Creating a Docker Swarm with Fabric and Raspberry Pi

Ryan L. Irey, M.A.
Indiana University
107 S Indiana Ave
Bloomington, Indiana 47405
rlirey@iu.edu

ABSTRACT

Docker's swarm mode allows for the use of their automated container distribution technology across a cluster of computers on a shared subnet. Each node of the swarm takes on the role of either a manger or a worker. One may foresee that the initial setup and management of each node could be a cumbersome experience, especially for swarms with many nodes. To mitigate this problem, a Python package called Fabric can be used to execute shell commands over ssh.

KEYWORDS

Cluster computing, Docker, Swarm, Big Data, i523, HID-318

1 INTRODUCTION

Clusters of computers offer many useful applications in the capturing and analysis of Big Data. This manuscript serves as a manual for creating a cluster using Raspberry Pi machines. This cluster model will operate on the Swarm platform offered by Docker, Inc., and will utilize the a Python package called Fabric to communicate with the machines in the Swarm. The manual will culminate with a demonstration of the cluster's Swarm capabilities by deploying a voting application.

Assumptions of this manual include the use of a Linux-based host machine, multiple Raspberry Pi modules, and a multi-port network switch. Here, we use four Raspberry Pi v3 model B+ are used along with a 5-port gigabit ethernet network switch. Static IP addresses will be used with each Raspberry Pi; the prefix 192.168.7.XXX is used here, although the reader will need to use the appropriate IP prefix for their network. A visual representation of the system architecture is given in Figure 1.

[Figure 1 about here.]

2 HARDWARE SETUP

2.1 Host Machine Setup: Part I

The host machine can be a laptop or desktop consumer grade computer. For the purposes of this manual, it is assumed that the computer is running a Linux environment and has common command line setup tools installed such as bash, nano, ssh, etc. Additionally, it is assumed that the user can access root privileges using the sudo command. It is not necessary for the host machine to have a static IP address. However, each of the Raspberry Pi nodes will have a static IP to ensure reproducible connections between system reboots. On the host system, access the hosts file in the root /etc folder. Once inside this file, add the information presented in Figure 2. This will eventually play a role in allowing ssh logins to the Raspberry Pi nodes from the host machine.

```
\$ cd /etc/
\$ sudo nano /etc/hosts
```

[Figure 2 about here.]

2.2 Raspberry Pis

Each Raspberry Pi requires a 5v power supply provided via either GPIO pins or micro USB. Here, a powerful module called the Bitscope Blade is used to distribute power to the Raspberry Pi units via GPIO pins and a single 12v power supply. The Bitscope Blade has many powerful capabilities for specialized applications, but here, its sole purpose is to power all of the Raspberry Pi units, as shown in Figure 3.

[Figure 3 about here.]

2.3 MicroSD cards

The system requirements of the Raspberry Pi system specify that microSD cards used for the Raspberry Pi must be class 10 and no larger than 32 GB in size.

3 RASPBERRY PI INITIALIZATION

The steps to initialize the Raspberry Pi modules are outlined as follows:

- Preparing the MicroSD cards
- SSH preparation
- Raspberry Pi configuration
- SSH configuration

Additionally, static IP addresses are used in this tutorial to ensure the system can be rebooted without losing connection to each node. Refer to Figure 1 for details on static IP assignments in this manual.

3.1 Preparing the MicroSD cards

Each microSD card must be flashed using the most recent version of the Raspbian operating system [3]. The disk image for the operating system can be found at https://www.raspberrypi.org/downloads/raspbian/. At the time of this writing, the current version is named 'Stretch'. This manual assumes utilization of the 'Lite' version of the operating system.

3.2 SSH preparation

The next set of instructions must be repeated for each of the newly-flashed microSD cards. With the microSD card mounted to the host machine, note the presence of two partitions, one named "boot"

and the other named "rootfs". There are two modifications that must be made for the purposes of this demonstration. First, a file called ssh will be created on the boot partition. Using the command line:

```
# Navigate to the boot partition using the cd command
\$ cd /media/yourusername/boot
\$ touch ssh
```

Second, navigate to the /etc folder of the rootfs partition. Open the dhcpcd.conf file for editing using nano. Navigate towards the bottom of the page and find a section with the header `# Example static IP configuration`. For each microSD card, edit the text in this section to correspond with the static IP information assigned to each node. Figure 4 gives an example.

```
# Navigate to the /etc folder in the rootfs partition using
    cd
\$ cd /media/yourusername/rootfs/etc/
\$ sudo nano dhcpcd.conf
# Enter password for sudo privileges
```

[Figure 4 about here.]

4 HOST MACHINE SETUP: PART II

5 USING FABRIC

Fabric is a Python package that can be used to send shell commands over ssh to connected network devices like the machines comprising the Raspberry Pi cluster [2]. To avoid the redundancy of configuring our Raspberry Pi nodes one at a time, Fabric is used here to issue shell commands to all of the nodes at the same time.

5.1 Fabric Installation

Fabric can be installed using `pip` in the following manner:

```
pip install fabric
```

It is also necessary to install a package called Paramiko.

```
pip install paramiko
```

5.2 Fabric Object Initialization

Now, each raspberry pi can be initialized as a `fabric` object, as can the cluster. From within Python:

```
from fabric import *
import paramiko

pi1 = Connection(host = 'pi1')
pi2 = Connection(host = 'pi2')
pi3 = Connection(host = 'pi3')
pi4 = Connection(host = 'pi4')

cluster = SerialGroup('pi1','pi2','pi3','pi4')
```

With the `fabric` connections in place for each Raspberry Pi as well as the cluster, it is now possible to send shell commands to the machines using the run method. Figure 5 gives an example using a simple 'Hello World!' command:

[Figure 5 about here.]

It is important to note, however, that there is nothing yet in place to make the Raspberry Pi nodes behave as a cluster. At this point, we are merely sending identical shell commands to each node. To convert these independent nodes into a cluster, a platform like Docker's swarm mode is needed.

6 DOCKER INSTALLATION WITH FABRIC

Each of the nodes in the Raspberry Pi cluster must have the Docker engine installed before they can be used in a swarm configuration [1]. The Docker engine can be installed simply across all nodes with an installation function in Python and a single `fabric` command:

6.1 Installing Docker and Docker Compose

```
# A function for installing Docker and Docker Compose
def installDocker(c):
    c.run('sudo curl -sSL https://get.docker.com/ | sh')
    c.run('sudo apt-get install docker-compose -y')
    c.run('sudo usermod -aG docker pi')

# Apply the installation function to each node
installDocker(cluster)
```

6.2 Swarm Initialization

```
# Initiate Docker Swarm
pi1.run('sudo docker swarm init --advertise-addr
     192.168.7.201')

# Add worker nodes
# sudo + the command generated by initialization

workers = SerialGroup('pi2','pi3','pi4')

# To add a worker to this swarm, run the following command:
docker swarm join-token worker

workers.run('docker swarm join --token {token}
     192.168.7.201:2377')
```

7 DEMONSTRATION: SWARM VISUALIZATION

With the swarm now up and running, a simple visualization tool can be downloaded to gain a better understanding of the swarm's underlying composition. Using `fabric`, the visualization tool will run via a Docker container on the manager node.

```
pi1.run('docker run -it -d -p 8080:8080 -v
     /var/run/docker.sock:/var/run/docker.sock
     dockersamples/visualizer')
```

This command will pull the Docker image for the container and run the container. The visualizer itself should now be broadcasting on port 8080 and visible in a web browser by navigating to pi@pi1:8080. A sample visualization is given in Figure 6

[Figure 6 about here.]

8 DEMONSTRATION: VOTING APPLICATION

To showcase the capabilities behind a Docker swarm, Docker has provided a voting application that utilizes five containers in a stack configuration [4]. Figure 7 demonstrates how the different containers of the voting application interact. The front-end user interface is written in Python, and collects the vote input, sending the data to a Redis in-memory key-value store. The worker container, written in .NET, processes the key-value store data and translates this information into a PostgreSQL database for persistence. Finally, another front-end user interface, written in Node.js, displays the voting tallies as they come in through the system.

[Figure 7 about here.]

When deploying this software stack, the Docker Compose file includes the same visualization tool demonstrated above. Thus, the allocation and management of the different containers comprising the voting application can be observed.

```
# Download the voting application's Docker Compose file via
    curl command
pi1.run('curl -O https://raw.githubusercontent.com/docker/ \
example-voting-app/master/docker-stack.yml')
# Deploy the application on the manager node
pi1.run('docker stack deploy -c docker-stack.yml vote')
```

With the stack running on the Docker Swarm cluster, we can view the front end interfaces at root@pi1:5000 (voting-app) and root@pi1:5001 (result-app), respectively. The swarm visualization tool remains visible on port 8080 (root@pi1:8080). Figure 8 shows the distribution of containers among the nodes in the swarm. One may notice that the voting-app container is replicated twice, as a simple demonstration of redundancy within the swarm.

[Figure 8 about here.]

9 CONCLUSION

Docker's swarm platform offers a powerful infrastructure for deploying software stacks with automatic load balancing. Using the methods outlined in this manuscript, one can easily set up a cluster using Raspberry Pi machines to deploy the swarm platform. With a Docker swarm in place, one can easily create an environment for creating, testing, and shipping their container or software stack.

ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski and all teaching assistants of INFO-I523 for their thoughtful comments and guidance on previous versions of this manuscript.

REFERENCES

[1] Inc. Docker. 2018. *Create a Swarm.* https://docs.docker.com/engine/swarm/
[2] Jeff Forcier. 2018. Fabric Release 2.3. (May 2018). https://www.fabfile.org/index.html
[3] Raspberry Pi Foundation. 2018. *Installing Operating System Images.* https://www.raspberrypi.org/documentation/installation/installing-images/
[4] Luc Juggery. 2017. Deploy the Voting App on a Docker Swarm using Compose version 3. (Feb. 2017). https://medium.com/lucjuggery/deploy-the-voting-apps-stack-on-a-docker-swarm-4390fd5eee4

We include an appendix with common issues that we see when students submit papers. One particular important issue is not to use the underscore in bibtex labels. Sharelatex allows this, but the proceedings script we have does not allow this.

When you submit the paper you need to address each of the items in the issues.tex file and verify that you have done them. Please do this only at the end once you have finished writing the paper. To d this cange TODO with DONE. However if you check something on with DONE, but we find you actually have not executed it correcty, you will receive point deductions. Thus it is important to do this correctly and not just 5 minutes before the deadline. It is better to do a late submission than doing the check in haste.

A ISSUES

DONE:

Example of done item: Once you fix an item, change TODO to DONE

A.1 Assignment Submission Issues

Do not make changes to your paper during grading, when your repository should be frozen.

A.2 Uncaught Bibliography Errors

Missing bibliography file generated by JabRef

A.3 Formatting

Incorrect number of keywords or HID and i523 not included in the keywords

A.4 Writing Errors

Errors in title, e.g. capitalization

A.5 Citation Issues and Plagiarism

It is your responsibility to make sure no plagiarism occurs. The instructions and resources were given in the class

A.6 Character Errors

Erroneous use of quotation marks, i.e. use “quotes” , instead of ” ”

A.7 Structural Issues

Acknowledgement section missing

A.8 Details about the Figures and Tables

Capitalization errors in referring to captions, e.g. Figure 1, Table 2

List of Figures

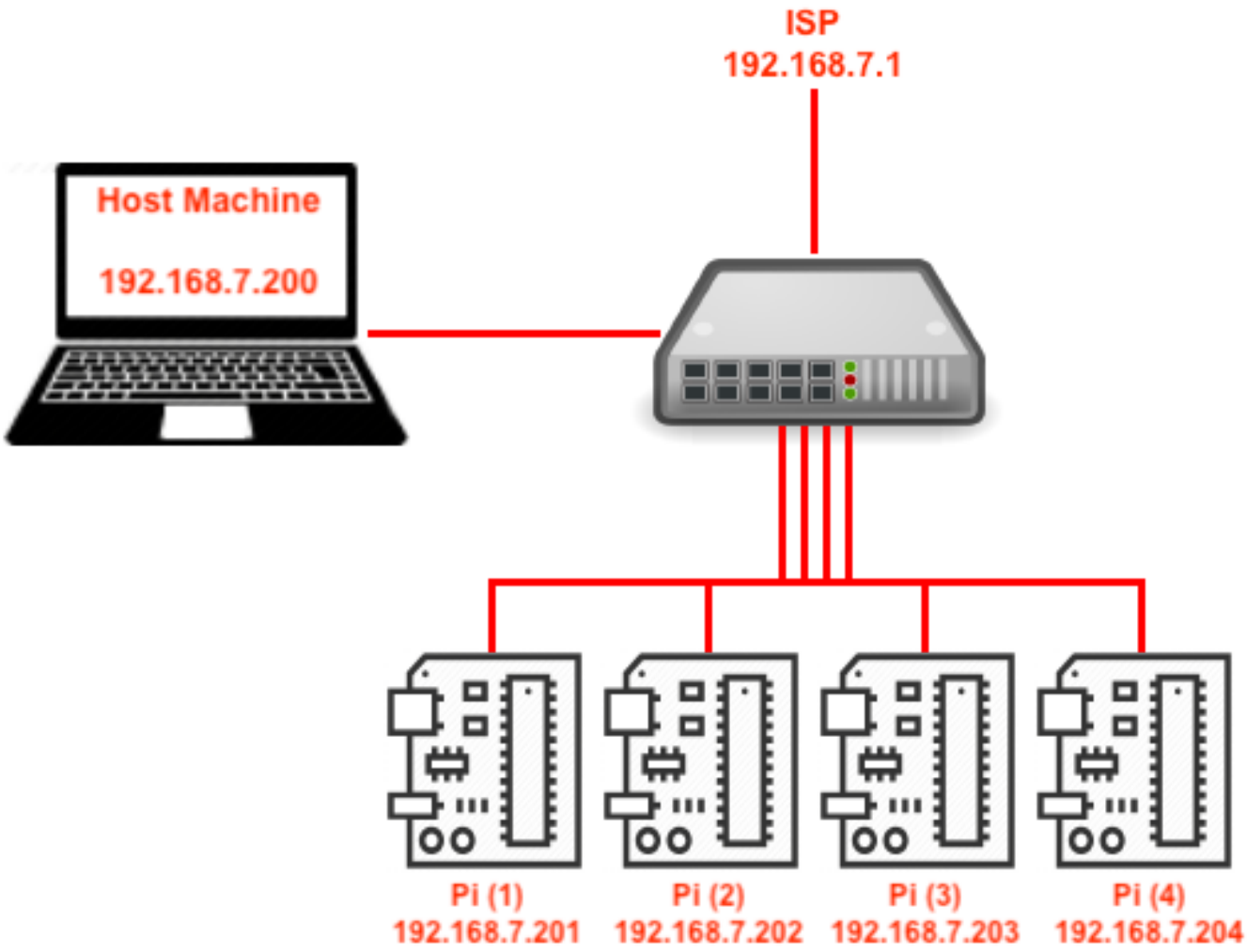

Figure 1: System configuration used in this manual

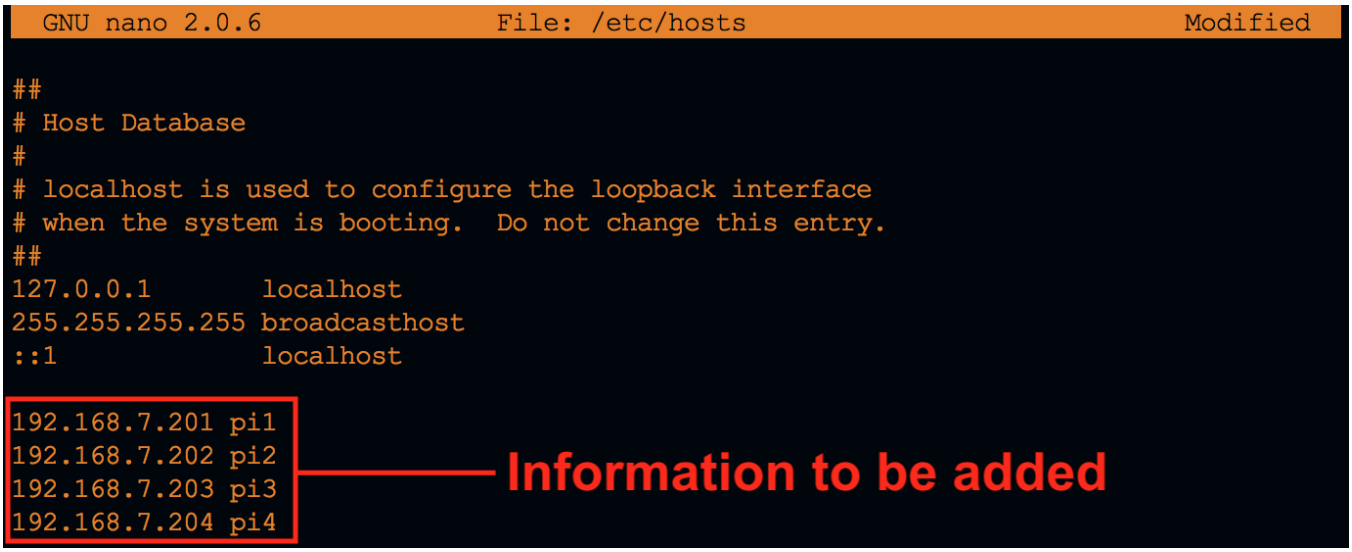

Figure 2: IP address and hostname information to be added to the file /etc/hosts/.

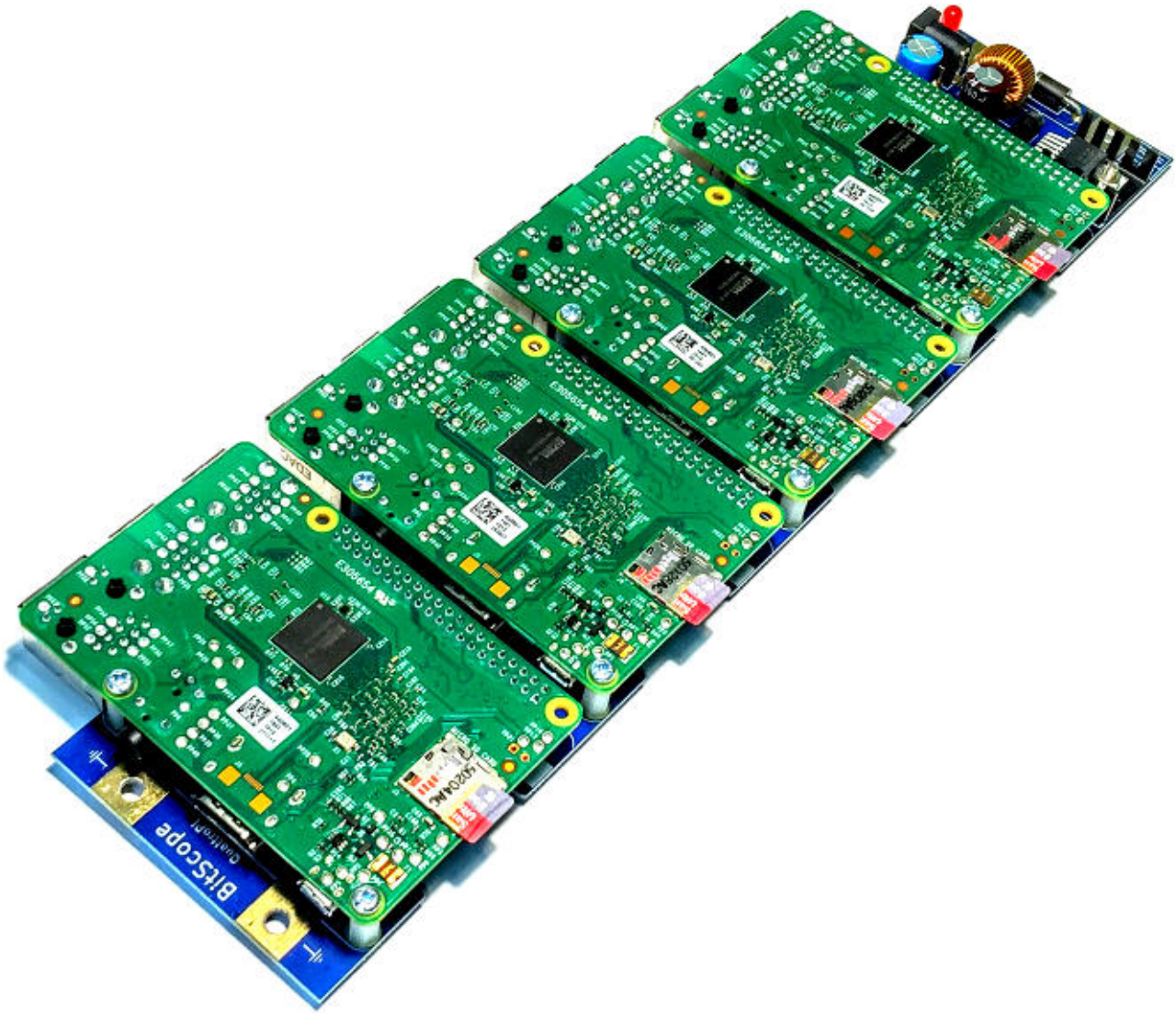

Figure 3: Raspberry Pi modules mounted to a Bitscope Blade Quattro module.

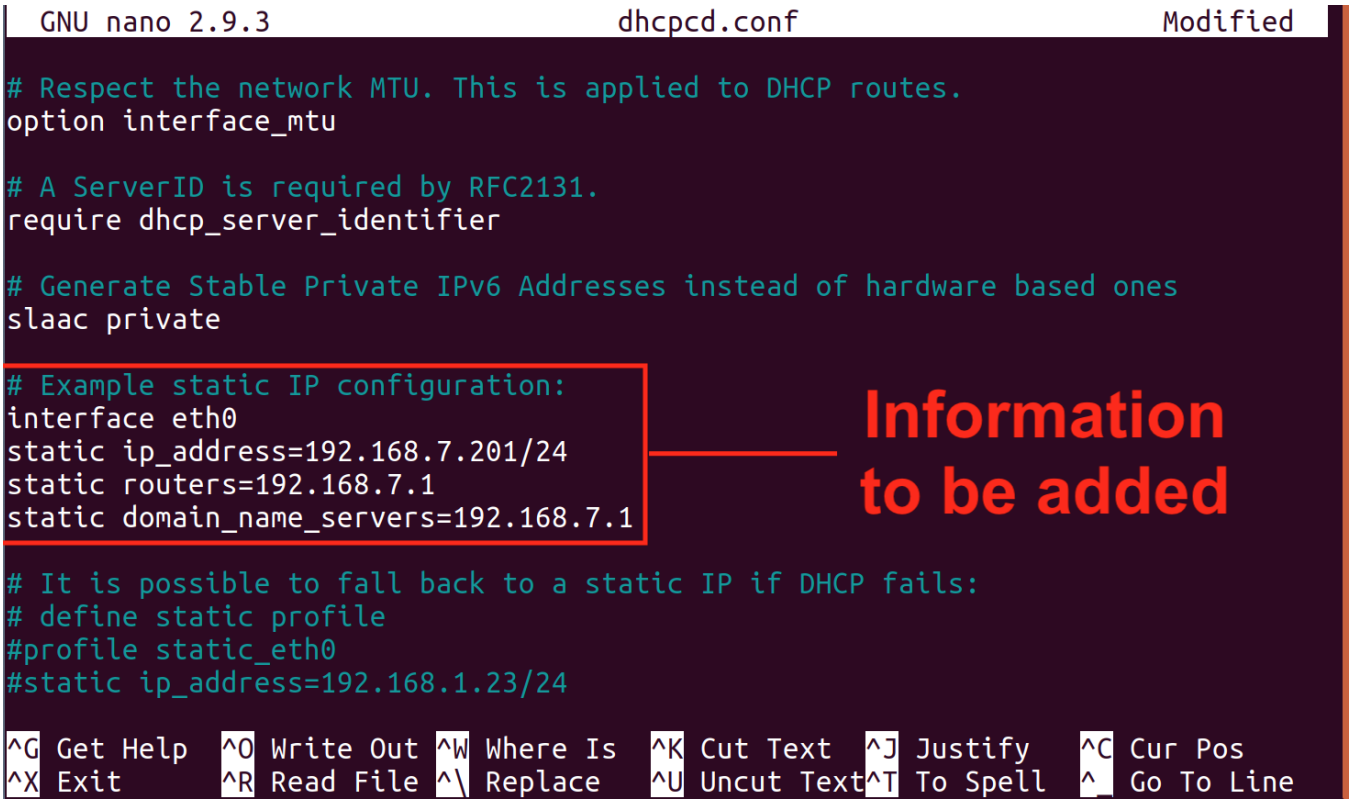

Figure 4: Configuration of the dhcpcd.conf file on the rootfs partition.

```
In [4]: cluster = SerialGroup('pi1','pi2')

In [5]: pi1 = Connection(host = 'pi1')

In [6]: pi2 = Connection(host = 'pi2')

In [7]: cluster.run('echo Hello World!')
Hello World!
Hello World!
Out[7]:
{<Connection host=pi1 user=pi>: <Result cmd='echo Hello World!' exited=0>,
 <Connection host=pi2 user=pi>: <Result cmd='echo Hello World!' exited=0>}
```

Figure 5: A simple 'Hello World!' test of the Fabric package.

Figure 6: A sample view of the Docker Swarm visualization tool.

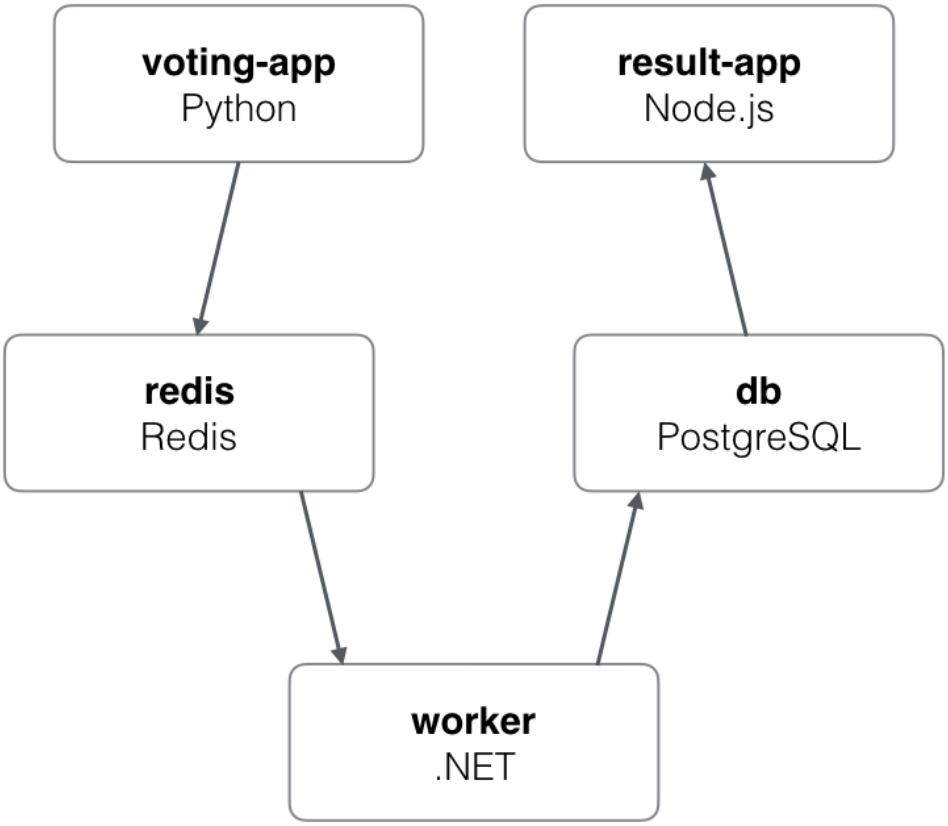

Figure 7: A schematic of the stack containers of Docker's voting application.

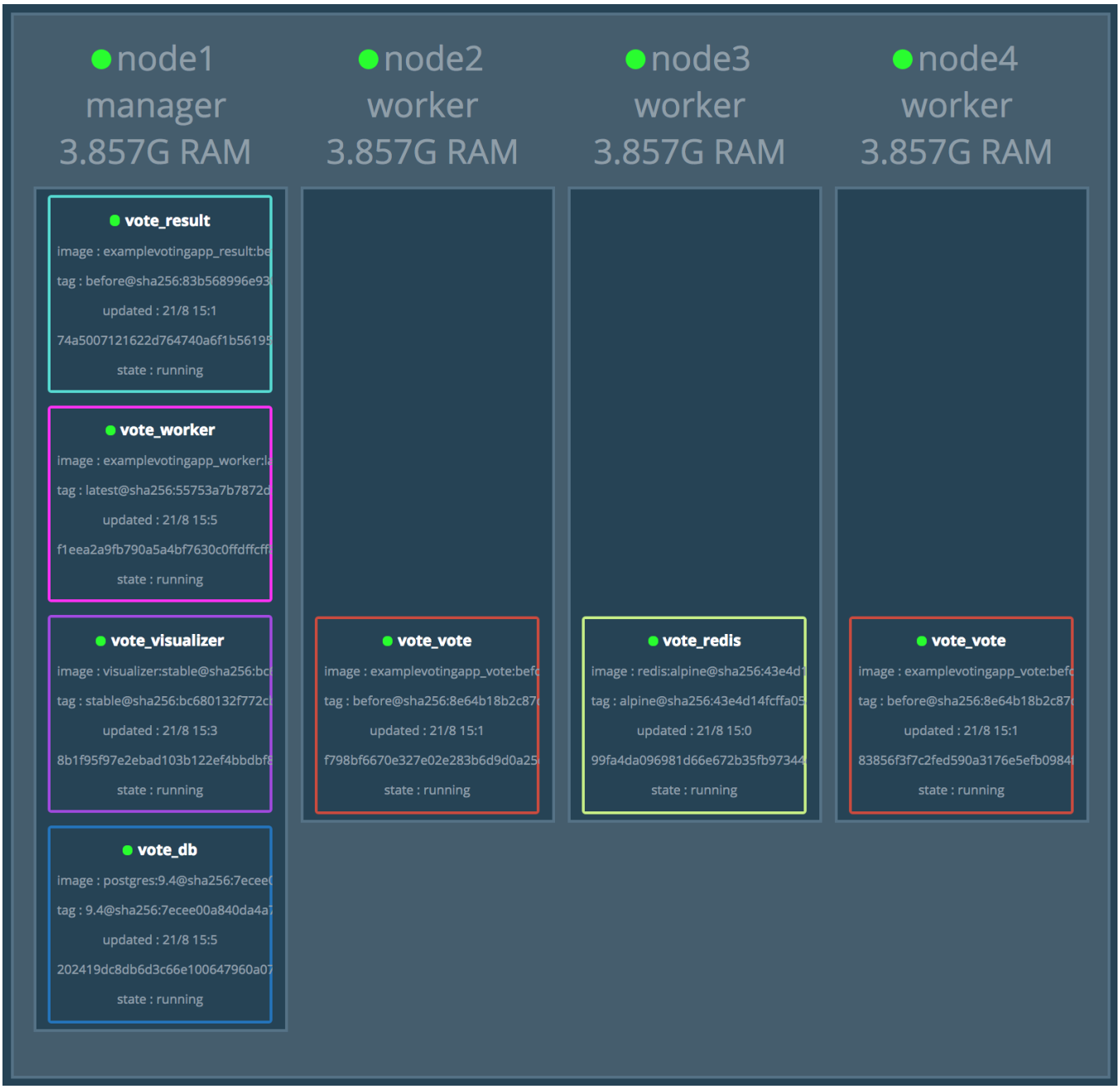

Figure 8: A sample view of the Docker voting application via the visualization tool.