Apache Hadoop 3.3.6 Two-Node Cluster Installation Guide

This guide provides step-by-step instructions for installing and configuring Apache Hadoop 3.3.6 in a fully distributed 2-node cluster mode on Ubuntu 24.04 LTS.

Overview

This installation sets up Hadoop in fully distributed mode with:

- **Master Node**: Runs NameNode, ResourceManager, and JobHistoryServer
- **Worker Node**: Runs DataNode and NodeManager
- **Replication Factor**: 2 (data is replicated across both nodes)

Prerequisites

System Requirements

Master Node:

- Ubuntu 22.04 LTS or 24.04 LTS
- At least 4GB RAM (8GB recommended)
- 20GB of available disk space
- Root or sudo access
- Internet connection for downloading packages

Worker Node:

- Ubuntu 22.04 LTS or 24.04 LTS
- At least 4GB RAM (8GB recommended)
- 20GB of available disk space
- Root or sudo access
- Internet connection for downloading packages

Network Requirements

- Both nodes must be able to communicate via network
- Static IP addresses recommended
- Ports 8020-8040, 9000, 9870, 8088, 19888 must be open between nodes
- SSH access between nodes (passwordless)

Network Setup

1. Configure Hostnames

First, determine the IP addresses of both nodes by running `ip addr show` or `hostname -I` on each machine.

On Master Node (hadoop-master):

```
# Set the hostname
sudo hostnamectl set-hostname hadoop-master

# Backup current hosts file
sudo cp /etc/hosts /etc/hosts.backup

# Edit /etc/hosts file using nano or vim
sudo nano /etc/hosts
```

Add the following lines to `/etc/hosts`:

```
127.0.0.1 localhost
<master-ip> hadoop-master
<worker-ip> hadoop-worker1
```

On Worker Node (hadoop-worker1):

```
# Set the hostname
sudo hostnamectl set-hostname hadoop-worker1

# Backup current hosts file
sudo cp /etc/hosts /etc/hosts.backup

# Edit /etc/hosts file using nano or vim
sudo nano /etc/hosts
```

Add the following lines to `/etc/hosts`:

```
127.0.0.1 localhost
<master-ip> hadoop-master
<worker-ip> hadoop-worker1
```

Example with actual IP addresses: If master node IP is `192.168.1.10` and worker node IP is `192.168.1.11`, the `/etc/hosts` file on both nodes should contain:

```
127.0.0.1 localhost
192.168.1.10 hadoop-master
192.168.1.11 hadoop-worker1
```

Why configure hostnames? Proper hostname configuration ensures:

- Services can locate each other across the network
- Hadoop daemons can communicate correctly
- Web interfaces display proper node names
- Easier management and debugging of the cluster

2. Basic Network Connectivity Test

From Master Node:

```
ping -c 3 hadoop-worker1
```

From Worker Node:

```
ping -c 3 hadoop-master
```

Why test basic connectivity first? This confirms:

- Hostname resolution is working correctly
- Basic network communication between nodes is established
- Both nodes can reach each other at the network level
- No firewall issues blocking basic communication

Note: Full SSH connectivity testing will be done after SSH key setup in the next section.

System Setup (Both Nodes)

1. Verify Operating System

```
lsb_release -a
```

Why do this? This command verifies your Ubuntu version and helps ensure compatibility with Hadoop 3.3.6. Both nodes should run the same Ubuntu version for consistency and to avoid compatibility issues.

2. Install Java Development Kit (JDK)

On **both nodes**, run:

```
sudo apt update
sudo apt install -y openjdk-11-jdk
```

Why do this on both nodes?

- Every Hadoop daemon requires Java to run
- Consistent Java versions across nodes prevent compatibility issues

- OpenJDK 11 is stable, well-tested with Hadoop 3.3.6, and provides long-term support
- The `-y` flag automates the installation process

3. Verify Java Installation

On **both nodes**, run:

```
java -version
javac -version
```

Why verify on both nodes? Ensures that:

- Java was installed correctly on all cluster nodes
- All nodes have the same Java version for consistency
- Both the Java Runtime Environment (`java`) and compiler (`javac`) are available
- Prevents runtime errors due to missing or mismatched Java installations

4. Find Java Installation Path

On **both nodes**, run:

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

Why do this on both nodes? Hadoop needs the exact path to your Java installation on each node:

- Ensures JAVA_HOME is set correctly on all nodes
- Different systems might install Java in slightly different locations
- Consistent JAVA_HOME settings prevent "Java not found" errors

Hadoop Installation (Both Nodes)

1. Download Hadoop

On **both nodes**, run:

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

Why download on both nodes?

- Each node needs its own Hadoop installation
- Ensures consistent versions across the cluster
- Prevents issues with mismatched Hadoop versions between nodes

2. Extract Hadoop

On **both nodes**, run:

```
sudo tar -xzf hadoop-3.3.6.tar.gz -C /usr/local/
sudo mv /usr/local/hadoop-3.3.6 /usr/local/hadoop
sudo chown -R $USER:$USER /usr/local/hadoop
```

Why install on both nodes?

- Every Hadoop component (NameNode, DataNode, ResourceManager, NodeManager) runs as a separate process
- Each node needs the Hadoop binaries to run its assigned services
- Consistent installation paths simplify configuration and management

3. Set Environment Variables

On **both nodes**, add the following to `~/.bashrc`:

```
# Hadoop Environment Variables
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Why set environment variables on both nodes?

- All Hadoop processes need to know where Java and Hadoop are installed
- Consistent environment settings ensure predictable behavior across the cluster
- PATH modification allows running Hadoop commands from any directory

Apply the environment variables on **both nodes**:

```
source ~/.bashrc
```

4. Update Hadoop Environment Configuration

On **both nodes**, edit `/usr/local/hadoop/etc/hadoop/hadoop-env.sh` and set:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Why configure on both nodes?

- Each Hadoop daemon process needs to know where Java is located
- Prevents "JAVA_HOME not found" errors when starting services
- Ensures all cluster components have consistent Java environment

SSH Setup for Cluster Communication

1. Generate SSH Key on Master Node

On Master Node:

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

Why generate SSH keys? Hadoop uses SSH to:

- Start and manage daemons on worker nodes
- Execute commands across the cluster
- Provide secure communication between nodes

2. Copy Master Node SSH Key to Worker Node

Step 1: On Master Node - Display the public key

```
cat ~/.ssh/id_rsa.pub
```

This will output the public key (copy this entire line):

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDYsGMu4aH1F30RGfvpHwu10s9kmbbQpJApfeM2AxcYhCC
RcolYpTYUi/yxNWmba4FJavgrEIrALPLceXbJuTAlugytEzfiBMwZk9KQO9C/xzMNEUnWaPhVsz
Erkv8y+FLgbmw4doQI8qMEppCva+OUb37R6Bm2EMfZmTd8yYwcERtQQcWnhhSCS9PgHceQ1bO3h
hh6kFtCgv1+OB/wQ7+6VT8sYLQiu23IQxiDAp8Zk34og4LNyyfHwx0seO0ZB/Hhu2D4v3VIe2xT
PLfvAWIc20zV04X+5uvpz1+hw1KRvsWEQKPUMhuOt8B/Kx5TfEg6ptm/f659gt8iNN0JhKrcL4U
LLq28cIeThkTlVY61MIoc6B7xLhOZAPSjANHuAC+T+YaeSwMnsmR1rJE09WBfcQxy/XqruGl9wu
OBlsXtAFKHrD5PtkjexchgBdg4TsP+7z/we/YAaFuihJUiH2+gDu/y3uxaMEHATz7Q2yTg57WH4
herD2/akcAF61TkW1E= ubuntu@hadoop-master
```

Step 2: On Worker Node - Add master's key to authorized_keys

```
nano ~/.ssh/authorized_keys
```

Paste the master's public key (from Step 1) into the file, save and exit.

Why copy master key to worker? This enables:

- Master node to start Hadoop services on worker node
- Passwordless SSH from master to worker
- Automatic service startup across the cluster
- Hadoop cluster management functionality

Important Notes:

- Copy the entire public key as one single line (no line breaks)
- The `.ssh` directory should already exist from step 1 (SSH key generation)

3. Copy Worker Node SSH Key to Master Node

For full cluster functionality, you also need to copy the worker's public key to the master node.

Step 1: On Worker Node - Display the public key

```
cat ~/.ssh/id_rsa.pub
```

This will output the worker's public key (copy this entire line):

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDXJkK9JNq4vF7l2X8mY6rZpWqLk3VhR8tG5sB2nM4kQ6w
P7rJ1lX9zS0tY8uN3vE2dG7cA5fB4jK6lM9qR8tX2wZ5vN7bP1mK3cS8jL6rY4wT9xV2bN5zM8k
F7qP3sR1tV4wX6yZ2nN9mP8lK5jQ4xS7rV3bN6zM9kF8qP2sR1tV4wX6yZ2nN9mP8lK5jQ4xS7r
V3bN6z ubuntu@hadoop-worker1
```

Step 2: On Master Node - Add worker's key to authorized_keys

```
nano ~/.ssh/authorized_keys
```

Paste the worker's public key (from Step 1) into the file, save and exit.

Why copy worker key to master? This enables:

- Worker node to communicate back to master node
- Bidirectional cluster communication
- Self-SSH operations during service startup
- Full redundancy in cluster operations

Important Notes:

- Copy the entire public key as one single line (no line breaks)
- The `.ssh` directory should already exist from step 1 (SSH key generation)

4. Verify Complete Cluster Connectivity

From Master Node:

```
# Test SSH to worker node (should work without password)
ssh hadoop-worker1 'echo SSH to worker works'

# Test SSH to localhost
ssh localhost 'echo SSH to localhost works'

# Test hostname resolution via SSH
ssh hadoop-worker1 'hostname'
```

From Worker Node:

```
# Test SSH to localhost
ssh localhost 'echo SSH to localhost works'

# Test hostname resolution
hostname

# Test network connectivity to master
ping -c 2 hadoop-master
```

Why verify complete connectivity? This confirms:

- Passwordless SSH is working correctly between all nodes
- Hostname resolution works across the cluster
- All required SSH connections for Hadoop will work
- Network connectivity is fully established for cluster operations
- Hadoop will be able to start and manage services on all nodes

If SSH connection fails:

- Ensure you've copied the SSH keys correctly in step 2
- Check that the `.ssh` directory permissions are correct (`chmod 700 ~/.ssh`)
- Verify that `authorized_keys` has correct permissions (`chmod 600 ~/.ssh/authorized_keys`)
- Make sure the user accounts match between nodes (both should be `ubuntu`)

IMPORTANT NOTE: DO NOT PROCEED UNTIL ALL CONNECTIVITY TESTS PASS

You must verify that ALL of the following connectivity tests work successfully before proceeding with Hadoop installation:

✅ **Required Working Connections:**

- Master → Worker: `ssh hadoop-worker1 'hostname'` (must work without password)
- Worker → Master: `ssh hadoop-master 'hostname'` (must work without password)
- Master → Self: `ssh localhost 'hostname'` (must work without password)
- Worker → Self: `ssh localhost 'hostname'` (must work without password)

- Network ping: `ping hadoop-worker1` and `ping hadoop-master` (must respond)

Why this is critical:

- Hadoop cluster startup depends entirely on passwordless SSH between nodes
- If any SSH connection fails, Hadoop services will not start properly
- The cluster will be unable to distribute tasks and manage resources
- You will encounter "Connection refused" or "Permission denied" errors during cluster startup

Troubleshooting until connectivity works:

- Re-copy SSH keys if SSH prompts for passwords
- Check file permissions in `.ssh` directory
- Verify hostnames are correctly set in `/etc/hosts`
- Test basic network connectivity with ping commands
- Continue testing until ALL commands execute without errors or password prompts

Only proceed to Hadoop configuration after every single connectivity test passes successfully.

Hadoop Configuration

1. Core Configuration (core-site.xml)

On both nodes, edit `/usr/local/hadoop/etc/hadoop/core-site.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <!-- NameNode URI for HDFS -->
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop-master:9000</value>
        <description>The default file system URI for HDFS</description>
    </property>

    <!-- Directory for Hadoop temporary files -->
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/usr/local/hadoop/tmp</value>
        <description>Base directory for other temporary
directories</description>
    </property>

    <!-- Buffer size for reading files -->
    <property>
        <name>io.file.buffer.size</name>
        <value>4096</value>
        <description>Buffer size for reading files</description>
    </property>
</configuration>
```

Why configure this way? The key change from single-node setup:

- `fs.defaultFS` now points to `hadoop-master:9000` instead of `localhost:9000`
- This tells all Hadoop clients to connect to the master node for HDFS
- Other nodes will use this configuration to find the NameNode

2. HDFS Configuration (hdfs-site.xml)

On Master Node, edit `/usr/local/hadoop/etc/hadoop/hdfs-site.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <!-- Replication factor for 2-node cluster -->
    <property>
        <name>dfs.replication</name>
        <value>2</value>
        <description>Default block replication for HDFS
blocks</description>
    </property>

    <!-- NameNode data directory -->
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/usr/local/hadoop/data/namenode</value>
        <description>Directory for NameNode data storage</description>
    </property>

    <!-- DataNode data directory -->
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/local/hadoop/data/datanode</value>
        <description>Directory for DataNode data storage</description>
    </property>

    <!-- Block size configuration -->
    <property>
        <name>dfs.blocksize</name>
        <value>128m</value>
        <description>Default block size for HDFS files</description>
    </property>

    <!-- Web UI for NameNode -->
    <property>
        <name>dfs.http.address</name>
        <value>hadoop-master:9870</value>
        <description>NameNode HTTP web UI address</description>
    </property>

    <!-- Enable permission checking -->
    <property>
        <name>dfs.permissions</name>
```

```
        <value>false</value>
        <description>Disable permission checking for cluster
setup</description>
    </property>
</configuration>
```

On Worker Node, edit `/usr/local/hadoop/etc/hadoop/hdfs-site.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <!-- Replication factor for 2-node cluster -->
    <property>
        <name>dfs.replication</name>
        <value>2</value>
        <description>Default block replication for HDFS
blocks</description>
    </property>

    <!-- DataNode data directory -->
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/local/hadoop/data/datanode</value>
        <description>Directory for DataNode data storage</description>
    </property>

    <!-- Block size configuration -->
    <property>
        <name>dfs.blocksize</name>
        <value>128m</value>
        <description>Default block size for HDFS files</description>
    </property>

    <!-- Enable permission checking -->
    <property>
        <name>dfs.permissions</name>
        <value>false</value>
        <description>Disable permission checking for cluster
setup</description>
    </property>
</configuration>
```

Why different configurations?

- Master node includes NameNode configuration (only runs on master)
- Worker node only needs DataNode configuration
- Replication factor set to 2 for data redundancy across both nodes
- dfs.http.address points to master for centralized web UI access

3. MapReduce Configuration (mapred-site.xml)

On both nodes, edit `/usr/local/hadoop/etc/hadoop/mapred-site.xml`:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <!-- MapReduce framework name -->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
        <description>Execution framework for MapReduce jobs</description>
    </property>

    <!-- MapReduce job history server address -->
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>hadoop-master:10020</value>
        <description>Address for MapReduce job history server</description>
    </property>

    <!-- MapReduce job history web UI address -->
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>hadoop-master:19888</value>
        <description>Web UI address for MapReduce job history
server</description>
    </property>

    <!-- Directory for MapReduce application logs -->
    <property>
        <name>mapreduce.jobtracker.system.dir</name>
        <value>file:/usr/local/hadoop/data/mapred/system</value>
        <description>Directory for MapReduce system files</description>
    </property>

    <!-- Directory for MapReduce staging -->
    <property>
        <name>mapreduce.cluster.local.dir</name>
        <value>file:/usr/local/hadoop/data/mapred/local</value>
        <description>Local directory for MapReduce tasks</description>
    </property>

    <!-- Environment variables for MapReduce -->
    <property>
        <name>yarn.app.mapreduce.am.env</name>
        <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
        <description>Environment variables for MapReduce Application
Master</description>
    </property>

    <property>
        <name>mapreduce.map.env</name>
        <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
        <description>Environment variables for Map tasks</description>
```

```
    </property>

    <property>
        <name>mapreduce.reduce.env</name>
        <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
        <description>Environment variables for Reduce tasks</description>
    </property>
</configuration>
```

Why configure MapReduce this way? Key cluster considerations:

- JobHistory server runs on master node only (`hadoop-master`)
- All nodes know where to find job history services
- HADOOP_MAPRED_HOME variables prevent classpath issues
- Consistent MapReduce configuration across all nodes

4. YARN Configuration (yarn-site.xml)

On Master Node, edit `/usr/local/hadoop/etc/hadoop/yarn-site.xml`:

```
<?xml version="1.0"?>
<configuration>
    <!-- NodeManager settings -->
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
        <description>Shuffle service for MapReduce</description>
    </property>

    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        <description>Shuffle handler class</description>
    </property>

    <!-- ResourceManager settings -->
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoop-master</value>
        <description>ResourceManager hostname</description>
    </property>

    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>hadoop-master:8030</value>
        <description>Scheduler address for ResourceManager</description>
    </property>

    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>hadoop-master:8031</value>
```

```
        <description>Resource tracker address for
ResourceManager</description>
    </property>

    <property>
        <name>yarn.resourcemanager.address</name>
        <value>hadoop-master:8032</value>
        <description>ResourceManager address for applications</description>
    </property>

    <property>
        <name>yarn.resourcemanager.admin.address</name>
        <value>hadoop-master:8033</value>
        <description>ResourceManager admin address</description>
    </property>

    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>hadoop-master:8088</value>
        <description>ResourceManager web UI address</description>
    </property>

    <!-- Memory and CPU allocation -->
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>4096</value>
        <description>Memory available for NodeManager in MB</description>
    </property>

    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>2</value>
        <description>Number of virtual cores available for
NodeManager</description>
    </property>

    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>512</value>
        <description>Minimum memory allocation for containers</description>
    </property>

    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>4096</value>
        <description>Maximum memory allocation for containers</description>
    </property>

    <!-- Application log aggregation -->
    <property>
        <name>yarn.log-aggregation-enable</name>
        <value>true</value>
        <description>Enable log aggregation for applications</description>
    </property>
```

```
    <property>
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>4</value>
        <description>Virtual memory to physical memory ratio</description>
    </property>
</configuration>
```

On Worker Node, edit `/usr/local/hadoop/etc/hadoop/yarn-site.xml`:

```
<?xml version="1.0"?>
<configuration>
    <!-- NodeManager settings -->
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
        <description>Shuffle service for MapReduce</description>
    </property>

    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        <description>Shuffle handler class</description>
    </property>

    <!-- ResourceManager settings -->
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoop-master</value>
        <description>ResourceManager hostname</description>
    </property>

    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>hadoop-master:8030</value>
        <description>Scheduler address for ResourceManager</description>
    </property>

    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>hadoop-master:8031</value>
        <description>Resource tracker address for
ResourceManager</description>
    </property>

    <property>
        <name>yarn.resourcemanager.address</name>
        <value>hadoop-master:8032</value>
        <description>ResourceManager address for applications</description>
    </property>

    <!-- Memory and CPU allocation -->
```

```
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>4096</value>
        <description>Memory available for NodeManager in MB</description>
    </property>

    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>2</value>
        <description>Number of virtual cores available for
NodeManager</description>
    </property>

    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>512</value>
        <description>Minimum memory allocation for containers</description>
    </property>

    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>4096</value>
        <description>Maximum memory allocation for containers</description>
    </property>

    <!-- Application log aggregation -->
    <property>
        <name>yarn.log-aggregation-enable</name>
        <value>true</value>
        <description>Enable log aggregation for applications</description>
    </property>

    <property>
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>4</value>
        <description>Virtual memory to physical memory ratio</description>
    </property>
</configuration>
```

Why configure YARN this way? Important cluster considerations:

- ResourceManager only runs on master node
- All ResourceManager addresses point to `hadoop-master`
- Worker nodes know where to connect for resource management
- NodeManager configuration is consistent across worker nodes
- Web UI access centralized on master node

5. Workers Configuration

On Master Node, edit `/usr/local/hadoop/etc/hadoop/workers` and replace contents with:

```
hadoop-master
hadoop-worker1
```

On Worker Node, edit `/usr/local/hadoop/etc/hadoop/workers` and replace contents with:

```
hadoop-worker1
```

Why different workers files?

- Master node's workers file includes both nodes because it can start services on all nodes
- Worker node's workers file only includes itself to prevent it from trying to start services on master
- This prevents duplicate service startup and conflicts

HDFS Setup

1. Create Required Directories

On Master Node:

```
mkdir -p /usr/local/hadoop/tmp
mkdir -p /usr/local/hadoop/data/namenode
mkdir -p /usr/local/hadoop/data/datanode
mkdir -p /usr/local/hadoop/data/mapred/system
mkdir -p /usr/local/hadoop/data/mapred/local
```

On Worker Node:

```
mkdir -p /usr/local/hadoop/tmp
mkdir -p /usr/local/hadoop/data/datanode
mkdir -p /usr/local/hadoop/data/mapred/system
mkdir -p /usr/local/hadoop/data/mapred/local
```

NOTE: Worker node does NOT need `namenode` directory since NameNode only runs on master.

Why different directories?

- Master node needs namenode directory (only runs NameNode)
- Both nodes need datanode directory for storing data blocks
- MapReduce directories needed on nodes that run MapReduce tasks
- Worker nodes will automatically start their services when master initiates cluster startup

2. Format NameNode (Master Node Only)

On Master Node:

```
hdfs namenode -format
```

Why format only on master?

- NameNode only runs on master node in this configuration
- Formatting initializes the HDFS metadata storage
- **Warning**: Only do this once. Re-formatting erases all HDFS data
- Worker nodes don't need formatting as they only store data blocks

3. Verify Directory Permissions

On both nodes:

```
ls -la /usr/local/hadoop/data/
```

Expected output on Master Node:

```
total 20
drwxrwxr-x  5 ubuntu ubuntu 4096 Oct 22 14:10 .
drwxr-xr-x 14 ubuntu ubuntu 4096 Oct 23 08:07 ..
drwx------  3 ubuntu ubuntu 4096 Oct 22 14:48 datanode
drwxrwxr-x  4 ubuntu ubuntu 4096 Oct 22 14:10 mapred
drwxrwxr-x  3 ubuntu ubuntu 4096 Oct 23 10:00 namenode
```

Expected output on Worker Node:

```
total 16
drwxrwxr-x  4 ubuntu ubuntu 4096 Oct 22 14:10 .
drwxr-xr-x 14 ubuntu ubuntu 4096 Oct 23 08:07 ..
drwx------  3 ubuntu ubuntu 4096 Oct 22 14:48 datanode
drwxrwxr-x  4 ubuntu ubuntu 4096 Oct 22 14:10 mapred
```

Why check permissions?

- Hadoop processes need write access to data directories
- Incorrect permissions are a common cause of startup failures
- Ensures the user running Hadoop can create and manage files
- Master node should have `namenode`, `datanode`, and `mapred` directories
- Worker node should only have `datanode` and `mapred` directories (no namenode)
- Directories should be owned by your user (ubuntu:ubuntu in this example)
- `datanode` directory will show `drwx------` permissions after NameNode formatting

Starting the Hadoop Cluster

1. Start HDFS (From Master Node)

On Master Node:

```
start-dfs.sh
```

Why start from master?

- This script uses SSH to start HDFS services on all nodes listed in workers file
- Starts NameNode on master, DataNodes on all configured nodes
- Master coordinates the startup sequence across the cluster

2. Start YARN (From Master Node)

On Master Node:

```
start-yarn.sh
```

Why start YARN separately?

- YARN has different startup requirements than HDFS
- ResourceManager starts on master, NodeManagers on workers
- Separating starts allows for better debugging and control

3. Start JobHistory Server (From Master Node)

On Master Node:

```
mr-jobhistory-daemon.sh start historyserver
```

Why start JobHistory Server?

- Tracks completed MapReduce jobs for debugging and analysis
- Provides historical job information via web UI
- Only runs on master node in this configuration

4. Verify Services

On Master Node:

```
jps
```

Expected output:

```
NameNode
DataNode
ResourceManager
NodeManager
SecondaryNameNode
JobHistoryServer
Jps
```

On Worker Node:

```
jps
```

Expected output:

```
DataNode
NodeManager
Jps
```

Why verify services?

- Confirms all Hadoop services started successfully on each node
- Shows which services are running on which nodes
- Essential troubleshooting step before using the cluster

Cluster Verification

1. Check HDFS Cluster Status

On Master Node:

```
hdfs dfsadmin -report
```

Why check cluster status?

- Shows cluster health and capacity across all nodes
- Lists active DataNodes and their storage capacity
- Displays configured vs. available disk space
- Confirms data replication is working (should show 2 live nodes)

2. Test HDFS Operations

On Master Node:

```

```

```
# Create directories
hdfs dfs -mkdir -p /user/ubuntu/input

# Create test file
echo "Hello Hadoop Cluster World
This is a test file for distributed WordCount example
Hadoop makes big data processing easy across multiple nodes
MapReduce is the programming model for distributed computing
WordCount counts words in text files using parallel processing
The cluster distributes work across multiple machines for scalability" >
test.txt

# Upload to HDFS
hdfs dfs -put test.txt /user/ubuntu/input/

# List files
hdfs dfs -ls /user/ubuntu/input

# Check file replication
hdfs dfs -stat %r /user/ubuntu/input/test.txt
```

Why test HDFS operations?

- Verifies HDFS is working correctly across the cluster
- Tests file replication across multiple nodes
- Confirms your user has proper permissions
- Creates test data for the upcoming MapReduce job
- The replication factor should show 2 (data stored on both nodes)

3. Run Distributed MapReduce WordCount

On Master Node:

```
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.3.6.jar wordcount /user/ubuntu/input/test.txt /user/ubuntu/output
```

Why run distributed WordCount?

- Tests the complete distributed MapReduce pipeline
- Verifies YARN can schedule tasks across multiple nodes
- Confirms data shuffling between nodes works correctly
- Tests the entire cluster ecosystem working together
- Demonstrates the power of distributed processing

4. Monitor Job Execution

On Master Node:

```
# Check running applications
yarn application -list

# Track job progress
yarn application -status <application_id>
```

Why monitor job execution?

- Shows how YARN distributes tasks across nodes
- Confirms both nodes are participating in job execution
- Helps understand cluster resource utilization
- Demonstrates distributed processing in action

5. View Results

On Master Node:

```
hdfs dfs -cat /user/ubuntu/output/part-r-00000
```

Expected output should show word counts with distributed processing benefits:

```
Hadoop  2
Hello   1
Cluster 1
World   1
MapReduce   1
distributed 2
...
```

Why check results?

- Confirms the distributed job completed successfully
- Verifies output was written correctly to HDFS
- Validates the entire pipeline from input to distributed processing to output
- Demonstrates successful cluster operation

Cluster Management

Starting the Cluster

From Master Node:

```
# Start all HDFS services
start-dfs.sh
```

```
# Start all YARN services
start-yarn.sh

# Start JobHistory Server
mr-jobhistory-daemon.sh start historyserver
```

Stopping the Cluster

From Master Node:

```
# Stop JobHistory Server
mr-jobhistory-daemon.sh stop historyserver

# Stop YARN services
stop-yarn.sh

# Stop HDFS services
stop-dfs.sh
```

Checking Cluster Health

From Master Node:

```
# Check all running processes
jps

# Check HDFS cluster status
hdfs dfsadmin -report

# Check YARN applications
yarn application -list

# Check node status
yarn node -list
```

From any node:

```
# Check local services
jps

# Check HDFS health
hdfs fsck /
```

Web Interfaces

After starting services, you can access the following web interfaces (all on master node):

- **NameNode Web UI**: http://hadoop-master:9870
- **ResourceManager Web UI**: http://hadoop-master:8088
- **NodeManager Web UI**: http://hadoop-master:8042 (master) / http://hadoop-worker1:8042 (worker)
- **MapReduce Job History**: http://hadoop-master:19888

Why access web interfaces?

- Monitor cluster health and resource usage
- Track job execution and performance
- Debug issues and view logs
- Understand cluster topology and data distribution

Advanced Cluster Operations

Adding Data to Cluster

From Master Node:

```
# Upload large dataset for distributed processing
hdfs dfs -put /path/to/large/dataset /user/ubuntu/bigdata/

# Set replication factor for specific files
hdfs dfs -setrep -w 3 /user/ubuntu/critical_data/

# Check data distribution
hdfs dfsadmin -report
```

Running Multiple Jobs

From Master Node:

```
# Submit multiple jobs concurrently
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.3.6.jar pi 10 1000 &
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.3.6.jar grep /user/ubuntu/input /user/ubuntu/grep_output 'distributed' &

# Monitor queue
yarn queue -status default
```

Cluster Load Balancing

From Master Node:

```
# Check data distribution
hdfs dfsadmin -report

# Trigger balancer if needed (run during maintenance window)
hdfs balancer -threshold 10
```

Troubleshooting

Common Cluster Issues

1. DataNode Won't Connect to NameNode

Symptoms: DataNode process starts but doesn't appear in cluster report

Solutions:

```
# On worker node, check DataNode logs
tail -f /usr/local/hadoop/logs/hadoop-*-datanode-*.log

# Check network connectivity
ping hadoop-master

# Verify firewall ports are open
telnet hadoop-master 9000

# Restart DataNode
hadoop-daemon.sh stop datanode
hadoop-daemon.sh start datanode
```

2. NodeManager Won't Register with ResourceManager

Symptoms: NodeManager starts but ResourceManager shows no live nodes

Solutions:

```
# Check YARN configuration
cat /usr/local/hadoop/etc/hadoop/yarn-site.xml | grep resourcemanager

# Verify ResourceManager hostname resolution
nslookup hadoop-master

# Check ResourceManager logs
tail -f /usr/local/hadoop/logs/yarn-*-resourcemanager-*.log

# Restart NodeManager
yarn-daemon.sh stop nodemanager
yarn-daemon.sh start nodemanager
```

3. SSH Connection Issues

Symptoms: "Connection refused" or "Permission denied" when starting cluster

Solutions:

```
# Test SSH connectivity
ssh hadoop-worker1 'echo SSH works'

# Check SSH keys
ls -la ~/.ssh/

# Regenerate SSH keys if needed
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
ssh-copy-id ubuntu@hadoop-worker1

# Check SSH service
sudo systemctl status ssh
```

4. Memory Issues on Worker Nodes

Symptoms: Containers fail with OutOfMemoryError

Solutions:

```
# Reduce memory allocation in yarn-site.xml on affected node
# Edit /usr/local/hadoop/etc/hadoop/yarn-site.xml
<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>2048</value>
</property>

# Restart NodeManager
yarn-daemon.sh restart nodemanager
```

5. Replication Issues

Symptoms: HDFS shows under-replicated blocks

Solutions:

```
# Check replication status
hdfs fsck /

# Force replication
hdfs dfs -setrep -R 2 /
```

```
# Check DataNode disk space
hdfs dfsadmin -report
```

Log Locations

Master Node:

- **NameNode logs**: `/usr/local/hadoop/logs/hadoop-*-namenode-*.log`
- **ResourceManager logs**: `/usr/local/hadoop/logs/yarn-*-resourcemanager-*.log`
- **JobHistory logs**: `/usr/local/hadoop/logs/mr-*-historyserver-*.log`

Worker Node:

- **DataNode logs**: `/usr/local/hadoop/logs/hadoop-*-datanode-*.log`
- **NodeManager logs**: `/usr/local/hadoop/logs/yarn-*-nodemanager-*.log`

Network Troubleshooting

```
# Check all Hadoop ports
netstat -tulpn | grep -E
'(8020|8030|8031|8032|8033|8040|8042|8088|9000|9870|10020|19888)'

# Test connectivity between nodes
nc -zv hadoop-master 9000
nc -zv hadoop-worker1 8042

# Check firewall status
sudo ufw status
```

Performance Tuning for 2-Node Cluster

Memory Optimization

On both nodes, edit `/usr/local/hadoop/etc/hadoop/yarn-site.xml`:

```
<!-- Optimize memory allocation for 2-node cluster -->
<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>6144</value>
</property>

<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>6144</value>
</property>

<property>
    <name>yarn.app.mapreduce.am.resource.mb</name>
    <value>1536</value>
```

```
</property>

<property>
    <name>mapreduce.map.memory.mb</name>
    <value>1024</value>
</property>

<property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>2048</value>
</property>
```

HDFS Optimization

On Master Node, edit `/usr/local/hadoop/etc/hadoop/hdfs-site.xml`:

```
<!-- Optimize for 2-node cluster -->
<property>
    <name>dfs.namenode.handler.count</name>
    <value>10</value>
</property>

<property>
    <name>dfs.datanode.handler.count</name>
    <value>10</value>
</property>

<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
```

MapReduce Optimization

On both nodes, edit `/usr/local/hadoop/etc/hadoop/mapred-site.xml`:

```
<!-- Optimize for 2-node cluster -->
<property>
    <name>mapreduce.task.io.sort.mb</name>
    <value>200</value>
</property>

<property>
    <name>mapreduce.reduce.shuffle.parallelcopies</name>
    <value>5</value>
</property>

<property>
    <name>mapreduce.map.speculative</name>
```

```
    <value>true</value>
</property>

<property>
    <name>mapreduce.reduce.speculative</name>
    <value>true</value>
</property>
```

Security Considerations

Firewall Configuration

On both nodes:

```
# Open Hadoop ports
sudo ufw allow 22/tcp
sudo ufw allow 8020:8040/tcp
sudo ufw allow 9000/tcp
sudo ufw allow 9870/tcp
sudo ufw allow 8088/tcp
sudo ufw allow 19888/tcp
sudo ufw enable
```

Basic Security Hardening

On both nodes:

```
<!-- In hdfs-site.xml -->
<property>
    <name>dfs.permissions</name>
    <value>true</value>
</property>

<!-- In core-site.xml -->
<property>
    <name>hadoop.security.authentication</name>
    <value>simple</value>
</property>
```

User Management

On both nodes:

```
# Create hadoop user group
sudo groupadd hadoop

# Add users to hadoop group
```

```
sudo usermod -a -G hadoop ubuntu

# Set appropriate permissions
sudo chown -R ubuntu:hadoop /usr/local/hadoop
```

Backup and Recovery

Configuration Backup

On Master Node:

```
# Backup all configuration files
tar -czf hadoop-config-backup.tar.gz /usr/local/hadoop/etc/hadoop

# Store backup off-site
scp hadoop-config-backup.tar.gz backup-server:/backups/
```

Data Backup Strategies

From Master Node:

```
# Backup important HDFS data
hdfs dfs -copyToLocal /user/ubuntu/critical_data /backup/hdfs_backup/

# Create HDFS snapshot
hdfs dfsadmin -saveNamespace

# Backup to external storage
hdfs dfs -put /backup/hdfs_backup/ hdfs://backup-cluster:9000/backups/
```

Disaster Recovery

If Master Node Fails:

1. Promote worker to master (advanced procedure)
2. Restore configuration from backup
3. Format new NameNode (data loss)
4. Restore data from backups

If Worker Node Fails:

1. Replace failed node
2. Install Hadoop with same configuration
3. Add node back to cluster
4. HDFS will automatically rebalance data

Scaling Beyond 2 Nodes

Adding Additional Worker Nodes

1. **Prepare New Node**:
    - Install Ubuntu and Java
    - Configure network and hostnames
    - Set up SSH access from master
2. **Install Hadoop**:
    - Copy configuration from master
    - Create required directories
    - Set environment variables
3. **Add to Cluster**:
    - Add new hostname to `/usr/local/hadoop/etc/hadoop/workers` on master
    - Start DataNode and NodeManager on new node
    - HDFS will automatically start replicating data to new node

High Availability Setup

For production clusters, consider:

- Multiple NameNodes (active/passive)
- ZooKeeper for coordination
- Multiple ResourceManagers
- Load balancers for web interfaces

References

- Apache Hadoop Official Documentation
- Hadoop 3.3.6 Release Notes
- Hadoop Cluster Setup Guide
- Hadoop Configuration Guide

License