### **Assignment No. 03: Installation of Hadoop in standalone mode on Ubuntu**

### **Introduction** Hadoop is a Java-based programming framework that supports the processing and storage of extremely large datasets on a cluster of inexpensive machines. It was the first major open source project in the big data playing field and is sponsored by the Apache Software Foundation.

Hadoop is comprised of four main layers:

* **Hadoop Common** is the collection of utilities and libraries that support other Hadoop modules.
* **HDFS**, which stands for Hadoop Distributed File System, is responsible for persisting data to disk.
* **YARN**, short for Yet Another Resource Negotiator, is the “operating system” for HDFS.
* **MapReduce** is the original processing model for Hadoop clusters. It distributes work within the cluster or map, then organizes and reduces the results from the nodes into a response to a query. Many other processing models are available for the 3.x version of Hadoop.

Hadoop clusters are relatively complex to set up, so the project includes a stand-alone mode which is suitable

for learning about Hadoop, performing simple operations, and debugging.

In this tutorial, we’ll install Hadoop in stand-alone mode and run one of the example example MapReduce programs it includes to verify the installation.

## **Prerequisites**

To follow this installation, we will need:

* **An Ubuntu 18.04 server with a non-root user with sudo privileges**: You can learn more about how to set up a user with these privileges in our [Initial Server Setup with Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04) guide.

Once we’ve completed this prerequisite, we’re ready to install Hadoop and its dependencies.

Before you begin, you might also like to take a look at [An Introduction to Big Data Concepts and Terminology](https://www.digitalocean.com/community/tutorials/an-introduction-to-big-data-concepts-and-terminology)

or [An Introduction to Hadoop](https://www.digitalocean.com/community/tutorials/an-introduction-to-hadoop)

## **Step 1 — Installing Java**

To get started, we’ll update our package list:

$ sudo apt update

Next, we’ll install OpenJDK, the default Java Development Kit on Ubuntu 18.04:

$ sudo apt install default-jdk

Once the installation is complete, let’s check the version.

$ java -version

Output

openjdk 10.0.1 2018-04-17

OpenJDK Runtime Environment (build 10.0.1+10-Ubuntu-3ubuntu1)

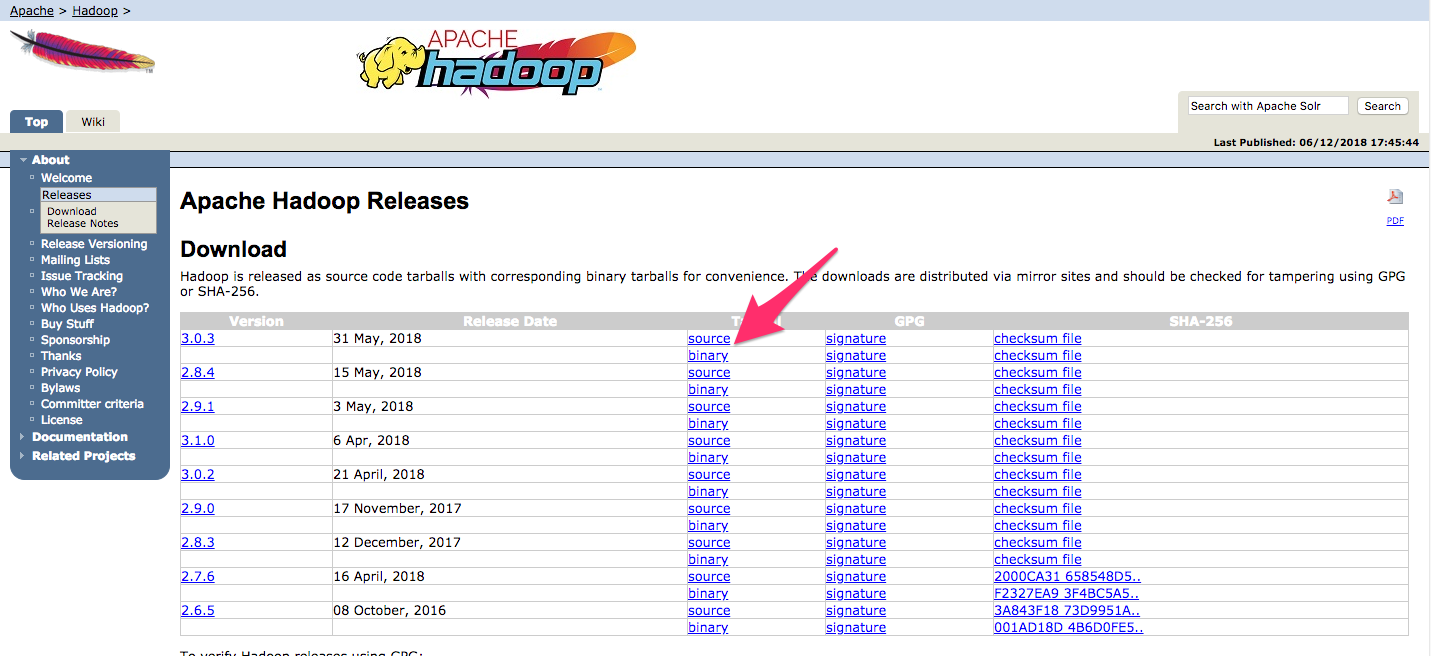
OpenJDK 64-Bit Server VM (build 10.0.1+10-Ubuntu-3ubuntu1, mixed mode)

This output verifies that OpenJDK has been successfully installed.

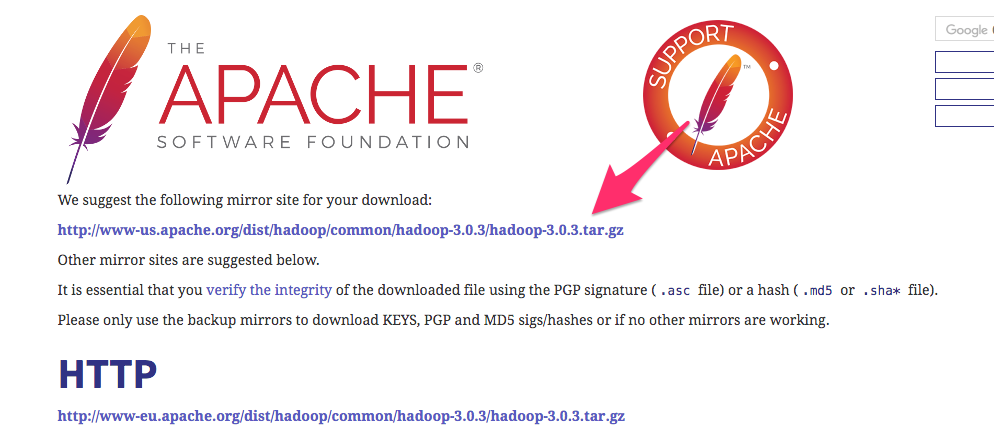
## **Step 2 — Installing Hadoop**

With Java in place, we’ll visit the [Apache Hadoop Releases page](http://hadoop.apache.org/releases.html) to find the most recent stable release.

Navigate to **binary** for the release you’d like to install. In this guide, we’ll install Hadoop 3.0.3.



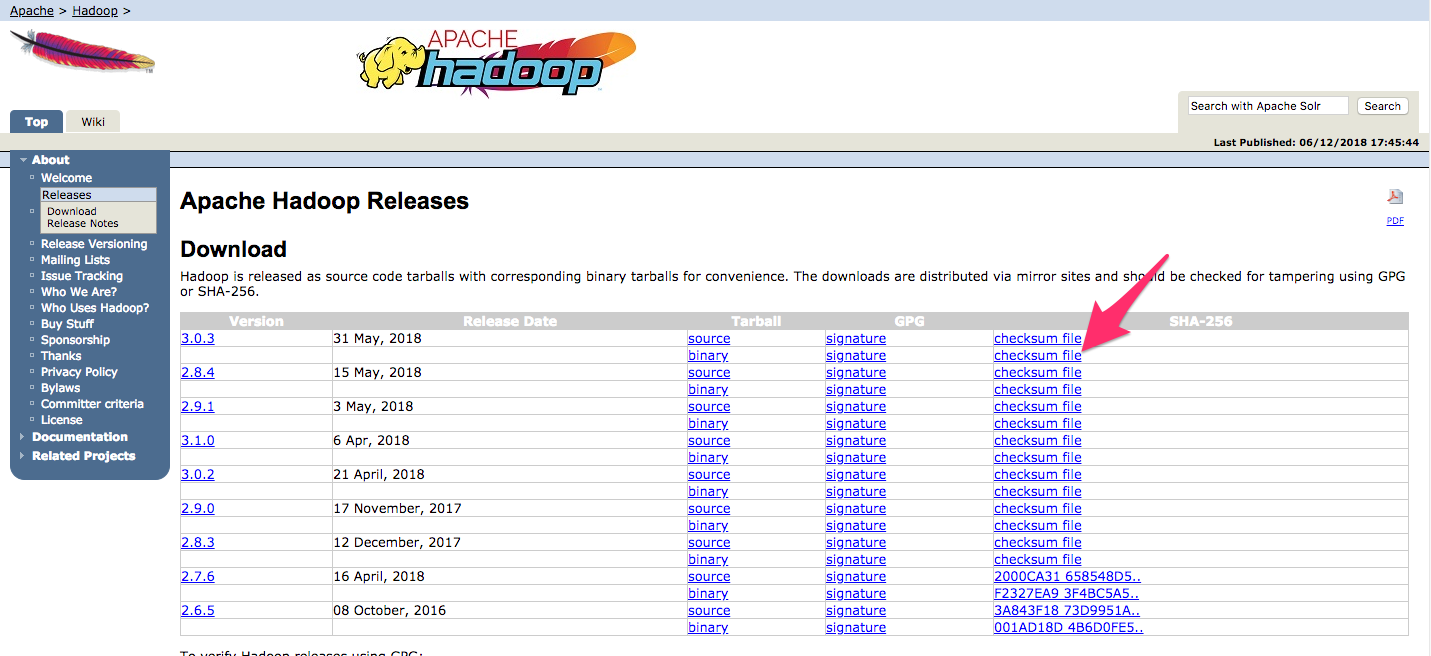
On the next page, right-click and copy the link to the release binary.



On the server, we’ll use wget to fetch it:

$ wget <http://www-us.apache.org/dist/hadoop/common/hadoop-3.0.3/hadoop-3.0.3.tar.gz>

In order to make sure that the file we downloaded hasn’t been altered, we’ll do a quick check using SHA-256. Return to the [releases page](http://hadoop.apache.org/releases.html), then right-click and copy the link to the checksum file for the release binary you downloaded:



Again, we’ll use wget on our server to download the file:

$ wget https://dist.apache.org/repos/dist/release/hadoop/common/hadoop-3.0.3/hadoop-3.0.3.tar.gz.mds

Then run the verification:

$ shasum -a 256 hadoop-3.0.3.tar.gz

Output

db96e2c0d0d5352d8984892dfac4e27c0e682d98a497b7e04ee97c3e2019277a hadoop-3.0.3.tar.gz

Compare this value with the SHA-256 value in the .mds file:

$ cat hadoop-3.0.3.tar.gz.mds

~/hadoop-3.0.3.tar.gz.mds

...

/build/source/target/artifacts/hadoop-3.0.3.tar.gz:

SHA256 = DB96E2C0 D0D5352D 8984892D FAC4E27C 0E682D98 A497B7E0 4EE97C3E 2019277A

...

You can safely ignore the difference in case and the spaces. The output of the command we ran against the

file we downloaded from the mirror should match the value in the file we downloaded from apache.org.

Now that we’ve verified that the file wasn’t corrupted or changed, we’ll use the tar command with the -x flag

to extract, -z to uncompress, -v for verbose output, and -f to specify that we’re extracting from a file. Use

tab-completion or substitute the correct version number in the command below:

$ tar -xzvf hadoop-3.0.3.tar.gz

Finally, we’ll move the extracted files into /usr/local, the appropriate place for locally installed software. Change the version number, if needed, to match the version you downloaded.

$ sudo mv hadoop-3.0.3 /usr/local/hadoop

With the software in place, we’re ready to configure its environment.

## **Step 3 — Configuring Hadoop’s Java Home**

Hadoop requires that you set the path to Java, either as an environment variable or in the Hadoop configuration

file. The path to Java, /usr/bin/java is a symlink to /etc/alternatives/java, which is in turn a symlink to

default Java binary. We will use readlink with the -f flag to follow every symlink in every part of the path,

recursively. Then, we’ll use sed to trim bin/java from the output to give us the correct value for JAVA\_HOME.

To find the default Java path

$ readlink -f /usr/bin/java | sed "s:bin/java::"

Output

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

You can copy this output to set Hadoop’s Java home to this specific version, which ensures that if the default Java changes, this value will not. Alternatively, you can use the readlink command dynamically in the file so that Hadoop will automatically use whatever Java version is set as the system default.

To begin, open hadoop-env.sh:

$ sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh

Then, choose one of the following options:

### Option 1: Set a Static Value

/usr/local/hadoop/etc/hadoop/hadoop-env.sh

. . .

#export JAVA\_HOME=${JAVA\_HOME}

export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64/

. . .

Option 2: Use Readlink to Set the Value Dynamically

/usr/local/hadoop/etc/hadoop/hadoop-env.sh

. . .

#export JAVA\_HOME=${JAVA\_HOME}

export JAVA\_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")

. . .

**Note:** With respect to Hadoop, the value of JAVA\_HOME in hadoop-env.sh overrides any values that are set

in the environment by /etc/profile or in a user’s profile.

## **Step 4 — Running Hadoop**

Now we should be able to run Hadoop:

$ /usr/local/hadoop/bin/hadoop

Output

Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]

or hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]

where CLASSNAME is a user-provided Java class

OPTIONS is none or any of:

--config dir Hadoop config directory

--debug turn on shell script debug mode

--help usage information

buildpaths attempt to add class files from build tree

hostnames list[,of,host,names] hosts to use in slave mode

hosts filename list of hosts to use in slave mode

loglevel level set the log4j level for this command

workers turn on worker mode

SUBCOMMAND is one of:

. . .

The help means we’ve successfully configured Hadoop to run in stand-alone mode. We’ll ensure that it is

functioning properly by running the example MapReduce program it ships with. To do so, create a directory

called input in our home directory and copy Hadoop’s configuration files into it to use those files as our data.

$ mkdir ~/input

$ cp /usr/local/hadoop/etc/hadoop/\*.xml ~/input

Next, we can use the following command to run the MapReduce hadoop-mapreduce-examples program, a

Java archive with several options. We’ll invoke its grep program, one of the many examples included in hadoop-mapreduce-examples, followed by the input directory, input and the output directory grep\_example.

The MapReduce grep program will count the matches of a literal word or regular expression. Finally, we’ll

supply the regular expression allowed[.]\* to find occurrences of the word allowed within or at the end of a

declarative sentence. The expression is case-sensitive, so we wouldn’t find the word if it were capitalized

at the beginning of a sentence:

/usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.3.jar grep ~/input

$ ~/grep\_example 'allowed[.]\*'

Copy

When the task completes, it provides a summary of what has been processed and errors it has encountered, but this doesn’t contain the actual results.

Output

. . .

File System Counters

FILE: Number of bytes read=1330690

FILE: Number of bytes written=3128841

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

Map-Reduce Framework

Map input records=2

Map output records=2

Map output bytes=33

Map output materialized bytes=43

Input split bytes=115

Combine input records=0

Combine output records=0

Reduce input groups=2

Reduce shuffle bytes=43

Reduce input records=2

Reduce output records=2

Spilled Records=4

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=3

Total committed heap usage (bytes)=478150656

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=147

File Output Format Counters

Bytes Written=34

**Note:** If the output directory already exists, the program will fail, and rather than seeing the summary, the output will look something like:

Output

. . .

at java.base/java.lang.reflect.Method.invoke(Method.java:564)

at org.apache.hadoop.util.RunJar.run(RunJar.java:244)

at org.apache.hadoop.util.RunJar.main(RunJar.java:158)

Results are stored in the output directory and can be checked by running cat on the output directory:

$ cat ~/grep\_example/\*

Output

19 allowed. 1 allowed

The MapReduce task found 19 occurrences of the word allowed followed by a period and one occurrence

where it was not. Running the example program has verified that our stand-alone installation is working

properly and that non-privileged users on the system can run Hadoop for exploration or debugging.

## **Conclusion**

In this tutorial, we’ve installed Hadoop in stand-alone mode and verified it by running an example program

it was provided. To learn how to write your own MapReduce programs, you might want to visit Apache Hadoop’s [MapReduce tutorial](https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html) which walks through the code behind the example. When you’re ready to set up a cluster,

see the Apache Foundation [Hadoop Cluster Setup](https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html) guide.

**Assignment 4 : File Management tasks in Hadoop**

**Objective:** To understand the commands used in hadoop for file management

**Theory :**

**1. Create a directory in HDFS at given path(s).**

Usage: hadoop fs -mkdir <paths>

Example: hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

**2. List the contents of a directory.**

Usage : hadoop fs -ls <args>

Example: hadoop fs -ls /user/saurzcode

**3. Upload and download a file in HDFS.**

*Upload***:**

**hadoop fs -put:**

Copy single src file, or multiple src files from local file system to the Hadoop data file system

Usage: hadoop fs -put <localsrc> ... <HDFS\_dest\_Path>

Example: hadoop fs -put /home/saurzcode/Samplefile.txt /user/saurzcode/dir3/

**Download: hadoop fs -get:**

Copies/Downloads files to the local file system

Usage: hadoop fs -get <hdfs\_src> <localdst>

Example: hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

**4. See contents of a file**

Same as unix cat command:

Usage:hadoop fs -cat <path[filename]>

Example: hadoop fs -cat /user/saurzcode/dir1/abc.txt

**5. Copy a file from source to destination**

This command allows multiple sources as well in which case the destination must be a directory.

Usage: hadoop fs -cp <source> <dest>

Example: hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2

**6. Copy a file from/To Local file system to HDFS copyFromLocal**

Usage: hadoop fs -copyFromLocal <localsrc> URI

Example: hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/saurzcode/abc.txt

Similar to put command, except that the source is restricted to a local file reference.

**copyToLocal**

Usage:

hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is

restricted to a local file reference.

**7. Move file from source to destination.**

Note:- Moving files across file system is not permitted.

Usage : hadoop fs -mv <src> <dest>

Example: hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2

**8. Remove a file or directory in HDFS.**

Remove files specified as arguments. Deletes directory only when it is empty

Usage : hadoop fs -rm <arg>

Example: hadoop fs -rm /user/saurzcode/dir1/abc.txt

**Recursive version of delete.**

Usage : hadoop fs -rmr <arg>

Example: hadoop fs -rmr /user/saurzcode/

**9. Display the last few lines of a file.**

Similar to tail command in Unix.

Usage : hadoop fs -tail <path[filename]>

Example: hadoop fs -tail /user/saurzcode/dir1/abc.txt

**10. Display the aggregate length of a file.**

Usage : hadoop fs -du <path>

Example: hadoop fs -du /user/saurzcode/dir1/abc.txt

**Output :** Screenshots of the command executed

**Conclusion : *( Students should mention what was their learning from this assignment in one statement )***