# **Department of Computer Engineering**

**Academic Term: July-November 2023**

# **Rubrics for Lab Experiments**

| **Class** | **: B*.E. Computer*** | **Subject Name: *Distributed Computing*** |
| --- | --- | --- |
| **Semester** | **: VIII** | **Subject Code :** ***CSC801*** |

| **Practical No:** | 10 |
| --- | --- |
| **Title:** | To Study HDFS and MapReduce |
| **Date of Performance:** | 10/4/24 |
| **Roll No:** | 9209 |
| **Name of the Student:** | Rohan Mathew |

**Evaluation:**

| **Performance Indicator** | **Below average** | **Average** | **Good** | **Excellent** | **Marks** |
| --- | --- | --- | --- | --- | --- |
| **On time Submission (2)** | Not submitted(0) | Submitted after deadline (1) | Early or on time submission(2) | --- |  |
| **Test cases and output**  **(4)** | Incorrect output (1) | The expected output is  verified only a for few test  cases (2) | The expected output is Verified for all test cases but is  not presentable (3) | Expected output is obtained for all test cases. Presentable and easy to follow (4) |  |
| **Coding**  **efficiency (2)** | The code is not  structured at all (0) | The code is  structured but not efficient (1) | The code is  structured and efficient. (2) | - |  |
| **Knowledge(2)** | Basic concepts not clear  (0) | Understood the basic concepts (1) | Could explain the concept with  suitable example  (1.5) | Could relate the theory with real world  application(2) |  |
| **Total** |  | | | | |

**Signature of the Teacher :**

**Lab 10**

**Aim:** To Study HDFS and MapReduce

**Lab Outcome:**

Describe the concepts of distributed File Systems with some case studies **Theory:**

Hadoop:

With growing data velocity, the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called *distributed filesystems*. Since data is stored across a network all the complications of a network come in.

This is where Hadoop comes in. It provides one of the most reliable filesystems. HDFS (Hadoop Distributed File System) is a unique design that provides storage for *extremely large files* with streaming data access pattern and it runs on *commodity hardware*.

Let’s elaborate the terms:

• ***Extremely large files***: Here we are talking about the data in range of petabytes (1000 TB).

• ***Streaming Data Access Pattern***: HDFS is designed on principle of *write-once and read-many-times*. Once data is written large portions of dataset can be processed any number times.

• ***Commodity hardware:*** Hardware that is inexpensive and easily available in the market. This is one of feature which specially distinguishes HDFS from other file system.

**Nodes:** Master-slave nodes typically forms the HDFS cluster.

1. **NameNode(MasterNode):**

• Manages all the slave nodes and assign work to them.

• It executes filesystem namespace operations like opening, closing, renaming files and directories.

• It should be deployed on reliable hardware which has the high

config. not on commodity hardware.

2. **DataNode(SlaveNode):**

• Actual worker nodes, who do the actual work like reading, writing, processing etc.

• They also perform creation, deletion, and replication upon

instruction from the master.

• They can be deployed on commodity hardware.

**HDFS daemons:** Daemons are the processes running in background. • **Namenodes:**

• Run on the master node.

• Store metadata (data about data) like file path, the number of

blocks, block Ids. etc.

• Require high amount of RAM.

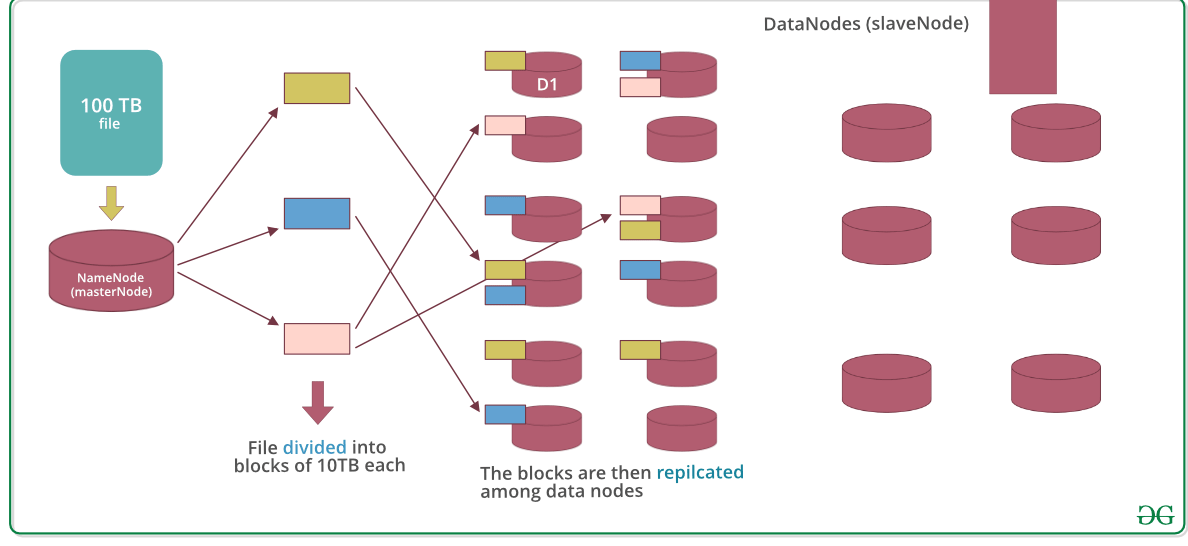
• Store meta-data in RAM for fast retrieval i.e to reduce seek time. Though a persistent copy of it is kept on disk.

• **DataNodes:**

• Run on slave nodes.

• Require high memory as data is actually stored here.

**Data storage in HDFS**: Now let us see how the data is stored in a distributed manner.



Assuming that 100TB file is inserted, then masternode(namenode) will first divide the file into blocks of 10TB (default size is 128 MB in Hadoop 2.x and above). Then these blocks are stored across different datanodes(slavenode). Datanodes(slavenode)replicate the blocks among themselves and the information of what blocks they contain is sent to the master. Default replication factor is 3 means for each block 3 replicas are created (including itself). In hdfs.site.xml we can increase or decrease the replication factor i.e we can edit its configuration here.

Terms related to HDFS: • HeartBeat: It is the signal that datanode continuously sends to namenode. If namenode doesn’t receive heartbeat from a datanode then it will consider it dead. • Balancing: If a datanode is crashed the blocks present on it will be gone too and the blocks will be under-replicated compared to the remaining blocks. Here master node(namenode) will give a signal to datanodes containing replicas of those lost blocks to replicate so that overall distribution of blocks is balanced. • Replication: It is done by datanode.

Features: • Distributed data storage. • Blocks reduce seek time. • The data is highly available as the same block is present at multiple datanodes. • Even if multiple datanodes are down we can still do our work, thus making it highly reliable. • High fault tolerance. MapReduce: 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. MapReduce Architecture explained in detail: • One map task is created for each split which then executes map function for each record in the split.

• It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel. • However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time. • For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default). • Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS. • Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation. • Map output is intermediate output which is processed by reduce tasks to produce the final output. • Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill. • In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output. • Reduce task doesn’t work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running. • On this machine, the output is merged and then passed to the user-defined reduce function. • Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output How MapReduce Organizes Work? Hadoop divides the job into tasks. There are two types of tasks: • Map tasks (Splits & Mapping) • Reduce tasks (Shuffling, Reducing) The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a • Jobtracker: Acts like a master (responsible for complete execution of submitted job) • Multiple Task Trackers: Acts like slaves, each of them performing the job For every job submitted for execution in the system, there is one Jobtracker that resides on Namenode and there are multiple tasktrackers which reside on Datanode.

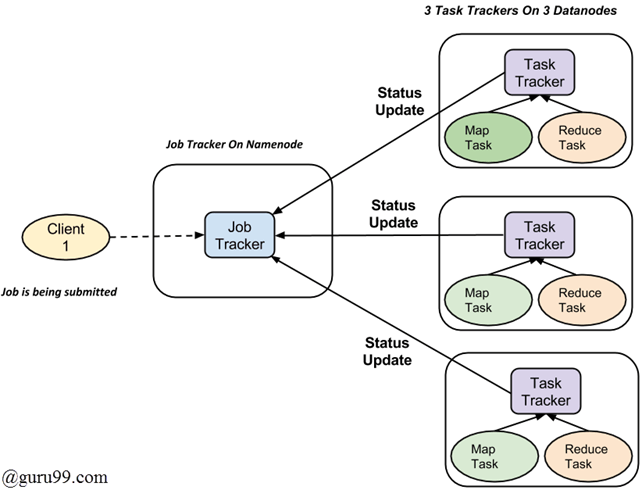


Fig. How Hadoop Mapreduce Works • A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster. • It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes. • Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job. • Task tracker’s responsibility is to send the progress report to the job tracker. • In addition, task tracker periodically sends ‘heartbeat’ signal to the Jobtracker so as to notify him of the current state of the system. • Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Implementation of Hadoop Cluster and HDFS:

To demonstrate the working and configuration of HDFS, a cluster consisting of 1 Master and 2 Slaves is configured. The machine nodes are independent AWS ec2 instances. For ease of operation a tool called MobaXterm is used for remote login (SSH) into the instances.

After successful configuration, the daemons are started in both master and slave nodes. Subsequently, our HDFS can be tested by storing files.

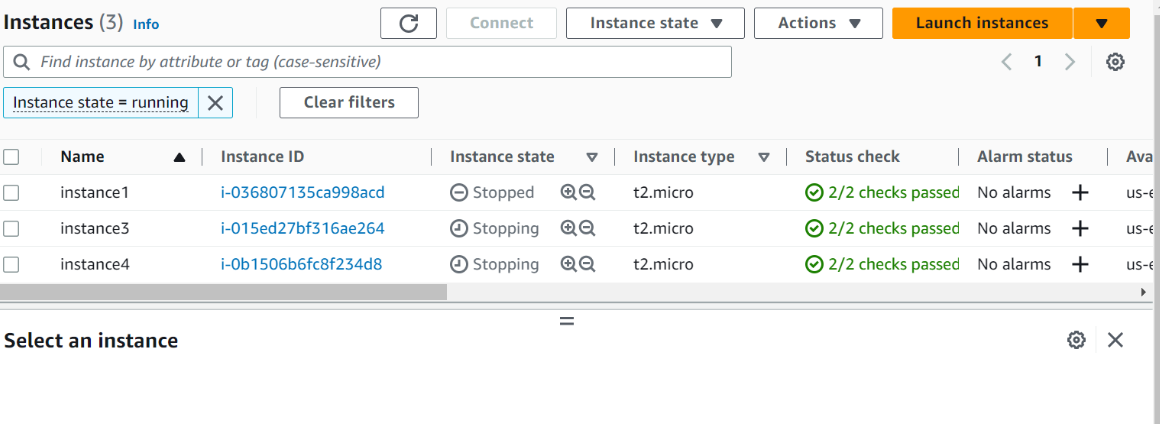
Step 1: Setting up Remote Machines

Three ec2 instances having Ubuntu as local OS were created using AWS Free Tier account.

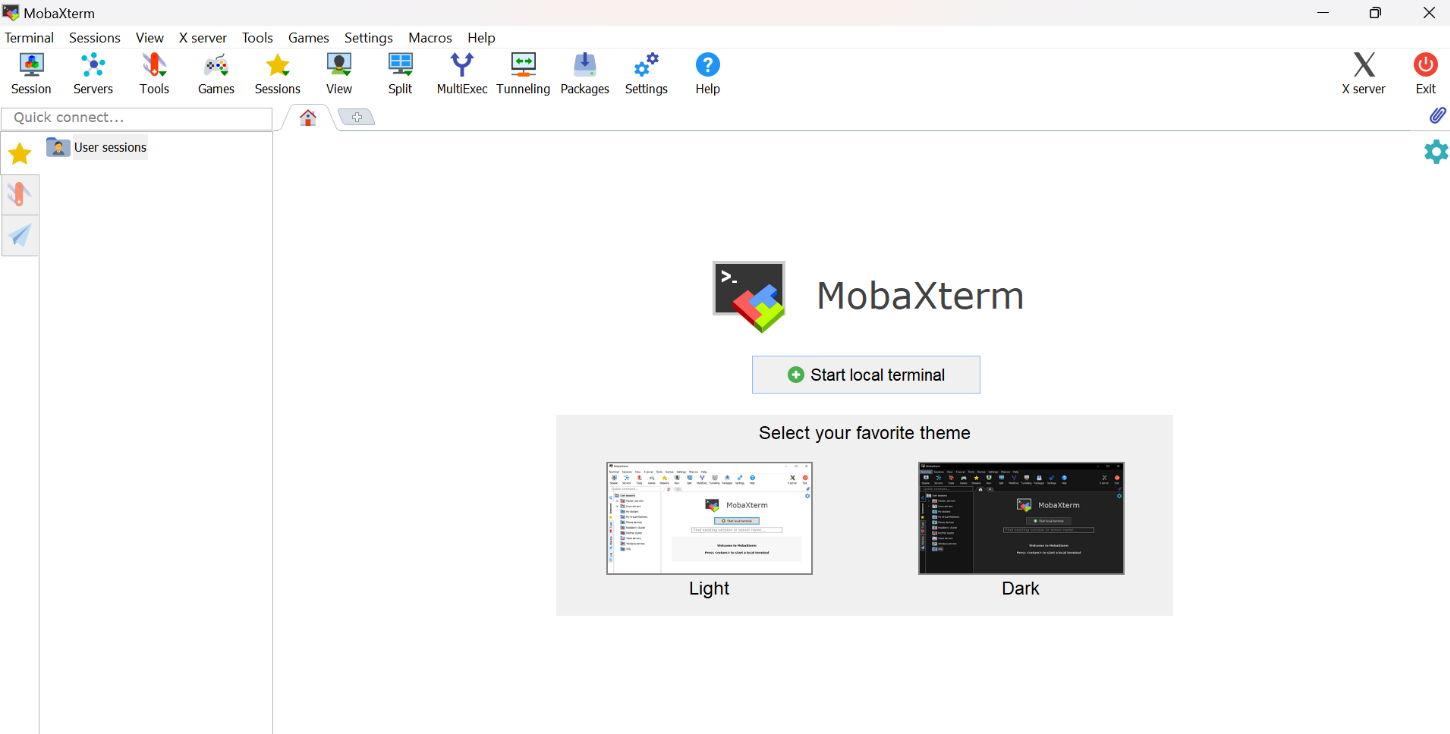
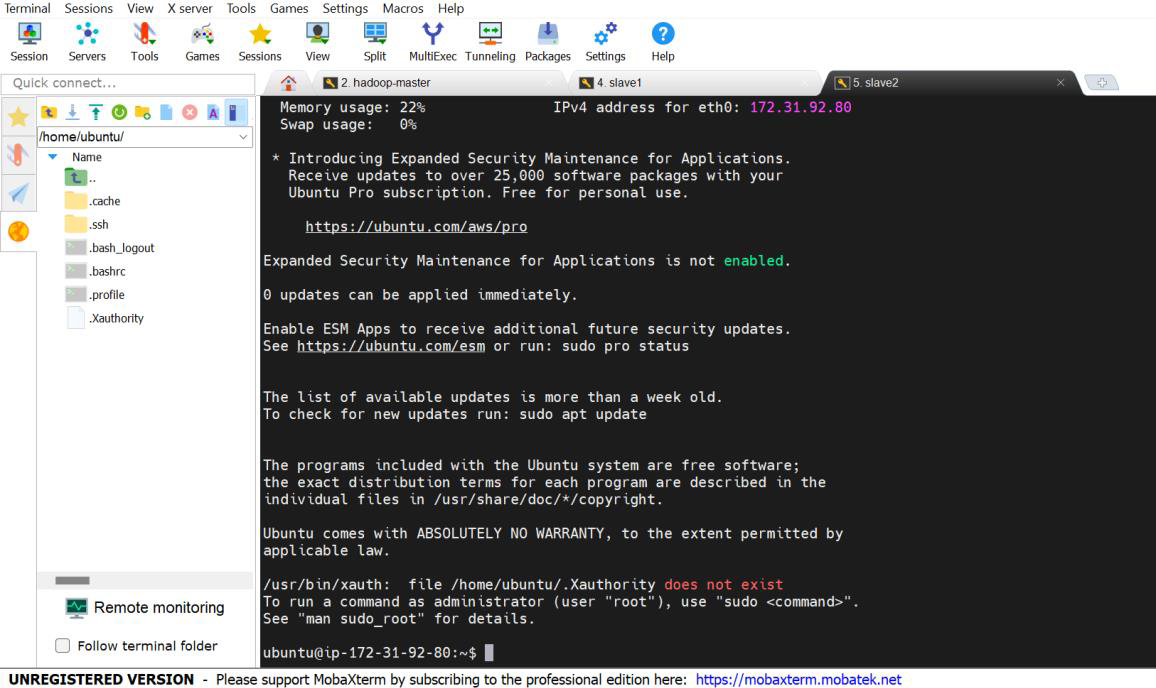
Instance1 -> Master Private IP: 172.31.87.194

Instance3 -> Slave1 Private IP: 172.31.92.80

Instance4 -> Slave2 Private IP: 172.31.28.27

Step 2: Setting up SSH Client

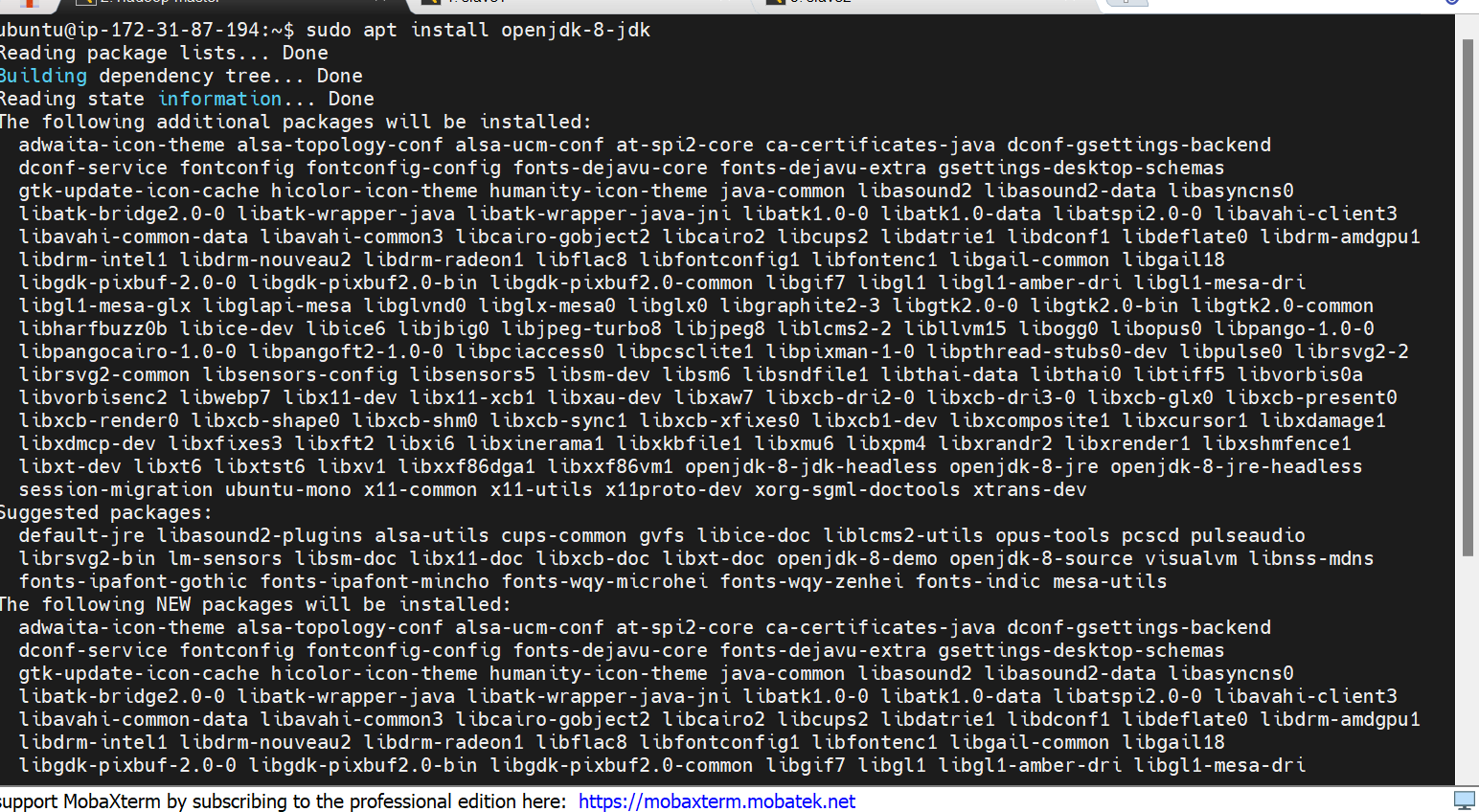
MobaXterm is used an SSH Client to login to remote machines

After connecting to all our nodes, the environment looks like this. SSH connections to our remote machines are lined up as tabsStep 3: Configuration of Hadoop Cluster

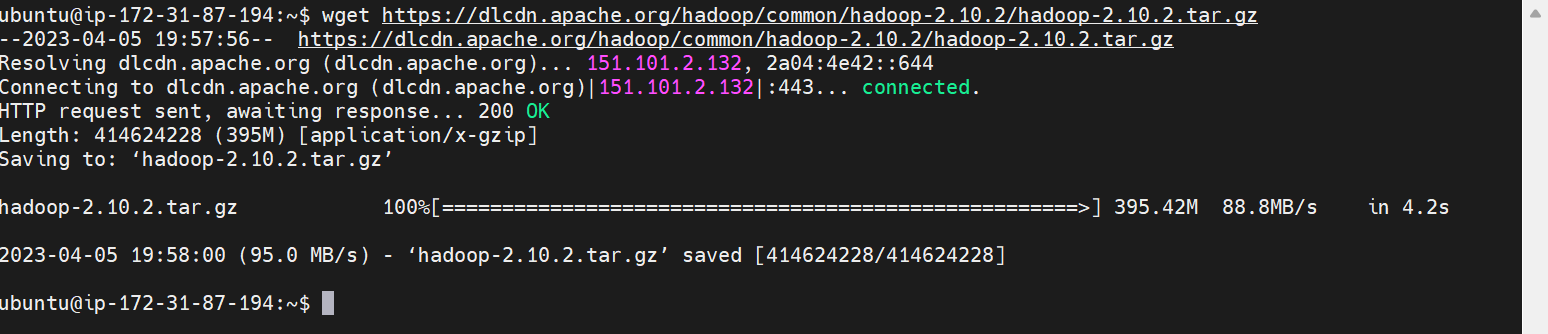
Step 3.1: Download Hadoop and JDK in all three machines

The step is repeated for all the 3 nodes

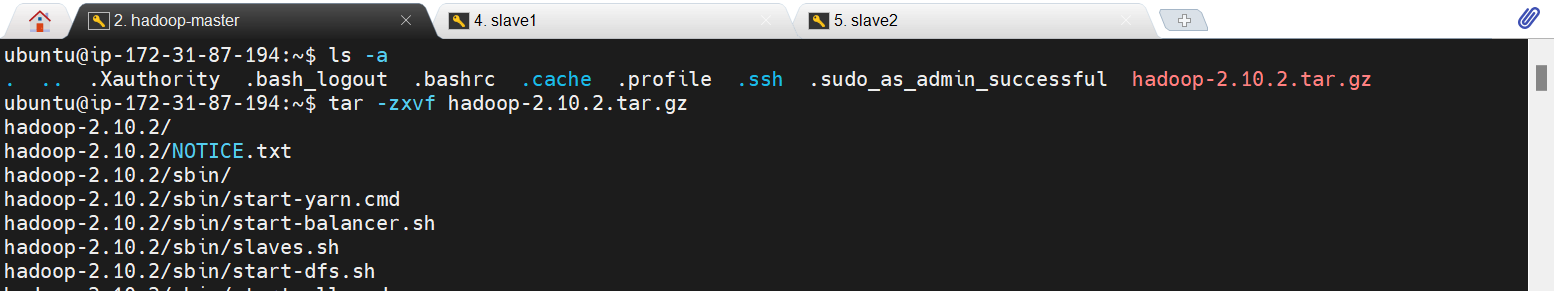
Download jdk-8



Download Hadoop



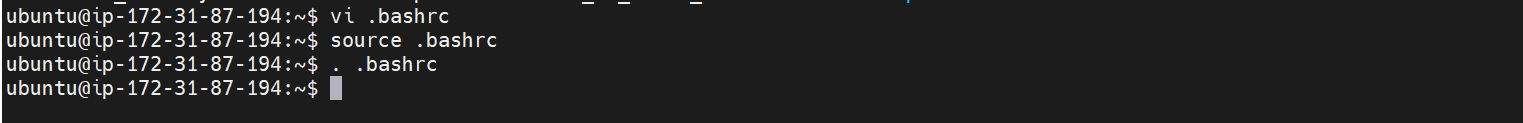
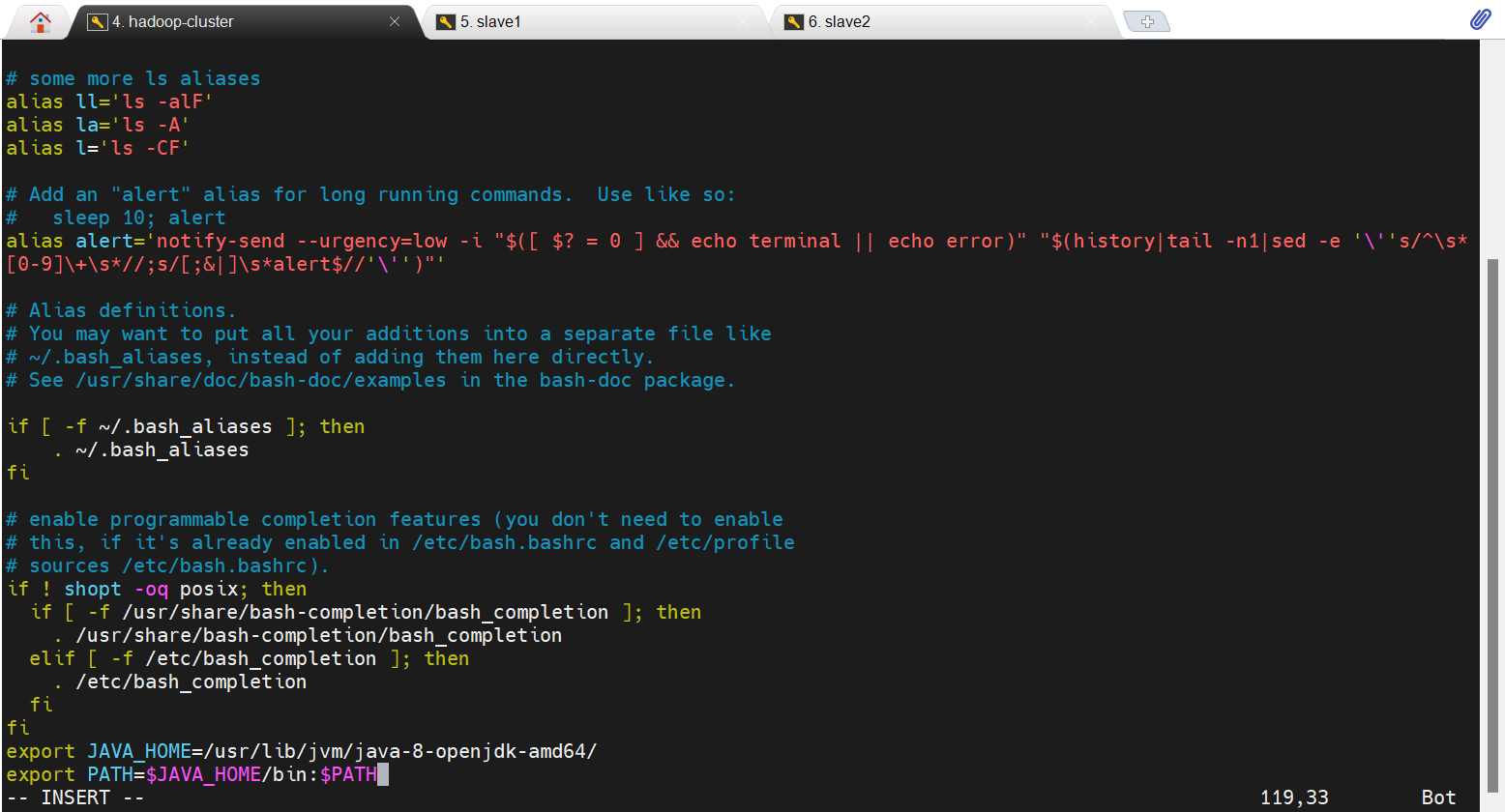
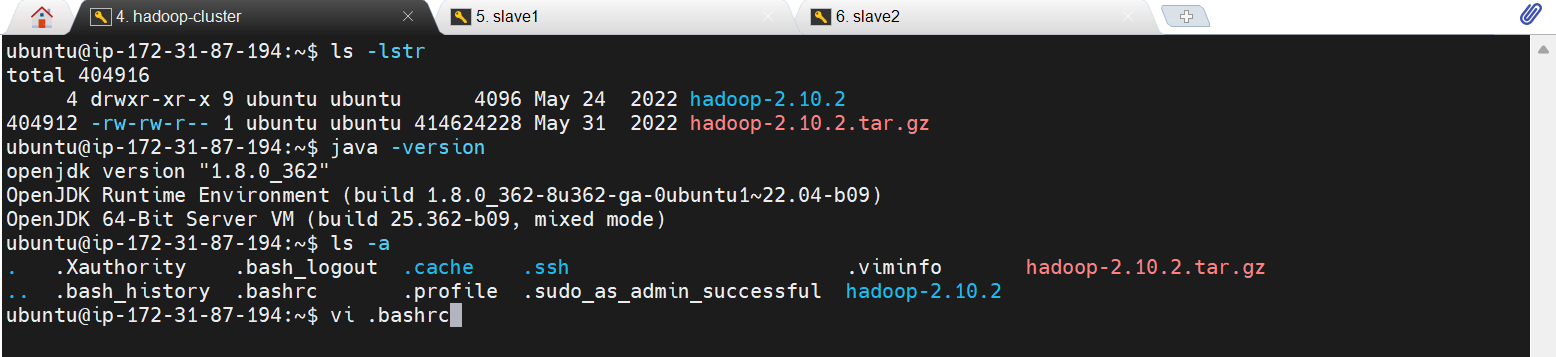
Extracting Hadoop

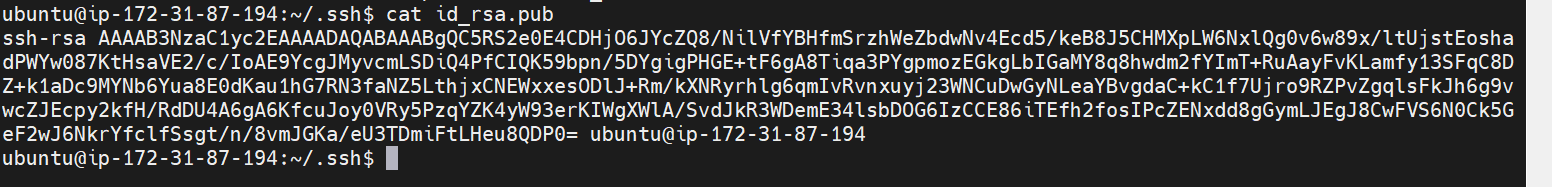
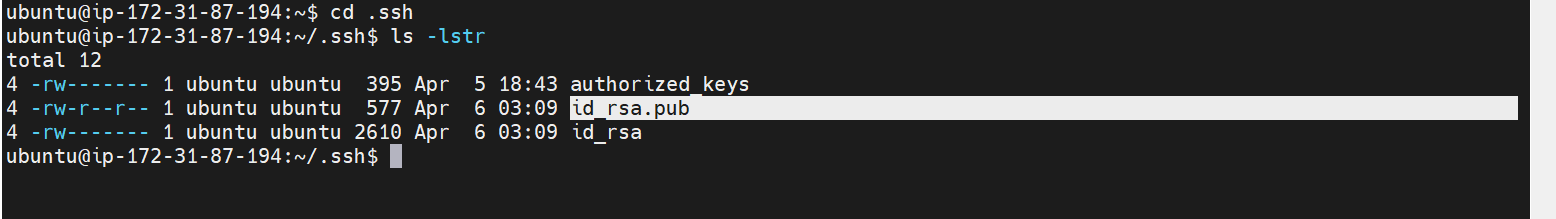
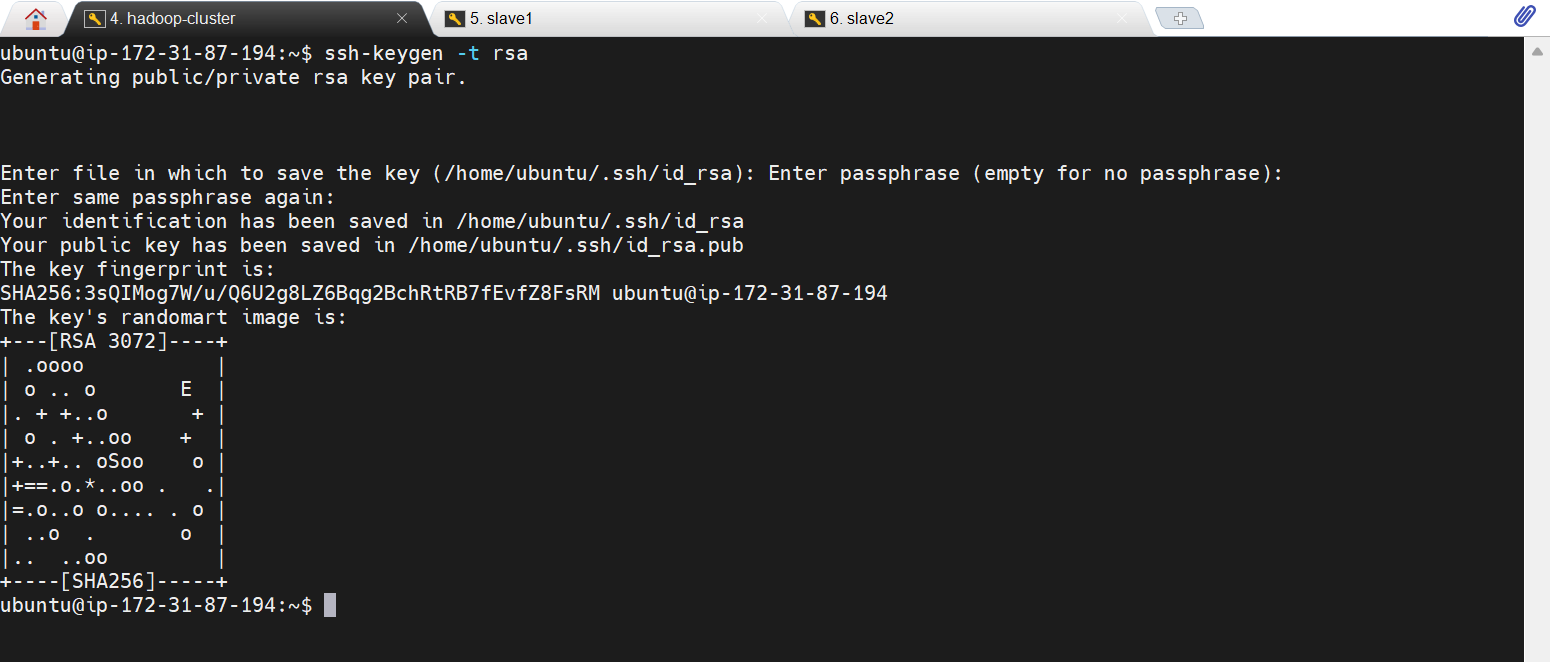


Step 3.2: Environment Setup for java

This step is repeated for all three nodes

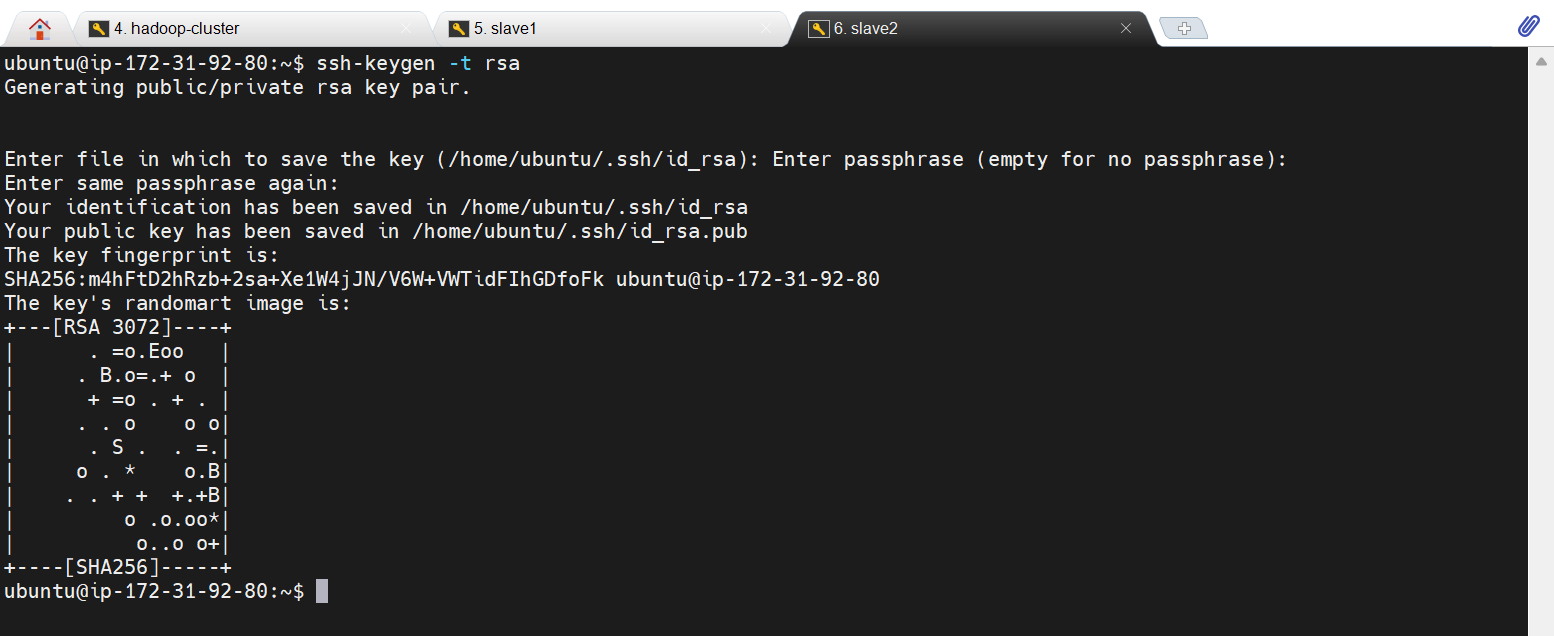
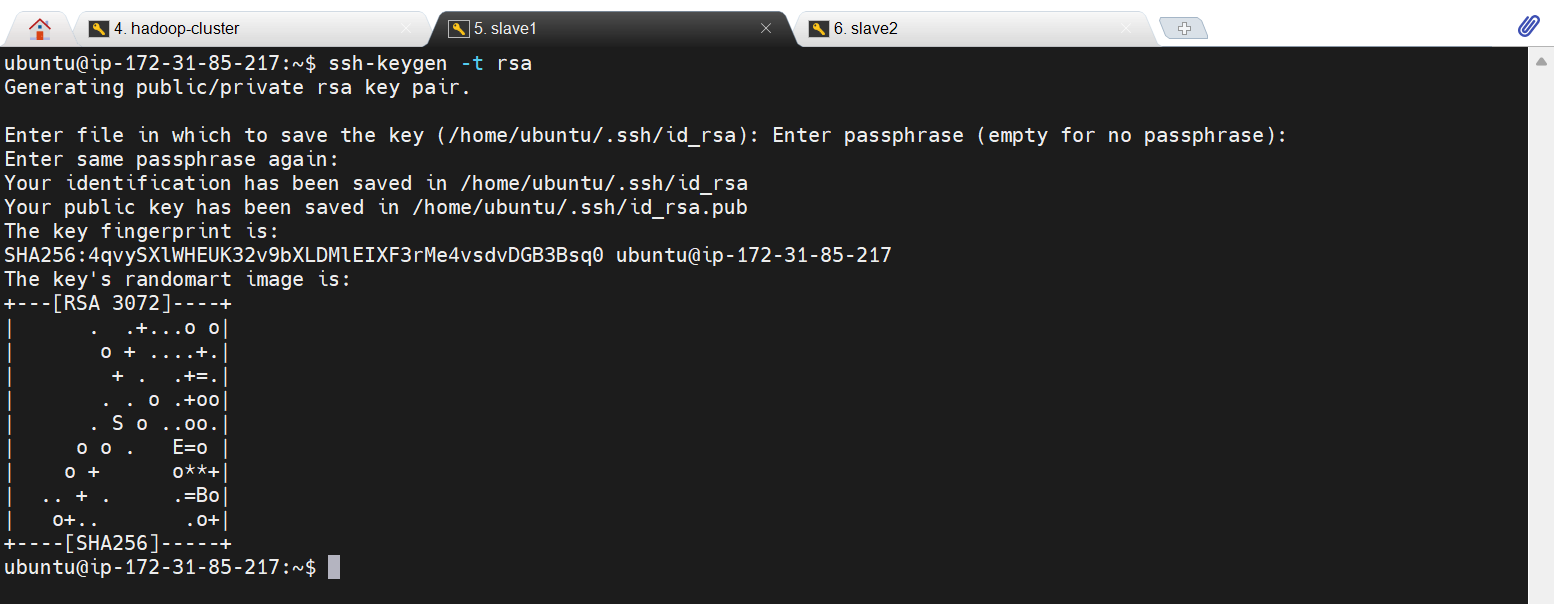
Adding JDK Path in .bashrc file and executing it

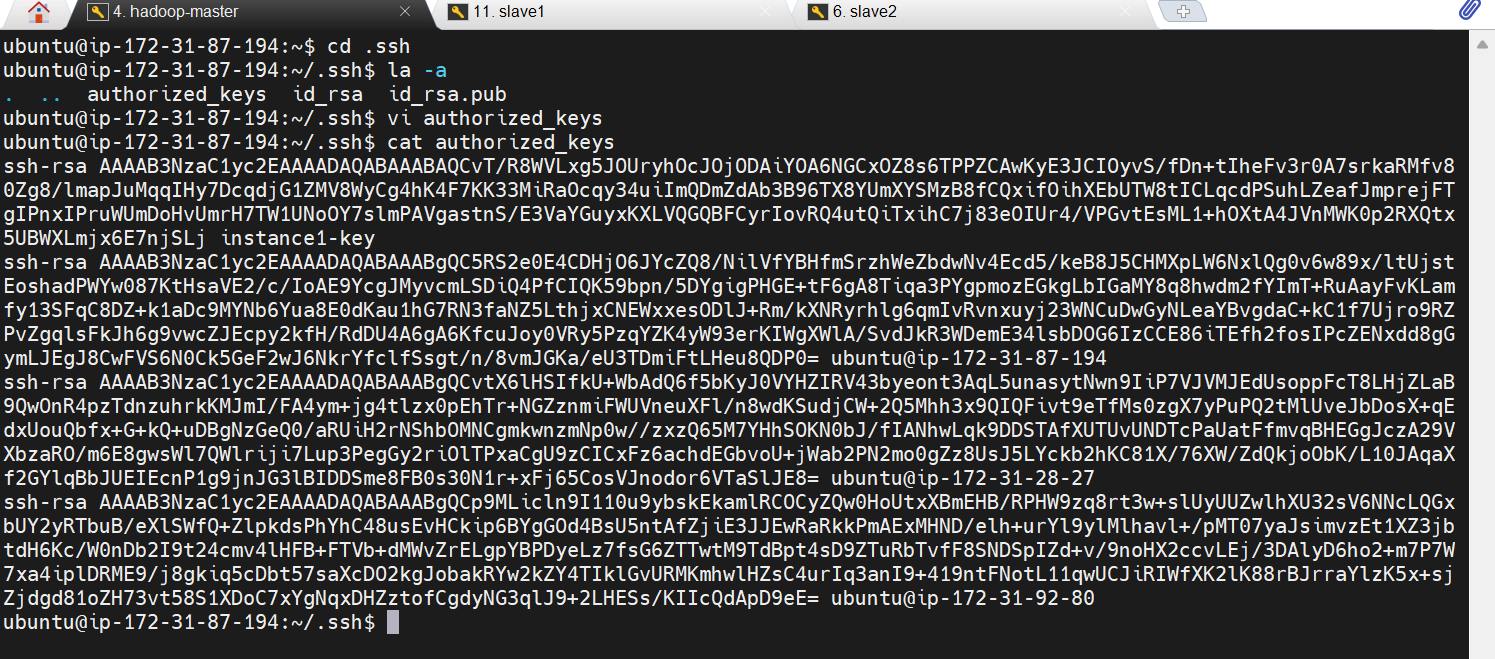


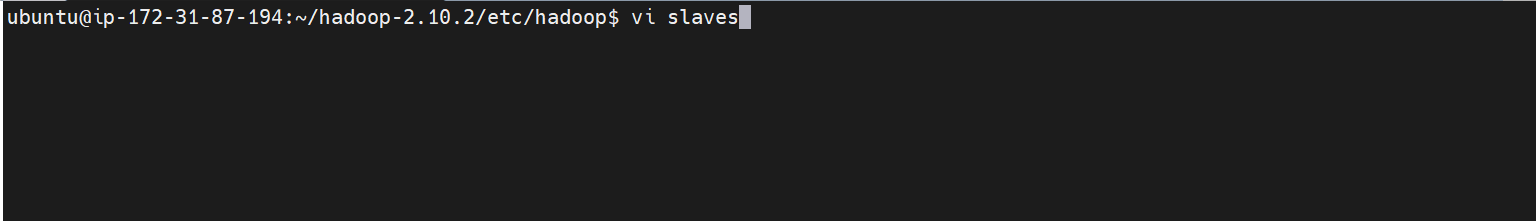
Step 3.3: SSH Setup Between Master and Slaves

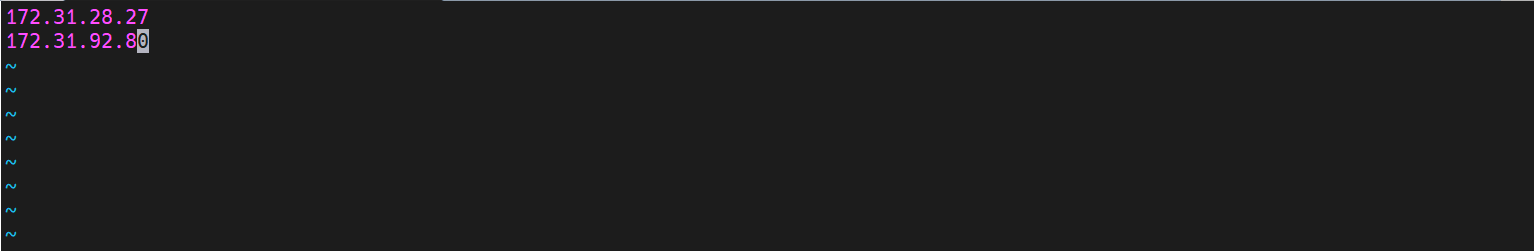
• Generate Key Pairs on each node using: ssh-keygen -t rsa. The generated pairs are stored in id\_rsa.pub in .ssh folder

• Key Pairs from all the nodes are copied together and pasted in authorized\_keys file present in .ssh folder at every node.

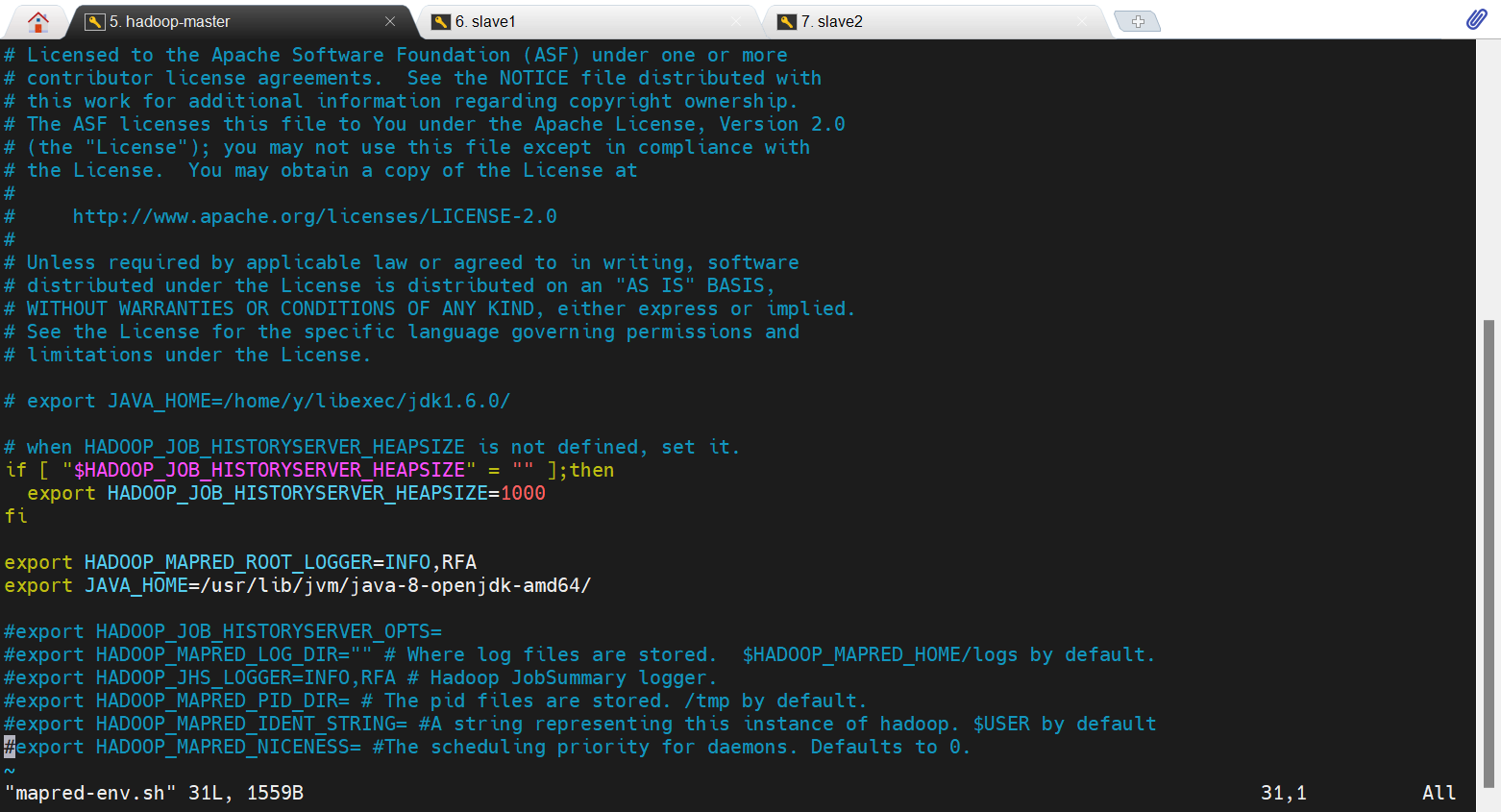
Key Pair generation at slave nodes

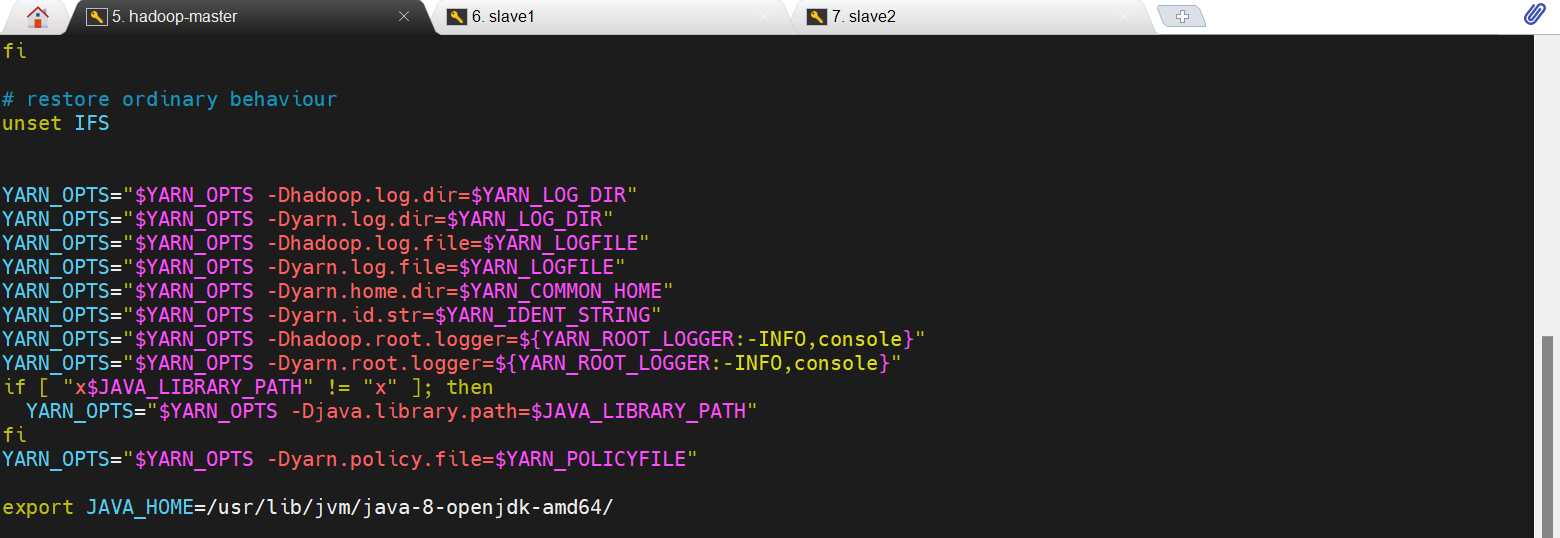
Final authorized\_keys file: This file needs to be maintained at every node

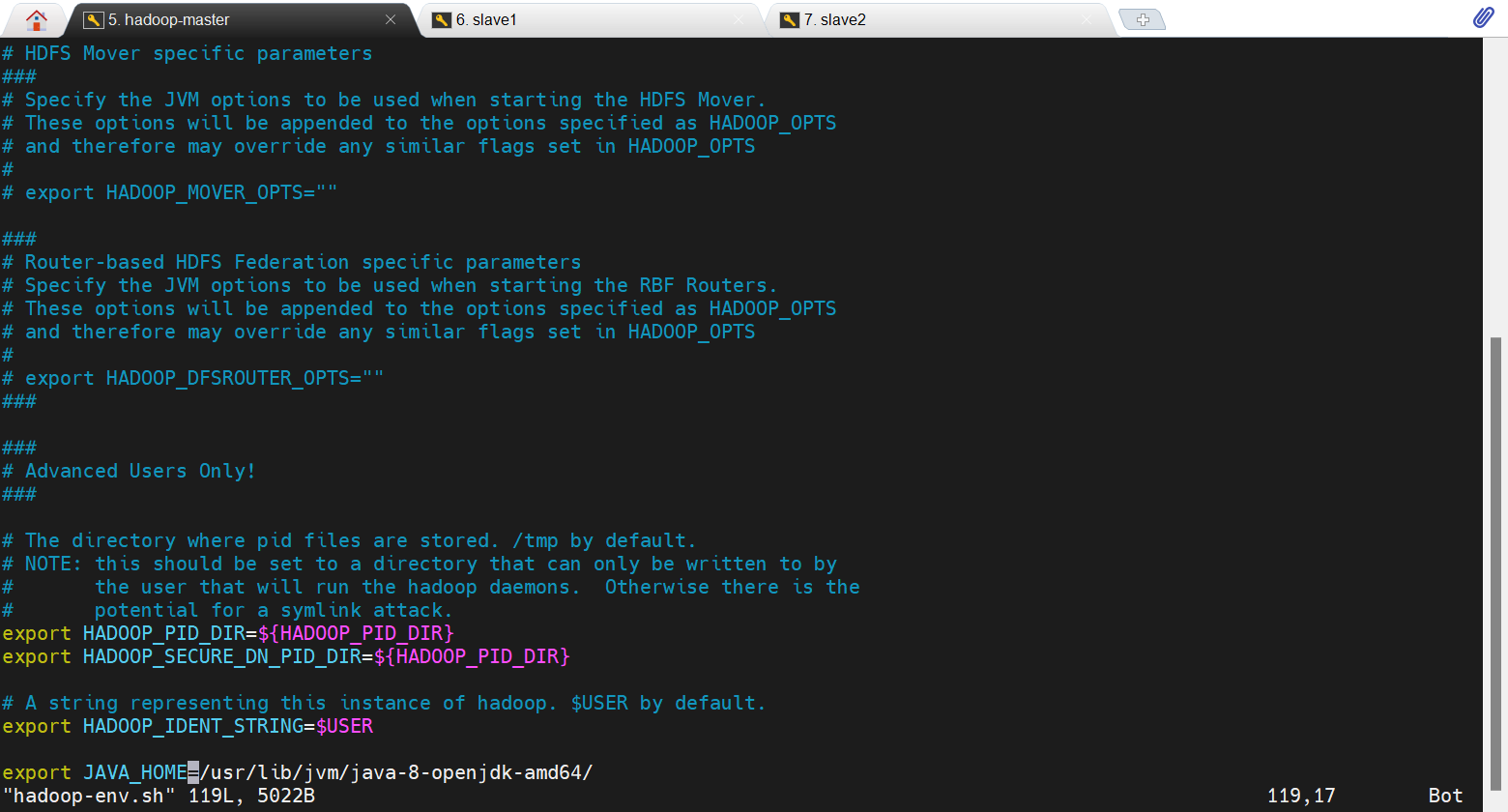
Step 3.4: Configuration of Hadoop Files. Files present in path: hadoop-2.10.2/etc/Hadoop

Configuration of slaves files (Only at Master Node): Add Private IPs of Slaves in slaves file present in master node

Configuration of env files (Needs to be repeated for every node): Adding JDK path to each file

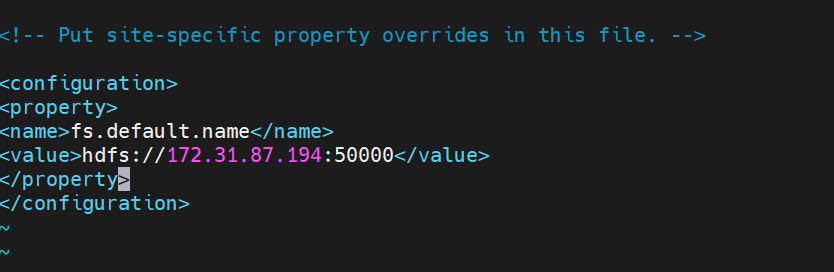
mapred-env.sh

yarn-env.sh

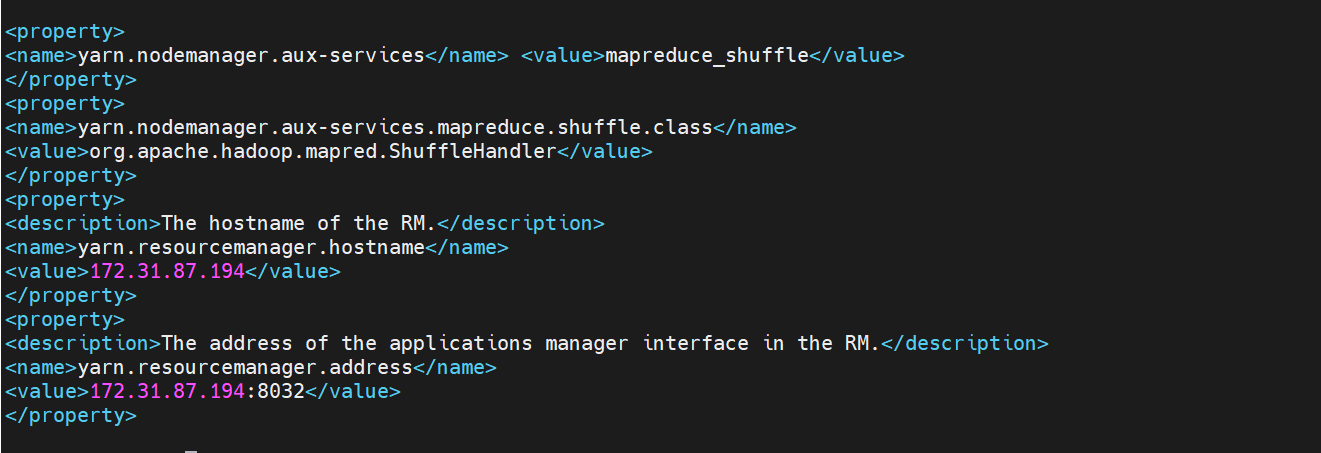
hadoop-env.sh

Configuration of XML Files:

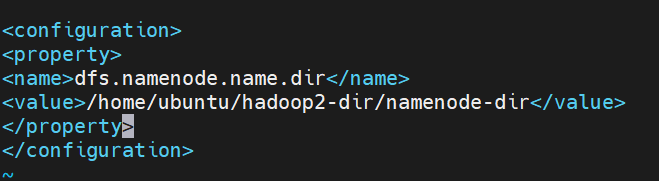
core-site.xml: Same for master as slaves

The IP address provided is the private IP address of the Master Node

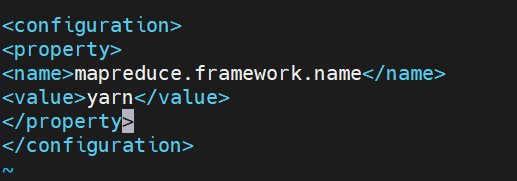
yarn-site.xml: Same for master and slaves

The IP address provided is the private IP address of the Master Node

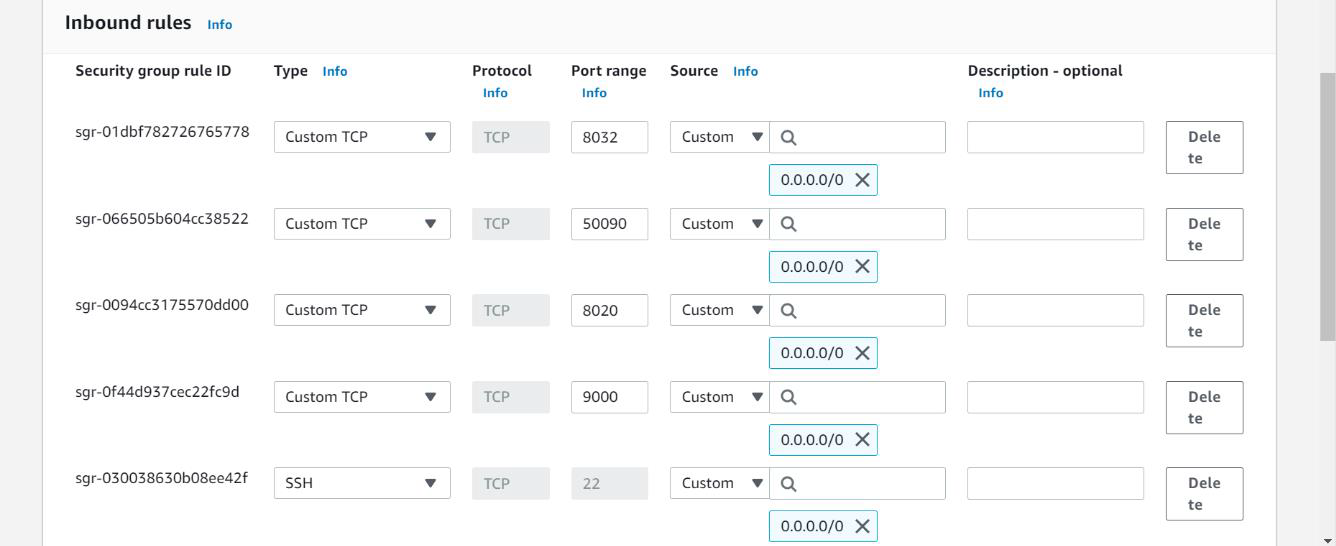
hdfs-site.xml:

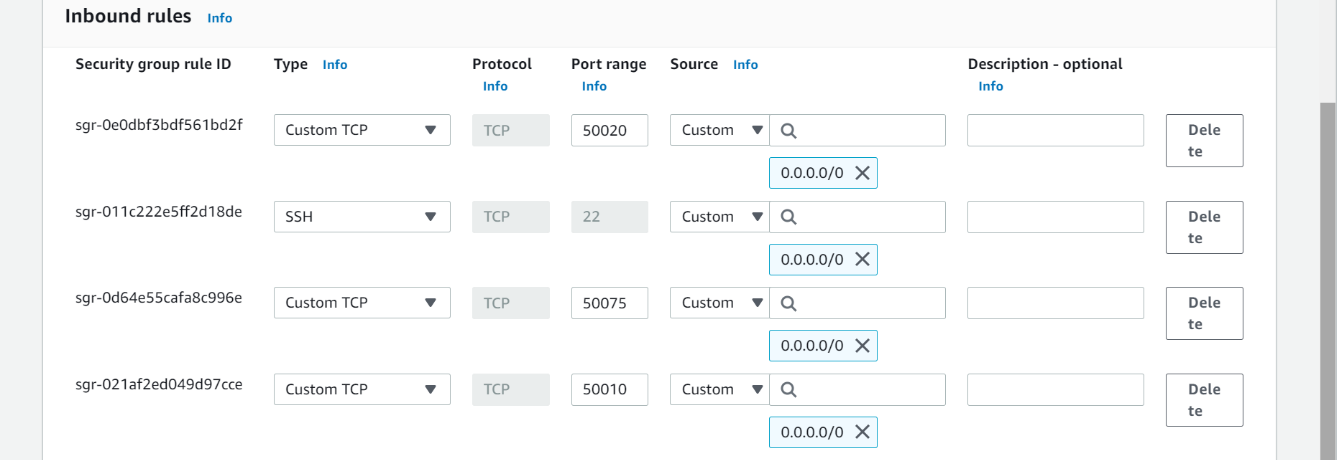
Mater Node:

Slave Nodes:

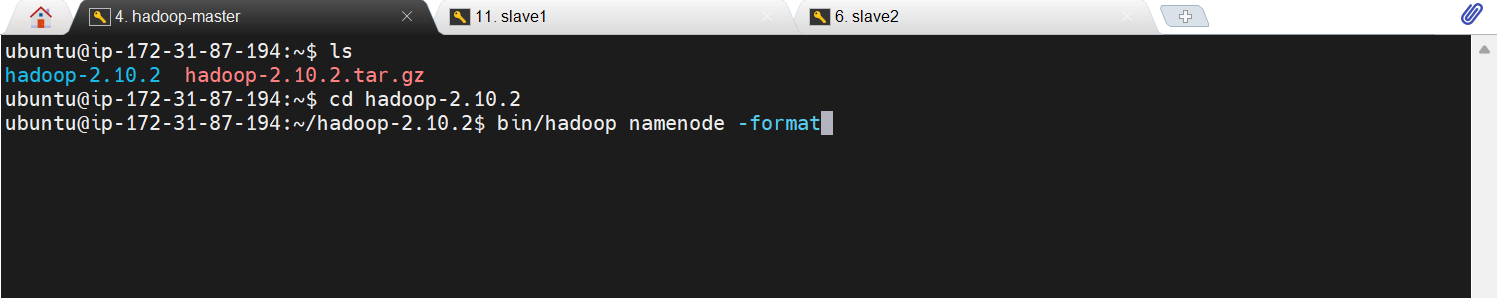
mapred-site.xml: Same for master and slaves

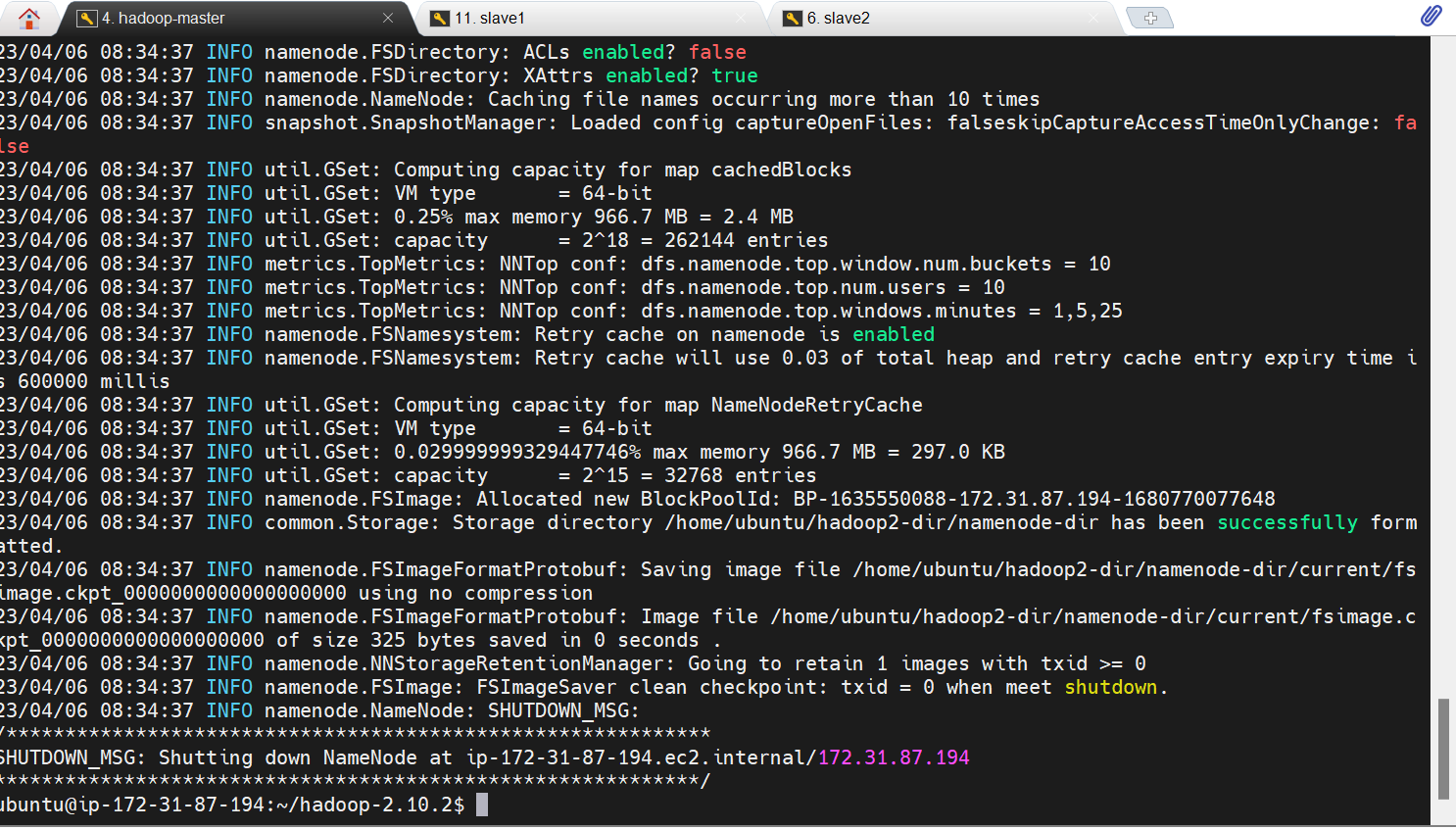
Step 4: Adding inbound rules in security groups of instances

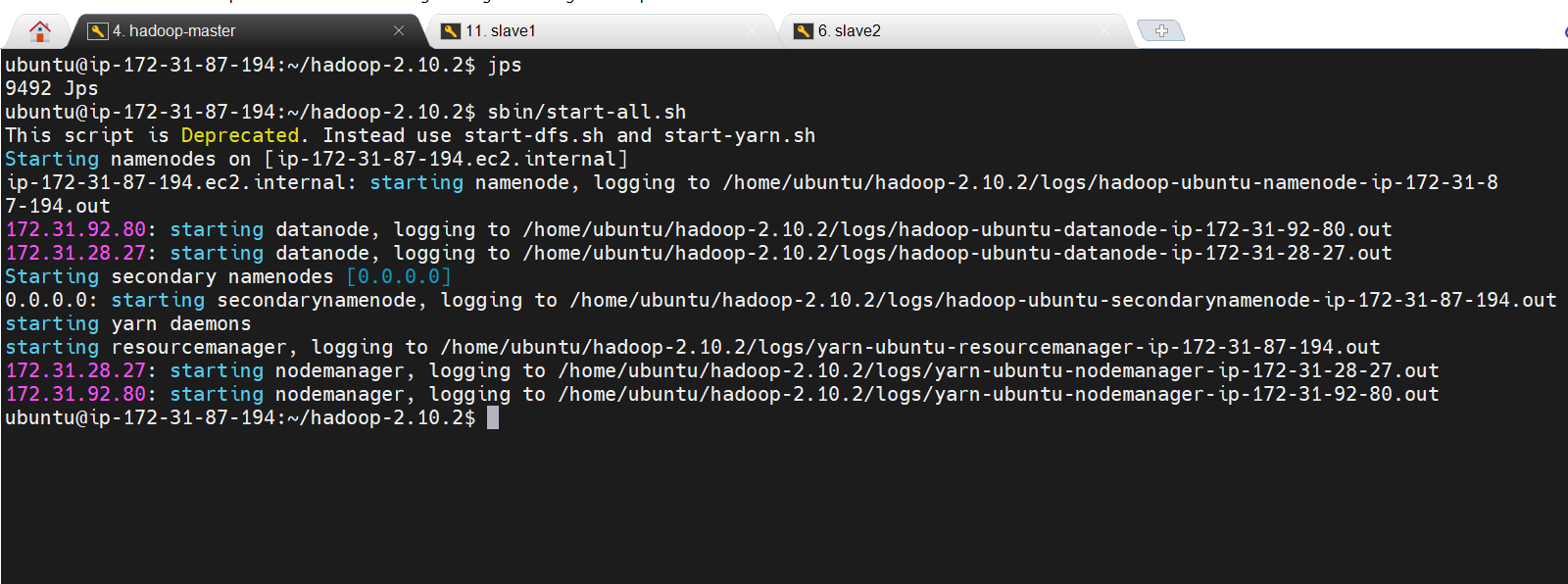
Master Instance:

Slave Instances:

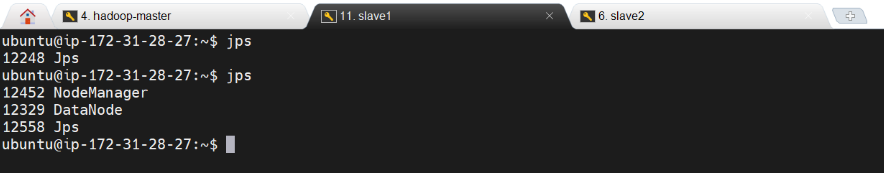
Step 5: Formatting of Hadoop Cluster and Starting of the daemons

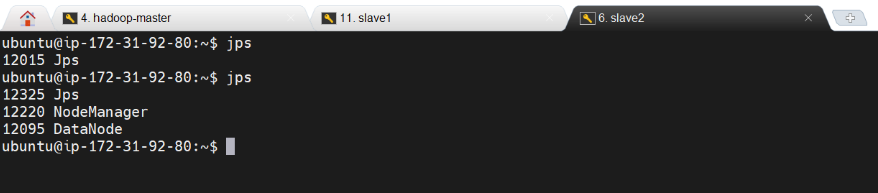
Format Hadoop Cluster: Command is executed only at the Master Node



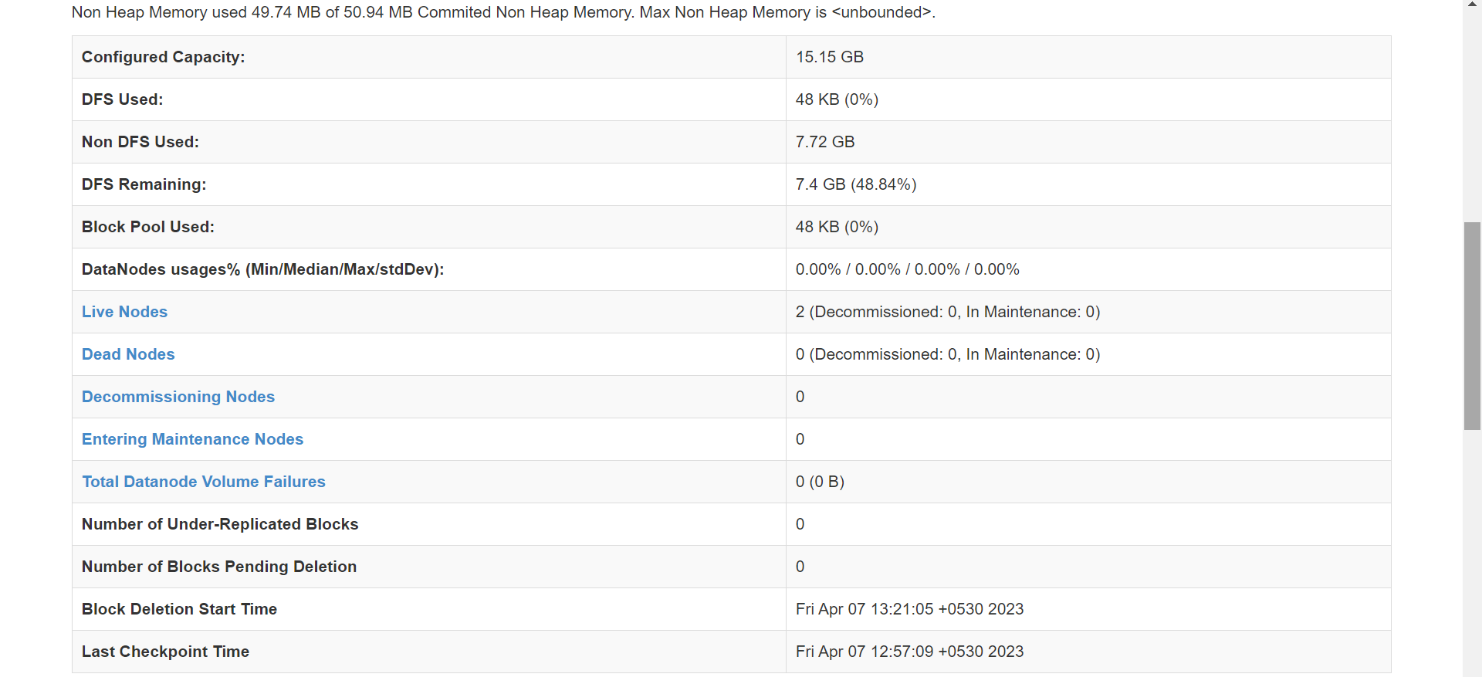
Starting of daemons: Command Executed only at Master Node

Daemons running at the Master:

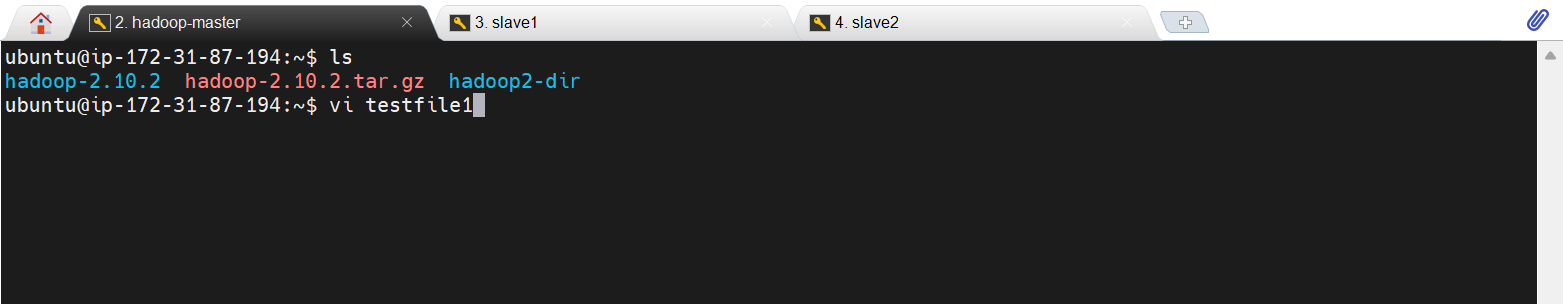
Daemons running at slave 1:

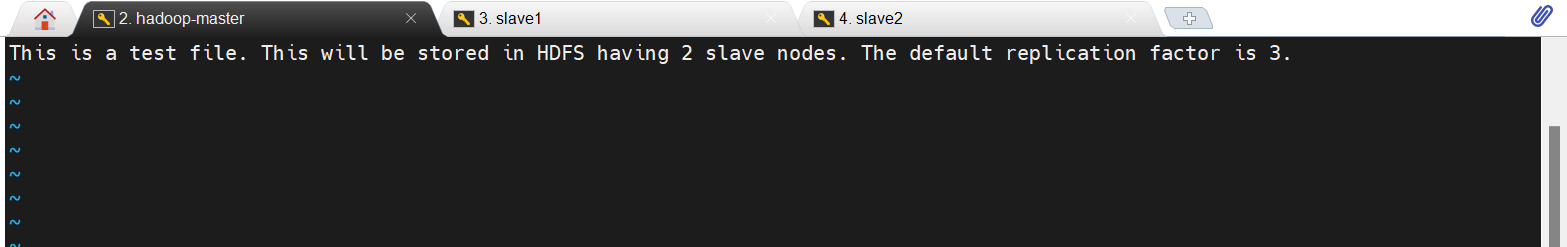
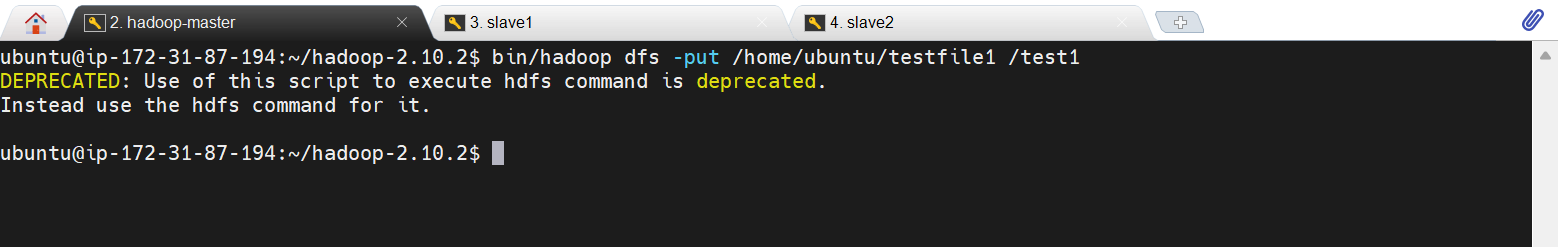
Daemons running at slave 2:

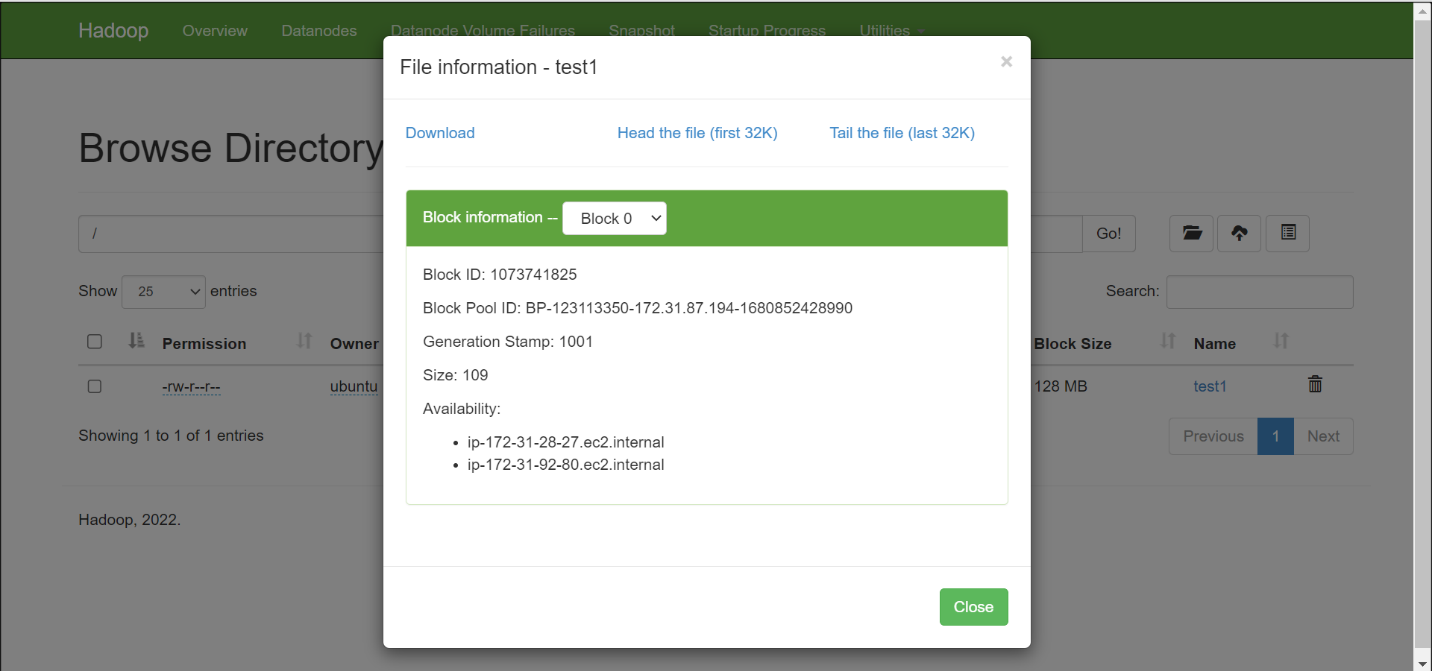
Step 6: Launching the Namenode Web Interface

The Web UI Provided by Apache, provides the following information about the cluster:

Step 7: Testing HDFS by storing files

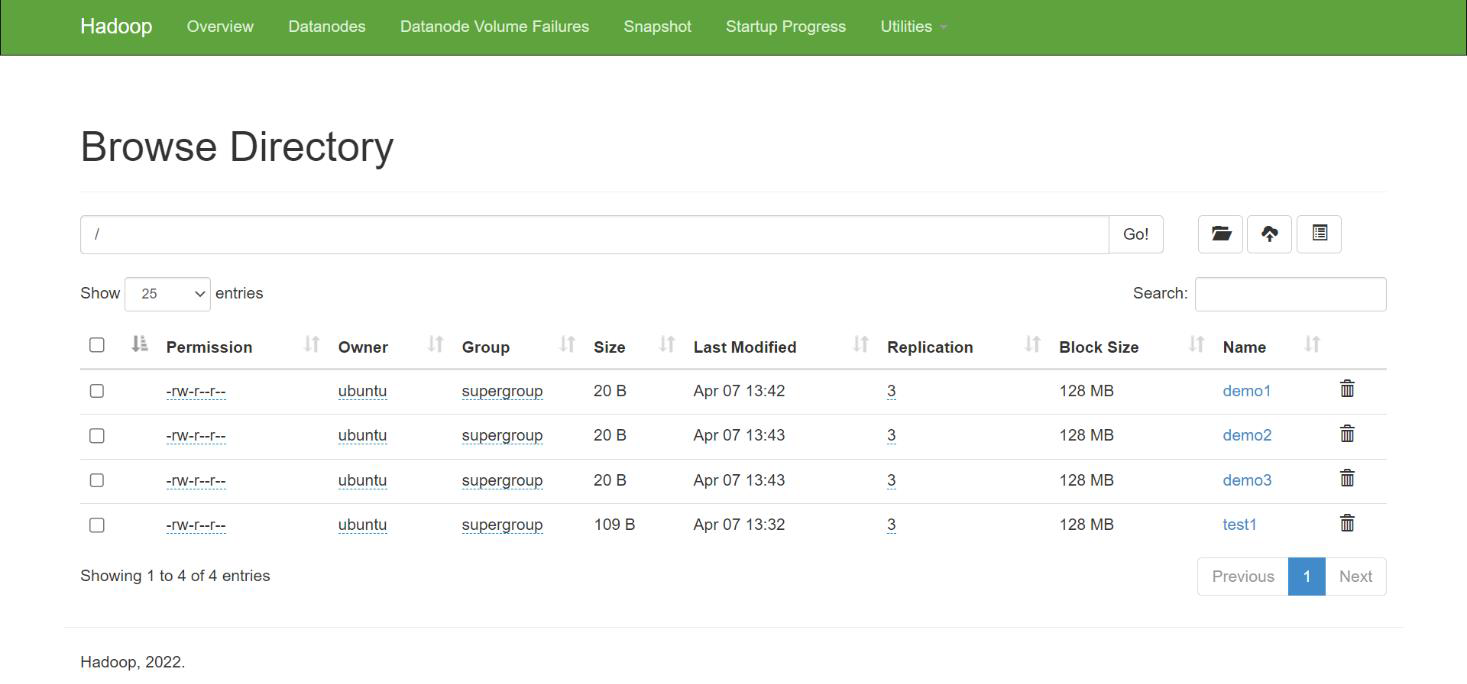
Creating a Test File

Storing it in the HDFS:

Successfully Stored in the DFS with replication factor of 3

Storing more files:





Conclusion:

In conclusion, our experiment studying MapReduce and Hadoop Distributed File System (HDFS) has shown that these technologies are powerful tools for managing and processing large-scale data sets in a distributed computing environment.

The implementation of Hadoop Distributed File System (HDFS) with one master and two slave nodes on Amazon Web Services (AWS) EC2 instances was successful. The experiment showed that HDFS can be deployed and managed on the cloud, allowing for the storage and processing of large amounts of data in a distributed manner.

References:

• Data Engineering. (2022, April 12). Hadoop Multi Node Cluster Setup [Video]. YouTube. https://www.youtube.com/watch?v=\_iP2Em-5Abw

• Kumar, M. R. N. (2022, September 17). Hadoop-3.3.1 Installation guide for Ubuntu - Dev Genius [Video]. Medium. https://blog.devgenius.io/install-configure-and-setup-hadoop-in-ubuntu-a3cdd6305a0e

• Gaurav Sharma. (2022, October 20). AWS Tutorials - 10 - Create First EC2 Instance | EC2 Instance Creation in AWS [Video]. YouTube. https://www.youtube.com/watch?v=f-T4xWUZWSk

Postlab Questions:

1. What are the differences between traditional file systems and HDFS?

| **Feature** | **Traditional File Systems** | **HDFS** |
| --- | --- | --- |
| Architecture | Typically single-node | Distributed across multiple nodes |
| Data storage | Data stored in a single server or disk | Data stored across a cluster of machines |
| Fault tolerance | Generally less fault-tolerant | Built-in fault tolerance through replication |
| Scalability | Limited scalability, especially for large datasets | Highly scalable, can handle petabytes of data |
| Access patterns | Optimized for random reads and writes | Optimized for streaming reads and writes |
| Consistency | Strong consistency, typically immediate updates | Eventual consistency, eventual updates |
| Metadata management | Centralized metadata management | Distributed metadata management |
| Data locality | Limited data locality, data may need to be moved to computation nodes | Maximizes data locality, computation happens where data resides |
| Processing framework | Not inherently integrated with big data processing frameworks | Integrated with Hadoop ecosystem for distributed processing |
| Use cases | General-purpose file storage for single servers or small clusters | Specifically designed for storing and processing large datasets in a distributed environment, commonly used in big data applications |

2. Enlist key features and components of HDFS.

1. Distributed Storage: HDFS distributes data across multiple nodes in a cluster, enabling scalability and fault tolerance.

2. High Fault Tolerance: Data replication across multiple nodes ensures high availability and fault tolerance. If a node fails, data can be retrieved from replicas stored on other nodes.

3. Scalability: HDFS is designed to scale horizontally, allowing it to handle massive amounts of data by adding more nodes to the cluster.

4. Data Locality: HDFS optimizes data processing by moving computation to where the data resides, minimizing data movement across the network.

5. Streaming Data Access: HDFS is optimized for streaming data access rather than random reads and writes, making it suitable for large-scale data processing.

6. Block-based Storage: Data is stored in large blocks (typically 128MB or 256MB), which improves throughput and reduces the overhead of managing a large number of small files.

7. Metadata Management: HDFS maintains metadata about files and directories in a centralized metadata server called the NameNode.

8. Data Replication: HDFS replicates data across multiple nodes to ensure fault tolerance and data durability. The replication factor is configurable, typically set to three.

9. Rack Awareness: HDFS is aware of the physical network topology, allowing it to place replicas across multiple racks for improved fault tolerance and data locality.

10. Checksums: HDFS uses checksums to detect and handle data corruption, ensuring data integrity during storage and transmission.

11. Command Line Interface (CLI): HDFS provides a command-line interface for users to interact with the file system, similar to traditional file systems.

12. Web User Interface (UI): HDFS includes a web-based UI that provides information about the cluster, file system, and data nodes.

13. Hadoop Ecosystem Integration: HDFS is a core component of the Apache Hadoop ecosystem, seamlessly integrating with other Hadoop projects such as MapReduce, HBase, Hive, and Spark for distributed data processing and analysis.