

**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*

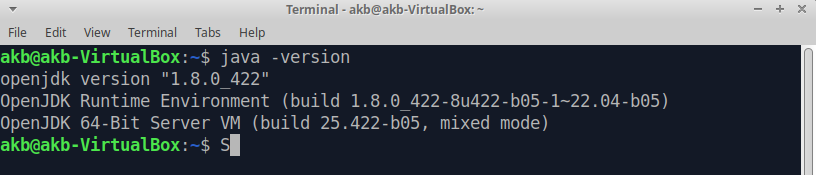
This command downloaded and installed the Java Development Kit (JDK) version 8. I could see the installation process in the terminal, where it fetched the required packages and set everything up.

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:



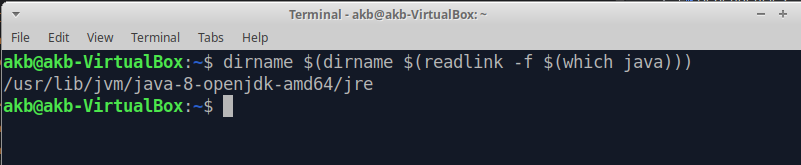
This output indicated that Java 8 was successfully installed on my virtual system.

**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.

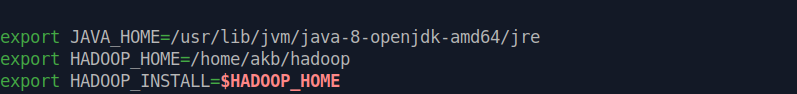


I then added this path to the .bashrc file. To do that, I opened .bashrc using:

*nano ~/.bashrc*

At the bottom of the file, I added:

*export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64*



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

*source ~/.bashrc*



**STEP 2: INSTALLING HADOOP 3.3.6**

**2.1 DOWNLOADING HADOOP**

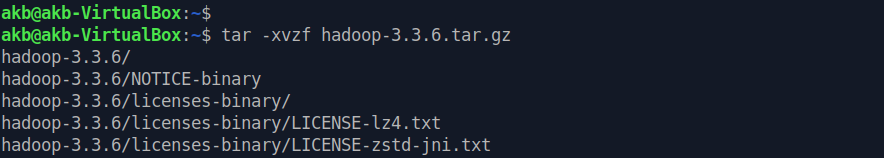
Next, I moved on to downloading Hadoop. I went to the Hadoop website and copied the download link for **Hadoop 3.3.6**. Then, in the terminal, I used wget to download hadoop:

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



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

*tar -xvzf hadoop-3.3.6.tar.gz*

**

**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*

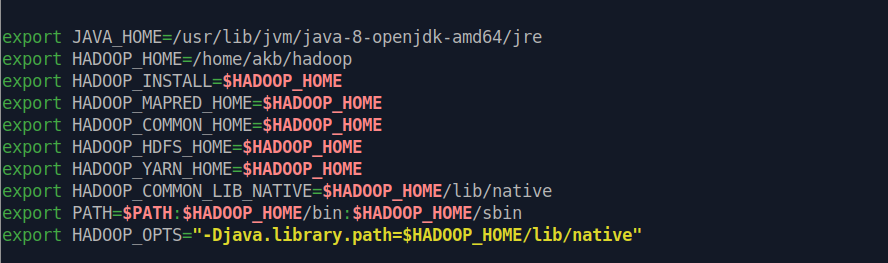
*export HADOOP\_COMMON\_HOME=$HADOOP\_HOME*

*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*



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>*



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>*

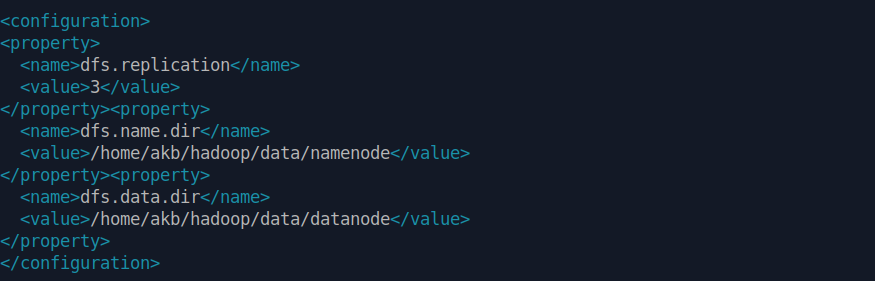
*</property>*

*<property>*

*<name>dfs.data.dir</name>*

*<value>/home/akb/hadoop/data/datanode</value>*

*</property>*



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*

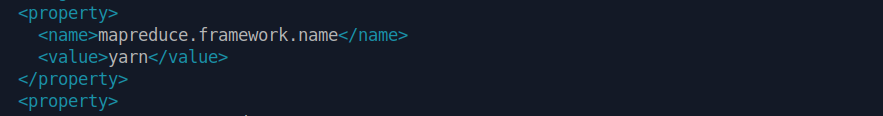
Inside, I added the properties:

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*



**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>*

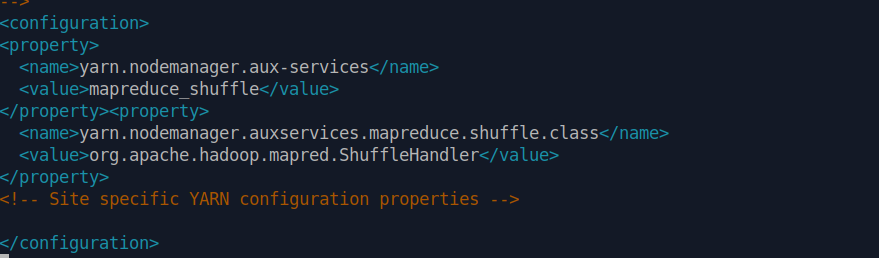
*</property>*

*<property>*

*<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>*

*<value>org.apache.hadoop.mapred.ShuffleHandler</value>*

*</property>*



**STEP 4: SETTING UP SSH**

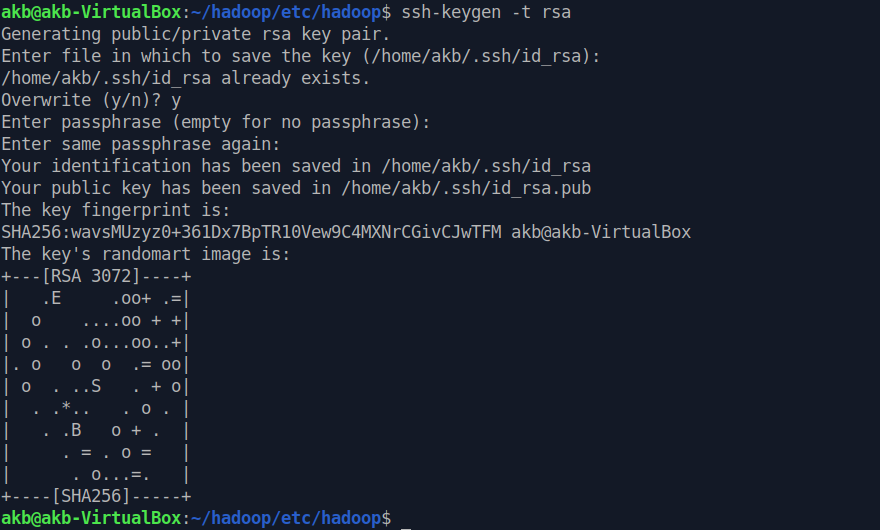
Hadoop requires SSH to communicate between different nodes. I set up SSH on my virtual machine as follows:

**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.



**4.2 AUTHORIZING SSH KEY**

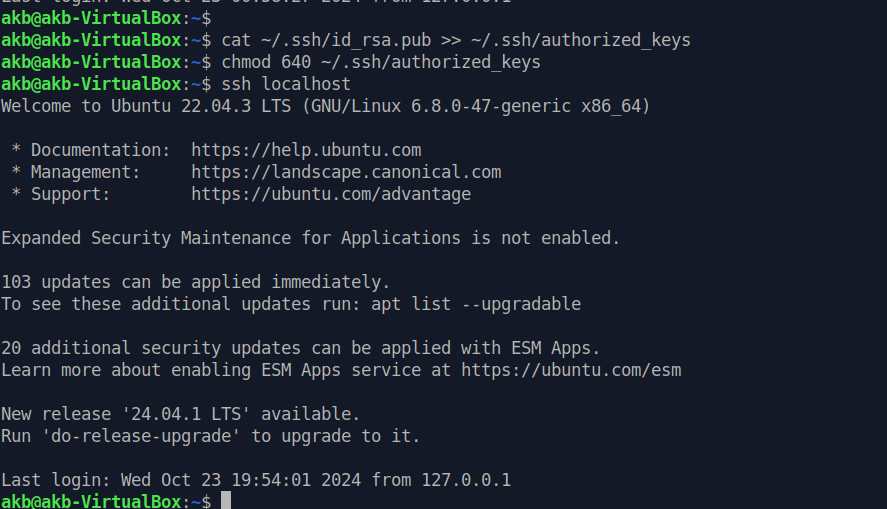
Then, I authorized the SSH key using the commands:

*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*



**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:

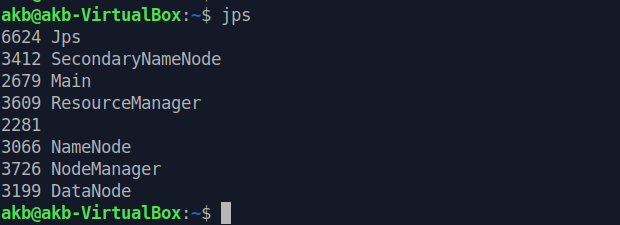
*start-yarn.sh*

**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.

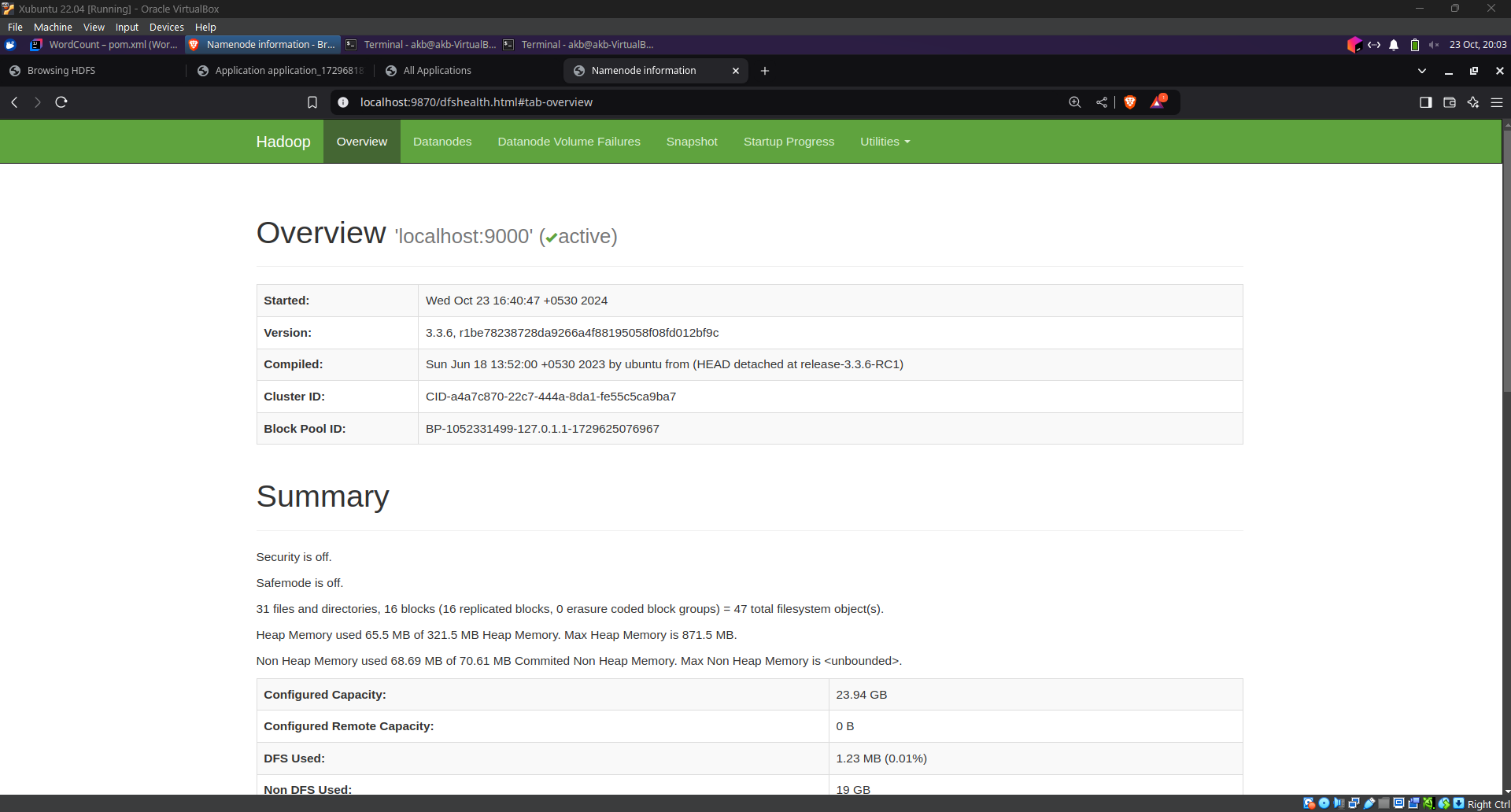


**STEP 6: ACCESSING HADOOP**

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.



**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>*

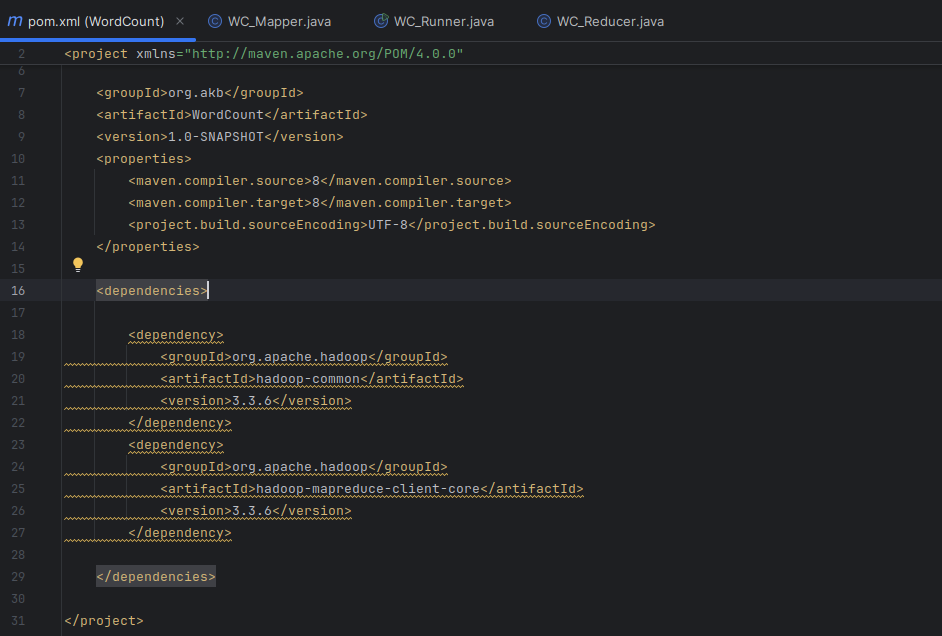
*<artifactId>hadoop-mapreduce-client-core</artifactId>*

*<version>3.3.6</version>*

*</dependency>*

*</dependencies>*

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



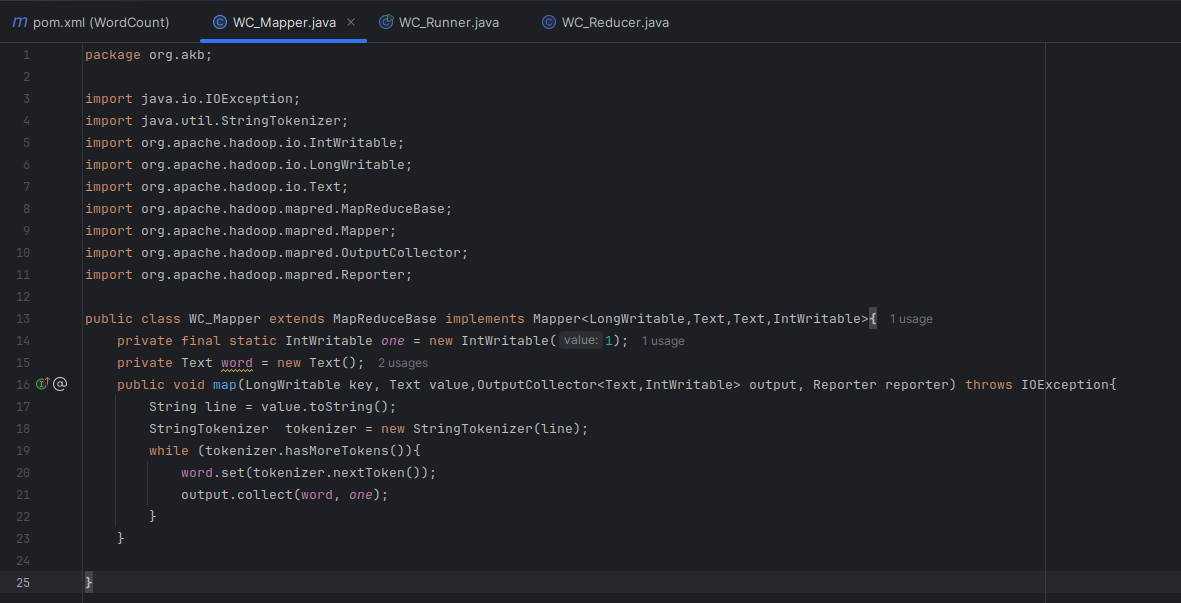
**WRITING THE WORDCOUNT PROJECT CODE**

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.

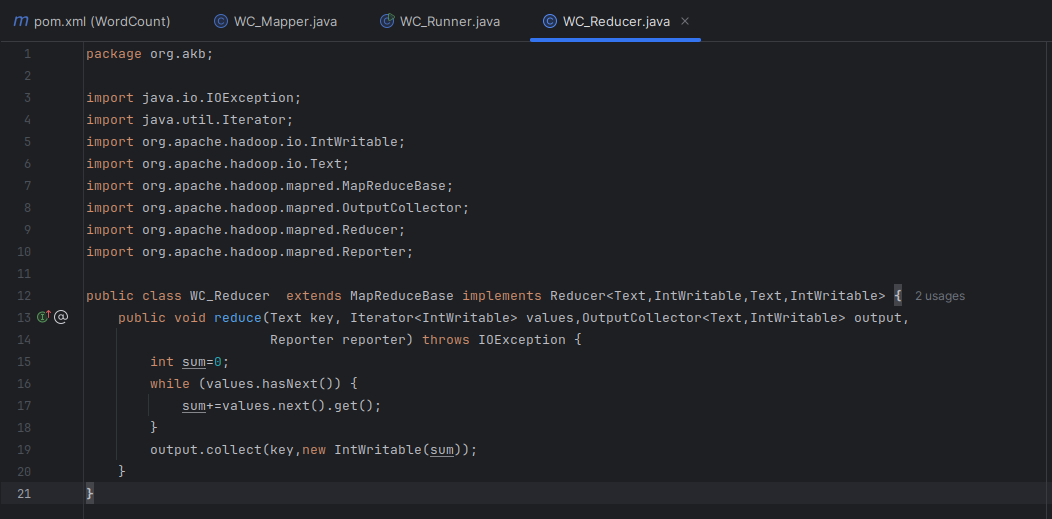
**1. Creating WC\_Mapper**

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.



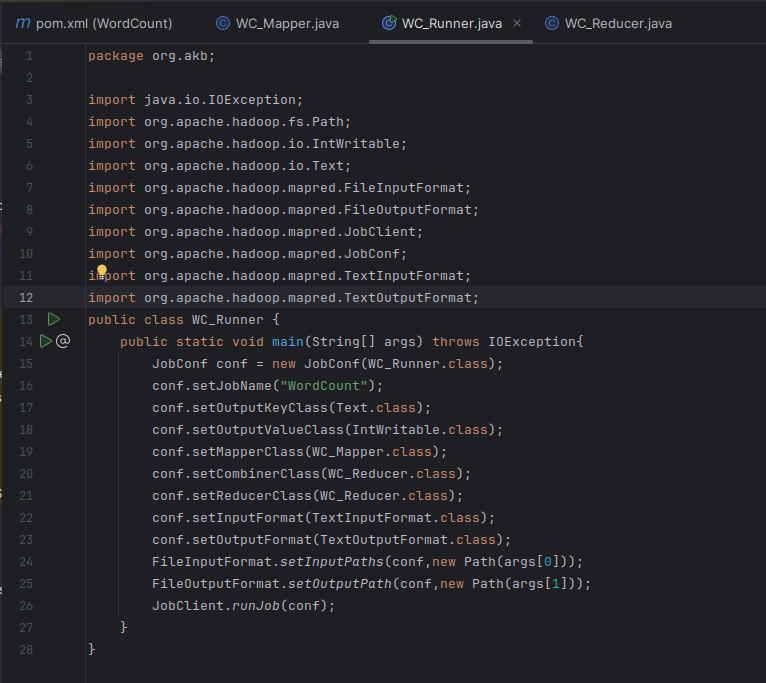
**2. Creating WC\_Reducer**

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.



**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.



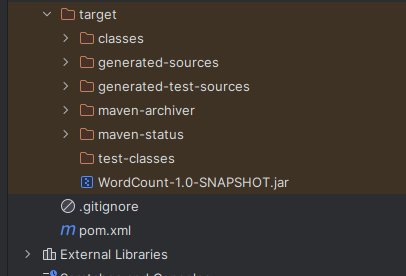
**BUILDING AND RUNNING THE PROJECT**

**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.



**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*

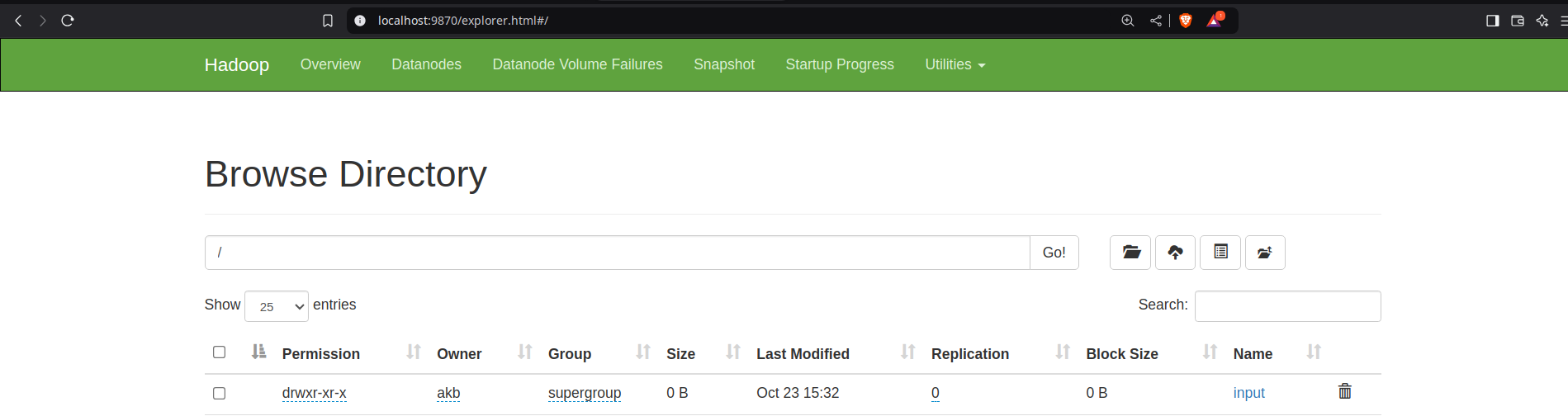
*This is done for now*

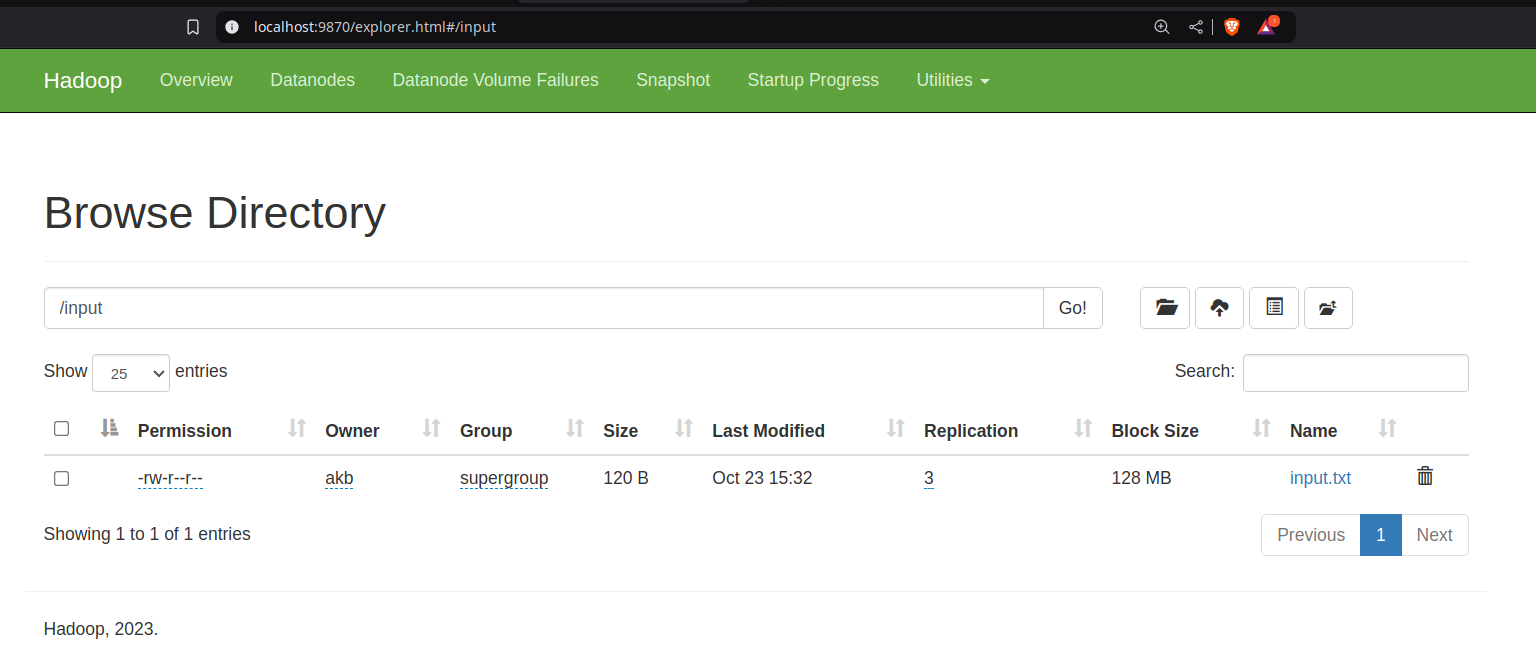
I then created a directory in HDFS to store the input file:

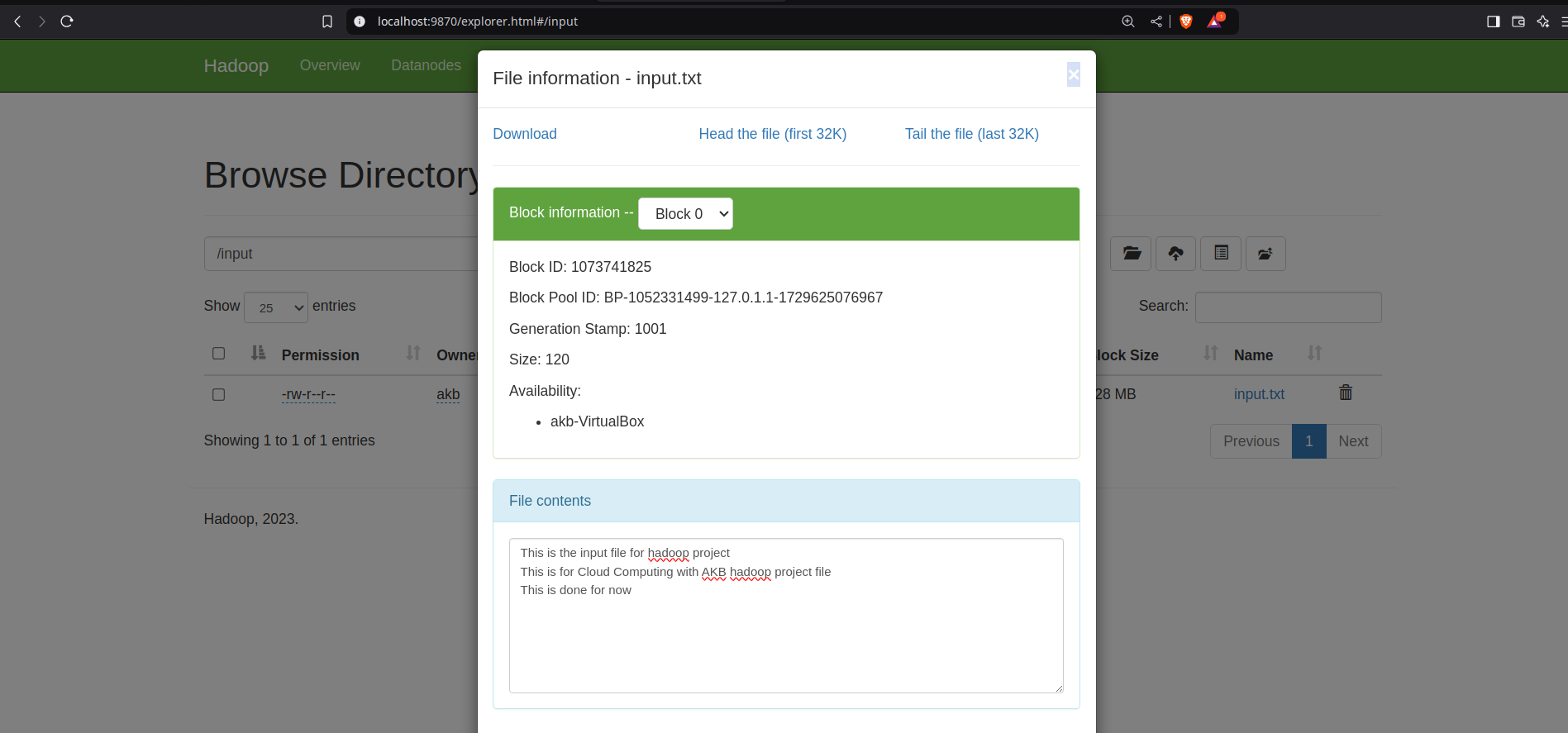
*hadoop fs -mkdir /input*

Then, I copied the input file to HDFS:

*hadoop fs -put input.txt /input*

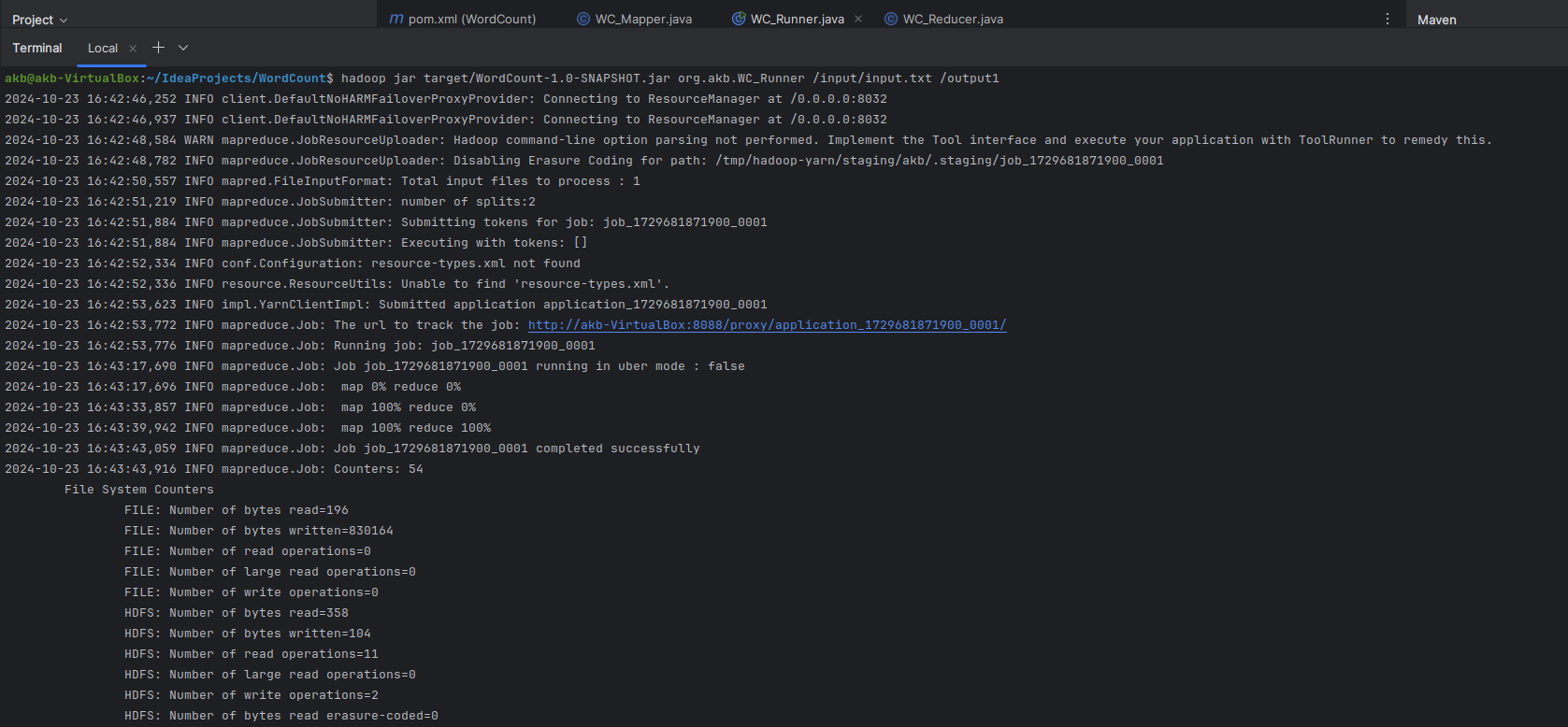


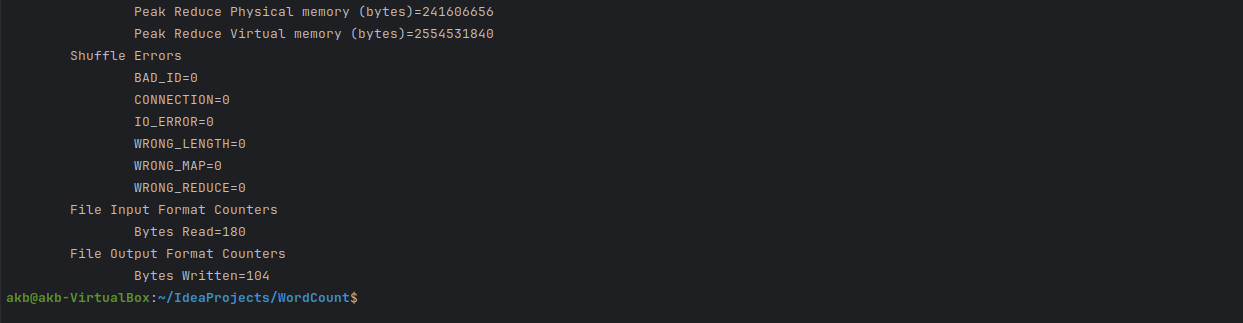
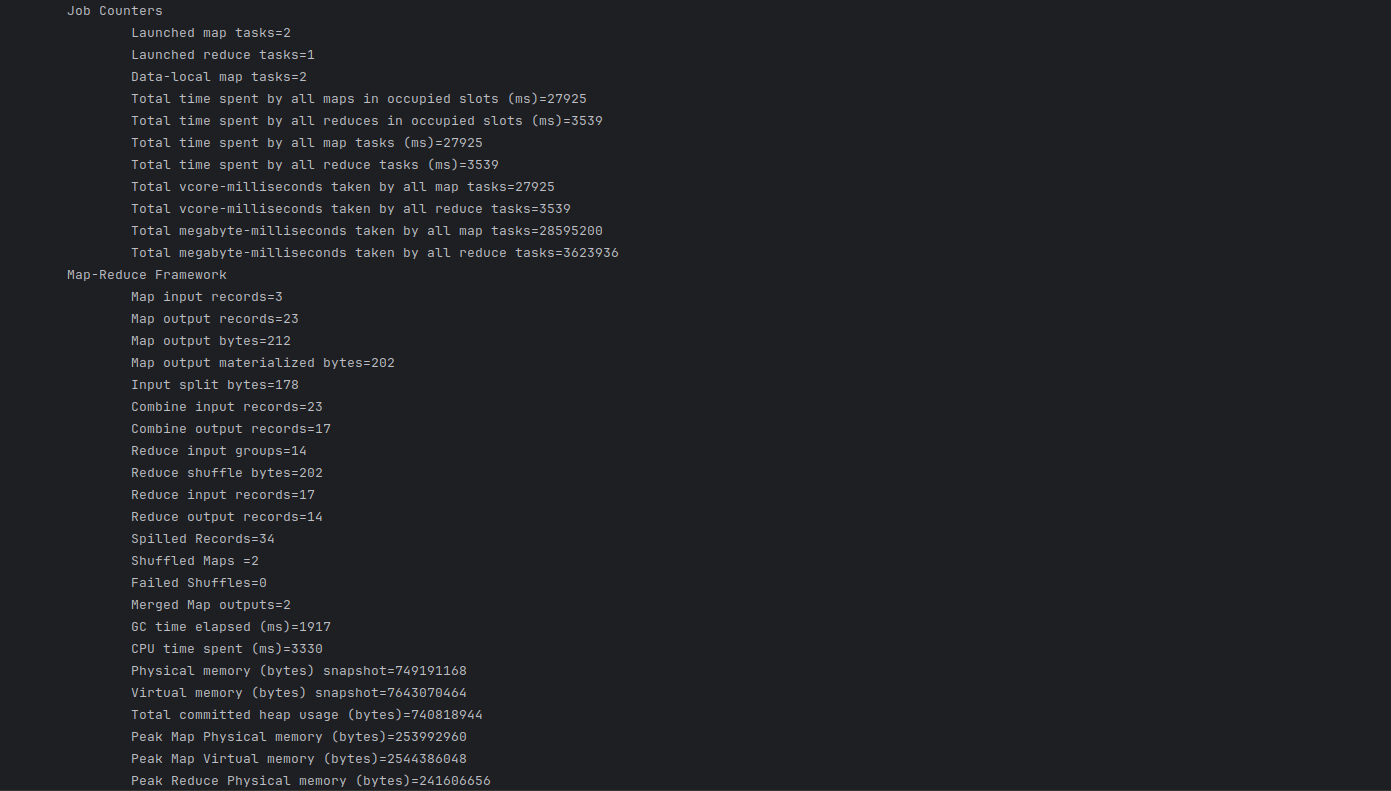




Finally, I ran the WordCount job by executing the following command:

*hadoop jar target/wordcount-1.0-SNAPSHOT.jar org.akb.WC\_Runner /input/input.txt /output*



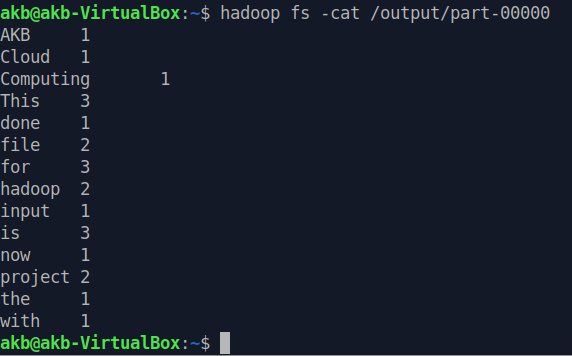


**3. Checking the Output**

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:



**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**

