Mat Hanson

11575596 | mathanson87@gmail.com

Assessment 3 – Virtual Server Practical

ITC514 – Linux Server Administration

# Part One

In this section, I will be comparing IaaS offerings from Microsoft, Amazon and 1&1. I chose 1&1 as the third service of comparison due to the number one overall ranking they were awarded in the Cloud Spectator 2017 Cloud IaaS Provider Report (Cloud Spectator, 2017, p. 6). Throughout this comparison, I will be comparing services from each company’s offerings in North America due to the higher amount of information and reporting available for that region.

## Service Features

Amazon’s server hosting service is called Elastic Compute Cloud (EC2) which offers a large variety of Linux distributions. New instances are created using AMIs (Amazon Machine Image) which are templates offering the following Linux distributions - Amazon Linux, SUSE, RHEL, Ubuntu, CentOS, Debian, Kali, openSUSE, Gentoo, Mint, FreeBSD, CoreOS, Oracle, along with several others specifically available through the Community AMI section (Amazon, 2017).

Microsoft’s server hosting service is called Azure Virtual Machines which offers a less diverse range of Linux distributions. The distributions that are currently available are CentOS, CoreOS, Debian, Oracle, RHEL, SUSE, openSUSE and Ubuntu (Zarkos, Foulds & Squillace, 2017, para. 2).

1&1’s server hosting service is called 1&1 Cloud Server and is the most limited in terms of distribution choice, offering Ubuntu, CentOS and Debian (1&1, 2017, para. 1).

Amazon offers two different storage options for use with EC2 – Elastic Block System and Elastic File System. EBS offers low latency volumes targeted at performance needs while EFS offers volumes which scale automatically. Amazon’s storage options allow the user to specify how much space they need and pay per gigabyte rather than be allocated a set amount.

Microsoft’s Managed Disk service handles the storage solutions for Virtual Machines. This service offers two tiers related to performance (Standard or Premium) and allows you to specify the size of the storage volume that is required. This service also manages automatic scalability of the volume.

1&1 have only one storage service provided for each individual cloud server instance. The maximum size for a server instance is 500GB.

## Redundancy and Backup

Amazon (2013) has an SLA in place which is an incentive for the company to reduce outages. Amazon’s SLA offers a service credit of 30% back to the customer if the service monthly uptime is less than 99.0%. If monthly uptime is between 99.0% and 99.95%, the customer will receive 10% service credit.

Microsoft (2017) has an SLA that is split between Single Instance VMs and VMs in an Availability Set. I will use the Single Instance for comparative sake. There are three levels within Microsoft’s SLAs - <99.9%, <99% and <95%. The service credit for each of these levels is 10%, 25%, and 100% respectively. Microsoft has acknowledged the impact of having the service below 95% uptime by crediting their customers with the entire cost of the service, which is better than Amazon’s offering of a maximum of 30%.

1&1 state that they have a guarantee of 99.9% uptime but I was unable to find the SLA policy anywhere on their website.

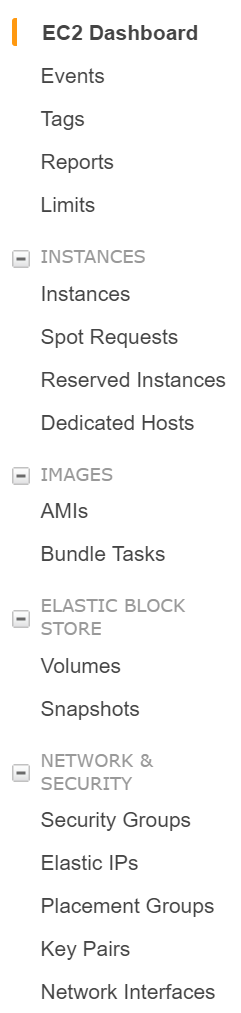
All three of these services allowed manual and automatic backup of server instances.

# Part Two

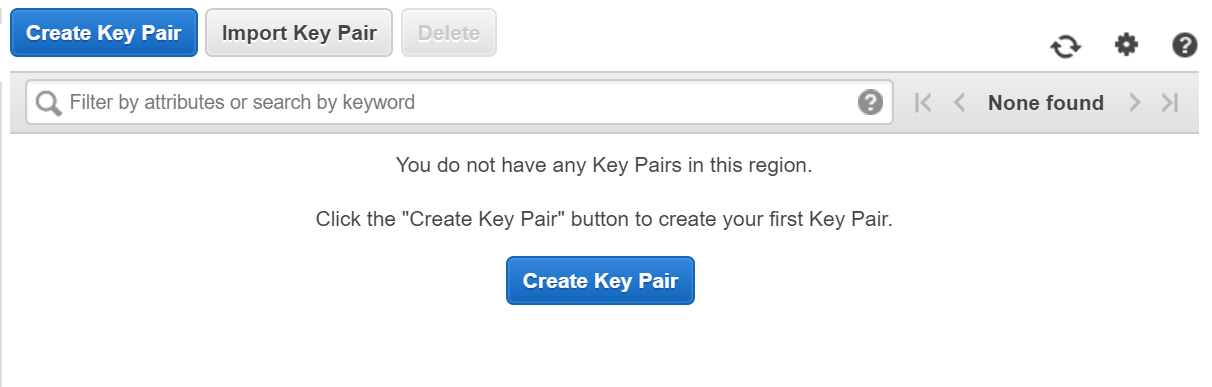
I decided to use Amazon AWS since I already have a trial account with Amazon and due to its widespread use. After reading the “Setting Up with Amazon EC2” guide (Amazon, 2017, p. 19) I found that I would need to set up a key pair before I create a virtual server instance.

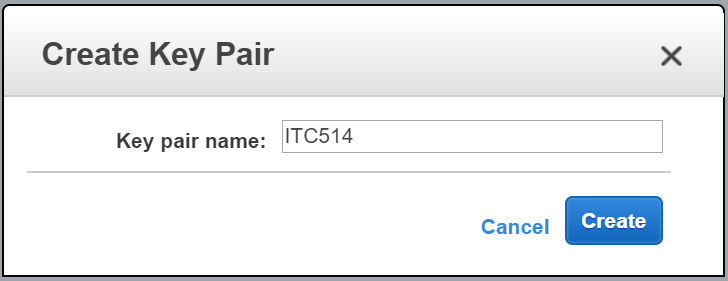
## Creating Key Pair

To create a key pair, I clicked on Network & Security>Key Pairs from the EC2 Dashboard.

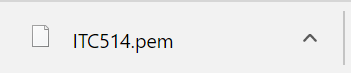


I then clicked on “Create Key Pair” and gave the key pair a name – “ITC514”.

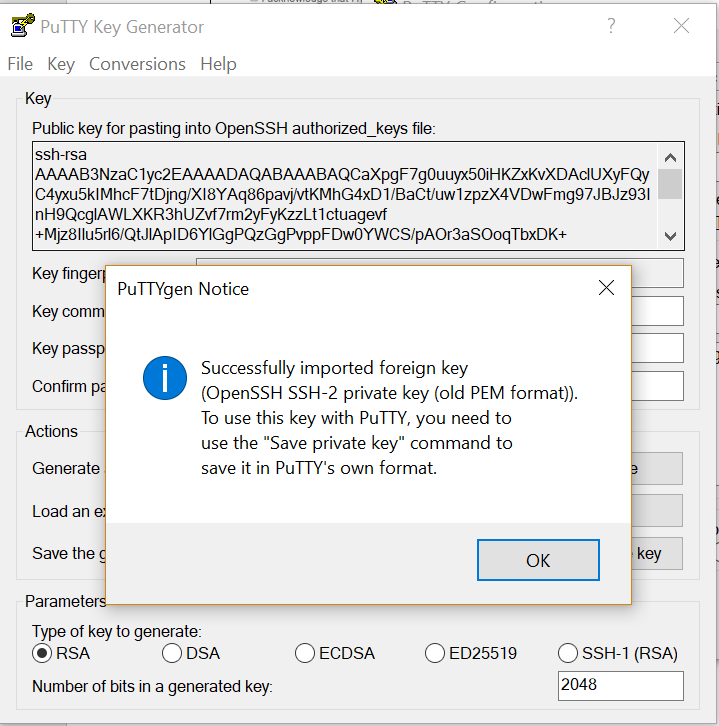


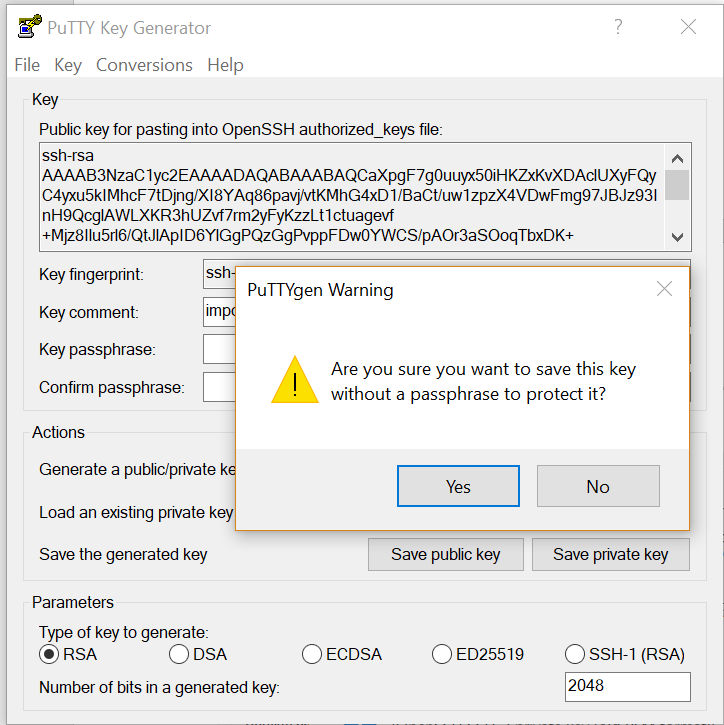


The private key of the key pair was then downloaded to my PC which I stored in a safe place.



I then used the PuTTY Key Generator to convert the .pem file to a .ppk file that is required for use with PuTTY, my SSH Client of choice.

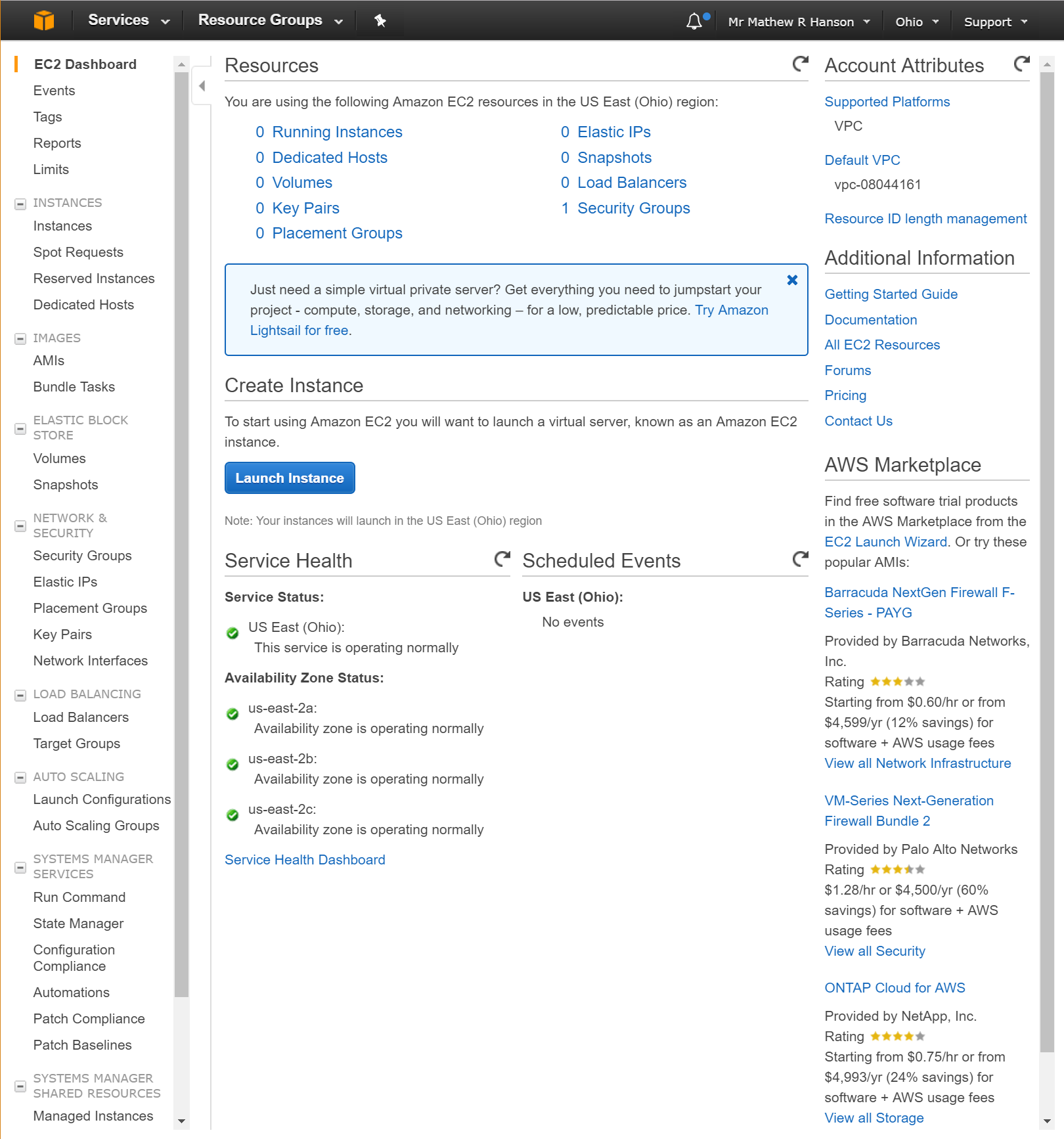




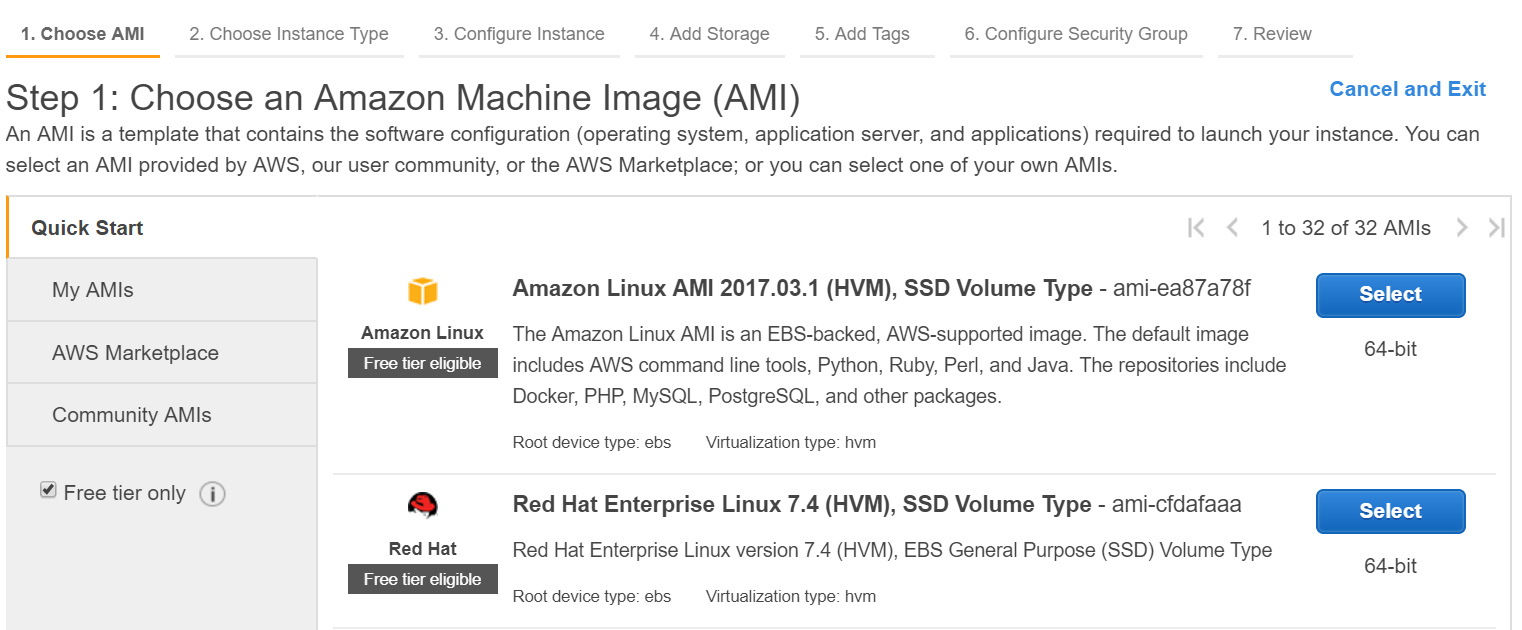
I now have the private key converted to a .ppk file.

## Creation of Virtual Server Instance

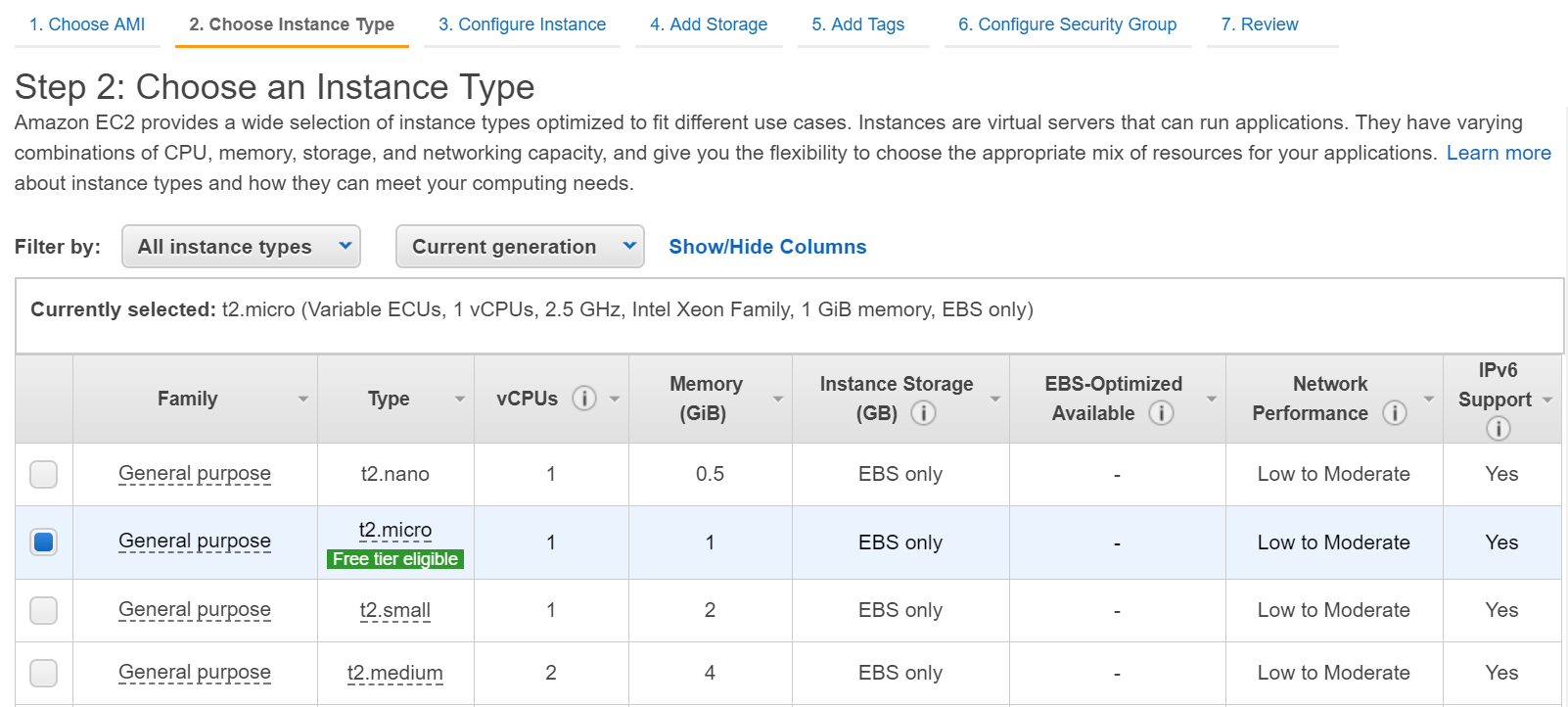
Now that I have the key pair created, I can create a virtual server instance. From the EC2 Dashboard, I clicked on “Launch Instance” under “Create Instance”.



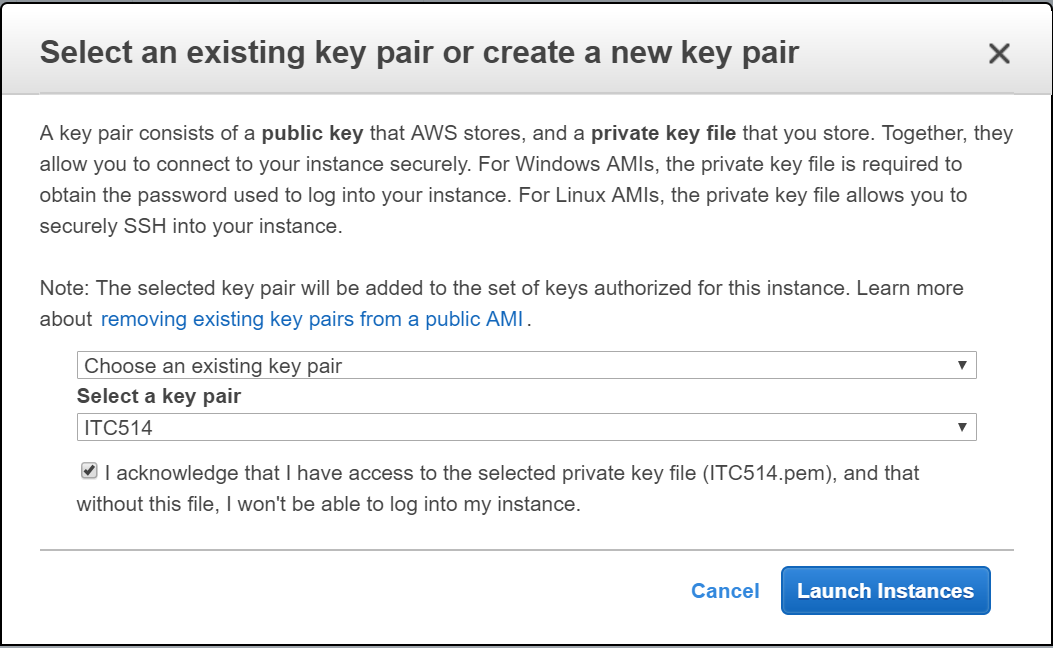
This then took me to a list of available AMIs to choose from. I will be choosing one of the default AMIs available in the free tier. I decided on using the Red Hat Enterprise Linux AMI due to its similarities to CentOS, which I have experience with from the first two assignments.



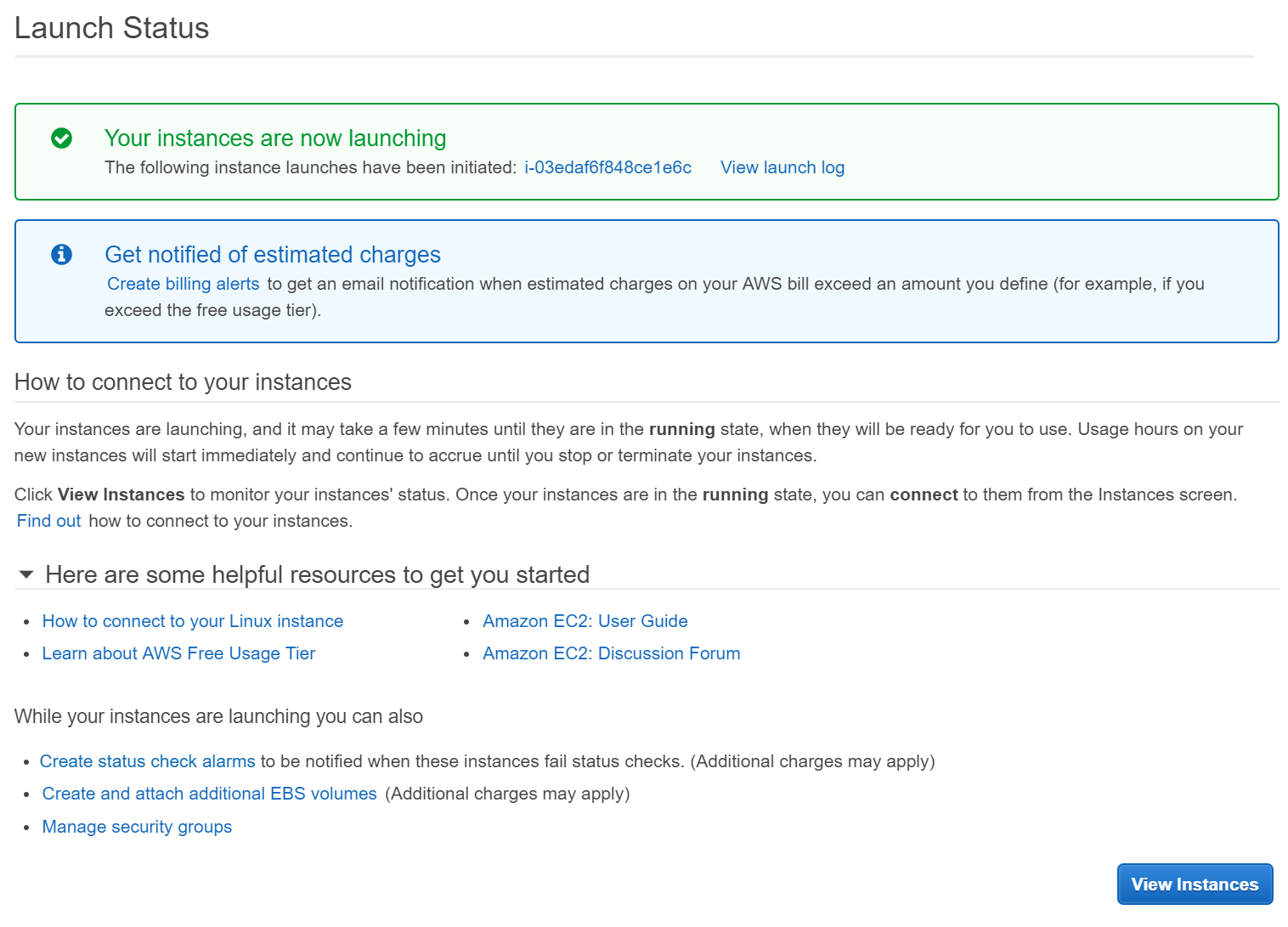
I then chose the only Instance Type that is available to free tier accounts, t2.micro.



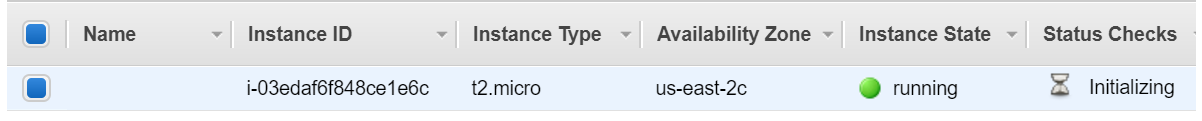
I checked the rest of the configuration and everything was fine as default. I then clicked “Review and Launch” and then “Launch” from the review page. I was then asked to either create a new key pair or choose an existing key pair. I chose the key pair that I created earlier – “ITC514” and clicked on “Launch Instances”.



The next screen showed confirmation that the instance launch had been initialised. I clicked on “View Instances”.

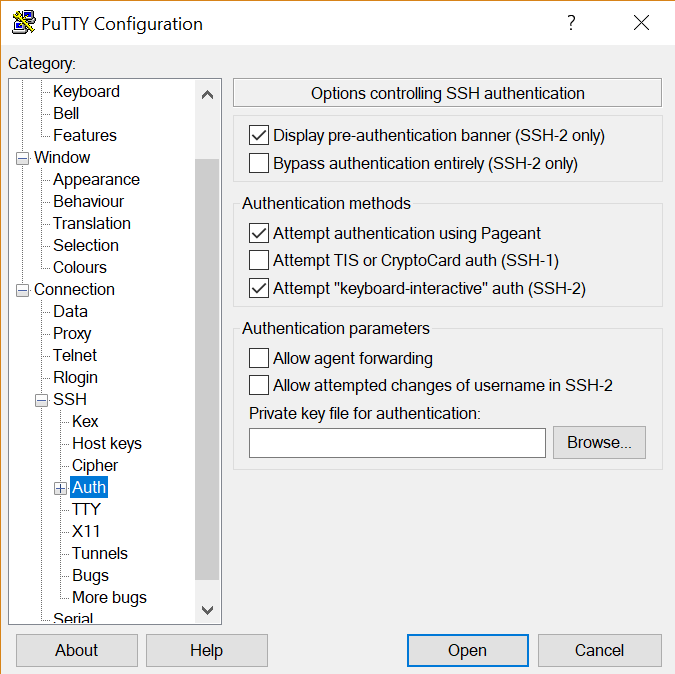


I was then able to see that the instance was still initialising.



## Connect to Instance Using SSH Client

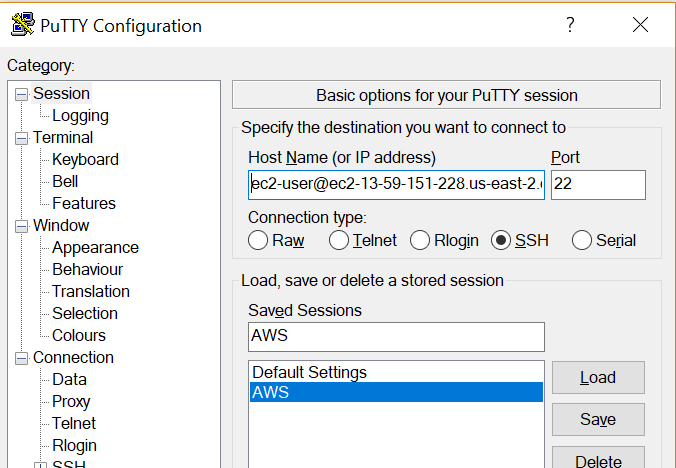
I opened PuTTY and loaded the .ppk file that was generated earlier from Connection>SSH>Auth within the Category tab as per the instructions in the PuTTY section of the EC2 manual (Amazon, 2017, p. 311).



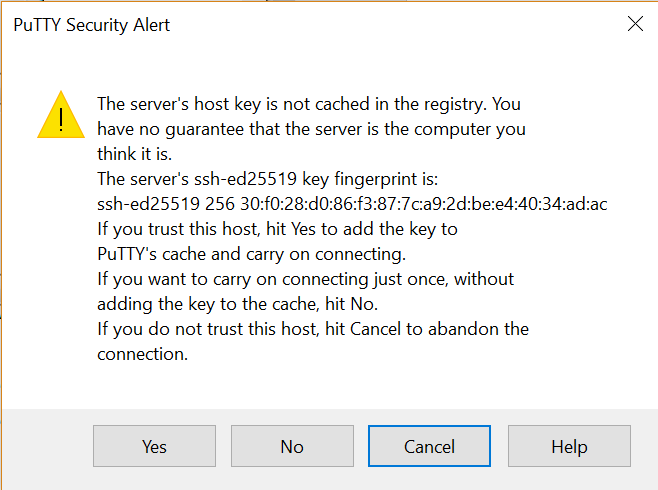
I then copied the Public DNS address of my virtual server instance from the EC2 Management Console.

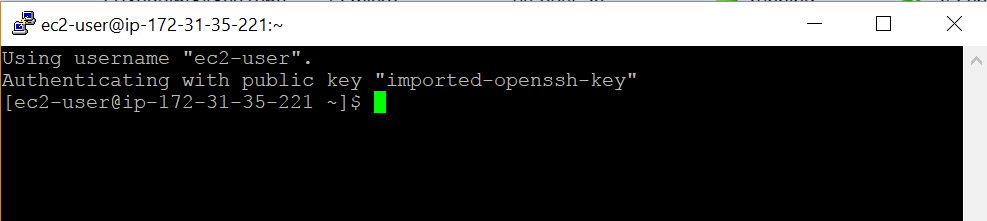


As per the manual, I entered the hostname as “ec2-user@[Public DNS]” and saved the session for later use.

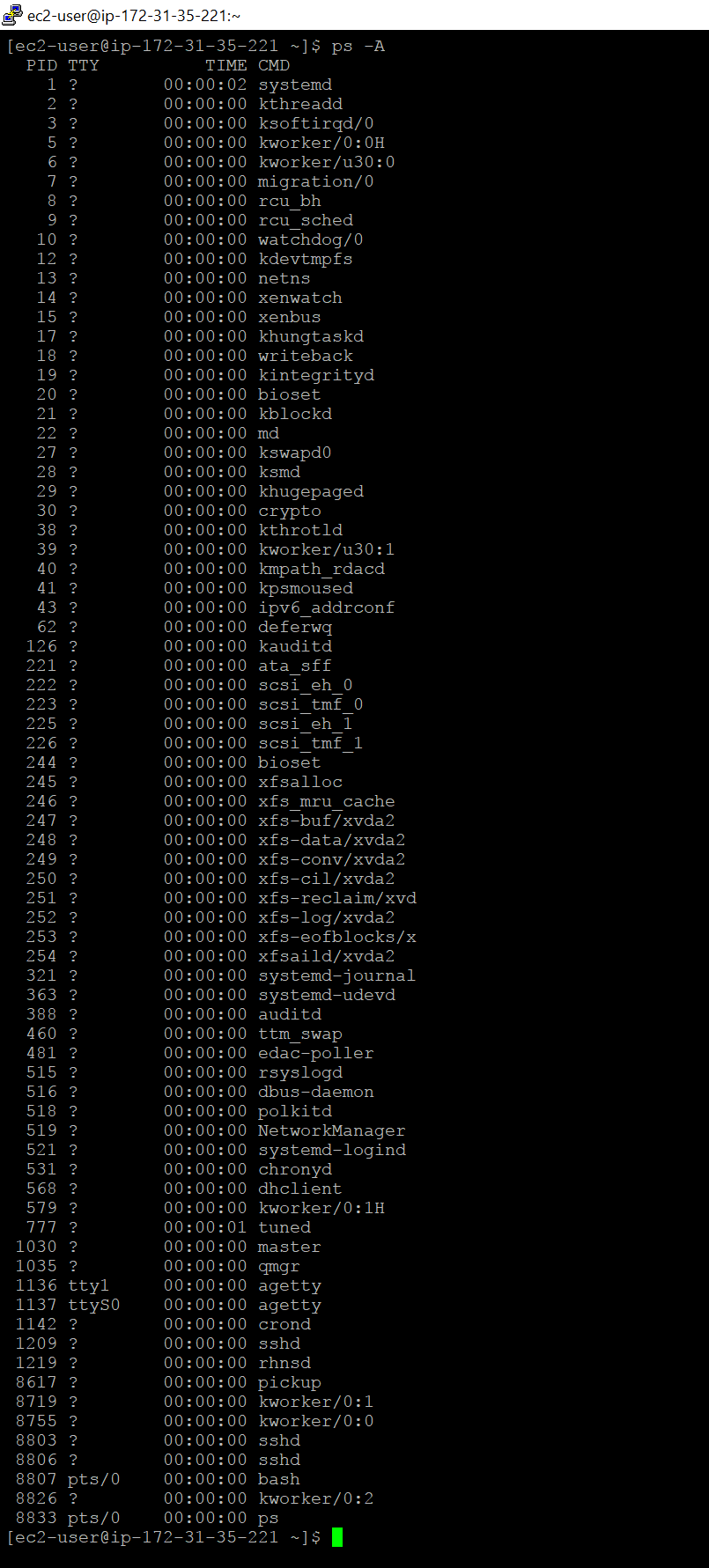


Since this was the first time that I had connected to this server via SSH, I received the following alert. I clicked Yes to add the key to cache.





Once I had successfully connected to the server, I ran a “ps -A” command to show the processes that were currently running.

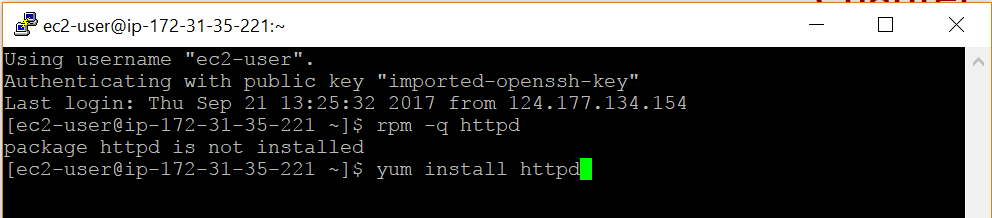


# Part Three

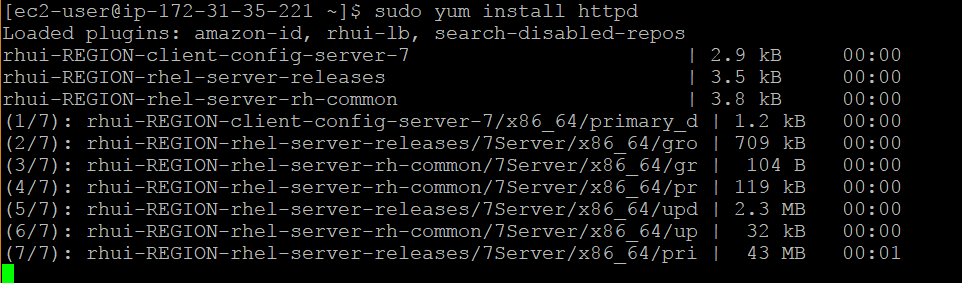
In this section, I will be choosing an HTTP server, and then installing and configuring my server instance to serve a basic webpage. I have taken the advice of Nemeth, Snyder, Hein & Whaley (2011, p. 963) and chosen Apache to be my HTTP server of choice.

## Installation of HTTP Server

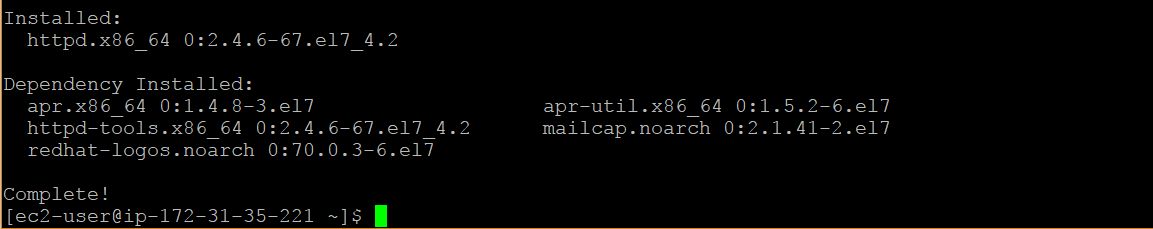
Firstly, I will need to install Apache on my server using “yum install httpd”.



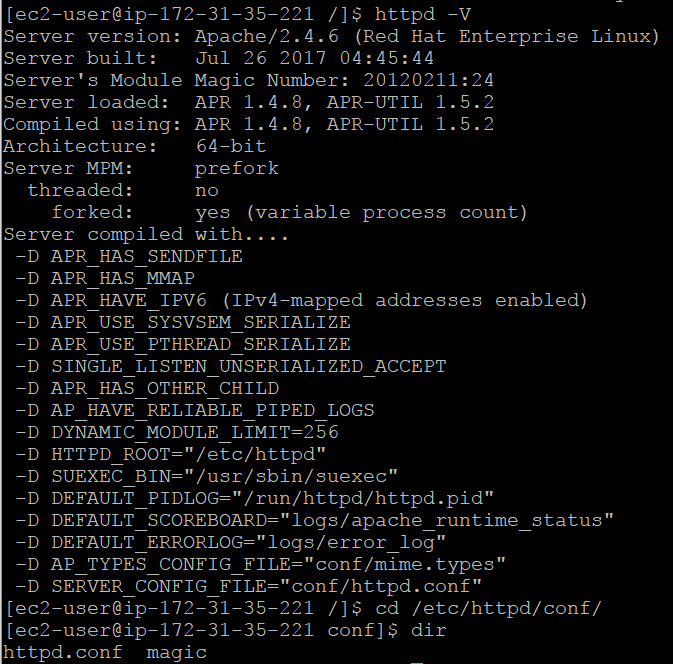
Due to being logged in under the ec2-user, not root, I will need to run the command using sudo.



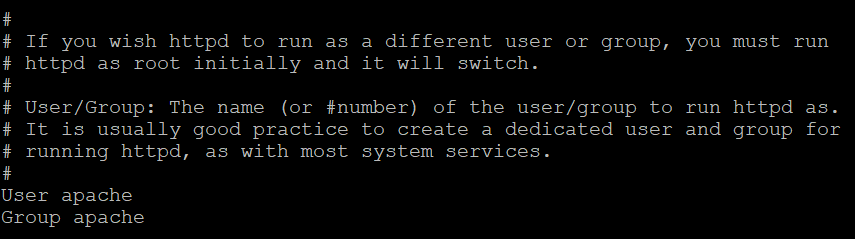
The package has now been installed.



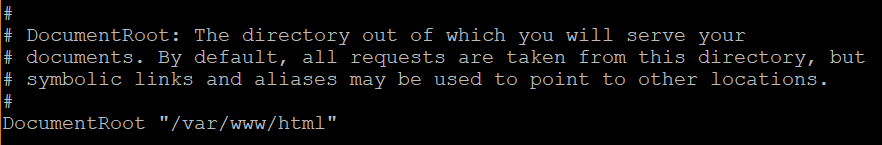
I found that the best way to find detailed information about the Apache configuration files was by using the “httpd -V” command (Rich Bowen, 2012). This command showed me where the httpd.conf file was located.



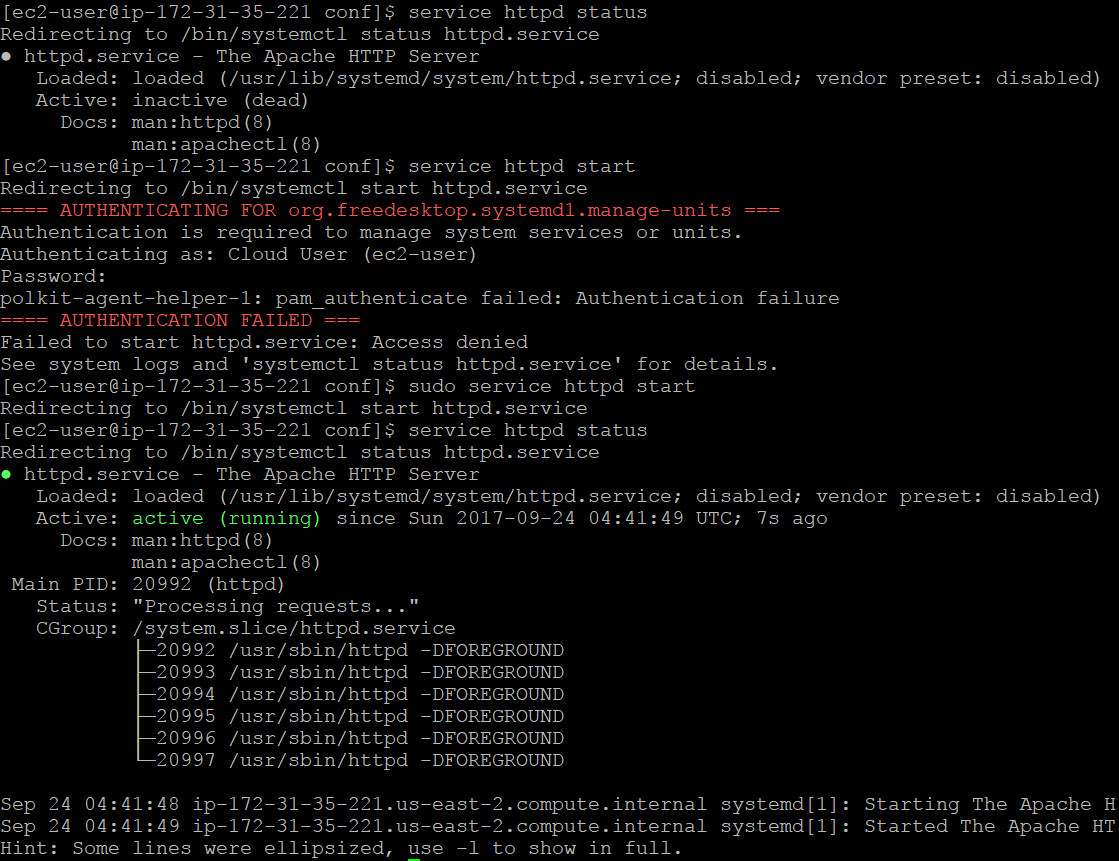
Opening the httpd.conf file, I checked a few of the default directives against what was mentioned in the textbook and everything looks fine for the basic website configuration that I required.



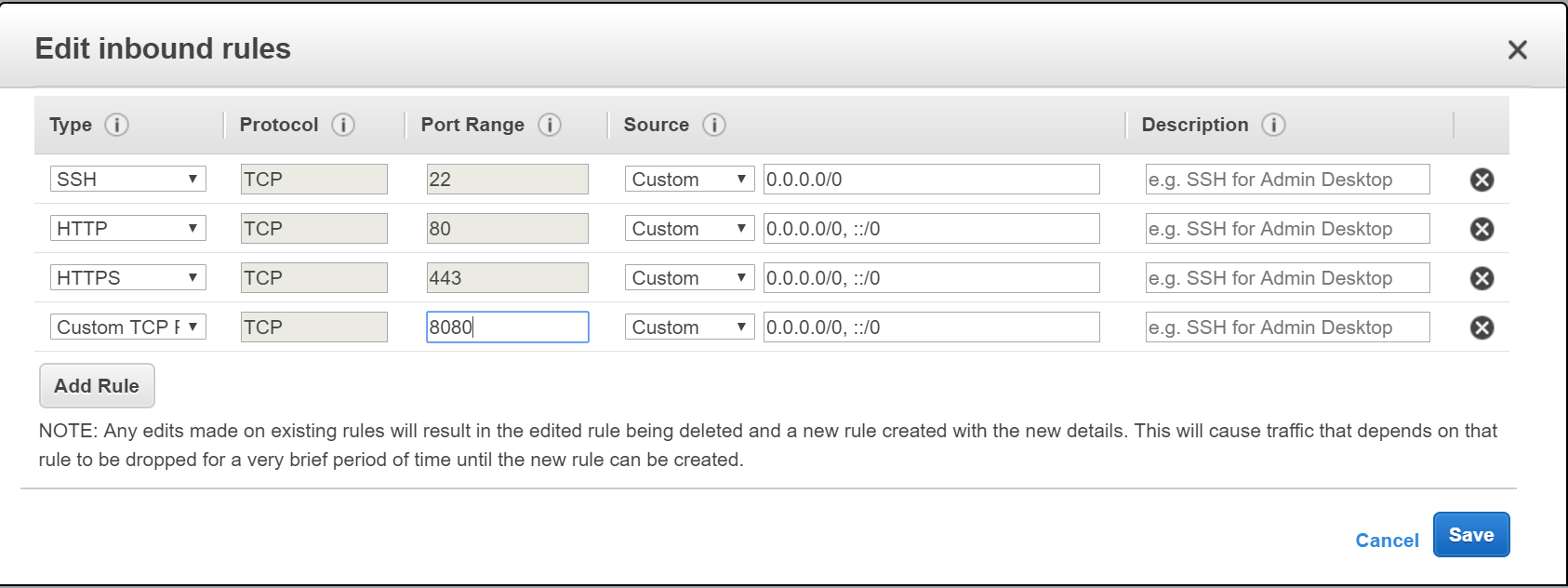
The DocumentRoot directive indicated to me where the index.html file would need to be located.



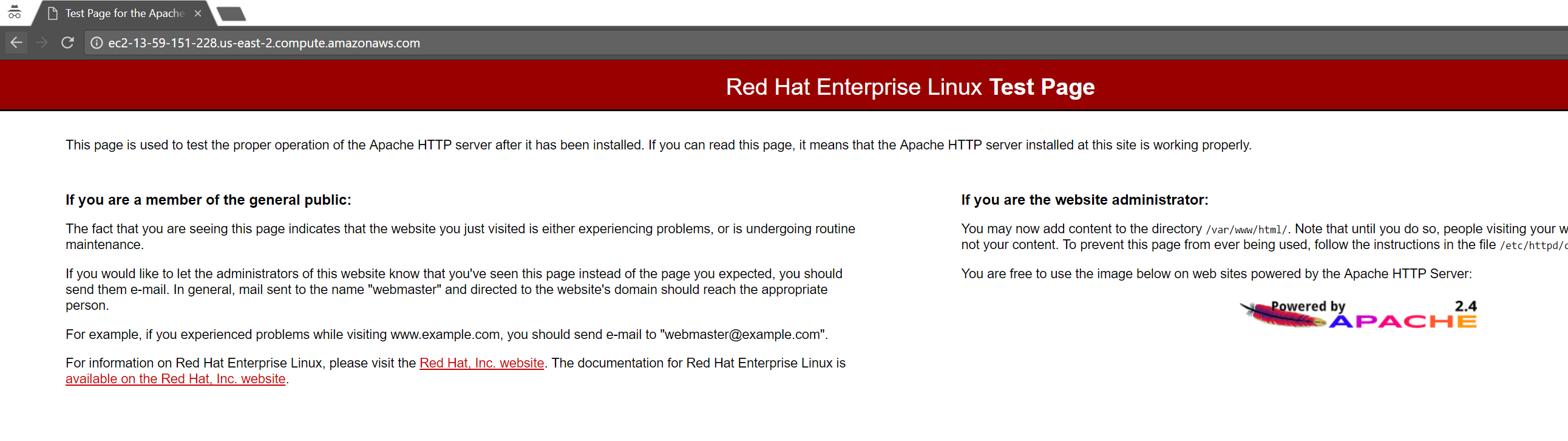
After checking the conf file, I was ready to start the service and test that it was working fine before setting up the index.html file.



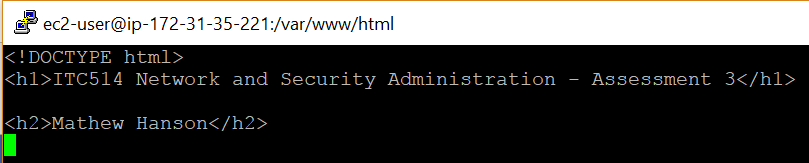
After I started the service, I attempted to navigate to the public DNS address (<http://ec2-13-59-151-228.us-east-2.compute.amazonaws.com/>) but the request timed-out. I then realised that I did not configure the security group for the instance when I was setting it up in EC2. I went back into the EC2 Management Console to check the default security group and the only Inbound Rule that was configured was for SSH. I edited the rules adding ports 80, 443 and 8080.



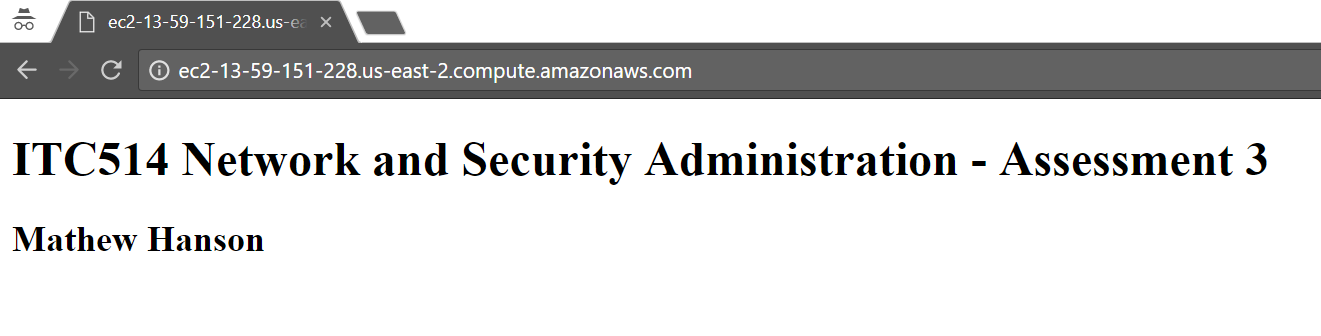
I then tested the public DNS address again, which worked this time and I was able to retrieve the RHEL Test Page.



The next step was to create the index.html file within /var/www/html. I wrote a simple piece of HTML code to display the information required.



I then tested the address again and the HTML that I wrote was displayed as expected.



## References

1&1. (2017). Cloud Server. Retrieved from <https://www.1and1.com/dynamic-cloud-server>

Amazon. (2013). Amazon EC2 Service Level Agreement. Retrieved from <https://aws.amazon.com/ec2/sla/>

Amazon. (2017). Setting Up with Amazon EC2. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html>

Amazon. (2017). Linux on AWS. Retrieved from <https://aws.amazon.com/mp/linux/>

Cloud Spectator. (2017). *Top 10 Cloud IaaS Providers Benchmark: 2017 North American Report.* Retrieved from <http://connect.cloudspectator.com/2017-cloud-iaas-providers-comparison>

Microsoft. (2017). SLA for Virtual Machines. Retrieved from <https://azure.microsoft.com/en-au/support/legal/sla/virtual-machines/v1_6/>

Nemeth, E., Snyder, G., Hein, T. R., & Whaley, B. (2011). Unix and Linux System Administration Handbook. Upper Saddle River, NJ: Pearson Education, Inc.

Rich Bowen. (2012, August 30). Re: Where is my httpd.conf file located apache [Blog comment]. Retrieved from <https://stackoverflow.com/questions/12202021/where-is-my-httpd-conf-file-located-apache>

Zarkos, S. A., Foulds, I., Squillace, R. (2017). Linux on distributions endorsed by Azure. Retrieved from <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/endorsed-distros>