

INTERNET & NETWORK SERVICES

ASSIGNMENT 2

ADAM LLOYD – R00117318 - WEB DEVELOPMENT –

<https://youtu.be/EIIL0E2ujIA>



ubuntu

TABLE OF CONTENTS

Overview	3
DHCP server.....	4
DNS server	6
Ruby on Rails	8
SSH server	12
Security.....	13
Final Report – Summary & Conclusions.....	15
References	17

OVERVIEW

For this project an Ubuntu server was required to provide DHCP and DNS services to the network, provide SSH login to the server and host a Ruby on Rails web application.

DHCP is a service which assigns IP addresses to the other devices on the network which makes for much easier connectivity by removing the need to manually configure network information on user's machines and other such devices.

The DNS service maps IP addresses to domain names. In our network it allows devices on the network to access services on the server which has the IP address of 192.168.0.100 by simply typing in example.com making for easier access.

In order to administrate the server and its contained Ruby on Rails installation, we will set up an SSH service which enables authorised users to connect to the server, log in and be able to use the command line just as if they were sitting at the keyboard on the server machine itself.

Finally a Ruby on Rails web application is required which is a framework which allows for fast development of web applications.

Over all of these services, a blanket of security is required to ensure no unauthorized access while still providing the functionality required.

(help.ubuntu.com, 2015)

DHCP (Dynamic Host Configuration Protocol) is a service that provides IP addresses to other devices on the network. When a device is connected to the network and configured to get its IP information via DHCP it broadcasts a signal on the network which the DHCP server intercepts and returns an IP address from within its specified range.

To install the DHCP service on the Ubuntu server use the command:

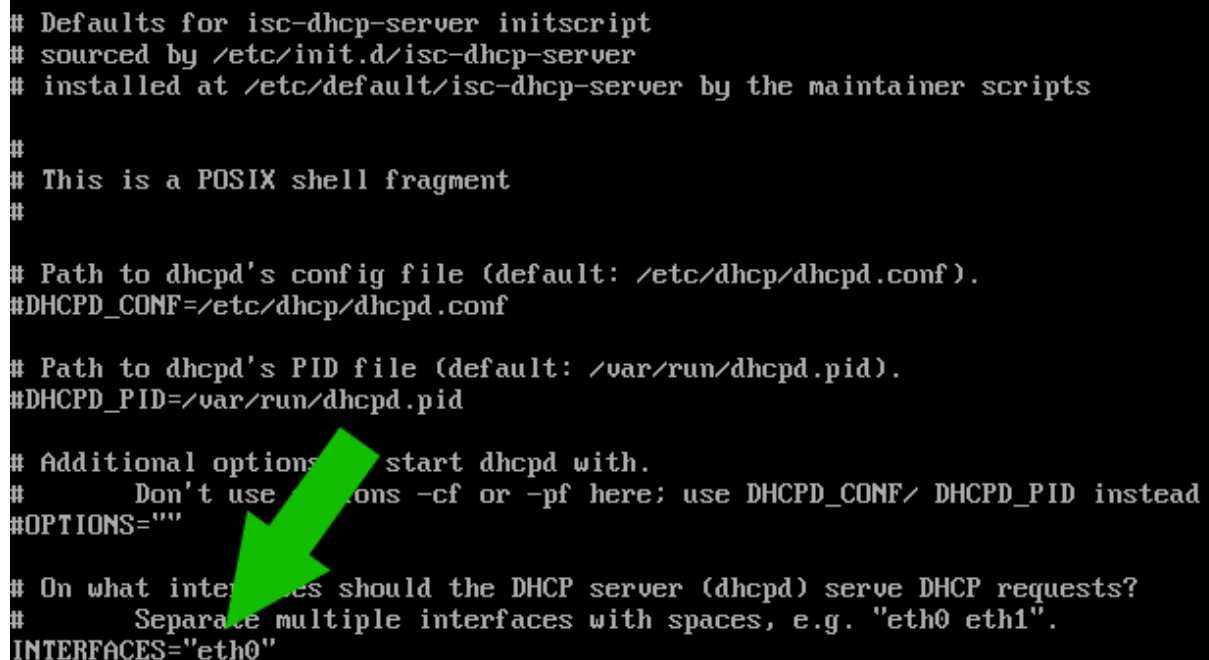
```
sudo apt-get install isc-dhcp-server
```

Once the installation is complete you will have a DHCP server service running but some configuration is required for it to work correctly.

Firstly you need to tell it which network adapter to use when listening for other devices on the network. You can modify this value by editing the following file:

```
sudo nano /etc/default/isc-dhcp-server
```

Once editing this file, modify the value shown in the screenshot to the name of the network adapter "eth0" or whichever network adapter your server is using. If you are unsure view the file **/etc/network/interfaces** and see what adapters you have configured.



```
# Defaults for isc-dhcp-server initscript
# sourced by /etc/init.d/isc-dhcp-server
# installed at /etc/default/isc-dhcp-server by the maintainer scripts
#
# This is a POSIX shell fragment
#
# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPD_CONF=/etc/dhcp/dhcpd.conf
#
# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPD_PID=/var/run/dhcpd.pid
#
# Additional options to start dhcpd with.
# Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""
#
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="eth0"
```

Now your DHCP service is listening on the correct network adapter, we need to tell the server what IP addresses to give out to devices on the network.

To do this we need to edit the configuration file for the DHCP server using the following command:

```
Sudo nano -w /etc/dhcp/dhcpd.conf
```

Modify the file to look like the following screenshot:

```
subnet 192.168.0.0 netmask 255.255.255.0 {  
    range 192.168.0.101 192.168.0.130;  
    option domain-name-servers 192.168.0.100, 8.8.8.8;  
    option domain-name "example.com";  
    option routers 192.168.0.100;  
    option broadcast-address 192.168.0.255;  
    default-lease-time 600;  
    max-lease-time 7200;  
}
```

This tells the DHCP server to provide IP addressed in the **range** 192.168.0.101 to 192.168.0.130 so any device requesting a DHCP IP address will get an IP in this range. We also specify which DNS servers to use, some of the network information that is not important and configure the lease time. The lease time is the measure of how long the DHCP server will “reserve” an IP address for a device. It may be quit inconvenient for get a different IP address each time you connect to a network so the DHCP server maintains a list of addressed it has given out which are matched with the MAC address of the device it gave the IP to. This enables it to identify devices and give them their “reserved” IP address when they reconnect to the network. Saving these reservations forever though may result in problems so we set a lease time to tell the server how long to keep these active for before releasing the IP address to use with another device.

Now the configuration is completed, restart the service with the following command and your DHCP server will be ready to provide IP addressed on the network.

```
sudo service isc-dhcp-server restart
```

DNS SERVER

(help.ubuntu.com, 2015)

Now that the server is providing DHCP IP addresses to devices on the network we need to install and configure a DNS (Domain Name Service). This service will translate domain names such as example.com into the IP address of the correct device.

For Ubuntu we use a service called Bind9 and it can be installed via the following command:

```
sudo apt-get install bind9
```

Our DNS server will be configured at a Primary Master DNS server which means that it will translate any domain names that its client devices request into the correct IP address using zone files. Basically it will look at any domain names given to it and see if it has a record of them with a matching IP address and provide that to the user if it does.

Next we need to add a DNS zone record to the Bind9 service. To do this edit the following file and change the content to match the image shown.

```
sudo nano /etc/bind/named.conf.local
```



```
zone "example.com" {
    type master;
    file "/etc/bind/db.example.com";
};

zone "1.168.192.in-addr-arpa" {
    type master;
    file "/etc/bind/db.192";
};
```

This step tells the Bind9 service that the zone example.com is handled using the file /etc/bind/db.example.com which contains all of the information it needs to get the correct IP address. The next section of this file is the reverse zone lookup which allows the DNS to do the reverse, changing the IP address back into a domain name.

The file this points to /etc/bind/db.example.com however does not yet exist so we must create it but we don't have to do it from scratch as there are existing zone files already there.

We can copy one of these and then edit it to make the changes we need.

```
sudo cp /etc/bind/db.local /etc/bind/db.example.com
```

This will copy the db.local zone file into a new file called db.example.com which we can then edit via:

```
sudo nano /etc/bind/db.example.com
```

Modify the file to match the image below.

```
;; BIND data file for local loopback interface
$TTL      604800
@         IN      SOA      example.com. root.example.com. (
                        2      ; Serial
                        604800 ; Refresh
                        86400  ; Retry
                        2419200 ; Expire
                        604800 ) ; Negative Cache TTL
;
@         IN      A        192.168.0.100
;
@         IN      NS       ns.example.com.
@         IN      A        192.168.0.100
@         IN      AAAA     ::1
ns        IN      A        192.168.0.100
```

Restart the service with the following command and your DNS server will be up and running.

```
sudo service bind9 restart
```

(gorails.com, 2015)

INSTALLING RUBY

To install Ruby on Rails it is not as simple as installing just one service from a repository. It requires several supporting applications to be installed and configured first.

Firstly we install `rbenv` which is a Ruby version management tool. This will make it easier for us to install the core dependency of Ruby on Rails, the tools required to run the Ruby programming language. To install this we use the following command:

```
sudo apt-get install rbenv
```

Next we run the following command to install all of the dependencies required for the next steps:

```
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev  
libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev  
libcurl4-openssl-dev python-software-properties libffi-dev
```

As you may be able to see from the above command we are installing `git`, a software versioning tool, `sqlite`, a simple database tool and various other packages which Ruby needs.

Then we need to install the Ruby build package which is not in the standard Ubuntu package repositories but rather on Github. Since we installed `git` with the other dependencies earlier we can use the `git` command to get the files we need for the Ruby build:

```
git clone git://github.com/sstephenson/ruby-build ~/.rbenv/plugins/ruby-  
build
```

This copies the files at the `git` URI provided into our `.rbenv` plugins folder. Next we get the Ruby build gem rehash files from:

```
git clone https://github.com/sstephenson/rbenv-gem-rehash.git  
~/.rbenv/plugins/rbenv-gem-rehash
```

And now we have all of the files required to install and build Ruby. To complete this process we install Ruby with the following command to `rbenv`:

```
rbenv install 2.2.3
```


INSTALLING RAILS

Now that Ruby is installed and ready to use we need to install the Rails framework. However Rails also has some dependencies in that it required a JavaScript runtime to run. The runtime we will use is Node.js and it can be installed by adding the following repository to apt:

```
sudo add-apt-repository ppa:chris-lea/node.js
```

Now that the repository is added we update apt:

```
sudo apt-get update
```

And install node.js like so:

```
sudo apt-get install nodejs
```

Rails is now ready to be installed with the following command:

```
gem install rails -v 4.2.4
```

The next command is required in order to make the rails executable available from the command line:

```
rbenv rehash
```

Rails is now installed and ready to create your app.

To create the application “myapp” use the following command:

```
rails new myapp
```

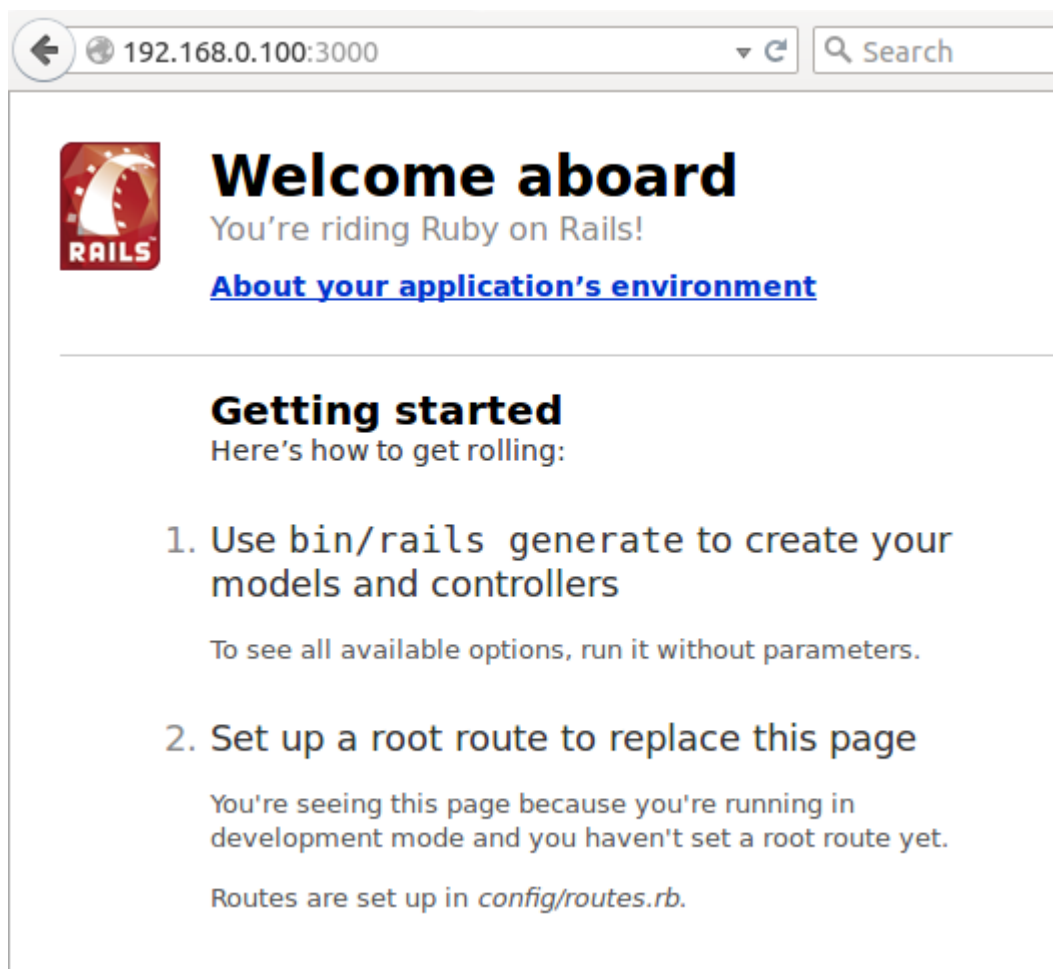
This will create all of the required files to run your app in a sub directory called “myapp”. Navigate into that directory using `cd myapp` and create the database that rails will use with:

```
rake db:create
```

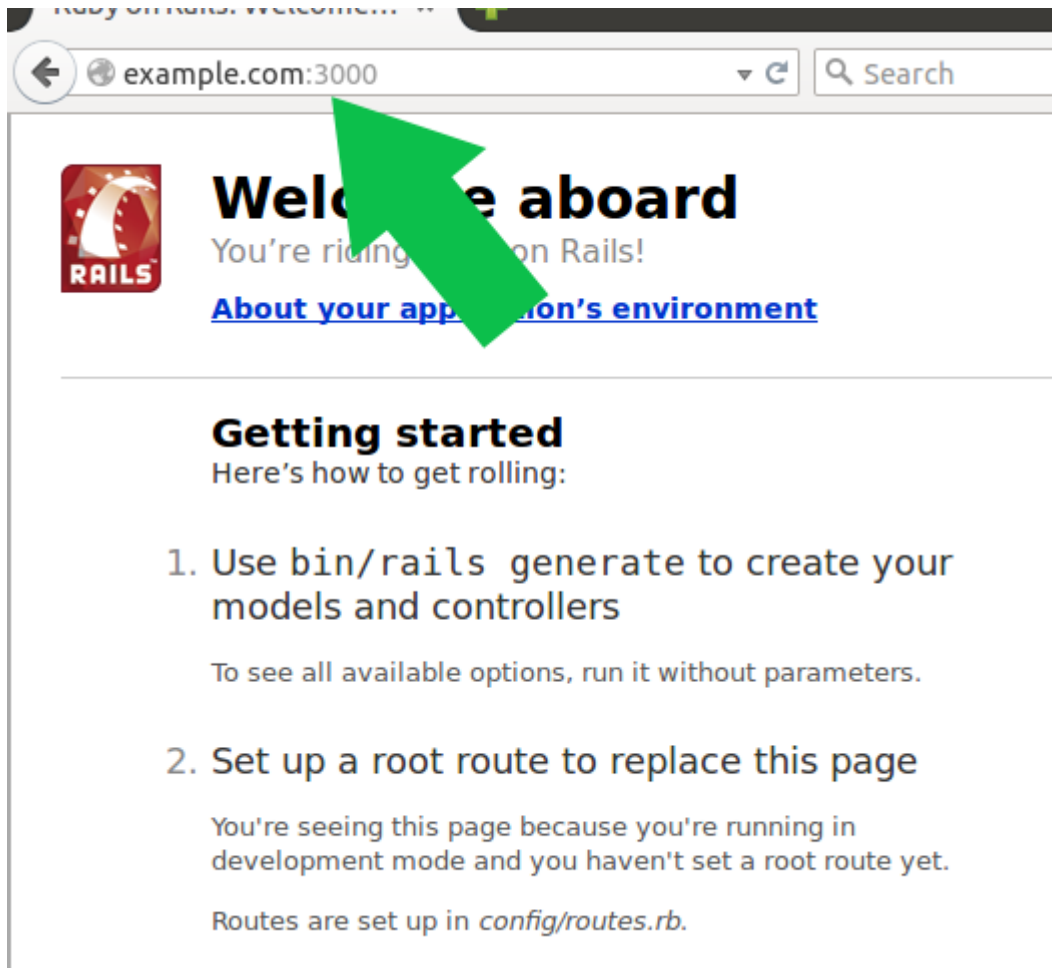
Then start the server using the IP address of the server 192.168.0.100 like so:

```
rails server -b 192.168.0.100
```

You will now be able to access the application from any client on the network by navigating to 192.168.0.100:3000 in your web browser. You will be presented with the following screen:



However, since we configured the DNS server earlier we have mapped the servers IP address to the domain name example.com. As such we can access the rails app by going to example.com:3000 as you can see below:



The screenshot shows a web browser window with the address bar displaying `example.com:3000`. A green arrow points to the address bar. The page content includes the Rails logo, a 'Welcome aboard' heading, a subheading 'You're riding the Rails!', a link 'About your application's environment', a 'Getting started' section, and two numbered steps for getting started.

Welcome aboard
You're riding the Rails!
[About your application's environment](#)

Getting started
Here's how to get rolling:

1. Use `bin/rails generate` to create your models and controllers
To see all available options, run it without parameters.
2. Set up a root route to replace this page
You're seeing this page because you're running in development mode and you haven't set a root route yet.
Routes are set up in `config/routes.rb`.

(help.ubuntu.com, 2015)

In order to gain remote access to the server we can use the SSH service to log in from a remote client via a terminal window or application such as Putty. However, before this can be done we need to install an SSH server and the one Ubuntu uses is Open SSH.

We can install it using the following command:

```
sudo apt-get install openssh-server
```

To configure the SSH server we need to edit the `sshd_config` file in the `/etc/ssh` directory. As this is a sensitive file it may be prudent to make a backup copy of the file in your home directory using the following command:

```
sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.factory-defaults
```

And then editing the file with:

```
sudo nano /etc/ssh/sshd_config
```

Once editing this file, many settings for the SSH server can be modified. We can restrict which users can access the SSH server with the `AllowUsers` setting. We can set a rate limit to prevent attackers making multiple requests to try and guess the login credentials using this command:

```
sudo ufw limit ssh
```

It may also be prudent to display a banner message to the user when they log in via SSH notifying them that the system is private and unauthorized access is prohibited. To do this, simply replace the line: `#Banner /etc/issue.net` with `Banner /etc/issue.net`

Lastly, we need to ensure that the SSH connections are not blocked by the firewall. This is detailed in the next section.

SECURITY

In order to add some increased security to the Ubuntu server several steps can be taken.

FIREWALL

(www.thefanclub.co.za, 2015)

UFW or Uncomplicated Firewall is a basic firewall that will prevent many malicious attacks on the server by default without complicated configuration and is extremely simple to install.

Just install the application with the following command:

```
sudo apt-get install ufw
```

Once complete adjust a few settings to allow for http, SSH and DNS using the following:

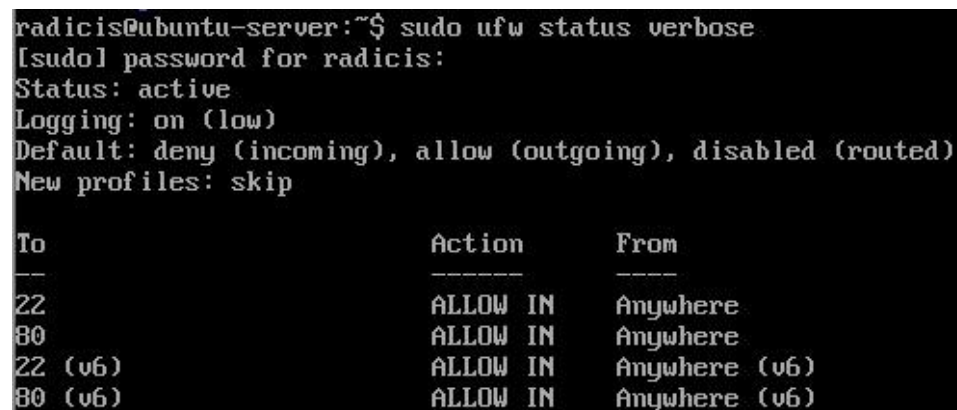
```
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow out 53 (DNS port)
```

You can then activate the firewall with the command:

```
sudo ufw enable
```

..and check the status of it using:

```
sudo ufw status verbose
```



```
radicis@ubuntu-server:~$ sudo ufw status verbose
[sudo] password for radicis:
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
```

To	Action	From
22	ALLOW IN	Anywhere
80	ALLOW IN	Anywhere
22 (v6)	ALLOW IN	Anywhere (v6)
80 (v6)	ALLOW IN	Anywhere (v6)

As you can see in the above screenshot, after using the ufw status verbose command you are shown a list of the open ports on the server. Port 80 is for basic web requests and anyone is “allowed in” so they can access your website and port 22 is open to allow for SSH or Secure Shell activity which lets a user securely log into the server from a remote location.

NMAP

(www.thefanclub.co.za, 2015)

While the UFW firewall status command was able to show open ports, a more advanced application called NMAP may be used to scan your server to ensure there are no other ports open. This is a popular application used to exploit vulnerabilities in systems so using it to check your own server may be beneficial.

Issue the following command to start the scan:

```
nmap -v -sT localhost
```

You will be presented with something that looks like the following screen:

```
PORT      STATE SERVICE
25/tcp    open  smtp
80/tcp    open  http
587/tcp    open  submission
3306/tcp   open  mysql
```

As you can see, much like the UFW status report, port 80 is open but also ports 25, 587 and 3306.

Port 3306 is open which may seem troubling but as you can see in the “SERVICE” column, it is only open to the MySQL service which is on the server itself so there is no threat of outside entities accessing it.

Port 25 allows for the server to send mail. This is fine but it is a well-known port which is widely used to distribute malware (viruses) so, as you can see, we also have port 587 open which can be used instead if the application supports it

FINAL REPORT – SUMMARY & CONCLUSIONS

As you can see from the above steps, the server is now handling a lot more services and providing much more functionality to the network, however this comes at a cost. All of these services pose a security risk to the server as more and more ports (DNS, SSH) are opened and more access is given to outside users (SSH) so security becomes much more important.

Choosing a Linux based operating system to base the project on ensures that the web services will remain as stable as possible and, with the added security tweaks, be as secure as possible. Securing the server with a firewall and allowing only the minimum amount of traffic required to run the services and securing what users can interact with what systems reduces the risk of successful attack.

Throughout the process of building this server it became clear that there was an order in which to install and configure the services such that everything stacked up correctly and complications were minimised.

I decided to install the DHCP server first as this would be the foundation for the client connecting to the network and accessing the provided services. This seemed simple at first but I was hindered by the VMware networking which would choose to connect the client to my host networks DHCP server rather than the server VM. I resolved this by removing all of the default DHCP settings in VMware network manager and the client was able to get the correct IP address.

Next I installed the DNS server so that all future interactions with the server from the client could be done more easily via example.com instead of the server IP address. I ran into issues here in that, while debugging the DHCP server I had modified the server's IP address and neglected to change it in both the Zone file and the db. Files resulting in the client looking being given the incorrect IP address resolution. I should have ensured that the DHCP was completely solid before attempting this step. Also I had enabled UFW prior to this and discovered that I had to open port 53 to allow these connections, again installing the firewall last would have been prudent here.

I then moved on to the SSH server and, using what I had learned from my previous mistakes, it installed very smoothly. I knew to open the port in UFW and the DNS was now mapped correctly so the client was able to connect using ssh example.com.

Ruby on Rails was quite troublesome in that RBENV needed to be configured along with GIT and all of the other dependencies. The guide I was following, while comprehensive, did not explain everything in detail so there was a bit of experimentation before it worked.

Once the app was created I then ran into trouble running the server. As it was running on port 3000 I had to open the port in UFW but then once connected I ran into problems with the MySQL2 gem file which just refused to install so ultimately opted to use an SQLite database for the application.

I discovered that I could not connect to the app from the client as it was running on localhost so needed to run the server with the `-b 192.168.0.100` option to bind it to the servers IP address on the network.

Luckily in the end I was able to demonstrate all of the required services working correctly and these can be viewed on YouTube with the link provided on the cover page of this document.

REFERENCES

- gorails.com. (2015, 11 31). *Go Rails*. Retrieved from <https://gorails.com>: <https://gorails.com/setup/ubuntu/15.10>
- help.ubuntu.com. (2015, 11 25). *help.ubuntu.com*. Retrieved from help.ubuntu.com:
<https://help.ubuntu.com/community/isc-dhcp-server>
- www.thefanclub.co.za. (2015, 10 22). Retrieved from The Fan Club: <https://www.thefanclub.co.za/how-to/how-secure-ubuntu-1204-lts-server-part-1-basics>