Andrew (he/him)
Posted on Jul 8, 2019 • Updated on Jul 22, 2019

Building a Raspberry Pi Hadoop / Spark Cluster

#raspberrypi #showdev #hadoop #java

Hadoop & Spark (4 Part Series)

1. Installing and Running Hadoop and Spark on Windows
2. Big Data Analysis with Hadoop, Spark, and R Shiny
3. **Building a Raspberry Pi Hadoop / Spark Cluster**
4. Installing and Running Hadoop and Spark on Ubuntu 18

> *Pro tip: if you're only looking for how to configure Hadoop and Spark to run on a cluster, start here.*

Table of Contents

- About Raspberry Pi
- Choosing a Pi Model
- Power over Ethernet (PoE) and Networking
- Disk Space
- Cost and Overall Specs
- Installing the PoE HATs

3. Software
   - Operating System
   - Networking
   - Securing the Cluster

4. Hadoop & Spark
   - Single-Node Setup
   - Cluster Setup

5. Conclusion

Motivation and Background

"Big Data" has been an industry buzzword for nearly a decade now, though agreeing on what that term means and what the field of Big Data Analytics encompasses have been points of contention. Usage of Big Data tools like The Apache Software Foundation's Hadoop and Spark (H&S) software has been met with scorn and praise alike. The truth is that, like any other tools, H&S are only helpful if you know when and how to use them appropriately.

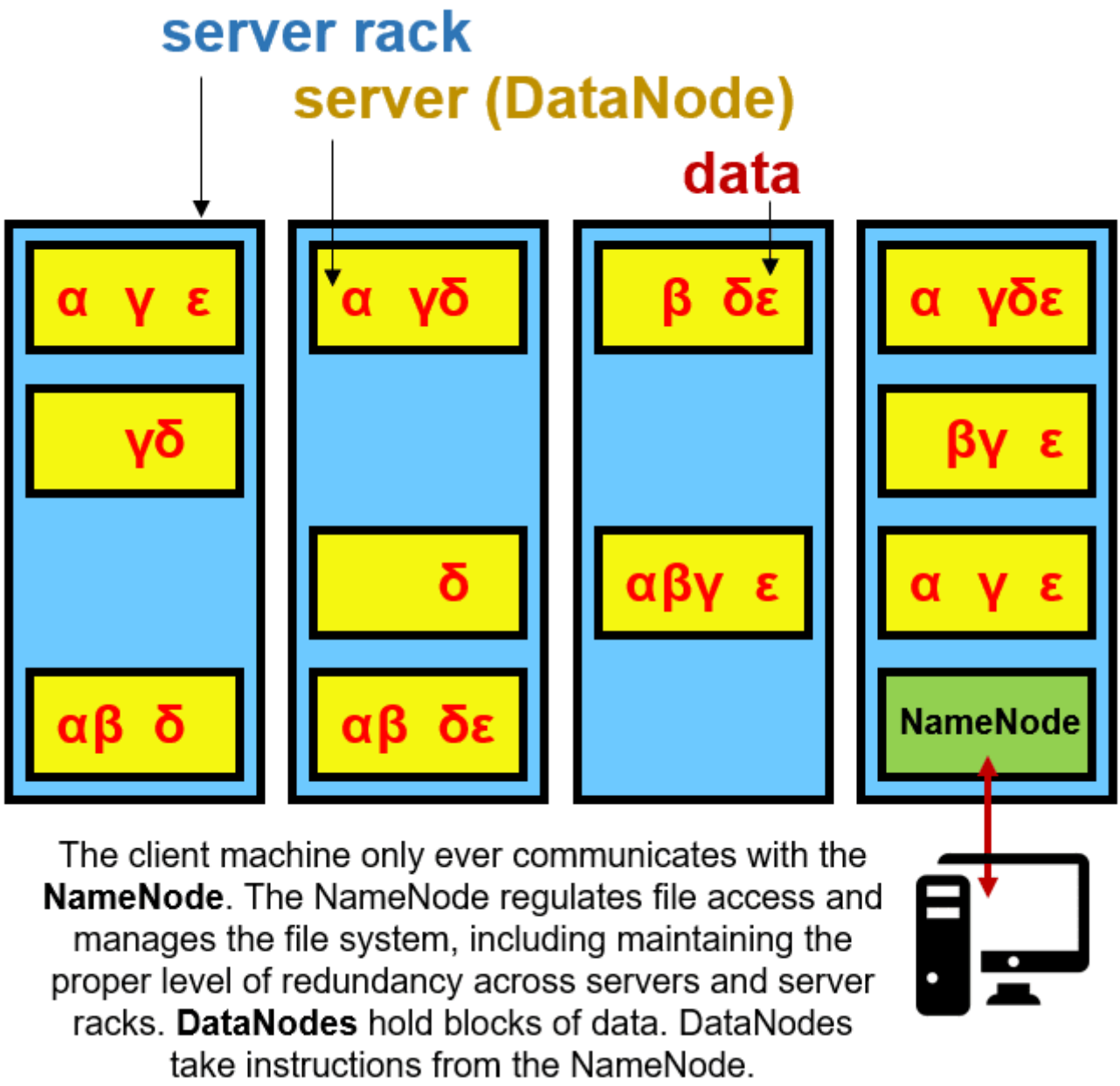
Many companies use H&S for data storage and analysis. Developers with these skills are in demand, but until recently, it was difficult to get the necessary experience with cluster computing without a big investment of time and/or money. The wide availability of small, cheap single-board computers like the brand-new Raspberry Pi 4 ($35 starting price) has all but eliminated these barriers to developing your own cluster.

In this guide, **I will teach you how to build a networked cluster of Raspberry Pis**. We'll get them communicating with each other through a network switch, install HDFS, and have Spark running distributed processing jobs via YARN across the whole cluster. This guide is a full-featured introduction to the hardware and software involved in setting up an H&S cluster, and can scale (just like H&S!) to any number or size of machines.

What is Apache Hadoop?

The Apache Hadoop Distributed Filesystem (HDFS) is a distributed (networked) filesystem which can be run on off-the-shelf, consumer-grade hardware. Running on networked commodity hardware means that HDFS is both scalable and cost-effective. HDFS splits files into "blocks" (typically 64MB in size) and stores multiple copies of these blocks across different "DataNodes"

on the same rack, different racks in the same data storage centre, or across multiple data storage centres.



HDFS Hardware Architecture Diagram

Data split and stored in this way has several advantages:

1. **Fault tolerance.** Redundant copies are made of each block and stored across multiple physical locations. This means that if a disk fails (as is to be expected with large volumes of commodity hardware), all of the blocks on that node can be sourced from other locations. HDFS can also make additional copies of the lost blocks so that the desired number of redundant blocks is always maintained.

> In older versions of Hadoop, the single NameNode was a potential vulnerability for the entire system. Since the NameNode holds all of the information about the filesystem and changes made to it, a failed NameNode compromises the whole cluster. In newer releases, a cluster can have multiple NameNodes, eliminating this single point of failure.

2. **Parallelisation.** Data split into blocks is ripe for parallel processing with a map-reduce analysis pipeline. A file split into 100 blocks across 100 similarly-powered machines can be processed in roughly 1/100th of the time it would take to process the same file on a single machine.

3. **Data Locality.** In-situ processing of blocks also eliminates the need to transfer large data files across the network. This can drastically reduce network bandwidth requirements. Time and computing power that would have otherwise been spent copying files from a repository can instead be used to process the data locally: "*moving computation is cheaper than moving data*".

4. **Large datasets.** Because HDFS breaks files into blocks, the size of any individual file is limited only by the total amount of storage available across the network. HDFS is POSIX-based but relaxes some POSIX requirements to allow fast streaming of data, among other benefits. HDFS can support hundreds of networked nodes and tens of millions of files.

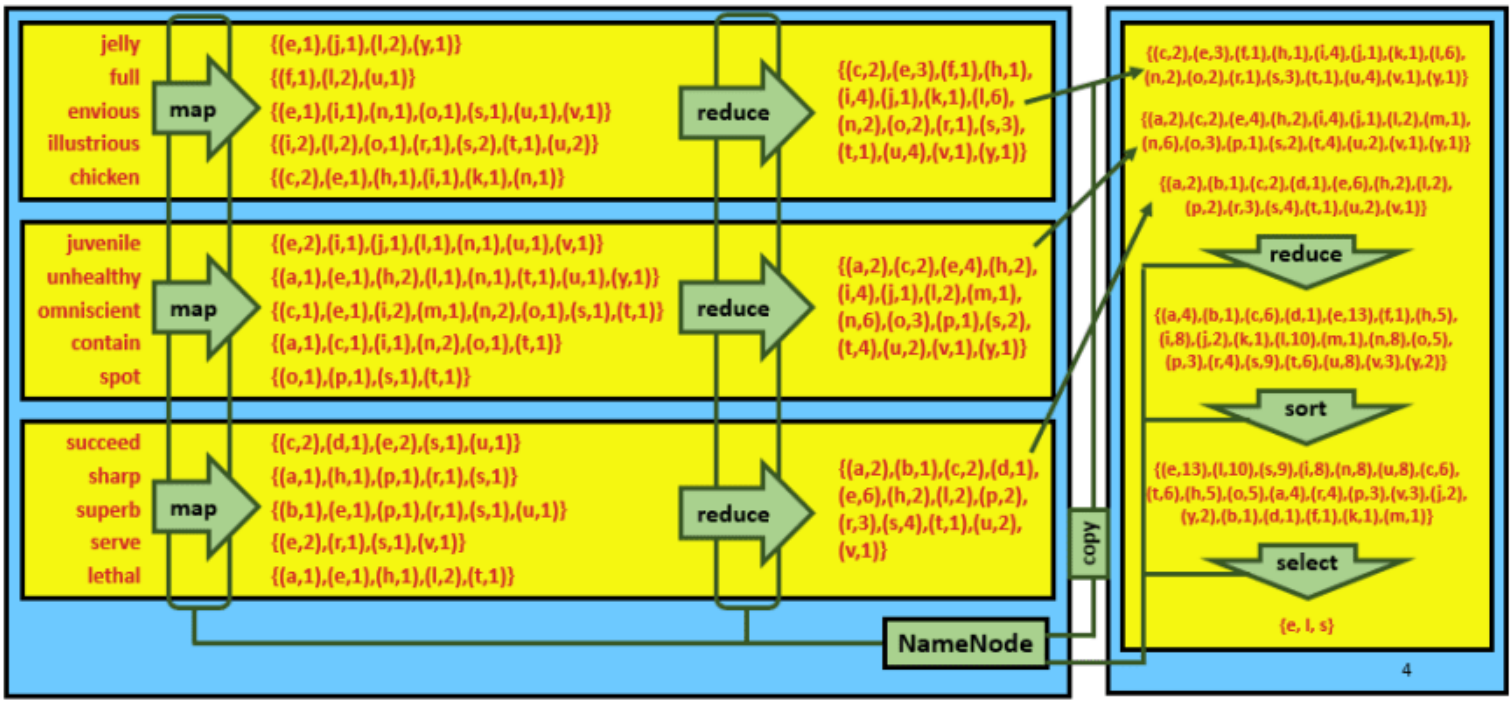
HDFS speeds up data processing by distributing the I/O (read/write) disk latency across all disks on the network. Instead of reading a file from a single location on a single disk, it's read simultaneously from multiple points by multiple machines. Those same machines can then operate on that same file in parallel, increasing processing speeds by orders of magnitude.

One point to note for HDFS is that, to facilitate data coherency, files written to HDFS are immutable. This means that, for a file to be edited, it must be downloaded from HDFS to a local file system, changed, and uploaded back to HDFS. This workflow is a bit different from what most users are accustomed to, but it is necessary for the high-throughput data access provided by HDFS. HDFS is not meant to replace your normal, interactive filesystem. It should be used as a repository for your Big Data, which won't change regularly, but which needs to be processed quickly and easily. It's a "write one time, read many times" file system.

> You can read much more about the nitty-gritty architectural details of HDFS here, if you're interested.

Map-Reduce and Parallelisation

The distributed nature of the data stored on HDFS makes it ideal for processing with a **map-reduce** analysis framework. Map-reduce (also "MapReduce", "Map-Reduce", etc.) is a programming technique where, as much as possible, parallelisable tasks are performed concurrently, followed by any non-parallelisable "bottlenecks". Map-reduce is a general framework for analysis and is not a particular algorithm. (In Big Data circles, however, it is sometimes synonymous with Apache's **Hadoop MapReduce**, which is discussed in detail below.)



An Example Map-Reduce Pipeline

Some data analysis tasks are *parallelisable*. For instance, if we wanted to find the most common letters among all of the words in a particular database, we might first want to count the number of letters in each word. As the frequency of letters in one word don't affect the freqency of letters in another, the two words can be counted separately. If you have 300 words of roughly equal length and 3 computers to count them, you can divvy up the database, giving 100 words to each machine. This approach is very roughly 3x as fast as having a single computer count all 300 words. Note that tasks can also be parallelised across CPU cores.

> Note: there is some overhead associated with splitting data up into chunks for parallel analysis. So if those chunks cannot be *processed* in parallel (if only one CPU core on one machine is available), a parallelised version of an algorithm will usually run *more slowly* than its non-parallelised counterpart.

Once each machine in the above example has analysed all of its 100 words, we need to synthesise the results. This is a *non-parallelisable* task. A single computer needs to add up all of the results from all of the other machines so that the results can be analysed. Non-parallelisable tasks are *bottlenecks*, because no further analysis can even be started until they are complete.

Common parallelisable tasks include:

- filtering (ex: remove invalid or incomplete data)
- transformation (ex: format string data, interpret strings as numeric)
- streaming calculations (ex: sum, average, standard deviation, etc.)
- binning (ex: frequency, histogramming)

Common non-parallelisable tasks include:

- aggregation (ex: collecting *partial* results into a single *global* result)
- text parsing (ex: regular expressions, syntax analysis)
- visualisation (ex: creating summary plots)
- mathematical modelling (ex: linear regressions, machine learning)

Sorting data is an example of an algorithm which doesn't fit nicely into either of the above categories. Although the entire dataset necessarily needs to be collected into one location for complete *global* sorting, sorting small collections of data which themselves are already *locally* sorted is much faster and easier than sorting the equivalent amount of unsorted data. Sorting data in this way is essentially both a *map* and a *reduce* task.

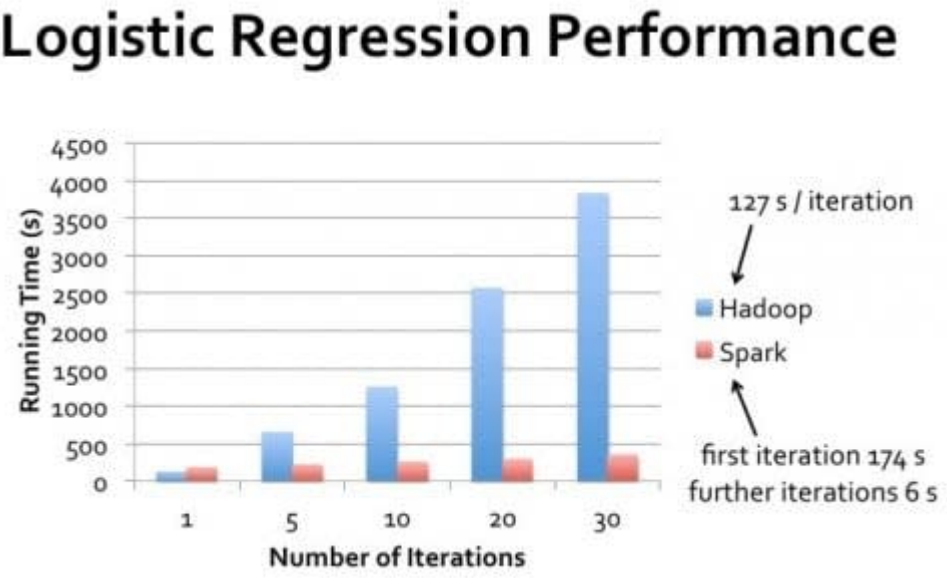
> Parallelisation is not appropriate for all tasks. Some algorithms are inherently sequential (aka. P-complete). These include *n-body problems*, the *circuit value problem*, *Newton's Method* for numerically approximating the roots of a polynomial function, and *hash-chaining*, which is widely used in cryptography.

What is Apache Spark?

When HDFS was first released in 2006, it was coupled with a map-reduce analysis framework called -- creatively enough -- Hadoop MapReduce (usually just "MapReduce"). Both HDFS and MapReduce were inspired by research at Google, and are Apache counterparts to Google's "Google File System" and "MapReduce", the latter of which Google was granted a patent for (which has been criticised).

Hadoop MapReduce is the original analysis framework for working with data stored on HDFS. MapReduce executes map-reduce analysis pipelines (described above), reading data from HDFS before the "map" tasks, and writing the result back to HDFS after the "reduce" task. This behaviour in particular is one of the reason's why Apache Spark, widely seen as a successor to MapReduce, offers a speedup of 10-100x, relative to MapReduce.

> *Hadoop MapReduce works with the HDFS to process "process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner."* [source]



Hadoop vs. Spark performance on a logistic regression

Spark offers at least four primary advantages over MapReduce:

1. **Spark minimises unnecessary disk I/O.** Spark offers several improvements over MapReduce in an effort to read from and write to disk as little as possible.* While MapReduce writes every intermediate result to disk, Spark tries to pipeline results as much as possible, only writing to disk when the user demands it, or at the end of an analysis pipeline. Spark will also cache data which is used for multiple operations in memory, so it doesn't need to be read from the disk multiple times. For these reasons, Spark is sometimes said to have "in-memory processing". (This is a bit of a misleading term, though, as both MapReduce and Spark necessarily process data in RAM.)

   > * In general, reading from and writing to a CPU's cache is an order of magnitude faster than RAM, and RAM is a few orders of magnitude faster than SSD (which is faster still than a conventional hard disk).

2. **Spark provides abstractions for fault-tolerant processing.** The primary data structure provided by Spark is the Resilient Distributed Dataset (**RDD**):

   - **Resilient** -- Spark keeps a *lineage* of how a given RDD is constructed from any "parent" RDDs. If any RDD is corrupted or lost, it can easily be recreated from its lineage graph (aka. its *logical execution plan*).
   - **Distributed** -- An RDD may physically exist in different pieces over several machines. Spark cleanly abstracts away the distributed nature of the files stored on HDFS. The same code which reads and processes a single file stored on a single machine can be used to process a distributed file, broken into chunks and stored over many different physical locations.
   - **Dataset** -- RDDs can store simple objects like `String`s and `Float`s, or more complex objects like tuples, records, custom `Object`s, and so on. These datasets are inherently parallelisable.

> RDDs are immutable collections of data, so they are **thread safe**: they can be processed in parallel without the programmer having to worry about race conditions or other multithreading pitfalls. (The logic for making files stored on HDFS immutable is similar.)
>
> RDDs are **lazily evaluated**. The sequence of calculations which must be traced to construct an RDD are not performed until the RDD needs to be *used* -- printed to the terminal, plotted, written to a file, etc. This reduces the amount of unnecessary processing that's performed by Spark.

3. **Spark has a different processing model.** Spark uses a Directed Acyclic Graph (**DAG**) processing model, rather than a simple, two-step map-reduce pipeline:

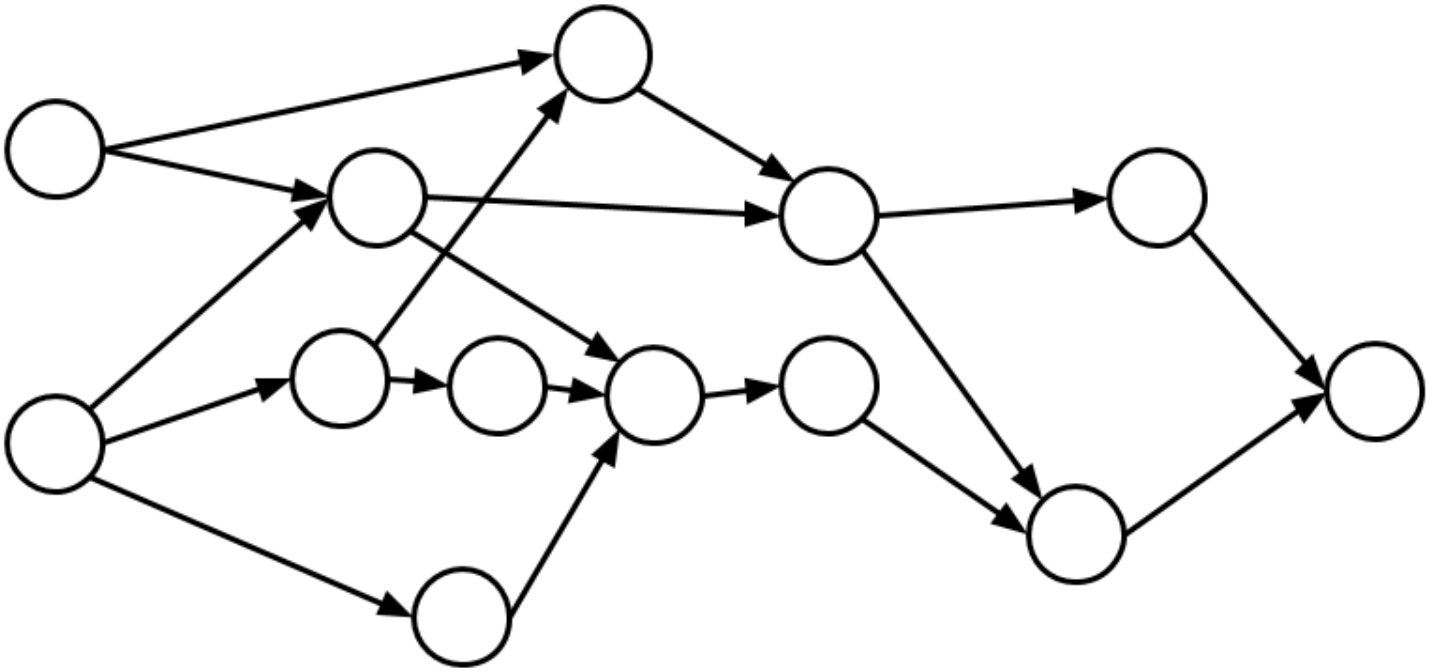


Diagram of a Directed Acyclic Graph (DAG)

What this means is that Spark takes a holistic view of the entire processing pipeline and attempts to optimise the process globally. While MapReduce will (1) read data from disk, (2) perform a "map" operation, (3) perform a "reduce" operation, (4) write data to disk, Spark is more flexible about what is completed when. As long as forward progress is made toward the final result, *maps* and *reduces* may be performed in parallel or at different times on different chunks of data. The DAG is a more general version of MapReduce's map-reduce pipeline -- it can also be viewed as the implementation in code of the idealised *lineage graph* of an RDD.

4. **Spark eliminates JVM boot times.** By booting up a Java Virtual Machine (JVM) on each DataNode at startup (as opposed to when a new job is executed), Spark eliminates the time required to load `*.jar`s, parse any configuration files, and so on. MapReduce opens a new JVM each time a new task is run, and all of this startup overhead is incurred on every job. It may only take a few seconds each time, but it adds up.

Additionally, Spark has built-in support for complex analytics like machine learning (MLlib) and graph processing (GraphX) and can be used interactively in the Python, R, Scala, Java, and SQL shells

The above features make Spark much faster, more fault-tolerant, and more feature-rich than MapReduce. With all of the benefits Spark has over MapReduce, essentially the only time you should prefer to use the latter is when you have legacy MapReduce code to maintain. Or if you don't care about processing time. Or if you don't want to spend time learning Spark.

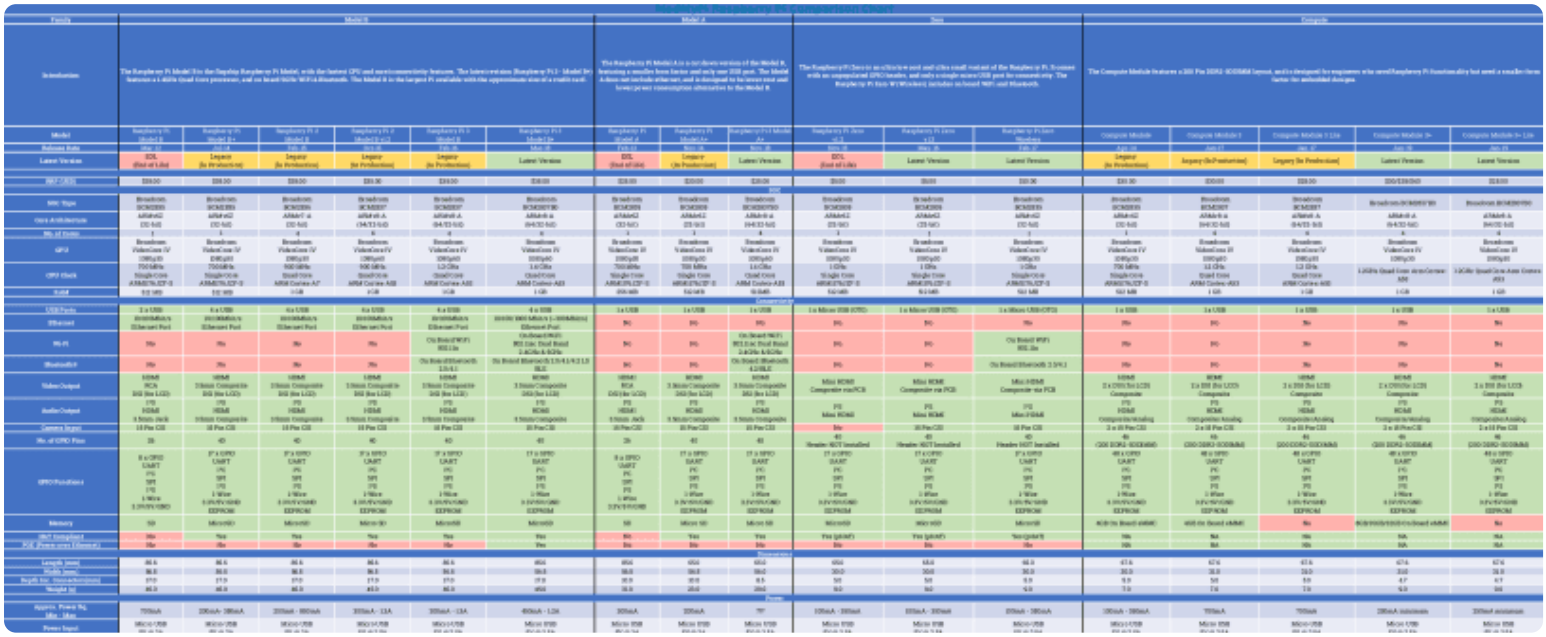
There are other distributed data storage / distributed analysis pipelines, but Apache Hadoop and Spark are free, fast, fault-tolerant, and relatively easy to install and understand. Equipped with this foundational understanding of H&S, we can now move on to constructing our own Raspberry Pi H&S cluster.

[back to top]

Hardware

About Raspberry Pi

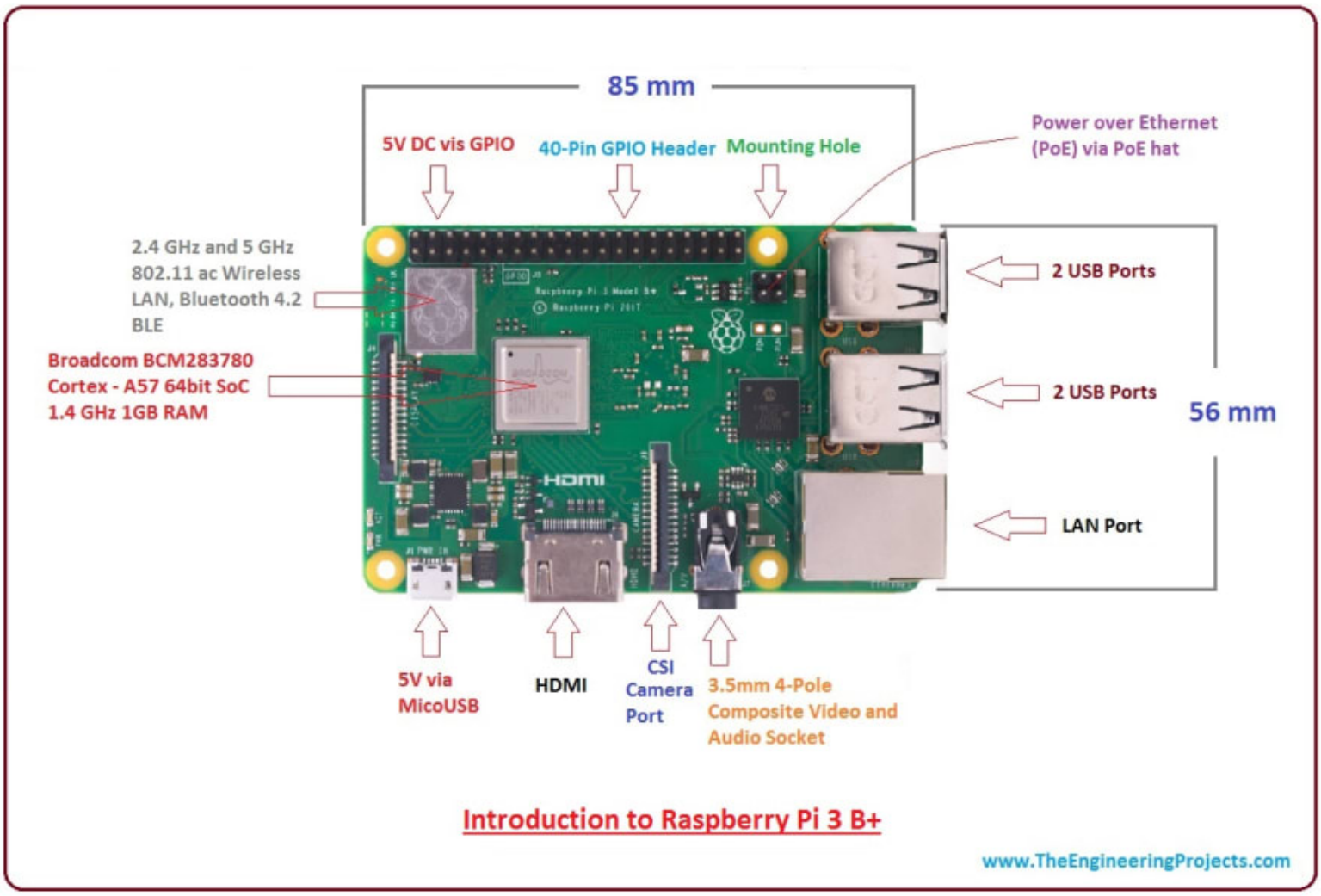
The Raspberry Pi Foundation have been developing and releasing single-board computers since early 2012. Originally conceived of as a way to inspire kids to get into electronics and programming, like the DIY "kit" computers of the 1970s and 80s, the popularity of the Raspberry Pi has far exceeded the expectations of the Foundation's creators, selling 12.5 million units to become the third most popular personal computer ever, beating the Commodore 64, but still trailing the Apple Macintosh and the Windows PC. Yes, Raspberry Pis have been used to introduce a new generation to computing, but they're also used for home automation, manufacturing, robotics, digital cameras, and more.



Raspberry Pi Comparison Chart (click to open PDF)

There are many different models of Raspberry Pi available today, and they fall into four families: Model A, Model B, Zero, and Compute. The **Model B** family are the flagship, full-featured Raspberry Pis. Model Bs all have Ethernet connectivity, which no other family of Pi has. Every Model B has an MSRP (Manufacturer's Suggested Retail Price) of $35 US. Model Bs all have the exact same form factor: 85.6mm x 56.5mm x 17.0mm, weighing in at only 17g.

> Model B Pis are *literally* credit-card-sized! And about as "thick" as a keyboard key is wide. They're roughly the size and weight of a deck of cards.



Raspberry Pi 3 B+ Diagram

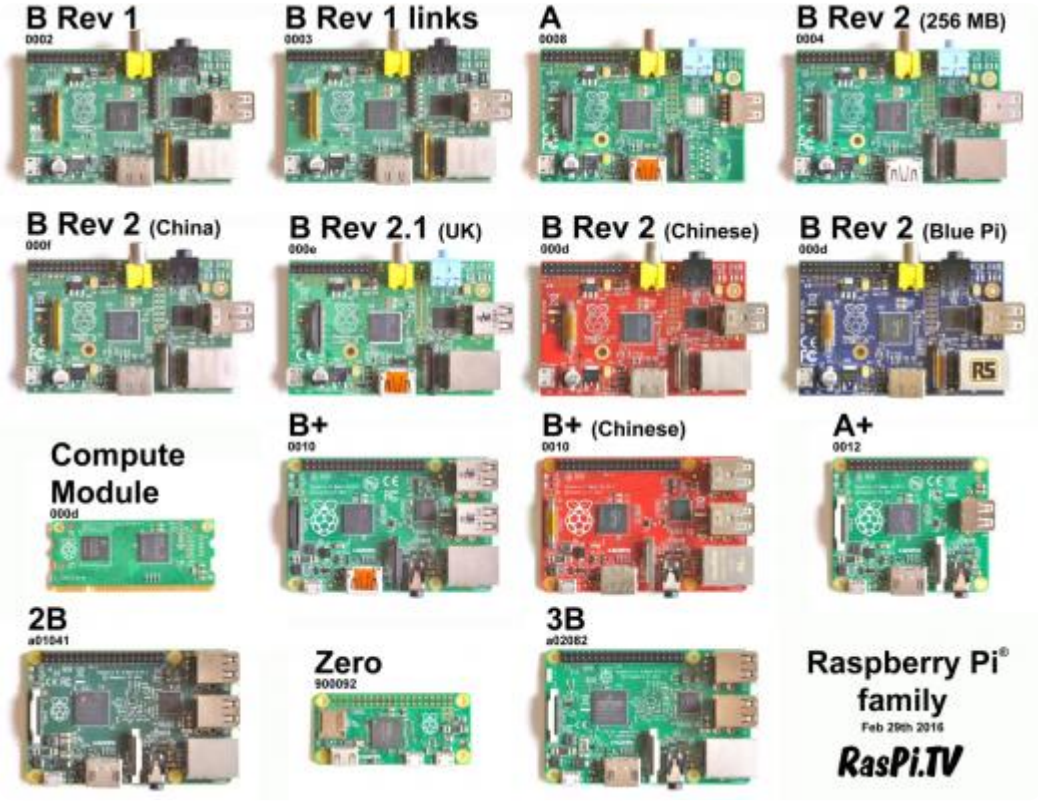
There are different *versions* of Raspberry Pi, as well. Version 1 included a Model B as well as a Model B+, which saw the inclusion of two additional USB ports, a switch from SD to microSD storage, and a change in the GPIO (General-Purpose Input/Output) header on the board which allowed for the addition of **Pi HATs** (Hardware Attached on Top). HATs connect to the GPIO pins and add additional features to the Pi: allowing PoE (Power over Ethernet), cooling fans, LCD screens, and more.

Version 1 also included a Model A and A+. The A series are physically smaller, cheaper ($20-$25), and have reduced specs (only 1 USB port, no Ethernet, less RAM). Model Zero Pis are even more stripped-down and even cheaper ($5-$10). Details on all of these models of Pi can be seen in the table above, or found in various comparison tables online.

Choosing a Pi Model

We want to network our Pis to build a Hadoop cluster, so we are restricted to models which have either Wi-Fi and/or Ethernet hardware. Of the Pis which are currently available, this limits us to the Pi 3 Model B, Pi 3 Model B+, Pi 3 Model A+, or the Pi Zero Wireless. All of these Pis have WiFi connectivity, but only the Model B and B+ have Ethernet.

> Note that the Version 4 Pis were just released a few weeks ago, but they're selling out as quickly as stores can stock them. If you can, I recommend using the Version 4 Pis over the Version 3 ones. They have faster processors and the ability to add more RAM (up to 4GB).



Raspberry Pi Models

If we network the Pis using WiFi, we'll need a wireless router. We'll also need to buy power supplies for each Pi. I'm going to build an eight-Pi cluster. Eight power supplies will be bulky. I'll have to carry around all of these, plus a multi-outlet, if I want to show off my cluster to anyone. To eliminate this bulk, we can instead power them through PoE. This takes care of both the networking and the power supply for each Pi, but it is more expensive. This choice restricts me to only the Model B or the Model B+.

Finally, to eliminate any possible network latency with the Pis (and since all Models B/B+ are the same price) I'm going to chose the Pi 3 Model B+, which has the largest bandwidth Ethernet support (~300Mbit/s). For my 8-Pi cluster, the total cost of the Pis is about **$280**.

Power over Ethernet (PoE) and Networking

Raspberry Pis Model B+ can be powered via a traditional DC power supply (micro-USB). These cost about $8 each (so $64 total for my 8-Pi cluster) and would require each Pi to have access to its own electrical outlet. You can buy a multi-outlet ("power strip") online for about $9.

Alternatively, Pis B+ can also be powered via Ethernet, through a technology known, creatively, as Power over Ethernet, or PoE. PoE requires a network switch which supports PoE; these are generally more expensive than those which don't support PoE. Non-PoE, 8-port network switches can be bought for as little as $10 on Amazon, and gigabit (1000Mbps) switches are available for as little as $15. An 8-port network switch with PoE support on all ports runs about $75. Networking via Ethernet requires 8 short Ethernet cables (about $14 for a 10-pack). If you opt for the non-PoE route, you could also network your Pis with a good, cheap wireless router (about $23 on Amazon).

At minimum, then, we're looking at $64 + $9 + $10 + $14 = **$97** in networking and power supply costs for a wired (Ethernet) cluster with DC power supplies, or $64 + $9 + $23 = **$96** in networking and supply costs for a WiFi cluster with DC power supplies.

Raspberry Pi cluster using Power over Ethernet

The cluster I'm building will be used for demonstrations, so it needs to be portable. Eight power supplies plus a multi-outlet is not ideal, so I opted for the PoE option. This eliminates all of that extra bulk, but it comes at a cost. A PoE Pi cluster requires 8 PoE HATs (about $21 each), plus the Ethernet cables (10 for $14), plus a PoE-enabled, 8-port network switch (about $75). This means that a PoE Pi cluster will incur at least **$257** in networking and power supply costs, or over 2.5x as much as a wireless cluster.

C4Labs Cloudlet Cluster Case, holding a Raspberry Pi cluster

I have one more small complication, too. I bought a special carrying case from C4Labs for my Pi cluster ($55), so the only network switch I could order from Amazon UK (due to space restrictions) is the TRENDnet V1.1R, at a cost of about $92. So my total networking and power supply cost (plus this case) is **$329**.

> Note that none of the above quoted prices include shipping. Also, the C4Labs case comes with cooling fans which are meant to draw their power from the GPIO 5V pins on the Pis. As these will be occupied by the PoE HAT, I bought some USB jacks and a soldering iron so I can solder the cooling fan wires to the USB jacks. The fans can then draw their power from the Pis via USB. These two purchases added about another **$32** to the cluster cost.

Disk Space

Raspberry Pis don't come with on-board storage space. They do come with microSD slots, though, so microSD cards can be used to hold the OS and data for the Pi. High-rated micro SD cards tend to cost around $17.50 for 128GB models, though most of these limit read speed to about 80MB/s. As of this writing, the cheapest 128GB SD cards with read speeds of up to 100MB/s cost around $21 each or **$168** for the entire cluster.

Cost and Overall Specs

In summary, the total cost of my 8-Pi cluster is roughly

- $280 for all eight Raspberry Pis Model B+
- $274 for all of the power supply and networking paraphernalia

- $168 for all of the microSD cards
- $55 for a nice carrying case
- $32 for USB jacks and a soldering iron

...or about $810 total. Shipping all of this stuff to Ireland cost about another ~$125, bringing the total cost of my cluster to about $935. Note that this doesn't include the cost of a keyboard, mouse, monitor, or HDMI cable, all of which I borrowed from other projects.

Selected specs of this 8-machine cluster are as follows:

- CPU: 8 x 1.4GHz 64-bit quad-core ARM Cortex-A53 Processors
- RAM: 8 x 1GB LPDDR2 SDRAM (max 1066 Mbps data rate)
- Ethernet: Gigabit Ethernet (max 300 Mbps), PoE support
- Storage: 8 x 128GB microSD storage (max 80MB/s read speed)
- Ports: 8 x HDMI ports, 8 x 4 x USB 2.0 ports

Not too bad for under $1000!

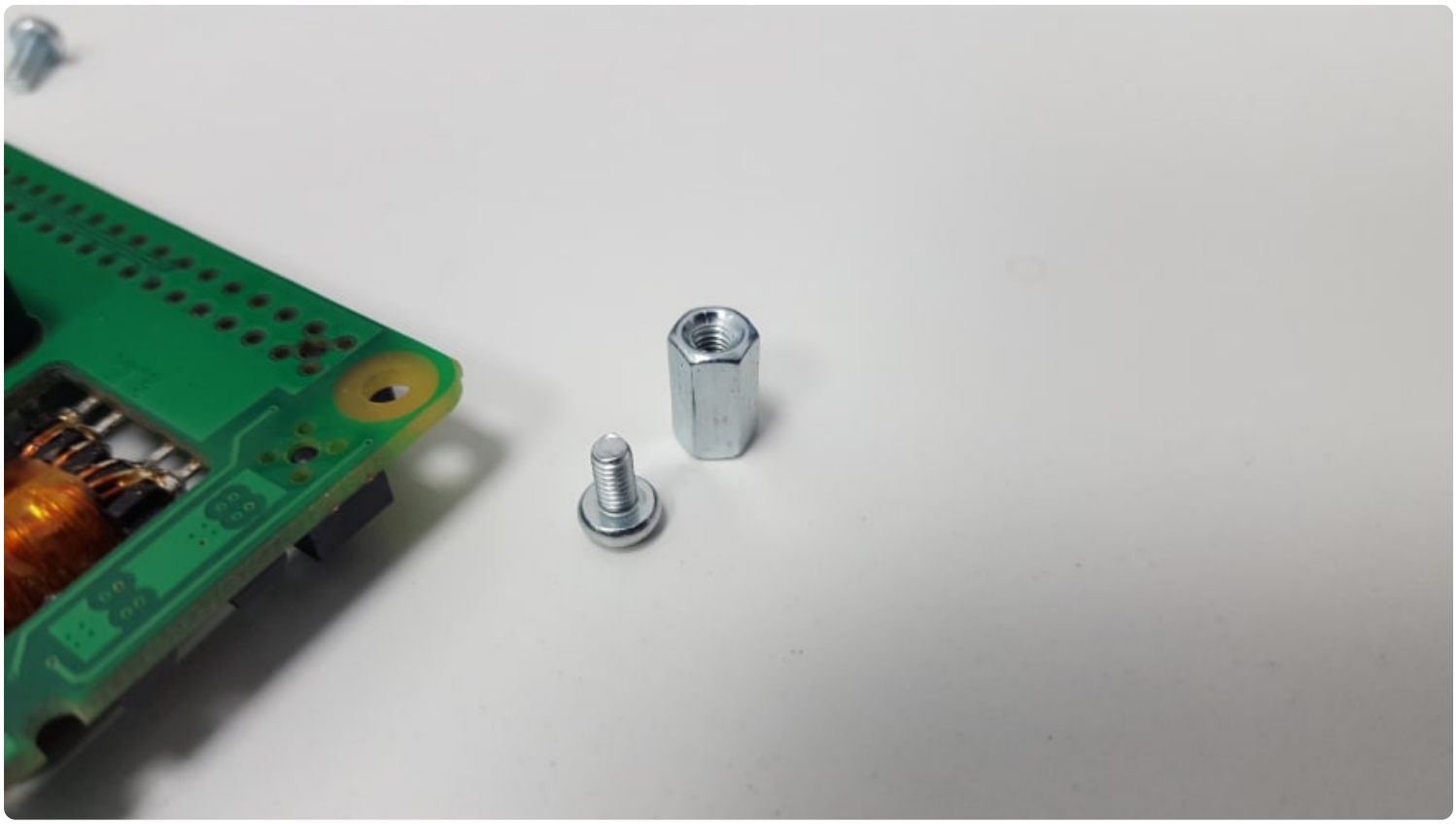
Installing the PoE HATs

For the most part, Raspberry Pis are plug-and-play. But since we're installing a HAT, we have to do a tiny bit of hardware work. The Raspberry Pi website gives instructions here for how to install the PoE HATs on top of a Raspberry Pi -- it's pretty easy:

1. Locate the 40-pin GPIO header and the nearby 4-pin PoE header on the top of the Raspberry Pi, and the corresponding 40-pin and 4-pin slots on the underside of the PoE HAT, but **don't put anything together yet**.

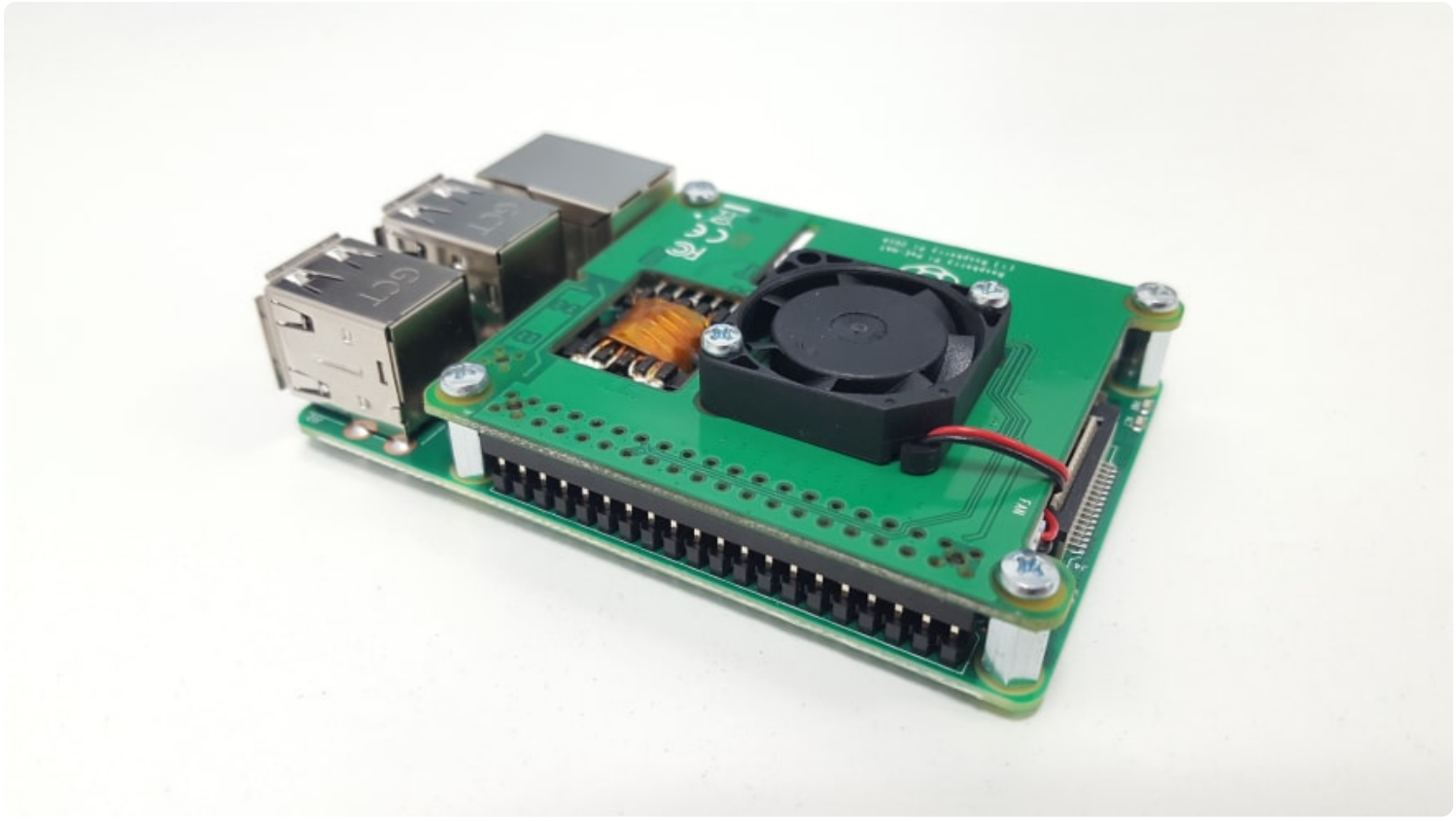
When the PoE HAT is flipped right-side-up, the pins on the Pi should align with the slots on the HAT:

2. Locate the spacers and screws that came packaged with the PoE HAT. There should be at least 4 spacers and at least 8 screws.

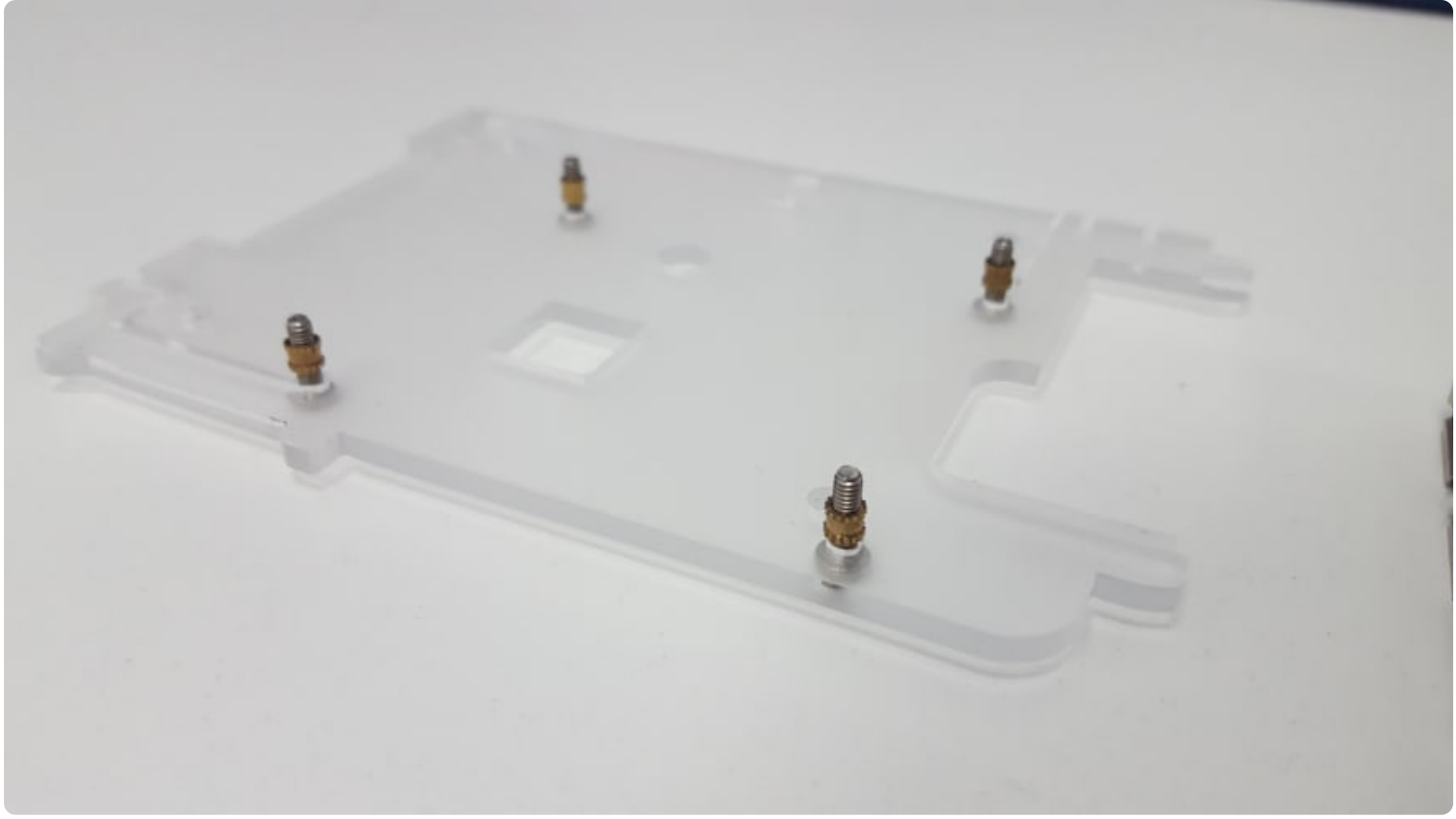


3. Screw the spacers to the HAT from the top.

4. At this point, if you are only assembling a single Pi, you'll attach the Pi to the HAT by aligning the spacers to the holes on the Pi and screwing into the spacers from the underside of the Pi, while making sure to carefully insert the Pi GPIO and PoE pins into the GPIO and PoE slots on the HAT.

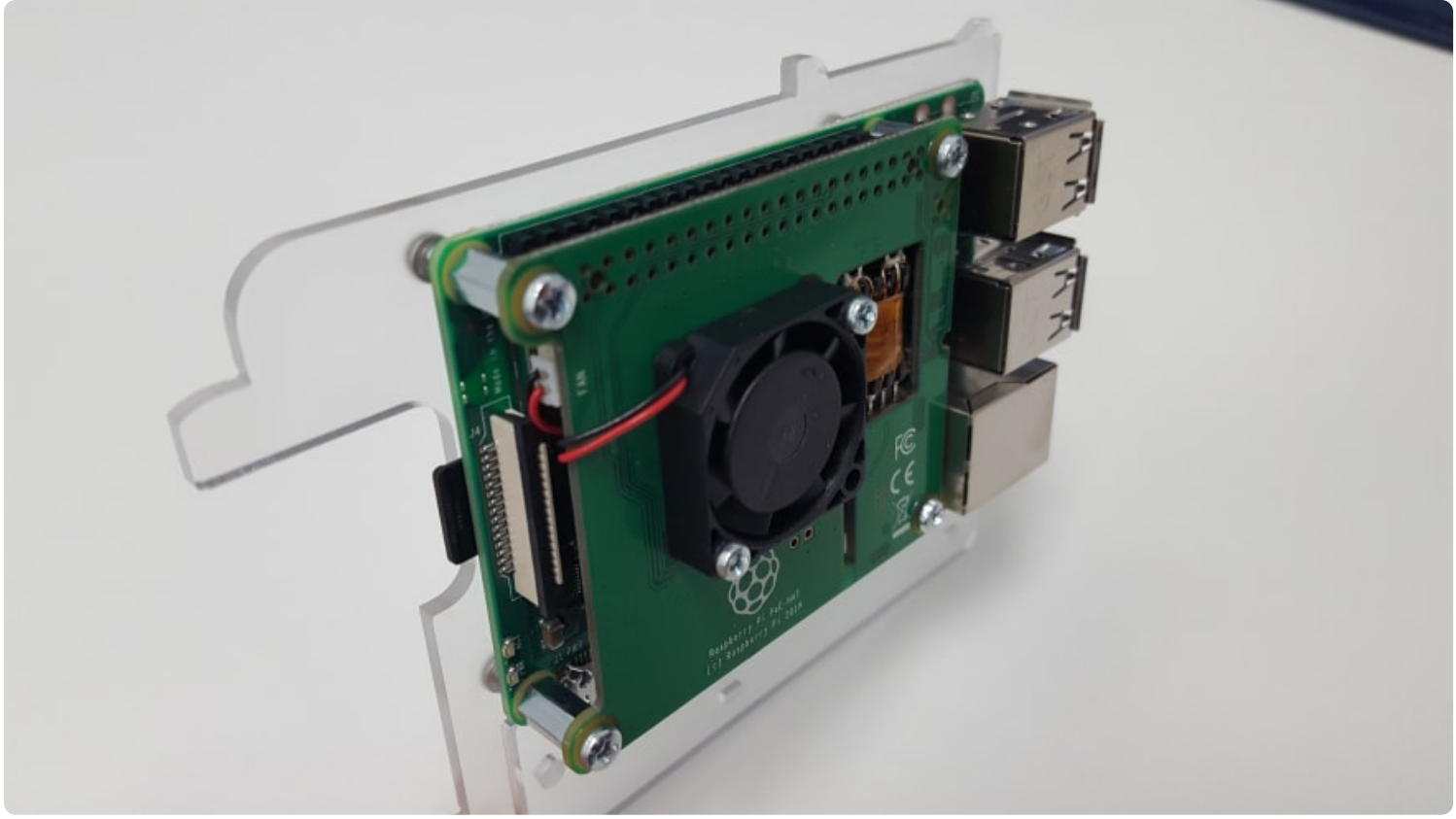


For my cluster, however, I'm assembling all of the Pis within a C4Labs Cloudlet / Cluster case, so the Pis need to be attached to the plexiglas mounts. I remove the protective paper from the plexiglas, and add the metal spacers and screws as shown in the following photograph:



Make sure to use the *longer* screws, because they need to make it through the plexiglas, past the metal spacers, through the Pi circuit board, and into the PoE HAT spacers:

Once everything is screwed together, the Pi should be securely attached to the plexiglas mount:



...that's it! PoE HAT successfully installed.

Be very careful if you attempt to take the PoE HAT off after you've pushed it down onto the Pi pins. It is very easy to wrench those pins off of the circuit board while trying to remove the PoE HAT. If that happens, you'll have to solder the pins back onto the Pi or switch to using a microUSB power supply instead (or buy a new Pi).

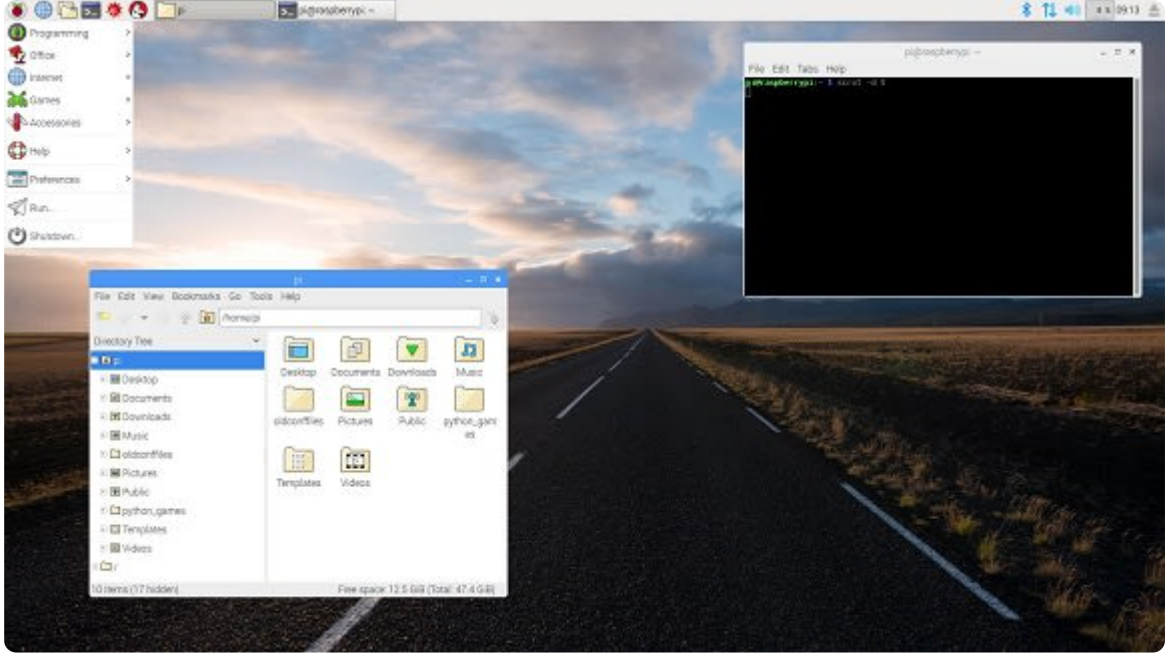
[back to top]

Software

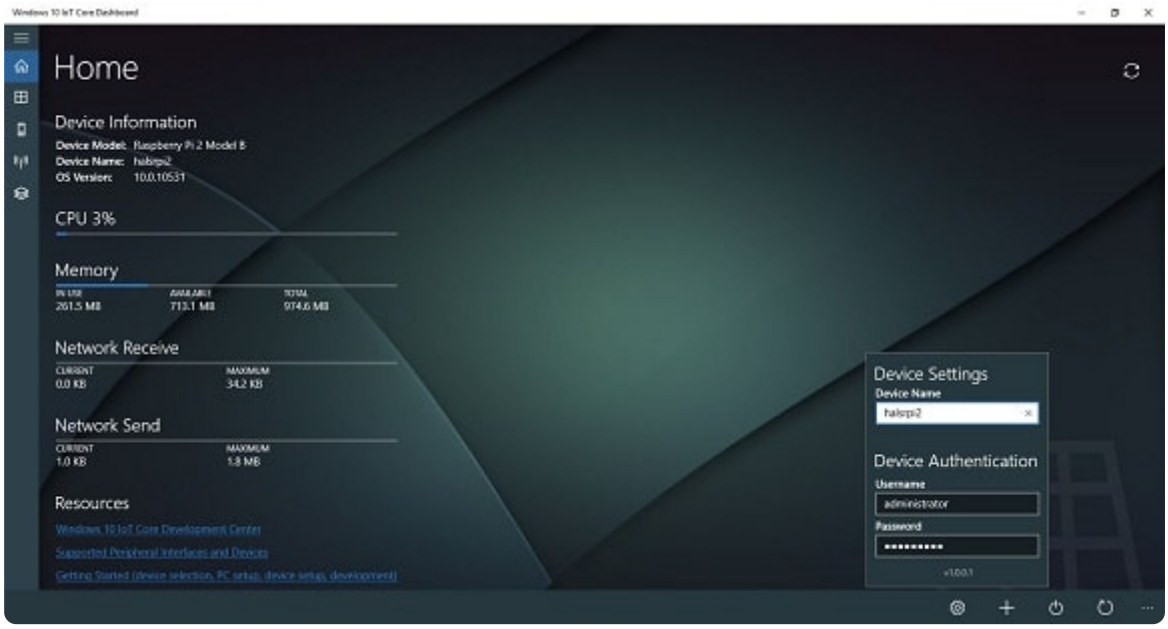
Operating System

There are dozens of operating systems available for the Raspberry Pi. Some of the more notable ones include:

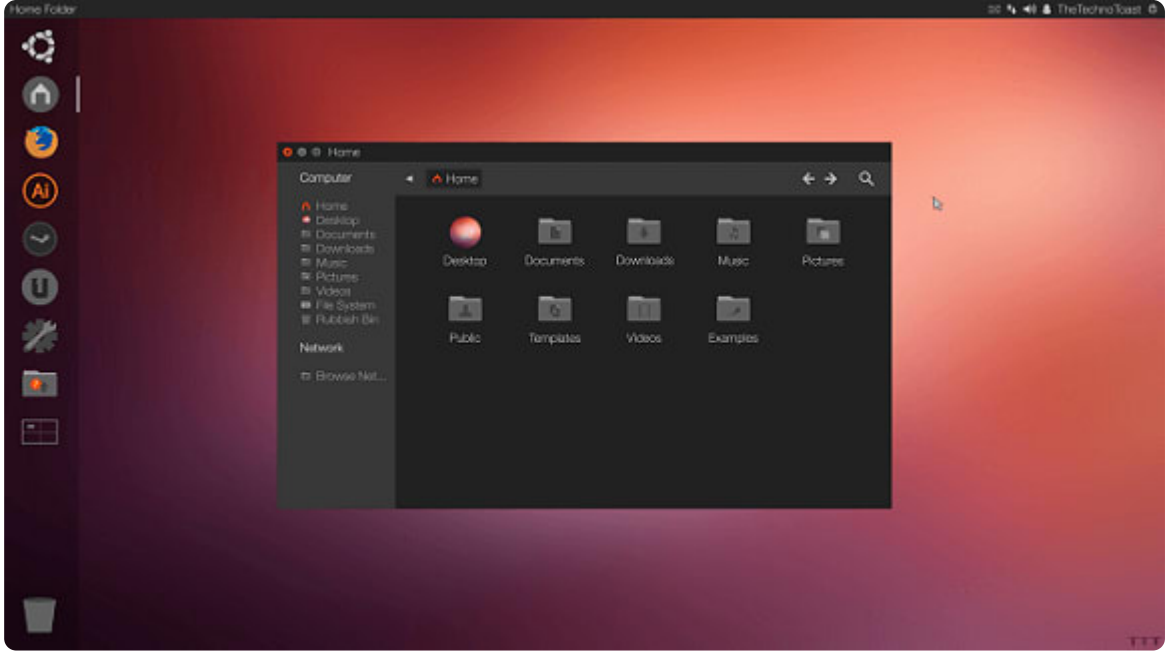
- **Raspbian** -- a Debian-based OS developed and maintained by the Raspberry Pi Foundation (the Foundation's recommended OS for the system)



- **Windows IoT Core** -- Windows' Internet of Things (IoT) operating system, for small devices; looks and feels similar to Windows 10



- **Ubuntu Core** -- A stripped-down version of one of the most popular Linux distributions, Ubuntu



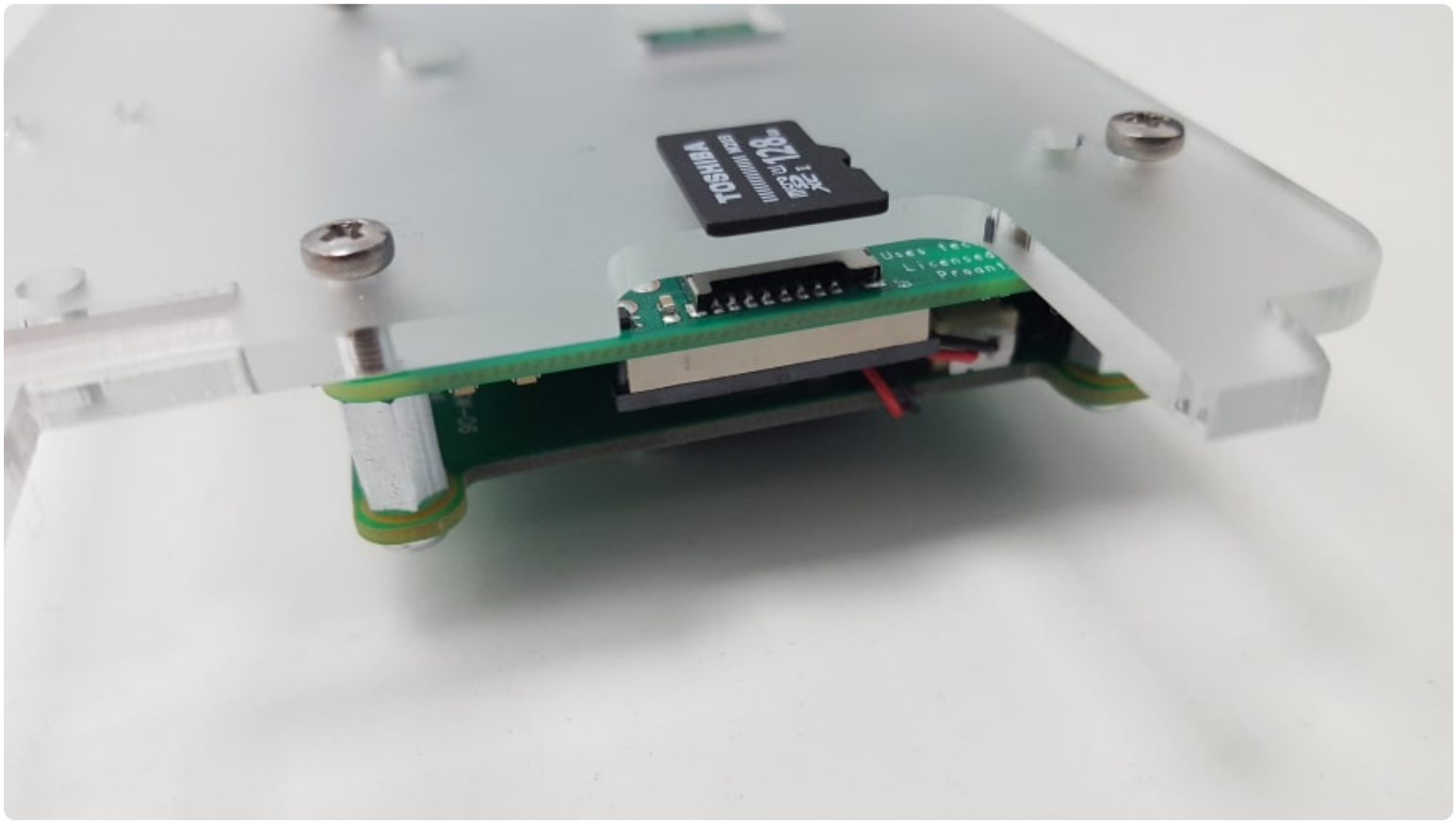
- **Android** -- The most popular operating system in the world; installed on over 2 billion devices, mostly mobile phones

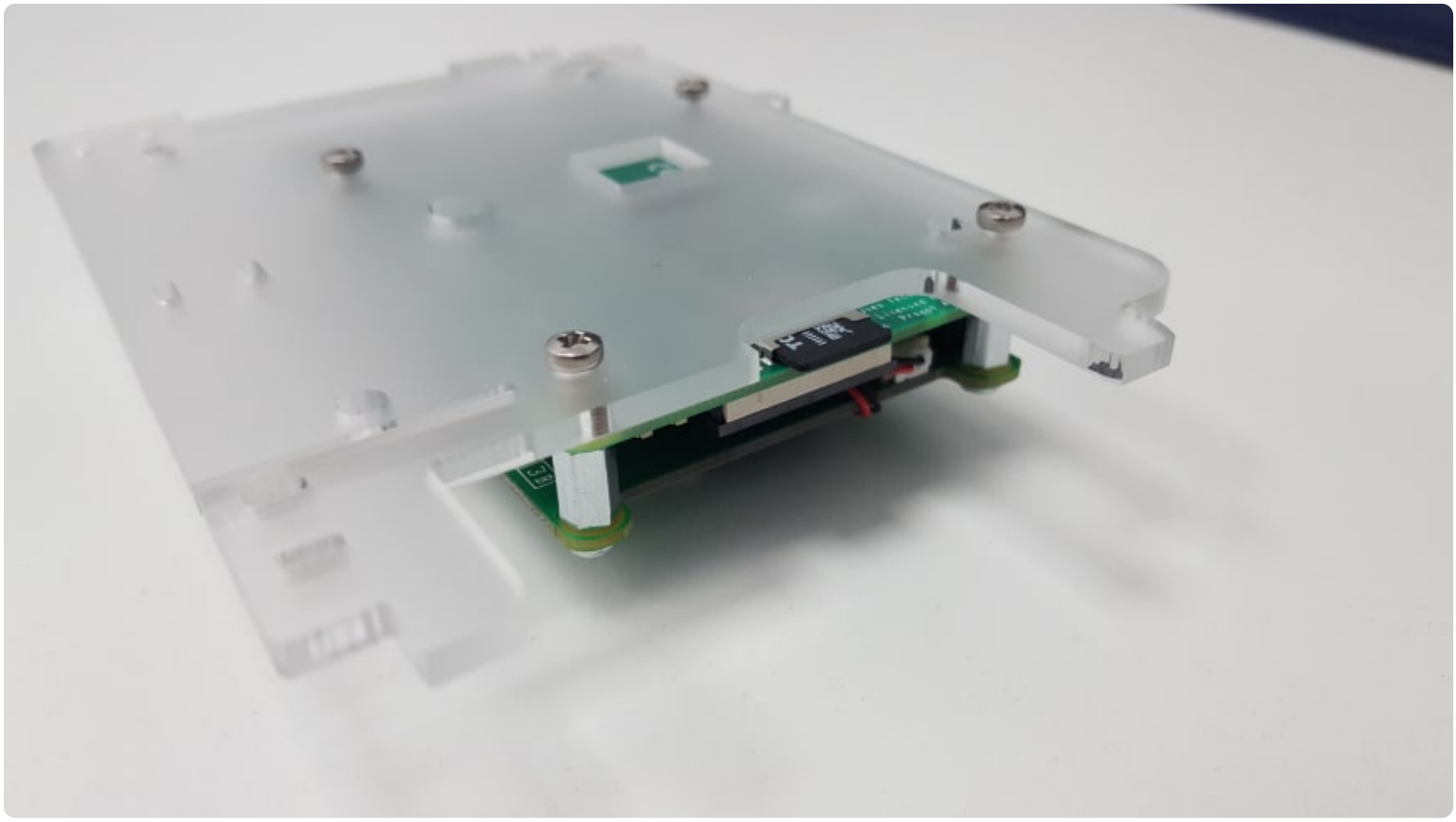
- **RaspBSD** -- An offshoot of the Berkeley Software Distribution (BSD) of UNIX for the Raspberry Pi; macOS is also BSD-based
- **Chromium OS** -- Google's browser-based operating system
- **RetroPie** -- an OS for turning your Raspberry Pi into a retro-gaming machine (no copyrighted games allowed!)
- **RISC OS Pi** -- a Pi-optimised version of RISC OS, an operating system developed specifically for Reduced Instruction Set Chips (RISC), like the AMD series used in Raspberry Pi

As fun as it would be to install RetroPie, I think I'll stick with Raspbian, as it's the OS that's recommended by The Raspberry Pi Foundation, and has been specially optimised for the low-performance hardware of the Raspberry Pi.

Raspbian's **NOOBS** (New Out Of the Box Software) installer is the easiest way to get Raspbian onto a Raspberry Pi. Just download it, and follow the instructions listed on Raspberry Pi's

website. You'll need to format the microSD card as FAT (**not ExFAT**!), extract the NOOBS `*.zip` archive, and copy the files within it to your newly-formatted microSD card. Then, insert the microSD card into the Pi; there's a slot on the underside of the Pi, on the side opposite the USB ports:

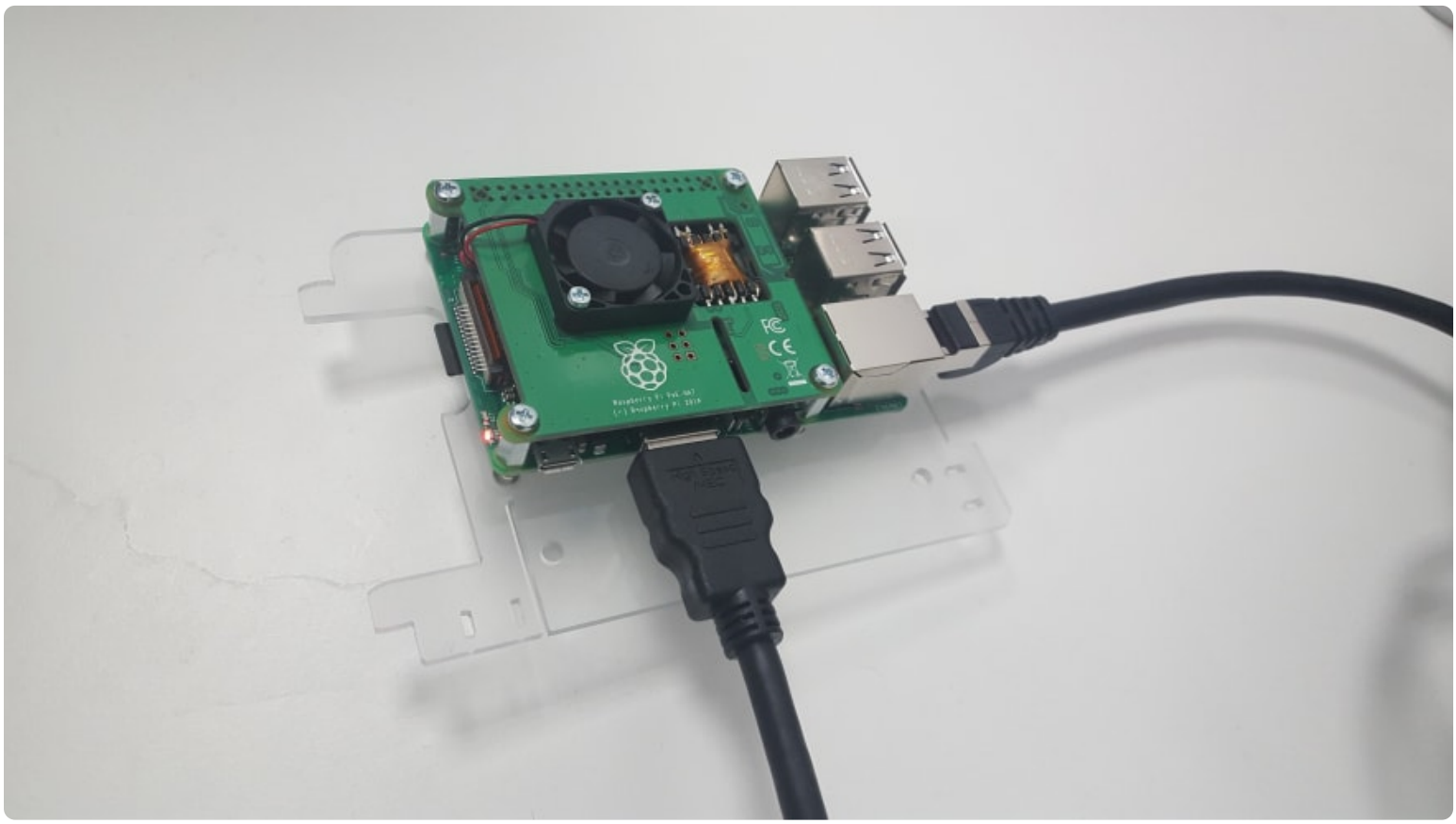




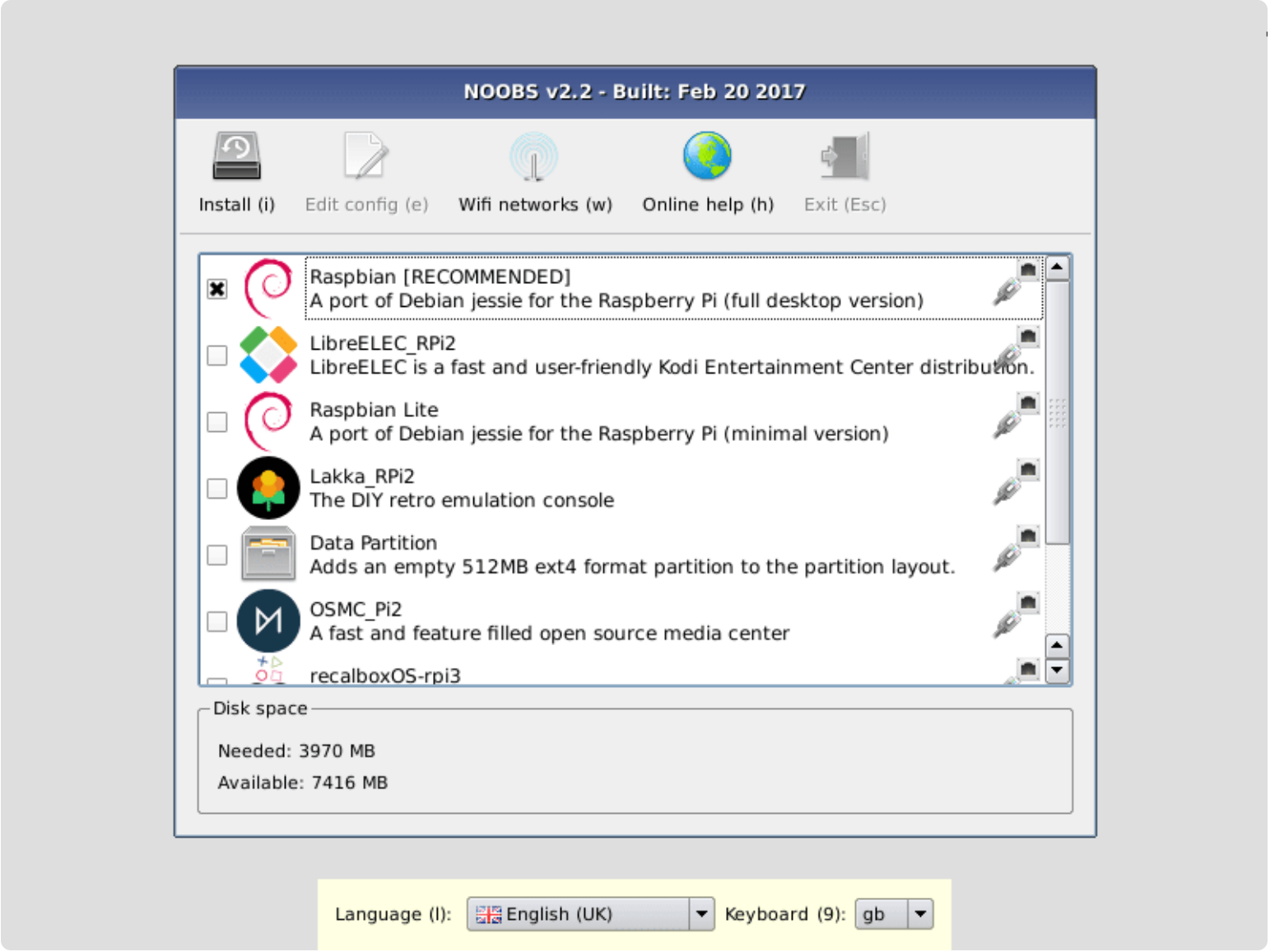
Power the Pi by plugging an Ethernet cable into it, with the other end of the cable in the PoE-enabled network switch. Plug in the network switch and turn it on. After a second or two, the Pi

should power up: the yellow light next to the Ethernet port, and the red light near the microSD card should both light up.

Then, connect the Pi to a monitor using the HDMI port on the side opposite the GPIO header.



Connect a mouse and a keyboard to the Pi via USB, and follow the on-screen instructions in the OS install wizard (it should look similar to the screenshot below).



NOOBS Installer on a Raspberry Pi

Once you've successfully installed your OS of choice on a single Pi, you can simply clone the microSD card to install the same OS on the other Pis. There's no need to go through the install wizard multiple times. Later on in this tutorial, I'll explain how to easily run a particular command (including installing software with `apt`) simultaneously across all Pis on the cluster, so you don't need to repeat the same manual tasks over and over like a monkey at a typewriter -- this is why we have technology, isn't it?

Networking

Configuring Static IP Addresses

To facilitate easy networking of the Pis, I'm going to set static IP addresses for each Pi on the network switch. I'll number the Pis 1-8 (inclusive) according to their positions on the network switch and in the carrying case. When looking at the ports on the Pis (the "front" of the case), #1 is the rightmost Pi and #8 is the leftmost Pi.

To enable user-defined, static IP addresses, I edit the file `/etc/dhcpcd.conf` on each Pi and uncomment / edit the lines:

```
interface eth0
static ip_address=192.168.0.10X/24
```

...where `X` should be replaced by `1` for Pi #1, `2` for Pi #2, etc. After this change has been made on a particular Pi, I reboot the machine. Once this is done for all eight Pis, they should all be able to `ping` each other at those addresses.

I've also installed `nmap` on Pi #1 so that the status of all eight Pis can be easily checked:

```
$ sudo nmap -sP 192.168.0.0/24
```

...will show the status of all seven other Pis, not including the one on which the command is run. The "`N hosts up`" shows how many Pis are properly configured according to the above, currently connected, and powered on.

Enabling `ssh`

To enable `ssh` on each Pi, we need to follow these instructions (reproduced here to avoid link rot):

> As of the November 2016 release, Raspbian has the SSH server disabled by default. It can be enabled manually from the desktop:
>
> 1. Launch `Raspberry Pi Configuration` from the `Preferences` menu
> 2. Navigate to the `Interfaces` tab
> 3. Select `Enabled` next to `SSH`
> 4. Click `OK`
>
> Alternatively, `raspi-config` can be used in the terminal:
>
> 1. Enter `sudo raspi-config` in a terminal window
> 2. Select `Interfacing Options`
> 3. Navigate to and select `SSH`
> 4. Choose `Yes`
> 5. Select `Ok`
> 6. Choose `Finish`
>
> Alternatively, use systemctl to start the service
>
> ```
> $ sudo systemctl enable ssh
> $ sudo systemctl start ssh
> ```
>
> When enabling SSH on a Pi that may be connected to the Internet, you should change its default password to ensure that it remains secure. See the Security page for more details.

Hostnames

Initially, all of the Pis are known as `raspberrypi`, and have a single user, `pi`:

```
$ hostname
raspberrypi

$ whoami
pi
```

This has the potential to become very confusing if we're constantly moving back and forth between the different Pis on the network. To simplify this, we're going to assign each Pi a hostname based on its position in the case / on the network switch. Pi #1 will be known as `pi1`, Pi #2 as `pi2`, and so on.

To accomplish this, two files must be edited: `/etc/hosts` and `/etc/hostname`. Within those files, there should be only one occurrence each of `raspberrypi`, which is the default hostname. We change each of those to `piX` where `X` is the appropriate number 1-8. Finally, in `/etc/hosts` only, we also add the IPs for all the other Pis at the end of the file, like:

```
192.168.0.101 pi1
192.168.0.102 pi2
...
192.168.0.108 pi8
```

This must be done manually on each Pi.

Once the above tasks are finished for the appropriate Pi, we reboot that particular Pi. Now, when the terminal is opened, instead of:

```
pi@raspberrypi:~ $
```

...you should see

```
pi@piX:~ $
```

...where `X` is the index of that Pi on the cluster. We do this on each Pi and reboot each of them after it's been done. From now on in these instructions, the command prompt will be abbreviated to simply `$`. Any other code is output or the text of a file.

Simplifying `ssh`

To connect from one Pi to another, having followed only the above instructions, would require the following series of commands:

```
$ ssh pi@192.168.0.10X
pi@192.168.0.10X's password: <enter password - 'raspberry' default>
```

Granted, this is not too much typing, but if we have to do it very often, it could become cumbersome. To avoid this, we can set up `ssh` aliases and passwordless `ssh` connections with public/private key pairs.

ssh aliases

To set up an `ssh` alias, we edit the `~/.ssh/config` file on a particular Pi and add the following lines:

```
Host piX
User pi
Hostname 192.168.0.10X
```

...replacing `x` with 1-8 for each of the eight Pis. Note that this is done on a single Pi, so that one Pi should have eight chunks of code within `~/.ssh/config`, which look identical to the above except for the `x` character, which should change for each Pi on the network. Then, the `ssh` command sequence becomes just:

```
$ ssh piX
pi@192.168.0.10X's password: <enter password>
```

This can be simplified further by setting up public/private key pairs.

public/private key pairs

On each Pi, run the following command:

```
$ ssh-keygen -t ed25519
```

This will generate a public / private key pair within the directory `~/.ssh/` which can be used to securely `ssh` without entering a password. One of these files will be called `id_ed25519`, this is the *private key*. The other, `id_ed25519.pub` is the *public key*. No passphrase is necessary to protect access to the key pair. The public key is used to communicate with the other Pis, and the private key never leaves its host machine and should never be moved or copied to any other device.

Each public key will need to be concatenated to the `~/.ssh/authorized_keys` file on every other Pi. It's easiest to do this once, for a single Pi, then simply copy the `authorized_keys` file to the other Pis. Let's assume that Pi #1 will contain the "master" record, which is then copied to the other Pis.

On Pi #2 (and #3, #4, etc.), run the following command:

```
$ cat ~/.ssh/id_ed25519.pub | ssh pi@192.168.0.101 'cat >> .ssh/authorized_keys'
```

This concatenates Pi #2's public key file to Pi #1's list of authorized keys, giving Pi #2 permission to `ssh` into Pi #1 without a password (the public and private keys are instead used to validate the connection). We need to do this for each machine, concatenating each public key file to Pi #1's list of authorized keys. We should also do this for Pi #1, so that when we copy the completed `authorized_keys` file to the other Pis, they all have permission to `ssh` into Pi #1, as well. Run the following command on Pi #1:

```
$ cat .ssh/id_ed25519.pub >> .ssh/authorized_keys
```

Once this is done, as well as the previous section, `ssh`-ing is as easy as:

```
$ ssh pi1
```

...and that's it! Additional aliases can be configured in the `~/.bashrc` file to shorten this further (see below) though this is not configured on our system:

```
alias p1="ssh pi1" # etc.
```

replicate the configuration

Finally, to replicate the passwordless `ssh` across all Pis, simply copy the two files mentioned above from Pi #1 to each other Pi using `scp`:

```
$ scp ~/.ssh/authorized_keys piX:~/.ssh/authorized_keys
$ scp ~/.ssh/config piX:~/.ssh/config
```

You should now be able to `ssh` into any Pi on the cluster from any other Pi with just `ssh piX`.

Ease of Use

Finally, a few miscellaneous ease-of-use-type enhancements for the cluster.

get the hostname of every Pi except this one

To get the hostname of a Pi, you can just use:

```
$ hostname
pi1
```

...to get the hostname of all *other* Pis on the cluster, define the function:

```
function otherpis {
  grep "pi" /etc/hosts | awk '{print $2}' | grep -v $(hostname)
}
```

> The `-v` flag tells `grep` to *invert* the selection. Only lines which *don't match* the `hostname` are returned.

...in your `~/.bashrc`, then, `source` your `~/.bashrc` file with:

```
$ source ~/.bashrc
```

(Note that, whenever you edit `~/.bashrc`, for those changes to take effect, you must `source` the file or log out and log back in.) Then, you can call the new function on the command line with:

```
$ otherpis
pi2
pi3
pi4
...
```

Note that this function relies on you having listed all of the IPs and hostnames of the Pis within the `/etc/hosts` file.

send the same command to all Pis

To send the same command to each Pi, add the following function to the `~/.bashrc` of the Pi from which you want to dictate commands (I add this to Pi #1's `~/.bashrc` and then copy the `~/.bashrc` file to all other Pis using the instructions below):

```
function clustercmd {
  for pi in $(otherpis); do ssh $pi "$@"; done
  $@
}
```

This will run the given command on each other Pi, and then on this Pi:

```
$ clustercmd date
Tue Apr  9 00:32:41 IST 2019
Tue Apr  9 05:58:07 IST 2019
Tue Apr  9 06:23:51 IST 2019
Tue Apr  9 23:51:00 IST 2019
Tue Apr  9 05:58:57 IST 2019
Tue Apr  9 07:36:13 IST 2019
Mon Apr  8 15:19:32 IST 2019
Wed Apr 10 03:48:11 IST 2019
```

...we can see above that all of our Pis have different system times. Let's fix that.

synchronise time across cluster

Simply tell each Pi to install the package `htpdate` and the date will be updated. That's it:

```
$ clustercmd "sudo apt install htpdate -y > /dev/null 2>&1"

$ clustercmd date
Wed Jun 19 16:04:22 IST 2019
Wed Jun 19 16:04:18 IST 2019
Wed Jun 19 16:04:19 IST 2019
Wed Jun 19 16:04:20 IST 2019
Wed Jun 19 16:04:48 IST 2019
Wed Jun 19 16:04:12 IST 2019
Wed Jun 19 16:04:49 IST 2019
Wed Jun 19 16:04:25 IST 2019
```

Notice how all the dates and times are accurate to within a minute now. If we reboot all of the Pis, they'll be even more closely aligned:

```
$ clustercmd date
Wed Jun 19 16:09:28 IST 2019
Wed Jun 19 16:09:27 IST 2019
Wed Jun 19 16:09:29 IST 2019
Wed Jun 19 16:09:31 IST 2019
Wed Jun 19 16:09:28 IST 2019
Wed Jun 19 16:09:30 IST 2019
Wed Jun 19 16:09:24 IST 2019
Wed Jun 19 16:09:28 IST 2019
```

They're now aligned to within 10 seconds. Note that it takes a few seconds to run the command across the cluster. To precisely synchronise all clocks to a remote server (a respected standard like `time.nist.gov` or `google.com`) you can do:

```
$ clustercmd sudo htpdate -a -l time.nist.gov
```

...this will take a few minutes, because the program slowly adjusts the clock in intervals of < 30ms so there are no "jumps" in any system file timestamps. Once this command finishes, the clocks will be synchronised (remember, it takes a second or two to communicate across the network, so they're still "off" by two or three seconds):

```
$ clustercmd date
Wed Jun 19 16:36:46 IST 2019
Wed Jun 19 16:36:47 IST 2019
Wed Jun 19 16:36:47 IST 2019
Wed Jun 19 16:36:48 IST 2019
Wed Jun 19 16:36:49 IST 2019
Wed Jun 19 16:36:49 IST 2019
Wed Jun 19 16:36:49 IST 2019
Wed Jun 19 16:36:49 IST 2019
```

reboot the cluster

I add the following function to my `~/.bashrc` on Pi #1:

```
function clusterreboot {
  clustercmd sudo shutdown -r now
}
```

This function makes it easy to reboot the entire cluster. Another one lets me shut down the entire cluster without rebooting:

```
function clustershutdown {
  clustercmd sudo shutdown now
}
```

send the same file to all Pis

To copy a file from one Pi to all others, let's add a function called `clusterscp` (cluster secure copy) to the `~/.bashrc` file of any particular Pi:

```
function clusterscp {
  for pi in $(otherpis); do
    cat $1 | ssh $pi "sudo tee $1" > /dev/null 2>&1
  done
}
```

Then we can copy all of the ease-of-use functions we've defined in `~/.bashrc` to every other Pi on the cluster:

```
$ source ~/.bashrc && clusterscp ~/.bashrc
```

Securing the Cluster

You might only be building a Hadoop cluster for fun, but I'm building one to do some data analysis for work. Since my cluster is going to hold proprietary data, cluster security is paramount.

There is a lot we can do to secure a computer against unauthorised access. The easiest (and most effective) thing that can be done (after setting up passwordless `ssh` following the instructions in the above section "Simplifying `ssh`") is to disable password-based authentication. This means that users can only log into the system if they have a public key corresponding to one of the private keys on one of the Pis. While it is possible (in theory) to crack these keys, ed25519 is the most secure public/private key pair algorithm to date and will likely not be breakable within the foreseeable future.

Disable Password-Based Authentication & `root` Login

To disable password-based authentication, we edit the file `/etc/ssh/sshd_config`. There are 6 options we want to change in particular. Search for the following keys in this file and change them so that they match the examples below. Make sure that none of the below lines begins with a '`#`', this comments out the line so that it is ignored by the system.

```
PermitRootLogin no
PasswordAuthentication no
ChallengeResponseAuthentication no
UsePAM no
X11Forwarding no
PrintMotd no
```

The above commands will, respectively: prevent hackers from trying to login as `root` (so they'll need to know that the username is `pi`); disallow password-based authentication (so they'll need the corresponding ed25519 public key to the particular private key on a given Pi); disable X11 (GUI) forwarding among / from / to the Pis (everything must be done command-line); and disable the "message of the day" (MOTD), which can sometimes include compromising information. Edit `/etc/ssh/sshd_config` and make the above changes on a particular Pi, then copy those changes to all other Pis with...

WARNING: BE EXTREMELY CAREFUL WITH THE FOLLOWING STEPS. If you make a typo in `sshd_config`, then restart the `ssh` service on a Pi, it will throw an error and you will not be able to `ssh` into that Pi any more. You will need to manually (connect the HDMI cable to the video output) connect to a Pi to reconfigure!

```
$ clusterscp /etc/ssh/sshd_config
```

...and restart the `ssh` service (reload the configuration) across all Pis with:

```
$ clustercmd sudo service ssh restart
```

To remove the MOTD entirely, delete the file `/etc/motd`:

```
$ clustercmd sudo rm /etc/motd
```

Monitor `ssh` Activity with `fail2ban`

Another big thing we can do to monitor and protect the integrity of the cluster is install a program called `fail2ban`. `fail2ban` "logs and temporarily ban[s] IPs based on possible malicious activity". To install this program on every machine in the cluster, run the command:

```
$ clustercmd sudo apt install fail2ban -y
```

Then, copy the configuration file on a single Pi:

```
$ cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

...and edit the lines within `jail.local` near the spot that starts with `[sshd]`. Note that `[sshd]` appears near the top of the file in a commented block – ignore that. `[sshd]` should be configured as follows:

```
[sshd]
enabled = true
port    = ssh
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

Once you've finished editing `jail.local`, copy it to all other Pis on the cluster with:

```
$ clusterscp /etc/fail2ban/jail.local
```

...and restart the `fail2ban` service on all Pis with:

```
$ clustercmd sudo service fail2ban restart
```

[back to top]

Hadoop & Spark

Single-Node Setup

Hadoop

Apache Hadoop v2.7+ requires Java 7+ to run. When I installed Raspbian on my Pis, I used NOOBS v3.0.1 (2019-04-08). This installs Raspbian and also Java 1.8.0_65 (Java 8 by HotSpot aka. Oracle). So all Pis have an acceptable version of Java installed by default. Next, we can download and install Hadoop.

I'll start by building a single-node setup on the master node (Pi #1), then I'll the worker nodes to create a multi-node cluster. On Pi #1, get Hadoop with the commands:

```
$ cd && wget https://bit.ly/2wa3Hty
```

(This is a shortened link to `hadoop-3.2.0.tar.gz`)

```
$ sudo tar -xvf 2wa3Hty -C /opt/
$ rm 2wa3Hty && cd /opt
$ sudo mv hadoop-3.2.0 hadoop
```

Then, make sure to change the permissions on this directory:

```
$ sudo chown pi:pi -R /opt/hadoop
```

Finally, add this directory to the `$PATH` by editing `~/.bashrc` and putting the following lines at the end of the file:

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
export HADOOP_HOME=/opt/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

...and edit `/opt/hadoop/etc/hadoop/hadoop-env.sh` to add the following line:

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
```

You can verify that Hadoop has been installed correctly by checking the version:

```
$ cd && hadoop version | grep Hadoop
Hadoop 3.2.0
```

Spark

We will download Spark in a similar manner to how we downloaded Hadoop, above:

```
$ cd && wget https://bit.ly/2HK6nTW
```

(This is a shortened link to `spark-2.4.3-bin-hadoop2.7.tgz`)

```
$ sudo tar -xvf 2HK6nTW -C /opt/
$ rm 2HK6nTW && cd /opt
$ sudo mv spark-2.4.3-bin-hadoop2.7 spark
```

Then, make sure to change the permissions on this directory:

```
$ sudo chown pi:pi -R /opt/spark
```

Finally, add this directory to your `$PATH` by editing `~/.bashrc` and putting the following lines at the end of the file:

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin
```

You can verify that Spark has been installed correctly by checking the version:

```
$ cd && spark-shell --version
... version 2.4.3 ... Using Scala version 2.11.12 ...
```

HDFS

To get the Hadoop Distributed File System (HDFS) up and running, we need to modify some configuration files. All of these files are within `/opt/hadoop/etc/hadoop`. The first is **`core-site.xml`**. Edit it so it looks like the following:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://pi1:9000</value>
  </property>
</configuration>
```

The next is **`hdfs-site.xml`**, which should look like:

```
<configuration>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///opt/hadoop_tmp/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///opt/hadoop_tmp/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

...this configures where the DataNode and NameNode information is stored and also sets the replication (the number of times a block is copied across the cluster) to 1 (we will change this later). Make sure that you also create these directories with the commands:

```
$ sudo mkdir -p /opt/hadoop_tmp/hdfs/datanode
$ sudo mkdir -p /opt/hadoop_tmp/hdfs/namenode
```

...and adjust the owner of these directories:

```
$ sudo chown pi:pi -R /opt/hadoop_tmp
```

The next file is **`mapred-site.xml`**, which should look like:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

...and finally `yarn-site.xml`, which should look like:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

Once these four files are edited, we can format the HDFS (**WARNING: DO NOT DO THIS IF YOU ALREADY HAVE DATA IN THE HDFS! IT WILL BE LOST!**):

```
$ hdfs namenode -format -force
```

...we then boot the HDFS with the following two commands:

```
$ start-dfs.sh
$ start-yarn.sh
```

...and test that it's working by creating a temporary directory:

```
$ hadoop fs -mkdir /tmp

$ hadoop fs -ls /
Found 1 items
drwxr-xr-x   - pi supergroup          0 2019-04-09 16:51 /tmp
```

...or by running the command `jps`:

```
$ jps
2736 NameNode
2850 DataNode
3430 NodeManager
3318 ResourceManager
3020 SecondaryNameNode
3935 Jps
```

This shows that the HDFS is up and running, at least on Pi #1. To check that Spark and Hadoop are working together, we can do:

```
$ hadoop fs -put $SPARK_HOME/README.md /

$ spark-shell
```

...this will open the Spark shell, with prompt `scala>`:

```
scala> val textFile = sc.textFile("hdfs://pi1:9000/README.md")
...

scala> textFile.first()
res0: String = # Apache Spark
```

Hide `execstack` Warning

While running the Hadoop commands above, you may get a warning like...

> "You have loaded library... which might have disabled stack guard."

...the reason this happens on our Raspberry Pis is because of a mismatch between the 32-bit runtime the Hadoop binaries were built for, and the 64-bit Raspbian version we're running. To ignore these warnings, change the line

```
# export HADOOP_OPTS="-Djava.net.preferIPv4Stack=true"
```

...in `/opt/hadoop/etc/hadoop/hadoop-env.sh` to

```
export HADOOP_OPTS="-XX:-PrintWarnings -Djava.net.preferIPv4Stack=true"
```

This will hide the warnings in the future. (Alternatively, you could also download the Hadoop source code and build from scratch.)

Hide `NativeCodeLoader` Warning

Another warning you may see is that `util.NativeCodeLoader` is "`Unable to load [the] native-hadoop library for your platform`". This warning cannot be resolved easily. It's due to the fact that Hadoop is compiled for 32-bit architectures and the version of Raspbian we have is 64-bit. We could recompile the library from scratch on the 64-bit machine, but I'd rather just not see the warning. To do this, we can add the following lines to the bottom of our `~/.bashrc` file:

```
export HADOOP_HOME_WARN_SUPPRESS=1
export HADOOP_ROOT_LOGGER="WARN,DRFA"
```

This will prevent those `NativeCodeLoader` warnings from being printed.

Cluster Setup

At this point, you should have a single-node cluster and that single node acts as both a master and a worker node. To set up the worker nodes (and distribute computing to the entire cluster), we must take the following steps...

Create the Directories

Create the required directories on all other Pis using:

```
$ clustercmd sudo mkdir -p /opt/hadoop_tmp/hdfs
$ clustercmd sudo chown pi:pi -R /opt/hadoop_tmp
$ clustercmd sudo mkdir -p /opt/hadoop
$ clustercmd sudo chown pi:pi /opt/hadoop
```

Copy the Configuration

Copy the files in /opt/hadoop to each other Pi using:

```
$ for pi in $(otherpis); do rsync -avxP $HADOOP_HOME $pi:/opt; done
```

This will take quite a long time, so go grab lunch.

When you're back, verify that the files copied correctly by querying the Hadoop version on each node with the following command:

```
$ clustercmd hadoop version | grep Hadoop
Hadoop 3.2.0
Hadoop 3.2.0
Hadoop 3.2.0
...
```

Configuring Hadoop on the Cluster

It's difficult to find a good guide for installing Hadoop across a networked cluster of machines. This link points to the one I followed, mostly without incident. To get HDFS running across the cluster, we need to modify the configuration files that we edited earlier. All of these files are within `/opt/hadoop/etc/hadoop`. The first is **`core-site.xml`**. Edit it so it looks like the following:

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://pi1:9000</value>
  </property>
</configuration>
```

The next is **`hdfs-site.xml`**, which should look like:

```
<configuration>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/opt/hadoop_tmp/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/opt/hadoop_tmp/hdfs/namenode</value>
```

```
  </property>
  <property>
    <name>dfs.replication</name>
    <value>4</value>
  </property>
</configuration>
```

The next file is **`mapred-site.xml`**, which should look like:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.resource.mb</name>
    <value>256</value>
  </property>
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>128</value>
  </property>
  <property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>128</value>
  </property>
</configuration>
```

...and finally **`yarn-site.xml`**, which should look like:

```
<configuration>
  <property>
    <name>yarn.acl.enable</name>
    <value>0</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>pi1</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>900</value>
```

```
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>900</value>
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>64</value>
  </property>
  <property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
  </property>
</configuration>
```

Make these changes to these files, then remove all old files from all Pis. I'm assuming you're working your way through this tutorial step-by-step and haven't yet added any big, important data to the single-node Pi #1 cluster. You can clean up all the Pis with:

```
$ clustercmd rm -rf /opt/hadoop_tmp/hdfs/datanode/*
$ clustercmd rm -rf /opt/hadoop_tmp/hdfs/namenode/*
```

If you don't do this, you may have errors with the Pis not recognizing each other. If you can't get the DataNodes (Pi #2-#8) to start, or communicate with the NameNode (Pi #1), cleaning the files above might be the solution (it was for me).

Next, we need to create two files in `$HADOOP_HOME/etc/hadoop/` which tell Hadoop which Pis to use as worker nodes and which Pi should be the master (NameNode) node. Create a file named `master` in the aforementioned directory and add only a single line:

```
pi1
```

Then, create a file named `workers` (**NOT** `slaves`, as was the case in previous versions of Hadoop) in the same directory and add all of the other Pis:

```
pi2
pi3
pi4
...
```

Then, you'll need to edit `/etc/hosts` again. On any Pi, remove the line which looks like

```
127.0.0.1 PiX
```

...where `x` is the index of that particular Pi. Then, copy this file to all other Pis with:

```
$ clusterscp /etc/hosts
```

We can do this now because this file is no longer Pi-specific. Finally, reboot the cluster for these changes to take effect. When all Pis have rebooted, on Pi #1, run the command:

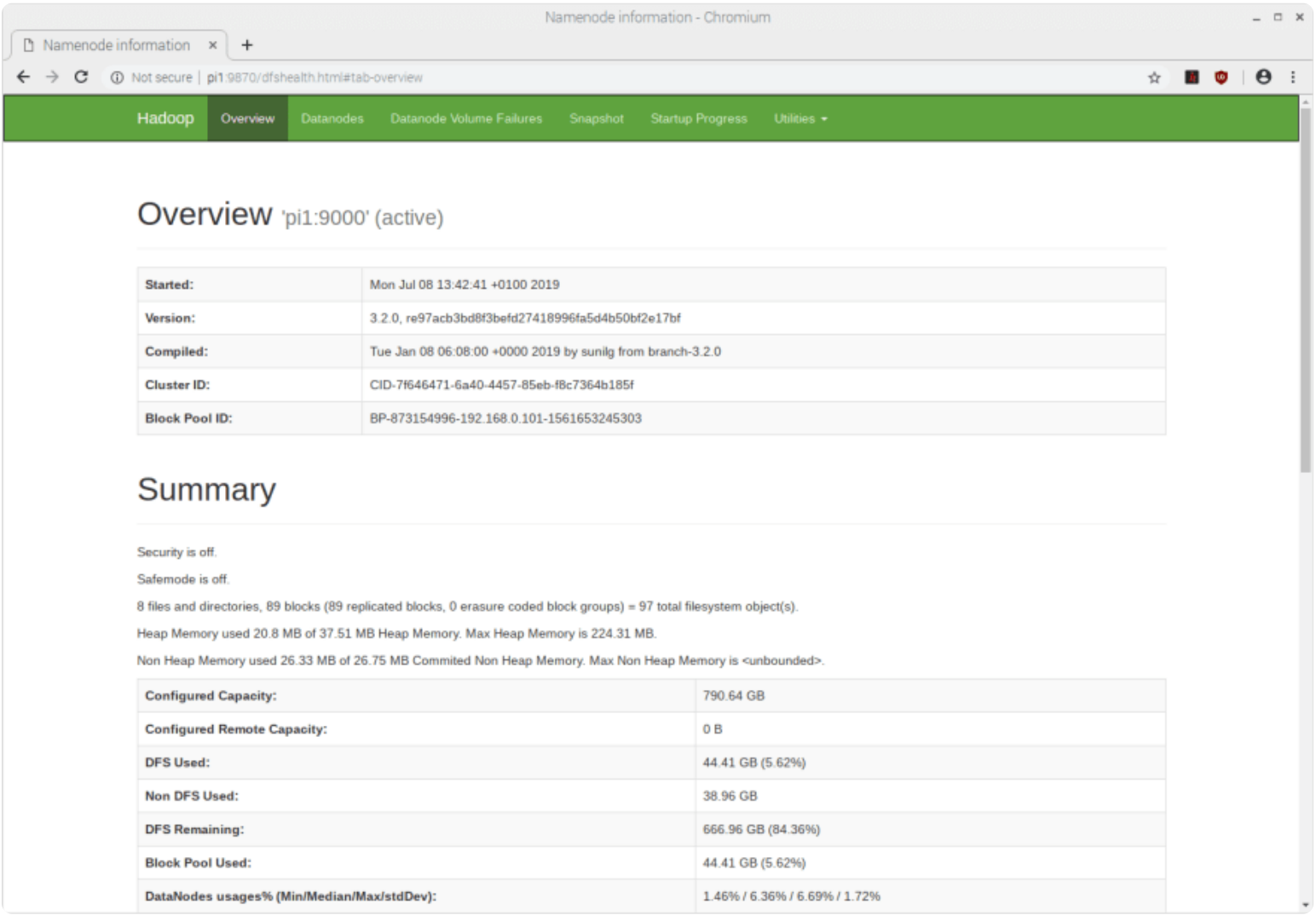
WARNING: DO NOT DO THIS IF YOU ALREADY HAVE DATA IN THE HDFS! IT WILL BE LOST!

```
$ hdfs namenode -format -force
```

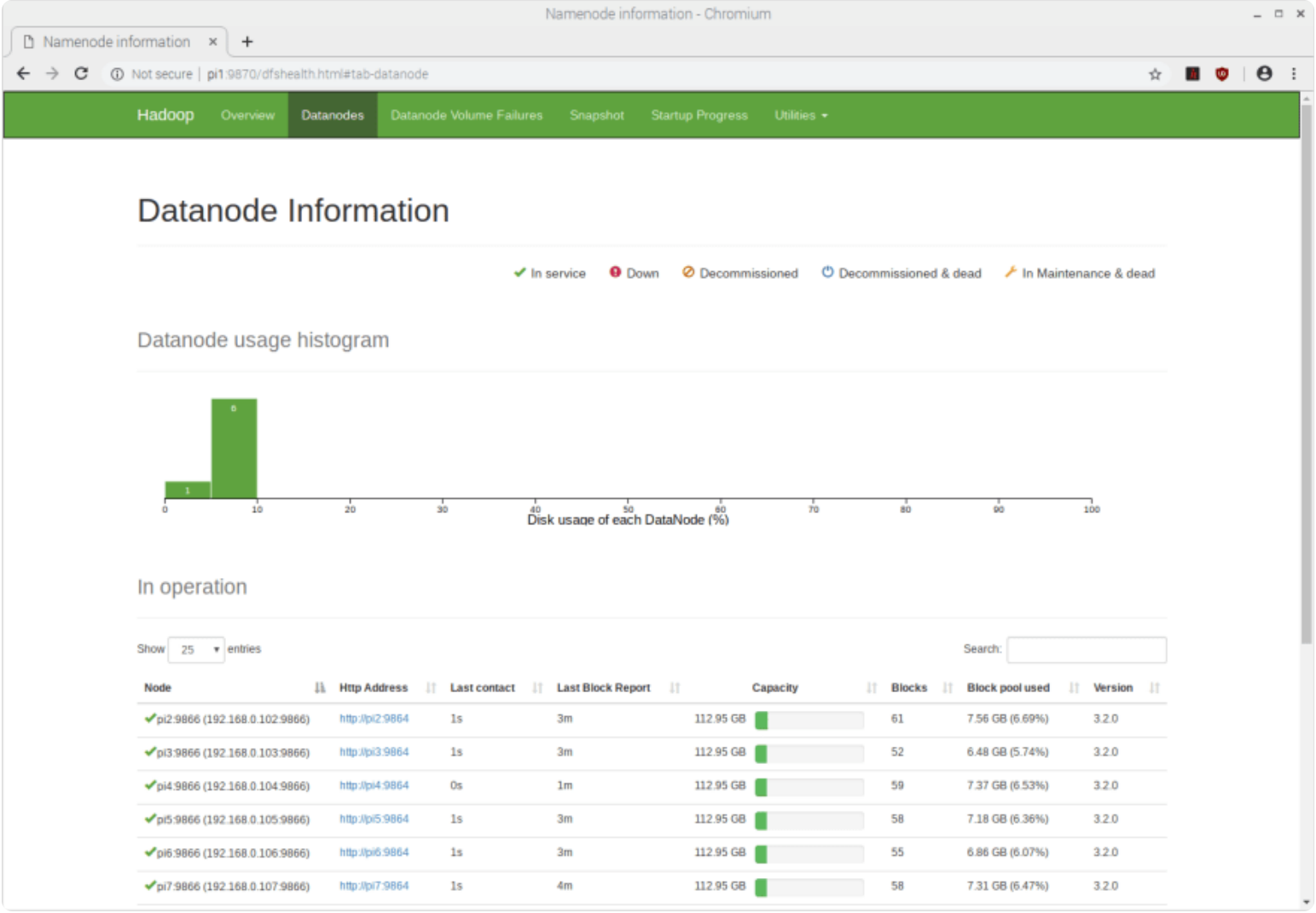
...we then boot the HDFS with the following two commands:

```
$ start-dfs.sh && start-yarn.sh
```

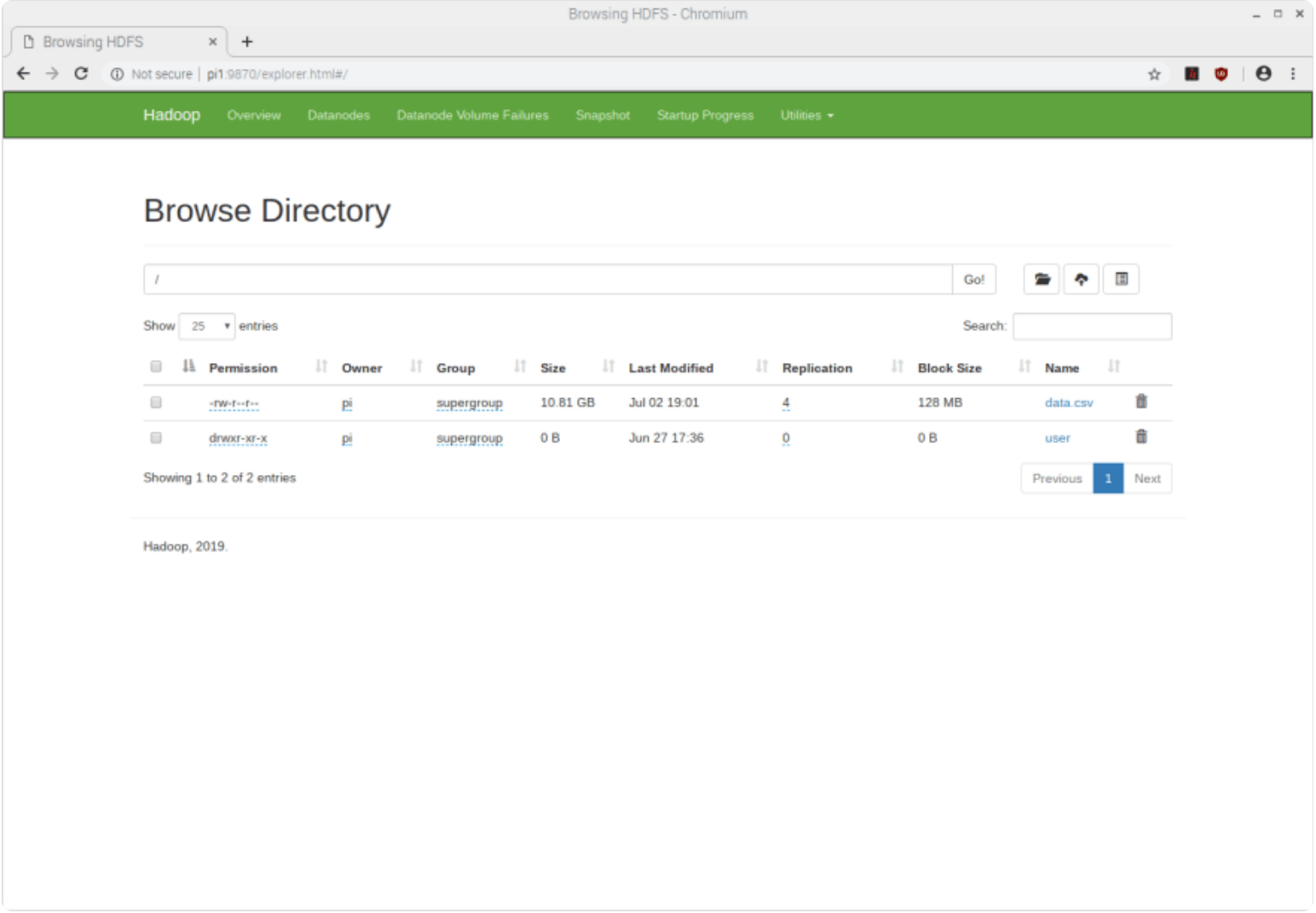
We can test the cluster by putting files in HDFS from any Pi (using `hadoop fs -put`) and making sure they show up on other Pis (using `hadoop fs -ls`). You can also check that the cluster is up and running by opening a web browser and navigating to http://pi1:9870. This web interface gives you a file explorer as well as information about the health of the cluster.



Hadoop web UI running on port 9870.



Hadoop web UI showing DataNode statistics.



File browser in Hadoop web UI.

Configuring Spark on the Cluster

Spark will run fine on a single machine, so we may trick ourselves into thinking we're using the full power of the Hadoop cluster when in reality we're not. Some of the configuration we performed above was for Hadoop YARN (Yet Another Resource Negotiator). This is a "resource negotiator" for HDFS, which orchestrates how files are moved and analysed around the cluster. For Spark to be able to communicate with YARN, we need to configure two more environment variables in Pi #1's `~/.bashrc`. Previously, we defined

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin
```

...in `~/.bashrc`. Just beneath this, we will now add two more environment variables:

```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
```

`$HADOOP_CONF_DIR` is the directory which contains all of the `*-site.xml` configuration files that we edited above. Next, we create the Spark configuration file:

```
$ cd $SPARK_HOME/conf
$ sudo mv spark-defaults.conf.template spark-defaults.conf
```

...and we add the following lines to the end of this file:

```
spark.master                yarn
spark.driver.memory         465m
spark.yarn.am.memory        356m
spark.executor.memory       465m
spark.executor.cores        4
```

The meaning of these values is explained at this link. But note that the above is **very machine-specific**. The configuration above works fine for me with the Raspberry Pi 3 Model B+, but it may not work for you (or be optimal) for a less- or more-powerful machine. Spark enforces hard lower bounds on how much memory can be allocated, as well. Where you see `465m` above, that's the minimum configurable value for that entry -- any less and Spark will refuse to run.

> A Raspberry Pi 3 Model B+ uses between 9-25\% of its RAM while idling. Since they have 926MB RAM in total, Hadoop and Spark will have access to at most about 840MB of RAM per Pi.

Once all of this has been configured, reboot the cluster. Note that, when you reboot, you should NOT format the HDFS NameNode again. Instead, simply stop and restart the HDFS service with:

```
$ stop-dfs.sh && stop-yarn.sh

$ start-dfs.sh && start-yarn.sh
```

Now, you can submit a job to Spark on the command line:

```
pi@pi1:~ $ spark-submit --deploy-mode client --class org.apache.spark.examples.SparkPi $SPARK_H
OpenJDK Client VM warning: You have loaded library /opt/hadoop/lib/native/libhadoop.so.1.0.0 wh
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with
2019-07-08 14:01:24,408 WARN util.NativeCodeLoader: Unable to load native-hadoop library for yo
2019-07-08 14:01:25,514 INFO spark.SparkContext: Running Spark version 2.4.3
2019-07-08 14:01:25,684 INFO spark.SparkContext: Submitted application: Spark Pi
2019-07-08 14:01:25,980 INFO spark.SecurityManager: Changing view acls to: pi
2019-07-08 14:01:25,980 INFO spark.SecurityManager: Changing modify acls to: pi
2019-07-08 14:01:25,981 INFO spark.SecurityManager: Changing view acls groups to:
2019-07-08 14:01:25,981 INFO spark.SecurityManager: Changing modify acls groups to:
2019-07-08 14:01:25,982 INFO spark.SecurityManager: SecurityManager: authentication disabled; u
2019-07-08 14:01:27,360 INFO util.Utils: Successfully started service 'sparkDriver' on port 460
2019-07-08 14:01:27,491 INFO spark.SparkEnv: Registering MapOutputTracker
2019-07-08 14:01:27,583 INFO spark.SparkEnv: Registering BlockManagerMaster
2019-07-08 14:01:27,594 INFO storage.BlockManagerMasterEndpoint: Using org.apache.spark.storage
2019-07-08 14:01:27,596 INFO storage.BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
2019-07-08 14:01:27,644 INFO storage.DiskBlockManager: Created local directory at /tmp/blockmgr
```

```
2019-07-08 14:01:27,763 INFO memory.MemoryStore: MemoryStore started with capacity 90.3 MB
2019-07-08 14:01:28,062 INFO spark.SparkEnv: Registering OutputCommitCoordinator
2019-07-08 14:01:28,556 INFO util.log: Logging initialized @10419ms
2019-07-08 14:01:28,830 INFO server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git
2019-07-08 14:01:28,903 INFO server.Server: Started @10770ms
2019-07-08 14:01:28,997 INFO server.AbstractConnector: Started ServerConnector@89f072{HTTP/1.1,
2019-07-08 14:01:28,997 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
2019-07-08 14:01:29,135 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1b32
2019-07-08 14:01:29,137 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@b726
2019-07-08 14:01:29,140 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@34b7
2019-07-08 14:01:29,144 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1e88
2019-07-08 14:01:29,147 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@b317
2019-07-08 14:01:29,150 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@165e
2019-07-08 14:01:29,153 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1ae4
2019-07-08 14:01:29,158 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@5a54
2019-07-08 14:01:29,161 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1ef7
2019-07-08 14:01:29,165 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1d9b
2019-07-08 14:01:29,168 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1489
2019-07-08 14:01:29,179 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@5672
2019-07-08 14:01:29,186 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@362c
2019-07-08 14:01:29,191 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@4ee9
2019-07-08 14:01:29,195 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1c47
2019-07-08 14:01:29,200 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@a360
2019-07-08 14:01:29,204 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@148b
2019-07-08 14:01:29,209 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@27ba
2019-07-08 14:01:29,214 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@336c
2019-07-08 14:01:29,217 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@156f
2019-07-08 14:01:29,260 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1e52
2019-07-08 14:01:29,265 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@159e
2019-07-08 14:01:29,283 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1dc9
2019-07-08 14:01:29,288 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@c494
2019-07-08 14:01:29,292 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@7728
2019-07-08 14:01:29,304 INFO ui.SparkUI: Bound SparkUI to 0.0.0.0, and started at http://pi1:40
2019-07-08 14:01:29,452 INFO spark.SparkContext: Added JAR file:/opt/spark/examples/jars/spark-
2019-07-08 14:01:33,070 INFO client.RMProxy: Connecting to ResourceManager at pi1/192.168.0.101
2019-07-08 14:01:33,840 INFO yarn.Client: Requesting a new application from cluster with 7 Node
2019-07-08 14:01:34,082 INFO yarn.Client: Verifying our application has not requested more than
2019-07-08 14:01:34,086 INFO yarn.Client: Will allocate AM container, with 740 MB memory includ
2019-07-08 14:01:34,089 INFO yarn.Client: Setting up container launch context for our AM
2019-07-08 14:01:34,101 INFO yarn.Client: Setting up the launch environment for our AM containe
2019-07-08 14:01:34,164 INFO yarn.Client: Preparing resources for our AM container
2019-07-08 14:01:35,577 WARN yarn.Client: Neither spark.yarn.jars nor spark.yarn.archive is set
2019-07-08 14:02:51,027 INFO yarn.Client: Uploading resource file:/tmp/spark-ca0b9022-2ba9-45ff
2019-07-08 14:04:09,654 INFO yarn.Client: Uploading resource file:/tmp/spark-ca0b9022-2ba9-45ff
2019-07-08 14:04:13,226 INFO spark.SecurityManager: Changing view acls to: pi
2019-07-08 14:04:13,227 INFO spark.SecurityManager: Changing modify acls to: pi
2019-07-08 14:04:13,227 INFO spark.SecurityManager: Changing view acls groups to:
2019-07-08 14:04:13,228 INFO spark.SecurityManager: Changing modify acls groups to:
2019-07-08 14:04:13,228 INFO spark.SecurityManager: SecurityManager: authentication disabled; u
2019-07-08 14:04:20,235 INFO yarn.Client: Submitting application application_1562589758436_0002
2019-07-08 14:04:20,558 INFO impl.YarnClientImpl: Submitted application application_15625897584
```

```
2019-07-08 14:04:20,577 INFO cluster.SchedulerExtensionServices: Starting Yarn extension servic
2019-07-08 14:04:21,625 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:04:21,680 INFO yarn.Client:
     client token: N/A
     diagnostics: [Mon Jul 08 14:04:20 +0100 2019] Scheduler has assigned a container for AM, w
     ApplicationMaster host: N/A
     ApplicationMaster RPC port: -1
     queue: default
     start time: 1562591060331
     final status: UNDEFINED
     tracking URL: http://pi1:8088/proxy/application_1562589758436_0002/
     user: pi
2019-07-08 14:04:22,696 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:04:23,711 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:04:24,725 INFO yarn.Client: Application report for application_1562589758436_0002
...
2019-07-08 14:05:45,863 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:05:46,875 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:05:47,883 INFO yarn.Client: Application report for application_1562589758436_0002
2019-07-08 14:05:47,884 INFO yarn.Client:
     client token: N/A
     diagnostics: N/A
     ApplicationMaster host: 192.168.0.103
     ApplicationMaster RPC port: -1
     queue: default
     start time: 1562591060331
     final status: UNDEFINED
     tracking URL: http://pi1:8088/proxy/application_1562589758436_0002/
     user: pi
2019-07-08 14:05:47,891 INFO cluster.YarnClientSchedulerBackend: Application application_156258
2019-07-08 14:05:47,937 INFO util.Utils: Successfully started service 'org.apache.spark.network
2019-07-08 14:05:47,941 INFO netty.NettyBlockTransferService: Server created on pi1:46437
2019-07-08 14:05:47,955 INFO storage.BlockManager: Using org.apache.spark.storage.RandomBlockRe
2019-07-08 14:05:48,178 INFO storage.BlockManagerMaster: Registering BlockManager BlockManagerI
2019-07-08 14:05:48,214 INFO storage.BlockManagerMasterEndpoint: Registering block manager pi1:
2019-07-08 14:05:48,265 INFO storage.BlockManagerMaster: Registered BlockManager BlockManagerId
2019-07-08 14:05:48,269 INFO storage.BlockManager: Initialized BlockManager: BlockManagerId(dri
2019-07-08 14:05:49,426 INFO cluster.YarnClientSchedulerBackend: Add WebUI Filter. org.apache.h
2019-07-08 14:05:49,441 INFO ui.JettyUtils: Adding filter org.apache.hadoop.yarn.server.webprox
2019-07-08 14:05:49,816 INFO ui.JettyUtils: Adding filter org.apache.hadoop.yarn.server.webprox
2019-07-08 14:05:49,829 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@136b
2019-07-08 14:05:49,935 INFO cluster.YarnClientSchedulerBackend: SchedulerBackend is ready for
2019-07-08 14:05:50,076 INFO cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: ApplicationMas
2019-07-08 14:05:52,074 INFO spark.SparkContext: Starting job: reduce at SparkPi.scala:38
2019-07-08 14:05:52,479 INFO scheduler.DAGScheduler: Got job 0 (reduce at SparkPi.scala:38) wit
2019-07-08 14:05:52,481 INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (reduce at Spar
2019-07-08 14:05:52,485 INFO scheduler.DAGScheduler: Parents of final stage: List()
2019-07-08 14:05:52,492 INFO scheduler.DAGScheduler: Missing parents: List()
2019-07-08 14:05:52,596 INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD
2019-07-08 14:05:53,314 WARN util.SizeEstimator: Failed to check whether UseCompressedOops is s
2019-07-08 14:05:53,404 INFO memory.MemoryStore: Block broadcast_0 stored as values in memory (
```

```
2019-07-08 14:05:53,607 INFO memory.MemoryStore: Block broadcast_0_piece0 stored as bytes in me
2019-07-08 14:05:53,625 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on pi
2019-07-08 14:05:53,639 INFO spark.SparkContext: Created broadcast 0 from broadcast at DAGSched
2019-07-08 14:05:53,793 INFO scheduler.DAGScheduler: Submitting 7 missing tasks from ResultStag
2019-07-08 14:05:53,801 INFO cluster.YarnScheduler: Adding task set 0.0 with 7 tasks
2019-07-08 14:06:08,910 WARN cluster.YarnScheduler: Initial job has not accepted any resources;
2019-07-08 14:06:23,907 WARN cluster.YarnScheduler: Initial job has not accepted any resources;
2019-07-08 14:06:38,907 WARN cluster.YarnScheduler: Initial job has not accepted any resources;
2019-07-08 14:06:47,677 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Registered execut
2019-07-08 14:06:48,266 INFO storage.BlockManagerMasterEndpoint: Registering block manager pi6:
2019-07-08 14:06:48,361 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, p
2019-07-08 14:06:48,371 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, p
2019-07-08 14:06:48,375 INFO scheduler.TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, p
2019-07-08 14:06:48,379 INFO scheduler.TaskSetManager: Starting task 3.0 in stage 0.0 (TID 3, p
2019-07-08 14:06:50,877 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on pi
2019-07-08 14:06:52,001 INFO scheduler.TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, p
2019-07-08 14:06:52,024 INFO scheduler.TaskSetManager: Starting task 5.0 in stage 0.0 (TID 5, p
2019-07-08 14:06:52,039 INFO scheduler.TaskSetManager: Starting task 6.0 in stage 0.0 (TID 6, p
2019-07-08 14:06:52,115 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) i
2019-07-08 14:06:52,143 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) i
2019-07-08 14:06:52,144 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) i
2019-07-08 14:06:52,156 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) i
2019-07-08 14:06:52,217 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) i
2019-07-08 14:06:52,249 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) i
2019-07-08 14:06:52,262 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 0.0 (TID 5) i
2019-07-08 14:06:52,270 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all c
2019-07-08 14:06:52,288 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38)
2019-07-08 14:06:52,323 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38
Pi is roughly 3.1389587699411
2019-07-08 14:06:52,419 INFO server.AbstractConnector: Stopped Spark@89f072{HTTP/1.1,[http/1.1]
2019-07-08 14:06:52,432 INFO ui.SparkUI: Stopped Spark web UI at http://pi1:4040
2019-07-08 14:06:52,473 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
2019-07-08 14:06:52,602 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
2019-07-08 14:06:52,605 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each execu
2019-07-08 14:06:52,640 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionSer
(serviceOption=None,
 services=List(),
 started=false)
2019-07-08 14:06:52,649 INFO cluster.YarnClientSchedulerBackend: Stopped
2019-07-08 14:06:52,692 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoi
2019-07-08 14:06:52,766 INFO memory.MemoryStore: MemoryStore cleared
2019-07-08 14:06:52,769 INFO storage.BlockManager: BlockManager stopped
2019-07-08 14:06:52,825 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2019-07-08 14:06:52,851 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:
2019-07-08 14:06:52,902 INFO spark.SparkContext: Successfully stopped SparkContext
2019-07-08 14:06:52,927 INFO util.ShutdownHookManager: Shutdown hook called
2019-07-08 14:06:52,935 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-1d1b5b79-6
2019-07-08 14:06:52,957 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-ca0b9022-2
```

If you sift through the output of the above command, you can see the the next result is:

```
took 60.246659 s
Pi is roughly 3.1389587699411
```

The very last thing I'd like to advise you to do is to add `stop-dfs.sh` and `stop-yarn.sh` to your `clusterreboot` and `clustershutdown` functions, if you've defined them. You should shut down the Hadoop cluster before powering off all of the machines, then reboot it when the machines boot back up:

```
function clusterreboot {
  stop-yarn.sh && stop-dfs.sh && \
  clustercmd sudo shutdown -r now
}

function clustershutdown {
  stop-yarn.sh && stop-dfs.sh && \
  clustercmd sudo shutdown now
}
```

[back to top]

Conclusion

So that's it! Hopefully this guide was helpful to you, whether you're setting up a Raspberry Pi, building your own Hadoop and Spark cluster, or you just want to learn about some Big Data technologies. Please let me know if you find any issues with the above guide; I'd be happy to amend it or make it clearer.

If you want more information about commands you can run on HDFS or how to submit a job to Spark, please click on those links. Thanks for making it to the end!

If you like content like this, be sure to follow me on Twitter and here on Dev.To. I post about Java, Hadoop and Spark, R, and more.

[back to top]

Hadoop & Spark (4 Part Series)

Discussion (38)

Rohit Das • Sep 23 '19

Hi. I am not able to get the hadoop version or java version for the nodes over ssh using clustercmd. I have set up a cluster of 5 Pis (model B+) with 32 GB micro SD cards in all of them. On running

```
clustercmd hadoop version | grep Hadoop
```

I get the following error:

```
bash:hadoop:command not found
bash:hadoop:command not found
bash:hadoop:command not found
bash:hadoop:command not found
Hadoop 3.2.0
```

I am attaching the .bashrc here. Please help. Thanks.

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "**" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
#[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
```

```
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
    # We have color support; assume it's compliant with Ecma-48
    # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
    # a case would tend to support setf rather than setaf.)
    color_prompt=yes
    else
    color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w \$\[
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
#alias ll='ls -l'
```

```
#alias la='ls -A'
#alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

# get hostname of other pis
function otherpis {
    grep "pi" /etc/hosts | awk '{print $2}' | grep -v $(hostname)
}

# send commands to other pis
function clustercmd {
    for pi in $(otherpis); do ssh $pi "$@"; done
    $@
}

# restart all pis
function clusterreboot {
    for pi in $(otherpis);do
        ssh $pi "sudo shutdown -r 0"
    done
}

# shutdown all pis
function clustershutdown {
    for pi in $(otherpis);do
        ssh $pi "sudo shutdown 0"
    done
}

# send files to all pis
function clusterscp {
    for pi in $(otherpis); do
        cat $1 | ssh $pi "sudo tee $1" > /dev/null 2>&1
    done
}

function clusterssh {
    for pi in $(otherpis); do
        ssh $pi $1
```

```
    done
}

# sudo htpdate -a -l time.nist.gov

export JAVA_HOME=/usr/local/jdk1.8.0/
export HADOOP_HOME=/opt/hadoop
export SPARK_HOME=/opt/spark
export PATH=/usr/local/jdk1.8.0/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SPARK_HOME/bin:$PATH
export HADOOP_HOME_WARN_SUPRESS=1
export HADOOP_ROOT_LOGGER="WARN,DRFA"
```

Andrew (he/him) • Sep 23 '19

Did you follow these steps?

Create the Directories

Create the required directories on all other Pis using:

```
$ clustercmd sudo mkdir -p /opt/hadoop_tmp/hdfs
$ clustercmd sudo chown pi:pi -R /opt/hadoop_tmp
$ clustercmd sudo mkdir -p /opt/hadoop
$ clustercmd sudo chown pi:pi /opt/hadoop
```

Copy the Configuration

Copy the files in /opt/hadoop to each other Pi using:

```
$ for pi in $(otherpis); do rsync -avxP $HADOOP_HOME $pi:/opt; done
```

This will take quite a long time, so go grab lunch.

When you're back, verify that the files copied correctly by querying the Hadoop version on each node with the following command:

```
$ clustercmd hadoop version | grep Hadoop
Hadoop 3.2.0
Hadoop 3.2.0
Hadoop 3.2.0
...
```

You can't run the `hadoop` command on the other Pis until you've copied over those `hadoop` directories. If you *have* done that, you also need to make sure that that directory is on the `$PATH`

of the other Pis by including the following lines in *each* of their `.bashrc` files (sorry, I don't think I included this step in the instructions):

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
export HADOOP_HOME=/opt/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

You could also simply `clusterscp` the `.bashrc` file from Pi #1 to each of the other Pis.

Rohit Das • Sep 23 '19

Hi. Thanks for the reply. Yes, I did the steps you mentioned. Since Java wasn't pre-installed, I installed it manually in each Pi, and checked them individually to see if they are working. As you can see below, the env variables are configured as you have suggested.

```
export JAVA_HOME=/usr/local/jdk1.8.0/
export HADOOP_HOME=/opt/hadoop
export SPARK_HOME=/opt/spark
export PATH=/usr/local/jdk1.8.0/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SPARK_HOME/bin:$PATH
export HADOOP_HOME_WARN_SUPRESS=1
export HADOOP_ROOT_LOGGER="WARN,DRFA"
```

Rohit Das • Sep 24 '19

Thanks. I resolved the issue by putting the PATH exports above the following part in .bashrc:

```
# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac
```

I also put the export PATH commands in /etc/profile of each Pi. Thanks.

Razvan T. Coloja • Nov 4 '19 • Edited on Nov 4

Try this for a more colourful clustercmd prompt put this in your ~/.bashrc:

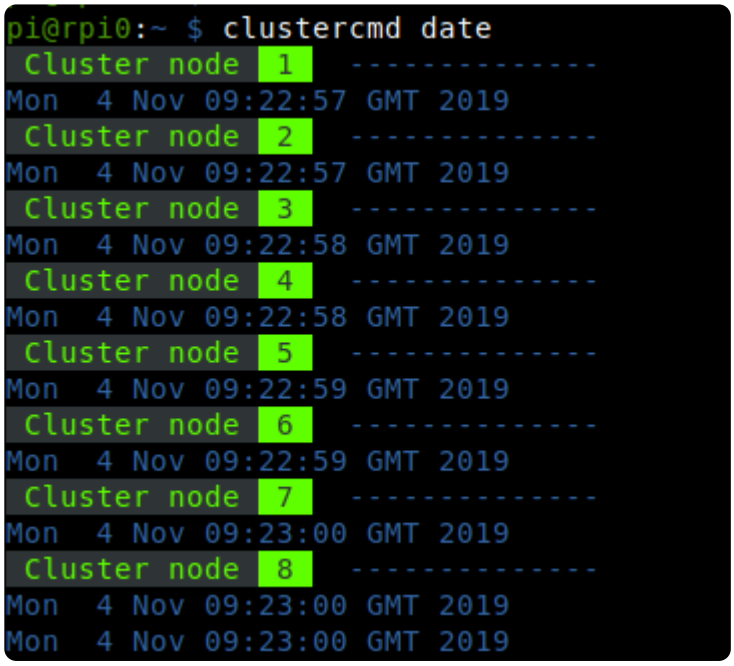
```
# clustercmd
function clustercmd {
```

```
        for I in 1 2 3 4 5 6 7 8; do echo -e "\e[40;38;5;82m Cluster node \e[30;48;5;82m $I \e[0m \
        $@
}
```

then do a

```
$ source ~/.bashrc
```

Looks like this:



Andrew (he/him) • Nov 4 '19

Looks nice! Thanks, Razvan!

PiDevi • Nov 9 '19

Hi Andrew, thanks for sharing this with us!

Following your instruction I have managed to run HDFS and YARN on my cluster using openJDK-8 instead of Oracle Java (is not pre-installed on Raspbian Buster and can't be installed via apt).

However, when running the wordcount example.

```
> yarn jar hadoop-mapreduce-examples-3.2.0.jar wordcount "/books/alice.txt" output
```

following error occurs

```
Error: Java heap space
Error: Java heap space
Error: Java heap space
```

and the job fails.

Do you have any idea what the reason might be?

Andreas Komninos • Aug 1 '19

Thank you for this superb article. I have been following it to deploy a Hadoop/Spark cluster using the latest Raspberry Pi 4 (4GB). I encountered one problem, which was that after completing the tutorial, the spark job was not being assigned. I got a warning:
INFO yarn.Client: Requesting a new application from cluster with 0 NodeManagers and then it sort of got stuck on
INFO yarn.Client: Application report for application_1564655698306_0001 (state: ACCEPTED). I will describe later how I solved this.

First, I want to note that the latest Raspbian version (Buster) does not include Oracle Java 8 which is required by Hadoop 3.2.0. There is no easy way to get it set-up, but it can be done. First you need to manually download the tar.gz file from Oracle's site (this requires a registration). I put it up on a personal webserver so it can be easily downloaded from the Pis. Then, on each Pi:

download java package

cd ~/Downloads
wget /jdk8.tar.gz

extract package contents

sudo mkdir /usr/java
cd /usr/java
sudo tar xf ~/Downloads/jdk8.tar.gz

update alternative configurations

sudo update-alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_221/bin/java 1000
sudo update-alternatives --install /usr/bin/javac javac /usr/java/jdk1.8.0_221/bin/javac 1000

select desired java version

sudo update-alternatives --config java

check the java version changes

java -version

Next, here is how I solved the YARN problem. In your tutorial section "Configuring Hadoop on the Cluster", after the modifications to the xml files have been made on Pi1, two files need to be copied across to the other Pis: these are yarn-site.xml and mapred-site.xml. After copying, YARN needs to be restarted on Pi1.

To set appropriate values for the memory settings, I found a useful tool which is described on this thread stackoverflow.com/questions/495791...

Copy-pasting the instructions:

get the tool

wget public-repo-1.hortonworks.com/HDP/...
tar zxvf hdp_manual_install_rpm_helper_files-2.6.0.3.8.tar.gz
rm hdp_manual_install_rpm_helper_files-2.6.0.3.8.tar.gz
mv hdp_manual_install_rpm_helper_files-2.6.0.3.8/ hdp_conf_files

run the tool

python hdp_conf_files/scripts/yarn-utils.py -c 4 -m 8 -d 1 false

-c number of cores you have for each node
-m amount of memory you have for each node (Giga)
-d number of disk you have for each node
-bool "True" if HBase is installed; "False" if not

This should provide appropriate settings to use. After the xml files have been edited and YARN has been restarted, you can try this command to check that all the worker nodes are active.

yarn node -list

PiDevi • Nov 9 '19 • Edited on Nov 9

Hi Andreas,

I am running Raspbian Buster on my PIs, too. I have downloaded the "Linux ARM 64 Hard Float ABI" (jdk-8u231-linux-arm64-vfp-hflt.tar.gz) and followed your instructions and I get following error bu running java -version

```
-bash: /usr/bin/java: cannot execute binary file: Exec format error
```

I guess this java-product is not compatible with the PI. Which exact file have you downloaded from the Orcale site?

Sliver88 • Jul 25 '20 • Edited on Jul 26

First of all i d like to thank Andrew for a superb tutorial. Besides some minor alternation i had to make, i was able to set up the hdfs etc. but i am running now on the same problem as you Andreas.
The first thing i d like to add to your recommendations is that downloading the java is easier.
sudo apt-get install openjdk-8-jdk
and then change the default (as you suggested already):

sudo update-alternatives --config java
sudo update-alternatives --config javac

Then change **export JAVA_HOME=$(readlink –f /usr/bin/java | sed "s:bin/java::")** to **export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-armhf** both in ~/.bashrc and in /opt/hadoop/etc/hadoop/hadoop-env.sh.

The part i have been stuck for a while though is that yarn node -list command stucks and if i try to run a spark job then i also get stuck on the ACCEPTED part.
I haven't yet tried your proposition.

PS I know it is a year-old article but still is the best i ve seen so far in my research.

Ευχαριστώ πολύ και τους 2 (I would like to thank you both)

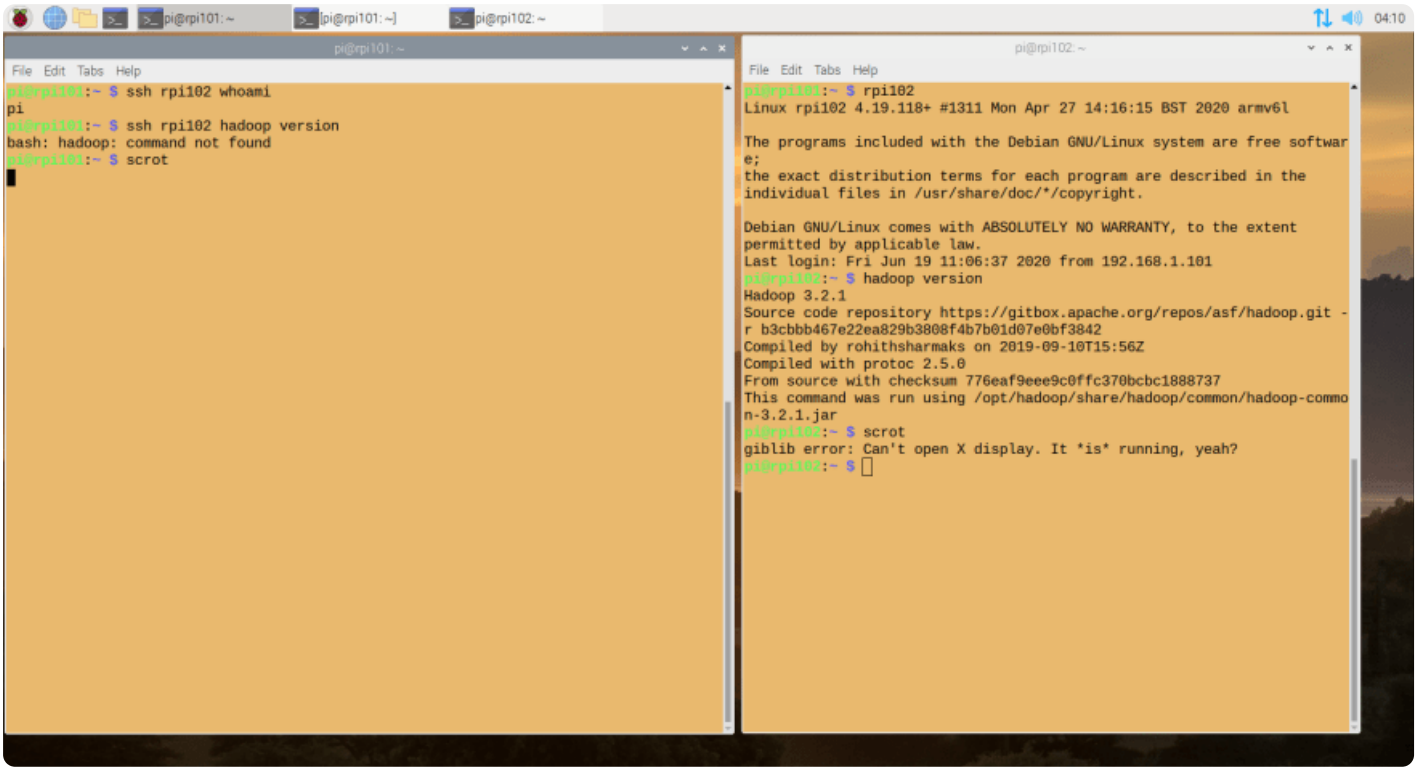
sshu2017 • Jun 19 '20 • Edited on Jun 19

Hi Andrew,
This is an excellent tutorial so THANK YOU very much!

I had a quite strange problem though related to the `clustercmd` function. At the moment, I have installed Java and Hadoop on both workers (which I call `rpi102` and `rpi103`). As you can see from the terminal on the right, I can SSH from the master (`rpi101`) into `rpi102` and run the `Hadoop version` command and got what I expected.

However, as you can see from the terminal on the left, when I did `ssh rpi102 hadoop version`, the hadoop command is not found. But if I try something else, like `ssh rpi102 whoami`, it worked fine.

This seems so odd and really puzzles me. Do you have any idea what the issue could be? Thanks in advance!



sshu2017 • Jun 19 '20

Never mind. This answer helped me out.
superuser.com/a/896454/955273

Yui Chun Leung • Aug 21 '21

Do you solve it by this command?

```
clustermod sudo -E env "PATH=$PATH" hadoop version | grep Hadoop
```

scurveedog • Jan 20 '20

Hello Andrew

Fantastic article/tutorial that made my deployment of a 4 node pi4 cluster almost trivial! But... I also think along the way, your use of bash shell scripting is brilliantly applied to the project. A model I will seek to emulate.

Jangbae • Jul 6 '20 • Edited on Jul 6

It was a great tutorial! I could successfully build a Hadoop system with four Raspberry Pi 4s. There were a few hiccups while I was following this tutorial but could be solved by following others postings. Thanks a lot!

Jangbae • Jul 8 '20

It was a bit early for me to declare my cluster is all set. The problem I'm facing is that the worker nodes are not showing up. I have a master node and three worker nodes in the cluster. "hdfs dfsadmin -report" only shows the master node. Double checked the XML files to make sure if there is any typo and they have been copied to the worker nodes correctly but found nothing. Do you have any suggestions to figure this out?

Scott Sims • Oct 4 '19

Hi Andrew,
First off, great stuff. Thank you so much for being as thorough as you are. This was a well put together #howto. Secondly, I'd like to highlight some issues I came across as following your post.

This is to be expected, but the links to the installations of hadoop and spark are no longer valid. But i just simply followed your link down the rabbit hole to find the most up to date available installer.

Next, I did run into the same issue as Rohit Das did as he noted below. I simply followed his fix and it worked perfectly. Thank you Rohit for figuring this out.

Thirdly, I ran into an issue of the namenodes and datanodes being registered, but the datanodes weren't able to connect to pi1. Looking in the UI, everything was 0 and 0 datanodes were registered. After chasing SO post after post, I finally got it to work by changing in core-site.xml

hdfs://pi1:9000
to
hdfs://{ip address}:9000

where {ip address} is the fully typed ip address of pi 1.

Lastly, I mounted 250GB SSD hard drives to my workers. Once I get the optimal resource limits worked out for this in the configuration xml's I'll be sure to post here what they were.

I should note my name node is a raspberry pi 3 but my workers are raspberry pi 4 with 4GB ram.

रमण मूर्ति • Jul 8 '19

Thanks for the article.

Andrew (he/him) • Jul 8 '19

Thanks for checking it out!

SunnyMo • Jul 16 '19

excellent article, I have a pi 3b+ and pi model b(the oldest 256m ver.), is it possible to run a spark cluster? just for study.

Andrew (he/him) • Jul 16 '19

My Pis are struggling as-is. I wouldn't recommend any lower specs than the Model 3 B+ (1 GB RAM)

SunnyMo • Jul 17 '19

I do want to make use of the old pi, even if it is used as a NameNode, I think it doesn't need much computation resource. I'm new to spark, so the question might be silly, sorry about that.

Andrew (he/him) • Jul 17 '19 • Edited on Jul 17

Even though NameNodes aren't processing data, they still have some CPU and memory requirements (they have to orchestrate the data processing, maintain records of the filesystem, etc.). I saw somewhere that 4GB per node was the recommended minimum. All I know from experience is that 1GB seems to barely work.

Spark sets minimum memory limits and I don't think 256MB is enough to do anything.

SunnyMo • Jul 19 '19

okay, the only thing that 256m can do may be running an Nginx reverse proxy in my private cloud or RIP, thanks for that.

Andrew (he/him) • Jul 19 '19

Maybe you could turn it into a Pi-Hole?

SunnyMo • Jul 25 '19

unfortunately, Pi-Hole project requires at least 512m memory. My old pi should R.I.P right now, I'll leave it to my children as the first gift form elder generation.

Wessel Valkenburg • Jan 23 '20

Thanks for this excellent guide! Haven't tried it yet, but looks remarkably complete and pedagogical.

Some questions:

- regarding cost: you give an awesome hardware summary. How much time did you invest? As always, probably the human resource is the most expensive part (given the price of your level of expertise on the free market).
- have you measured the performance of this cluster, and can you compare it to realistic real-world scenarios with x86 machines?
- have you installed / tried to install Apache Impala on the cluster? Not sure that's "physically" possible, but I would guess you are the authority who can give a final answer. I'm asking, because I'm under the impression that Impala is by far the fastest SQL solution for Hadoop.
- Further on SQL: how do queries on your cluster compare in performance to queries on a single big machine with a plain Postgress database (or MariaDB, or whatever)?
- Finally, do you think this setup fills an empty spot in computing space for the commercial market? I'm thinking of small companies who want a better data cluster but cannot afford full-fledged big-data solutions.

Andrea Luciano Damico • Jul 9 '19

Amazing! Now I kind of want to make one, although I'm not sure what I could do with such a cluster.

Andrew (he/him) • Jul 9 '19

Impress your friends!

Crush your enemies!

ADARI GIRISH KUMAR • Jul 8 '19

Excellent article ! You have covered the whole process in very detailed manner . Thanks.

Connor Luckett • Jul 9 '19

Hi, it looks like you've just tried this with computational tasks (calculating pi). We're trying this with Spark SQL and facing out-of-memory errors on our 3B+ cluster. Have you tried this on a memory-intensive job?

Andrew (he/him) • Jul 9 '19

Yes, we were doing some benchmarking on another machine, running random forests with X number of trees. I tried to run the same script on the Pi cluster and got `OutOfMemoryError`s with miniscule datasets.

Davor Vuković • Mar 17 '20

Thank you very much for this article! I am definitely going to recreate this at home.

Mihai Tanasescu • Feb 25 '21

Hi there,
Very cool page you have with Hadoop :). I'm building something similar with 8 x PI4Bs and want to start playing with K8S in fact.
One question though...the PIs with the official/original PoE hat..does the fan of the HAT still have space between it and the next slot/cardboard with the next PI to do its job? I'm more afraid of that.
I was looking at loverpi poe hat protected as an alternative for having more breathing space as I was worried the original one might end up not having space for air circulation.
Can you please let me know?
It's the only piece of the puzzle I still have to solve:)

Andrew (he/him) • Feb 26 '21

In this setup, yes, there is enough space (a few mm above the fan to bring in some air). Although if you get a different case / rack, I don't know what the dimensions would be.

The limiting factor for me wasn't really airflow, but RAM. 1GB RAM per node doesn't seem to be enough to run a H/S cluster, unfortunately. I would say 4GB minimum is probably the way to go.

Mihai Tanasescu • Feb 26 '21

I have the cloudlet case as well but Raspberry PI4 8Gb :) Was more worried that the fan on the hat would not have any real air to make up for a true airflow given the space.
Was now also looking at a minimal PoE hat from loveRPI but there I also read reviews about some people having burned their PIs :)
Having bought 8 PI4s would not like to create a camp fire out of them :)

Andrew (he/him) • Feb 27 '21

I don't know how how the 8GB ones would run (pretty hot, I imagine). The case is a little snug, as you know. If I were doing anything serious with that 8GB, I definitely wouldn't shove 8 Pis in that case. Maybe 4? And leave every other slot open? You'll probably want additional fans either way. Good luck!

Mihai Tanasescu • Feb 27 '21

:) let's see how my experiment goes.
The loveRPI PoE hat is quite compact and I will add some radiators on the components of the PI.
Then the fans of the case should have enough space for the air flow.
The original PoE hat is too size limiting and I think you're right with the air flow regarding it.

View full discussion (38 comments)

Code of Conduct • Report abuse

Andrew (he/him)

Got a Ph.D. looking for dark matter, but not finding any. Now I code full-time. Je parle un peu français.

LOCATION
Ottawa, Canada

EDUCATION
Ph.D. in [Astroparticle] Physics

WORK
Senior Software Developer at YoppWorks

JOINED
Sep 15, 2018

More from Andrew (he/him)

Elegant Multi-Line Shell Strings
#shell #bash #zsh #showdev

What's the Difference Between Nominal, Structural, and Duck Typing?
#java #typescript #javascript #computerscience

Eliminate Unnecessary Builds With Git Hooks in Bash, Java, and Scala
#bash #java #scala #git