Apache Hadoop 3.3.6 Single-Node Installation Guide

This guide provides step-by-step instructions for installing and configuring Apache Hadoop 3.3.6 in single-node (pseudo-distributed) mode on Ubuntu 24.04 LTS.

Overview

This installation sets up Hadoop in pseudo-distributed mode where all Hadoop daemons run on a single machine. This configuration is ideal for development, testing, and learning purposes.

Prerequisites

- 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

System Setup

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. Different Ubuntu versions may have different package availability, Java versions, and system configurations. Knowing your exact version helps troubleshoot compatibility issues and ensures you're following the correct installation procedures for your specific environment.

2. Install Java Development Kit (JDK)

Hadoop requires Java 11 or newer. We'll install OpenJDK 11:

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

Why do this?

- `sudo apt update` refreshes the package lists to ensure you get the latest available versions and security updates
- Hadoop is built in Java and requires a Java Development Kit (JDK) to run. OpenJDK 11 is the recommended version as it's stable, well-tested with Hadoop 3.3.6, and provides long-term support
- The `-y` flag automatically confirms the installation, which is useful for scripted installations

3. Verify Java Installation

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

Why do this? This verification step ensures that:

- Java was installed correctly and is accessible in your PATH
- You have the correct version (Java 11+) that's compatible with Hadoop 3.3.6
- Both the Java Runtime Environment (`java`) and compiler (`javac`) are available
- If either command fails, it indicates a problem with the Java installation that needs to be fixed before proceeding

Expected output:

```
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1,
mixed mode, sharing)
```

4. Find Java Installation Path

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

Why do this? Hadoop needs the exact path to your Java installation to set the JAVA_HOME environment variable correctly. This command:

- Uses `which java` to find where the java executable is located in your PATH
- Uses `readlink -f` to resolve any symbolic links and give you the actual installation directory
- The resulting path is used to configure JAVA_HOME, which is critical for Hadoop to find and use Java properly

Expected output:

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/java
```

Hadoop Installation

1. Download Hadoop

Hadoop 3.3.6 is used in this guide. Download from Apache archive:

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

Why do this? This step downloads the official Hadoop 3.3.6 binary distribution from Apache's servers:

- Version 3.3.6 is a stable release with good performance and security fixes
- Downloading from Apache ensures you get an authentic, unmodified package
- If you already have the file locally (as mentioned in your requirements), you can skip this step

2. Extract Hadoop

```
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 do these commands?

- `sudo tar -xzf` extracts the compressed archive to `/usr/local/`, a standard location for software installations
- The `-xzf` flags extract, gzip-decompress, and use the filename parameter
- `mv` renames the versioned directory to a simple `hadoop` name for easier path management
- `chown -R $USER:$USER` changes ownership to your user account, preventing permission issues when Hadoop tries to write logs or data files

3. Set Environment Variables

Add the following to your `~/.bashrc` file:

```
# Hadoop Environment Variables
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop/hadoop-3.3.6 # make sure in this
folder has etc, bin, include, lib, logs, share folders.
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 do this? Environment variables are crucial because:

- `JAVA_HOME` tells Hadoop where to find Java (required for all Hadoop processes)
- `HADOOP_HOME` and related variables help Hadoop locate its own files, libraries, and configurations
- `PATH` modification allows you to run Hadoop commands from anywhere without typing full paths

- Adding to `~/.bashrc` ensures these variables persist across shell sessions and reboots

Apply the environment variables:

```
source ~/.bashrc
```

Why do this? The `source` command reloads your `.bashrc` file, making the environment variables available in your current shell session immediately. Without this, you'd need to open a new terminal or log out and back in for the changes to take effect.

4. Update Hadoop Environment Configuration

Edit `/usr/local/hadoop/etc/hadoop/hadoop-env.sh` and set:

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

Why do this? This step is critical because:

- Hadoop daemons (services) run as separate processes and need to know where Java is located
- The `hadoop-env.sh` file provides environment variables specifically for Hadoop processes
- Even if JAVA_HOME is set in your shell, Hadoop services might not inherit it, leading to "JAVA_HOME not found" errors
- This is one of the most common failure points in Hadoop installations if not done correctly

Hadoop Configuration

1. Core Configuration (core-site.xml)

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://localhost: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 file? The `core-site.xml` file contains fundamental Hadoop settings:

- `fs.defaultFS` tells Hadoop which file system to use by default. For single-node mode, we use `hdfs://localhost:9000` to specify the local HDFS instance running on port 9000
- `hadoop.tmp.dir` sets the base directory for Hadoop's temporary files. This is crucial because Hadoop needs space for intermediate data during processing
- `io.file.buffer.size` optimizes I/O performance by setting an appropriate buffer size for file operations

2. HDFS Configuration (hdfs-site.xml)

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 single node setup -->
    <property>
        <name>dfs.replication</name>
        <value>1</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>localhost: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 single node
setup</description>
    </property>
</configuration>
```

Why configure these HDFS settings? These configurations are essential for HDFS operation:

- `dfs.replication` set to `1` because we only have one node. In multi-node clusters, this would be higher for data redundancy
- `dfs.namenode.name.dir` specifies where the NameNode stores its metadata (file system structure, block locations, etc.)
- `dfs.datanode.data.dir` tells DataNodes where to store actual data blocks
- `dfs.blocksize` sets the default block size to 128MB, which balances memory usage and performance
- `dfs.http.address` enables the NameNode web UI on port 9870 for monitoring and management
- `dfs.permissions` disabled for simplicity in single-node mode (in production, you'd enable this for security)

3. MapReduce Configuration (mapred-site.xml)

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>localhost: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>localhost: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/hadoop-3.3.6</value>
        <description>Environment variables for MapReduce Application
Master</description>
    </property>

    <property>
        <name>mapreduce.map.env</name>
        <value>HADOOP_MAPRED_HOME=/usr/local/hadoop/hadoop-3.3.6</value>
        <description>Environment variables for Map tasks</description>
    </property>

    <property>
        <name>mapreduce.reduce.env</name>
        <value>HADOOP_MAPRED_HOME=/usr/local/hadoop/hadoop-3.3.6</value>
        <description>Environment variables for Reduce tasks</description>
    </property>
</configuration>
```

Why configure MapReduce settings? These settings control how MapReduce jobs run:

- `mapreduce.framework.name` set to `yarn` tells MapReduce to use YARN for resource management and job scheduling
- Job history server settings enable tracking and debugging of completed jobs via web UI

- The HADOOP_MAPRED_HOME environment variables are critical - they prevent "ClassNotFoundException" errors that commonly occur when MapReduce tasks can't find Hadoop's MapReduce libraries
- Local directories provide space for intermediate MapReduce data and job-specific files

4. YARN Configuration (yarn-site.xml)

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>localhost</value>
        <description>ResourceManager hostname</description>
    </property>

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

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

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

    <property>
        <name>yarn.resourcemanager.admin.address</name>
```

```
        <value>localhost:8033</value>
        <description>ResourceManager admin address</description>
    </property>

    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>localhost: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>
```

Why configure YARN settings? YARN manages resources and schedules applications:

- `mapreduce_shuffle` service enables the critical shuffle phase that transfers data from mappers to reducers
- ResourceManager settings define how nodes communicate and where the web UI runs
- Memory and CPU settings allocate resources appropriately for single-node operation
- Log aggregation helps debug applications by collecting logs from all containers

5. Workers Configuration

Edit `/usr/local/hadoop/etc/hadoop/workers` and ensure it contains:

```
localhost
```

Why configure this file? The workers file tells Hadoop which machines should run NodeManager and DataNode services. For single-node mode, we list `localhost` so these services start on the local machine. In multi-node clusters, you'd list all worker node hostnames here.

HDFS Setup

1. Create Required Directories

```
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
```

Why create these directories? Hadoop needs specific directories to store different types of data:

- `tmp/` for temporary files during Hadoop operations
- `data/namenode/` for NameNode metadata (file system structure, block locations)
- `data/datanode/` for actual data blocks stored by DataNodes
- `data/mapred/` directories for MapReduce intermediate data and system files
- The `-p` flag creates parent directories as needed and doesn't fail if directories already exist

2. Format NameNode

```
hdfs namenode -format
```

Why format the NameNode? This is a crucial one-time setup step that:

- Initializes the NameNode's metadata storage directories
- Creates the basic file system structure
- Sets up the necessary directories and files for HDFS operation
- **Warning**: This should only be done once. Re-formatting will erase all existing HDFS data

3. Set Up Passwordless SSH (for localhost)

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
ssh localhost 'echo SSH works'
```

Why set up SSH? Hadoop uses SSH to start and manage daemons on different nodes:

- `ssh-keygen` creates an RSA key pair for authentication (empty password with `-P ''`)
- Adding the public key to `authorized_keys` enables passwordless login to localhost
- `chmod 600` sets proper permissions on the authorized_keys file (SSH requires this for security)
- The final test command verifies that passwordless SSH to localhost works correctly
- Even in single-node mode, Hadoop needs SSH to start services like DataNode and TaskTracker

Starting Hadoop Services

1. Start HDFS

```
start-dfs.sh
```

Why start HDFS first? This script starts the HDFS components in the correct order:

- Starts the NameNode (master node that manages file system metadata)
- Starts DataNodes (store actual data blocks)
- Starts SecondaryNameNode (helps with NameNode recovery)
- HDFS must be running before applications can use the file system

2. Start YARN

```
start-yarn.sh
```

Why start YARN separately? YARN manages resources and applications:

- Starts the ResourceManager (central resource scheduler)
- Starts NodeManagers (manage resources on each node)
- YARN runs on top of HDFS, so HDFS must be running first
- Separating the starts allows you to debug issues more easily

3. Verify Services

```
jps
```

Why verify with jps? This command lists all running Java processes:

- Confirms all Hadoop services started successfully
- Shows the exact process names (NameNode, DataNode, etc.)
- Helps identify which services failed to start
- Essential troubleshooting step before proceeding

Expected output:

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

Verification

1. Check HDFS Status

```
hdfs dfsadmin -report
```

Why check HDFS status? This command provides important information:

- Shows cluster health and capacity
- Lists active DataNodes and their storage
- Displays configured vs. available disk space
- Confirms HDFS is functioning properly before running jobs

2. Test HDFS Operations

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

# Create test file
echo "Hello Hadoop World
This is a test file for WordCount example
Hadoop makes big data processing easy
MapReduce is the programming model
WordCount counts words in text files" > test.txt

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

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

Why test HDFS operations? These commands verify that:

- HDFS is working correctly for file operations
- Your user has proper permissions to create directories and upload files
- The file system is accessible and functional
- Data can be stored and retrieved successfully
- Creates test data for the upcoming MapReduce job

3. Run MapReduce WordCount Example

```
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 WordCount? This is the classic "Hello World" of Hadoop:

- Tests the complete MapReduce pipeline (input → map → shuffle → reduce → output)
- Verifies YARN can schedule and execute jobs
- Confirms MapReduce libraries are properly configured
- Uses the built-in examples jar included with Hadoop
- Validates the entire Hadoop ecosystem is working together

4. View Results

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

Why check the results? This verification step confirms:

- The MapReduce job completed successfully
- Output files were created in HDFS
- The word counting logic worked correctly
- You can access and read the results
- The entire big data processing pipeline is functional

Expected output should show word counts like:

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

Service Management

Starting Services

```
# Start HDFS only
start-dfs.sh

# Start YARN only
start-yarn.sh

# Start all services
start-dfs.sh && start-yarn.sh
```

Stopping Services

```
# Stop YARN only
stop-yarn.sh

# Stop HDFS only
stop-dfs.sh

# Stop all services
stop-yarn.sh && stop-dfs.sh
```

Checking Service Status

```
# Check running Java processes
jps

# Check HDFS status
hdfs dfsadmin -report

# Check YARN applications
yarn application -list
```

Web Interfaces

After starting services, you can access the following web interfaces:

- **NameNode Web UI**: http://localhost:9870
- **ResourceManager Web UI**: http://localhost:8088
- **NodeManager Web UI**: http://localhost:8042
- **MapReduce Job History**: http://localhost:19888

Troubleshooting

Common Issues and Solutions

1. JAVA_HOME Not Found

Error: `ERROR: JAVA_HOME is not set and could not be found`

Solution:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
```

2. NameNode Format Issues

Error: NameNode fails to start or shows formatting errors

Solution:

```
# Stop all services
stop-dfs.sh && stop-yarn.sh

# Remove existing NameNode data
rm -rf /usr/local/hadoop/data/namenode/*

# Format again
hdfs namenode -format
```

3. Memory Issues

Error: Container exits with memory errors

Solution: Reduce memory allocation in `yarn-site.xml`:

```
<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>2048</value>
</property>
```

4. Port Conflicts

Error: Services fail to start due to port conflicts

Solution: Check and kill processes using conflicting ports:

```
# Check ports
netstat -tulpn | grep :8088
netstat -tulpn | grep :9870

# Kill conflicting processes
sudo kill -9 <PID>
```

5. DataNode Connection Issues

Error: DataNode fails to connect to NameNode

Solution:

```
# Stop all services
stop-dfs.sh && stop-yarn.sh

# Remove DataNode data
rm -rf /usr/local/hadoop/data/datanode/*

# Restart services
start-dfs.sh
```

Log Locations

- **NameNode logs**: `/usr/local/hadoop/hadoop-3.3.6/logs/hadoop-ubuntu-namenode-*.log`
- **DataNode logs**: `/usr/local/hadoop/hadoop-3.3.6/logs/hadoop-ubuntu-datanode-*.log`
- **ResourceManager logs**: `/usr/local/hadoop/hadoop-3.3.6/logs/yarn-ubuntu-resourcemanager-*.log`
- **NodeManager logs**: `/usr/local/hadoop/hadoop-3.3.6/logs/yarn-ubuntu-nodemanager-*.log`

Common Hadoop Commands

HDFS Commands

```
# List files
hdfs dfs -ls /path

# Create directory
hdfs dfs -mkdir /path/directory

# Copy from local to HDFS
hdfs dfs -put localfile /hdfs/path

# Copy from HDFS to local
hdfs dfs -get /hdfs/file localfile

# View file contents
hdfs dfs -cat /hdfs/file

# Remove file/directory
hdfs dfs -rm /hdfs/file
```

```
hdfs dfs -rm -r /hdfs/directory

# Check disk usage
hdfs dfs -du -h /path

# Check HDFS health
hdfs dfsadmin -report
hdfs fsck / -files -blocks -locations
```

YARN Commands

```
# List running applications
yarn application -list

# Kill application
yarn application -kill <application_id>

# List running containers
yarn container -list

# Check node status
yarn node -list

# Queue information
yarn queue -status
```

MapReduce Commands

```
# List running jobs
mapred job -list

# Kill job
mapred job -kill <job_id>

# Job history server
mr-jobhistory-daemon.sh start
mr-jobhistory-daemon.sh stop
```

Environment Variables Check

```
# Check Hadoop version
hadoop version

# Check Hadoop configuration
hadoop classpath
```

```
# Check native libraries
hadoop checknative -a
```

Maintenance

Cleaning Up HDFS

```
# Clean up old checkpoints
hdfs dfsadmin -saveNamespace

# Clean up trash
hdfs dfs -expunge
```

Backup Configuration

```
# Backup configuration files
cp -r /usr/local/hadoop/hadoop-3.3.6/etc/hadoop ~/hadoop-config-backup/

# Backup important data
hdfs dfs -copyToLocal /user /home/ubuntu/hdfs-backup
```

Security Considerations

1. **Firewall**: Configure firewall to allow Hadoop ports (8020-8040, 9000, 9870, 8088, 19888)
2. **Authentication**: In production, enable Kerberos authentication
3. **Permissions**: Enable file permissions by setting `dfs.permissions` to `true`
4. **Encryption**: Enable HDFS encryption for sensitive data

Performance Tuning

Memory Optimization

```
<!-- In yarn-site.xml -->
<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>8192</value>
</property>
```

HDFS Optimization

```
<!-- In hdfs-site.xml -->
<property>
    <name>dfs.namenode.handler.count</name>
```

```
    <value>10</value>
</property>

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

Uninstallation

If you need to uninstall Hadoop:

```
# Stop all services
stop-dfs.sh && stop-yarn.sh

# Remove Hadoop installation
sudo rm -rf /usr/local/hadoop

# Remove environment variables from ~/.bashrc
# Edit ~/.bashrc and remove Hadoop-related lines

# Remove SSH keys (optional)
rm -rf ~/.ssh/id_rsa*
```

References

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

License