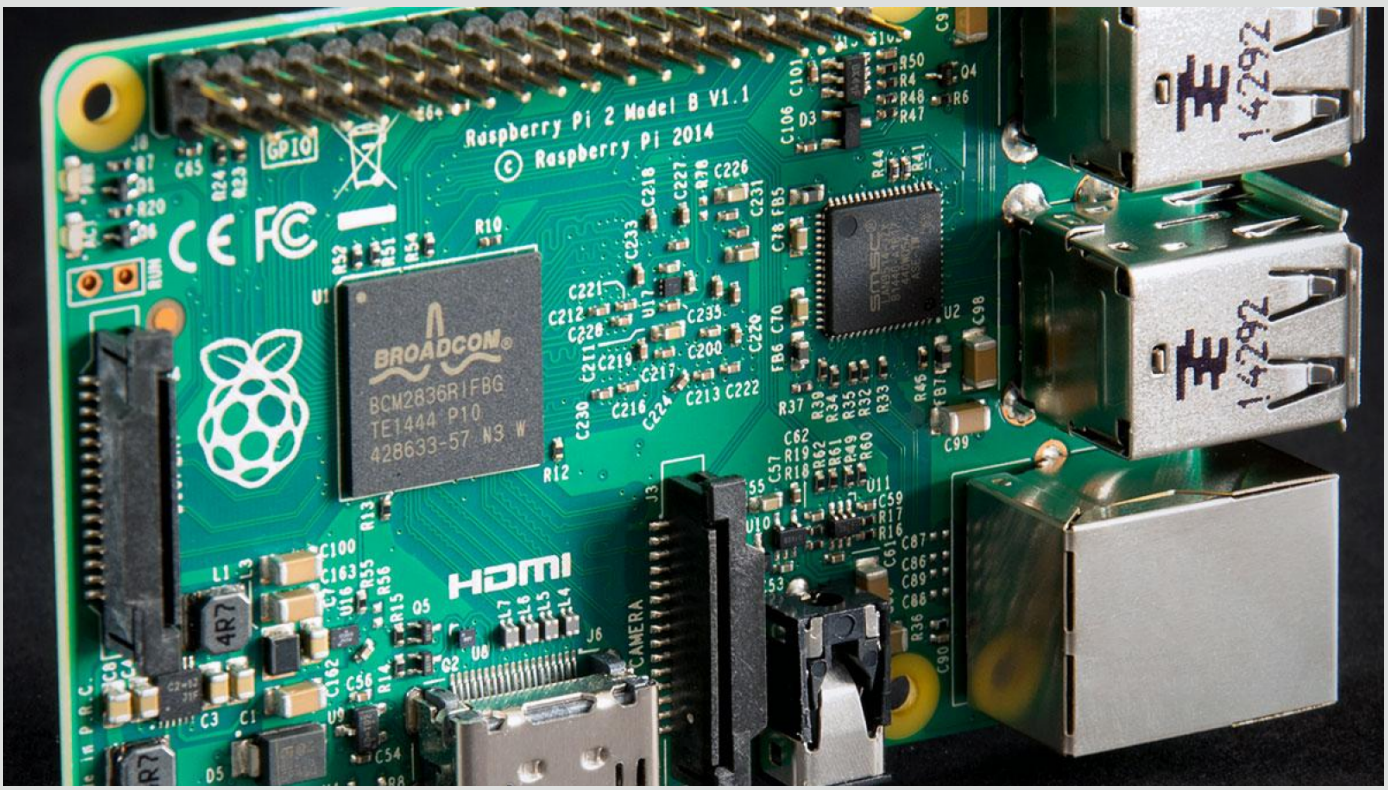
# Building a Hadoop cluster with Raspberry Pi

Installing, configuring and testing a distributed storage and processing cluster with single-board computers.

AlanVerdugo

Created on March 23, 2016 / *Modified on July 28, 2016*






**Contents**

# Overview

Skill Level: Any Skill Level

The tutorial was written as detailed as reasonably possible. Basic Linux and networking skills are needed.

# Ingredients

- Raspberry Pi 2 (two minimum)
- Micro SD cards, 8GB minimum (one for each Raspberry Pi)
- Wireless adapters or means to build a wired network

# Step-by-step

## 1 Introduction

Hadoop has great potential and is one of the best known projects for big data. In this tutorial, we will install and configure a Hadoop cluster using Raspberries. Our cluster will consists on twelve nodes (one master and eleven slaves).

Since Raspberries have a low price and Hadoop is open source, together they offer a great opportunity for anybody who wants to start working with big data. We will be configuring the cluster from a Linux workstation, we will be working exclusively with Linux commands. The cluster can be built with other families of operating systems, and the general steps in this tutorial would remain to be useful, but the specific commands would be different.

## 2 Download and install an Operating System into the micro SD cards.

Any OS that is compatible with Hadoop will work. In our case, we selected Raspbian Jessy Lite as our OS. It is a lightweight OS, designed specifically for Raspberries. It can be downloaded from the official Raspberry Pi Foundation website: https://www.raspberrypi.org/downloads/raspbian/

Once we have downloaded our .img file, we can install it into a micro SD card. This process will overwrite any data that the SD card may have, so it is recommended to use an empty card.

There are many ways to install the OS into the SD card. We will use the dd Linux command:

```
sudo dd if=2016-02-26-raspbian-jessie-lite.img of=/dev/sdd
```

The **if** parameter is the image file that we downloaded, The **of** parameter is the mount point of our micro SD card.

We will repeat this installation in every micro SD card.

Once that is done, you can insert the micro SD cards into the Raspberries and login into them via SSH as the default user “pi” using the password “raspberry”.

You may need to expand the filesystem in order to use the full capacity of the micro SD cards. If this is the case, login into the Raspberries with SSH and run:

```
sudo raspi-config
```

Then select “Expand Filesystem”. After this is done, you will need to reboot the Raspberry. You can do it like this:

```
shutdown -r now
```

## 3 Create the installation and HDFS directories

We need a installation directory for Hadoop. We also need a directory where the HDFS (Hadoop Distributed File System) will place its files. In our example, we selected **/opt/hadoop/** as the installation directory and **/opt/hadoop_tmp/** as the path that will be used by HDFS. So let’s create those directories

```
sudo mkdir /opt/hadoop/
sudo mkdir /opt/hadoop_tmp/
```

Again, make sure to do this in all the Raspberries. Take note of the HDFS directory, we will need it later while editing the configuration files.

## 4 Install Java

Since Hadoop requires it, Java needs to be installed in all the nodes. Hadoop 2.7 and later require Java 7. Hadoop 2.6 still supports Java 6. Install the version compatible with your desired Hadoop environment. In Raspbian, to install Oracle Java 8 we would do this:

```
sudo apt-get update
sudo apt-get install oracle-java8-jdk
```

Verify that it was installed correctly:

```
java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) Client VM (build 25.65-b01, mixed mode)
```

## 5 Setup connectivity.

Depending on your environment, this step could vary. If you are building the cluster on a local wifi network you will need a wireless adapter for each Raspberry (unless you are using Raspberry Pi 3, which has integrated wireless). Otherwise you may need to setup a wired network. What we need is that every Raspberry has its own IP address and that they can connect to each other without problems. This tutorial will not cover how to setup wireless connectivity in the Raspberries, but this is easy to do and there are many tutorials explaining how to do it. The official documentation is a good example: https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

To make our job easier, we will use an alias for each node from now on. Make a list of all the IP addressess and add them to the **/etc/hosts** file in every Raspberry. In our case, it looks like this:

```
9.42.157.175    master
9.42.157.182    slave-01
9.42.157.186    slave-02
9.42.157.187    slave-03
```

```
9.42.157.194    slave-04
9.42.157.204    slave-05
9.42.157.210    slave-06
9.42.157.233    slave-07
9.42.157.237    slave-08
9.42.157.239    slave-09
9.42.157.241    slave-10
9.42.157.242    slave-11
```

You can designate any Raspberry as the master node, just make sure to have a proper list in order to avoid confusion.

## 6 Generate and replicate SSH keys.

We need all the Raspberies to be able to communicate between each other without prompting for passwords. For that, we will generate SSH keys in all of them.

```
ssh-keygen -t rsa -b 4096 -C your_email@example.com
```

And then replicate the keys to all the Raspberies.

```
ssh-copy-id user@master
ssh-copy-id user@slave-01
ssh-copy-id user@slave-02
ssh-copy-id user@slave-03
ssh-copy-id user@slave-04
ssh-copy-id user@slave-05
ssh-copy-id user@slave-06
ssh-copy-id user@slave-07
ssh-copy-id user@slave-08
ssh-copy-id user@slave-09
ssh-copy-id user@slave-10
ssh-copy-id user@slave-11
```

After you are done, *every* Raspberry should have *all* the keys. This is the step where most people make mistakes, and most of the problems when starting the cluster come from an erroneous configuration of the SSH keys. So take your time to verify that all the nodes can connect to each other.

## 7 Add environment variables.

Now, we will add some Hadoop environment variables to our **~/.bashrc** file. Change the paths according to your environment.

```
# -- HADOOP ENVIRONMENT VARIABLES START -- #
export HADOOP_HOME=/opt/hadoop/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
# -- HADOOP ENVIRONMENT VARIABLES END -- #
export JAVA_HOME=/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/jre
```

If you installed Hadoop in a different path, make sure to change **HADOOP_HOME**. Make sure to add **JAVA_HOME** as well, which may be different in your environment.

## 8 Install Hadoop in the namenode

The namenode is our master node. The slave nodes are refered as datanodes. We will install Hadoop on our namenode and perform some configurations, then, we will work on the datanodes. Since this is the basic configuration, this and the following steps will be done only in the namenode. Then we will copy this basic configuration to the remaining 11 nodes, and then we will perform specific customizations.

In the releases page (https://hadoop.apache.org/releases.html) of the Hadoop site you can see which is the newest Hadoop version.

Go to the installation directory that we created in the second step. Select a Hadoop version from the download page and get the URL of the tarball. The following commands will download a Hadoop package and uncompress it.

cd /opt/hadoop/
sudo wget http://www-us.apache.org/dist/hadoop/common/hadoop-2.6.4/hadoop-2.6.4.tar.gz
sudo tar xvf hadoop-2.6.4.tar.gz
mv hadoop-2.6.4 hadoop

## 9 Add the master and slaves files

This step should only be done in the Master node. We must add two files in **$HADOOP_HOME/hadoop/etc/**, one of them will be named **master** and the other will be named **slaves**,

In this new **master** file we only need to add the hostname or alias of the master node. So in our case this file will only contain one line:

```
master
```

In the **slaves** file we will only add our slaves nodes, one alias per line. In our case:

```
slave-01
slave-02
slave-03
slave-04
slave-05
slave-06
slave-07
slave-08
slave-09
slave-10
slave-11
```

## 10 Setup HDFS

A couple of directories need to be created in our HDFS directory. In the namenode. go to the HDFS directory (**/opt/hadoop_tmp/** in our case) and create a directory named “hdfs”, then, inside that directory create another directory named “namenode”.

```
sudo mkdir /opt/hadoop_tmp/hdfs
sudo mkdir /opt/hadoop_tmp/hdfs/namenode
```

Next, we will go to every slave node, and do the same, except that the newest directory will be named “datanode”:

```
sudo mkdir /opt/hadoop_tmp/hdfs
sudo mkdir /opt/hadoop_tmp/hdfs/datanode
```

## 11 Copy the basic configuration to the slave nodes

The previous steps for the Hadoop installation were done only in the master node. Now we will take those files and copy them to the remaining Raspberries.

On the master node, run the following commands in order to copy the basic setup to the slaves (you can also do this with scp):

```
rsync -avxP <$HADOOP_HOME> <hadoop_user>@<slave-hostname>:<$HADOOP_HOI
```

In our case, it would be something like this:

```
rsync -avxP /opt/hadoop/ user@slave-01:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-02:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-03:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-04:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-05:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-06:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-07:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-08:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-09:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-10:/opt/hadoop/
rsync -avxP /opt/hadoop/ user@slave-11:/opt/hadoop/
```

## 12 Edit the configuration files

There are several XML files that need to be modified according to our architecture. These changes need to be done in the Master and in all the Slave nodes.

Update **core-site.xml** (located in **$HADOOP_HOME/hadoop/etc/hadoop/core-site.xml**), we will change “localhost” to our master node hostname, IP, or alias. In our case, it would be like this:

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://master:9000/</value>
    </property>
    <property>
        <name>fs.default.FS</name>
        <value>hdfs://master:9000/</value>
    </property>
</configuration>
```

Update **hdfs-site.xml** (located in **$HADOOP_HOME/hadoop/etc/hadoop/hdfs-site.xml**), we will change the replication factor to the number of datanodes you have, and specify the

datanode and namenodes directories (which we created in the previous step), as well as add an http address:

```
<configuration>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>/opt/hadoop_tmp/hdfs/datanode</value>
        <final>true</final>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/opt/hadoop_tmp/hdfs/namenode</value>
        <final>true</final>
    </property>
    <property>
        <name>dfs.namenode.http-address</name>
        <value>master:50070</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>11</value>
    </property>
</configuration>
```

Update **yarn-site.xml** (located in **$HADOOP_HOME/hadoop/etc/hadoop/yarn-site.xml**). There will be 3 properties that we need to update with our master node hostname or alias:

```
<configuration>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>master:8025</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>master:8035</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>master:8050</value>
    </property>
</configuration>
```

Update **mapred-site.xml** (located in **$HADOOP_HOME/hadoop/etc/hadoop/mapred-site.xml**), we will add the following properties:

```
<configuration>
    <property>
        <name>mapreduce.job.tracker</name>
        <value>master:5431</value>
    </property>
    <property>
        <name>mapred.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

## 13 Format the namenode and start the services

Go to the namenode, in **$HADOOP_HOME/bin/** execute the following:

```
./hdfs namenode -format
```

That will format the master as a proper namenode. Finally, start the services. Previously, Hadoop used the start-all.sh script for this, but it has been deprecated and the recommended method now is to use the start-<service>.sh scripts individually. In each node, go to **$HADOOP_HOME/sbin/** and execute the following:

```
./start-yarn.sh
./start-dfs.sh
```

The most common problem here is lack connection between the nodes. If you see any errors while starting the nodes, check the SSH keys.

## 14 Verify the cluster

There are several ways to confirm that everything is running properly, for example, you can point your browser to http://master:50070 (the http address configured earlier) or to http://master:8088. There you can see status reports of the cluster. Alternatively, you can also check the Java processes in the nodes. In the master node you should see at least 3 processes: NameNode, SecondaryNameNode and ResourceManager. In the slaves you should see only two: NodeManager and DataNode. My favourite way is to use the hdfs report command. This is the output report of my configuration:

```
/opt/hadoop/hadoop/bin/hdfs dfsadmin -report
Configured Capacity: 182240477184 (169.72 GB)
Present Capacity: 153073315840 (142.56 GB)
DFS Remaining: 153073020928 (142.56 GB)
DFS Used: 294912 (288 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
Live datanodes (12):

Name: 9.42.157.210:50010 (slave-06)
Hostname: Pi007-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2411073536 (2.25 GB)
DFS Remaining: 12775608320 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.204:50010 (slave-05)
Hostname: Pi006-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2411417600 (2.25 GB)
DFS Remaining: 12775264256 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
```

```
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.237:50010 (slave-08)
Hostname: Pi009-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2410917888 (2.25 GB)
DFS Remaining: 12775763968 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.186:50010 (slave-02)
Hostname: Pi003-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2411122688 (2.25 GB)
DFS Remaining: 12775559168 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.187:50010 (slave-03)
Hostname: Pi004-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
```

```
DFS Used: 24576 (24 KB)
Non DFS Used: 2410520576 (2.24 GB)
DFS Remaining: 12776161280 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.13%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.175:50010 (master)
Hostname: Pi001-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2626191360 (2.45 GB)
DFS Remaining: 12560490496 (11.70 GB)
DFS Used%: 0.00%
DFS Remaining%: 82.71%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.182:50010 (slave-01)
Hostname: Pi002-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2409967616 (2.24 GB)
DFS Remaining: 12776714240 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.13%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
```

```
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.242:50010 (slave-11)
Hostname: Pi012-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2410094592 (2.24 GB)
DFS Remaining: 12776587264 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.13%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.241:50010 (slave-10)
Hostname: Pi011-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2409664512 (2.24 GB)
DFS Remaining: 12777017344 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.13%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.233:50010 (slave-07)
Hostname: Pi008-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
```

```
Non DFS Used: 2410676224 (2.25 GB)
DFS Remaining: 12776005632 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.13%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.194:50010 (slave-04)
Hostname: Pi005-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2410934272 (2.25 GB)
DFS Remaining: 12775747584 (11.90 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016

Name: 9.42.157.239:50010 (slave-09)
Hostname: Pi010-rtp
Decommission Status : Normal
Configured Capacity: 15186706432 (14.14 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2434580480 (2.27 GB)
DFS Remaining: 12752101376 (11.88 GB)
DFS Used%: 0.00%
DFS Remaining%: 83.97%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
```

```
Xceivers: 1
Last contact: Tue Mar 22 21:10:30 EDT 2016
```

You can see that HDFS takes the space available in each node and combines it as one single file system.

Congratulations! At this point, Hadoop is configured and running. You can stop here, but I would prefer to run a few tests, just to make sure everything is working correctly. We will upload some files to the HDFS.

First, check the space usage:

```
/opt/hadoop/hadoop/bin/hadoop fs -df -h
Filesystem Size Used Available Use%
hdfs://master:9000 169.7 G 10.3 G 132.2 G 6%
```

Create a test directory:

```
/opt/hadoop/hadoop/bin/hadoop fs -mkdir hdfs://master:9000/testdir/
```

I have a test file in the local file system (specifically, in **/tmp**) we will upload that file into HDFS, inside the directory we just created:

```
/opt/hadoop/hadoop/bin/hadoop fs -copyFromLocal /tmp/testFile.txt hdfs
```

We can list the files in HDFS and confirm that the file was correctly uploaded:

```
/opt/hadoop/hadoop/bin/hadoop fs -ls hdfs://master:9000/
Found 1 items
drwxr-xr-x - root supergroup 0 2016-03-23 13:36 hdfs://master:9000/tes

/opt/hadoop/hadoop/bin/hadoop fs -ls hdfs://master:9000/testdir/
Found 1 items
-rw-r--r-- 11 root supergroup 542456 2016-03-23 13:36 hdfs://master:9(
```

For every file and directory in the HDFS, we can also check things like the replication factor, missing replicas, number of racks, file size and many other things. Here, we will check the status of our test directory and our test file:

```
/opt/hadoop/hadoop/bin/hdfs fsck /testdir/ -files -blocks -racks
Connecting to namenode via http://master:50070
FSCK started by root (auth:SIMPLE) from /9.42.157.175 for path /testd
/testdir/ <dir>
/testdir/testFile.txt 542456 bytes, 1 block(s): OK
0. BP-153620240-127.0.1.1-1457972366593:blk_1073741834_1010 len=54245
[/default-rack/9.42.157.175:50010,
/default-rack/9.42.157.204:50010,
/default-rack/9.42.157.233:50010,
/default-rack/9.42.157.194:50010,
/default-rack/9.42.157.239:50010,
/default-rack/9.42.157.187:50010,
/default-rack/9.42.157.237:50010,
/default-rack/9.42.157.182:50010,
/default-rack/9.42.157.210:50010,
/default-rack/9.42.157.186:50010,
/default-rack/9.42.157.242:50010]
Status: HEALTHY
 Total size: 542456 B
 Total dirs: 1
 Total files: 1
 Total symlinks: 0
 Total blocks (validated): 1 (avg. block size 542456 B)
 Minimally replicated blocks: 1 (100.0 %)
 Over-replicated blocks: 0 (0.0 %)
 Under-replicated blocks: 0 (0.0 %)
 Mis-replicated blocks: 0 (0.0 %)
 Default replication factor: 11
 Average block replication: 11.0
 Corrupt blocks: 0
 Missing replicas: 0 (0.0 %)
 Number of data-nodes: 12
 Number of racks: 1
FSCK ended at Wed Mar 23 13:48:04 EDT 2016 in 16 milliseconds

The filesystem under path '/testdir/' is HEALTHY
```

Since this is a very small text file (only 529KB), and our block size is 128MB there is only need for one block to store it. However, if we upload or create a bigger file in HDFS we can see that it needs more blocks, and these blocks are distributed in the cluster following our replication factor parameter. Take, for example, a 1GB file that I created using the Teragen sample program bundled with Hadoop (the following output was abridged for brevity reasons):

```
/opt/hadoop/hadoop/bin/hdfs fsck /teraInput/ -files -blocks -racks
Connecting to namenode via http://master:50070
FSCK started by root (auth:SIMPLE) from /9.42.157.175 for path /teraI
/teraInput/ <dir>
/teraInput/_SUCCESS 0 bytes, 0 block(s): OK

/teraInput/part-m-00000 1000000000 bytes, 8 block(s): OK
0. BP-153620240-127.0.1.1-1457972366593:blk_1073741825_1001 len=13421
Status: HEALTHY
 Total size: 1000000000 B
 Total dirs: 1
 Total files: 2
 Total symlinks: 0
 Total blocks (validated): 8 (avg. block size 125000000 B)
 Minimally replicated blocks: 8 (100.0 %)
 Over-replicated blocks: 0 (0.0 %)
 Under-replicated blocks: 0 (0.0 %)
 Mis-replicated blocks: 0 (0.0 %)
 Default replication factor: 11
 Average block replication: 11.0
 Corrupt blocks: 0
 Missing replicas: 0 (0.0 %)
 Number of data-nodes: 12
 Number of racks: 1
FSCK ended at Wed Mar 23 13:59:53 EDT 2016 in 13 milliseconds

The filesystem under path '/teraInput/' is HEALTHY
```

## 15 Conclusion

Hadoop is a very powerful tool for big data. In this tutorial we installed and configured it, and also worked a little bit with HDFS. However, Hadoop offers many more possibilities, for example, distributed computing with MapReduce, databases like Hive and many other projects. Also, this is a small test cluster that is only storing small files. Clusters consisting of hundreds of nodes, storing terabytes or petabytes of data, are common. In fact, big clusters like those are where Hadoop and other distributed computing platforms really show how useful they can be.

Installing and configuring Hadoop is not difficult, but we always need to pay very close attention to many aspects, otherwise we will get obscure errors that could be difficult to troubleshoot. Hopefully, this recipe will help those who want to learn about big data technologies.

Doing all this work in the Raspberry Pi platform did not offer any extra difficulty. In fact, it was as easy as working on a “real” server. Obviously, the tiny Raspberries do not offer the same power as more expensive solutions, but building a cluster with Raspberries is *much* more affordable, and they are generally easier to get, so they are the perfect solution for startups, students, small teams, and projects that do not require high-end hardware.



**TAGS** · BIGDATA, · CLUSTER, · HADOOP, · RASPBERRY

*By AlanVerdugo*

## 8 Comments on "Building a Hadoop cluster with Raspberry Pi"

**CarloAppugliese** · August 15, 2016

Great post Alan.. I want to do a similar cluster with HDFS and Apache Spark. Maybe we can chat some time this week?

Log in to Reply

**shanoudi** · September 12, 2016

Greate post. I was able to do step 13, and this is the tail of the command ./hdfs namenode -format

/************************************************************

SHUTDOWN_MSG: Shutting down NameNode at controller/127.0.1.1

************************************************************/

However, when I run ./start-dfs.sh I get the following worning:

WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform… using builtin-java classes where applicable

and when I run /opt/hadoop/hadoop/bin/hdfs dfsadmin -report I get the following:

16/09/12 18:59:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform… using builtin-java classes where applicable

report: Call From controller/127.0.1.1 to master:9000 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused

I have one maser/name node and 3 secondery nodes. I can shh to each one without entering passwords.

I hope that any one can help me.
Best,

Log in to Reply

**AlanVerdugo** · September 12, 2016

Hello shanoudi, “Unable to load native-hadoop library for your platform” is a normal message. I receive it too.
Regarding the “connection refused” error, check if you can SSH into each server from each of the other servers. For example, if you have a Master node and your slaves are A, B and C, make sure that A can connect to B, then A to C, then B to C, etc…
Let me know if you need further help.

Log in to Reply

**shanoudi** · September 13, 2016

I want to thank you for your reply.
I am able to SSS from all pies to all pies. In in the /etc/hosts file, my master node is named master, but my slaves are named p1,p2, and p3 (since I have 3 pi zeros). Do you think that is an issue?
Best,
Samer

Log in to Reply

**AlanVerdugo** · September 13, 2016

Well, any names should be ok, as long as they are consistent trough all the /etc/hosts files and trough the configuration files.

Are the Raspberries running Raspbian? I remember I had a similar issue with the IP Address 127.0.1.1. In http://wiki.apache.org/hadoop/ConnectionRefused they talk about this issue and how to solve it (I think they just comment the 127.0.1.1 line and restart the network daemons). Maybe that is the issue you are having?

Log in to Reply

**YaqingChyi** · October 23, 2016

Hi Alan, When I do Step 13, run ./start-yarn.sh script.I get Error like this:
”
# pi @ master in /opt/hadoop/hadoop/sbin [12:10:46]
$ ./start-yarn.sh
starting yarn daemons
resourcemanager running as process 2408. Stop it first.
localhost: Error: JAVA_HOME is not set and could not be found.
“,However,my profiles file “~/.zshrc ” have” export JAVA_HOME=/usr/lib/jvm/jdk-8-oracle-arm32-vfp-

hflt/jre”
I use two raspberry Pi (one Raspberry pi 3B(master),another Raspberry pi 2B(salve-01)).
I hope that any one can help me.
best,

Log in to Reply

**AlanVerdugo** · October 28, 2016

Hi! I am not sure about that problem, but you could try the following options:
a) Use “env | grep JAVA” to see if the JAVA_HOME variable is assigned correctly.
b) Run “ls -la /usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/jre” to see if the file exists and has the correct permissions.
c) Try to do the export in the command line just before ./start-yarn.sh

Let me know if that helps!

Log in to Reply

**ioan-s** · November 25, 2016

Hello and thanks for this tutorial :).
In step 9 : The folder for master and slaves files should be “$HADOOP_HOME/hadoop/etc/hadoop”, not “$HADOOP_HOME/hadoop/etc/”.
All the best!

Log in to Reply

## Join The Discussion

You must be logged in to post a comment.