

# **Hadoop Cluster Implementation**

**By**

**Aysha Binta Sayed**

**ID:2013-1-60-068**

**Supervised By**

**Dr. Md. Shamim Akhter**

**Assistant Professor**

**Department of Computer Science and Engineering**

**East West University**

**A project submitted in partial Fullfillment of the Requirements for the Degree of Bachelors  
of Science in Computer Science and Engineering**

**to the**



**Department of Computer Science and Engineering**

**East West University**

**Dhaka, Bangladesh**

# Abstract

Recently, data driven science is an interdisciplinary field to gather, process, manage, analyze and extract inherit meaning from unstructured data and formulate them as structural information. Later, that information can be employed in many practical applications to solve real life problems. Hadoop is an open source data science tool and is able to process large amount of data sets in distributed manner across cluster of computers (a single server and several worker machines). Hadoop allows running several tasks in parallel and processing huge amount of complex data efficiency with respect to time, performance and cost. Thus, learning Hadoop with its different sub modules is important. This project work covers the implementation of Hadoop cluster with SSH public key authentication for processing large volumes of data, using cheap, easily available personal computer hardware (Intel/AMD based pcs) and freely available open source software (Ubuntu Linux, Apache Hadoop etc). In addition, Mapreduce and Yarn based distributed applications are ported and tested the cluster's workability.

# Declaration

We hereby declare that, this report was done under CSE497 and has not been submitted elsewhere for requirement of any degree of diploma or for any purpose except for publication

---

Aysha Binta Sayed

ID:2013-1-60-068

Department of Computer Science and Engineering

East West University

# Letter of Acceptance

I here declare that this thesis is from the students' own work and best effort of mine, and all other sources of information used have been acknowledged. This thesis has been submitted with or approval.

---

**Dr. Md. Shamim Akhter**

**Supervisor**

Assistant Professor

Department of Computer Science and Engineering

East West University

---

**Dr. Ahmed Wasif Reza**

**Chairperson**

Chairperson and Associate Professor

Department of Computer Science and Engineering

East West University

# Acknowledgement

First of all, I would like to thank Almighty Allah for giving me the strength and patience.

Then I would like to thank and pay my sincere gratitude to my mentor and thesis coordinator Dr. Md. Shamim Akhter for supporting me throughout my endeavor with his extensive knowledge of the related grounds, his tremendous efforts and robustness that compelled me succeeding endure the ordeals and his extraordinary visions that enthralled me to refine my work to the finest point.

I also thank my beloved family and friends, for supporting me throughout the time of our experience that allowed me to able to come up with what I really have today.

<b>Abstract</b>	<b>ii</b>
<b>Declaration</b>	<b>iii</b>
<b>Letter of Acceptance</b>	<b>iv</b>
<b>Acknowledgement</b>	<b>v</b>

## **Chapter 1**

<b>Introduction</b>	<b>1-2</b>
1.1 Overview	1
1.2 Motivation	1
1.3 Objective	2
1.4 Scope and limitation	2
1.5 Outline	2

## **Chapter 2**

<b>Computer Cluster</b>	<b>3-4</b>
-------------------------	------------

## **Chapter 3**

<b>Hadoop Cluster</b>	<b>5-12</b>
3.1 Hadoop	5
3.2 Hadoop Architecture	6
3.3 Hadoop Distributed File System	7-8
3.4 YARN	9
3.5 Hadoop Mapreduce	10-11

3.6 Hadoop Cluster	12
<b>Chapter 4</b>	
<b>Hadoop Multi-Node Cluster Configuration</b>	13-37
<b>Chapter 5</b>	
<b>Implementing MapReduce Application</b>	38-45
<b>Chapter 6</b>	
<b>Result &amp; Discussion</b>	46-50
<b>Chapter 7</b>	
<b>Conclusion &amp; Future Work</b>	51
7.1 Conclusion	51
7.2 Future Work	51
<b>References</b>	52

# Chapter 1

## Introduction

### 1.1 Overview

Today's world is called a world of large data, ranging from some petabytes to zetabytes. Many applications are dealing with large amount of data and the results should be obtained within a particular time limit. So, Apache has come with an appropriate framework to handle large data. The framework is Hadoop. It enables a number of applications to work with many computational independent machines and large datasets. Many leading companies like Yahoo, Facebook are using Hadoop, nowadays. Using a simple programming model, Apache has developed a framework for distributed processing of large datasets across the many computers or clusters of computers. The framework is Apache Hadoop software library. A Hadoop cluster can be scaled up to thousands of machines and offer separate computation and storage in each single machine. Hadoop itself has the capability to detect and handle failure jobs.

Map/Reduce (MapReduce) is a computational paradigm implemented on Hadoop. MapReduce is a job manager and divides jobs/applications into smaller fragments to execute them in computational nodes in parallel. Hadoop supports HDFS (Hadoop Distributed File System) to store data on computational nodes. Both MapReduce and HDFS are designed to support fault tolerance system and able to handle failure nodes automatically. Hadoop cluster is able to boost the processing speed of data driven application. It is highly scalable as additional computational nodes can be added to increase the throughput. Hadoop has the ability to resist from task failure because each piece of processing data is copied onto other computational nodes and they confirms not to loss data even the corresponding node fails. Hadoop cluster can be implemented with only a single-node or using multi-nodes. Single-node Hadoop is the basic form. In multi-noded Hadoop, there is one master node and several computational nodes. Master node consists NameNode , ResourceManager, JobhistoryServer and each computing node consist DataNode and NodeManager. This project work covers the implementation of Hadoop cluster with SSH public key authentication for processing large volumes of data, using cheap, easily available personal computer hardware (Intel/AMD based PCs) and freely available open source software (Ubuntu Linux, Apache Hadoop etc). In addition, MapReduce and Yarn based distributed applications are ported and tested the cluster's workability.

### 1.2 Motivation

Future data scientist needs to learn the tools to analyze, process and visualize data. Hadoop provides a compact tool to fulfill the above requirements and also use distributed processing to implement the jobs. Thus, learning Hadoop covers two different data science domains knowledge including:



- Distributed processing and distributed environment.
- Data visualization and analytics.

### **1.3 Objective**

The main objective of this project is to implement a complete Hadoop cluster using SSH key authentication. The sub objectives are as follows: -

- Design, implement and test the cluster computer connection with SSH key authentication.
- Implement and test the Hadoop tool on cluster master and slaves.
- Import a Yarn and MapReduce based applications and monitor their runtime environment through web based system.

### **1.4 Scope and limitation**

#### **Scope :**

The use of Hadoop cluster might result in count word occurrences, search word, compute mathematical function of huge amount of data. We can put these huge data in Hadoop distributed file system instead of local computer system. As the work is distributed among parallel nodes, it takes less time to process large amount of data. MapReduce based applications can run on this Hadoop cluster.

#### **Limitation:**

Our implementation of Hadoop cluster can work only on broadband link, not able to work in wireless system. Also the computer RAM is 4 GB which is not sufficient for Hadoop cluster.

### **1.5 Outline**

- Chapter 2, which contains a brief overview on computer cluster.
- Chapter 3, which contains a brief overview on Hadoop , Hadoop Architecture, Hadoop Distributed File System, Hadoop MapReduce, Hadoop Cluster
- The implementation of Hadoop cluster is discussed in Chapter 4 and will give a clear view about cluster behavior. In this chapter we are using screen shot of every command output after execution to make things easier.
- Chapter 5 covers implementing MapReduce program on Hadoop Cluster.
- Chapter 6 covers result and discussion of fulfilling the goal of our objective.
- Chapter 7 is all about conclusion and future work.

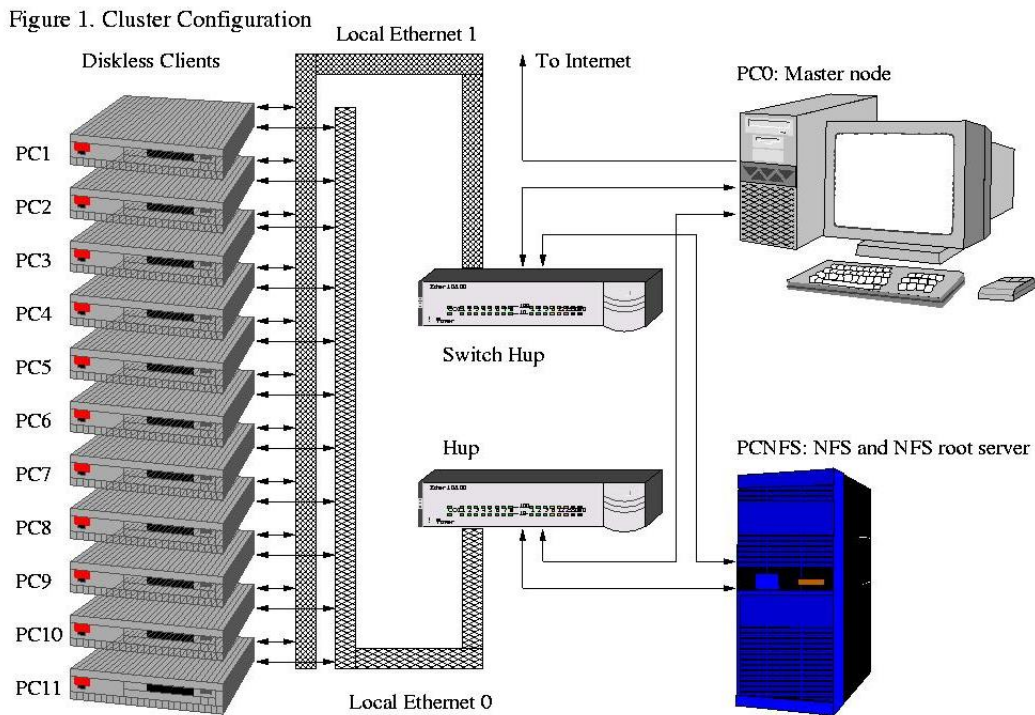
# Chapter 2

## Computer Cluster

A computer cluster is a single logical unit consisting of multiple computers that are linked through a LAN. The networked computers essentially act as a single, much more powerful machine. A computer cluster provides much faster processing speed, larger storage capacity, better data integrity, superior reliability and wider availability of resources.

Computer clusters are, however, much more costly to implement and maintain. These results in much higher running overhead compared to a single computer.

Many organizations use computer clusters to maximize processing time, increase database storage and implement faster data storing & retrieving techniques.



There are many types of computer clusters, including:

- Load-balancing clusters
- High availability (HA) clusters
- High performance (HP) clusters

The major advantages of using computer clusters are clear when an organization requires large scale processing. When used this way, computer clusters offer:

- **Cost efficiency:** The cluster technique is cost effective for the amount of power and processing speed being produced. It is more efficient and much cheaper compared to other solutions like setting up mainframe computers.
- **Processing speed:** Multiple high-speed computers work together to provide unified processing, and thus faster processing overall.
- **Improved network infrastructure:** Different LAN topologies are implemented to form a computer cluster. These networks create a highly efficient and effective infrastructure that prevents bottlenecks.
- **Flexibility:** Unlike mainframe computers, computer clusters can be upgraded to enhance the existing specifications or add extra components to the system.
- **High availability of resources:** If any single component fails in a computer cluster, the other machines continue to provide uninterrupted processing. This redundancy is lacking in mainframe systems.

# Chapter 3

## Hadoop Cluster

### 3.1 Hadoop

Hadoop is an open source, Java-based programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Its distributed file system facilitates rapid data transfer rates among nodes and allows the system to continue operating in case of a node failure. This approach lowers the risk of catastrophic system failure and unexpected data loss, even if a significant number of nodes become inoperative. Consequently, Hadoop quickly emerged as a foundation for big data processing tasks, such as scientific analytics, business and sales planning, and processing enormous volumes of sensor data, including from internet of things sensors.

Hadoop was created by computer scientists Doug Cutting and Mike Cafarella in 2006 to support distribution for the Nutch search engine. It was inspired by Google's MapReduce, a software framework in which an application is broken down into numerous small parts. Any of these parts, which are also called fragments or blocks, can be run on any node in the cluster. After years of development within the open source community, Hadoop 1.0 became publically available in November 2012 as part of the Apache project sponsored by the Apache Software Foundation.

Hadoop framework includes following four models:

**Hadoop Common:** Hadoop common is Java libraries and utilities, which are required by other Hadoop modules. They provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

**Hadoop YARN:** Hadoop YARN is a framework which is used for job scheduling and cluster resource management.

**Hadoop Distributed File System (HDFS):** A distributed file system that provides high throughput access to application data.

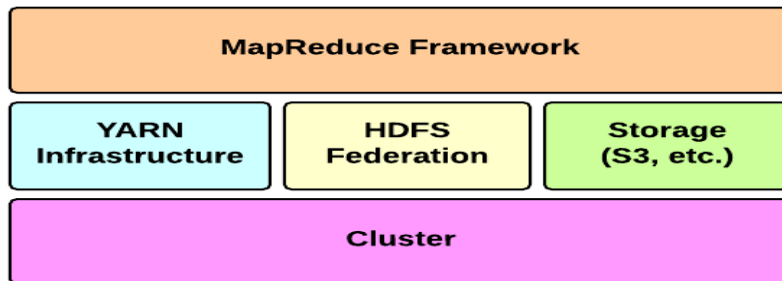
**HadoopMapReduce:** HadoopMapReduce is a YARN-based system for parallel processing of large data sets.

## 3.2 Hadoop Architecture

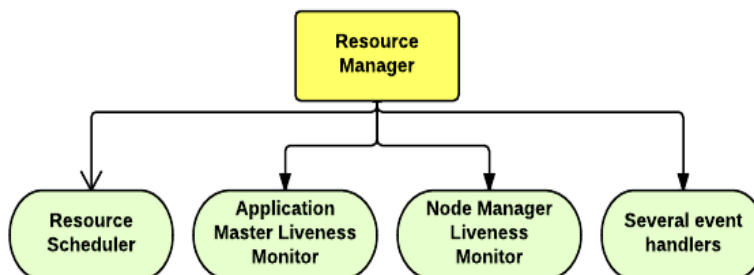
Hadoop offers a scalable, flexible and reliable distributed computing big data framework for a cluster of systems with storage capacity and local computing power by leveraging commodity hardware. Hadoop follows a Master Slave architecture for the transformation and analysis of large datasets using HadoopMapReduce paradigm. The 3 important Hadoop components that play a vital role in the Hadoop architecture are -

- i. Hadoop Distributed File System (HDFS) – Patterned after the UNIX file system
- ii. HadoopMapReduce
- iii. Yet Another Resource Negotiator (YARN)

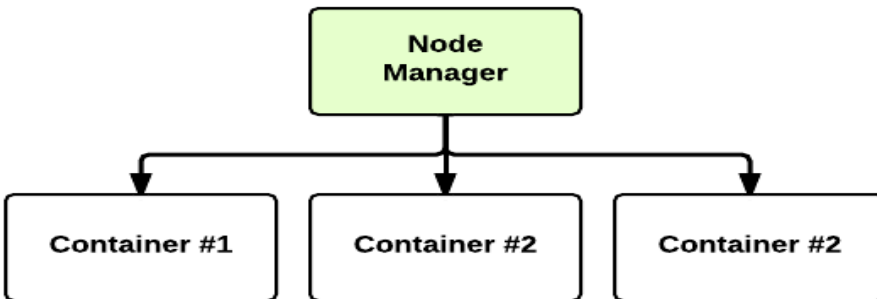
There are mainly five building blocks inside this runtime environment (from bottom to top):



- **Cluster** is the set of host machines (**nodes**). Nodes may be partitioned in **racks**. This is the hardware part of the infrastructure.
- **YARN Infrastructure** (Yet Another Resource Negotiator) is the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions. Two important elements are:
  - **Resource Manager** (one per cluster) is the master. It knows where the slaves are located (Rack Awareness) and how many resources they have. It runs several services; the most important is the **Resource Scheduler** which decides how to assign the resources.



- **Node Manager** (many per cluster) is the slave of the infrastructure. When it starts, it announces itself to the Resource Manager. Periodically, it sends a heartbeat to the Resource Manager. Each Node Manager offers some resources to the cluster. Its resource capacity is the amount of memory and the number of cores. At run-time, the Resource Scheduler will decide how to use this capacity: a **Container** is a fraction of the NM capacity and it is used by the client for running a program



- **HDFS Federation** is the framework responsible for providing permanent, reliable and distributed storage. This is typically used for storing inputs and output (but not intermediate ones).
- Other alternative storage solutions. For instance, Amazon uses the Simple Storage Service (S3).
- The **MapReduce Framework** is the software layer implementing the MapReduce paradigm.

### 3.3 Hadoop Distributed File System

A file on HDFS is split into multiple blocks and each is replicated within the Hadoop cluster. A block on HDFS is a blob of data within the underlying file system with a default size of 64MB. The size of a block can be extended up to 256 MB based on the requirements.

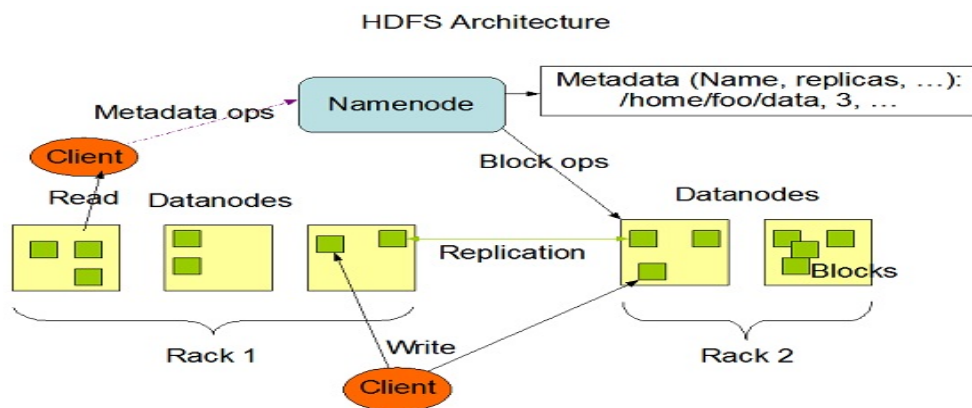


Image Credit :Apache.org

Hadoop Distributed File System (HDFS) stores the application data and file system metadata separately on dedicated servers. NameNode and DataNode are the two critical components of the Hadoop HDFS architecture. Application data is stored on servers referred to as DataNodes and file system metadata is stored on servers referred to as NameNode. HDFS replicates the file content on multiple DataNodes based on the replication factor to ensure reliability of data. The NameNode and DataNode communicate with each other using TCP based protocols. For the Hadoop architecture to be performance efficient, HDFS must satisfy certain pre-requisites –

- All the hard drives should have a high throughput.
- Good network speed to manage intermediate data transfer and block replications.

- **NameNode**

NameNode is the center of HDFS file system. NameNode does not store data itself but it tracks where during the cluster the data file is held. When client wish to locate a file or want to add/copy/move/delete a file, they contact to the NameNode. Then the NameNode return the list of relevant DataNodes servers where the data are located and like this NameNode gives the successful responds of client's request. The HDFS file system is highly rely on NameNode. If the NameNode goes down then the file system also goes offline. There is an secondaryNameNode which also can be hosted on a separate machine. Secondary NameNode only creates checkpoints of the Namespace by merging the edits file into the fsimage file. The NameNode stores file system metadata into two different files. One is the fsimage and the other is the edit log.

- The fsimage file stores a complete snapshot of the file system's metadata at a specific moment in time.
- The edit log stores the incremental changes such as renaming or appending a few bytes to a file for durability.

When the NameNode starts, fsimage file is loaded and then the contents of the edits file are applied to recover the latest state of the file system. The only problem with this is that over the time the edits file grows and consumes all the disk space resulting in slowing down the restart process. If the Hadoop cluster has not been restarted for months together then there will be a huge downtime as the size of the edits file will be increase.

## **DataNode**

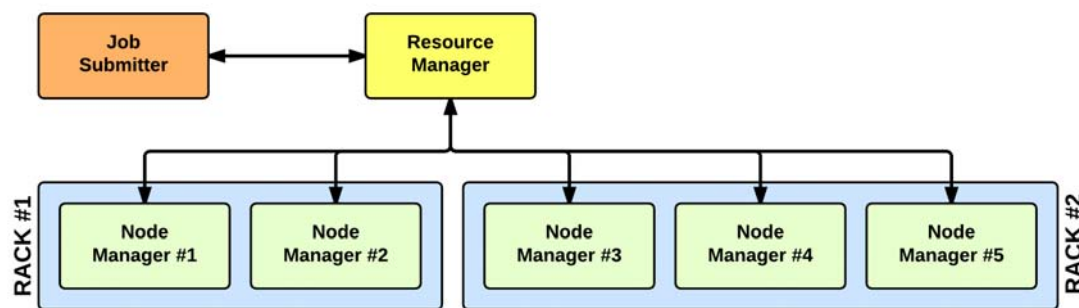
In Hadoop Distribute File System, DataNode store the actual data. When Hadoop cluster is starts, DataNode connects to the NameNode. Then DataNode gives responds to the request of NameNode fro filesystem operations. Once NameNode provide the location of the data to the client, client application can talk directly to a DataNode.

### 3.4 Yarn

YARN (Yet Another Resource Negotiator) is a cluster management technology. YARN is one of the key features in the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. Originally described by Apache as a redesigned resource manager, YARN is now characterized as a large-scale, distributed operating system for big data applications.

The YARN infrastructure and the HDFS federation are completely decoupled and independent: the first one provides resources for running an application while the second one provides storage. The MapReduce framework is only one of many possible framework which runs on top of YARN (although currently is the only one implemented).

#### YARN: Application Startup



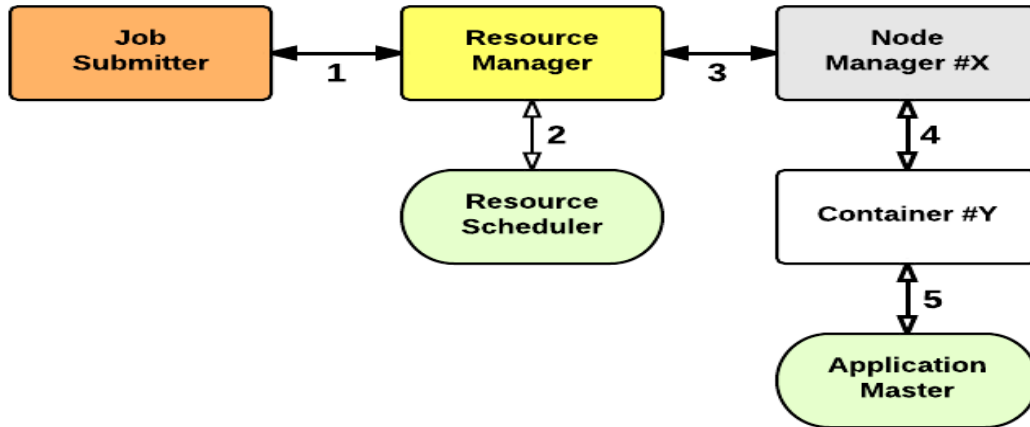
In YARN, there are at least three actors:

- **Job Submitter** (the client)
- **Resource Manager** (the master)
- **Node Manager** (the slave)

The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the **Application Master**





The Application Master is responsible for the execution of a single application. It asks for containers to the Resource Scheduler (Resource Manager) and executes specific programs (e.g., the main of a Java class) on the obtained containers. The Application Master knows the application logic and thus it is framework-specific. The MapReduce framework provides its own implementation of an Application Master.

The Resource Manager is a single point of failure in YARN. Using Application Masters, YARN is spreading over the cluster the metadata related to running applications. This reduces the load of the Resource Manager and makes it fast recoverable.

### 3.5 Hadoop MapReduce

The execution of a MapReduce job begins when the client submits the job configuration to the Job Tracker that specifies the map, combine and reduce functions along with the location for input and output data. On receiving the job configuration, the job tracker identifies the number of splits based on the input path and select Task Trackers based on their network vicinity to the data sources. Job Tracker sends a request to the selected Task Trackers.

The processing of the Map phase begins where the Task Tracker extracts the input data from the splits. Map function is invoked for each record parsed by the “InputFormat” which produces key-value pairs in the memory buffer. The memory buffer is then sorted to different reducer nodes by invoking the combine function. On completion of the map task, Task Tracker notifies the Job Tracker. When all Task Trackers are done, the Job Tracker notifies the selected Task Trackers to begin the reduce phase. Task Tracker reads the region files and sorts the key-value pairs for each key. The reduce function is then invoked which collects the aggregated values into the output file.

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

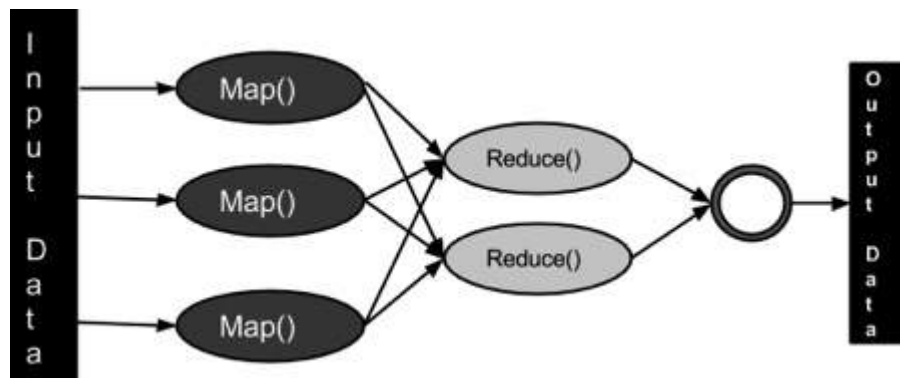
MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken

down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Algorithm:

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



### **3.6 Hadoop Cluster**

A Hadoop cluster is a special type of computational cluster which is designed specially for storing and analyzing large amounts of unstructured data in a distributed computing environment. Hadoop cluster can be implemented through Hadoop open source distributed software ( which is free) on low cost commodity computers. Typically one machine is create as a Master Node and other machines are creates as Slave Nodes. In a small cluster there should be at least one muster and one slave. Master Node is consist of NameNode, SecondaryNameNode, ResourceManager, JobHistoryServer. These jobs run as a java process in Hadoop cluster. On the other hand, Slave Node consists of DataNode and NodeManager. These jobs also run as a java process in Hadoop cluster. There are two types of Hadoop cluster. One is single node Hadoop cluster and another is multi-node Hadoop cluster. In chapter 4, we will implement multi-node Hadoop cluster.

# Chapter 4

## Hadoop Multi-Node Cluster Configuration

This chapter explains the set up of the Hadoop Multi -Node cluster on a distributed environment.

As the whole cluster cannot be demonstrated we are explaining the Hadoop cluster environment using two systems ( one master and one slave ); given below are their IP addresses.

- Hadoop Master: 192.168.2.130 (hadoopmaster )
- Hadoop Slave: 192.168.2.131 (hadoopslave1)

Follow the steps given below to have Hadoop Multi-Node cluster set up.

- **Hadoop Version**  
Hadoop-2.7.3
- **Operating System Used**  
Linux ( Ubuntu , Version 14.04 )

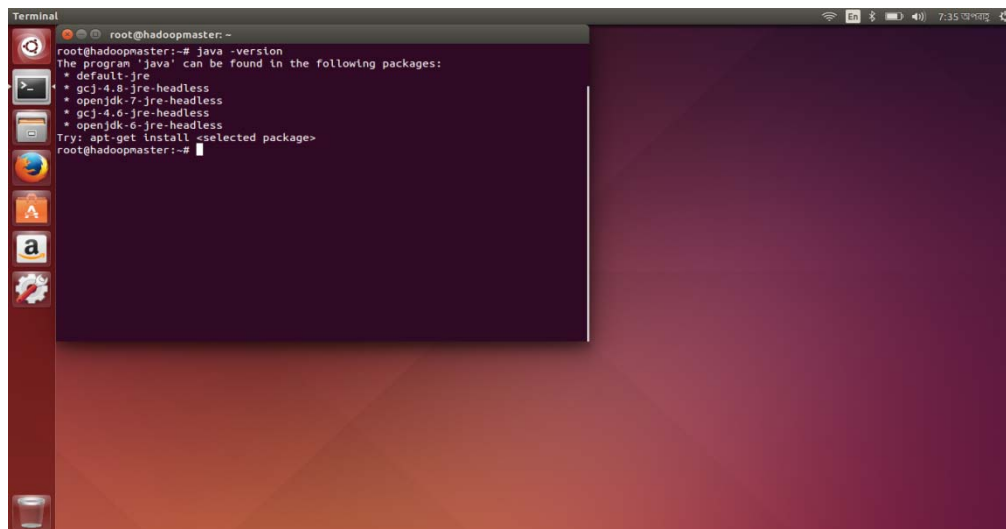
### Creating The hadoopmaster

- **Installing JAVA**

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in your system using “ java -version” . The syntax of java version command is given below.

```
root@hadoopmaster:~# java -version
```

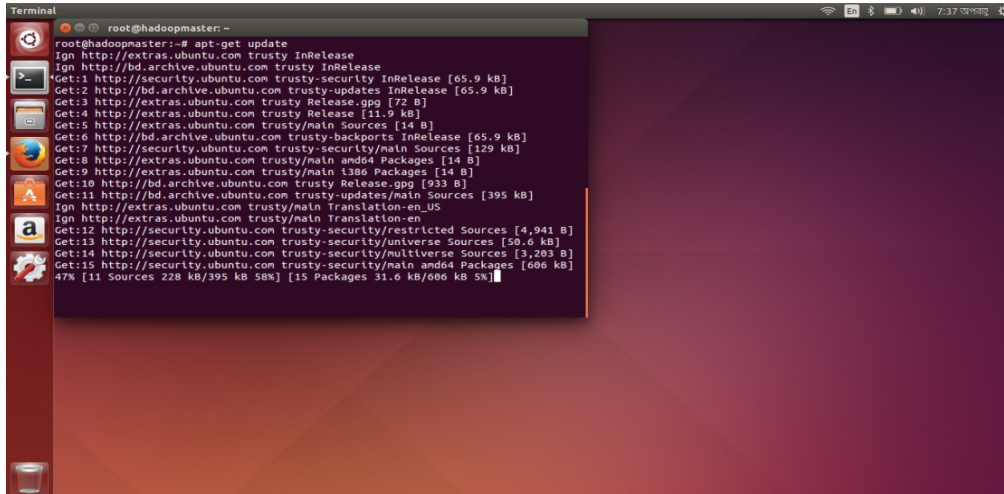
If there is no java installs in your system then you shall get the following option to install java .



```
Terminal
root@hadoopmaster:~# java -version
The program 'java' can be found in the following packages:
 * default-jre
 * gcj-4.8-jre-headless
 * openjdk-7-jre-headless
 * gcj-4.6-jre-headless
 * openjdk-6-jre-headless
Try: apt-get install <selected package>
root@hadoopmaster:~#
```

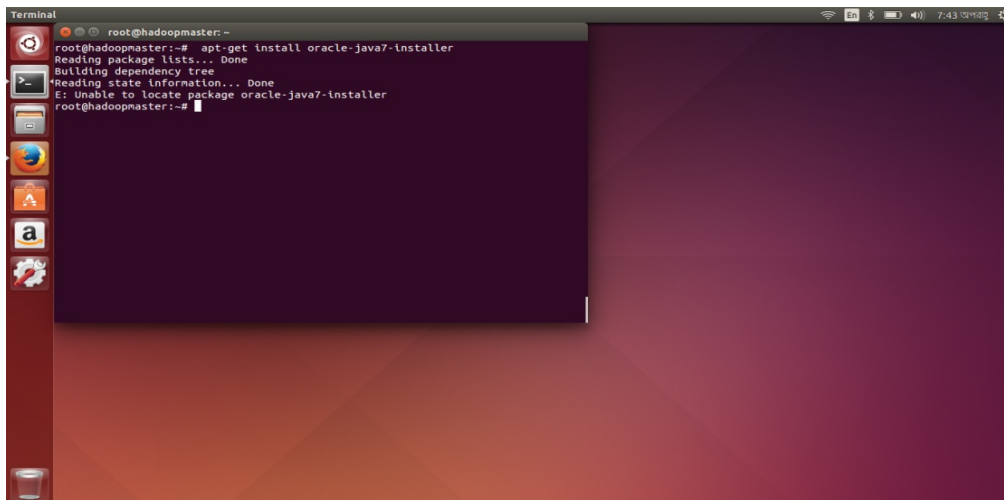
Now let us update the system & install Oracle java 1.7. During the installation, if it prompts for a configuration, press Y.

```
root@hadoopmaster:~# apt-get update
```



```
Terminal
root@hadoopmaster:~# apt-get update
Ign http://extras.ubuntu.com trusty InRelease
Ign http://bd.archive.ubuntu.com trusty InRelease
Get:1 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
Get:2 http://bd.archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:3 http://extras.ubuntu.com trusty Release.gpg [72 B]
Get:4 http://extras.ubuntu.com trusty Release [11.9 kB]
Get:5 http://extras.ubuntu.com trusty/main Sources [14 B]
Get:6 http://bd.archive.ubuntu.com trusty-backports InRelease [65.9 kB]
Get:7 http://security.ubuntu.com trusty-security/main Sources [129 kB]
Get:8 http://extras.ubuntu.com trusty/main amd64 Packages [14 B]
Get:9 http://extras.ubuntu.com trusty/main i386 Packages [14 B]
Get:10 http://bd.archive.ubuntu.com trusty Release.gpg [933 B]
Get:11 http://bd.archive.ubuntu.com trusty-updates/main Sources [395 kB]
Ign http://extras.ubuntu.com trusty/main Translation-en_US
Ign http://extras.ubuntu.com trusty/main Translation-en
Get:12 http://security.ubuntu.com trusty-security/restricted Sources [4,941 B]
Get:13 http://security.ubuntu.com trusty-security/universe Sources [50.6 kB]
Get:14 http://security.ubuntu.com trusty-security/multiverse Sources [3,203 B]
Get:15 http://security.ubuntu.com trusty-security/main amd64 Packages [606 kB]
47% [11 Sources 228 kB/395 kB 58%] [15 Packages 31.6 kB/606 kB 5%]
```

```
root@hadoopmaster:~# apt-get install oracle-java7-installer
```



```
Terminal
root@hadoopmaster:~# apt-get install oracle-java7-installer
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package oracle-java7-installer
root@hadoopmaster:~#
```

Here occurs an error during the installation. So for installing the java we need to give the following command

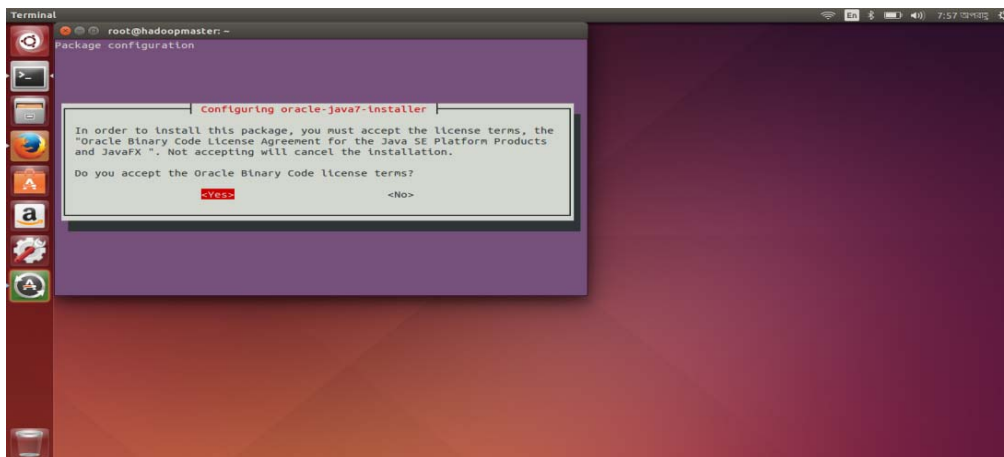
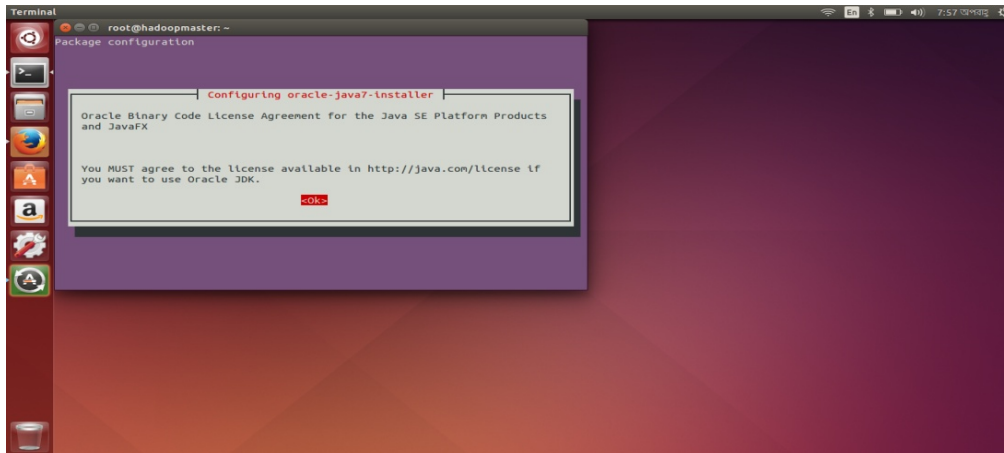
```
root@hadoopmaster:~# add-apt-repository ppa:webupd8team/java
```

Now again update the system by giving the following command

```
root@hadoopmaster:~# apt-get update
```

Now again give the command

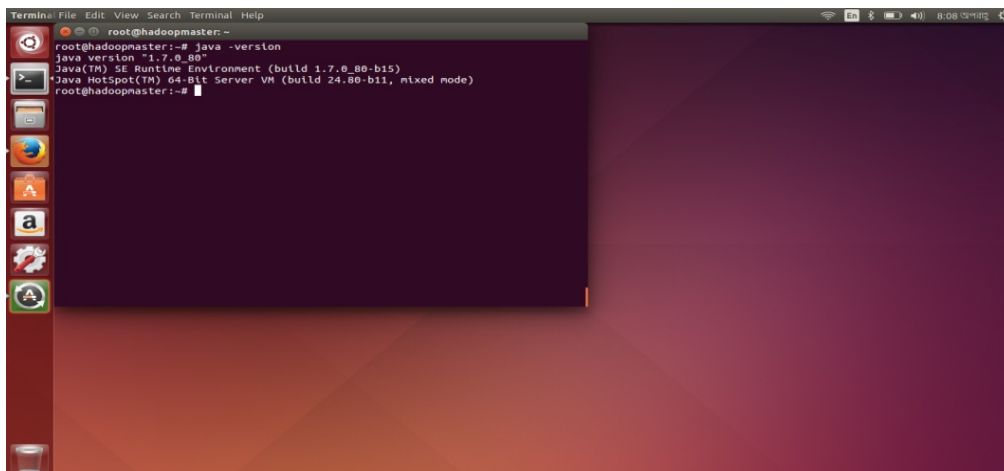
```
root@hadoopmaster:~# apt-get install oracle-java7-installer
```



Now check whether the installation is complete or not by giving the following command

```
root@hadoopmaster:~# java -version
```

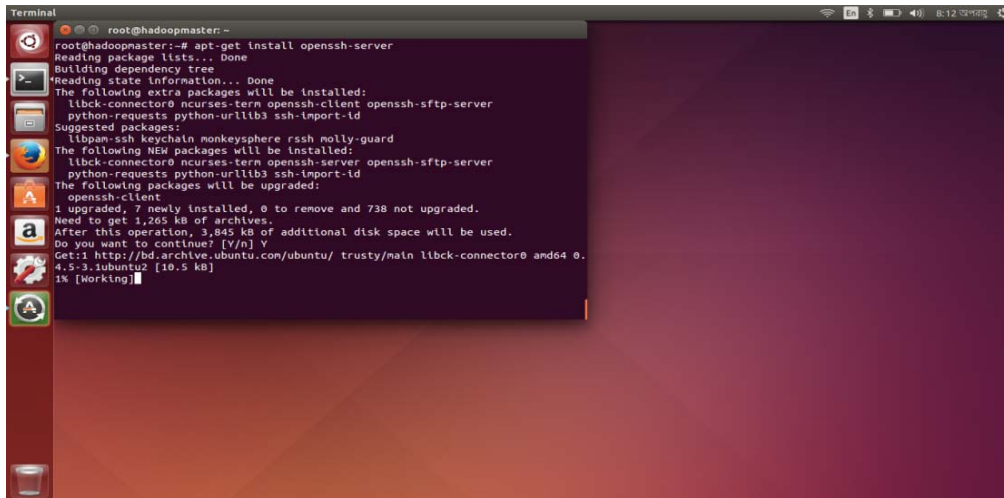
Output looks like below



- **Installing ssh**

Hadoop requires ssh access to manage its nodes for remotely controlling or transferring files between nodes. For installing ssh give the following command.

```
root@hadoopmaster:~# apt-get install openssh-server
```



Write yes and wait until the installation is complete.

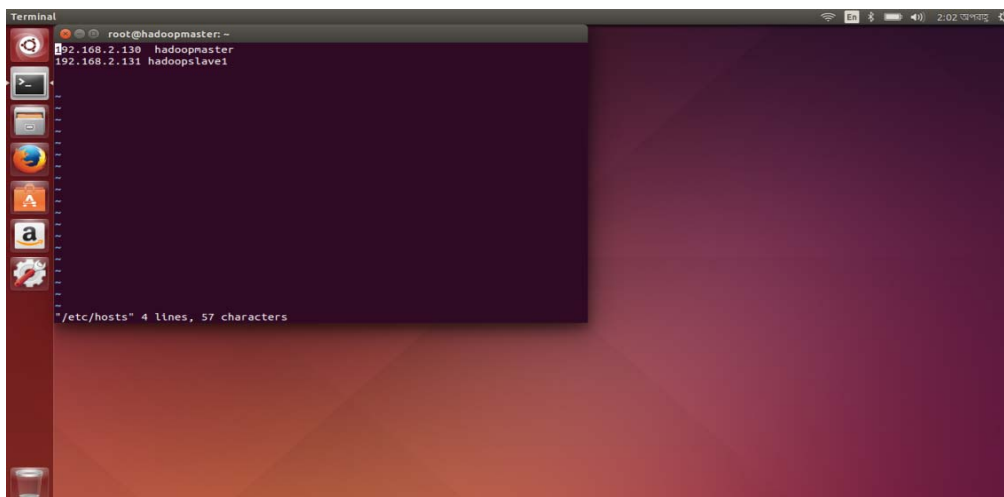
- **Mapping the nodes :**

```
root@hadoopmaster:~# vi /etc/hosts
```

Enter the following lines in the /etc/hosts files.

```
192.168.2.130 hadoopmaster
```

```
192.168.2.131 doopslave1
```



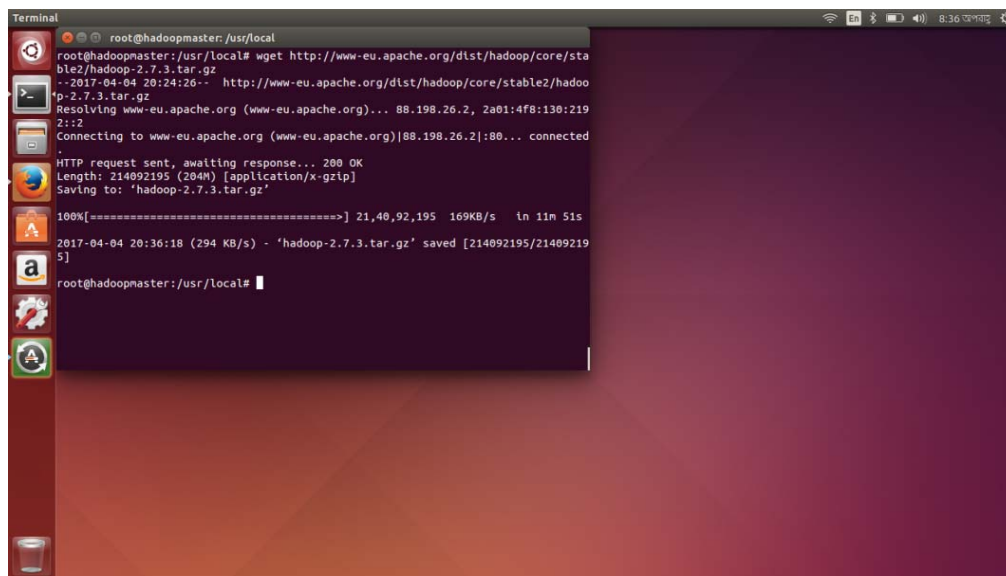
- **Download and Install Hadoop distribution**

First of all check Latest Stable Hadoop Release Available at:

<http://hadoop.apache.org/releases.html> . In this configuration we are downloading and installing Hadoop-2.7.3 . We will install it under /usr/local/ directory. After that we will also create few additional directories like namenode for hadoop to store all namenode information and namesecondary to store the checkpoints images.

```
root@hadoopmaster:~# cd /usr/local
```

```
root@hadoopmaster:/usr/local/# wget http://www-eu.apache.org/dist/hadoop/core/stable2/hadoop-2.7.3.tar.gz
```



```
root@hadoopmaster:/usr/local/# tar -xzvf hadoop-2.7.3.tar.gz >> /dev/null
```

```
root@hadoopmaster:/usr/local/# mv hadoop-2.7.3 /usr/local/hadoop
```

```
root@hadoopmaster:/usr/local/# mkdir -p /usr/local/hadoop_work/hdfs/namenode
```

```
root@hadoopmaster:/usr/local/# mkdir -p /usr/local/hadoop_work/hdfs/namesecondary
```

- **Setup Environment Variables**

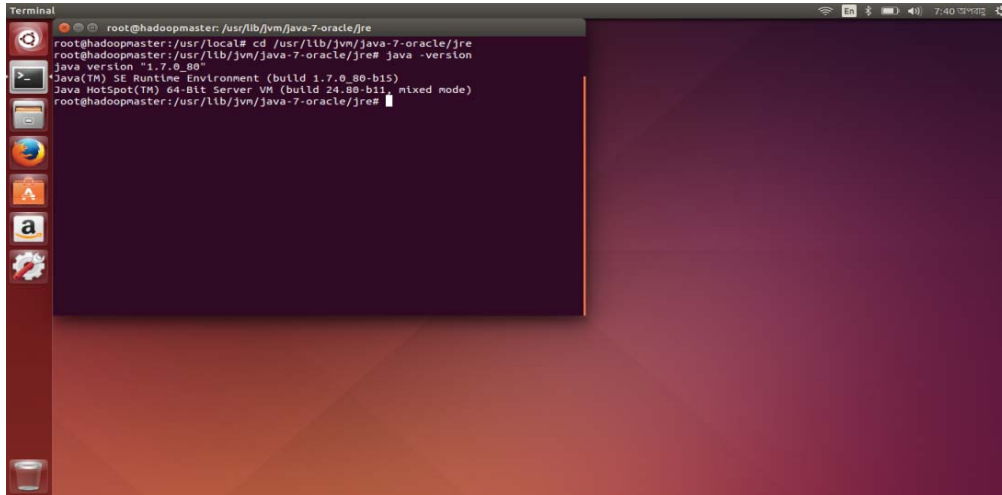
We shall setup some environment variables in .bashrc so that every time we restart our machines, it knows where to find Java or Hadoop installation location inside the machine. To do this first, we need to find out where JAVA has been installed. According to this Configuration java has been installed within one of the subdirectories under /usr/lib/jvm/ directory. So we need to go to the location and confirm that java is there.



For that write the following command in the terminal .

```
root@hadoopmaster:/usr/local/# cd /usr/lib/jvm/java-7-oracle/jre
```

```
root@hadoopslave1:/usr/lib/jvm/java-7-oracle/jre# java -version
```



```
Terminal
root@hadoopmaster: /usr/lib/jvm/java-7-oracle/jre
root@hadoopmaster:/usr/local# cd /usr/lib/jvm/java-7-oracle/jre
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre# java -version
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre#
```

The above output shows that Java is there inside the default-java directory

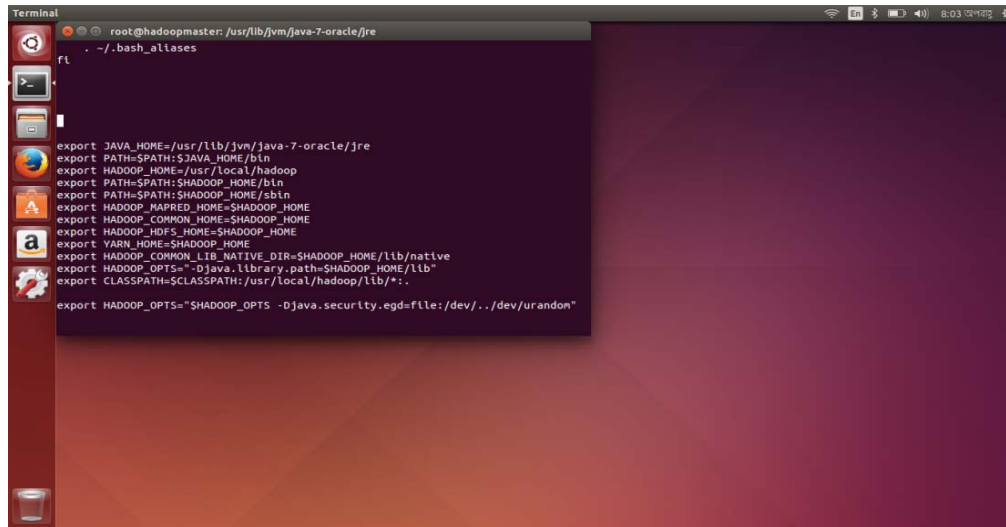
Now open .bashrc and put these lines at the end of your .bashrc file

```
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre# vi ~/.bashrc
```

And put these lines at the end of your .bashrc file

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_CLASSPATH=/usr/lib/jvm/java-7-oracle/lib/tools.jar
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/lib/*:.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.security.egd=file:/dev/./dev/urandom"
```

After setup the environment variable in the .bashrc :

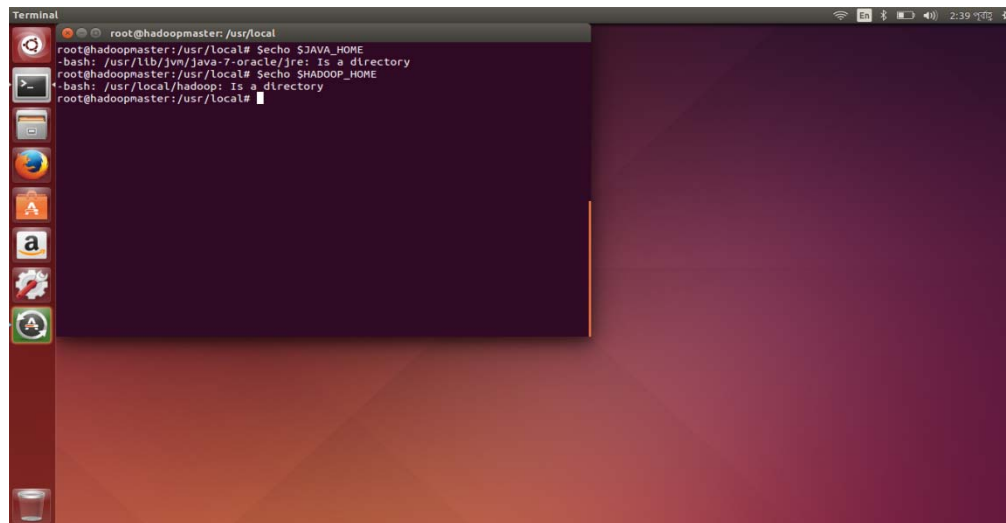


```
Terminal
root@hadoopmaster: /usr/lib/jvm/java-7-oracle/jre
~/.bash_aliases
ft
export JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/lib/*:.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.security.egd=file:/dev/./dev/urandom"
```

Verify .bashrc by writing the following commands

```
root@hadoopmaster:/usr/local/# $echo $JAVA_HOME
```

```
root@hadoopmaster:/usr/local/# $echo $SHADOOP_HOME
```



```
Terminal
root@hadoopmaster: /usr/local
root@hadoopmaster: /usr/local# $echo $JAVA_HOME
-bash: /usr/lib/jvm/java-7-oracle/jre: Is a directory
root@hadoopmaster: /usr/local# $echo $SHADOOP_HOME
-bash: /usr/local/hadoop: Is a directory
root@hadoopmaster: /usr/local#
```

- **Configuring Password-less & Passphrase-less Login**

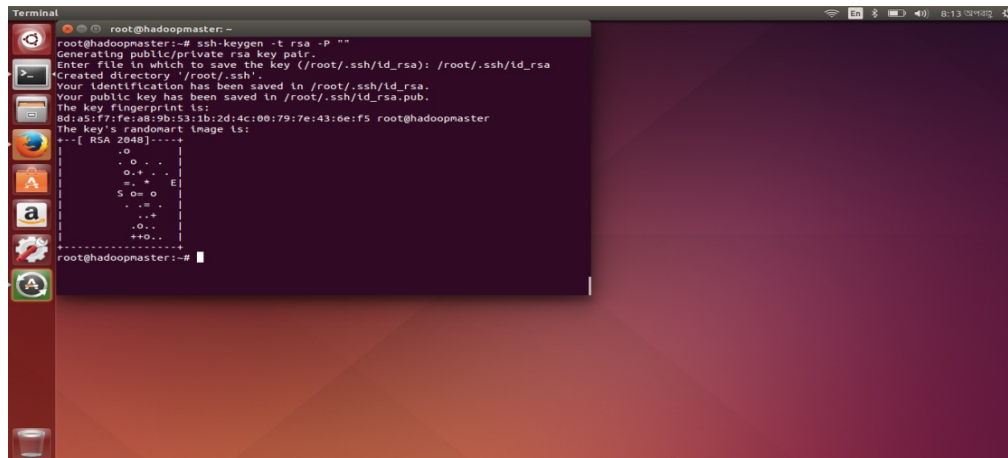
Hadoop provide scripts ( e.g. start-mapred.sh and start-dfs.sh). Use ssh in order to start and stop the various daemons and some other utilities. To work seamlessly, ssh need to be setup to allow password-less & passphrase-less login for the root/hadoop user from machines in the cluster. The simplest way to achieve this is to generate a public/private key pair, and it will be shared across the cluster.

Now check that you can ssh to the localhost without passphrase .

```
root@hadoopmaster:/usr/local/# ssh localhost
```

If you cannot ssh to localhost without a passphrase, generate an ssh key for the root user

```
root@hadoopmaster:~# ssh-keygen -t rsa -P ""
```



```
Terminal
root@hadoopmaster:~# ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /root/.ssh/id_rsa
Created directory /root/.ssh/.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
8d:a5:f7:fe:a8:9b:53:1b:2d:4c:00:79:7e:43:6e:f5 root@hadoopmaster
The key's randomart image is:
+--[ RSA 2048 ]-----+
|.O.+.
|O+.
|=+ +E
|S.O=O
|+.
|..+
|.O.
|+O.
+-----+
root@hadoopmaster:~#
```

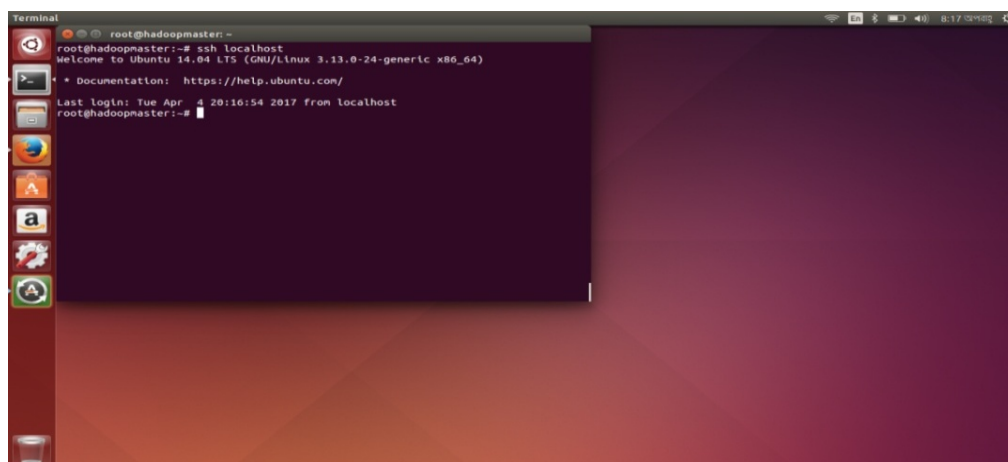
If prompted for SSH key file name, Enter file in which to save the key (/root/.ssh/id\_rsa) and press ENTER. Next put the key to authorized\_keys directory for future password-less access.

```
root@hadoopmaster:~# cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
```

```
root@hadoopmaster:~# chmod 700 ~/.ssh
```

```
root@hadoopmaster:~# chmod 600 ~/.ssh/authorized_keys
```

Connect and validate ssh password-less login to localhost.



```
Terminal
root@hadoopmaster:~# ssh localhost
root@hadoopmaster:~#
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)
 * Documentation:  https://help.ubuntu.com/
Last login: Tue Apr 4 20:16:54 2017 from localhost
root@hadoopmaster:~#
```

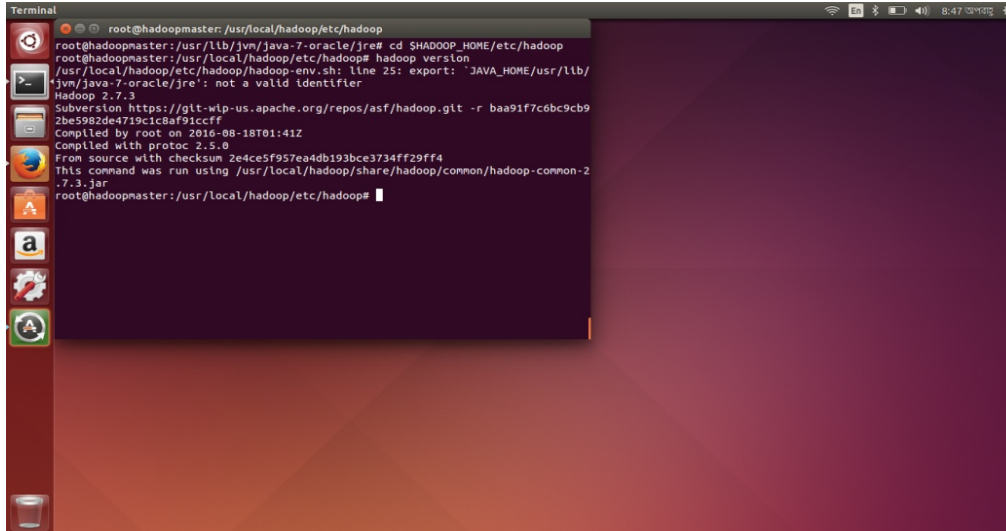
- **Confirm Hadoop is installed**

At this point, we should confirm if hadoop command is accessible from the terminal

```
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre# cd $HADOOP_HOME/etc/hadoop
```

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# hadoop version
```

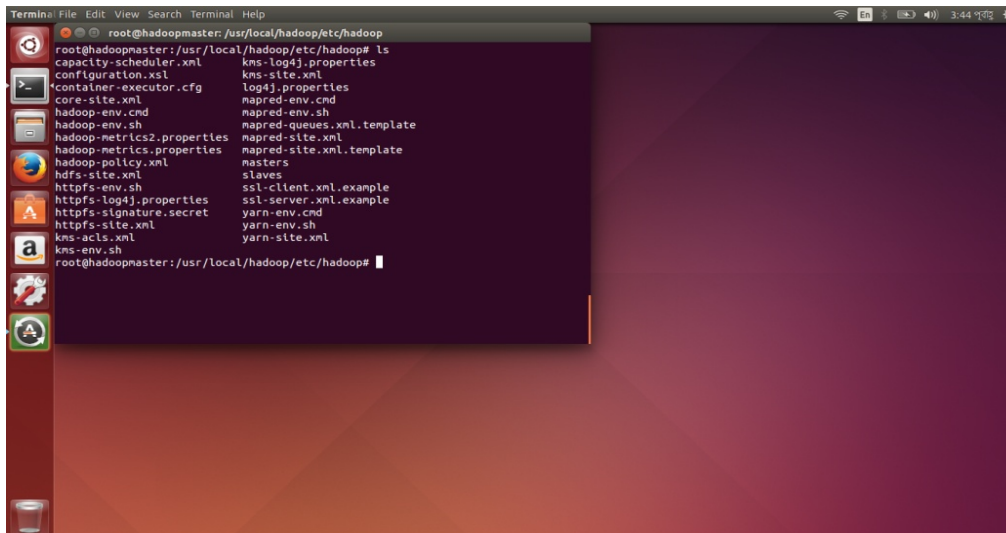
#output looks like below



```
Terminal
root@hadoopmaster: /usr/local/hadoop/etc/hadoop
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre# cd $HADOOP_HOME/etc/hadoop
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# hadoop version
/usr/local/hadoop/etc/hadoop/hadoop-env.sh: line 25: export: `JAVA_HOME/usr/lib/
jvm/java-7-oracle/jre': not a valid identifier
Hadoop 2.7.3
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r baa91f7c6bc9b9
2be59e2de4719c1c8af91ccff
Compiled by root on 2016-08-18T01:41Z
Compiled with protoc 2.5.0
From source with checksum 2e4ce5f957ea4db193bce3734ff29ff4
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2
.7.3.jar
root@hadoopmaster:/usr/local/hadoop/etc/hadoop#
```

- **Configuring the hadoopmaster**

You have to configure some Hadoop server files so that we can start the Hadoop Cluster. These configuration files are located under /usr/local/hadoop/etc/hadoop folder. In this configuration we will use YARN as the cluster management framework.



```
Terminal File Edit View Search Terminal Help
root@hadoopmaster: /usr/local/hadoop/etc/hadoop
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# ls
capacity-scheduler.xml      kms-log4j.properties
configuration.xml           kms-site.xml
container-executor.cfg      log4j.properties
core-site.xml               mapred-env.cmd
hadoop-env.cmd              mapred-env.sh
hadoop-env.sh               mapred-queues.xml.template
hadoop-metrics2.properties  mapred-site.xml
hadoop-metrics.properties   mapred-site.xml.template
hadoop-policy.xml           masters
hdfs-site.xml               slaves
https-env.sh                ssl-client.xml.example
https-log4j.properties      ssl-server.xml.example
https-signature.secret      yarn-env.cmd
https-site.xml              yarn-env.sh
kms-acls.xml                 yarn-site.xml
kms-env.sh
root@hadoopmaster:/usr/local/hadoop/etc/hadoop#
```

Among all this configuration files we will configure some minimal configuration files just to get us started on the Hadoop Cluster. Here is a description why we configure these files.

Configuration Filenames	Description of Log Files
hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Configuration Settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce
hdfs-site.xml	Configuration settings for HDFS daemons, the NameNode, the SecondaryNameNode and the DataNode
mapred-site.xml	Configure settings for run as Yarn application and MapReduce daemons : JobHistoryServer
yarn-site.xml	Configure settings for Yarn daemons : the ResourceManager and the NodeManager
masters	A list of machines(one per line) that each run a NameNode &SecondaryNameNode
slaves	A list of machines(one per line) that each run a DataNode and NodeManager

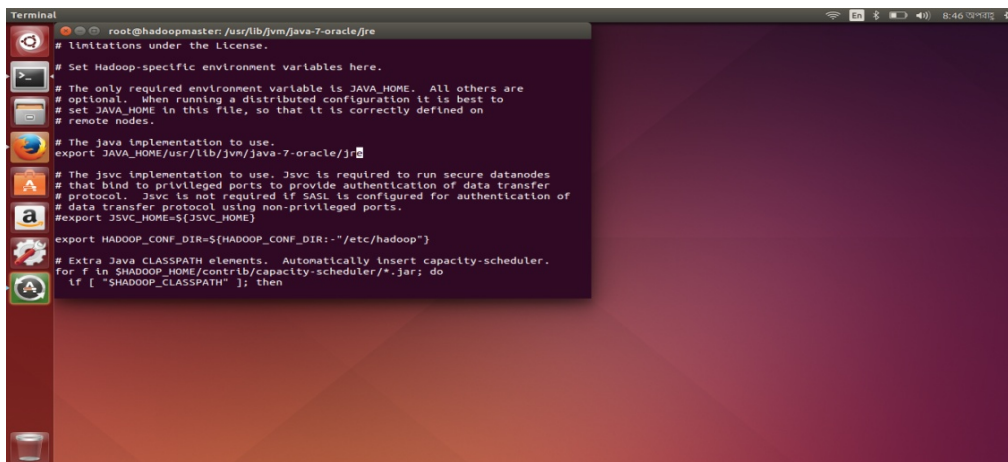
- **Configure hadoop-env.sh**

One of the important environment variable for Hadoop daemon is \$JAVA\_HOME in hadoop-env.sh. This variable direct Hadoop daemon to the Java path in the system. From the base of Hadoop installation path ( e.g. /usr/local/hadoop ), open the etc/hadoop/hadoop-env.sh file,

```
root@hadoopmaster:/usr/lib/jvm/java-7-oracle/jre# vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

Inside the file, find the line `export JAVA_HOME=${JAVA_HOME}`. Replace the line like below

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
```



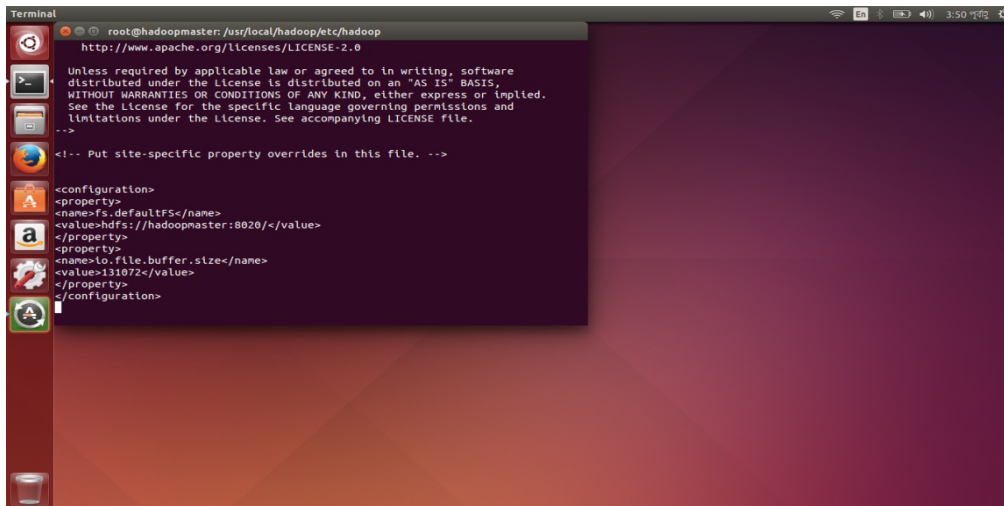
- **Configure core-site.xml**

From the base of Hadoop installation path (e.g. /usr/local/hadoop ), edit the etc/hadoop/core-site.xml file. The original installed file will have no entrees other than the <configuration> </configuration> tags. There are two properties that need to be set. The first is the fs.defaultFS property that sets the host and request port for the hadoopmaster (Metadata server for HDFS ). The second is io.file.buffer.size, which is the size of read/write buffer used in sequenceFiles. Open the core-site.xml from the terminal by giving the following command.

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi core-site.xml
```

Copy the following lines and remove the original empty <configuration> </configuration> tags.

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://NameNode:8020</value>
</property>
<property>
<name>io.file.buffer.size</name>
<value>131072</value>
</property>
</configuration>
```



- **Configure hdfs-site.xml**

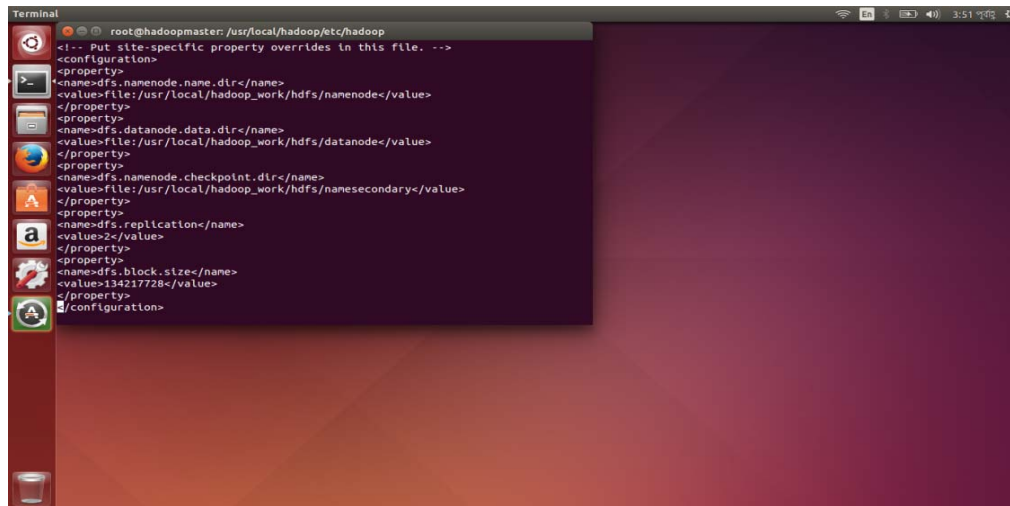
From the base of the Hadoop installation path, edit the etc/hadoop/hdfs-site.xml. In hdfs-site.xml we specify where is the name node directory (which we created previously at the end of Hadoop installation) and how many backup copies of the data files to be created in the system (called replication) inside the file under the configuration tag. Open the hdfs-site.xml from the terminal by giving the following command.



```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi hdfs-site.xml
```

Copy the following lines and remove the original empty `<configuration>` `</configuration>` tags.

```
<configuration>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_work/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_work/hdfs/datanode</value>
</property>
<property>
<name>dfs.namenode.checkpoint.dir</name>
<value>file:/usr/local/hadoop_work/hdfs/namesecondary</value>
</property>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
<property>
<name>dfs.block.size</name>
<value>134217728</value>
</property>
</configuration>
```



- **Configure mapred-site.xml**

From the base of the Hadoop installation, edit the etc/hadoop/mapred-site.xml file. In this install we will use the value of “yarn” to tell MapReduce that it will run as a YARN application. Also we will configure the MapReduce Job History Server. First, copy the template file to the mapred-site.xml.

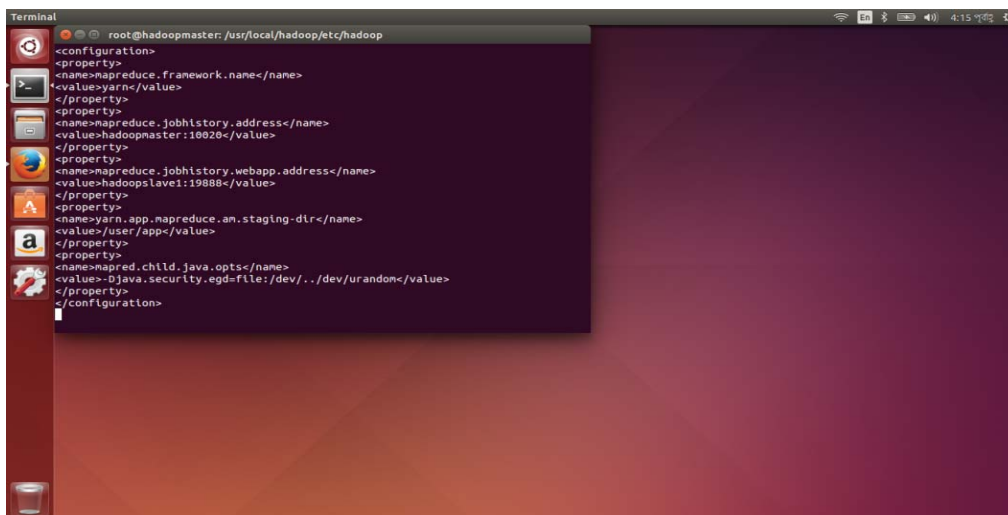
```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# cp mapred-site.xml.template mapred-site.xml
```

Then open the mapred-site.xml from terminal by giving the following command

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi mapred-site.xml
```

Next, copy the following lines and remove the original empty <configuration> </configuration> tags.

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
<property>
<name>mapreduce.jobhistory.address</name>
<value>hadoopmaster:10020</value>
</property>
<property>
<name>mapreduce.jobhistory.webapp.address</name>
<value>hadoopmaster:19888</value>
</property>
<property>
<name>yarn.app.mapreduce.am.staging-dir</name>
<value>/user/app</value>
</property>
<property>
<name>mapred.child.java.opts</name>
<value>-Djava.security.egd=file:/dev/./dev/urandom</value>
</property>
</configuration>
```

A screenshot of a terminal window on a Linux system. The terminal title is "Terminal" and the prompt is "root@hadoopmaster: /usr/local/hadoop/etc/hadoop". The terminal displays the XML configuration for mapred-site.xml, which is identical to the text provided in the previous block. The configuration includes properties for the mapreduce framework name, job history address, job history webapp address, yarn app staging directory, and mapred child java options. The terminal background is dark, and the text is white. The window title bar shows system icons and the time "4:15".



- **Configure yarn-site.xml**

From the base of the Hadoop installation, edit the etc/hadoop/yarn-site.xml file. In this configuration we setup YARN site specific properties for Resource Manager & Node Manager. Open the file :

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi yarn-site.xml
```

Copy the following to the Hadoop etc/hadoop/yarn-site.xml file and remove the original empty <configuration> </configuration> tags.

```
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>hadoopmaster</value>
</property>
<property>
<name>yarn.resourcemanager.bind-host</name>
<value>0.0.0.0</value>
</property>
<property>
<name>yarn.nodemanager.bind-host</name>
<value>0.0.0.0</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
<name>yarn.log-aggregation-enable</name>
<value>true</value>
</property>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>file:/usr/local/hadoop_work/yarn/local</value>
</property>
<property>
<name>yarn.nodemanager.log-dirs</name>
<value>file:/usr/local/hadoop_work/yarn/log</value>
</property>
<property>
<name>yarn.nodemanager.remote-app-log-dir</name>
<value>hdfs://hadoopmaster:8020/var/log/hadoop-yarn/apps</value>
</property>
</configuration>
```

## yarn-site.xml

```
root@hadoopmaster: /usr/local/hadoop/etc/hadoop
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>hadoopmaster</value>
</property>
<property>
<name>yarn.resourcemanager.bind-host</name>
<value>0.0.0.0</value>
</property>
<property>
<name>yarn.nodemanager.bind-host</name>
<value>0.0.0.0</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
<name>yarn.log-aggregation-enable</name>
<value>true</value>
</property>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>file:/usr/local/hadoop_work/yarn/local</value>
</property>
<property>
<name>yarn.nodemanager.log-dirs</name>
<value>file:/usr/local/hadoop_work/yarn/log</value>
</property>
<property>
<name>yarn.nodemanager.remote-app-log-dir</name>
<value>hdfs://hadoopmaster:8020/var/log/hadoop-yarn/apps</value>
</property>
</configuration>
```

- **Setup the masters**

From the base of the Hadoop installation, edit the etc/hadoop/masters . The masters file at Master server contains a hostname of Secondary Name Node servers. We will configure hadoopmaster as the host name of secondary name node. Open the terminal

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi masters
```

Inside the file, put one line:  
hadoopmaster

```
Terminal File Edit View Search Terminal Help
root@hadoopmaster: /usr/local/hadoop/etc/hadoop
hadoopmaster
"masters" 1 line, 13 characters
```

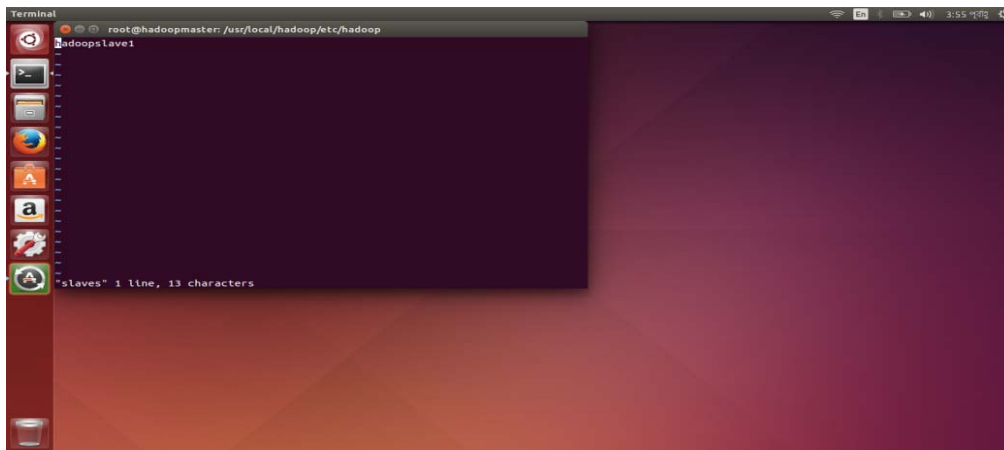
- **Setup the slaves**

From the base of the Hadoop installation, edit the etc/hadoop/slaves. The slaves file at Master node contain a list of host, one per line, that are to host of DataNode and NodeManager servers. Here we define hadoopslave1 as the host of DataNode and NodeManager . Open the terminal:

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# vi slaves
```

Inside the file put one line:

```
hadoopslave1
```



## Creating The hadoopslave1

- **Installing JAVA in hadoopslave1**

```
root@hadoopslave1:~# apt-get update
```

```
root@hadoopslave1:~# add-apt-repository ppa:webupd8team/java
```

```
root@hadoopslave1:~# apt-get update
```

```
root@hadoopslave1:~# apt-get install oracle-java7-installer
```

- **Installing ssh**

```
root@hadoopslave1:~# apt-get install openssh-server
```

- **Mapping the nodes :**

```
root@hadoopslave1:~# vi /etc/hosts
```

Enter the following lines in the /etc/hosts files.

192.168.2.130 hadoopmaster

192.168.2.131 hadoopslave1

- **Setup Environment Variables**

Open the .bashrc file

```
root@hadoopslave1:~# vi ~/.bashrc
```

And put these lines at the end of your .bashrc file

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_CLASSPATH=/usr/lib/jvm/java-7-oracle/lib/tools.jar
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/lib/*:.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.security.egd=file:/dev/./dev/urandom"
```

- **Configuring Password-less & Passphrase-less Login**

```
root@hadoopslave1:~# ssh-keygen -t rsa -P ""
```

```
root@hadoopslave1:~# cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
```

```
root@hadoopslave1:~# chmod 700 ~/.ssh
```

```
root@hadoopslave1:~# chmod 600 ~/.ssh/authorized_keys
```

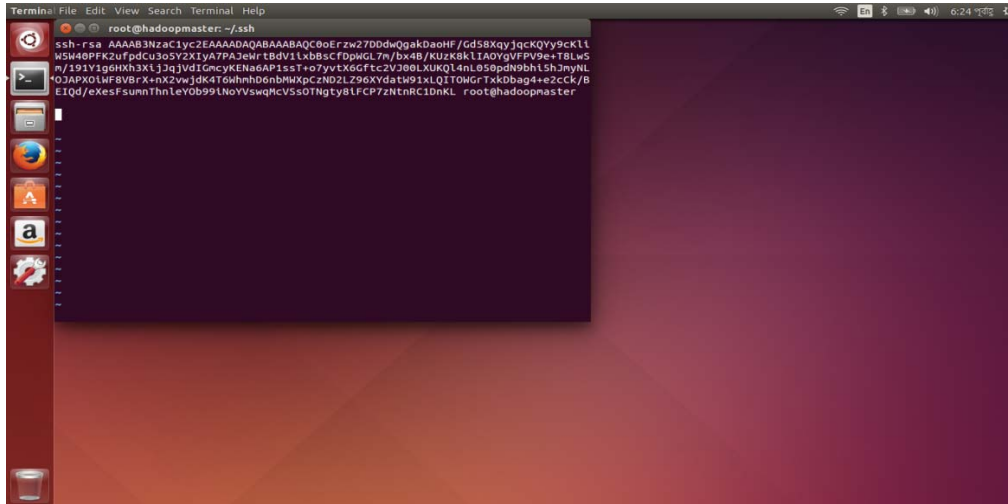
- **Hadoop Installation**

This time we will not install Hadoop from beginning as we have already installed it in the hadoopmaster . Instead, we will copy the installation directories from the hadoopmaster to the hadoopslave1. In order to copy the directory, we will first need to include the public key of the hadoopmaster to the authorized keys of the hadoopslave1 and also the public key of the hadoopslave1 to the authorized keys of the hadoopmaster so that we can use the SSH file copy program (scp) for copying.

Now copy the public key from hadoopslave1 and paste it to hadoopmaster

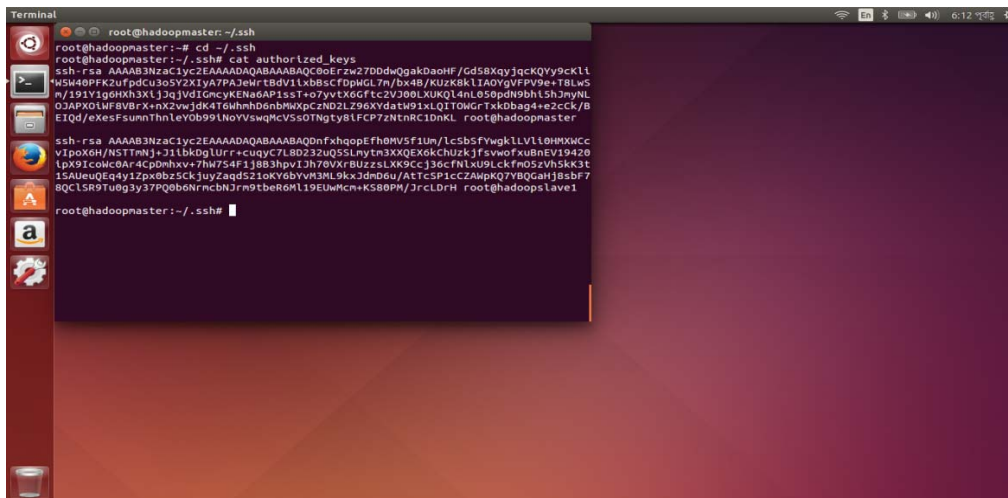
```
root@hadoopmaster:~# cd ~/.ssh
```

```
root@hadoopmaster:~/.ssh# vi authorized_keys
```



Now paste the public key of hadoopslave1 and then confirm the editing by the following command.

```
root@hadoopmaster:~/.ssh# cat authorized_keys
```

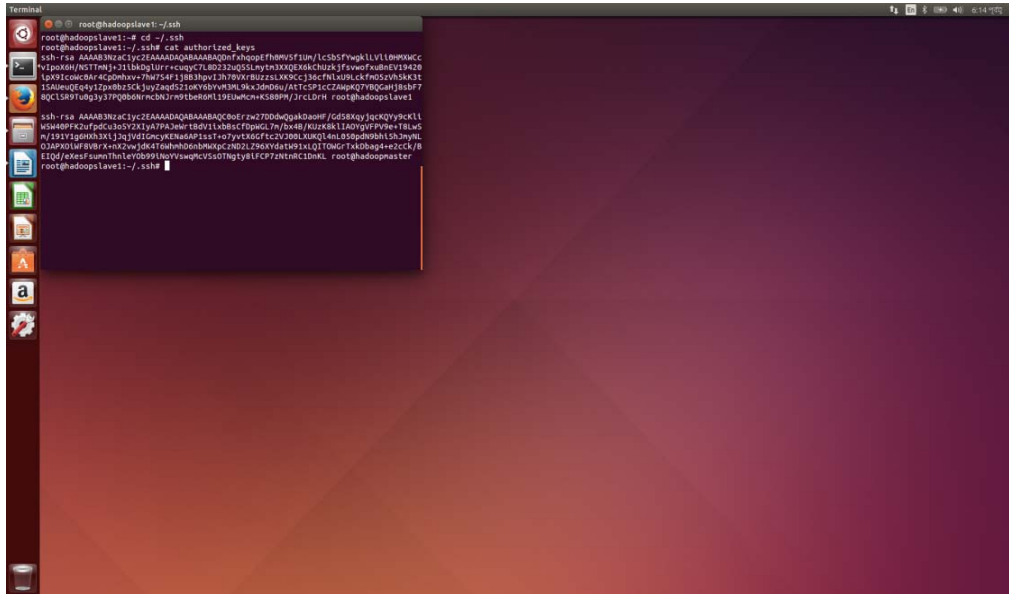


Now copy the public key hadoopmaster and paste it to hadoopslave1

```
root@hadoopslave1:~# cd ~/.ssh
```

```
root@hadoopslave1:~/.ssh# vi authorized_keys
```

root@hadoopslave1:~/ssh# cat authorized\_keys

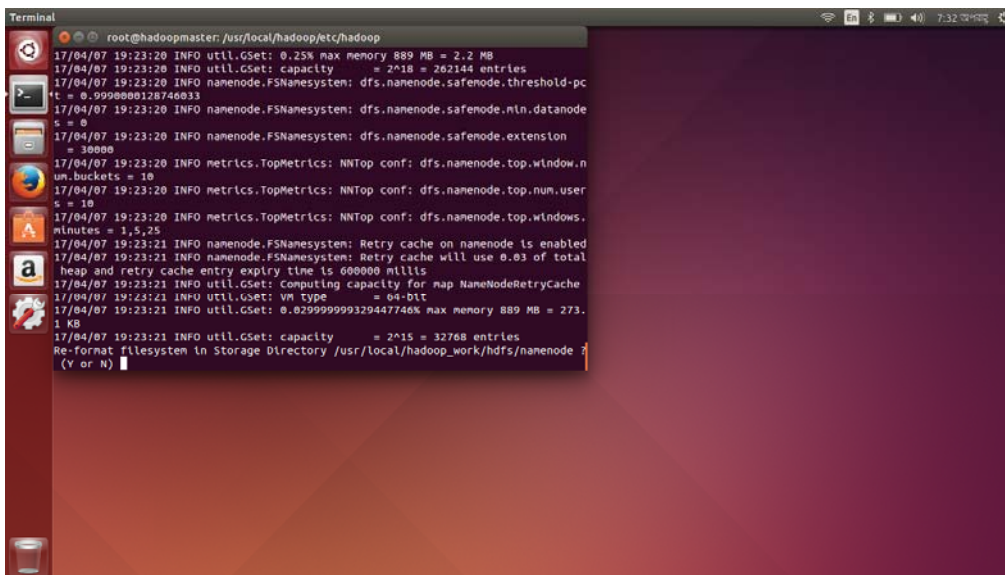


- **Format Name Node on hadoopmaster**

We will format the name node before starting anything now.

```
root@hadoopmaster:/usr/local/hadoop/etc/hadoop# /usr/local/hadoop/bin/hadoop namenode -format
```

To check whether namenode has format give the command again and you should see Re-format filesystem in storage Directory /usr/local/hadoop\_work/namenode. Always give no command for re-format. This means the namenode format has been done successfully.



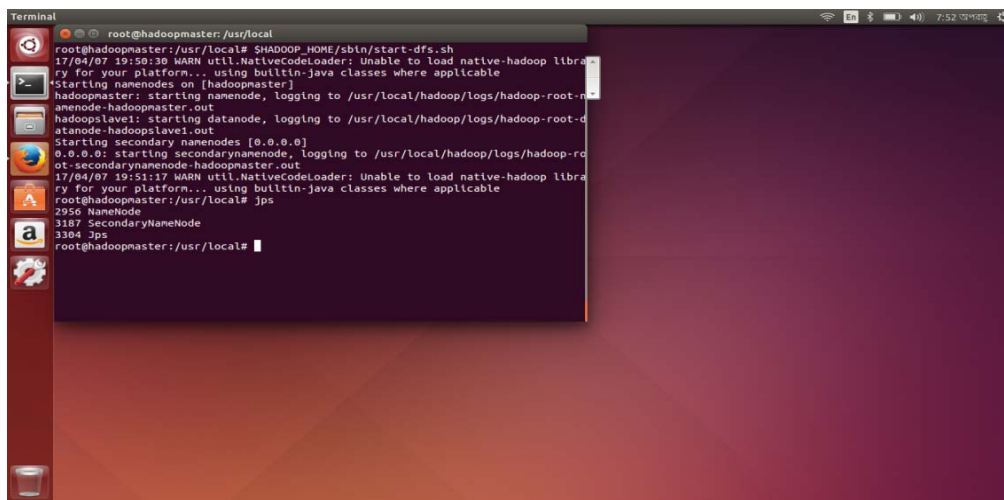
- **start HDFS services**

It's time to start the Hadoop Distributed File System in the hadoopmaster as well as the hadoopslave1 in the Hadoop Cluster.

```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/start-dfs.sh
```

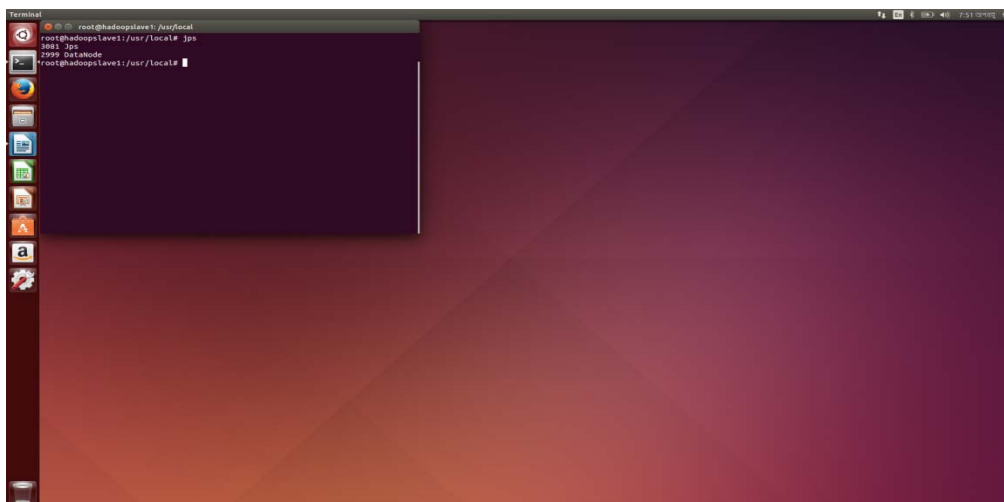
After the command executed you should see NameNode and SecondaryNameNode started as java process in hadoopmaster and DataNode as java process in hadoopslave1. To check whether the services are running or not give a jps command on both hadoopmaster and hadoopslave1 .

```
root@hadoopmaster:/usr/local# jps
```



```
Terminal
root@hadoopmaster:/usr/local
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/start-dfs.sh
17/04/07 19:50:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [hadoopmaster]
hadoopmaster: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-hadoopmaster.out
hadoopslave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-hadoopslave1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-hadoopmaster.out
17/04/07 19:51:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@hadoopmaster:/usr/local# jps
2956 NameNode
3187 SecondaryNameNode
3304 Jps
root@hadoopmaster:/usr/local#
```

```
root@hadoopslave1:/usr/local# jps
```



```
Terminal
root@hadoopslave1:/usr/local
root@hadoopslave1:/usr/local# jps
3982 Jps
2999 DataNode
root@hadoopslave1:/usr/local#
```

If the process did not start, it may be helpful to inspect the log files.

All Hadoop services can be stopped using the following command.

```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/stop-dfs.sh
```

Once the Namenode & Datanodes starts successfully, we have to create few directories in hadoop filesystem which has been listed in our site specific configuration files. These HDFS directories will be used by YARN MapReduce Staging, YARN Log & JobHistoryServer.

```
root@hadoopmaster:/usr/local# hadoop fs -mkdir /tmp
root@hadoopmaster:/usr/local# hadoop fs -chmod -R 1777 /tmp
```

```
root@hadoopmaster:/usr/local# hadoop fs -mkdir /user
root@hadoopmaster:/usr/local# hadoop fs -chmod -R 1777 /user
```

```
root@hadoopmaster:/usr/local# hadoop fs -mkdir /user/app
root@hadoopmaster:/usr/local# hadoop fs -chmod -R 1777 /user/app
```

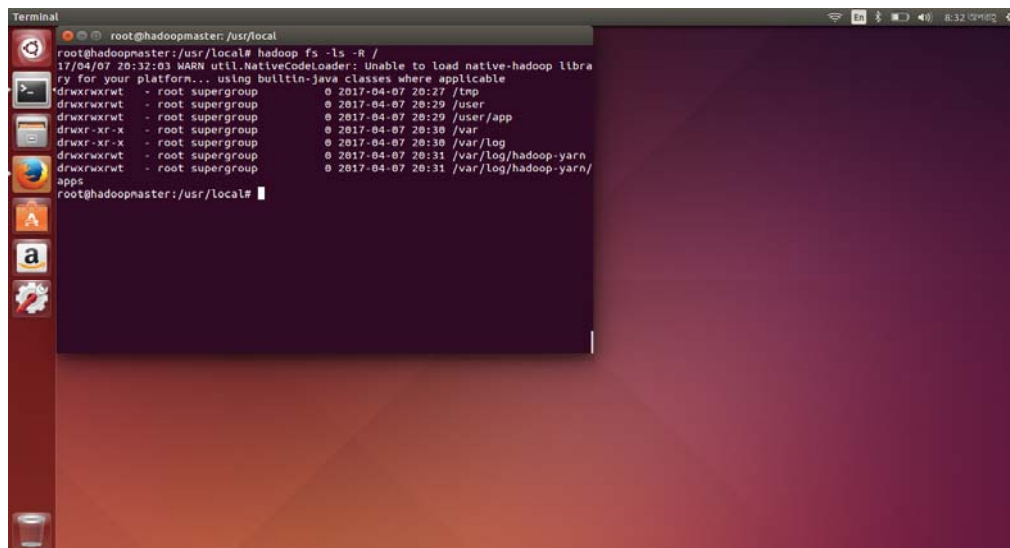
```
root@hadoopmaster:/usr/local# hadoop fs -mkdir -p /var/log/hadoop-yarn
root@hadoopmaster:/usr/local# hadoop fs -chmod -R 1777 /var/log/hadoop-yarn
```

```
root@hadoopmaster:/usr/local# hadoop fs -mkdir -p /var/log/hadoop-yarn/apps
root@hadoopmaster:/usr/local# hadoop fs -chmod -R 1777 /var/log/hadoop-yarn/apps
```

# Now Verify the HDFS File Structure

```
root@hadoopmaster:/usr/local# hadoop fs -ls -R /
```

# Output should look like below:



```
Terminal
root@hadoopmaster:/usr/local
root@hadoopmaster:/usr/local# hadoop fs -ls -R /
17/04/07 20:32:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
drwxr-xrwt - root supergroup @ 2017-04-07 20:27 /tmp
drwxr-xrwt - root supergroup @ 2017-04-07 20:29 /user
drwxr-xrwt - root supergroup @ 2017-04-07 20:29 /user/app
drwxr-xr-x - root supergroup @ 2017-04-07 20:30 /var
drwxr-xr-x - root supergroup @ 2017-04-07 20:30 /var/log
drwxr-xrwt - root supergroup @ 2017-04-07 20:31 /var/log/hadoop-yarn
drwxr-xrwt - root supergroup @ 2017-04-07 20:31 /var/log/hadoop-yarn/apps
root@hadoopmaster:/usr/local#
```



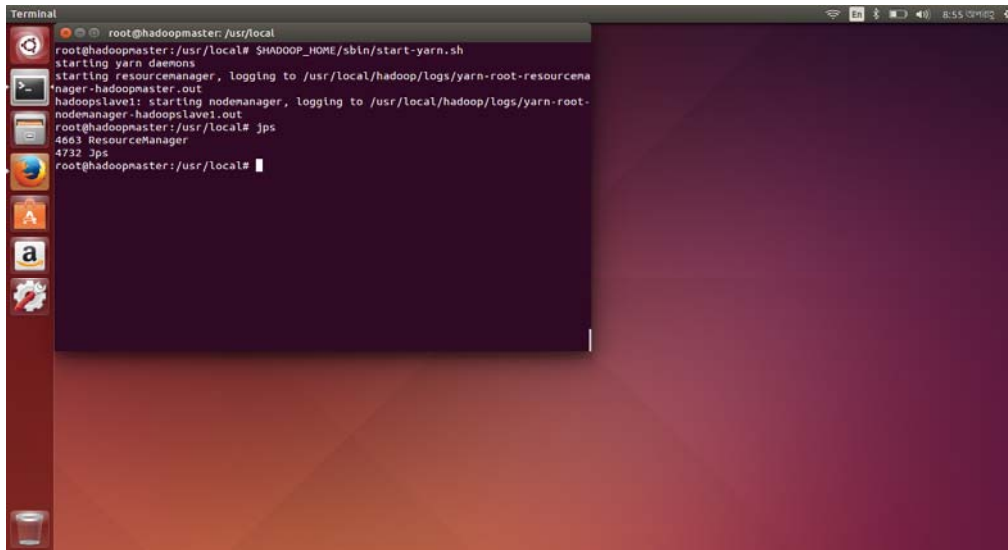
- **start YARN services**

After HDFS services started, YARN services need to be started.

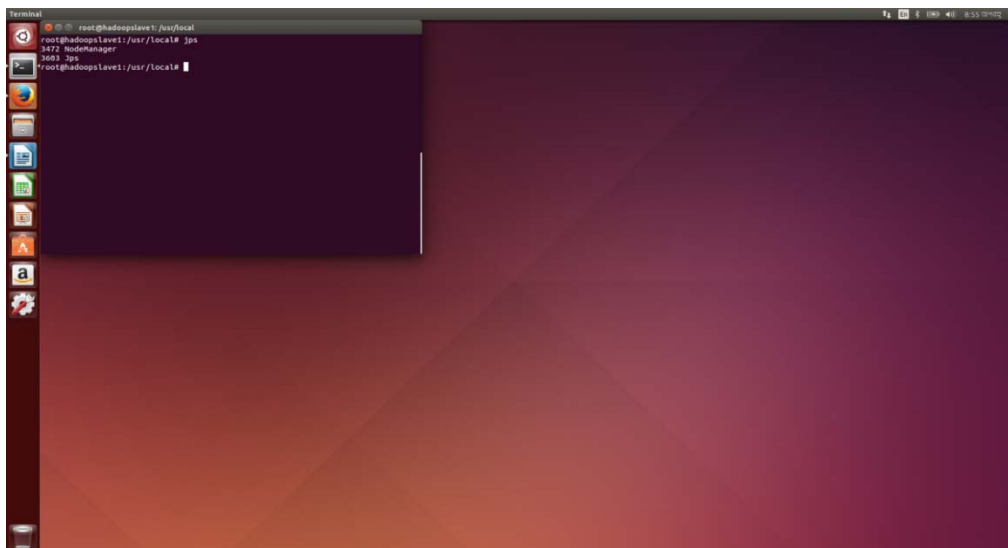
```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/start-yarn.sh
```

ResourceManager in hadoopmaster and NodeManager in hadoopslave must be started.  
To check whether the services are running or not give a jps command on both hadoopmaster and hadoopslave1 .

```
root@hadoopmaster:/usr/local# jps
```



```
root@hadoopslave1:/usr/local# jps
```



- **start MapReduce HistoryServer**

Before starting the HistoryServer quickly edit the \$HADOOP\_HOME/etc/hadoop/mapred-site.xml in the hadoopmaster. Replace the hostname in the value for the property names as below from hadoopmaster to 0.0.0.0:

mapreduce.jobhistory.address from hadoopmaster:10020 to 0.0.0.0:10020

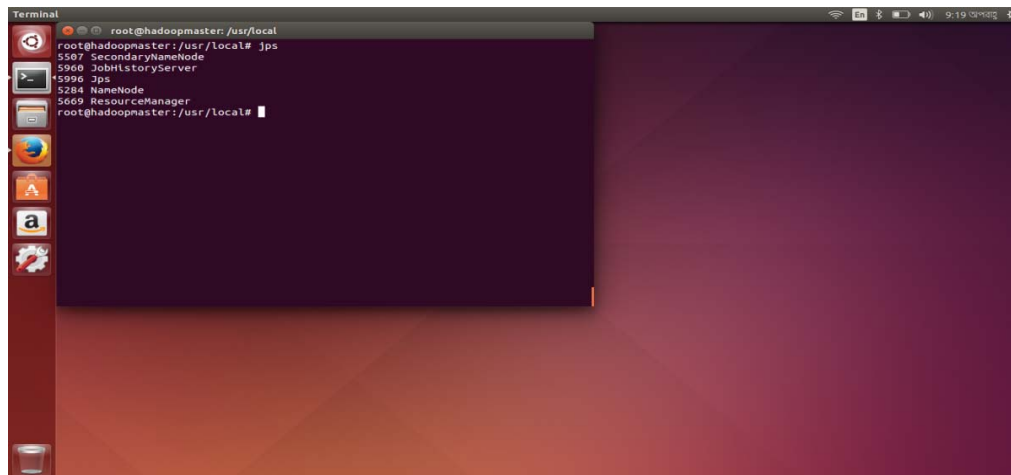
mapreduce.jobhistory.webapp.address from hadoopmaster:19888 to 0.0.0.0:19888

Now run the below command if it runs well you should see JobHistoryServer in hadoopmaster .

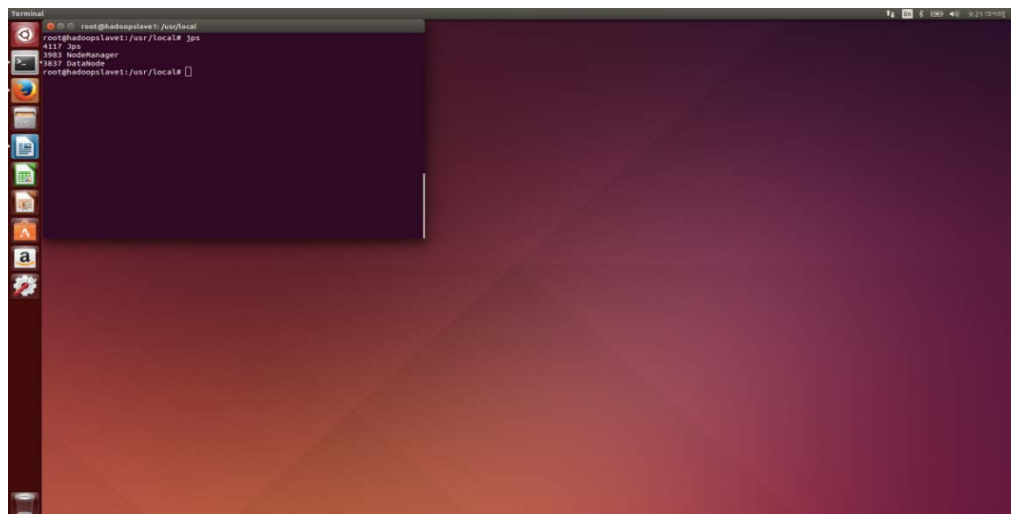
```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

Finally see all the daemons running in the hadoopmaster as well as the hadoopslave1.

```
root@hadoopmaster:/usr/local# jps
```



```
root@hadoopslave1:/usr/local# jps
```

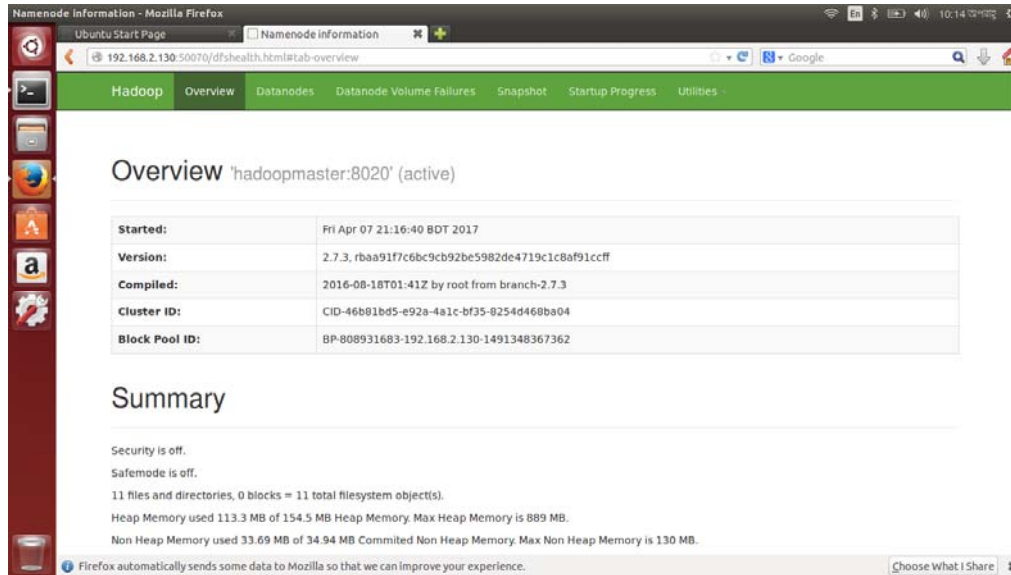


Now the cluster is set up and running .

- **Verify the running services using the Web Interface**

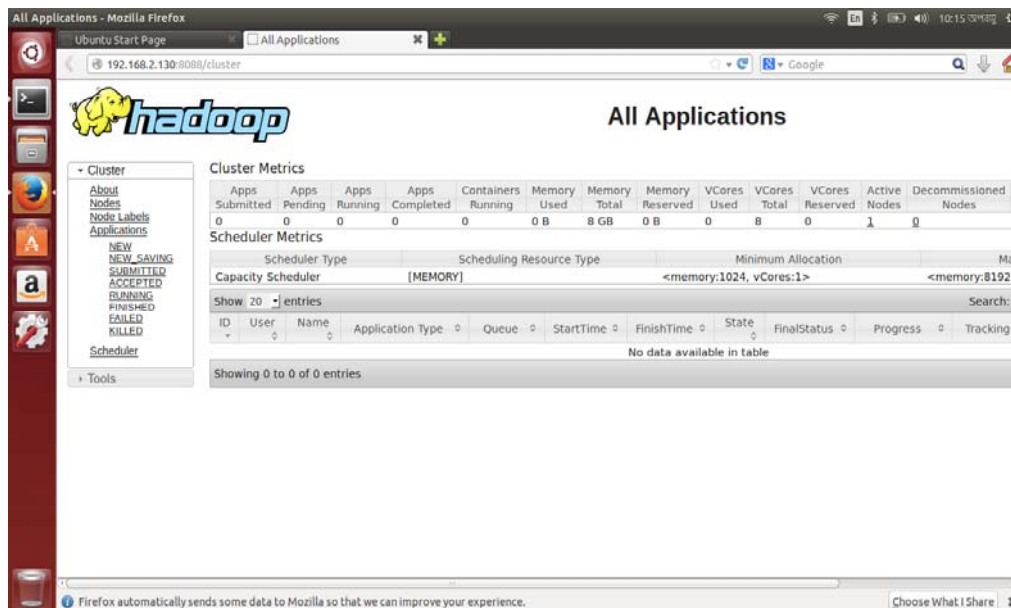
Let us check the cluster status in the web browser. HDFS, ResourceManager & HistoryServer have a web interface. To monitor HDFS enter the following web address:

<http://192.168.2.130:50070>



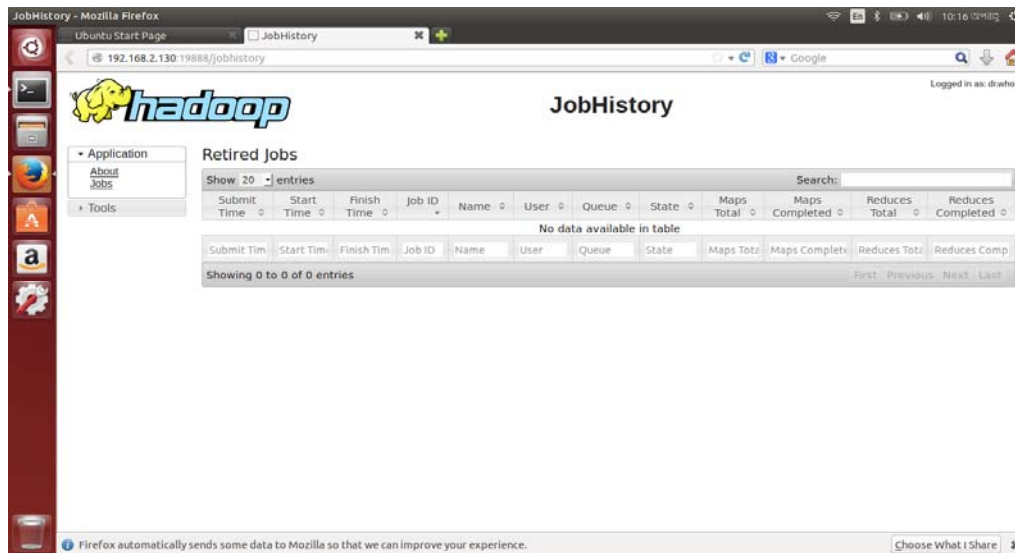
To monitor ResourceManager enter the following address:

<http://192.168.2.130:8088>



To monitor HistoryServer enter the following address:

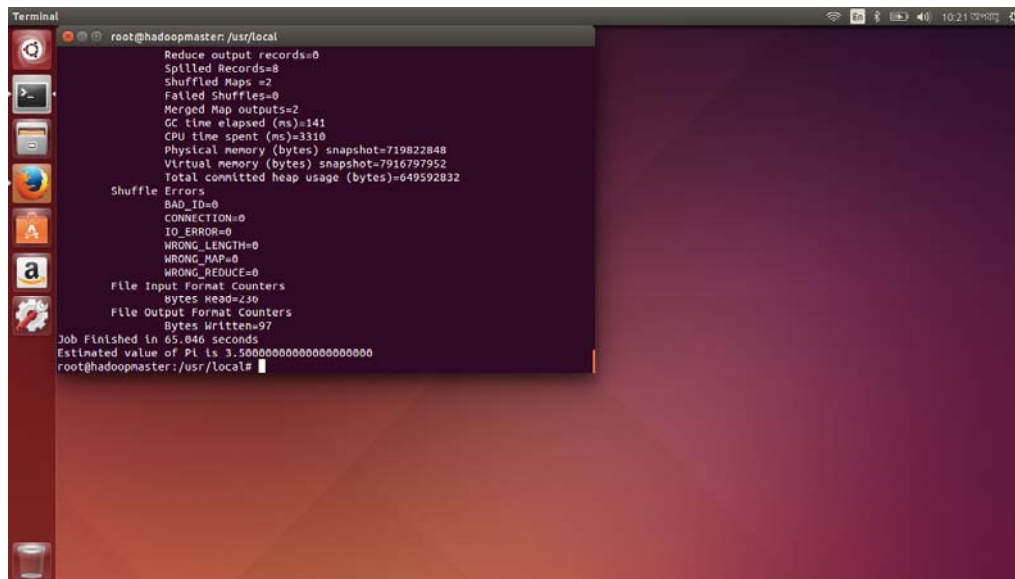
<http://192.168.2.130:19888>



Finally to test installation we will run sample MapReduce example. We will run the sample “pi” program that calculate the value of pi.

```
root@hadoopmaster:/usr/local# hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar pi 2 4
```

If the program worked correctly, the following should be displayed at the end of the program output stream. Estimated value of pi is 3.50000000000000000000



# Chapter 5

## Implementing MapReduce Application

MapReduce is known as a core component of the Apache Hadoop software framework. MapReduce is define as a framework using MapReduce we can implement applications to process huge amounts of data on Hadoop Cluster. MapReduce can be define as a processing technique and a program model for distributed computing which is based on java.

A MapReduce Application is consist of three class.

- i. Mapper class
- ii. Reducer class
- iii. Driver class

**Mapper class:** The Mapper class reads the input files as <key1,value1> pair and then split the words and finally give the output as <key2,value2> pair to the reducer.

**Reducer class:** The Reducer class read the output of Mapper class as <key2,value2> pair and combine them. Finally give the output to Hadoop Cluster as <key3,value3>.

**Driver class:** The Driver class is used to build the configuration of the job and then submit the job to the Hadoop cluster. Also the Driver class contains the main() method and the main() method accepts argument from the command line.

### Example :WordCount

Here, we take an example of WordCount MapReduce application to get a flavor of how MapReduce work. We implementing WordCount as a simple application that is used to search word and count the number of occurrences of that search word in a given input data.

### Set Environment Variables:

We assume that environment variables are set as given below in hadoop multi-node cluster.

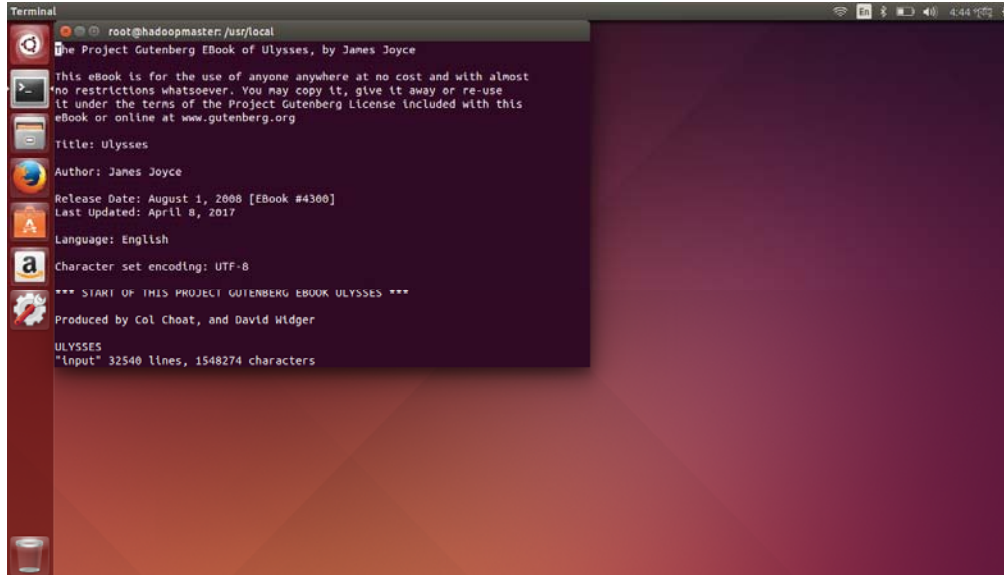
```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
export PATH=/usr/lib/jvm/java-7-oracle/lib/tools.jar
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

As we have not installed jdk so the tools.jar in not in the default JAVA\_HOME/lib/ . We have installed jre so the tools.jar present at /usr/lib/jvm/java-7-oracle/lib/tools.jar

In case you have not install jdk so insure the path of tools.jar otherwise javac command will not executed because of mentioning wrong path of tools.jar at .bashrc file.

## Input Data

We have taken an Ebook Ulysses by James Joyce for testing WordCount program. This input file contains 268,030 words. The input file looks as shown below:



## Source Program: WordCount.java

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
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.io.LongWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    public static class WordMapper extends Mapper<LongWritable, Text, Text, LongWritable> {

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            Configuration confWithValue = context.getConfiguration();
            String searchValue = confWithValue.get("val");
            String[] splitSearchValue = searchValue.split(",");

            for(String splitValue : splitSearchValue) {
                context.write(new Text(splitValue.toLowerCase()), new LongWritable(0));
            }
        }
    }
}
```

```

String line = value.toString();
String[] words = line.split(" ");
for (String word : words) {
    for (String splitValue : splitSearchValue) {
        if (word.equalsIgnoreCase(splitValue)) {
            context.write(new Text(word.toLowerCase()), new LongWritable(1));
        }
    }
}
}

public static class WordReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}

public static void main(String[] args) throws Exception {

    String searchKey = "";
    Scanner scanner;
    int i = 0;
    scanner = new Scanner(new File("search_key.txt"));
    while (scanner.hasNext()) {
        if (i == 0) {
            searchKey = scanner.next();
        } else {
            searchKey += "," + scanner.next();
        }
        i++;
    }

    Configuration conf = new Configuration();
    conf.set("val", searchKey);

    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(WordMapper.class);
    job.setCombinerClass(WordReducer.class);
    job.setReducerClass(WordReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

## Compilation and Execution

Let us assume we are in the /usr/local directory where we have install Hadoop on Hadoop multi-node cluster.

Follow the steps in order to compile and execute above program.

### Step 1

The following command is to create an input file in your local system.

```
root@hadoopmaster:/usr/local# vi input
```

### Step 2

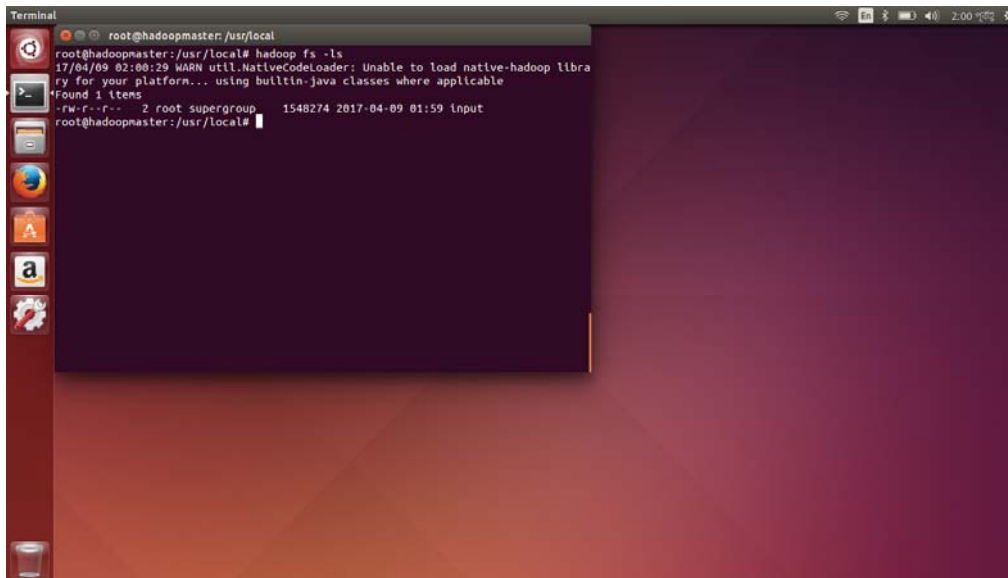
The following command is to copy the input data from your local system to HDFS .

```
root@hadoopmaster:/usr/local# hadoop fs -put input
```

### Step 3

The following command is to check whether the input file is copied to HDFS or not.

```
root@hadoopmaster:/usr/local# hadoop dfs -ls
```

A terminal window screenshot showing the execution of the 'hadoop dfs -ls' command. The terminal output includes a warning message about native Hadoop libraries and a listing of a file named 'input' in the HDFS root directory. The file details are: '-rw-r--r-- 2 root supergroup 1548274 2017-04-09 01:59 input'. The terminal prompt is root@hadoopmaster:~/usr/local#.

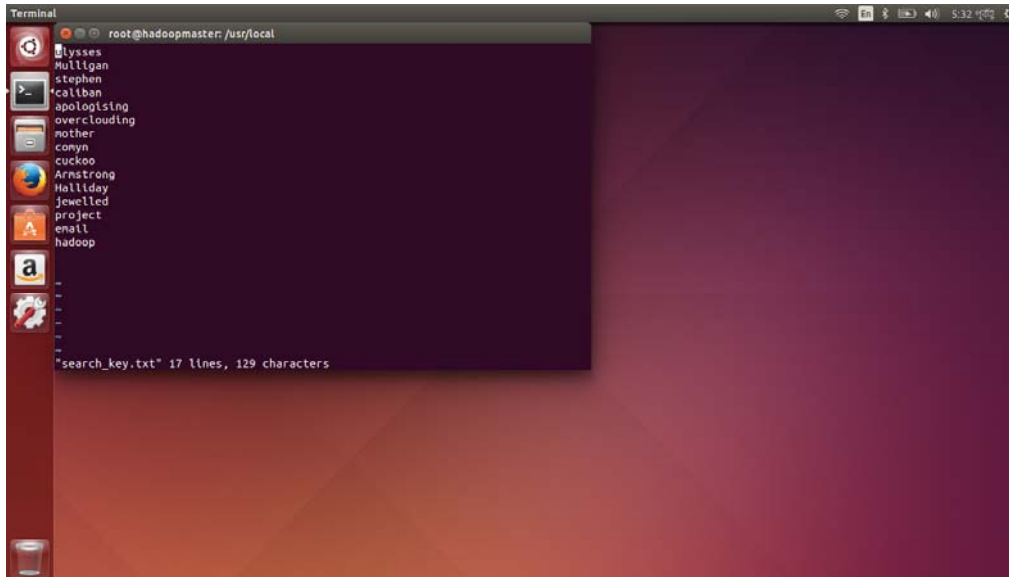
```
Terminal
root@hadoopmaster:~/usr/local
root@hadoopmaster:~/usr/local# hadoop fs -ls
17/04/09 02:00:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
*Found 1 items
-rw-r--r-- 2 root supergroup 1548274 2017-04-09 01:59 input
root@hadoopmaster:~/usr/local#
```



## Step 4

The following command is to create a search file in which is consisting of list of word for searching in input file. The search file should be creating your local system where you would create your source code.

```
root@hadoopmaster:/usr/local# vi search_key.txt
```



## Step 5

The following command is to create .java file in your local system.

```
root@hadoopmaster:/usr/local# vi WordCount.java
```

Now copy the above code in this file.

## Step 6

The following commands are used for compiling the **WordCount.java** program and creating a jar for the program.

```
root@hadoopmaster:/usr/local# hadoop com.sun.tools.javac.Main WordCount.java
```

```
root@hadoopmaster:/usr/local# jar -cf wordcount.jar WordCount*.class
```

Before running step 7 we need to ensure that all Hadoop master and slave node are running their jobs. If the jobs are not running then give the following commands in master node.

```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/start-dfs.sh
```

```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/start-yarn.sh
```

```
root@hadoopmaster:/usr/local# $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

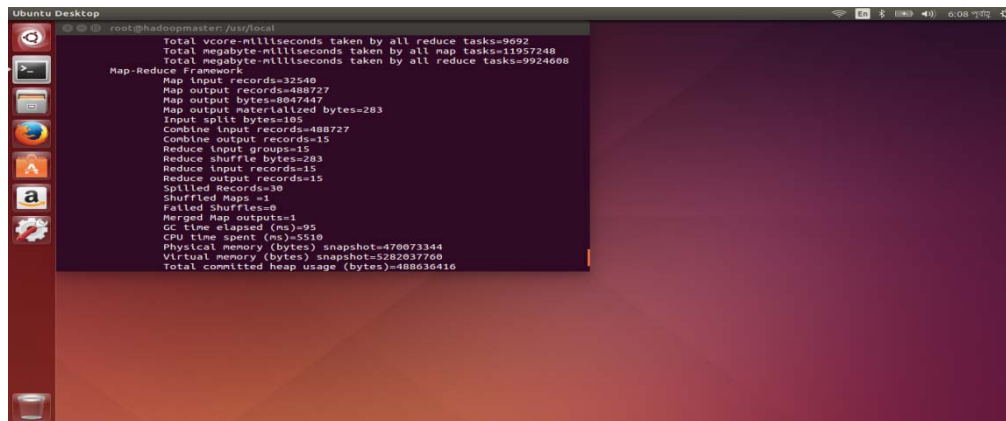
Now we can move to step 7

## Step 7

The following command is to run WordCount application by taking the input file from HDFS.

```
root@hadoopmaster:/usr/local# hadoop jar wordcount.jar WordCount input output search_key.txt 2 16
```

Wait until the execution is complete . At the execution time you will get an output like this.

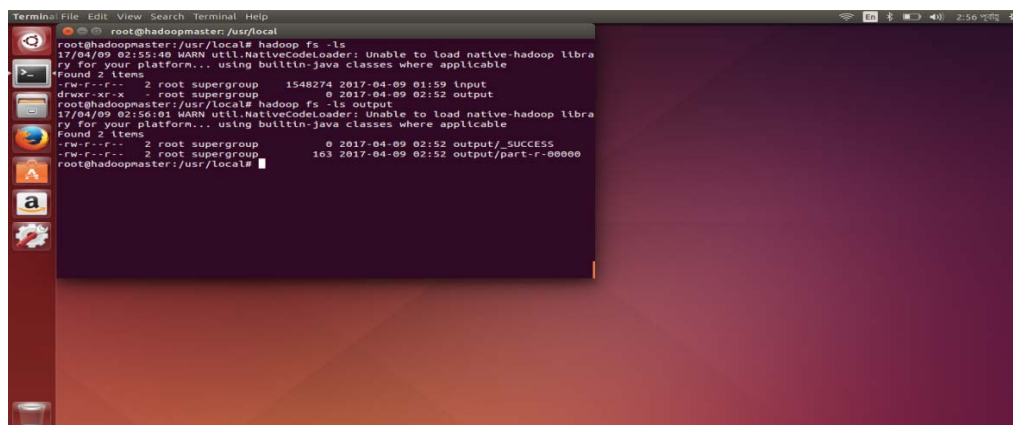


```
Ubuntu Desktop
root@hadoopmaster:/usr/local
Total vcore-millisecons taken by all reduce tasks=9692
Total megabyte-millisecons taken by all map tasks=11957248
Total megabyte-millisecons taken by all reduce tasks=9924098
Map-Reduce Framework
  Map input records=32540
  Map output records=488727
  Map output bytes=8847447
  Map output materialized bytes=283
  Input split bytes=105
  Combine input records=488727
  Combine output records=15
  Reduce input groups=15
  Reduce shuffle bytes=283
  Reduce input records=15
  Reduce output records=15
  Spilled Records=30
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=95
  CPU time spent (ms)=5510
  Physical memory (bytes) snapshot=470073344
  Virtual memory (bytes) snapshot=528237760
  Total committed heap usage (bytes)=488636416
```

## Step 8

The following command is to verify the output folder in HDFS .

```
root@hadoopmaster:/usr/local# hadoop dfs -ls output
```

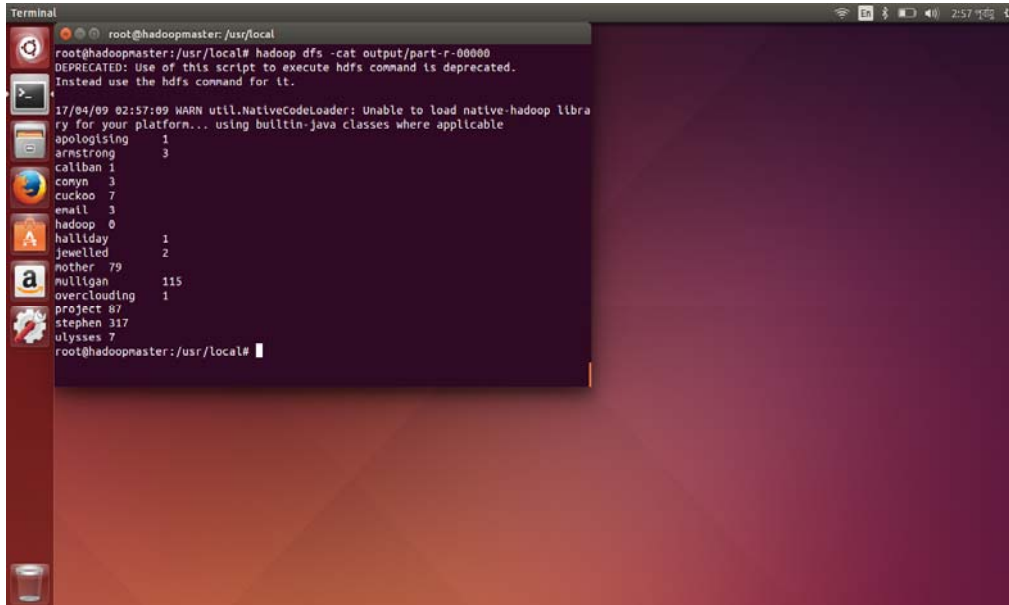


```
Terminal File Edit View Search Terminal Help
root@hadoopmaster:/usr/local
root@hadoopmaster:/usr/local# hadoop fs -ls
17/04/09 02:55:40 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 2 root supergroup 1548274 2017-04-09 01:59 input
drwxr-xr-x 2 root supergroup 0 2017-04-09 02:52 output
root@hadoopmaster:/usr/local# hadoop fs -ls output
17/04/09 02:56:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 2 root supergroup 0 2017-04-09 02:52 output/_SUCCESS
-rw-r--r-- 2 root supergroup 163 2017-04-09 02:52 output/part-r-00000
root@hadoopmaster:/usr/local#
```

## Step 9

The following command is to see the output in part-r-00000 which is in HDFS output folder.

```
root@hadoopmaster:/usr/local# hadoop dfs -cat output/part-r-00000
```



The screenshot shows a terminal window with the following content:

```
root@hadoopmaster:/usr/local# hadoop dfs -cat output/part-r-00000
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

17/04/09 02:57:09 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
apologising 1
armstrong 3
caliban 1
conyn 3
cuckoo 7
enall 3
hadoop 0
halliday 1
jewelled 2
mother 79
nullgan 115
overclouding 1
project 87
stephen 317
ulysses 7
root@hadoopmaster:/usr/local#
```

## Step 10

The following command is to copy the output folder from the HDFS to local file system to analyze the output.

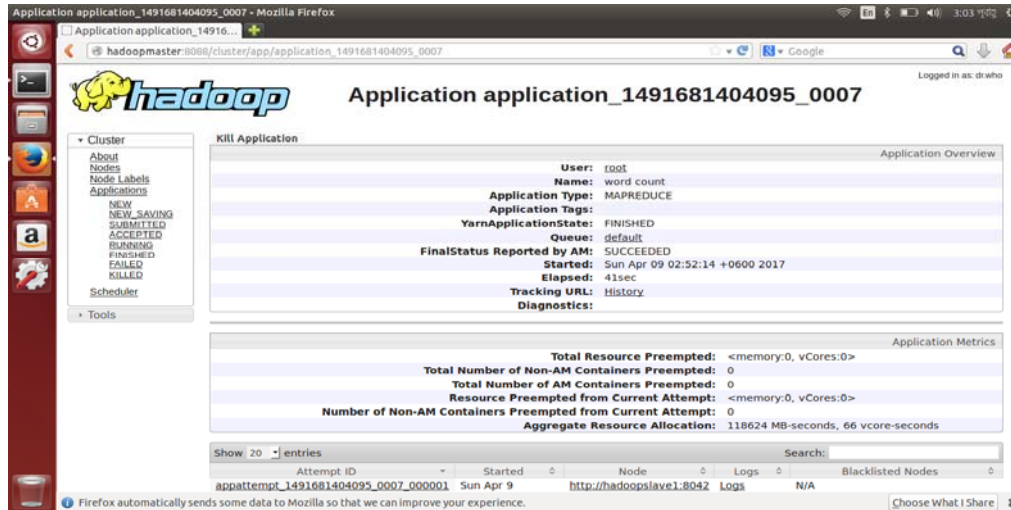
```
root@hadoopmaster:/usr/local# hadoop dfs copyToLocal output
```

Therefore we complete the execution of MapReduce based application on Hadoop Cluster.

Now let us see some web interfaces for running the program.

To see the execution time for processing 268.030 data we need to go to the ResourceManager web interface

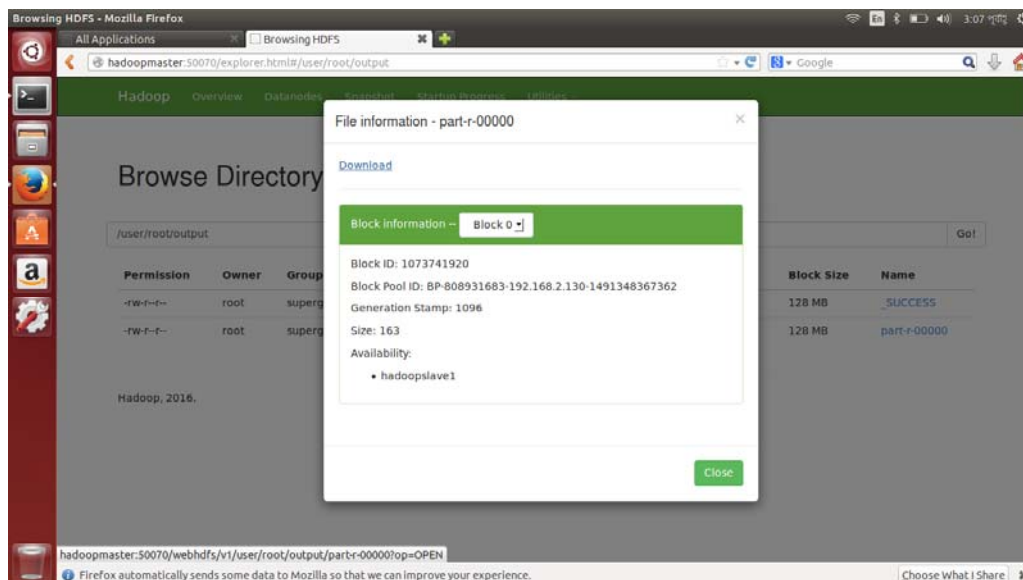
<http://hadoopmaster:8088>



Here elapsed time is 41sec .

We can also browse HDFS web interface to download the output in our local file system from HDFS

<http://hadoopmaster:50070>



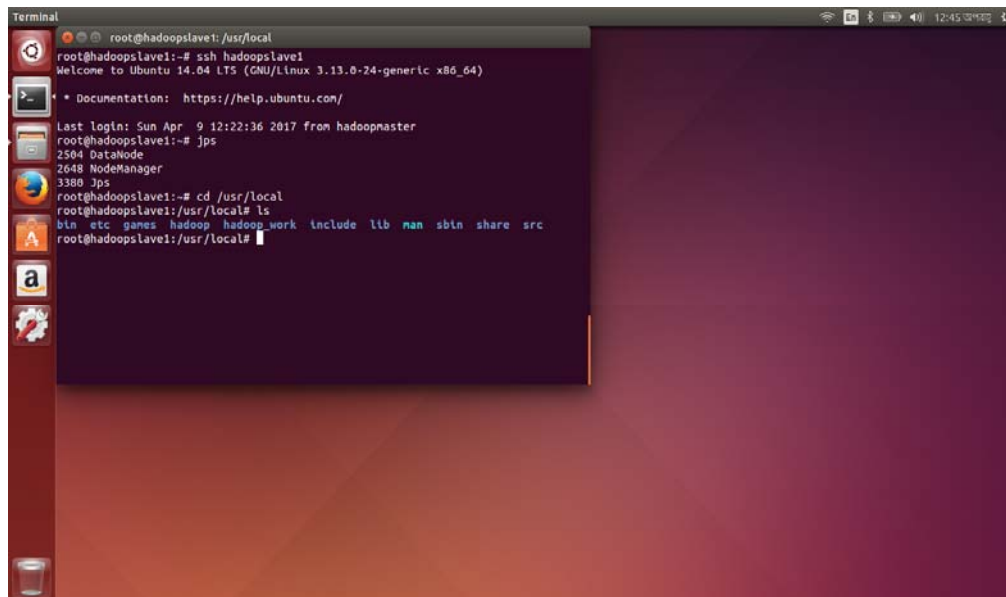
# Chapter 6

## Result & Discussion

In objective we have said that we want to establish 3 things in this implementation.

i. The first one is ssh authentication. We wanted to establish a password-less ssh by sharing public key of master node and slave node. For that first we had to compute public key and then share the keys so that transferring data or starting jobs there occurs no difficulties. Now we want to test whether we could establish remote system through ssh where to communicate with each other there need no password. For that we will give a command from MasterNode to remote access SlaveNode.

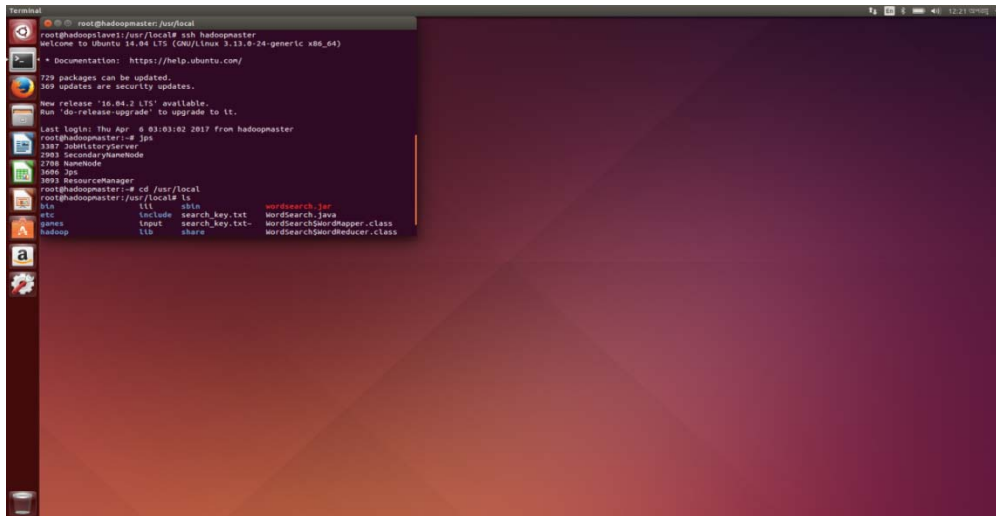
```
root@hadoopmaster: ssh hadoopslave1
```



Here we can see that we got a remote access of hadoopslave1 in hadoopmaster. From hadoopmaster we can see what jobs are running in hadoopslave1 and also we can access other contents of hadoopslave1

Now we will try to remote host hadoopmaster from hadoopslave1 so that we can confirm that we have fulfilled our first object ssh authentication.

root@hadoopslave1: ssh hadoopmaster



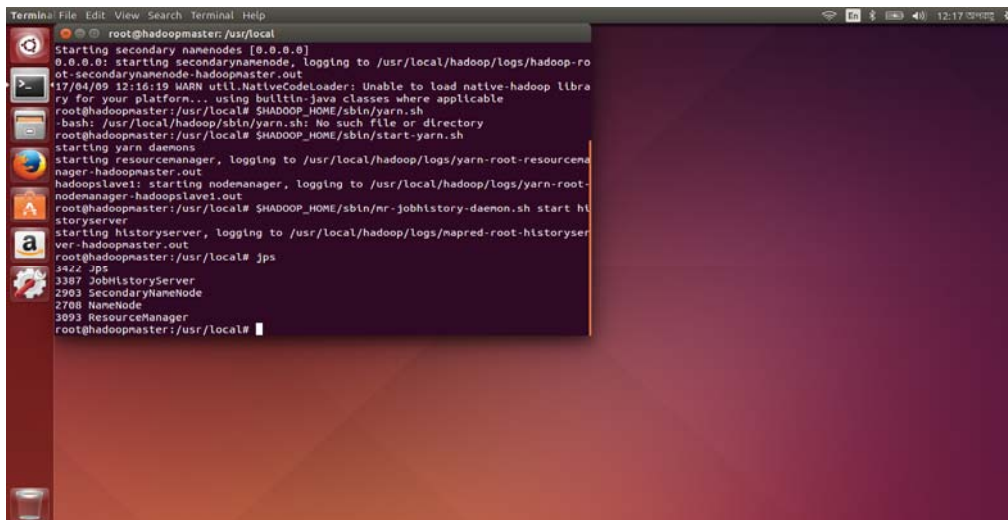
Here we can see that we got a remote access of hadoopmaster from hadoopslave1 without password. Here we can run the jobs that are supposed to run in hadoopmaster. And also we can see other contents of hadoopmaster without any password.

Therefore, our first objective is fulfill of our project.

ii. The second one objective was to implement Hadoop cluster successfully. The hadoop cluster is successfully implemented or not we can check by inspecting the running jobs in Master and SlaveNode . For that we will follow the following command.

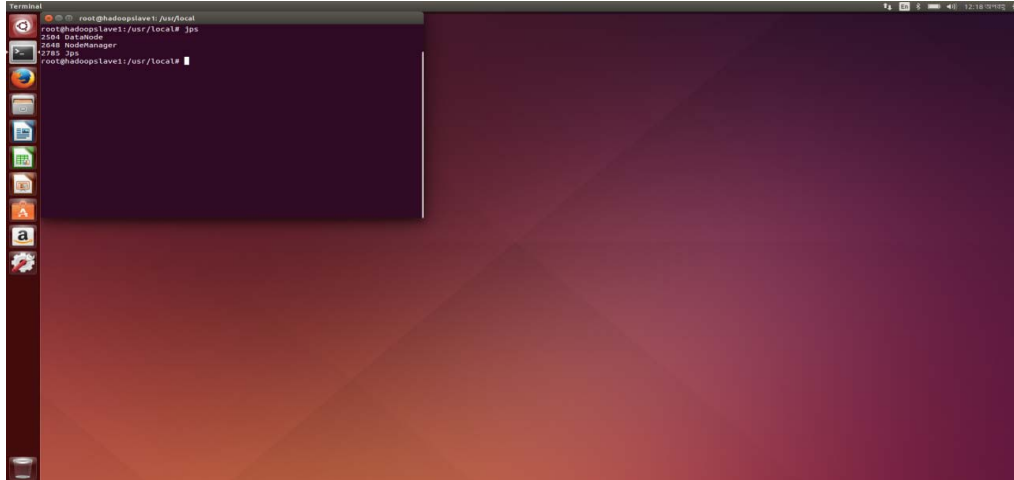
root@hadoopmaster: jps

if NameNode , SecondaryNameNode, ResourceManager, JobHistoryServer are running in hadoopmaster then we can say that hadoopmaster is created successfully.



root@hadoopslave1: jps

If DataNode, NodeManager are running in hadoopslave1 then we can say that hadoopslave1 is successfully created.

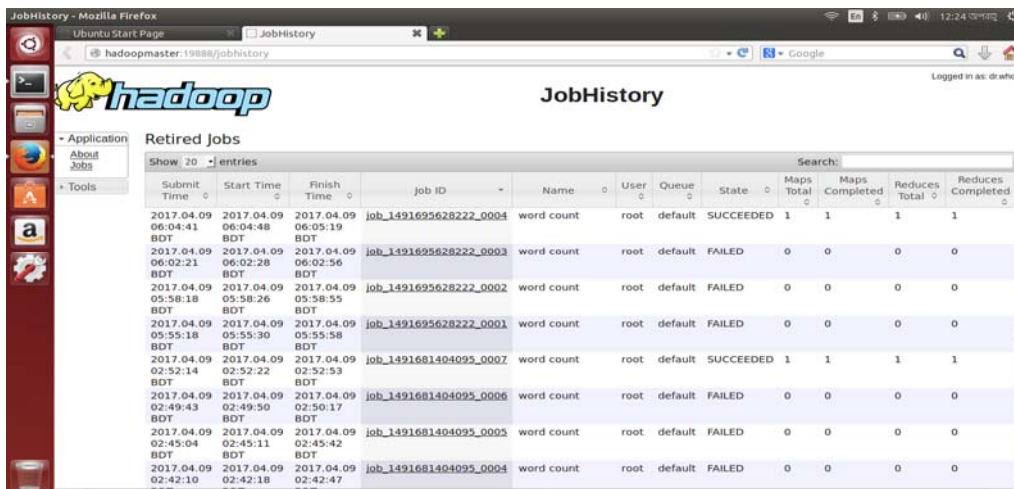


Therefore our second objective of this project is also fulfilled. We have successfully implemented Hadoop cluster.

iii. The third objective was to run a MapReduce based application successfully. To understand the implementation of MapReduce on Hadoop cluster we will provide the web interfaces of Hadoop cluster.

We can see the MapReduce JobHistoryServer web page to identify whether MapReduce Application success or failed.

<http://hadoopmaster:8088>



Here we can see that MapReduce Application (Word Count ) SUCCEEDED at the above.



Now we want to confirm if the slave node is running or not through web browser. For that we need to browse the YARN ResourceManager.

<http://hadoopmaster:8088>

The screenshot shows the 'Nodes of the cluster' page in a Mozilla Firefox browser. The page title is 'Nodes of the cluster' and the URL is 'http://hadoopmaster:8088/cluster/nodes'. The page displays the Hadoop logo and a 'Nodes of the cluster' section. Under 'Cluster Metrics', there are several tables. The first table shows 'Cluster Metrics' with columns for Apps Submitted, Pending, Running, Completed, Containers Running, Memory Used, Total, Reserved, VCores Used, Total, Reserved, Active Nodes, Decommissioned Nodes, Lost Nodes, and Unhealthy Nodes. The second table shows 'Scheduler Metrics' with columns for Scheduler Type, Scheduling Resource Type, Minimum Allocation, and Maximum Allocation. The third table shows a list of nodes with columns for Node Labels, Rack, Node State, Node Address, Node HTTP Address, Last health-update, Health-report, Containers, Mem Used, Mem Avail, VCores Used, and VCores Avail. The table shows one node in the 'RUNNING' state with Node Address 'hadoopslave1:57087' and Node HTTP Address 'hadoopslave1:8052'. The last health-update is 'Sun Apr 09 12:28:49 +0600 2017'. The page also shows 'Showing 1 to 1 of 1 entries' and navigation links for 'First', 'Previous', and 'Next'.

Here you can see hadoopslave1 Node is RUNNING .

Now to confirm the hadoopmaster node is running through web page we need to browse the HDFS webpage.

<http://hadoopmaster:50070>

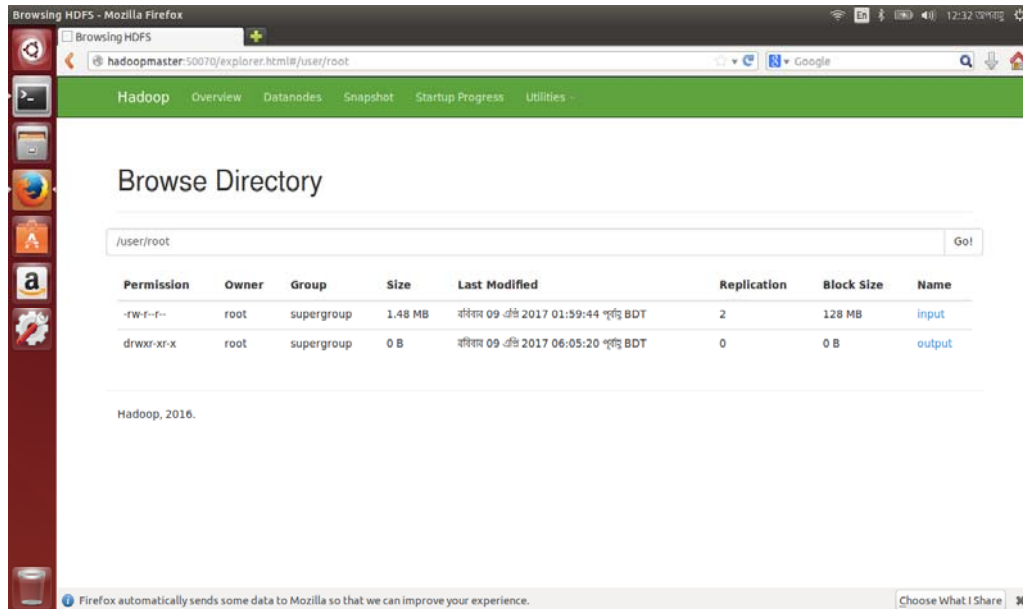
The screenshot shows the 'NameNode Information' page in a Mozilla Firefox browser. The page title is 'NameNode Information' and the URL is 'http://hadoopmaster:50070/dfs/health.html#tab-overview'. The page displays the Hadoop logo and a 'NameNode Information' section. Under 'Overview', there is a table with columns for 'Started:', 'Version:', 'Complied:', 'Cluster ID:', and 'Block Pool ID:'. The table shows the following information: 'Started: Sun Apr 09 12:15:49 BDT 2017', 'Version: 2.7.3, rbaa91f7c6bc9cb92be5982de4719c1c8af91ccff', 'Complied: 2016-08-18T01:41Z by root from branch-2.7.3', 'Cluster ID: CID-46b81bd5-e92a-4a1c-bf35-8254d468ba04', and 'Block Pool ID: BP-808931683-192.168.2.130-1491348367362'. Below the table, there is a 'Summary' section with the following information: 'Security is off.', 'Safemode is off.', '71 files and directories, 35 blocks = 106 total filesystem object(s).', 'Heap Memory used 72.02 MB of 127 MB Heap Memory. Max Heap Memory is 889 MB.', and 'Non Heap Memory used 31.93 MB of 33.44 MB Committed Non Heap Memory. Max Non Heap Memory is 130 MB.' The page also shows 'Firefox automatically sends some data to Mozilla so that we can improve your experience.' and a 'Choose What I Share' button.

Here you can see hadoopmaster node is active.

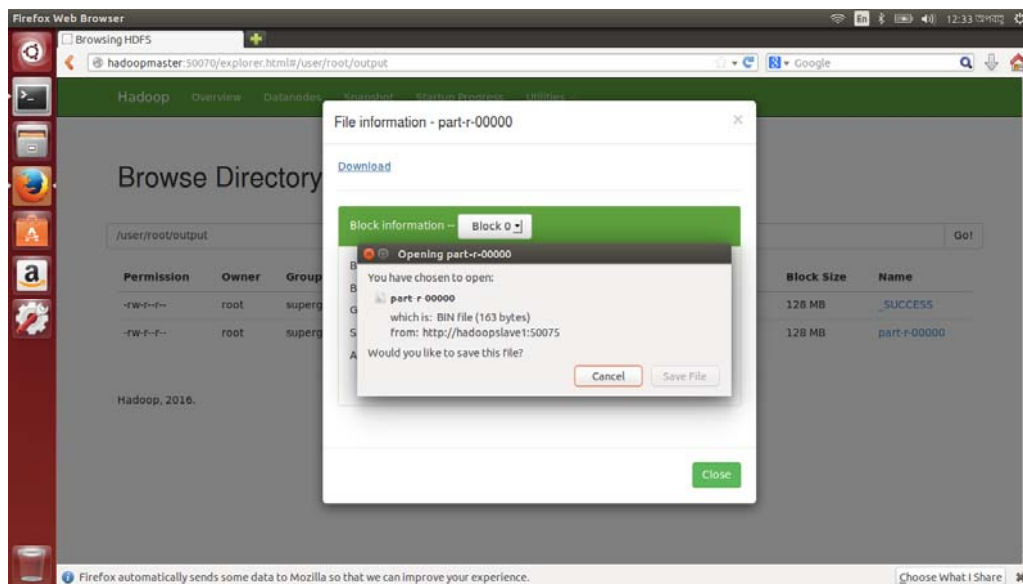


Finally we want to see if the output is created in the HDFS through web browser. For that we need to browse the HDFS webpage

<http://hadoopmaster:50070>



Now want to see the content of output and also we want to download the output/part-r-00000 file.



Therefore here we proved that our third objective of this project is successful. So we fulfill our three objective of this project.

# Chapter 7

## Conclusion & Future Work

### 7.1 Conclusion

Data are increasing day by day. For processing this large amount of data in low cost and less time, Hadoop cluster is an effective platform. We have implemented Hadoop cluster in a very simpler way. We have also presented MapReduce based application in Hadoop Cluster for processing data as MapReduce gives privileges to work faster and distributed way. We have monitored successful implementation of Hadoop cluster by web interfaces. So, our goal was to give a clear perspective about Hadoop cluster along with MapReduce. Now-a-days many organizations are hiring Hadoop developer for processing large amount of data. So for stepping forward in Hadoop field this project work will be very helpful to understand the basic of Hadoop and the implementation of Hadoop cluster.

### 7.2 Future Work

We have implemented a small Hadoop cluster which consists of one master and one slave. In future we want to implement Hadoop cluster among one master and multiple slaves. As the master node RAM is 4 GB, in future we want to create master node at least with 8 GB RAM. We have implemented existing MapReduce application by modifying to understand the mechanism of MapReduce. But in near future we want to implement our own invention MapReduce application to understand the mechanism of MapReduce. For that we have a great wish to implement Adaptive Merge Sort on Hadoop Cluster as MapReduce application. We have chosen Adaptive Merge Sort as an application of MapReduce because in Adaptive Merge Sort there is a partitioning step which can describe the MapReduce mechanism very nicely.

## Reference

- [1] What is Computer Cluster? - Definition from Techopedia. (2017). Techopedia.com. Retrieved 22 April 2017, from <https://www.techopedia.com/definition/6581/computer-cluster>
- [2] HDFS Architecture Guide . (2017). Hadoop.apache.org. Retrieved 22 April 2017, from [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)
- [3] NameNode - Hadoop Wiki. (2017). Wiki.apache.org. Retrieved 22 April 2017, from <https://wiki.apache.org/hadoop/NameNode>
- [4] DataNode - Hadoop Wiki. (2017). Wiki.apache.org. Retrieved 22 April 2017, from <https://wiki.apache.org/hadoop/DataNode>
- [5] An Automation Tool for Single-node and Multi-node Hadoop Cluster. (2017). Scialert.net. Retrieved 9 April 2017, from <http://scialert.net/fulltext/?doi=jai.2013.82.88>
- [6] Hadoop clusters: Benefits and challenges for big data analytics. (2017). SearchStorage. Retrieved 9 April 2017, from <http://searchstorage.techtarget.com/tip/Hadoop-clusters-Benefits-and-challenges-for-big-data-analytics>
- [7] Hadoop - Multi Node Cluster. (2017). www.tutorialspoint.com. Retrieved 22 April 2017, from [https://www.tutorialspoint.com/hadoop/hadoop\\_multi\\_node\\_cluster.html](https://www.tutorialspoint.com/hadoop/hadoop_multi_node_cluster.html)
- [8] Hedlund, B. (2011). Understanding Hadoop Clusters and the Network. Bradhedlund.com. Retrieved 9 April 2017, from <http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/>
- [9] device, H., speed, D., switch, J., Linux, H., array, H., & data, H. (2016). Why does hadoop require SSH passwordless access - Fibrevillage. Fibrevillage.com. Retrieved 9 April 2017, from <http://fibrevillage.com/storage/627-why-hadoop-require-ssh-passwordless-accesshadoop-installation>
- [10] Official, D. (2016). How to Setup Hadoop Multi Node Cluster - Step By Step. Dwbi.org. Retrieved 9 April 2017, from <https://dwbi.org/etl/bigdata/183-setup-hadoop-cluster>
- [11] Apache Hadoop 2.7.3 – MapReduce Tutorial. (2017). Hadoop.apache.org. Retrieved 9 April 2017, from <https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>
- [12] Hadoop MapReduce. (2017). www.tutorialspoint.com. Retrieved 22 April 2017, from [https://www.tutorialspoint.com/hadoop/hadoop\\_mapreduce.html](https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.html)
- [13] Hadoop Tutorial 1 -- Running WordCount - DftWiki. (2017). Cs.smith.edu. Retrieved 22 April 2017, from [http://cs.smith.edu/dftwiki/index.php/Hadoop\\_Tutorial\\_1\\_--\\_Running\\_WordCount](http://cs.smith.edu/dftwiki/index.php/Hadoop_Tutorial_1_--_Running_WordCount)