



# **Final Report: Raspberry Pi Server Cluster**

## **Embedded Linux Spring 2016**

### **Instructor: Professor Easwaran**

## **Description**

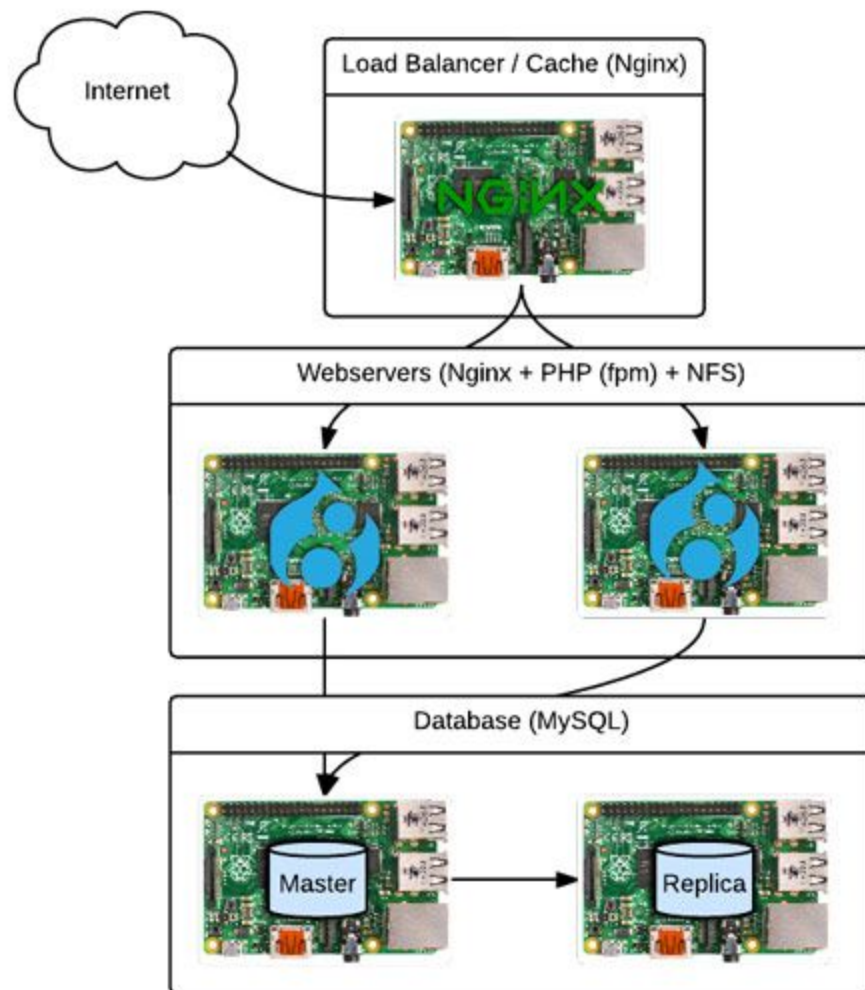
Ansible is a free software platform used to manage multiple computers via SSH. While Ansible is used to remotely control networked devices, the software itself is installed solely onto a controlling machine which has the correct credentials to interface with the remote devices in question.

This allows the user to use Ansible to remotely control a number of devices connected to a single network. Commands in Ansible can be automated using YAML, a data oriented mark-up language. YAML is used in Ansible “playbooks” to coordinate commands to be given to the array of linked devices. In this way ansible can be used to remotely interface with a number of devices simultaneously.

Ansible can be used to manage a number of devices simultaneously for monitoring, updating, configuration, and countless other applications. By using playbooks Ansible can also be used to seamlessly configure a networked system. Our task in this project was to deploy a distributed system involving a load balancer, web server, and distributed database to deploy a drupal web application.

## Goals

Our overall goal in this project is to build a raspberry pi cluster server controlled by Ansible. We will be installing Ansible on a control machine which will be used to connect to a centralized network of Raspberry Pis. The network of pis includes a load balancer which distributes traffic across the system. The load balancer directs traffic to one of two web server pis which implement a lightweight nginx webserver. The nginx servers are to be used to host a drupal website relying upon two database pis hosting mysql databases. This system working in congress allows for much more powerful hosting performance than one would be capable of using a single pi to handle all tasks.



**Figure 1:** Diagram detailing the structure of the pi cluster server (source: [pidramble.com](http://pidramble.com))

Furthermore, Ansible can be used to run scripts on the network of pis simultaneously. While accomplishing distributed processing across the pis is outside of the scope of this project, we will be using ansible to run scripts on all of the pis simultaneously. By issuing tasks to all pis simultaneously we can benchmark the performance on the respective pis. Scripts running across our distributed system will include controlling LEDs on the pis using direct scripts as well as cron jobs, and performing simple number sorting algorithms in order to measure performance power for the pis.

## Project Components

### Materials:

1. 4x Raspberry Pi 2 Model B
  - 900 MHz quad-core ARM Cortex-A7 CPU
  - 1 GB RAM
2. 2x Raspberry Pi 2 Model A
  - 700 MHz ARM11 ARM1176JZF-S core
  - 512 MB RAM
3. Network Switch
4. Control Computer
5. Ethernet cables

### Key Components:

1. Install and configure Ansible on the control PC.
2. Use YAML playbook to install and configure load balancer, nginx web server, and hosting databases for drupal deployment
3. Deploy Drupal website to server cluster, create web application.
4. Perform server benchmarking
5. Use Ansible to test performance of pis with distributed tasks, implement basic cron jobs using ansible and test performance of scripts.

## Project Summary

### Good

- Ansible is a user friendly software environment with good online documentation
- Drupal seems to be a strong web development platform
- Project puts into practice applicable skills and familiarizes student with a number of fundamental networking practices within the linux environment.

### Bad

- Networking the pis, specifically on the campus network can be very difficult.
- Ansible documentation with respect to this specific task is limited.
- Steep learning curve with a lot of different syntaxes and protocols to learn (SSH, YAML, Ansible, Drupal, DHCP, etc.)
- Working with PHP, nginx, mysql, python, and associated libraries, dependencies, and versions can become hairy. Especially considering the fact that many examples sourced for this project featured legacy code inhibiting smooth completion of project goals.

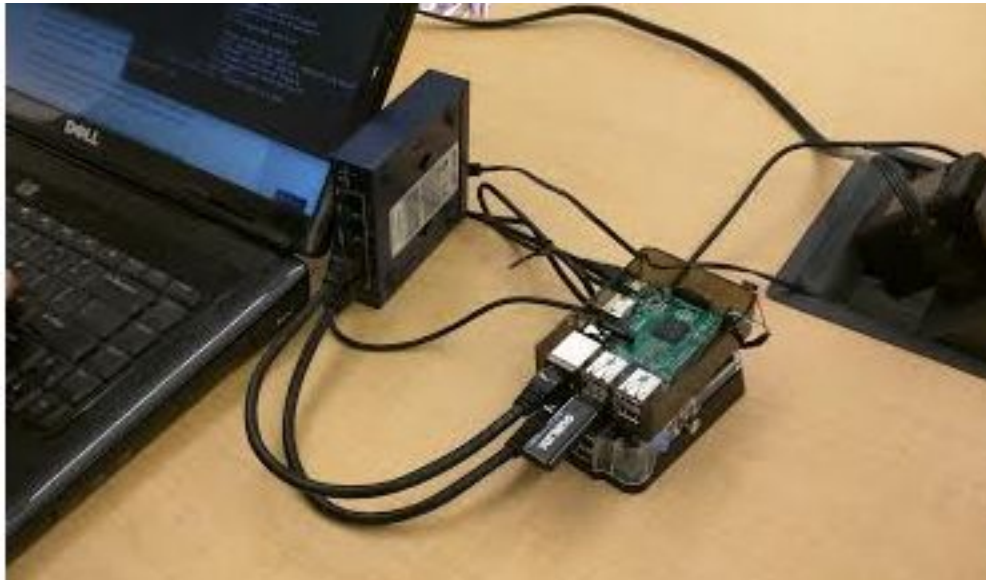
## Plan

### Week 1

```
$ ansible all -i inventory -m ping -k
SSH password:
24.168.39.215 | success => {
  "changed": false,
  "ping": "pong"
}
```

Get Ansible service working to interact with two pis simultaneously. Once this is achieved that adding devices to and configuring the server will be easy. Currently default hardware settings on the pi are making interacting via wifi and ethernet difficult. Thus far we have been able to use ansible to establish communications with a single pi

off-campus. Moving forward we will need to establish communications with multiple pis using a single RSA public key.



## Week 2

At this point we are still having trouble with some of the YAML configuration for the pis. In configuring the hosts we have been overwriting the routing tables in the pis which is inhibiting internet access which is vital for installation of software utilities and dependencies for each of the server components. We have begun issuing simple commands to each of the pis using ansible.

```
christmas@christasmute ~/Spring2016/embeddedLinux/dramble/raspberry-pi-dramble
$ ansible all -i inventory -a "free -m" -k
SSH password:
192.168.0.43 | SUCCESS | rc=0 >>
      total        used        free      shared  buffers   cached
Mem: 973          94          878         6         13        43
-/+ buffers/cache: 38          935
Swap: 99           0           99

192.168.0.44 | SUCCESS | rc=0 >>
      total        used        free      shared  buffers   cached
Mem: 973          94          879         6         12        43
-/+ buffers/cache: 38          935
Swap: 99           0           99

192.168.0.50 | SUCCESS | rc=0 >>
      total        used        free      shared  buffers   cached
Mem: 482          85          396         4         13        40
-/+ buffers/cache: 32          450
Swap: 99           0           99
```

**Figure 2:** Demonstrating Ansible's capabilities in issuing shell commands to multiple devices simultaneously. Here Ansible is pulling up memory usage information from three raspberry pis.

Our task was further complicated by the pi network being compromised resulting in the default user (pi, raspberry) being blocked by a malicious actor. We had been using port forwarding from a residential router to the pi network in order to work on the system from campus. Because we were using plaintext default passwords it was easy for a bad actor to get access to the pi network by simply infiltrating through port 22 on the residential router. This has necessitated a re-installation of raspbian and a move away from port forwarding in order to avoid any further complications.

### **Week 3**

We have re-installed Raspbian and implemented passwordless ssh access into pies using public key encryption. This task is easy and allows for easy access of all the pis using ansible commands without having to continually type the plaintext password in the shell. This also heightens security for the distributed system as access to the system can only be made from the control machine containing the associated RSA private key.

```
< TASK [Set up networking-related files.] >
-----
      ^
      (oo)\
      ( )\-----w\
      ||-----w  |
      ||          ||

changed: [192.168.0.43] => (item={u'dest': u'/etc/hostname', u'template': u'hostname.j2'})
changed: [192.168.0.44] => (item={u'dest': u'/etc/hostname', u'template': u'hostname.j2'})
changed: [192.168.0.43] => (item={u'dest': u'/etc/hosts', u'template': u'hosts.j2'})
changed: [192.168.0.44] => (item={u'dest': u'/etc/hosts', u'template': u'hosts.j2'})
changed: [192.168.0.50] => (item={u'dest': u'/etc/hostname', u'template': u'hostname.j2'})
changed: [192.168.0.43] => (item={u'dest': u'/etc/network/interfaces', u'template': u'interfaces.j2'})
changed: [192.168.0.44] => (item={u'dest': u'/etc/network/interfaces', u'template': u'interfaces.j2'})
changed: [192.168.0.44] => (item={u'dest': u'/etc/resolv.conf', u'template': u'resolv.conf.j2'})
changed: [192.168.0.43] => (item={u'dest': u'/etc/resolv.conf', u'template': u'resolv.conf.j2'})
changed: [192.168.0.50] => (item={u'dest': u'/etc/hosts', u'template': u'hosts.j2'})
changed: [192.168.0.43] => (item={u'dest': u'/etc/dhcpd.conf', u'template': u'dhcpd.conf.j2'})
changed: [192.168.0.44] => (item={u'dest': u'/etc/dhcpd.conf', u'template': u'dhcpd.conf.j2'})
changed: [192.168.0.50] => (item={u'dest': u'/etc/network/interfaces', u'template': u'interfaces.j2'})
changed: [192.168.0.50] => (item={u'dest': u'/etc/resolv.conf', u'template': u'resolv.conf.j2'})
changed: [192.168.0.50] => (item={u'dest': u'/etc/dhcpd.conf', u'template': u'dhcpd.conf.j2'})
```

**Figure 3:** Ansible output for task of changing hostname, host, and network interfaces information on the pi cluster

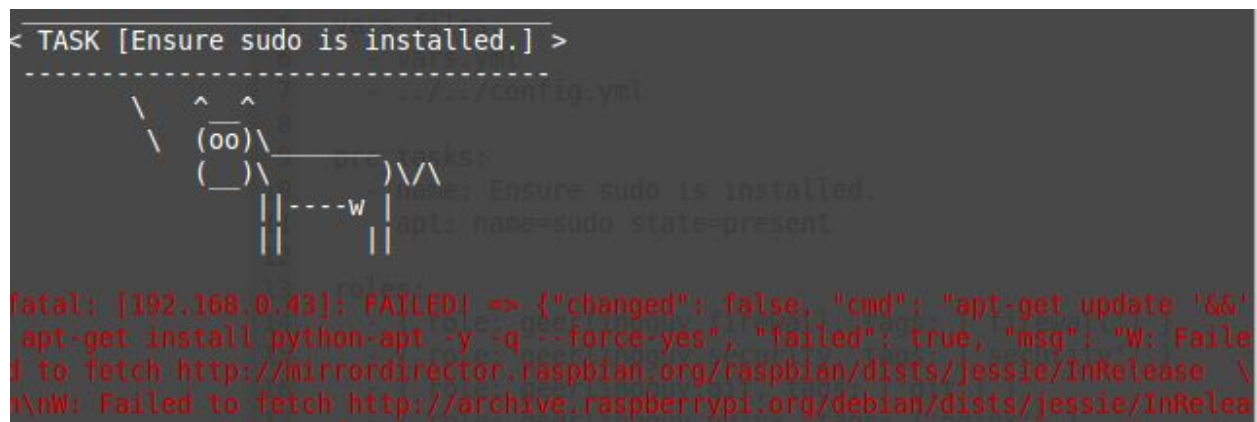
We have been moving forward with configuration of the server cluster using ansible to automate configuration. The first step in this process is to use ansible to configure the host information on each of the pis in order to make each pi identifiable when it comes time to install role dependencies. Ansible makes this task easy, rather than opening up six separate ssh clients we can use ansible playbooks to issue the necessary commands to all six pis simultaneously performing the necessary edits to the files containing networking information.

## Week 4

We were held up in configuring the balancer, web servers, and databases due to network configurations made in the initial steps of the project. While ansible makes it easy to perform tasks across all of the pis simultaneously we have made erroneous changes to the routing tables of the pis which have prohibited access from the pi server cluster to the outside internet. Our cluster can speak within itself but when it comes time



to download packages for role dependencies the server cannot access the outside web. While this is an easy to fix task, it was hard to identify exactly why the server did not have access to the web. A simple ifconfig command revealed that the wlan0 interface was reaching the web, but the routing tables as altered by our ansible playbook were preventing the cluster from reaching that interface.



```
< TASK [Ensure sudo is installed.] >
-----
      ^ ^
      (oo)\
      ( _)\
      ||---w||
      ||
      ||

fatal: [192.168.0.43]: FAILED! => {"changed": false, "cmd": "apt-get update '&&'
apt-get install python-apt -y -q --force-yes", "failed": true, "msg": "W: Failed
to fetch http://mirrordirector.raspbian.org/raspbian/dists/jessie/InRelease \
n\\W: Failed to fetch http://archive.raspberrypi.org/debian/dists/jessie/InRelea
```

**Figure 4:** Ansible Error message indicating that the pi in question is not reaching the internet.

Once the network issue was identified it could be fixed by simply appending the default router gateway to the routing table in the pi by performing the following command:

```
sudo route add default gw 192.168.0.1 wlan0
```

Issues such as this one were abundant when using the playbook outlines we sourced from the web (<https://github.com/geerlingguy/raspberry-pi-dramble>) As a lot of the code here relies on older versions of some of the software involved and that a single type in a given variable field can lead to the halting of system configuration.

One such problem occurred when the playbook tried to delete the root user at the local host for the raspberry pi. Even though our playbook changed host file to give the host a new ip address and name, we found that when the playbook tried to log in to the SQL server using the root@localhost user it was unable to gain access to the server. This issue could be due to changes made to the host file in the pi, so this step had to be completed manually by creating a root@% (wildcard) user for the completion of this task.



After much troubleshooting and head scratching, we were ultimately capable of installing all of the required elements for a drupal deployment to the server cluster. Unfortunately the initial configuration errors which made installation difficult up to this point had caused problems throughout our server cluster and the drupal deployment we prepared was ultimately inaccessible from the load balancer. While we were capable of hosting a website on the web server pi itself, we were not able to access this site via the load balancer and hence this portion of the project was left incomplete.

## **Week 5**

Refocusing our efforts we have decided to focus on getting a better handle on using Ansible to issue basic commands in order to see how ansible can be used to issue commands and complete tasks on multiple devices simultaneously. We prepared a number of tasks and experiments in order to test the capabilities and applicability of ansible.

In order to test the processing times for the various pis we created a simple python script which generates a ten thousand index array populated with pseudo-random integers between 1 and 1000000, the script then uses the bubble sort algorithm to sort the array (this code is available as /code/numSort.py). This test revealed that the amount of time elapsed when sorting the array was dependent upon hardware as well as upon the tasks and interrupt handlers loaded into memory on the pis. Generally it was discovered that in sorting the 10000 index array the two models of pis used had the following average times elapsed:

MODEL A: 265 seconds

MODEL B: 86 - 128 seconds

The range of times discovered for the model B shows that the time elapsed is not necessarily hardware dependant with the pi handling NGINX server requests taking a greater amount of time to complete it's task.

```

dramble $ ansible all -i inventory -a "python ELSpring2016/code/numSort.py" -k
SSH password:
192.168.0.43 | SUCCESS | rc=0 >>
--- 0.145666837692 seconds to generate ---
--- 10000 entries in array ---
--- 86.3854980469 seconds to sort ---
192.168.0.44 | SUCCESS | rc=0 >>
--- 0.214700937271 seconds to generate ---
--- 10000 entries in array ---
--- 128.186233997 seconds to sort ---
192.168.0.50 | SUCCESS | rc=0 >>
--- 0.527886867523 seconds to generate ---
--- 10000 entries in array ---
--- 265.817492962 seconds to sort ---

```

**Figure 5:** Showing the output for one run of the numSort.py program. The pis may be identified as follows, 192.168.0.43 is the balancer pi (model B), 192.168.0.44 is the webserver pi (model B), and 192.168.0.50 is the database pi (model A)

### Learning Outcomes:

Structurally we need to reassess how we approach a project, it is important to whiteboard the project early and approach the task holistically. We took an ad-hoc approach to this project and our progress was inhibited accordingly. Had we done more of the requisite research regarding networking, NGINX, and Ansible before starting the project we would have been more well prepared to troubleshoot problems when they occurred.

Furthermore this task has revealed a lot of the difficulties and challenges that are faced in network administration. While one-click installs and preconfigured networks have greatly increased the ease of use for clients of web servers, this simply makes a black box out of all the components involved in hosting a website.

Finally, we discovered that the networking issues we were having with our drupal deployment and communication within the server cluster had been resulting from attempting to assign ip addresses to the pis outside of the range allotted by the residential network router we were using. This oversight contributed greatly to the hinderance of our project progression.

Overall, this project has made us far more confident and capable using Linux systems to accomplish distributed tasks. While we failed in getting the drupal deployment to work on our server cluster we were able to learn something about networking while becoming very well attuned to using the linux command shell and associated utilities to accomplish tasks.

```
dramble $ ansible all -i inventory -m ping
192.168.0.43 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.0.44 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.0.39 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.0.50 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.0.37 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
192.168.0.62 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
}
```

Extras:

Video: USING ANSIBLE TO CHANGE COLOR OF LEDs

<https://youtu.be/U52EKBXSJqo>

<https://youtu.be/VdlbMvuBpsk>

## Group

Joseph Giannitti

<https://github.com/N03048839>

- NGINX server configuration

- Hawks WPA Troubleshooting
- Presentation Slideshow
- Ansible LED Scripting

Shane Kelly

<https://github.com/N03173726>

- Documentation
- Final Report
- Processing Analysis