Management Server Virtual Machine User Manual

Table Of Contents

Introduction

This user manual will help you get started with the Management Server Virtual Machine and you will be able to deploy your first cloud environment. If you haven't downloaded the VM, please check out https://github.com/sanket28/Cloud-Deployment-Utility for instructions.

The Management Server contains a DHCP server, TFTP Server, HTTP Server, MQTT Server, Cloud Images and a minimal Live Linux OS containing the deployment shell script. The management server hosts a web application for the user to select the cloud configuration and options to deploy on client systems. The shell script and the web application communicate with each other using the MQTT protocol (More information about MQTT: http://mqtt.org) to indicate when client machines are online and also to send diagnostic messages.

When the user selects the desired cloud configuration on the web application, he needs to boot the client machines. The client machines boot over the network using Preboot Execution Environment (PXE) and fetch the Debian Live OS. The Debian OS boots up and executes the shell script to deploy the cloud and send diagnostic messages to the web application.

Almost everything is set up in the VM and the user needs to do a minimal setup to get all the components of this utility working together. The newest version of Hyperdrive deploys using a single command. To test to see if your version of Hyperdrive is up to date, type 'deployCloud' into the terminal. The command is only present in the newest version of Hyperdrive.

Using the latest version, quickly setup and deploy the cloud over a private network using the 'deployCloud' command. This command checks that all the necessary servers are running and then listens for commands from the client. It should prompt you for a PEM password, enter 'cs8674'. Using a web browser, navigate to either 'ubuntu.local' or directly with the IP address '192.168.1.42'. If you are not able to connect to the Hyperdrive page, it means that the nginx web server is not running. Sometimes the 'deployCloud' command fails to start the nginx web server, if that happens simply rerun the deployCloud command a second time or manually start the nginx server with 'sudo service nginx start' and then rerun the deployCloud command.

Setting up the Management Server VM

Everything on the management server is setup to point to one IP address: 192.168.1.42

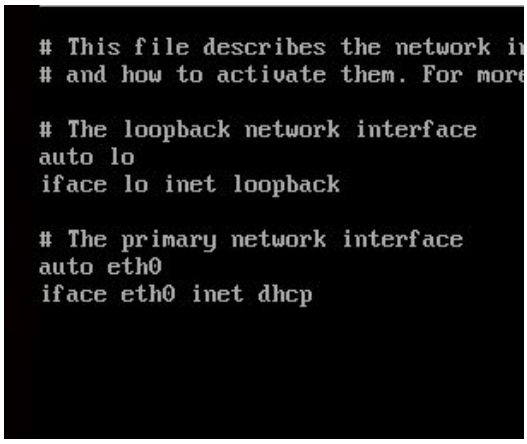
This means that the IP address of the management server should be 192.168.1.42. If you need to change the IP address or its associated hostname, you can assign this IP address to your management server manually by executing the following command:

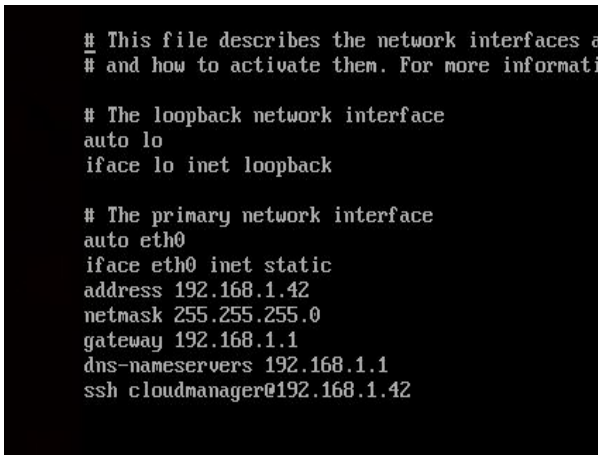
```
sudo ifconfig eth0 192.168.1.42
```

If you choose to use a different IP or a domain name you don't need to execute the above command. We will make changes to the source as we go along to use your choice of IP or domain name. Just make a note of your automatically assigned IP or domain name before we proceed.

Troubleshooting Networking Issues

Sometimes setting the eth0 interface to a static IP through "ifconfig" can lead to a loss of networking capabilities later on in the setup process. In that case, assign the server with a static IP by editing the /etc/network/interfaces file. To do so, change the interfaces file so it resembles what is shown below. Replace the "192.168.1.42" address with the desired static IP.





Before and after static IP fixes

Get the latest source code

When you first boot the VM, you will be prompted to login. The credentials are as follows:

- Username: cloudmanager
- Password: cs8674-cloudmanager

The management server hosts a web application for the user to select the cloud configuration and options to deploy on client systems. It also contains shell scripts to automate the deployment.

Now, we want to make sure that you have all the latest source code for the web application and the shell script. So navigate to your “home” directory:

```
cd /home/cloudmanager
```

You should see something similar to this:

The folder name “Cloud-Deployment-Utility” contains all the source code for the web application. It is a clone of the git repository https://github.com/sanket28/Cloud-Deployment-Utility .

Note: As of the current version of Hyperdrive, these files come pre-installed.

Execute `cd Cloud-Deployment-Utility/` to go inside the directory and execute the following commands:

```
git config --global user.email "YOUR_EMAIL_ADDRESS"
git stash
git pull
```

Replace YOUR_EMAIL_ADDRESS with your own email address. The above commands will pull all the latest code from the repository to your management server.

As mentioned in the Introduction section, if you chose to use a different IP address than 192.168.1.42, you will need to change the MQTT client configuration in the web application. Execute the following commands:

```
Cd
/home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/manager/templates/manager/
nano status.html
```

Scroll down till you see the following block of code:

```
<script>
        // This block of code connects to the MQTT server running on the management server. It is subscribed to the
        // 'cs8674/InstallStatus' topic. The shell script will send diagonstic messages to this topic which will
        // populate on the web page
        var hostname = "192.168.1.42"; // MQTT server address
        var port = 9001;
        var clientid = "cs8674-status";
        var receivedMessage = "";
        var client = new Paho.MQTT.Client(hostname, port, clientid); // MQTT client object
```

Replace the hostname with your IP address or domain name. Hit Ctrl + O to write to file and then Ctrl + X to exit.

The Management Server also contains a Debian Live Linux boot and configuration files. This Live Linux distribution is booted on the client machine during deployment and it executes a shell script to fetch the cloud configuration, perform disk partitioning and restoring the cloud images to the hard drive. You will also need to update this shell script with the latest code from the github repository.

Go to your home directory: `cd /home/cloudmanager`

```
cloudmanager@ubuntu:~$ cd /home/cloudmanager
cloudmanager@ubuntu:~$ ls
certs  cloud_configuration.txt  Cloud-Deployment-Utility  deployment_log  full-mbr.bin  livework  mbr.bin  mosquitto-1.4.2  reboot.txt
cloudmanager@ubuntu:~$
```

Now we are going to mount(sort of) the Debian Live filesystem on our management server to make some changes. So it will be similar to as if you were actually executing commands inside the Debian Live OS. To mount the Debian Live filesystem execute the following commands:

```
cd livework/
sudo chroot chroot (It will ask you for the password: Enter cs8674-cloudmanager)
mount none -t proc /proc
mount none -t sysfs /sys
mount none -t devpts /dev/pts
export HOME=/root
export LC_ALL=C
export PS1="\e[01;31m(live):\W \$ \e[00m"
```

After executing the above commands, you will enter the Debian Live Shell:

```
cloudmanager@ubuntu:~/livework$ sudo chroot chroot
root@ubuntu:/# mount none -t proc /proc
root@ubuntu:/# mount none -t sysfs /sys
root@ubuntu:/# mount none -t devpts /dev/pts
root@ubuntu:/# export HOME=/root
root@ubuntu:/# export LC_ALL=C
root@ubuntu:/# export PS1="\e[01;31m(live):\W \$ \e[00m"
(live):/ $
```

Now execute the following commands:

```
cd /home/cloudmanager
cd Cloud-Deployment-Utility/
git config --global user.email "YOUR_EMAIL_ADDRESS"
git stash
git pull
cd CS8674-Shell-Scripts/
chmod +x install_hypervisor.sh
cd ..
cd mqtt/
```

```
cp mqttcli /home/cloudmanager/mqttclient/
chmod +x /home/cloudmanager/mqttclient/mqttcli
```

Replace YOUR_EMAIL_ADDRESS with your own email. This will pull down the latest code for the shell script from the github repository. The name of the shell script is “install_hypervisor.sh”. The shell script executes when the Debian Live OS boots up on the client machine.

As discussed in the Introduction section, if you want to use an IP or domain name other than 192.168.1.42, you will need to update the shell script. Open the shell script using nano and look for the **Management_Server_IP** variable and change its value to your IP or domain name.

The shell script execution is indicated in ***/etc/rc.local***:

```
(live):cloudmanager $ cd /etc/
(live):etc $ cat rc.local
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
/home/cloudmanager/Cloud-Deployment-Utility/CS8674-Shell-Scripts/install_hypervisor.sh
exit 0
(live):etc $
```

Now, in the above commands we copied a MQTT client executable to the **/home/cloudmanager/mqttclient** directory. This client is configured to talk to a MQTT server running on 192.168.1.42. So, if you have chosen to use a different IP or domain name for the management server, you will need to update its configuration file. Execute the following commands to change the MQTT client configuration:

```
cd /home/cloudmanager/mqttclient
nano server.json
```

The server.json file will contain the following JSON Object:

```
{“host”:”192.168.1.42”, “port”:1883}
```

Change the host IP address to the IP address or domain name of your choice and the hit Ctrl + O to write and Ctrl + X to exit.

This concludes the changes to the Debian Live OS. Execute the following commands to exit from the Debian Live System:

```
umount /proc /sys /dev/pts
exit
sudo mksquashfs chroot binary/live/filesystem.squashfs -comp xz -e boot
sudo cp binary/live/filesystem.squashfs /usr/share/nginx/html/debian/
```

The “mksquashfs” command packages the Debian Live filesystem into a single file called filesystem.squashfs. The directory /usr/share/nginx/html/debian contains all the necessary files to boot the Debian Live OS on the client machines.

Now you need to reboot the system: Run `sudo reboot`

After reboot, you need to start the web server to host the web application. Run the following commands to start the web server:

```
sudo service nginx start
```

It will ask you for a PEM passphrase. Enter **cs8674**

You can check the status by running `sudo service nginx status`

If the web server is running, go to a web browser and enter http://DomainName_OR_IP and replace DomainName_OR_IP with the IP address or domain name of the management server

DHCP Server Setup

The DHCP server hands out IP addresses to client machines and gives them the location of all the necessary files required for cloud deployment. As mentioned in the Introduction section, the client machines boot using PXE. As PXE, is not secure and is a legacy standard, we use a more secure and updated open source version of it know as iPXE. When the client machines boot using PXE, they communicate with the DHCP server to get the location of the iPXE binary.

After fetching the iPXE binary, iPXE boots up and again requests the DHCP server for the location of the Debian Live OS boot files and then finally the Debian Live OS boots up.

Check if the DHCP server is running: `sudo service isc-dhcp-server status`
If its not running start it by running: `sudo service isc-dhcp-server start`

The location of the DHCP configuration file is **/etc/dhcp/dhcpd.conf.** If you have chosen to use a different IP address than 192.168.1.42, you will need to edit the dhcp configuration file. Execute the following commands to edit the configuration file:

`sudo nano /etc/dhcp/dhcpd.conf`
Scroll down till you see the follow block of code:

```
# A slightly different configuration for an internal subnet.
subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.200 192.168.1.240;
   option domain-name-servers 192.168.1.1;
#  option domain-name "internal.example.org";
   option routers 192.168.1.1;
   option subnet-mask 255.255.255.0;
   option broadcast-address 192.168.1.255;
#  default-lease-time 600;
#  default-lease-time 21600;
   max-lease-time 43200;
   max-lease-time 7200;
   next-server 192.168.1.42;
#if option arch = 00:06 {
#      filename "http://192.168.1.10/nbi_img/bootia32.efi";
#} else if option arch = 00:07 {
#       filename "http://192.168.1.10/nbi_img/bootx64.efi";
if exists user-class and option user-class = "iPXE" {
        filename "https://192.168.1.42/debian/debian.ipxe";
}  else {
      filename "undionly.kpxe";
  }
```

The if...else clause in the code above code checks whether the client machine has iPXE. If it does not, then it gives the location of the iPXE binary “undionly.kpxe” to the client. The file “undionly.kpxe” is transferred using TFTP to the client (TFTP Server setup will be discussed in the TFTP Server Setup section). After the client machine installs iPXE, it again communicates with the DHCP server and this time the first condition in the if else clause is true since the client already has iPXE installed. So, the DHCP server gives the debian boot script location to the client.

Replace the HTTP URL address with the IP address or domain name as per your requirement. Hit Ctrl + O to write to file and then Ctrl + X to exit. You can also change the subnet, range, routers and dns server according to your network configuration.

Now restart the DHCP server by running: `sudo service isc-dhcp-server restart`

Custom Script for live booting Debian

The Debian Live boot script contains all the commands to make the OS boot on the client machines. It is a script that iPXE understands. We need to make a few changes to this script so that we can use the security features of iPXE like code signing. The instructions for implementing the security features by compiling your own binary has been given in a later section of this document.

Execute the following commands:

```
sudo nano /usr/share/nginx/html/debian/debian.ipxe
```

Uncomment a few lines in the script to make it look like the following:

```
#!ipxe
imgtrust
kernel https://192.168.1.42/debian/vmlinuz
imgstat
imgverify vmlinuz https://192.168.1.42/debian/vmlinuz.sig
imgstat
module https://192.168.1.42/debian/initrd.img
imgargs vmlinuz boot=live config username=live fetch=http://192.168.1.42/debian/filesystem.squashfs
boot
```

If you have chosen to use a different IP or domain name than 192.168.1.42, you can replace it. Then hit Ctrl + O and then Ctrl + X.

This script fetches the linux kernel “vmlinuz” from the web server. It also fetches the file “vmlinuz.sig” from the web server. “vmlinuz.sig” is a signature file of the linux kernel. The script then verifies the linux kernel with its signature. If they match, it proceeds further to download the remaining files from the web server to boot the Debian Live OS.

Now we need to copy the latest signature file to the web server directory. Execute the following commands:

```
cd /home/cloudmanager/Cloud-Deployment-Utility/signatures
sudo cp vmlinuz.sig /usr/share/nginx/html/debian/
```

TFTP Server Setup

The TFTP server hands out the iPXE binary to clients that don’t have iPXE installed. The iPXE binary is stored at **/var/lib/tftpboot**. Now, we will copy the latest iPXE binary to that location. Execute the following commands:

```
cd /home/cloudmanager/Cloud-Deployment-Utility/ipxe_final
sudo cp undionly.kpxe /var/lib/tftpboot
```

Now we need to restart the TFTP server:

```
sudo service tftpd-hpa restart
sudo service tftpd-hpa status
```

Deploying your first Cloud

If you have already gone through the process of setting up each of the servers manually, it's time to test if everything works as it should. As discussed earlier, the shell script communicates with the web application using MQTT. The ‘deployCloud’ tool simply checks that each of the required servers are running and then hands off the deployment process to mosquitto. If you’ve gone through the previous steps, the deployCloud command is unnecessary.

Start the MQTT Server on the management server by executing the following command:

```
mosquitto -c /etc/mosquitto/mosquitto.conf
```

Now, fire up the web application in your browser and select the desired configuration and click the “Configure” button. On the next page, it will tell you to boot your client machines using PXE. Wait for the client to pop up on the grey box after boot up and then hit the “Deploy” button next to the client. To boot a local copy of the cloud, deploy a separate client VM through Virtualbox. Make sure that the client vm has at least 1GB of RAM as well as at least 8GB of disk memory. When deploying a local client VM, use the “bridged” auto detect setting to ensure that the client connects to the server.You will see the deployment status in the black box next to it. If everything goes well, you will have deployed your first cloud environment over the network fully automated!

Troubleshooting Deployment

SSL Issues:

If you run into issues preventing the client from interacting with the management server, it may be due to misconfigured or expired SSL certificates. If that occurs, you may need to reissue the SSL certificate and then recompile the iPXE binary. Instructions for this can be found within the “Creating a custom iPXE binary” section.

DHCP Issues:

Another common issue that can occur is if the router refuses to connect the management server to live hardware. If you are able to deploy a cloud locally, but are failing to deploy it on a server within a subnet, the issue may be DHCP related. Many routers enforce that they are the only DHCP provider on the subnet and may refuse to connect Hyperdrive's DHCP server with any other devices on the network. If this occurs, access the router configuration page and allow for external DHCP providers.

Creating a custom iPXE binary

iPXE is an open source firmware with full support for Pre-boot Execution (PXE) environment implementation with some additional features. The reason for using iPXE in this project are the security features that can be implemented using iPXE. This section covers the steps that you can follow in order to create a custom iPXE binary which supports code signing, HTTPS protocol and client authentication. Please note that the commands have been directly quoted from the cryptography section of the iPXE website. If you want to read more details about the commands, please refer to the the link : http://ipxe.org/crypto . All commands stated in this section use the openssl utility. Please make sure that you have it installed on your system. If you have any doubts, please refer the openssl documentation for more information. For the sake of convenience, we recommend that you have all the files generated via the openssl utility in one folder. All the commands given henceforth should be executed within the same directory. Otherwise, please remember to give the full path of the folder while specifying any file names.

Creating your own private root certificate

To create your own private root certificate and the corresponding private key, issue the following command:

```
mkdir signed          # Create a new directory which will contain all files

cd signed        # enter the directory


openssl req -x509 -newkey rsa:2048 -out ca.crt -keyout ca.key -days 1000  #Generate a private key and a private root certificate
```

After entering commands to generate private keys, you are asked to give some information. It's important to note that the 'common name' section of the ssl certificate is very important. iPXE checks the common name of the certificate against the IP of the web server. So it's very important that the 'common name' section be the IP address of the nginx webserver. You can fill anything in the other sections but please be consistent across certificates and to remember the passphrase that is used to protect the key.

Creating your own Certificate Authority(CA) infrastructure

First, we create few files that will be used by the CA infrastructure. TO create those files, enter the following commands:

Make sure that you are in the signed directory before entering the following commands

```
echo 01 > ca.srl
```

```
touch ca.idx
```

```
touch ca.cnf
```

Edit the ca.cnf file to contain the following information:

```
   [ ca ]
  default_ca                = ca_default

  [ ca_default ]
  certificate               = ca.crt
  private_key               = ca.key
  serial                    = ca.srl
  database                  = ca.idx
  new_certs_dir             = /home/cloudmanager/signed  # Please enter
the full path of the directory here
  default_md                = default
  policy                    = policy_anything
  preserve                  = yes
  default_days              = 90
  unique_subject            = no

  [ policy_anything ]
  countryName               = optional
  stateOrProvinceName       = optional
  localityName              = optional
  organizationName          = optional
  organizationalUnitName    = optional
  commonName                = optional
  emailAddress              = optional

  [ cross ]
  basicConstraints          = critical,CA:true
  keyUsage                  = critical,cRLSign,keyCertSign
```

```
  [ codesigning ]
  keyUsage                = digitalSignature
  extendedKeyUsage        = codeSigning
```

Creating a server certificate for the nginx web server

Enter the following commands to create the server certificate and the corresponding key:

```
openssl req -newkey rsa -keyout server.key -out server.req
```

```
openssl ca -config ca.cnf -in server.req -out server.crt
```

Now, we concatenate the root certificate and the server certificate we generated to create a full certificate chain.

```
cat server.crt ca.crt > server-full.crt
```

This certificate (server-full.crt) will be used on the nginx web server. It should be stored on the physical machine running the nginx instance and the path to the certificate and the key should be given using the “ssl_certificate” and the “ssl_certificate_key” parameter in the configuration file. The default configuration file of the nginx web server is “ /etc/nginx/sites-available/default ”. In the virtual machine provided by us, we have created our own configuration file with the name ‘cloudmanager’. This is the file that is currently deployed. The contents of the configuration file are as follows:

```
upstream cloudmanager_app_server {

  # fail_timeout=0 means we always retry an upstream even if it failed

  # to return a good HTTP response (in case the Unicorn master nukes a

  # single worker for timing out).
```

```
server
unix:///home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/run/gu
nicorn.sock fail_timeout=0;

}

server {

    listen   80;

    server_name cloudmanager.cs8674.com;

    listen 443 ssl;

    root /usr/share/nginx/html;



    # Make site accessible from http://localhost/

    #server_name cloudmanager.cs8674.com;

    ssl_certificate /etc/nginx/ssl/server.crt;

    ssl_certificate_key /etc/nginx/ssl/server.key;

    client_max_body_size 4G;

   access_log
/home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/logs/nginx-ac
cess.log;

    error_log
/home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/logs/nginx-er
ror.log;

    location /static/ {

        alias
/home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/manager/stati
c/;

    }
```

```
    location / {
        # an HTTP header important enough to have its own Wikipedia
entry:
        #   http://en.wikipedia.org/wiki/X-Forwarded-For
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # enable this if and only if you use HTTPS, this helps Rack
        # set the proper protocol for doing redirects:
        # proxy_set_header X-Forwarded-Proto https;

        # pass the Host: header from the client right along so
redirects
        # can be set properly within the Rack application
        proxy_set_header Host $http_host;

        # we don't want nginx trying to do something clever with
        # redirects, we set the Host: header above already.
        proxy_redirect off;

        # set "proxy_buffering off" *only* for Rainbows! when doing
        # Comet/long-poll stuff.  It's also safe to set if you're
        # using only serving fast clients with Unicorn + nginx.
        # Otherwise you _want_ nginx to buffer responses to slow
```

```
        # clients, really.
        # proxy_buffering off;

        # Try to serve static files from nginx, no point in making an
        # *application* server like Unicorn/Rainbows! serve static
files.
        if (!-f $request_filename) {
            proxy_pass http://cloudmanager_app_server;
            break;
        }
    }

    # Error pages
    error_page 500 502 503 504 /500.html;
    location = /500.html {
        root
/home/cloudmanager/Cloud-Deployment-Utility/cloudmanager/manager/stati
c/;
    }
}
```

Creating a signature of the linux kernel binary for implementing code signing

Please enter the following commands to create a digital signature of a binary named as “vmlinuz”.

```
openssl req -newkey rsa -keyout codesign.key -out codesign.req
```

```
openssl ca -config ca.cnf -extensions codesigning -in codesign.req -out codesign.crt
```

```
openssl cms -sign -binary -noattr -in vmlinuz -signer codesign.crt -inkey codesign.key -certfile ca.crt -outform DER -purpose any -out vmlinuz.sig
```

Now, the signature and the linux kernel binary are stored in the “/usr/share/nginx/html/debian” folder on the nginx web server.

Custom iPXE script

A custom ipxe script file is needed in order to implement code signing and HTTPS. This iPXE script is stored in the /usr/share/nginx/html/debian path on the nginx web server. Once the iPXE binary is loaded, it runs this script and then boots the live debian instance while providing confidentiality, authenticity and integrity.. The contents of our custom iPXE script are as follows:

```
#!ipxe

imgtrust

kernel https://192.168.1.42/debian/vmlinuz

imgstat

imgverify vmlinuz https://192.168.1.42/debian/vmlinuz.sig

imgstat

module https://192.168.1.42/debian/initrd.img

imgargs vmlinuz boot=live config username=live
fetch=http://192.168.1.42/debian/filesystem.squashfs

boot
```

Generating certificate for client authentication with the web server

Client authentication can provide additional security to the application. If the iPXE binary is having required client certificate and the corresponding private key then only it will boot, otherwise not. To generate client certificate, please navigate to the directory containing the CA files and execute the following commands:

```
openssl req -newkey rsa -keyout client.key -out client.req

 openssl ca -config ca.cnf -in client.req -out client.crt
```

Command for binary compilation

As we have implemented code signing, HTTPS support and client authentication in the custom iPXE binary, the following steps can be followed to compile the binary.

1. Clone the iPXE github repository using this command:

Git clone https://github.com/ipxe/ipxe.git

2. Navigate to the src/config directory and edit the “general.h” file by enabling the DOWNLOAD_PROTO_HTTPS and the IMAGE_TRUST_CMD option either by uncommenting the option or changing #undef to #define.

3. Navigate back to the ipxe directory and enter the following command :

```
make bin/undionly.kpxe CERT=cert1.crt,cert2.crt PRIVKEY=client.key
TRUST=cert1.crt DEBUG=x509
```

Where

cert1.crt is the name of the private root certificate (with full path)

cert2.crt is the client certificate

CERT options specifies the certificates to be embedded

DEBUG options will help you to resolve any certificate related issues

TRUST option makes the ipxe binary trust the private root certificate.

PRIVKEY options specifies the private key corresponding to the client certificate to be embedded

client.key is the private key the private key corresponding to the client certificate to be embedded (with full path)

If you chose to keep the current directory structure and store all certificates within ‘signed’ in the home directory, then the resulting compilation command should look like:

make bin/undionly.kpxe
CERT=/home/cloudmanager/signed/ca.crt,/home/cloudmanager/signed/client.crt

PRIVKEY=/home/cloudmanager/signed/client.key TRUST=/home/cloudmanager/signed/ca.crt DEBUG=x509

After you execute the command, you will be prompted for passphrase of the key several times. If you enter the correct passphrase, you will find the final compiled binary named “undionly.kpxe” in the bin folder. You may omit some options depending upon which security features you don’t want to implement. Please note that in the server configuration of the nginx web server in the virtual machine, we have disabled the client authentication option.