# Lab 01: A Gentle Introduction to Hadoop

CSC14118 Introduction to Big Data 20KHMT1

All in

2023-02-17

# Contents

- Author: [All in] group
- Date: 21/03/2023
- Subtitle: CSC14118 Introduction to Big Data 20KHMT1
- Language: English

# Task progression

| No. of task | % completed |
|---|---|
| 1 | 100% |
| 2 | 100% |
| 3 | 100% |
| 4.1 | 100% |
| 4.2 | 15% |

# Self-reflection

- Advantages
  - All members in the group are supportive, are eager to learn, try to solve the problems/bugs as best as possible.
  - Groups have set up the plan, communication devices, document, working directory, etc. to work efficiently, as soon as this project started.
  - Everyone helped each other to solve the problems.
  - The group has learned and understood more about the workflow, components of Hadoop; knowed some tricky point in Hadoop; be ready for next projects.
- Disadvantage
  - Due to large amount of works in this project and other subjects, the quality of conducting plan has fallen off gradually.
  - We have not set up the real distributed Hadoop system. We will study this topics as soon as possible. As a result, we have not achieved the bonus scores fully.

# 1 Setting up Single-node Hadoop Cluster

Each member has completed setting up a Single node cluster on their local machine, as described in supplied tutorial. Also, this process is captured in screenshots.

## 1.1 Member 1: 20120090 - Nguyen The Hoang

> This member will present detail of the steps setting up Single-node Hadoop cluster. Other members will only show result in their machines. Some steps or result may be different between machines of each member, due to some difference of software/hardware environment and tutorials that they followed. But without loss of generality, the main steps are same.
>
> This Single-node Hadoop Cluster is installed in the Linux OS, Mint distro.

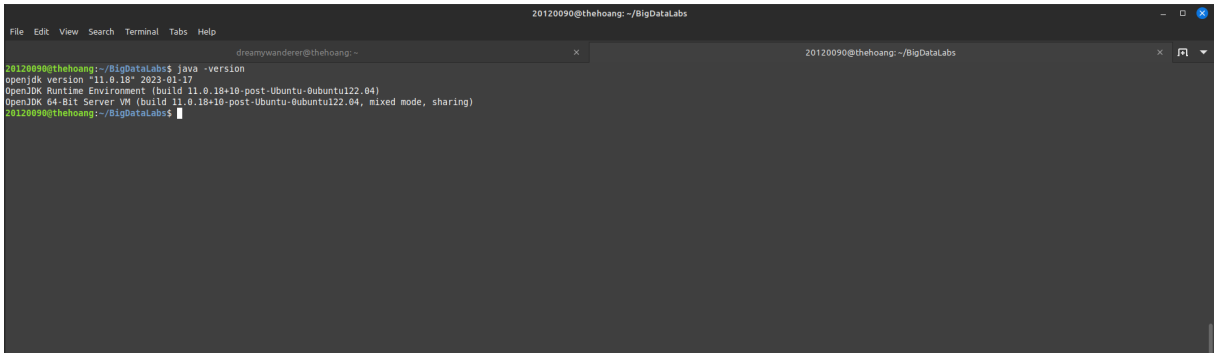



**Figure 1.1:** Step 1: Make sure that a suitable version of Java installed

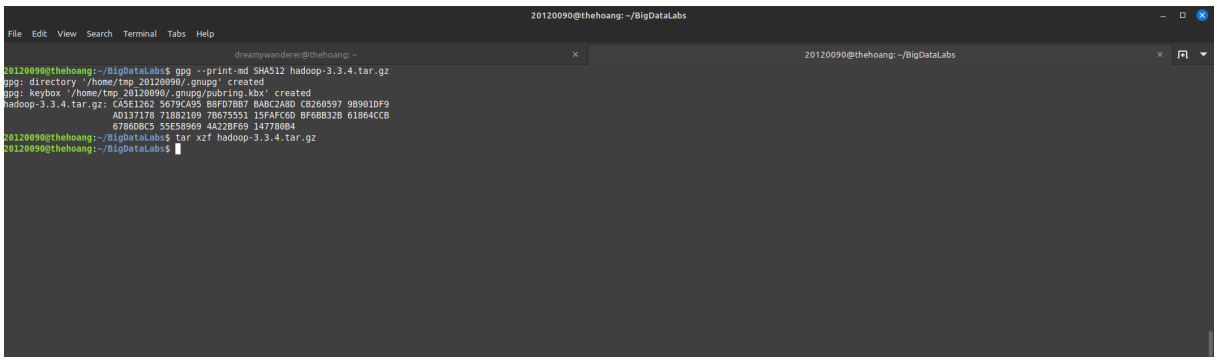



**Figure 1.2:** Step 2: Download and extracted the Hadoop packaged file from the Apache Hadoop releases page

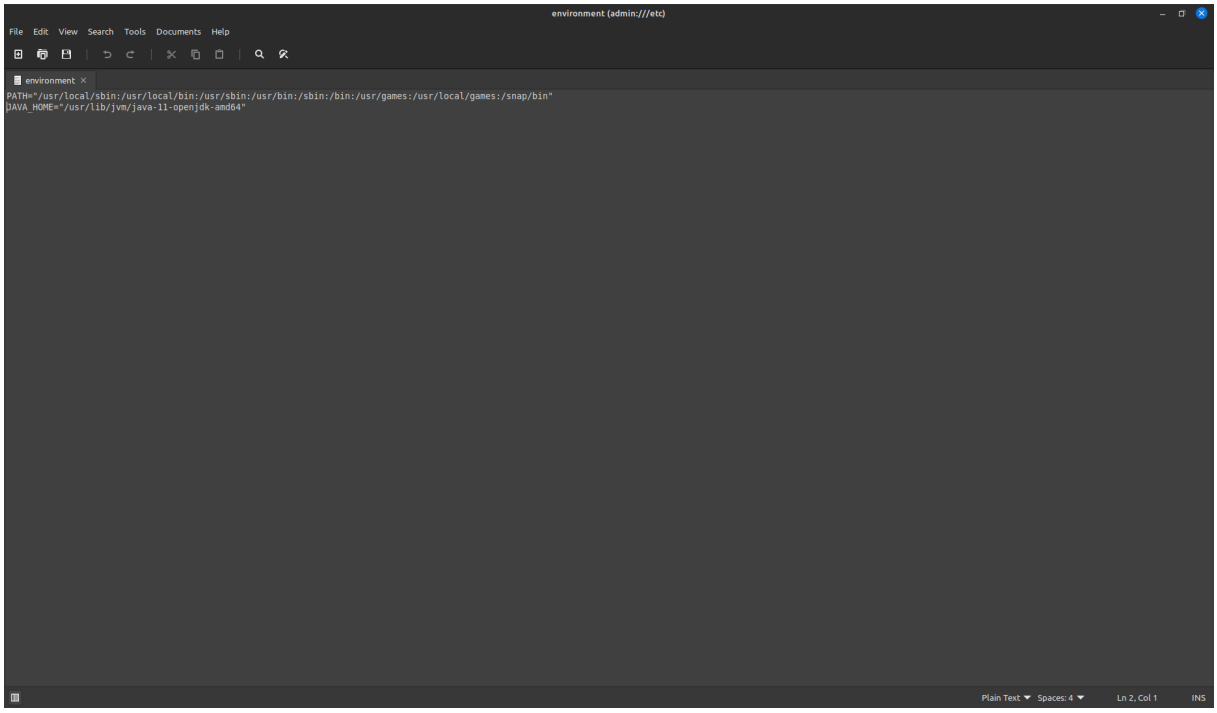



**Figure 1.3:** Step 3: Set up the environment variable JAVA_HOME in the file etc/environment which points to the binary folder of Java, so that Hadoop can use Java when compiling MapReduce programs and execute damemons/processes.

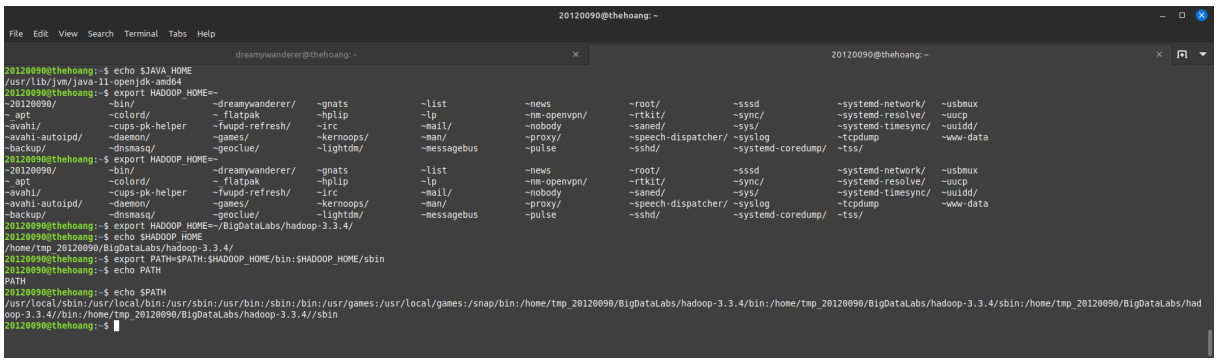



**Figure 1.4:** Step 4: Set up the environment variable HADOOP_HOME in the file ~/.bashrc (or by using export temporarily) which points to the extracted folder of Hadoop, so that it is more convenient to call Hadoop command after that.

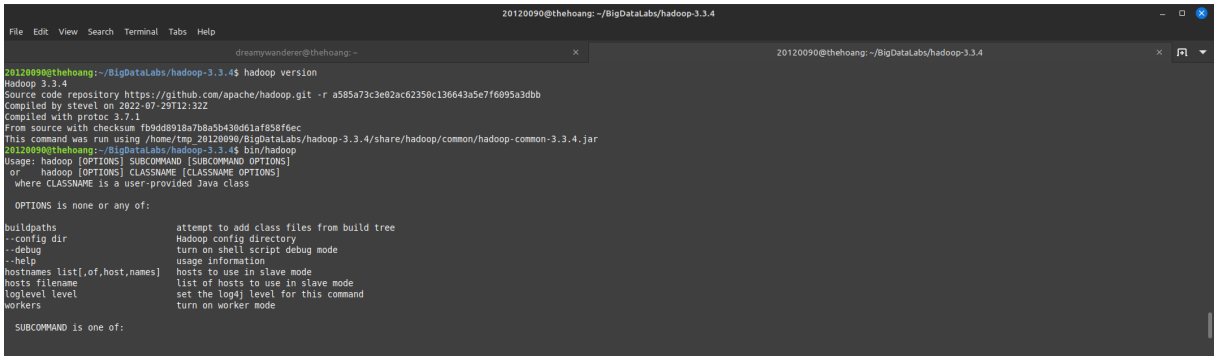



**Figure 1.5:** Step 5: Check that Hadoop runs by typing `hadoop version`

### 1.1.1 Running in Local (Standalone Mode)

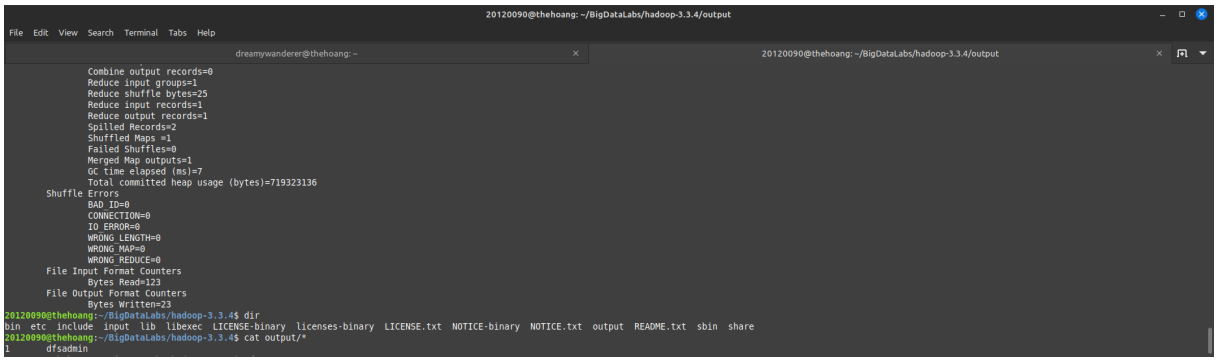



**Figure 1.6:** Step 6: Keep the default configure file of Hadoop. Conduct the Standalone Operation as instructed in [1], the above picture shows the result in the output folder

### 1.1.2 Running in Pseudo-Distributed Mode

**Step 7**: Change the configure files: core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml as instructed in [2] and [^b] to set up for Pseudo-Distributed Mode


**Figure 1.7:** Step 8: Install ssh


**Figure 1.8:** Step 9: Enable passwordless login by generating new SSH key with an empty passphrase

[1] Apache Hadoop, "Hadoop: Setting up a Single Node Cluster," Jul. 29, 2022. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html.
[2] Apache Hadoop, "Hadoop: Setting up a Single Node Cluster," Jul. 29, 2022. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html.

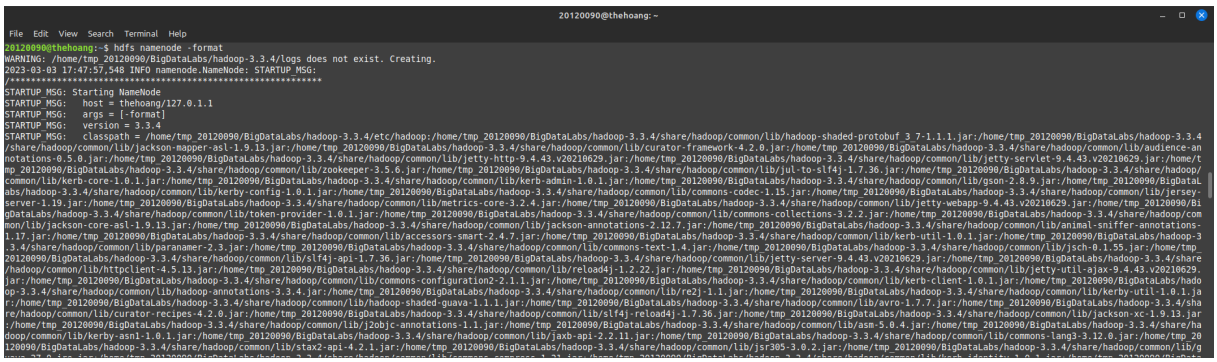



**Figure 1.9:** Step 10: Formatting the HDFS filesystem ssh

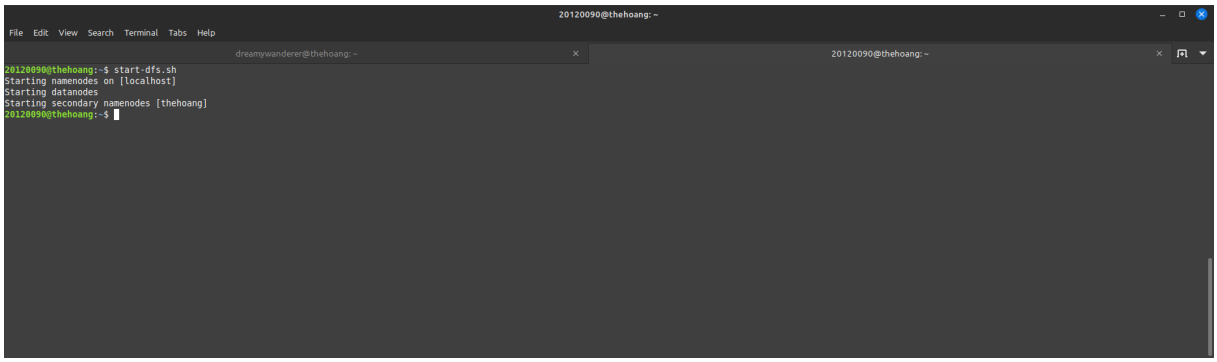



**Figure 1.10:** Step 11: Start NameNode daemon and DataNode daemon with start-dfs.sh

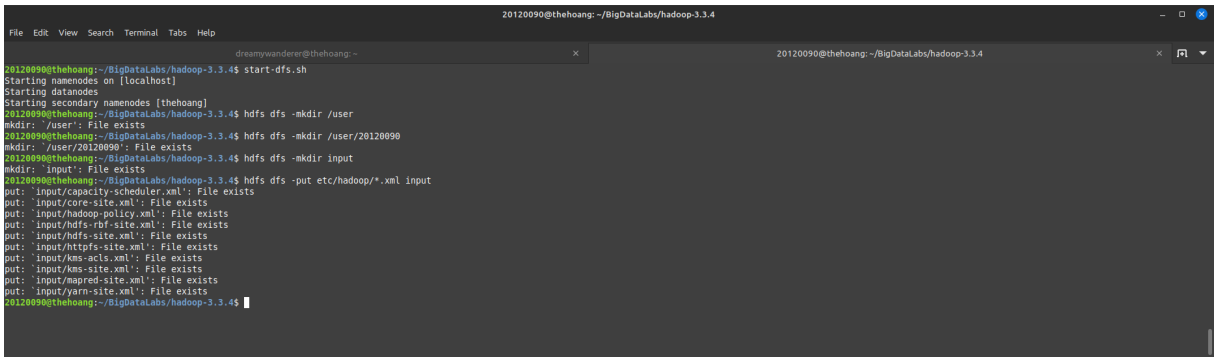



**Figure 1.11:** Step 12: As instructed in [3], make the HDFS directories required to execute MapReduce jobs; copy some local files into the distributed filesystem and run the MapReduce example provided

[3] Apache Hadoop, "Hadoop: Setting up a Single Node Cluster," Jul. 29, 2022. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html.

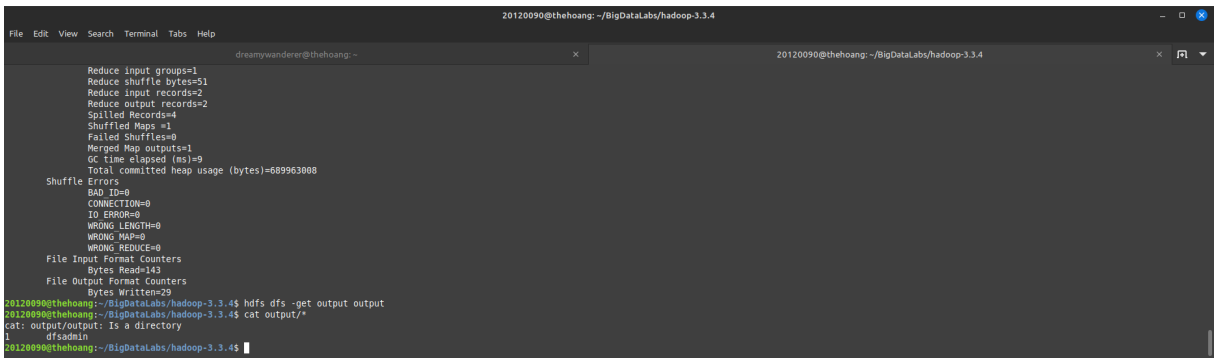



**Figure 1.12:** Step 13: Copy the output files from the distributed filesystem to the local filesystem to examine them

### 1.1.3 Running in Pseudo-Distributed Mode with YARN

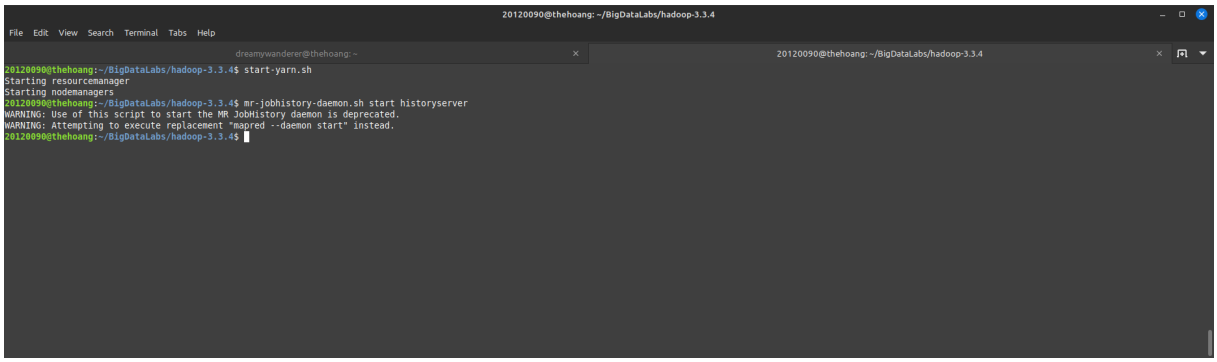



**Figure 1.13:** Step 14: Start the ResourceManager daemon and NodeManager daemon by running start-yarn.sh. Optionally, start the history server by running mr-jobhistory-daemon.sh start historyserver

### 1.1.4 Accessing the Web interfaces of running Hadoop process

These are the results when start all main components of a Single-node Hadoop Cluster

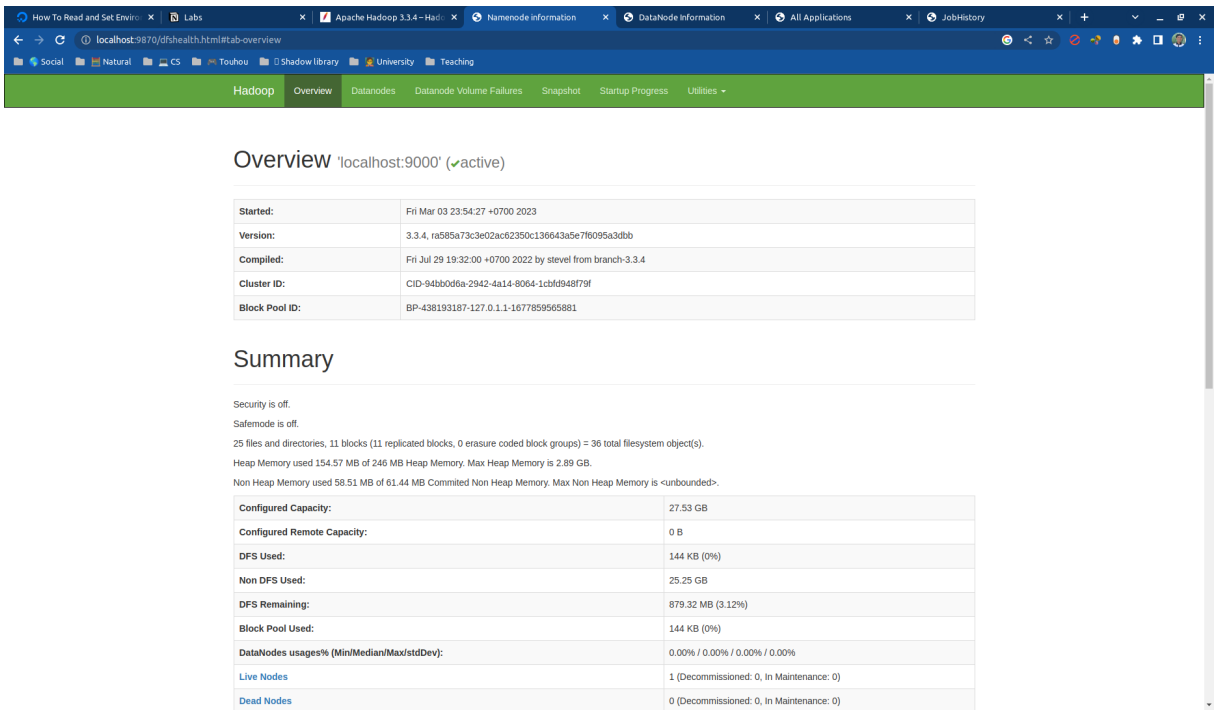



**Figure 1.14:** Web UI of NameNode through http://localhost:9870/

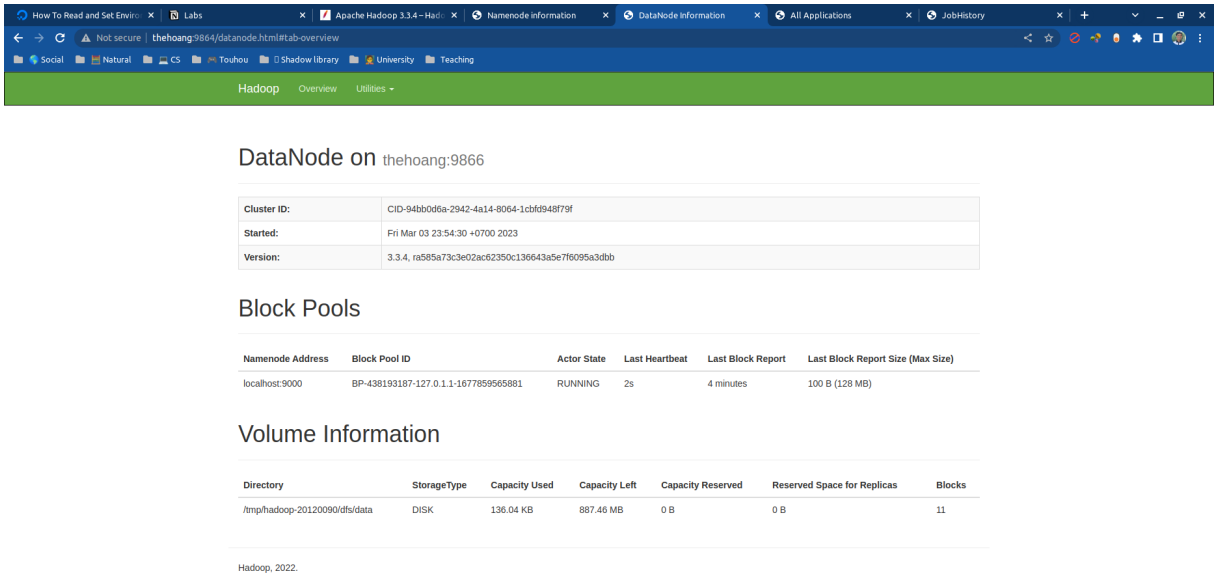



**Figure 1.15:** Web UI of a DataNode

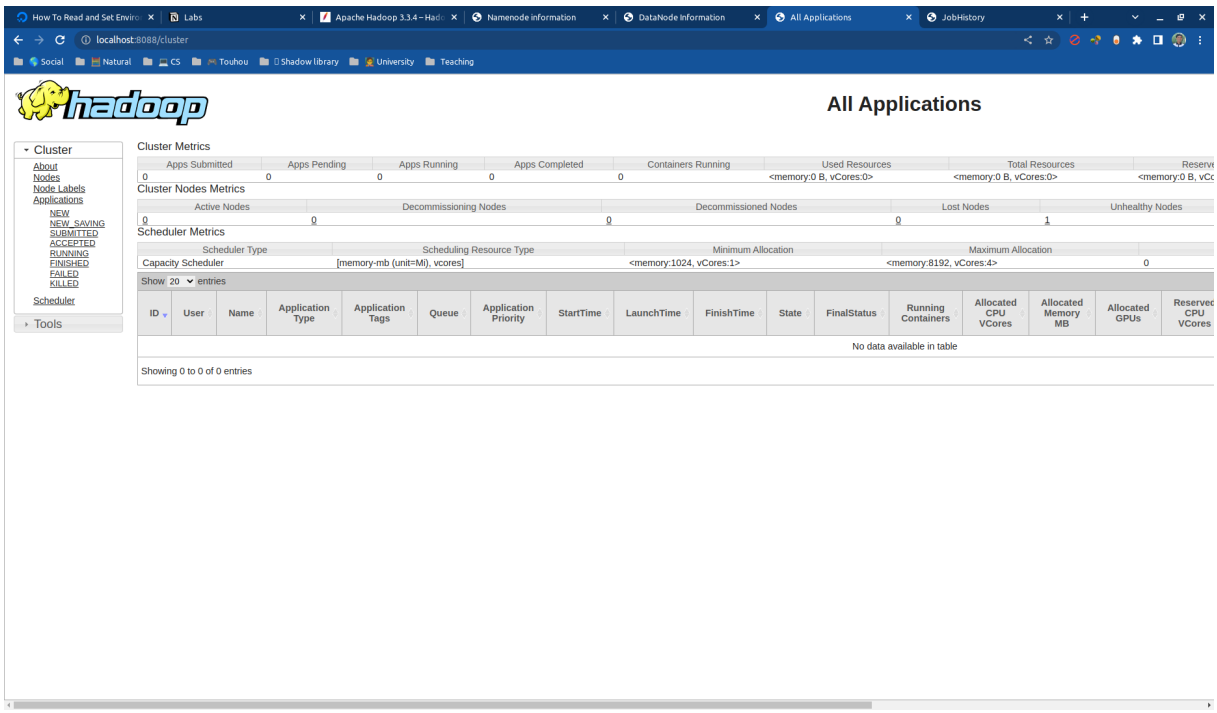



**Figure 1.16:** Web UI of a ResourceManager through http://localhost:8088/

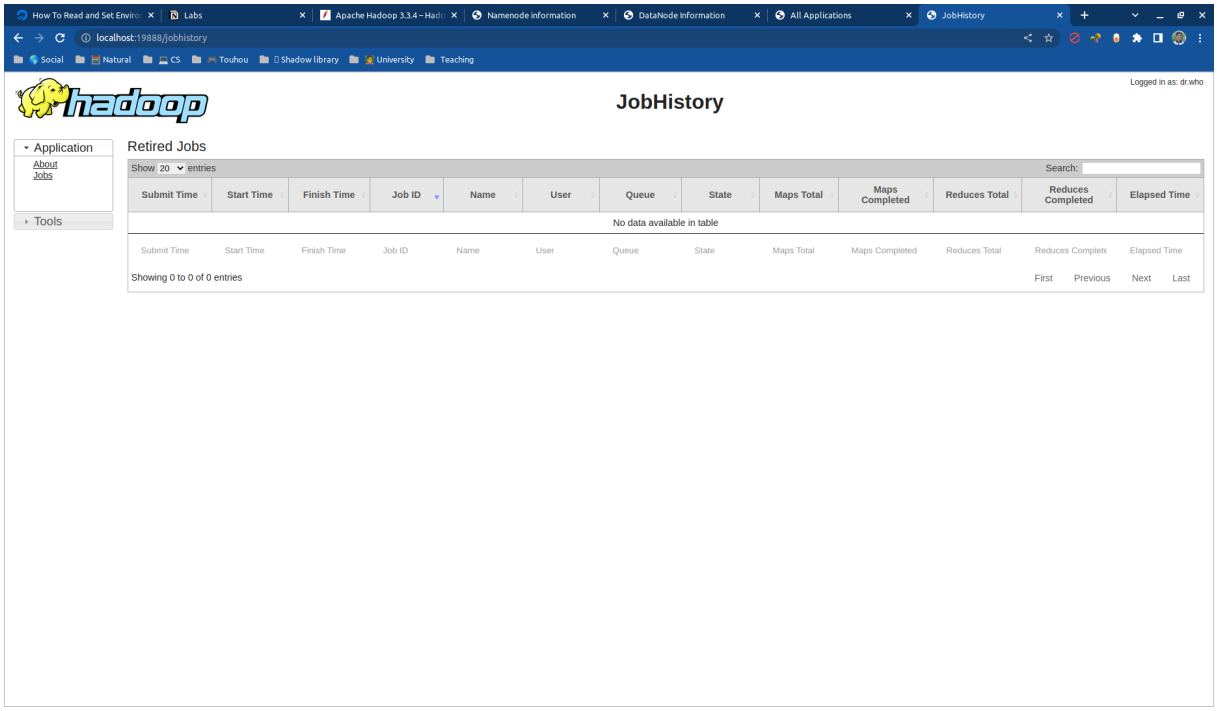



**Figure 1.17:** Web UI of MapReduce history server http://localhost:19888/

## 1.2 Member 2: 20120011 - Nguyen Hoang Huy

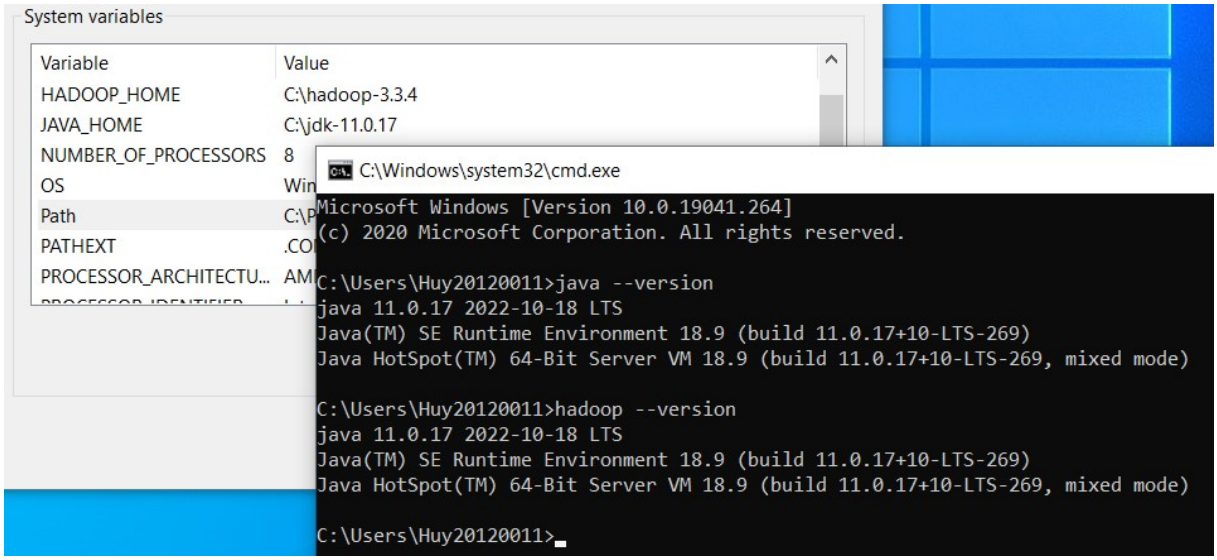



**Figure 1.18:** Install Java and Hadoop then setup the environment path

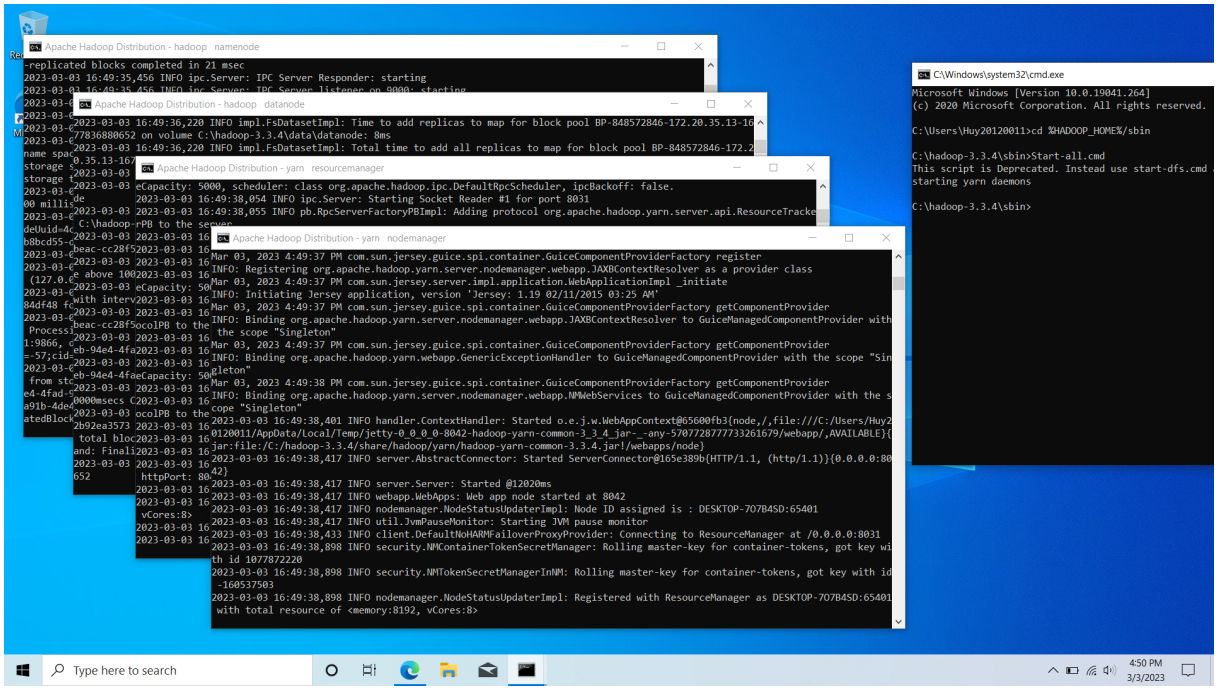



**Figure 1.19:** Run start-all.cmd to start namenode, datanode, resourcemanager, nodemanager program

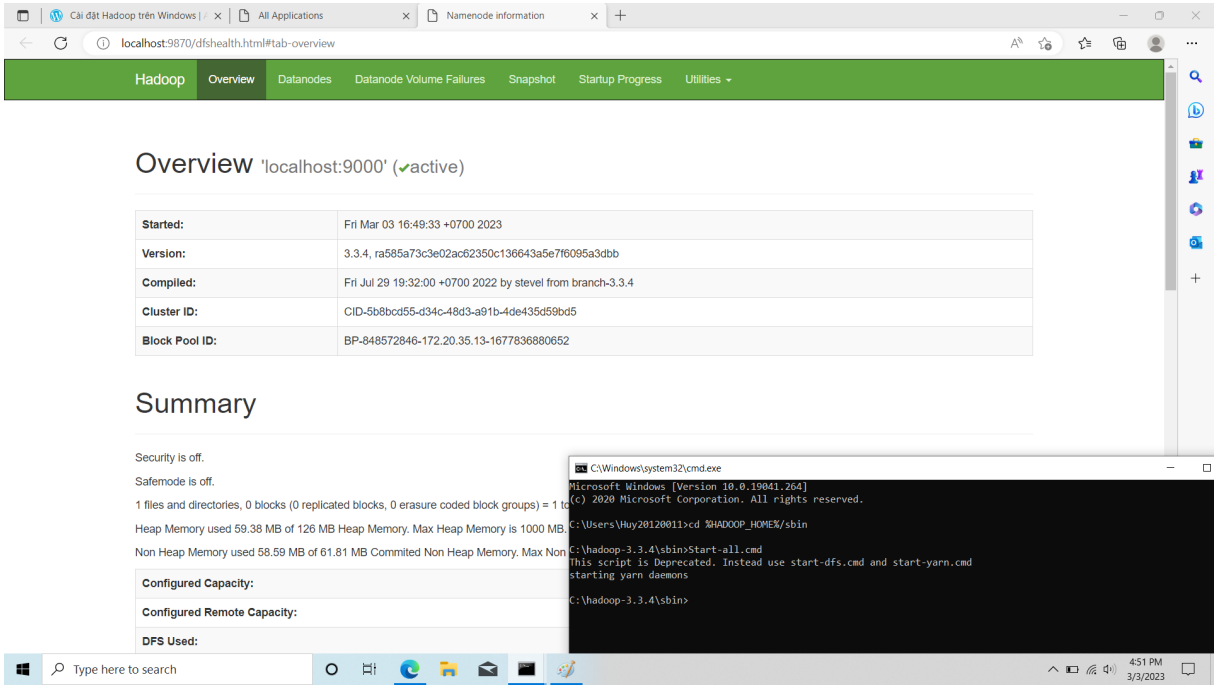



**Figure 1.20:** Access localhost:9000

## 1.3 Member 3: 20120030 - Nguyen Thien An

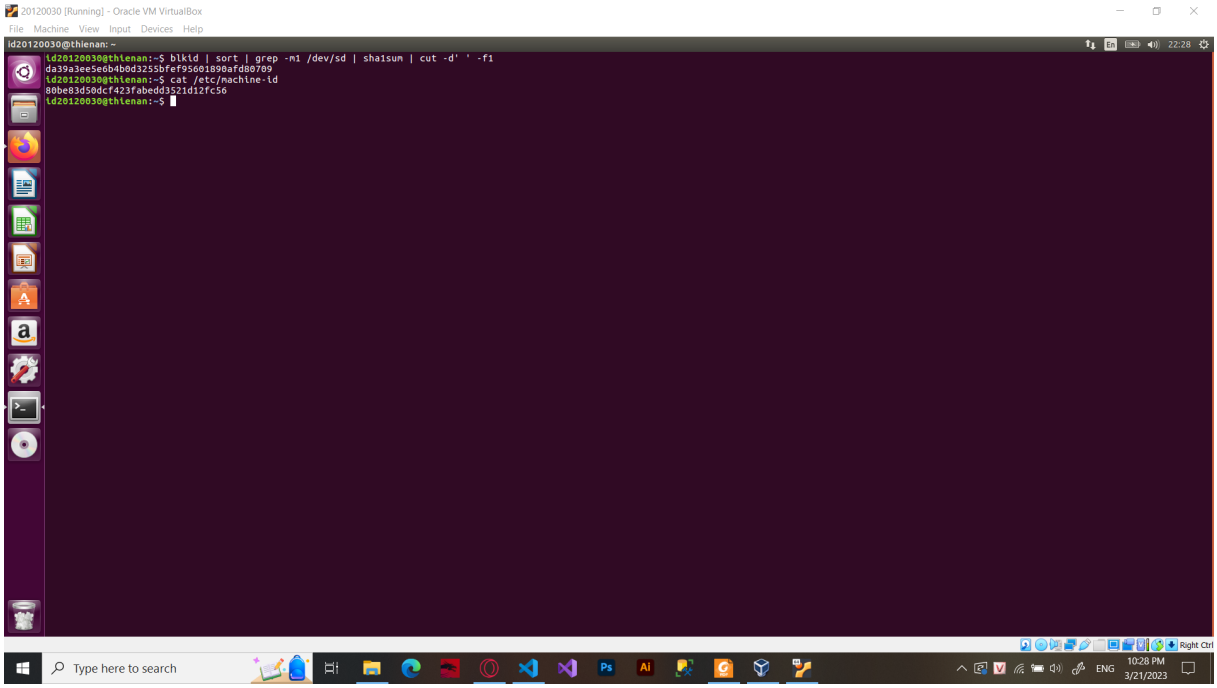



**Figure 1.21:** Id of the first block to proof this is a personal computer.

```
id20120030@thienan:~$ hadoop version
Hadoop 3.2.4
Source code repository Unknown -r 7e5d9983b388e372fe640f21f048f2f2ae6e9eba
Compiled by ubuntu on 2022-07-12T11:58Z
Compiled with protoc 2.5.0
From source with checksum ee031c16fe785bbb35252c749418712
This command was run using /home/id20120030/hadoop-3.2.4/share/hadoop/common/hadoop-common-3.2.4.jar
id20120030@thienan:~$ javac -version
javac 1.8.0_292
id20120030@thienan:~$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~16.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
id20120030@thienan:~$
```

**Figure 1.22:** Install Java and Hadoop then setup the environment path

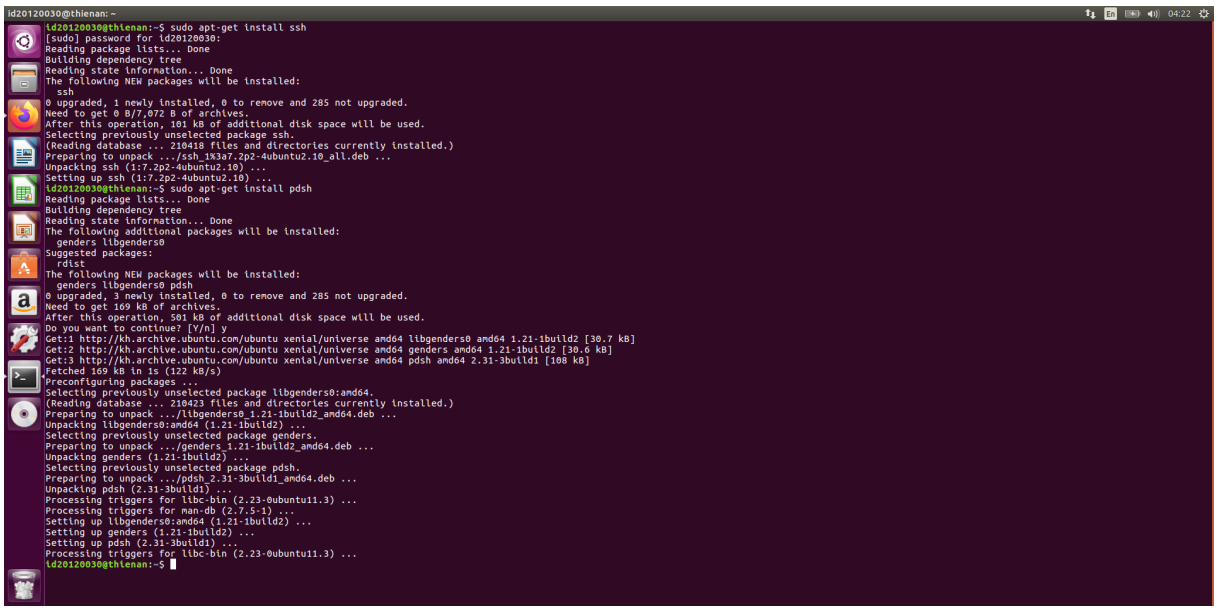



**Figure 1.23:** Install ssh and pdsh

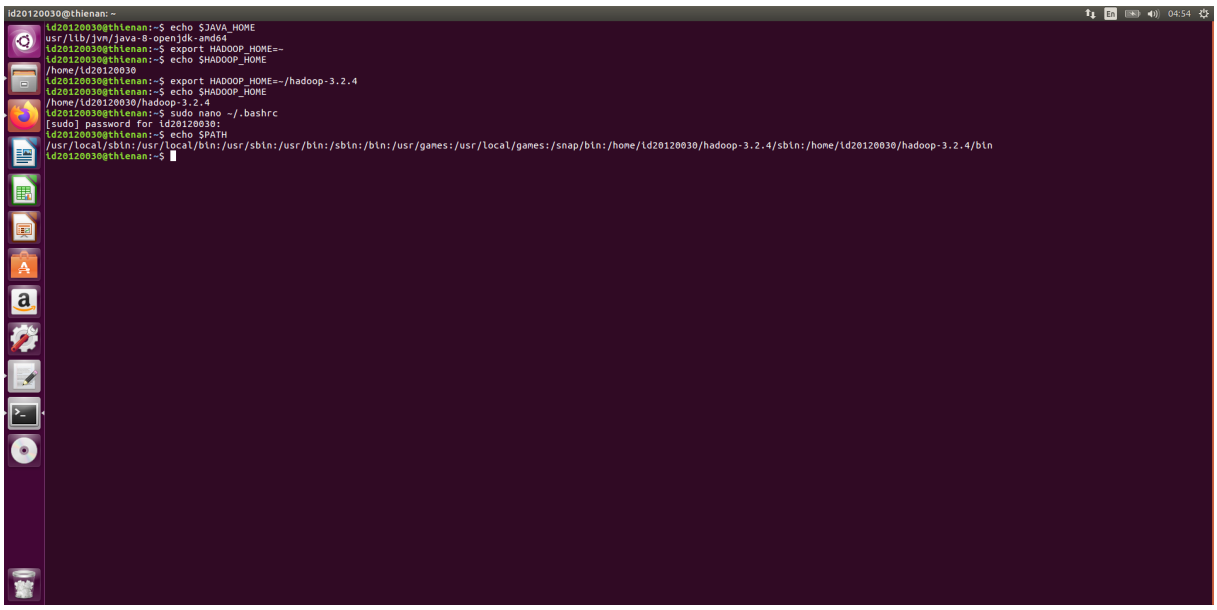



**Figure 1.24:** JAVA_HOME and HADOOP_HOME

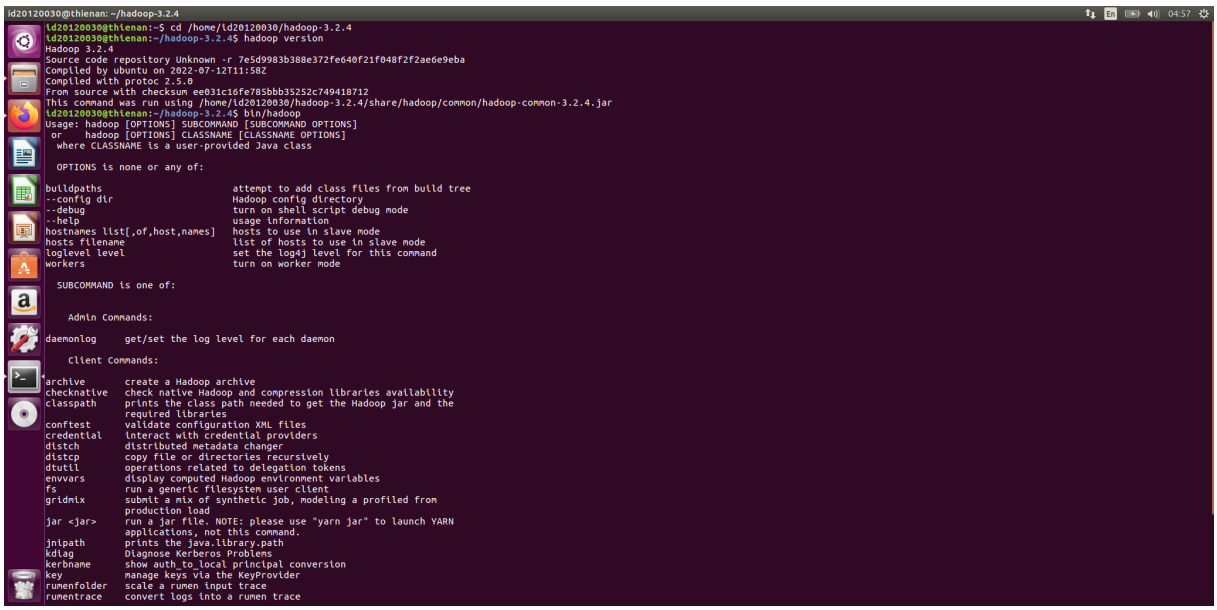



**Figure 1.25:** Show hadoop version and bin/hadoop

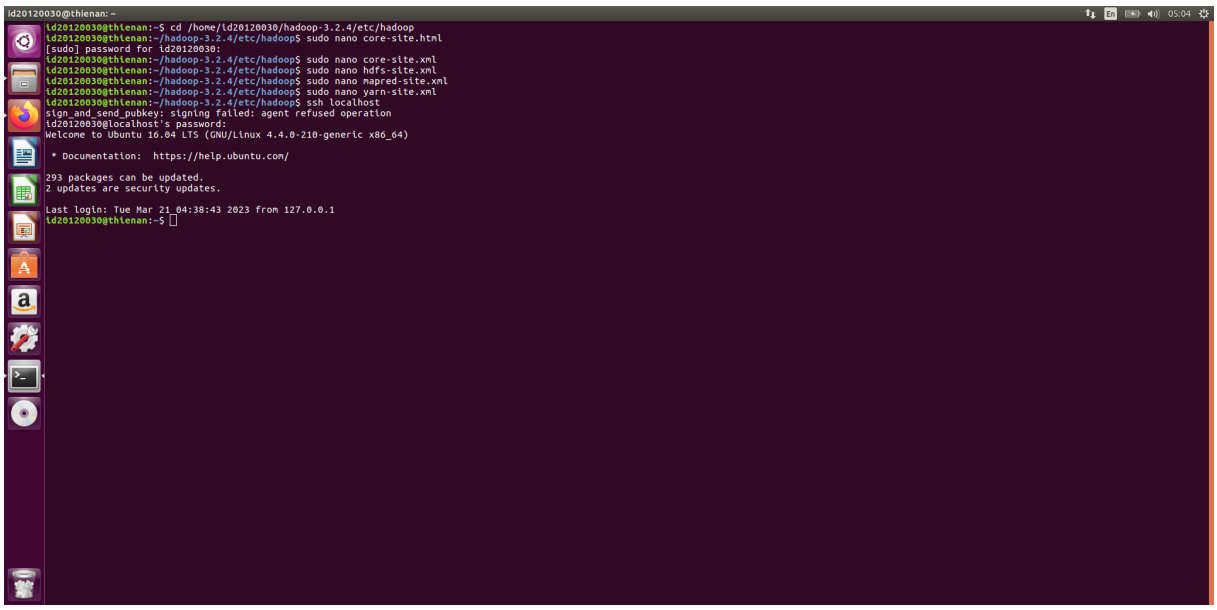



**Figure 1.26:** Make changes on core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml

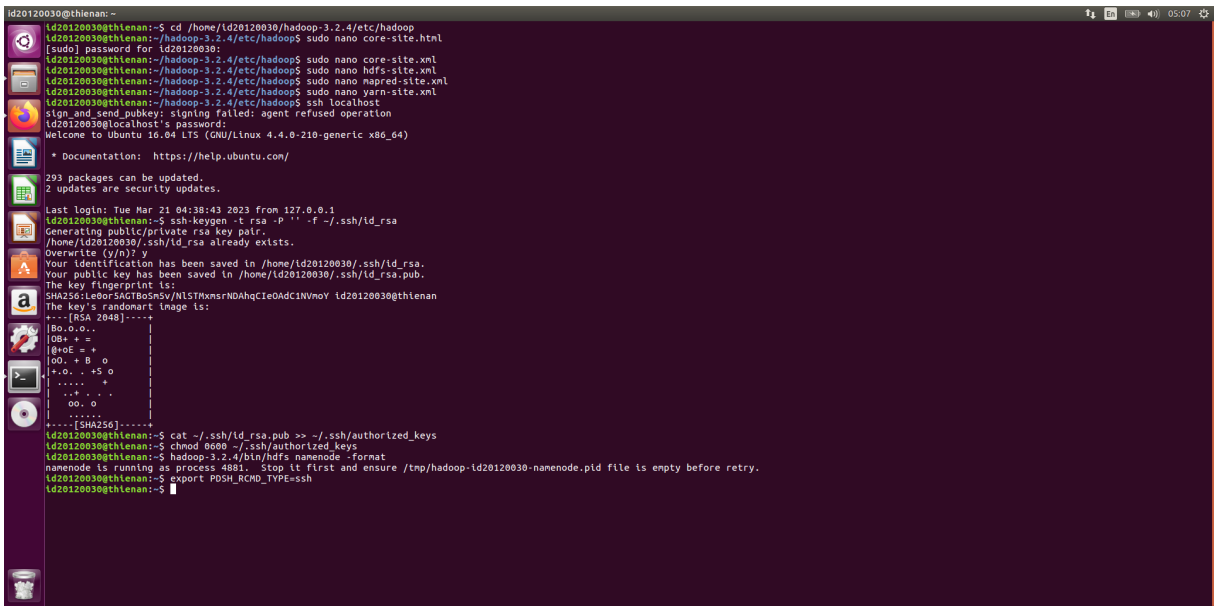



**Figure 1.27:** ssh localhost

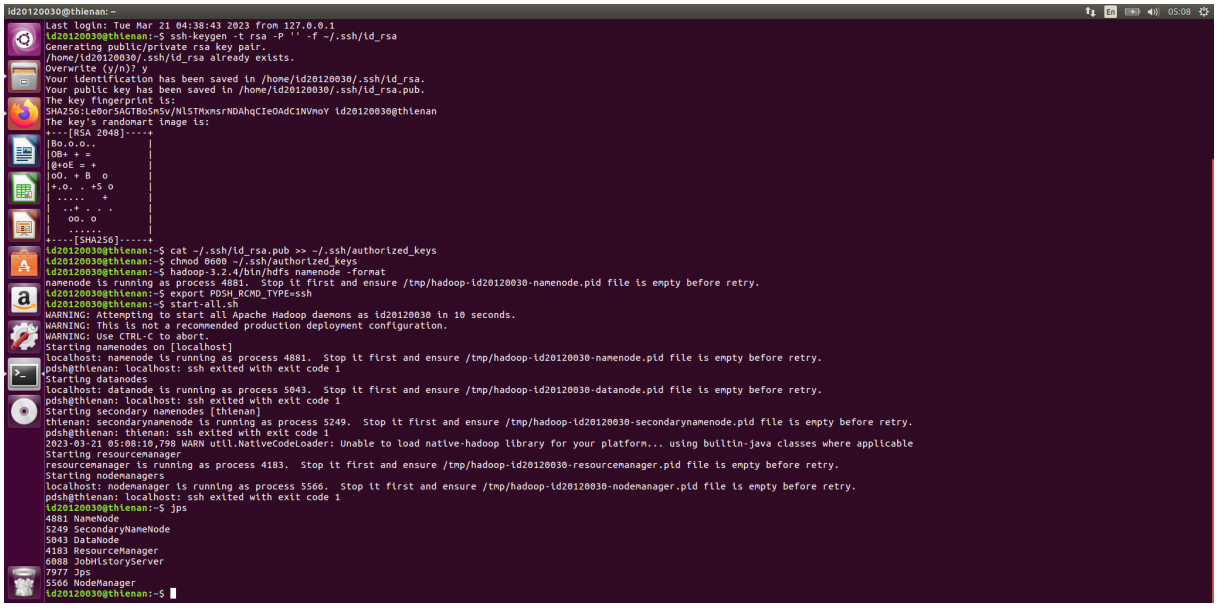



**Figure 1.28:** Start all daemons hadoop

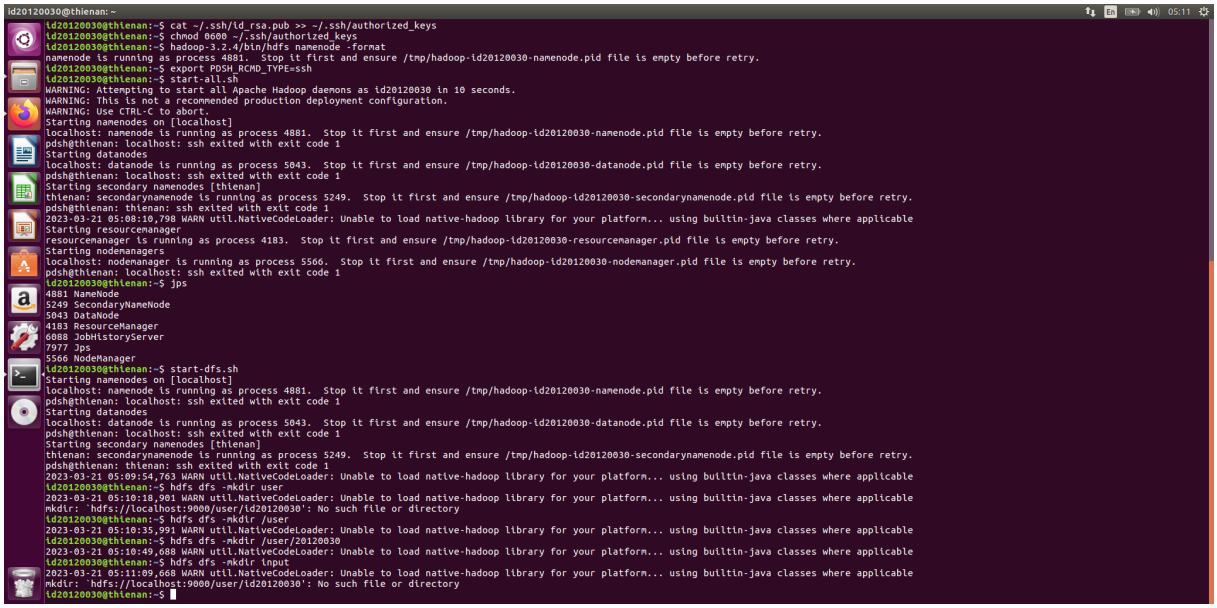



**Figure 1.29:** Create a folder named 20120030 in /user

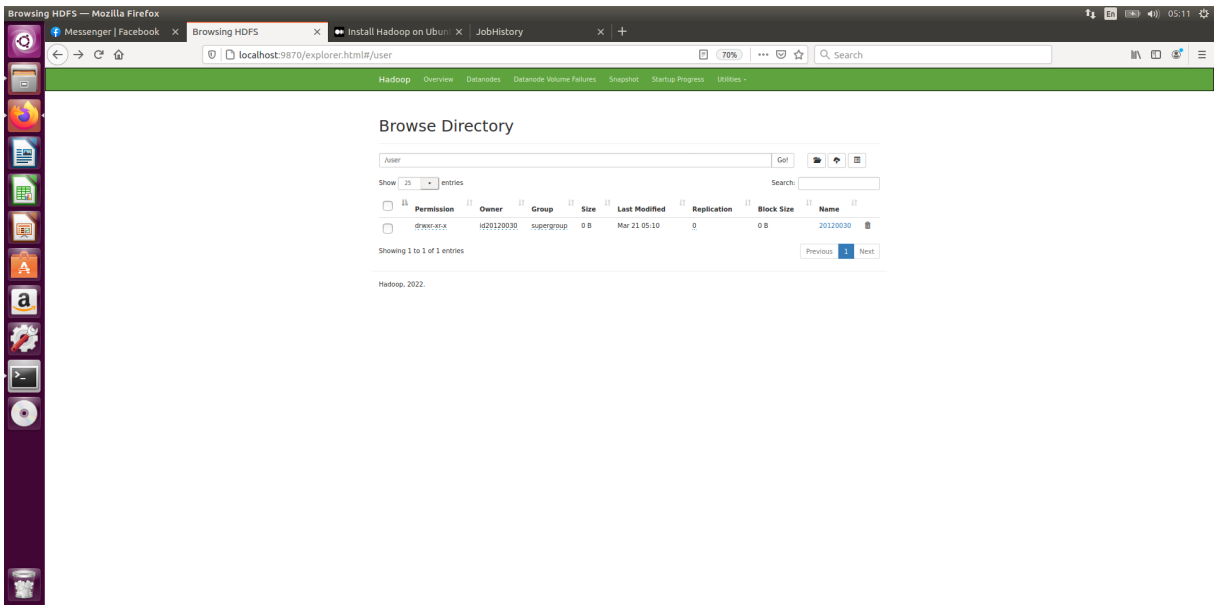



**Figure 1.30:** Create folder 20120030 successfully!

```
id20120030@thienan:~/hadoop-3.2.4$ start-yarn.sh
Starting resourcemanager
resourcemanager is running as process 18086.  Stop it first and ensure /tmp/hadoop-id20120030-resourcemanager.pid file is empty before retry.
Starting nodemanagers
localhost: nodemanager is running as process 18224.  Stop it first and ensure /tmp/hadoop-id20120030-nodemanager.pid file is empty before retry.
id20120030@thienan:~/hadoop-3.2.4$ mr-jobhistory-daemon.sh start historyserver
WARNING: Use of this script to start the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon start" instead.
id20120030@thienan:~/hadoop-3.2.4$
```

**Figure 1.31:** Start yarn and historyserver

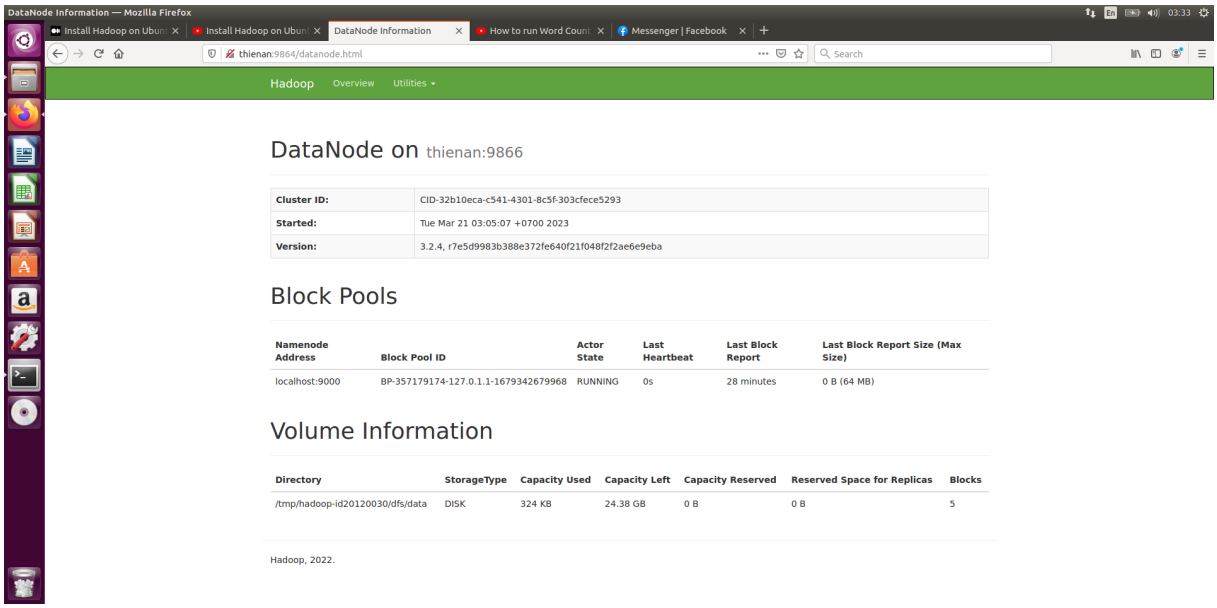



**Figure 1.32:** DataNode

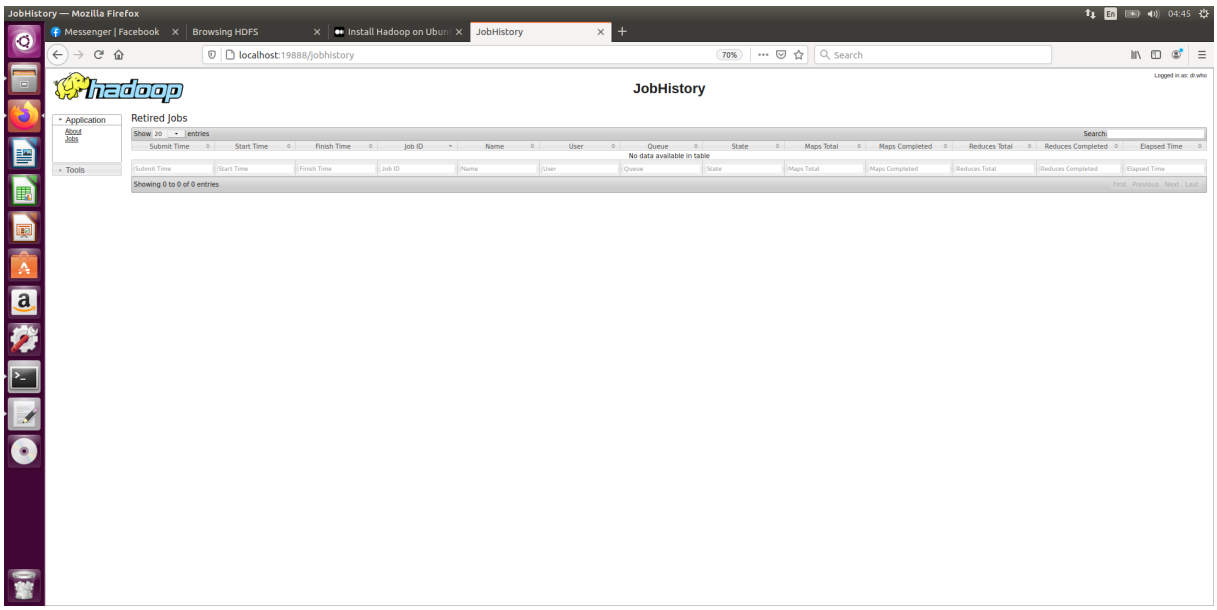



**Figure 1.33:** MapReduce Server

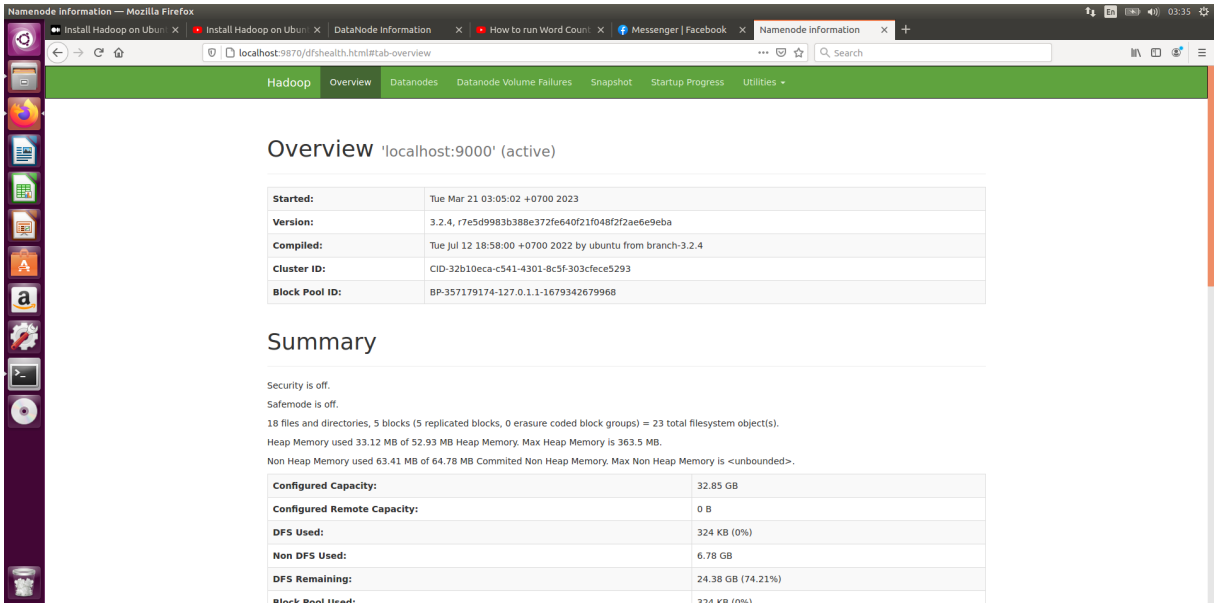



**Figure 1.34:** NameNode

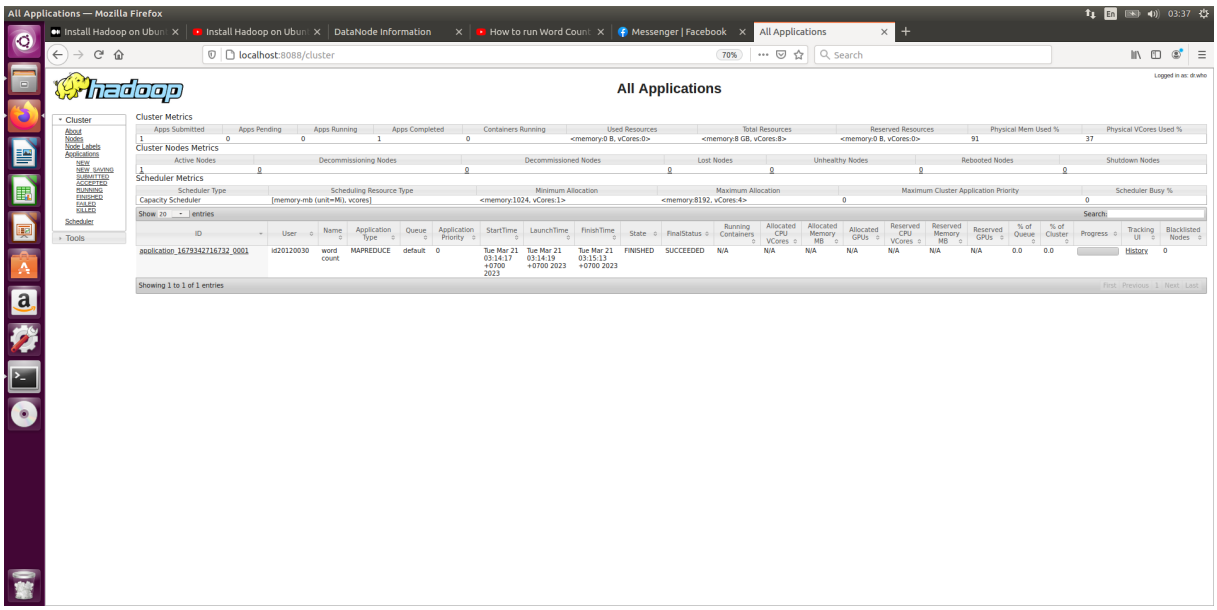



**Figure 1.35:** YarnNode

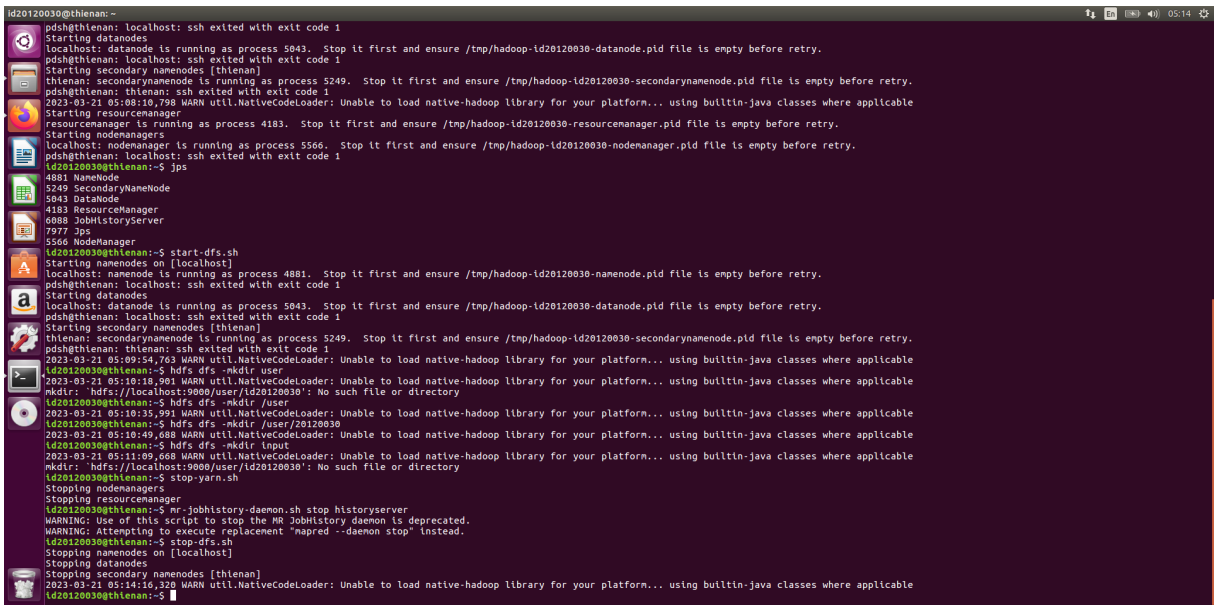



**Figure 1.36:** Stop all daemons hadoop

## 1.4 Member 4: 20120165 - Hong Nhat Phuong

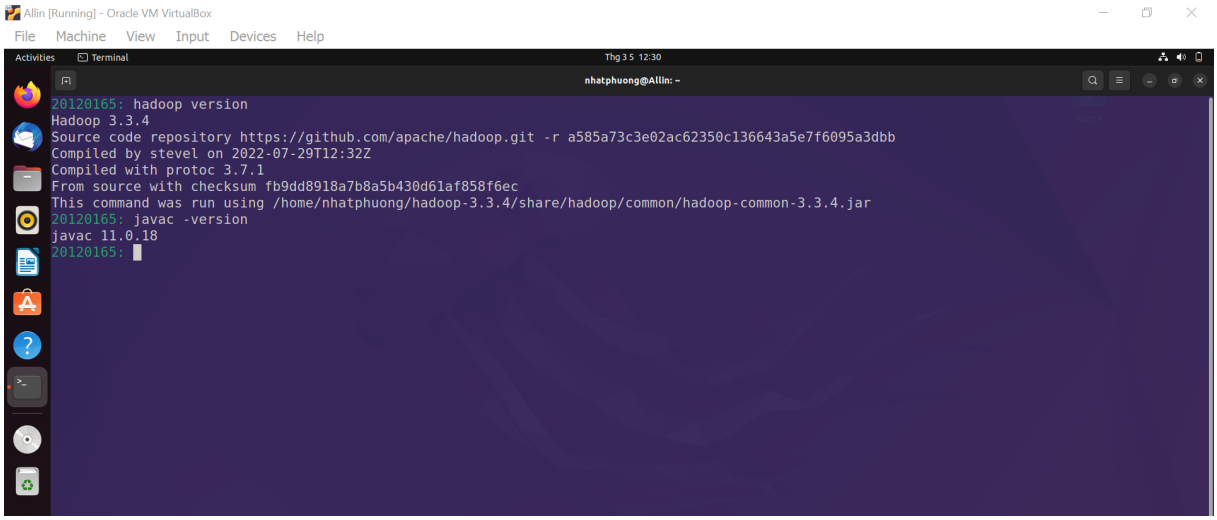



**Figure 1.37:** Install Java and Hadoop then setup the environment path

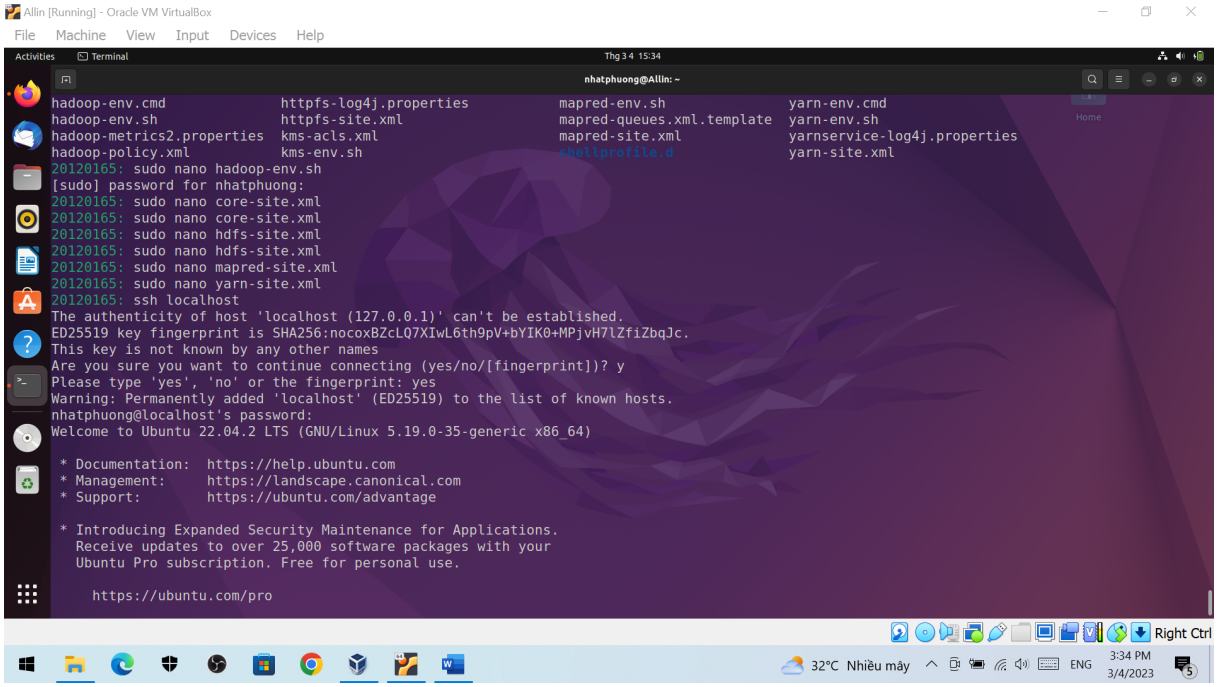



**Figure 1.38:** Make changes on core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml

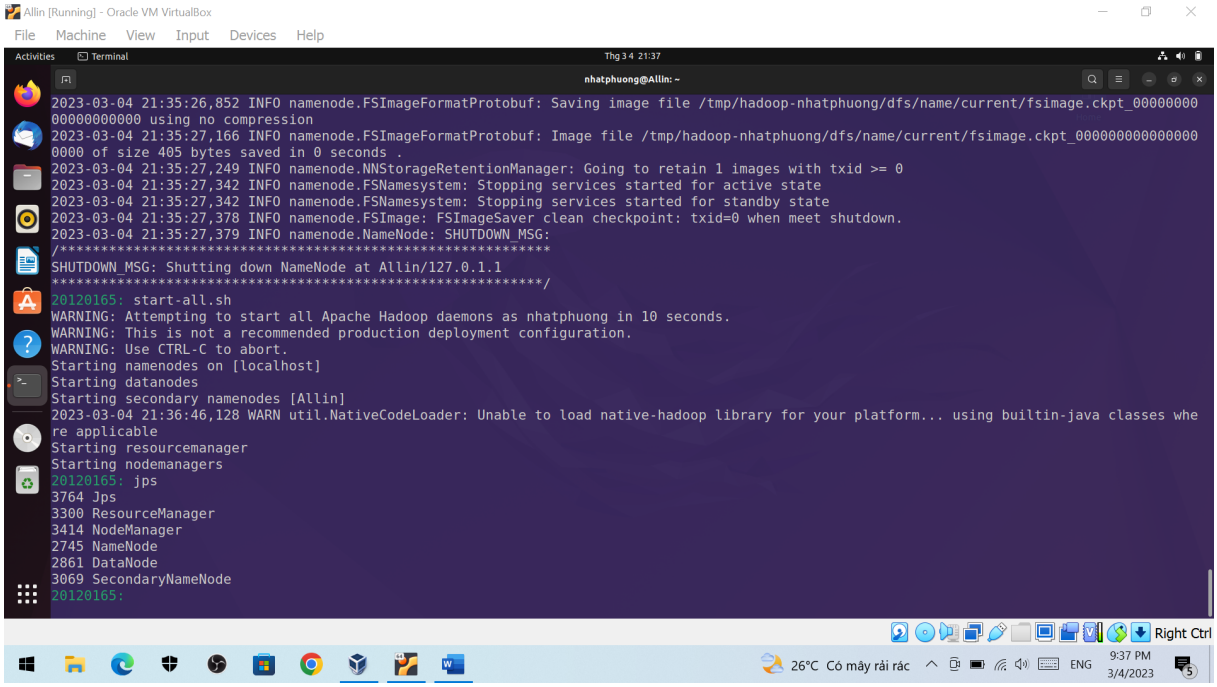



**Figure 1.39:** Format namenode and start all daemons hadoop

# 2 Introduction to MapReduce

1. How do the input keys-values, the intermediate keys-values, and the output keys-values relate?

   The input keys-values are the data sets that are processed by the Map function. The input data is divided into key-value pairs that are distributed across the nodes in the Hadoop cluster. The Map function processes each key-value pair and produces an intermediate key-value pair.

   The intermediate keys-values are produced by the Map function and are used as input to the Reduce function. The intermediate key-value pairs are sorted by key and partitioned based on the key. Each partition is processed by a separate instance of the Reduce function.

   The output keys-values are the final result of the MapReduce process. The Reduce function produces the final key-value pairs which are written to the Hadoop Distributed File System (HDFS) or to an external storage system.

2. How does MapReduce deal with node failures?

   When the compute node at which the master is executing fails, a new copy can be started from the last checkpointed state. Only this one node can bring the entire process down; other node failures will be managed by the master.

   The master sends heartbeat to every worker periodically. If there is no response from the worker within a certain period of time, the master marks the worker as failed. Map tasks that were assigned by that worker will be reset to their original idle state and eligible for scheduling by other workers, even if they have completed. Completed map tasks are re-executed because their output is stored at that compute node, and is now unavailable to access.

   Similarly, reduce task in progress on the failed worker is also reset to idle and will have to be redone. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system. When a map task is executed first by a worker and then later executed by another worker (because the first one failed), all workers executing reduce tasks are notified of the re-execution.

3. What is the meaning and implication of locality? What does it use?

   Locality means that taking advantage of the fact that the input data is stored on the local disks of the machine that make up the clusters. As a result, the network bandwidth is conserved, and

when running large MapReduce operations on a large cluster, most input is read locally, consume no network bandwidth, speed processes up significantly.

Each file is divided into same size blocks (default 64 MB), then each block is created multiple copies (typically 3 copies). These blocks are stored on different machine, but highly optimised location (maybe 2 copies in the same rack, and 1 copy in the other rack in case of failing). MapReduce master take the location of the input files and above blocks into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data. If that is impossible, it trys to schedule map tasks near a replica of that task's input data (maybe on the same rack). Locality is also the result of Rack awareness of Hadoop system.

4. Which problem is addressed by introducing a combiner function to the MapReduce model?

   The combiner function introduced to the MapReduce model addresses the problem of **network congestion** and **data transfer**. In detail, When data is processed by the Map function, it generates a large number of key-value pairs, which are then sorted and shuffled before being sent to the Reduce function. This sorting and shuffling process can generate a significant amount of network traffic, especially if there are many duplicate keys. The combiner function is introduced after the Map phase and before the Shuffle and Sort phase. Its purpose is to aggregate the output of the Map function for each key, reducing the amount of data that needs to be transferred across the network. This can greatly reduce network congestion and improve the overall performance and scalability of the MapReduce job. Hence, the combiner function reduces network traffic, which is the main problem addressed by its introduction to the MapReduce model.

# 3 Running a warm-up problem: Word Count

Each member has completed running the WordCount (v.1) program - a simple MapReduce program - on a Single node cluster on their local machine, as described in supplied tutorial. Also, this process is captured in screenshots.

## 3.1 Member 1: 20120011 - Nguyen Hoang Huy

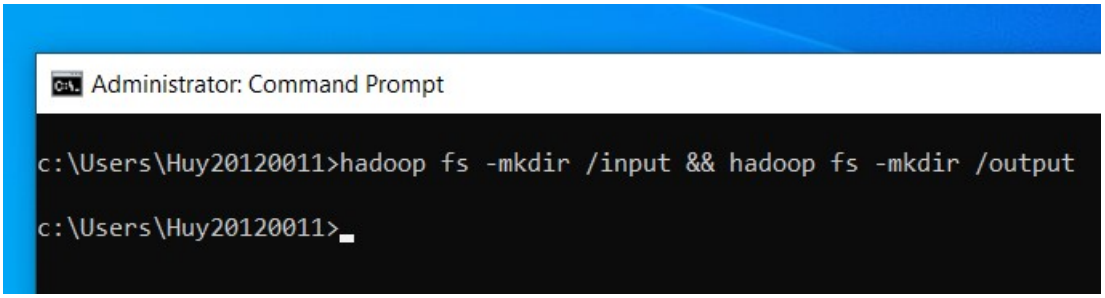



**Figure 3.1:** Create input and output directory in hadoop fs

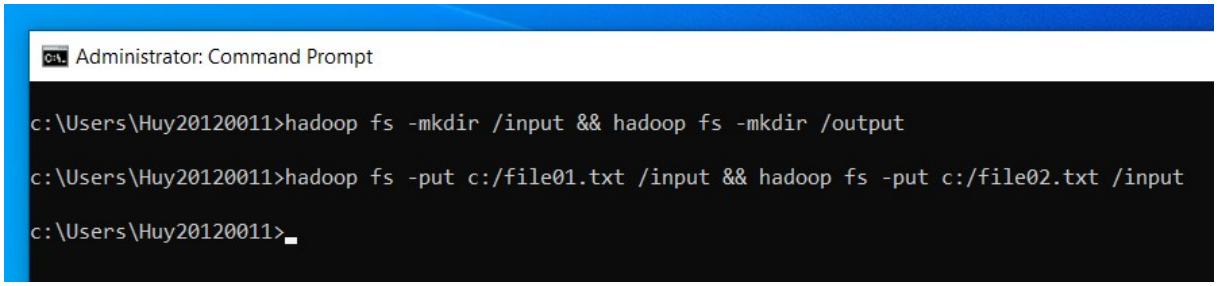



**Figure 3.2:** Put file01.txt and file02.txt into input directory

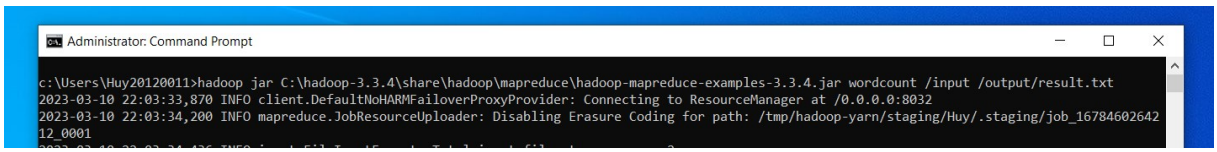



**Figure 3.3:** Run WordCount program

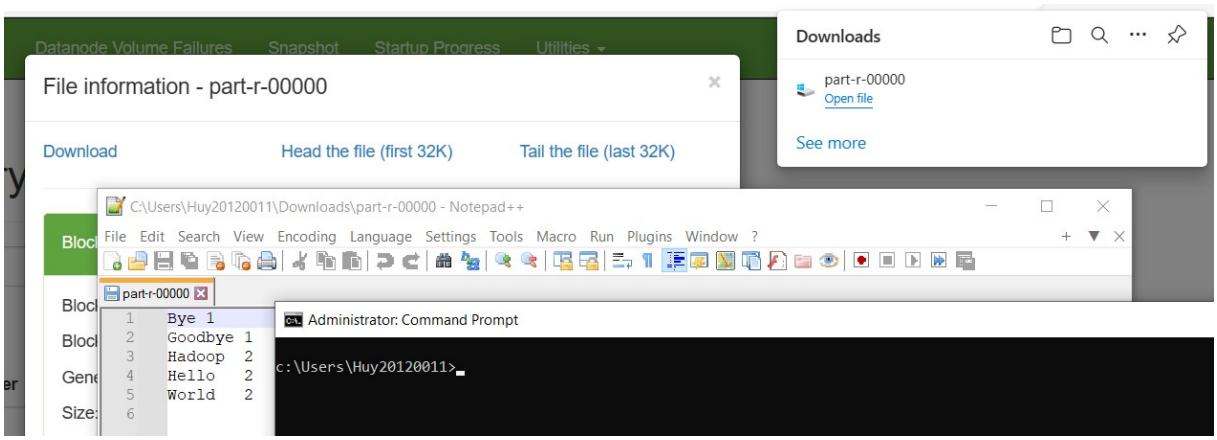



**Figure 3.4:** Let's show the result

## 3.2 Member 2: 20120030 - Nguyen Thien An

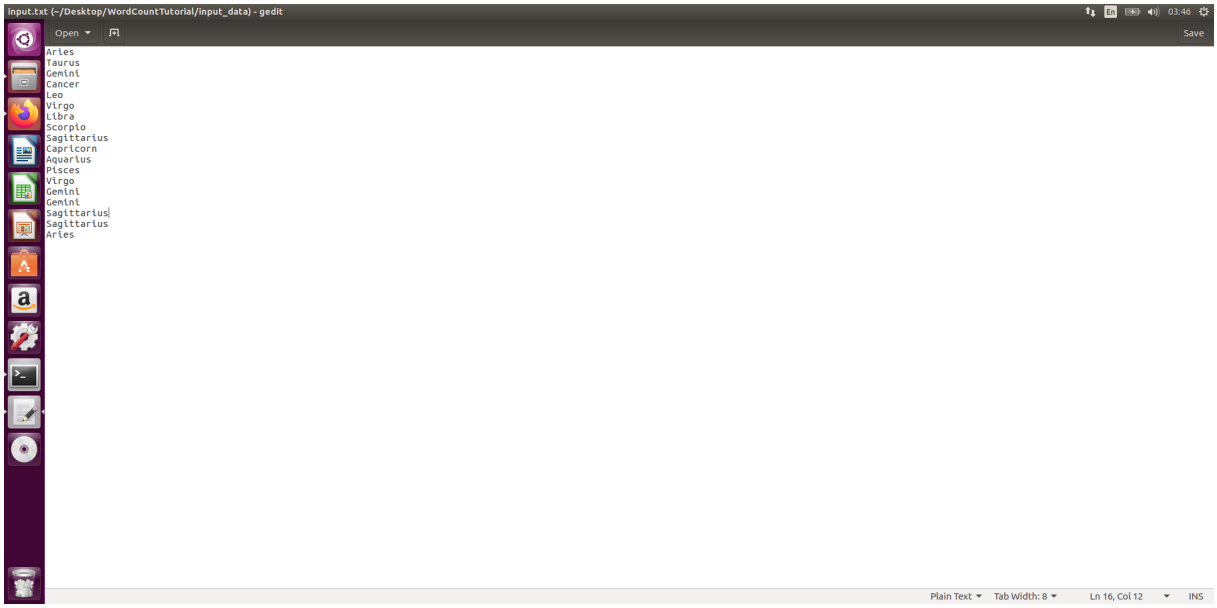



**Figure 3.5:** File "input.txt"

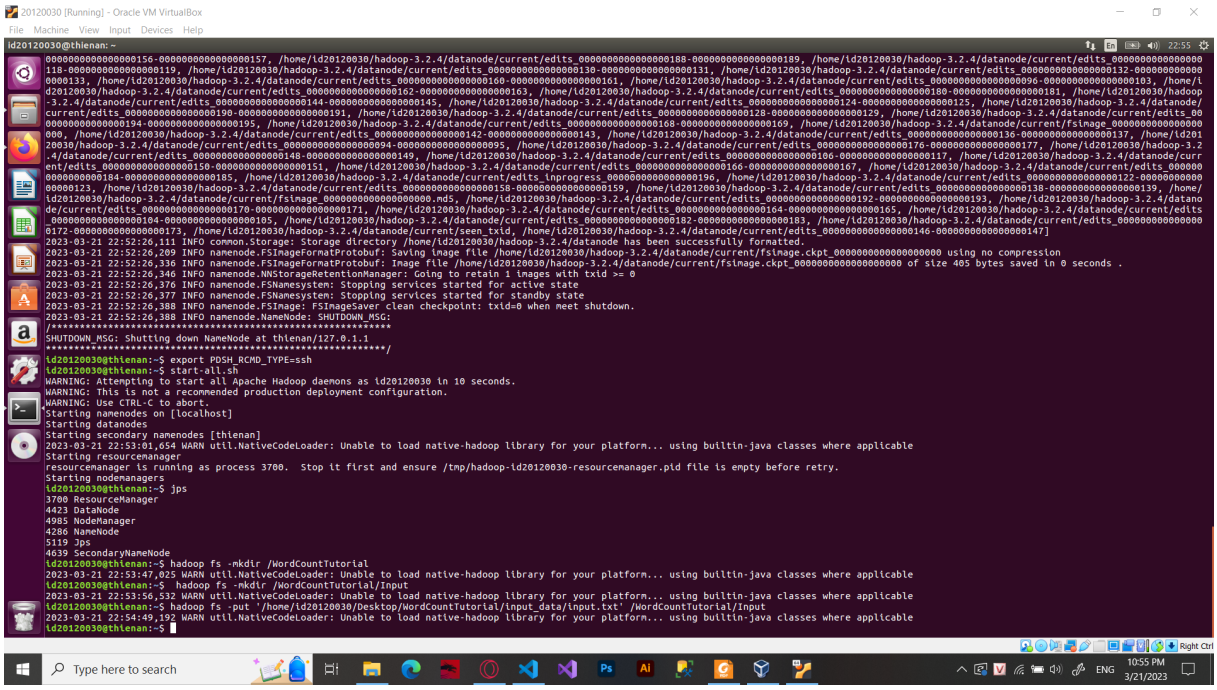



**Figure 3.6:** Create folder input and put input.txt into input directory

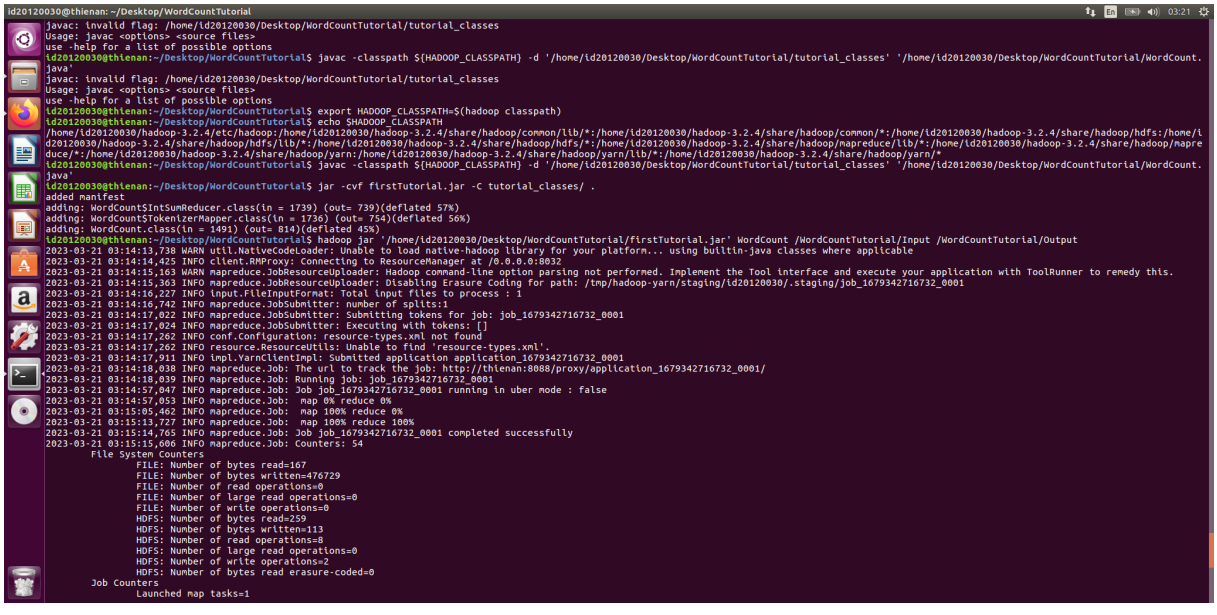



**Figure 3.7:** Compile WordCount.java, create a jar and run WordCount program


**Figure 3.8:** Show the output using -cat command

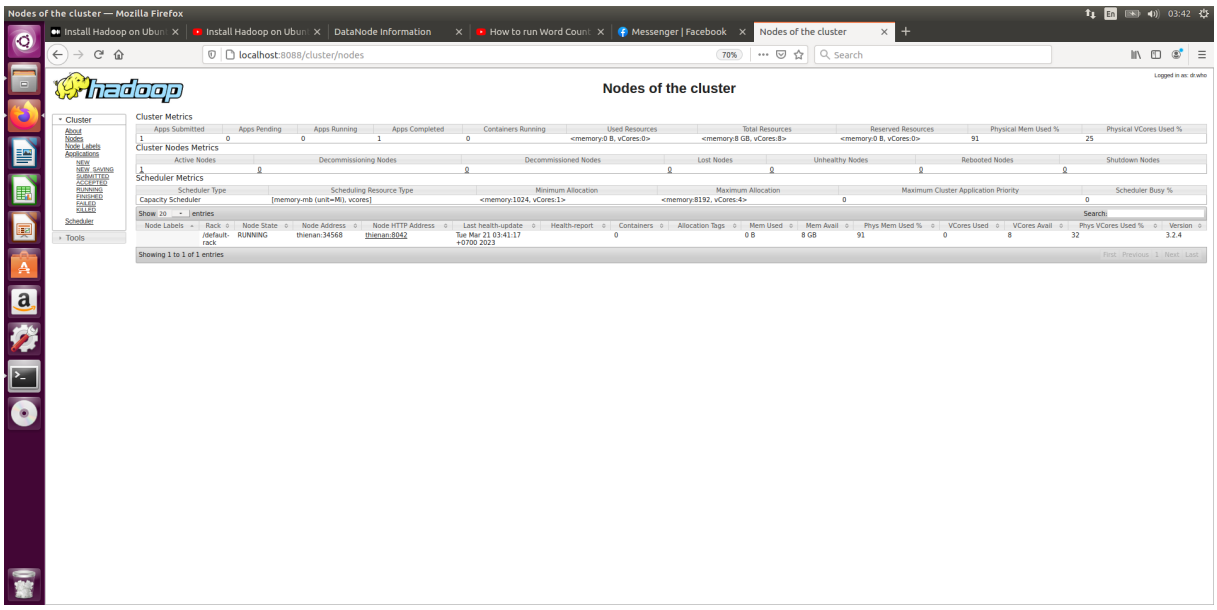



**Figure 3.9:** Nodes of the cluster

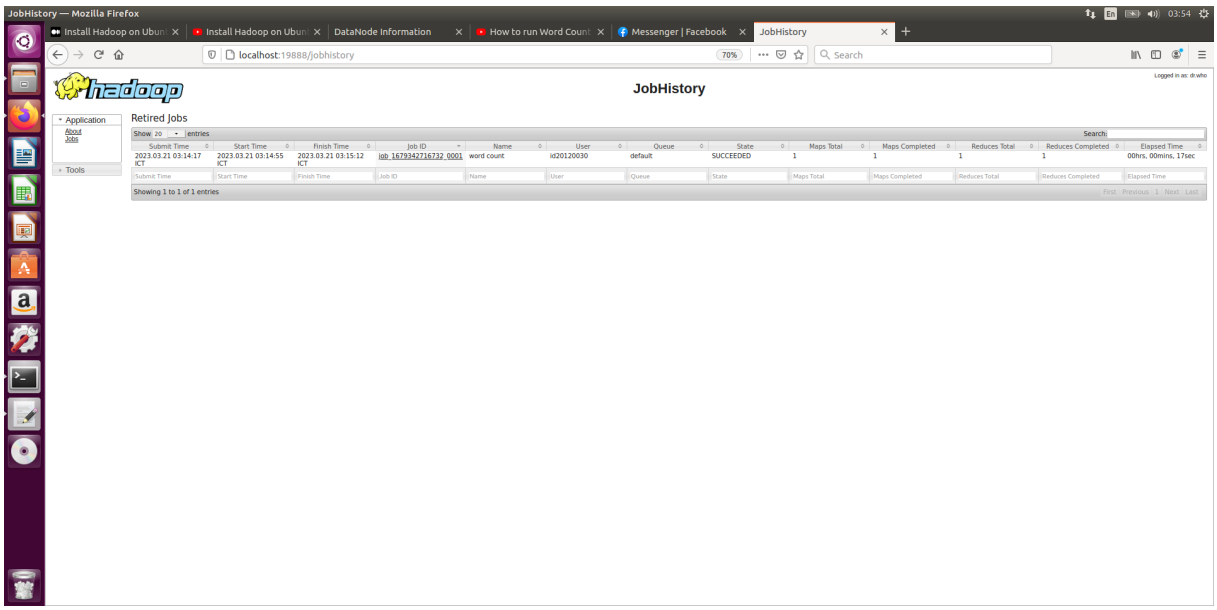



**Figure 3.10:** MapReduce Server

## 3.3 Member 3: 20120090 - Nguyen The Hoang

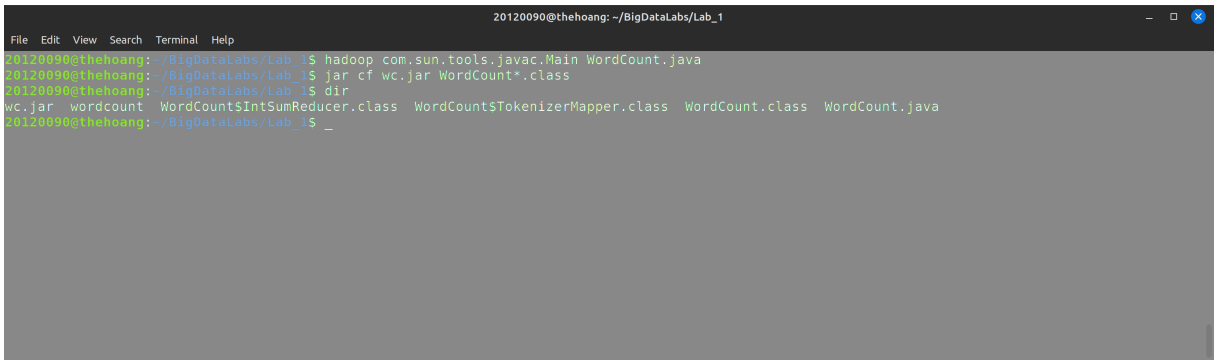



**Figure 3.11:** Step 1: Create the WordCount v1.0 file as instructed in [^c]. Compile this file to the class and jar file

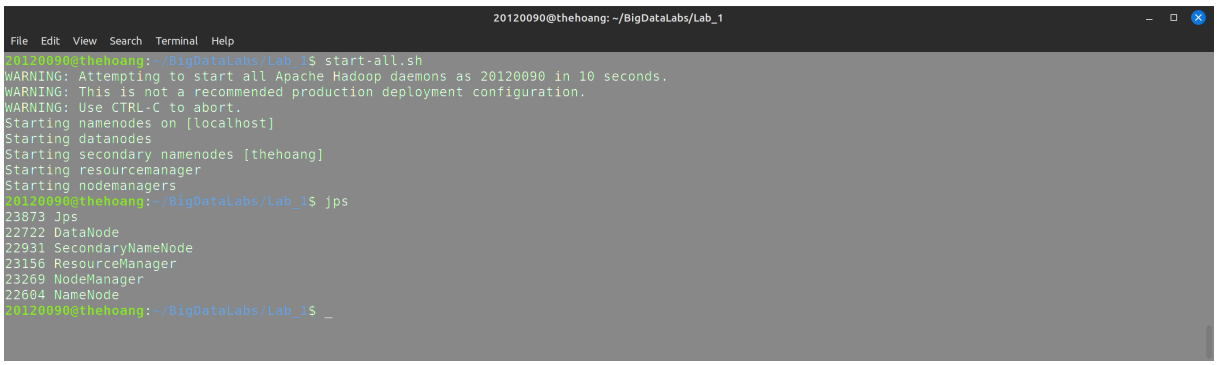



**Figure 3.12:** Step 2: Start the Hadoop on the Pseudo-distributed Mode

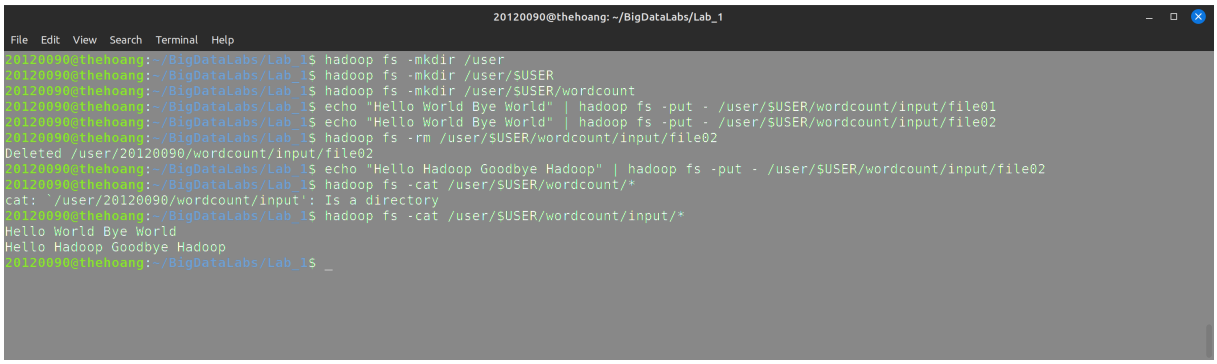



**Figure 3.13:** Step 3: As instructed in [^c], create two input files in the distributed filesystem, in the input folder

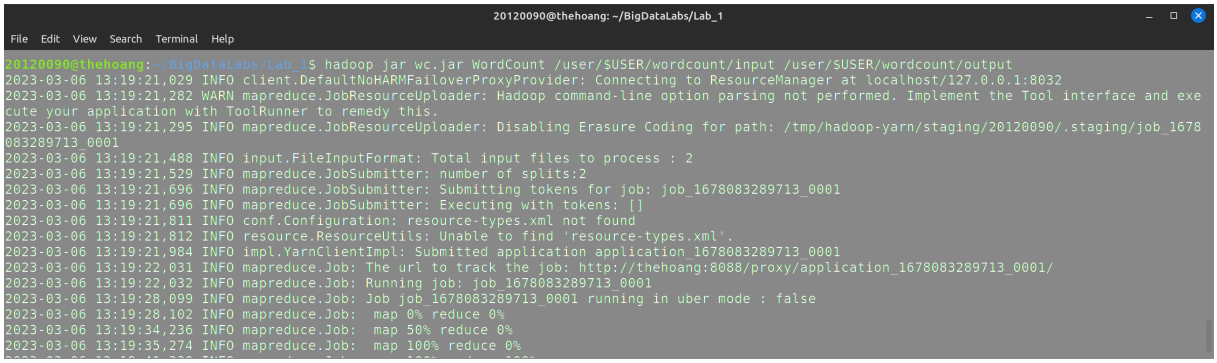
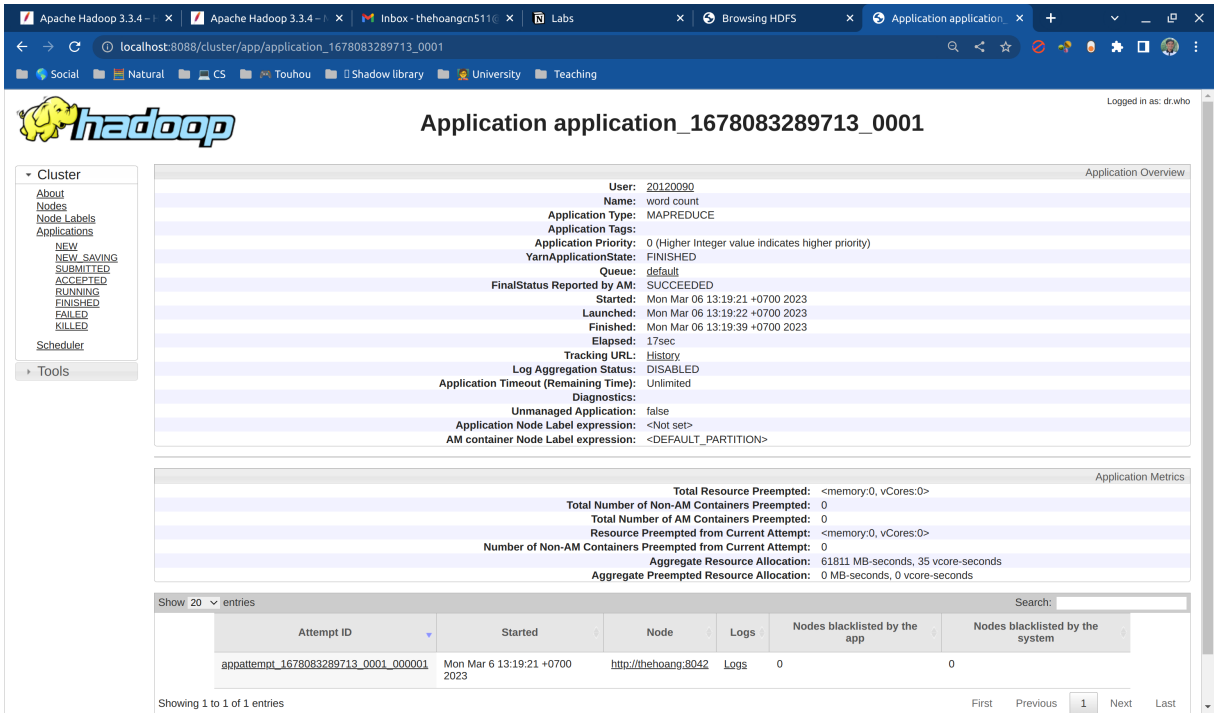
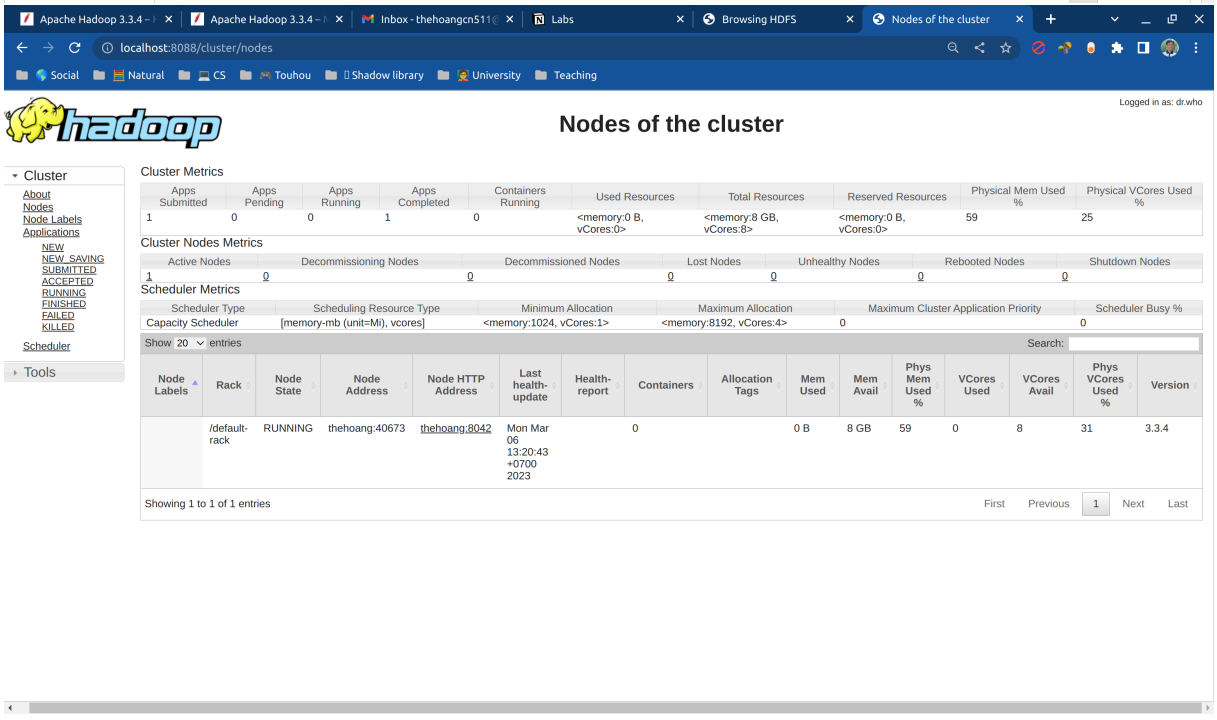
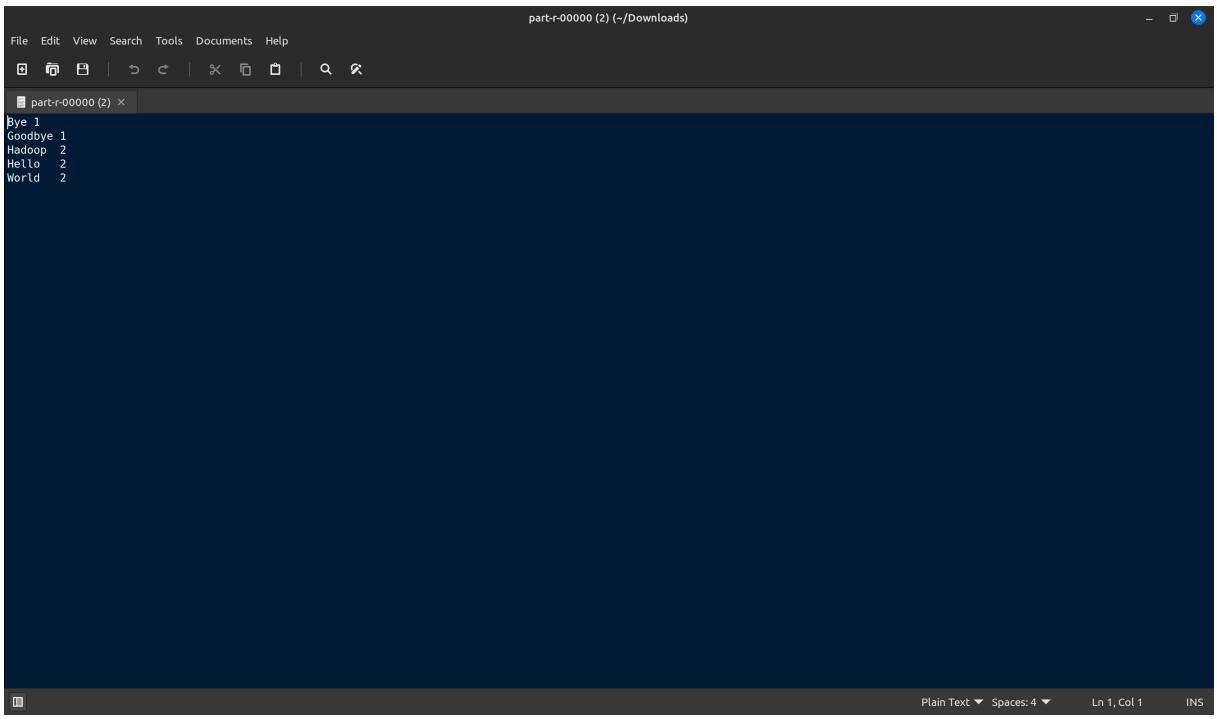



**Figure 3.14:** Step 4: Run the MapReduce program


Apache Hadoop 3.3.4 – I
Apache Hadoop 3.3.4 – I
Inbox - thehoangcn511
Labs
Browsing HDFS
Application application
localhost:8088/cluster/app/application_1678083289713_0001
Social
Natural
CS
Touhou
Shadow library
University
Teaching
Logged in as: dr.who
Application application_1678083289713_0001
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Application Overview
User: 20120090
Name: word count
Application Type: MAPREDUCE
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: FINISHED
Queue: default
FinalStatus Reported by AM: SUCCEEDED
Started: Mon Mar 06 13:19:21 +0700 2023
Launched: Mon Mar 06 13:19:22 +0700 2023
Finished: Mon Mar 06 13:19:39 +0700 2023
Elapsed: 17sec
Tracking URL: History
Log Aggregation Status: DISABLED
Application Timeout (Remaining Time): Unlimited
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>
Application Metrics
Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 61811 MB-seconds, 35 vcore-seconds
Aggregate Preempted Resource Allocation: 0 MB-seconds, 0 vcore-seconds
Show 20 entries
Search:
Attempt ID
Started
Node
Logs
Nodes blacklisted by the app
Nodes blacklisted by the system
appattempt_1678083289713_0001_000001
Mon Mar 6 13:19:21 +0700 2023
http://thehoang:8042
Logs
0
0
Showing 1 to 1 of 1 entries
First
Previous
1
Next
Last


Apache Hadoop 3.3.4 – I
Apache Hadoop 3.3.4 – I
Inbox - thehoangcn511
Labs
Browsing HDFS
Nodes of the cluster
localhost:8088/cluster/nodes
Social
Natural
CS
Touhou
Shadow library
University
Teaching
Logged in as: dr.who
Nodes of the cluster
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Cluster Metrics
Apps Submitted
Apps Pending
Apps Running
Apps Completed
Containers Running
Used Resources
Total Resources
Reserved Resources
Physical Mem Used %
Physical VCores Used %
1
0
0
1
0
<memory:0 B, vCores:0>
<memory:8 GB, vCores:8>
<memory:0 B, vCores:0>
59
25
Cluster Nodes Metrics
Active Nodes
Decommissioning Nodes
Decommissioned Nodes
Lost Nodes
Unhealthy Nodes
Rebooted Nodes
Shutdown Nodes
1
0
0
0
0
0
0
Scheduler Metrics
Scheduler Type
Scheduling Resource Type
Minimum Allocation
Maximum Allocation
Maximum Cluster Application Priority
Scheduler Busy %
Capacity Scheduler
[memory-mb (unit=Mi), vcores]
<memory:1024, vCores:1>
<memory:8192, vCores:4>
0
0
Show 20 entries
Search:
Node Labels
Rack
Node State
Node Address
Node HTTP Address
Last health-update
Health-report
Containers
Allocation Tags
Mem Used
Mem Avail
Phys Mem Used %
VCores Used
VCores Avail
Phys VCores Used %
Version
/default-rack
RUNNING
thehoang:40673
thehoang:8042
Mon Mar 06 13:20:43 +0700 2023
0
0 B
8 GB
59
0
8
31
3.3.4
Showing 1 to 1 of 1 entries
First
Previous
1
Next
Last


part-r-00000 (2) (~/Downloads)
File Edit View Search Tools Documents Help
part-r-00000 (2)
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
Plain Text
Spaces: 4
Ln 1, Col 1
INS

## 3.4 Member 4: 20120165 - Hong Nhat Phuong

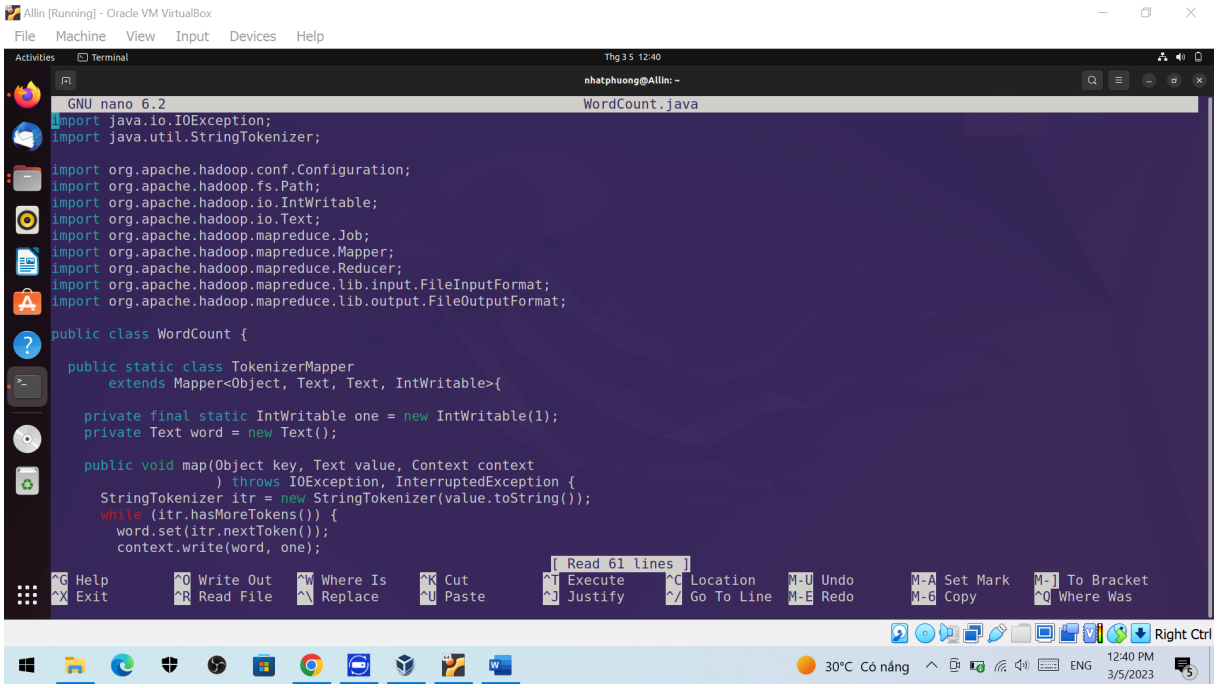



**Figure 3.15:** Create WordCount.java

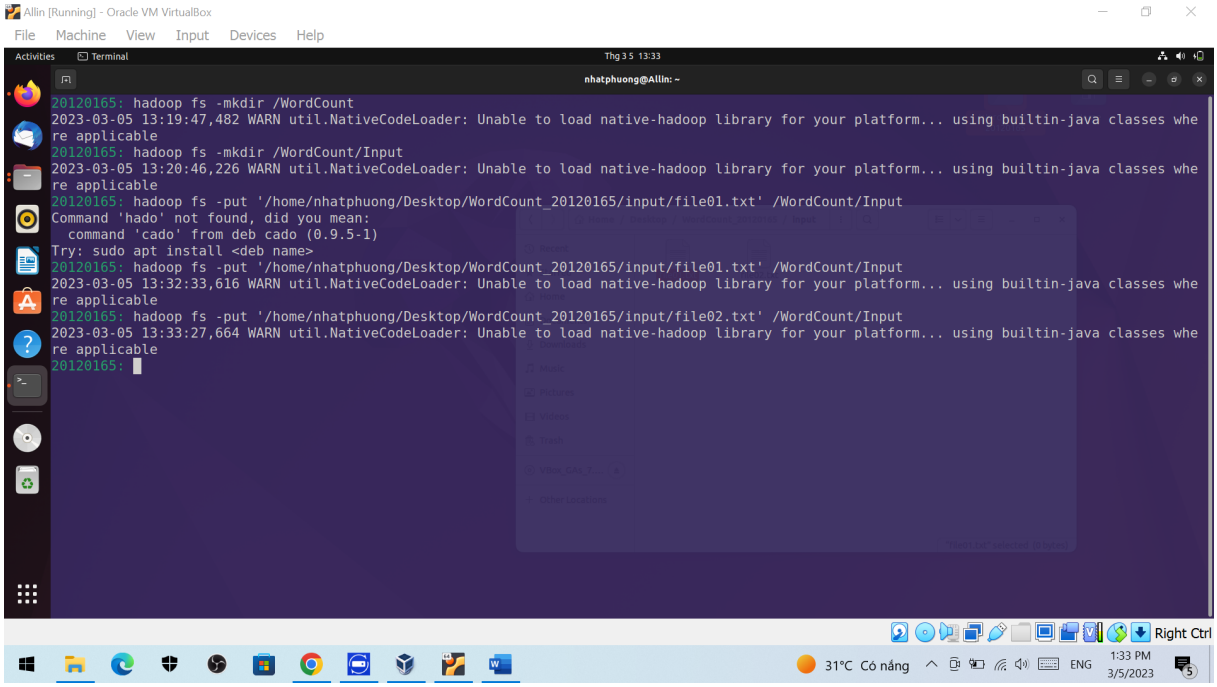



**Figure 3.16:** Create folder input and put file01.txt, file02.txt into input directory

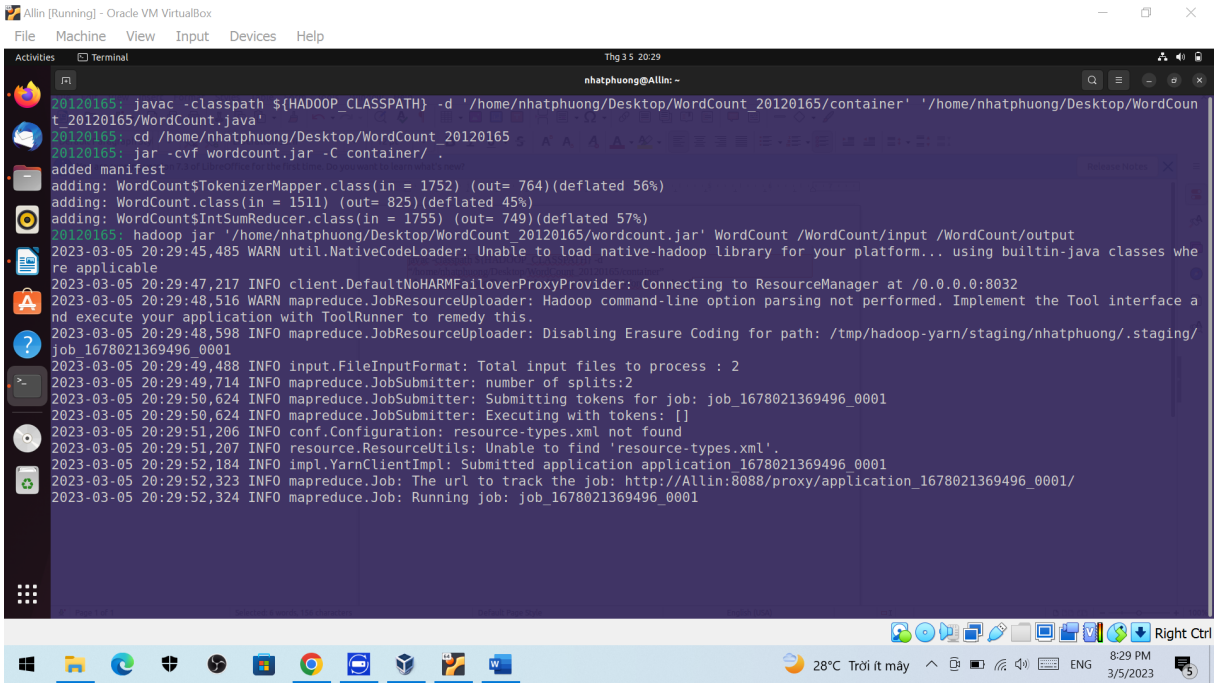



**Figure 3.17:** Compile WordCount.java, create a jar and run WordCount program

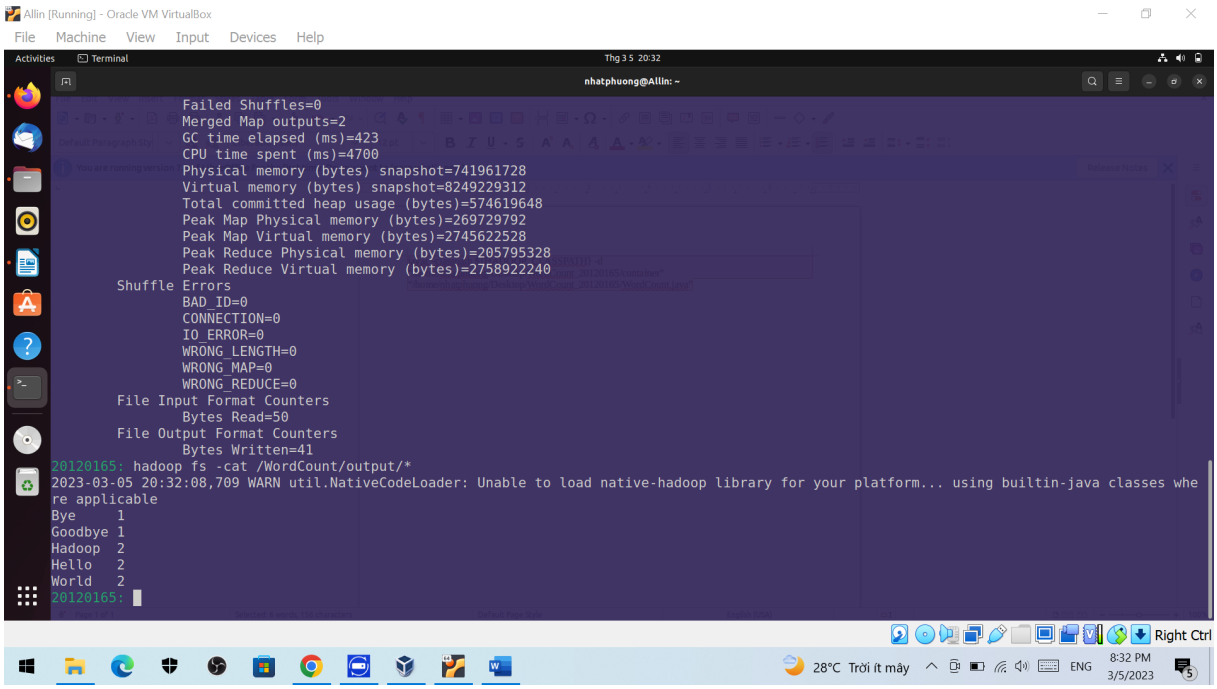



**Figure 3.18:** Show the output using -cat command

# 4 Bonus

## 4.1 Bad Relationship

The **Unhealthy_relationship.java** program includes two phases:

1. The Mappter implementation (`UnhealthyRelationshipMapper` class), via the `map()` method, processes the input files one line at a time, read them as `Text` type object. It then splits the line into two words (seperated by whitespace): `word_1` and `word_2`. It then emits two key-value pair of `< word_1, 1>` and `< word_2, -1>`. The first pair means that: `word_1` has one positive relation with other objects, so it will contribute value '1' to its final result. On the other hand, the second pair means that: `word_2` has one negative relation with other objects, so it will contribute value '-1' to its final result. In short, by considering each word's position in the input files, we can count each positive and negative relation of each word, and then we aggregate these value to know the final relation of each word.

   So the `map()` emits the output has type `<Text, IntWritable>`. This pair of key-value will be grouped by key and all the value of one key will be put into one list.

2. The Reducer implementation (`UnhealthyRelationshipReducer`), via the `reduce()` method, receive the key-value pairs `<word, [list_of_1_or_-1]>` for each distinct word. The `reduce()` method just aggregate all the value in `[list_of_1_or_-1]`. The final value `sum` will be used to determined if that word has negative/positive/equality relation. In specific:

   - `sum` > 0: That word has positive relation
   - `sum` = 0: That word has equality relation
   - `sum` < 0: That word has negative relation

   The `reduce()` method then just emit the final output for each word in the form key-value `<word, type_of_relation>`, with type `<Text, Text>`.

# 5 References

[^b] T. White, “Appendix A. Installing Apache Hadoop,” in Hadoop: The definite guide, 2012, p. 680.

[^c] Apache Hadoop, “MapReduce Tutorial,” Jul. 29, 2022. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.