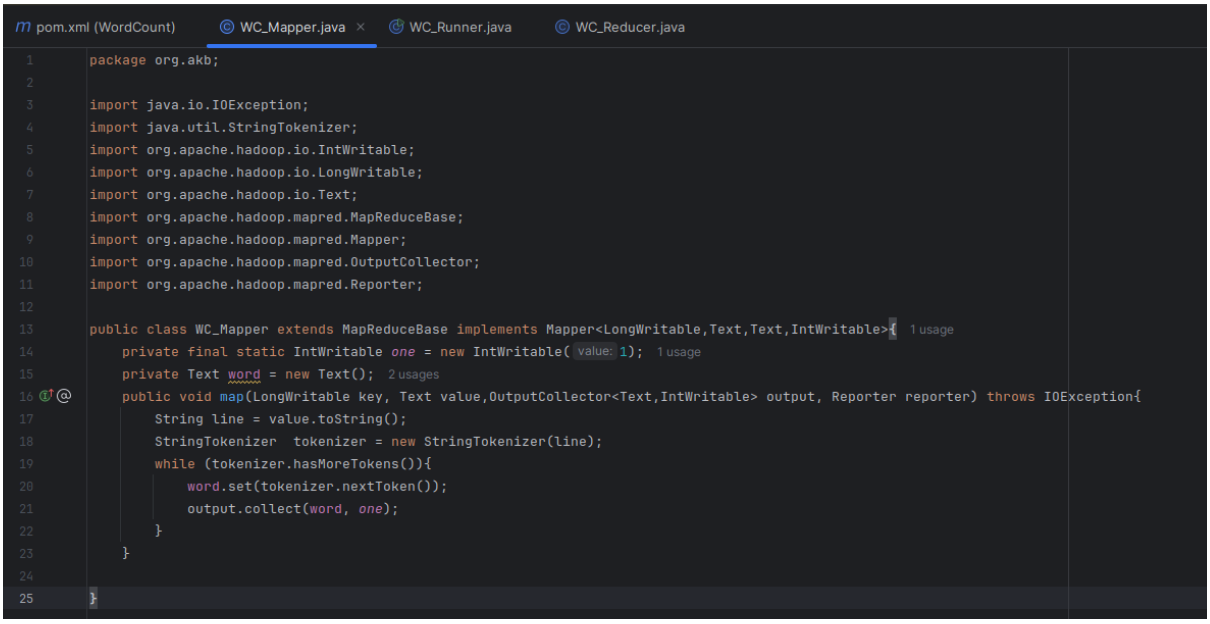
WRITING THE WORDCOUNT PROJECT CODE

The WordCount project consists of three main components:

1. **WC_Mapper** – Handles the **map** phase of the process.
2. **WC_Reducer** – Handles the **reduce** phase of the process.
3. **WC_Runner** – Acts as the driver to set up and run the MapReduce job.

1. Creating WC_Mapper

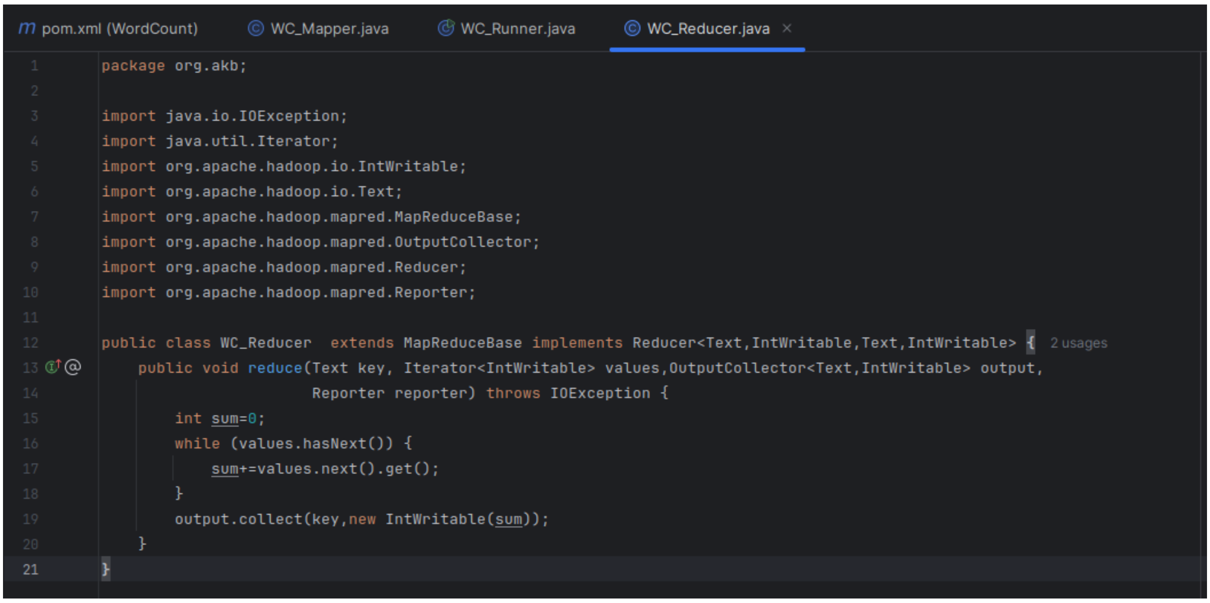
I created a new class called WC_Mapper.java under the package org.akb. The mapper's job is to split the input text into individual words and emit each word as a key with a count of one as its value.



```java
package org.akb;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer  tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }

}
```

2. Creating WC_Reducer

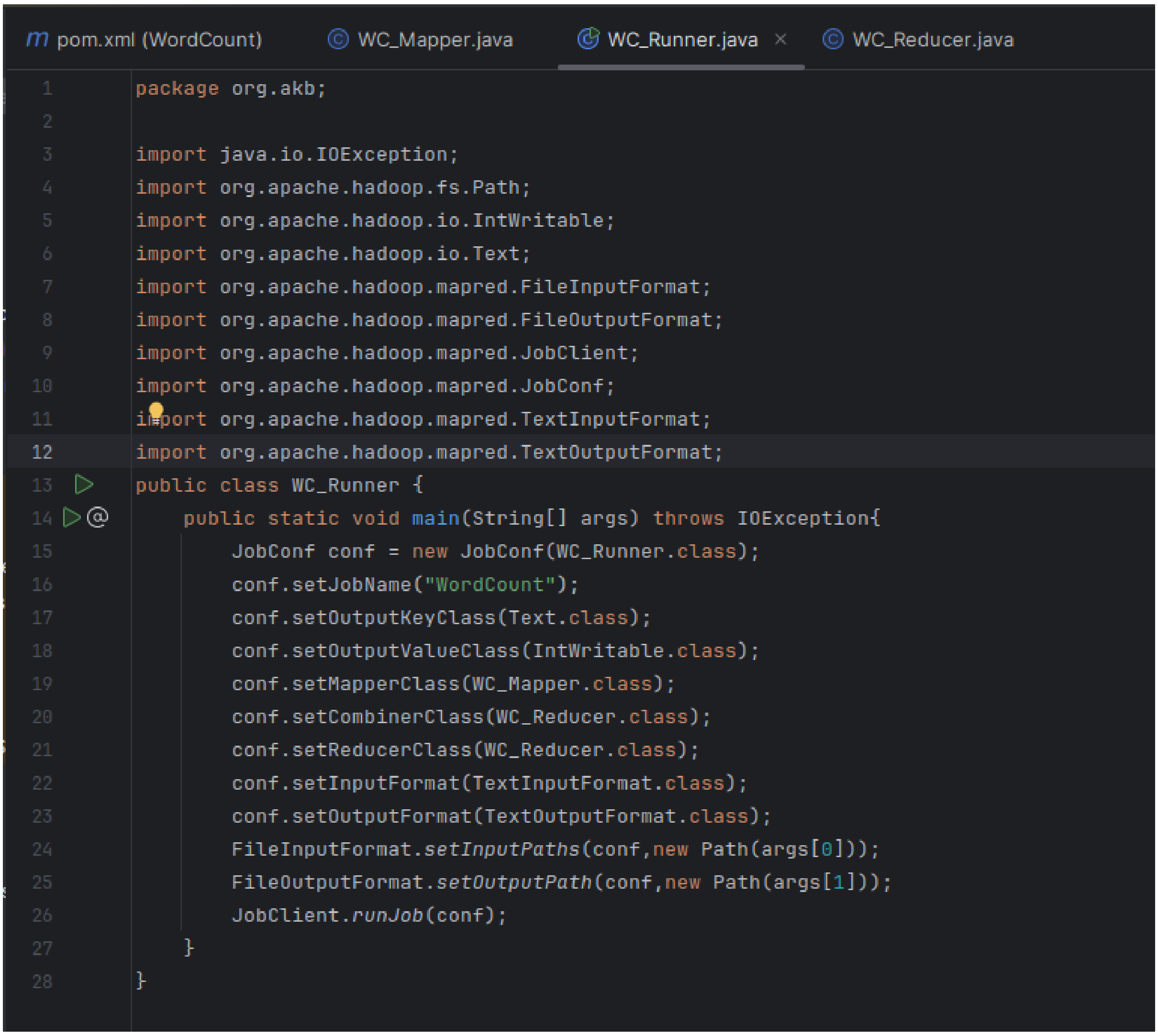
Next, I created WC_Reducer.java. The reducer receives each word and the corresponding list of counts from the mapper. It sums the counts for each word and outputs the final result.



```java
package org.akb;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
                       Reporter reporter) throws IOException {
        int sum=0;
        while (values.hasNext()) {
            sum+=values.next().get();
        }
        output.collect(key,new IntWritable(sum));
    }
}
```

3. Creating WC_Runner

Finally, I created WC_Runner.java, which configures and runs the MapReduce job. This class defines the input and output locations and specifies the Mapper and Reducer classes.



```java
package org.akb;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WC_Runner {
    public static void main(String[] args) throws IOException{
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

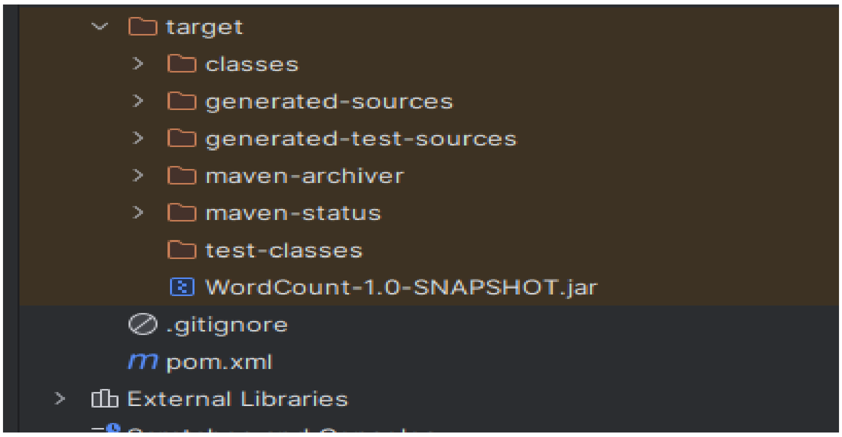
BUILDING AND RUNNING THE PROJECT

1. Creating the JAR File

Once all the code was written, I built the project using Maven. I opened the terminal in IntelliJ and ran the following commands to clean and package the project:

mvn clean package

This generated a JAR file in the target folder, which I would use to run the WordCount job.



```
target
  classes
  generated-sources
  generated-test-sources
  maven-archiver
  maven-status
  test-classes
  WordCount-1.0-SNAPSHOT.jar
.gitignore
pom.xml
External Libraries
```

2. Running the WordCount Job

Next, I created a simple text file as input. I used the terminal to create a file called input.txt with some random text using the commands:

nano input.txt

I added the following text:

This is the input file for hadoop project

This is for Cloud Computing with AKB hadoop project file

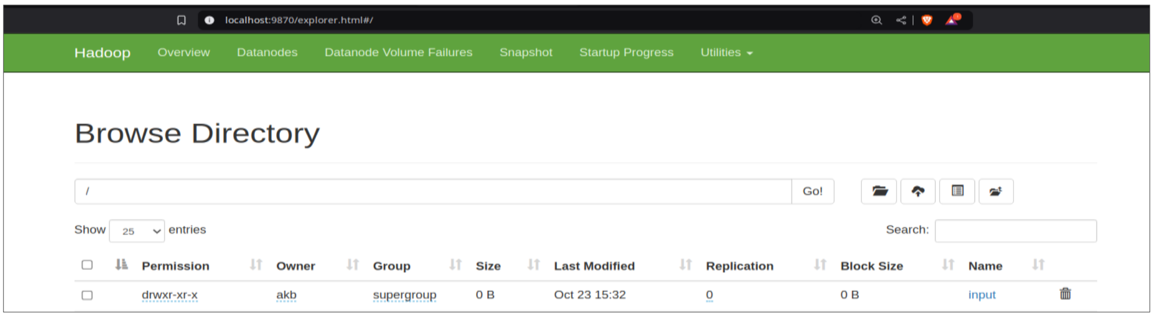
This is done for now

I then created a directory in HDFS to store the input file:

hadoop fs -mkdir /input

Then, I copied the input file to HDFS:

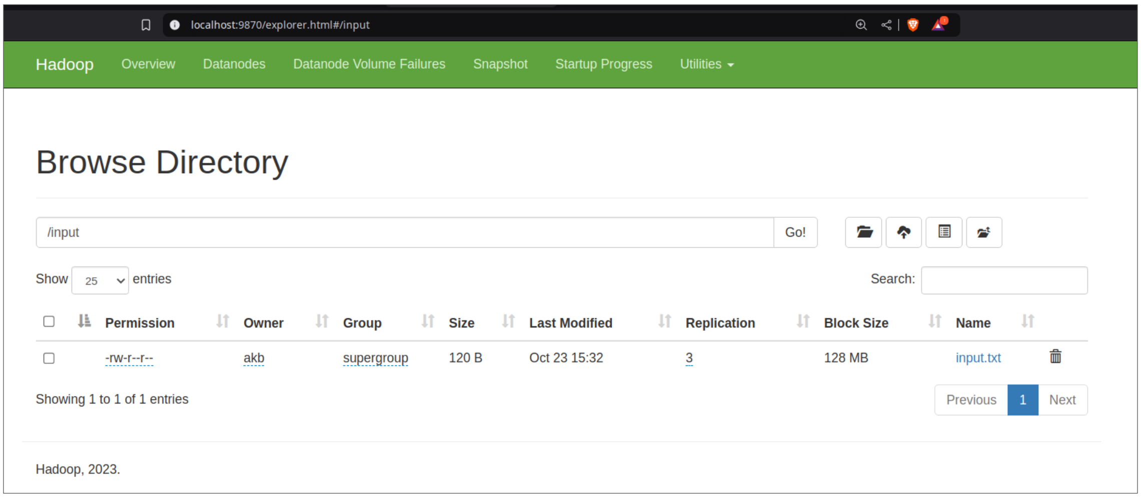
hadoop fs -put input.txt /input



Browse Directory

/

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	akb	supergroup	0 B	Oct 23 15:32	0	0 B	input



Browse Directory

/input

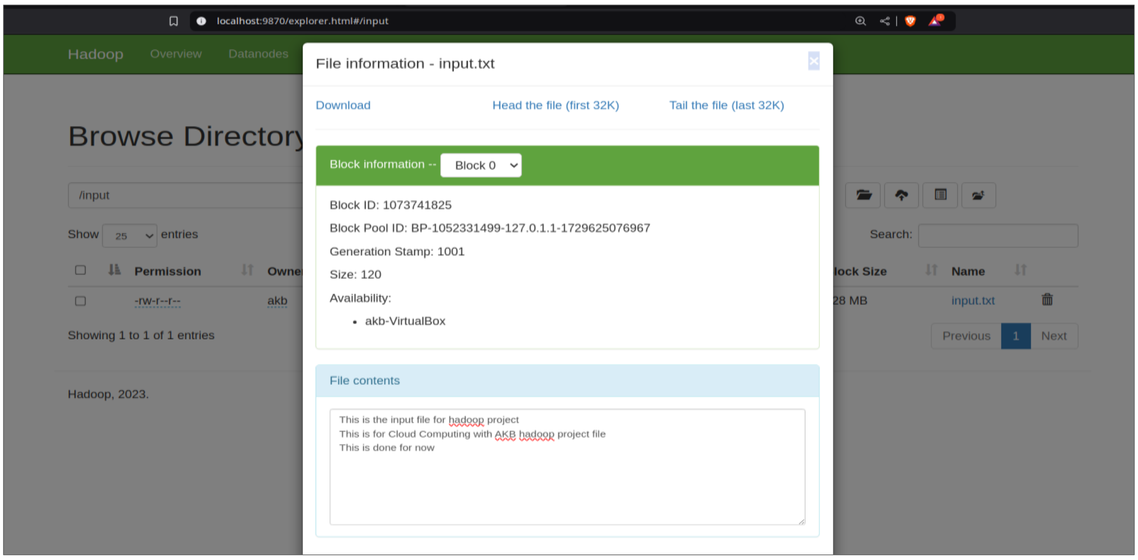
Show 25 entries Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	akb	supergroup	120 B	Oct 23 15:32	3	128 MB	input.txt

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2023.


localhost:9870/explorer.html#/input
Hadoop
Overview
Datanodes
Download
Head the file (first 32K)
Tail the file (last 32K)

File information - input.txt

Block information -- Block 0

Block ID: 1073741825

Block Pool ID: BP-1052331499-127.0.1.1-1729625076967

Generation Stamp: 1001

Size: 120

Availability:

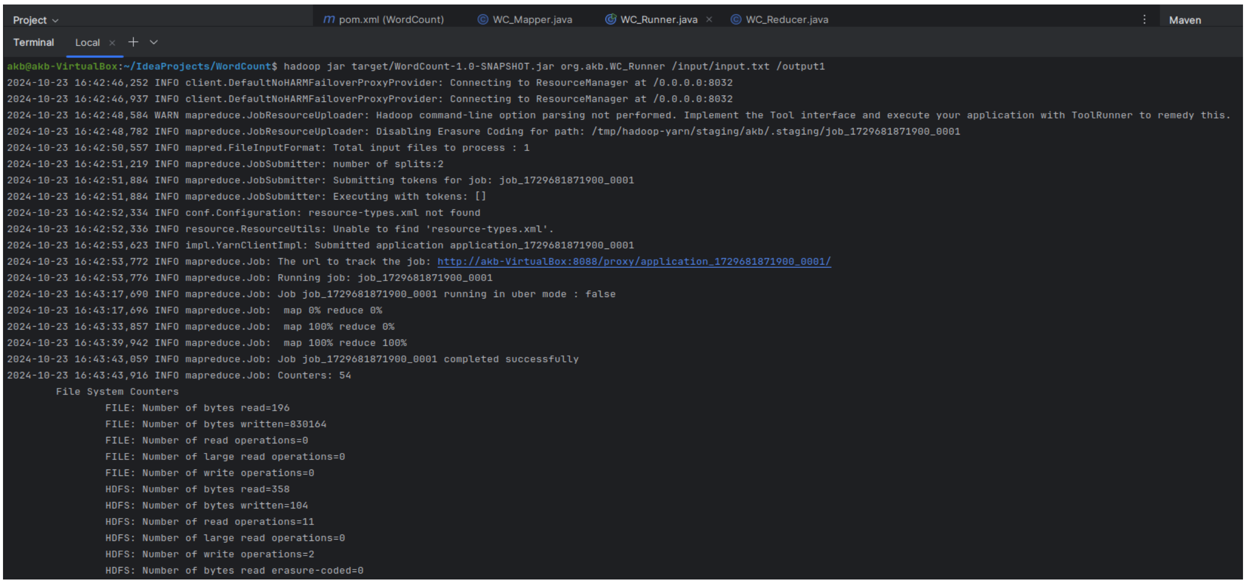
- akb-VirtualBox

File contents

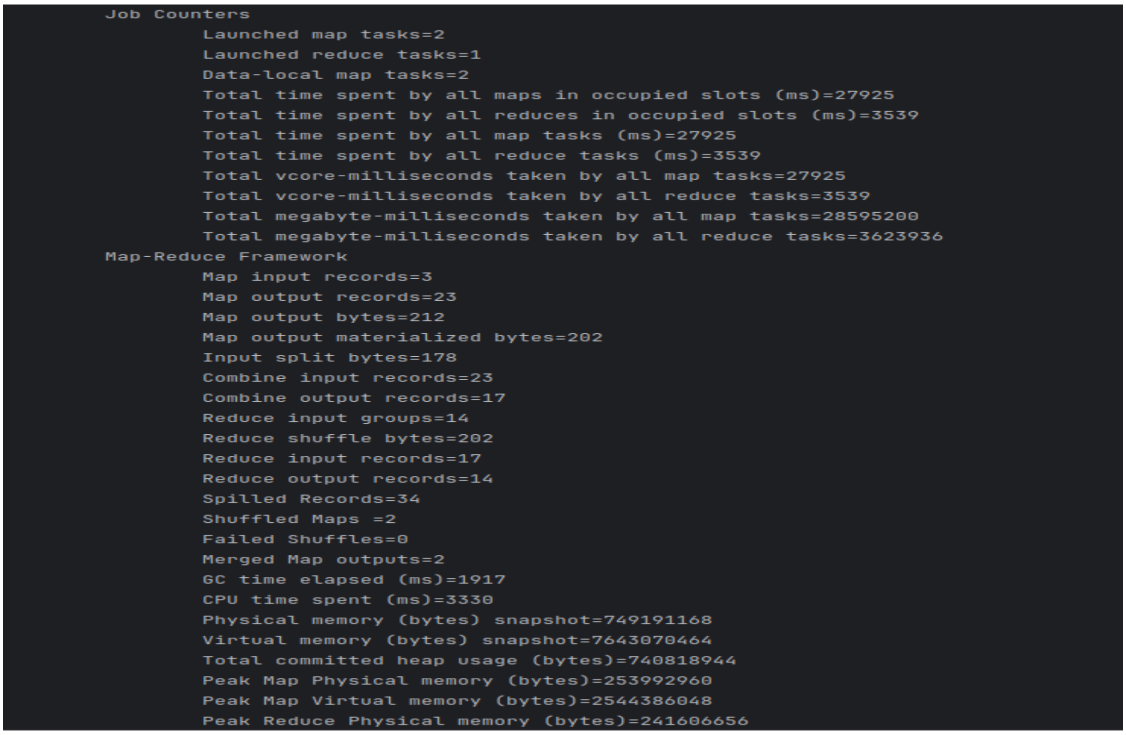
```
This is the input file for hadoop project
This is for Cloud Computing with AKB hadoop project file
This is done for now
```

Finally, I ran the WordCount job by executing the following command:

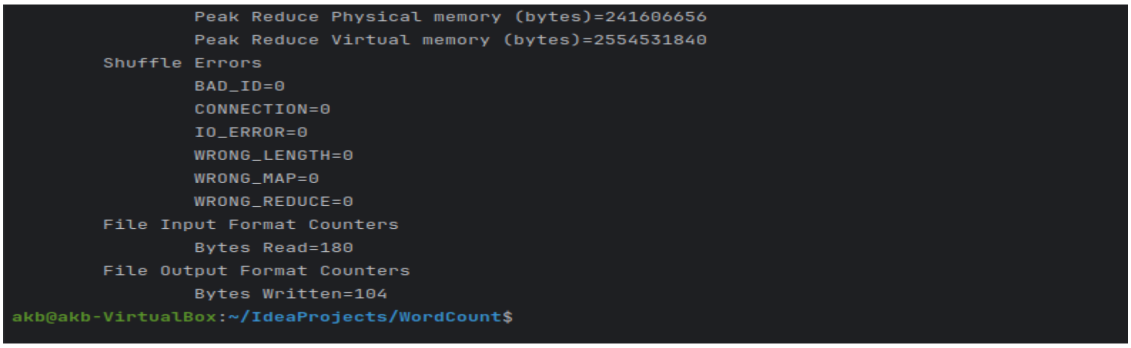
hadoop jar target/wordcount-1.0-SNAPSHOT.jar org.akb.WC_Runner /input/input.txt /output



```
akb@akb-VirtualBox:~/IdeaProjects/WordCount$ hadoop jar target/WordCount-1.0-SNAPSHOT.jar org.akb.WC_Runner /input/input.txt /output1
2024-10-23 16:42:46,252 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-10-23 16:42:46,937 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-10-23 16:42:48,584 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2024-10-23 16:42:48,782 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/akb/.staging/job_1729681871900_0001
2024-10-23 16:42:50,557 INFO mapred.FileInputFormat: Total input files to process : 1
2024-10-23 16:42:51,219 INFO mapreduce.JobSubmitter: number of splits:2
2024-10-23 16:42:51,884 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1729681871900_0001
2024-10-23 16:42:51,884 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-23 16:42:52,334 INFO conf.Configuration: resource-types.xml not found
2024-10-23 16:42:52,336 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-10-23 16:42:53,623 INFO impl.YarnClientImpl: Submitted application application_1729681871900_0001
2024-10-23 16:42:53,772 INFO mapreduce.Job: The url to track the job: http://akb-VirtualBox:8088/proxy/application_1729681871900_0001/
2024-10-23 16:42:53,776 INFO mapreduce.Job: Running job: job_1729681871900_0001
2024-10-23 16:43:17,690 INFO mapreduce.Job: Job job_1729681871900_0001 running in uber mode : false
2024-10-23 16:43:17,696 INFO mapreduce.Job:  map 0% reduce 0%
2024-10-23 16:43:33,857 INFO mapreduce.Job:  map 100% reduce 0%
2024-10-23 16:43:39,942 INFO mapreduce.Job:  map 100% reduce 100%
2024-10-23 16:43:43,059 INFO mapreduce.Job: Job job_1729681871900_0001 completed successfully
2024-10-23 16:43:43,916 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=196
                FILE: Number of bytes written=830164
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=358
                HDFS: Number of bytes written=104
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
```



```
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=27925
                Total time spent by all reduces in occupied slots (ms)=3539
                Total time spent by all map tasks (ms)=27925
                Total time spent by all reduce tasks (ms)=3539
                Total vcore-milliseconds taken by all map tasks=27925
                Total vcore-milliseconds taken by all reduce tasks=3539
                Total megabyte-milliseconds taken by all map tasks=28595200
                Total megabyte-milliseconds taken by all reduce tasks=3623936
        Map-Reduce Framework
                Map input records=3
                Map output records=23
                Map output bytes=212
                Map output materialized bytes=202
                Input split bytes=178
                Combine input records=23
                Combine output records=17
                Reduce input groups=14
                Reduce shuffle bytes=202
                Reduce input records=17
                Reduce output records=14
                Spilled Records=34
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=1917
                CPU time spent (ms)=3330
                Physical memory (bytes) snapshot=749191168
                Virtual memory (bytes) snapshot=7643070464
                Total committed heap usage (bytes)=740818944
                Peak Map Physical memory (bytes)=253992960
                Peak Map Virtual memory (bytes)=2544386048
                Peak Reduce Physical memory (bytes)=241606656
```



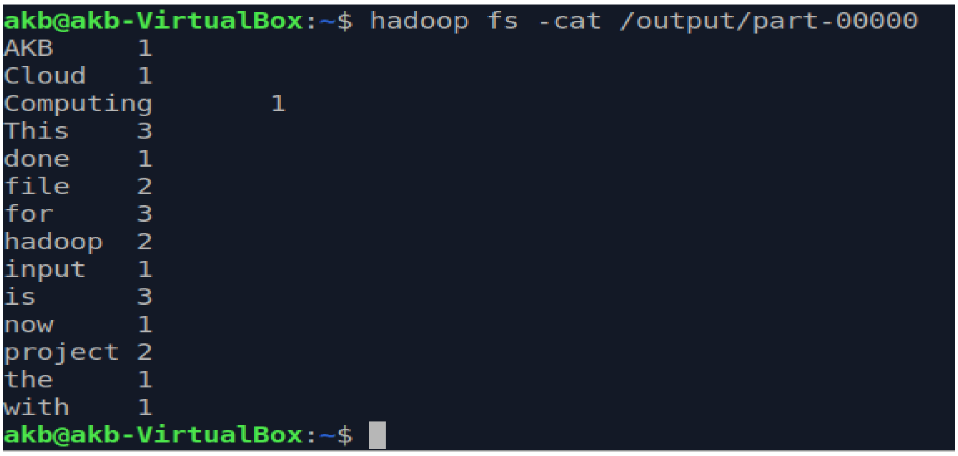
```
                Peak Reduce Physical memory (bytes)=241606656
                Peak Reduce Virtual memory (bytes)=2554531840
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=180
        File Output Format Counters
                Bytes Written=104
akb@akb-VirtualBox:~/IdeaProjects/WordCount$
```

3. Checking the Output

Once the job finished, I verified the output by listing the contents of the /output directory in HDFS:

hadoop fs -cat /output/part-00000

The output displayed the word count:

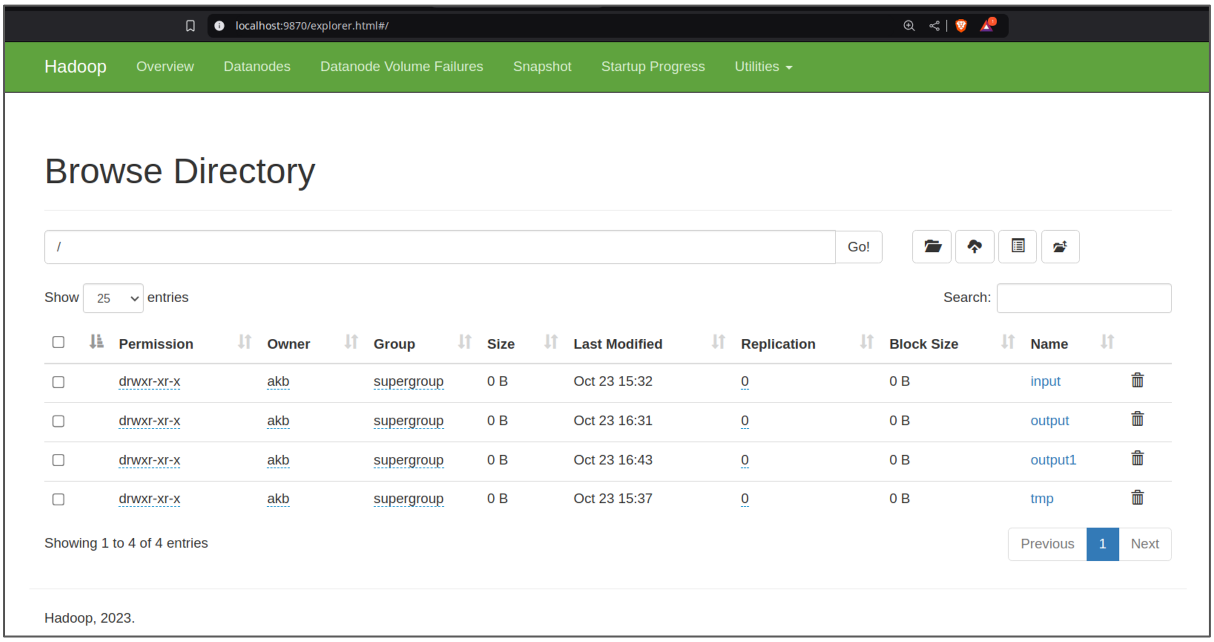


```
akb@akb-VirtualBox:~$ hadoop fs -cat /output/part-00000
AKB     1
Cloud   1
Computing       1
This    3
done    1
file    2
for     3
hadoop  2
input   1
is      3
now     1
project 2
the     1
with    1
akb@akb-VirtualBox:~$
```

CONCLUSION

By following these steps, I successfully created and ran a WordCount Hadoop project using Maven and IntelliJ IDEA. The WordCount example demonstrates how Hadoop's MapReduce framework works by distributing the workload and processing large amounts of data in parallel.

OUTPUT SCREENSHOTS



Browse Directory

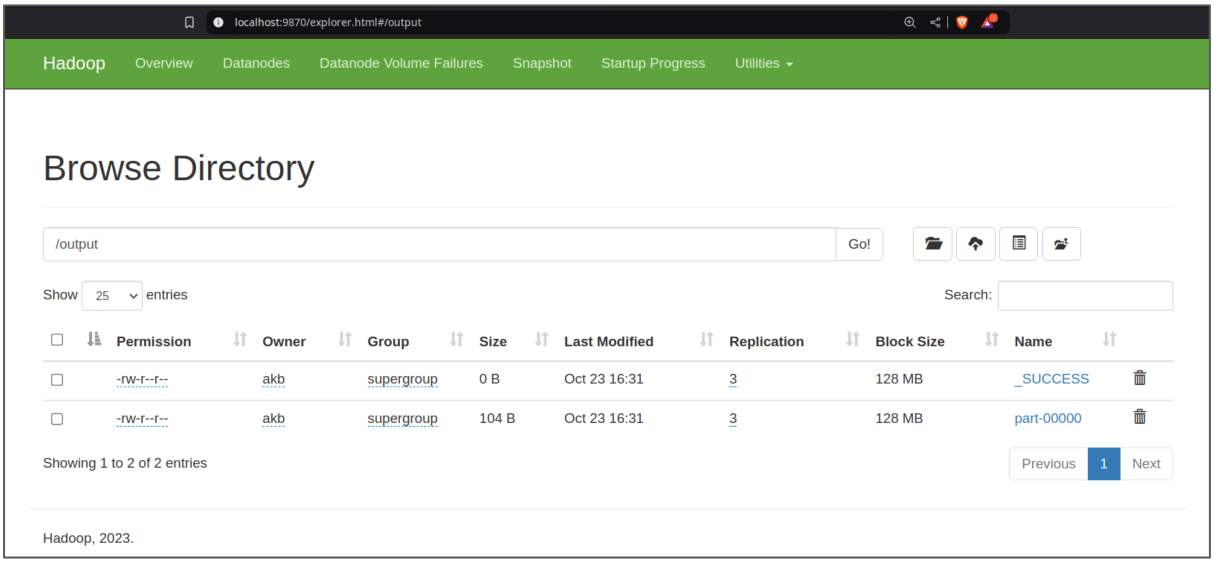
/

Show 25 entries Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	akb	supergroup	0 B	Oct 23 15:32	0	0 B	input
□	drwxr-xr-x	akb	supergroup	0 B	Oct 23 16:31	0	0 B	output
□	drwxr-xr-x	akb	supergroup	0 B	Oct 23 16:43	0	0 B	output1
□	drwxr-xr-x	akb	supergroup	0 B	Oct 23 15:37	0	0 B	tmp

Showing 1 to 4 of 4 entries

Hadoop, 2023.



Browse Directory

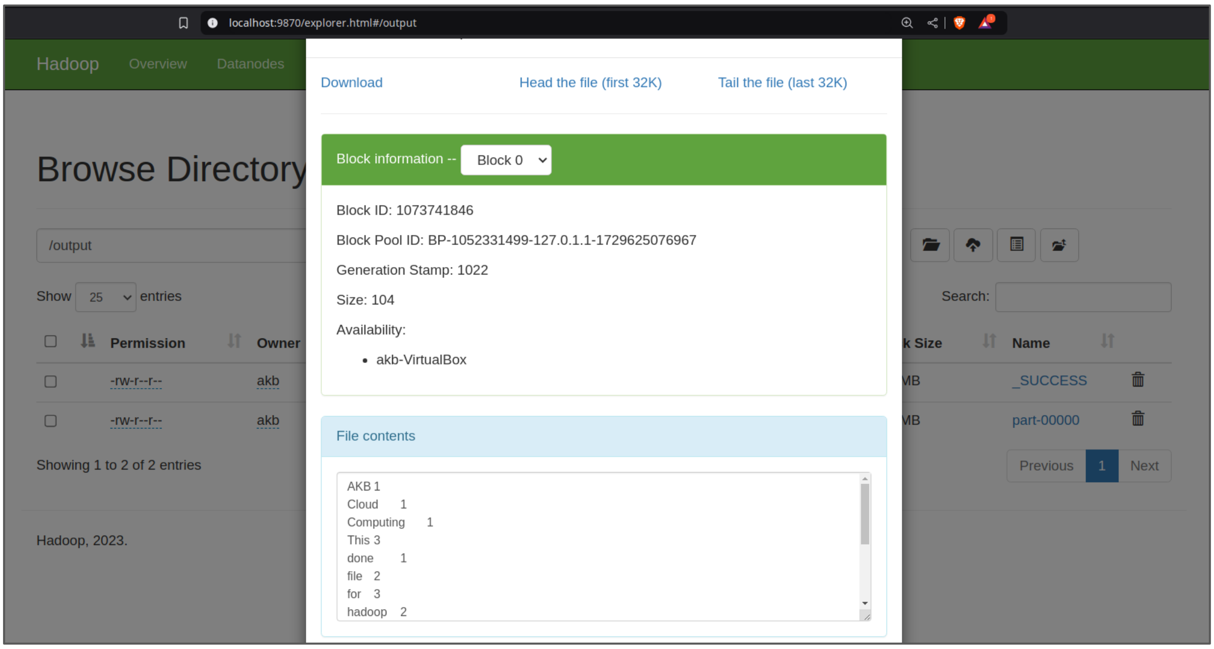
/output

Show 25 entries Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	akb	supergroup	0 B	Oct 23 16:31	3	128 MB	_SUCCESS
□	-rw-r--r--	akb	supergroup	104 B	Oct 23 16:31	3	128 MB	part-00000

Showing 1 to 2 of 2 entries

Hadoop, 2023.



Block ID: 1073741846

Block Pool ID: BP-1052331499-127.0.1.1-1729625076967

Generation Stamp: 1022

Size: 104

Availability:

- akb-VirtualBox

```
AKB 1
Cloud     1
Computing     1
This 3
done      1
file   2
for   3
hadoop   2
```