

How To Install Spark On Ubuntu

Introduction

Apache Spark is a framework used in cluster computing environments for **analyzing big data**. This platform became widely popular due to its ease of use and the improved data processing speeds over Hadoop.

Apache Spark is able to distribute a workload across a group of computers in a cluster to more effectively process large sets of data. This **open-source engine** supports a wide array of programming languages. This includes Java, Scala, Python, and R.

In this tutorial, you will learn **how to install Spark on an Ubuntu machine**. The guide will show you how to start a master and slave server and how to load Scala and Python shells. It also provides the most important Spark commands.

Prerequisites

- An Ubuntu system.
- Access to a terminal or command line.
- A user with sudo or **root** permissions.

Install Packages Required for Spark

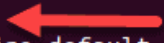
Before downloading and setting up Spark, you need to install necessary dependencies. This step includes installing the following packages:

- JDK
- Scala
- Git

Open a terminal window and run the following command to install all three packages at once:

```
sudo apt install default-jdk scala git -y
```

You will see which packages will be installed.

```
test@ubuntu1: ~  
File Edit View Search Terminal Help  
test@ubuntu1:~$ sudo apt install default-jdk scala git -y  
[sudo] password for test:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
git is already the newest version (1:2.17.1-1ubuntu0.5).  
The following packages were automatically installed and are no longer required:  
  liballegro4.4 libdevil1c2 libevent-core-2.1-6 libllvm7 liblua5.1-2  
  liblua5.1-common libmng2 libmodplug1 libopenal-data libopenal1  
  libphysfs1 libSDL1.2debian libSDL2-2.0-0 vim-runtime  
Use 'sudo apt autoremove' to remove them.  
The following additional packages will be installed:   
  ca-certificates-java default-jdk-headless default-jre default-jre-headless  
  fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni  
  libhawtjni-runtime-java libice-dev libjansi-java libjansi-native-java  
  libjline2-java libpthread-stubs0-dev libsm-dev libx11-dev libx11-doc  
  libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk  
  openjdk-11-jdk-headless openjdk-11-jre openjdk-11-jre-headless  
  scala-library scala-parser-combinators scala-xml x11proto-core-dev  
  x11proto-dev xorg-sgml-doctools xtrans-dev
```

Once the process completes, **verify the installed dependencies** by running these commands:

```
java -version; javac -version; scala -version; git --version
```

```
test@ubuntu1:~$ java -version; javac -version; scala -version; git --version  
openjdk version "11.0.6" 2020-01-14  
OpenJDK Runtime Environment (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1)  
OpenJDK 64-Bit Server VM (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1, mixed mode, sharing)  
javac 11.0.6  
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL  
git version 2.17.1
```

The output prints the versions if the installation completed successfully for all packages.

Download and Set Up Spark on Ubuntu

Now, **you need to download the version of Spark you want** from their website. We will go for *Spark 3.0.1* with *Hadoop 2.7* as it is the latest version at the time of writing this article.

Use the **wget** command and the direct link to download the Spark archive:

```
wget https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz
```

When the download completes, you will see the *saved* message.

```
goran@goran-test: ~  
goran@goran-test:~$ wget https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz  
--2020-09-14 19:21:23-- https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz  
Resolving downloads.apache.org (downloads.apache.org)... 88.99.95.219, 2a01:4f8:10a:201a::2  
Connecting to downloads.apache.org (downloads.apache.org)|88.99.95.219|:443... connected  
.  
HTTP request sent, awaiting response... 200 OK  
Length: 219929956 (210M) [application/x-gzip]  
Saving to: 'spark-3.0.1-bin-hadoop2.7.tgz.1'  
  
spark-3.0.1-bin-hadoo 100%[=====] 209.74M  2.78MB/s in 53s  
  
2020-09-14 19:22:16 (3.96 MB/s) - 'spark-3.0.1-bin-hadoop2.7.tgz.1' saved [219929956/219929956]
```

Note: If the URL does not work, please go to the [Apache Spark](#) download page to check for the latest version. Remember to replace the Spark version number in the subsequent commands if you change the download URL.

Now, extract the saved archive using the **tar** command:

```
tar xvf spark-*
```

Let the process complete. The output shows the files that are being unpacked from the archive.

Finally, move the unpacked directory *spark-3.0.1-bin-hadoop2.7* to the ***opt/spark*** directory.

Use the **mv** command to do so:

```
sudo mv spark-3.0.1-bin-hadoop2.7 /opt/spark
```

The terminal returns no response if it successfully moves the directory. If you mistype the name, you will get a message similar to:

```
mv: cannot stat 'spark-3.0.1-bin-hadoop2.7': No such file or directory.
```

Configure Spark Environment

Before starting a master server, you need to configure environment variables. There are a few Spark home paths you need to add to the user profile.

Use the **echo** command to add these three lines to *.profile*:

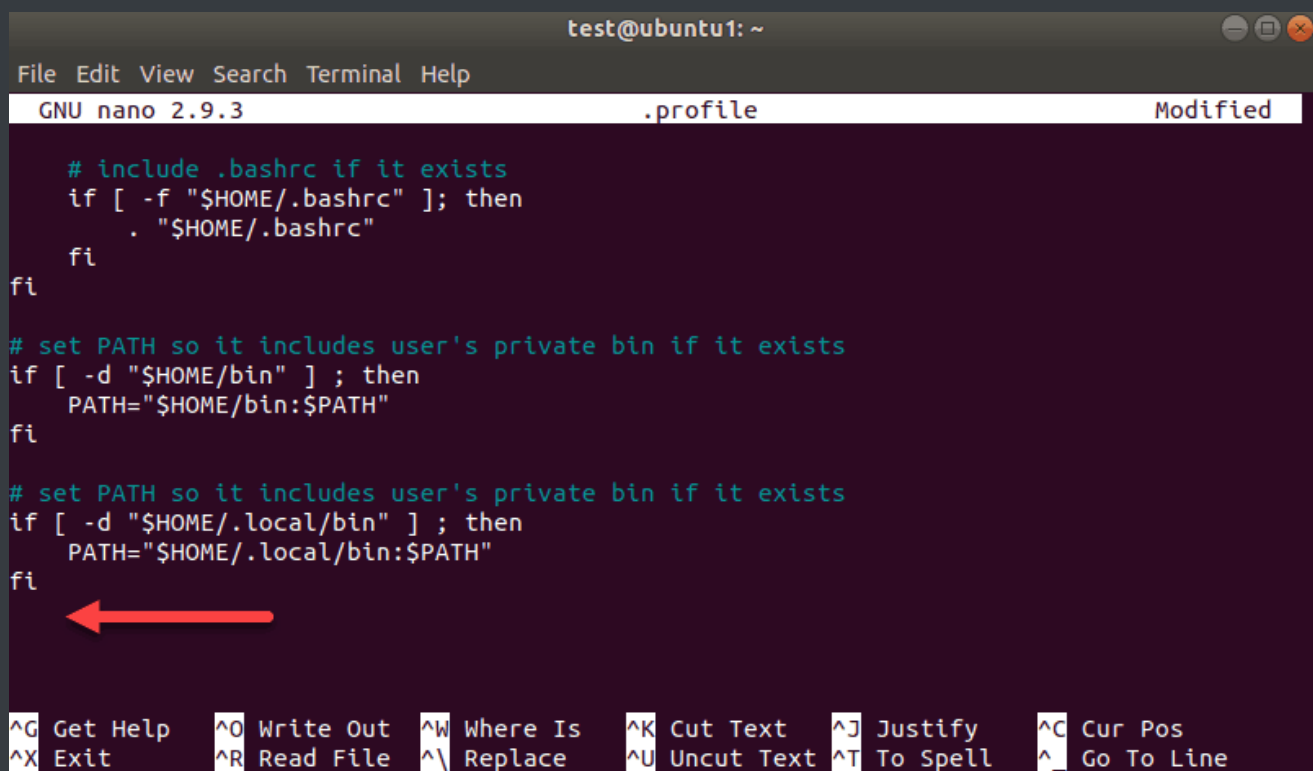
```
echo "export SPARK_HOME=/opt/spark" >> ~/.profile
echo "export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin" >> ~/.profile
echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.profile
```

You can also add the export paths by editing the *.profile* file in the editor of your choice, such as nano or vim.

For example, to use nano, enter:

```
nano .profile
```

When the profile loads, scroll to the bottom of the file.



```
test@ubuntu1: ~
File Edit View Search Terminal Help
GNU nano 2.9.3 .profile Modified

# include .bashrc if it exists
if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

←

^G Get Help	^O Write Out	^W Where Is	^K Cut Text	^J Justify	^C Cur Pos
^X Exit	^R Read File	^_\ Replace	^U Uncut Text	^T To Spell	^_ Go To Line

Then, add these three lines:

```
export SPARK_HOME=/opt/spark

export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

export PYSPARK_PYTHON=/usr/bin/python3
```

Exit and save changes when prompted.

When you finish adding the paths, load the *.profile* file in the command line by typing:

```
source ~/.profile
```

Start Standalone Spark Master Server

Now that you have completed configuring your environment for Spark, you can start a master server.

In the terminal, type:

```
start-master.sh
```

To view the Spark Web user interface, open a web browser and enter the localhost IP address on port 8080.

```
http://127.0.0.1:8080/
```

The page shows your **Spark URL**, status information for workers, hardware resource utilization, etc.

Spark Master at spark://ubuntu1:7077

URL: spark://ubuntu1:7077
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

▼ Workers (0)

Worker Id	Address	State	Cores	Memory
-----------	---------	-------	-------	--------

▼ Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

▼ Completed Applications (0)

Application ID	Name	Cores	Memory per Executor ▼	Submitted Time	User	State	Duration
----------------	------	-------	-----------------------	----------------	------	-------	----------

The URL for Spark Master is the name of your device on port 8080. In our case, this is ***ubuntu1:8080***. So, there are three possible ways to load Spark Master's Web UI:

1. 127.0.0.1:8080
2. localhost:8080
3. *deviceName*:8080

Note: Learn how to automate the deployment of Spark clusters on Ubuntu servers by reading our [Automated Deployment Of Spark Cluster On Bare Metal Cloud](#) article.

Start Spark Slave Server (Start a Worker Process)

In this single-server, standalone setup, we will start one slave server along with the master server.

To do so, run the following command in this format:

```
start-slave.sh spark://master:port
```


The **master** in the command can be an IP or hostname.

In our case it is **ubuntu1** :

```
start-slave.sh spark://ubuntu1:7077
```

```
test@ubuntu1:~$ start-slave.sh spark://ubuntu1:7077
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-test-or
g.apache.spark.deploy.worker.Worker-1-ubuntu1.out
test@ubuntu1:~$
```



Now that a worker is up and running, if you reload Spark Master's Web UI, you should see it on the list:

 **Spark Master at spark://ubuntu1:7077**

URL: spark://ubuntu1:7077
Alive Workers: 1
Cores in use: 2 Total, 0 Used
Memory in use: 1024.0 MB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

▼ Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20200331204050-10.0.2.15-46309	10.0.2.15:46309	ALIVE	2 (0 Used)	1024.0 MB (0.0 B Used)



Specify Resource Allocation for Workers

The default setting when starting a worker on a machine is to use all available CPU cores. You can specify the number of cores by passing the **-c** flag to the **start-slave** command.

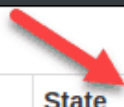
For example, to start a worker and assign only **one CPU core** to it, enter this command:

```
start-slave.sh -c 1 spark://ubuntu1:7077
```

Reload Spark Master's Web UI to confirm the worker's configuration.

▼ Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20200401122203-10.0.2.15-33497	10.0.2.15:33497	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)



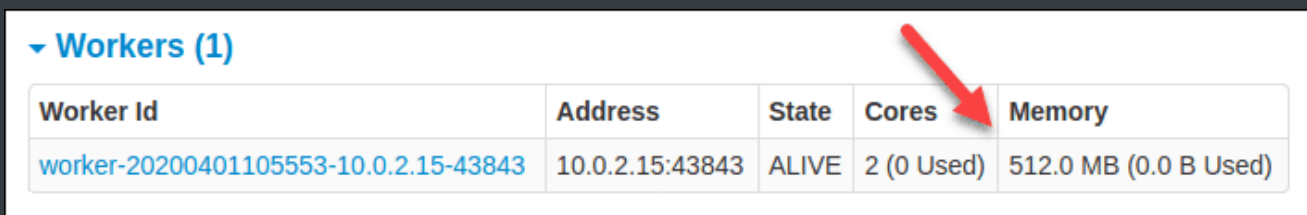
Similarly, you can assign a specific amount of memory when starting a worker. The default setting is to use whatever amount of RAM your machine has, minus 1GB.

To start a worker and assign it a specific amount of memory, add the `-m` option and a number. For gigabytes, use **G** and for megabytes, use **M**.

For example, to start a worker with 512MB of memory, enter this command:

```
start-slave.sh -m 512M spark://ubuntu1:7077
```

Reload the Spark Master Web UI to view the worker's status and confirm the configuration.



Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20200401105553-10.0.2.15-43843	10.0.2.15:43843	ALIVE	2 (0 Used)	512.0 MB (0.0 B Used)

Test Spark Shell


After you finish the configuration and start the master and slave server, test if the Spark shell works.

Load the shell by entering:

```
spark-shell
```

You should get a screen with notifications and Spark information. Scala is the default interface, so that shell loads when you run *spark-shell*.


The ending of the output looks like this for the version we are using at the time of writing this guide:



version 2.4.5

Test Python in Spark

Make sure you quit Scala and then run this command:



version 2.4.5

Basic Commands to Start and Stop Master Server and Workers

Below are the basic commands for starting and stopping the Apache Spark master server and workers. Since this setup is only for one machine, the scripts you run default to the localhost.

To **start a master server** instance on the current machine, run the command we used earlier in the guide:

```
start-master.sh
```

To **stop the master** instance started by executing the script above, run:

```
stop-master.sh
```

To **stop a running worker** process, enter this command:

```
stop-slave.sh
```

The Spark Master page, in this case, shows the worker status as DEAD.

▼ Workers (2)				
Worker Id	Address	State	Cores	Memory
worker-20200331183244-10.0.2.15-45371	10.0.2.15:45371	DEAD	2 (0 Used)	1024.0 MB (0.0 B Used)
worker-20200331203427-10.0.2.15-37971	10.0.2.15:37971	ALIVE	2 (0 Used)	1024.0 MB (0.0 B Used)

You can **start both master and server** instances by using the start-all command:

```
start-all.sh
```

Similarly, you **can stop all instances** by using the following command:

```
stop-all.sh
```

Conclusion

This tutorial showed you **how to install Spark on an Ubuntu machine**, as well as the necessary dependencies.

The setup in this guide enables you to perform basic tests before you start configuring a Spark cluster and performing advanced actions.