C4 M2 L6 Qwiklab: Getting Familiar with DNS and DHCP

1 hour 1 Credit

Rate Lab

Introduction

Managing DNS and DHCP are very common tasks for system administrators. In large enterprises, this can be done using complex software and might even be done by separate teams. In smaller networks, you might be the only person behind all networking services. In a case like this, using a simple solution like dnsmasq might be appropriate

In this lab, you'll modify existing dnsmasq configuration to adapt it to the DNS and DHCP needs of your organization. Then you will proceed to verify that it works as expected.

Head's up: Make sure to click the **"Start Lab"** button at the top of the screen. It may take a while for the lab to load. Please wait until the lab is running. To mark this lab as completed, make sure to click **"End Lab"** when you're done!

You'll have 60 minutes to complete this lab.

Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

Note: For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:

Accessing the virtual machine

Please find one of the three relevant options below based on your device's operating system.

Note: Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you'll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM's External IP address to connect.

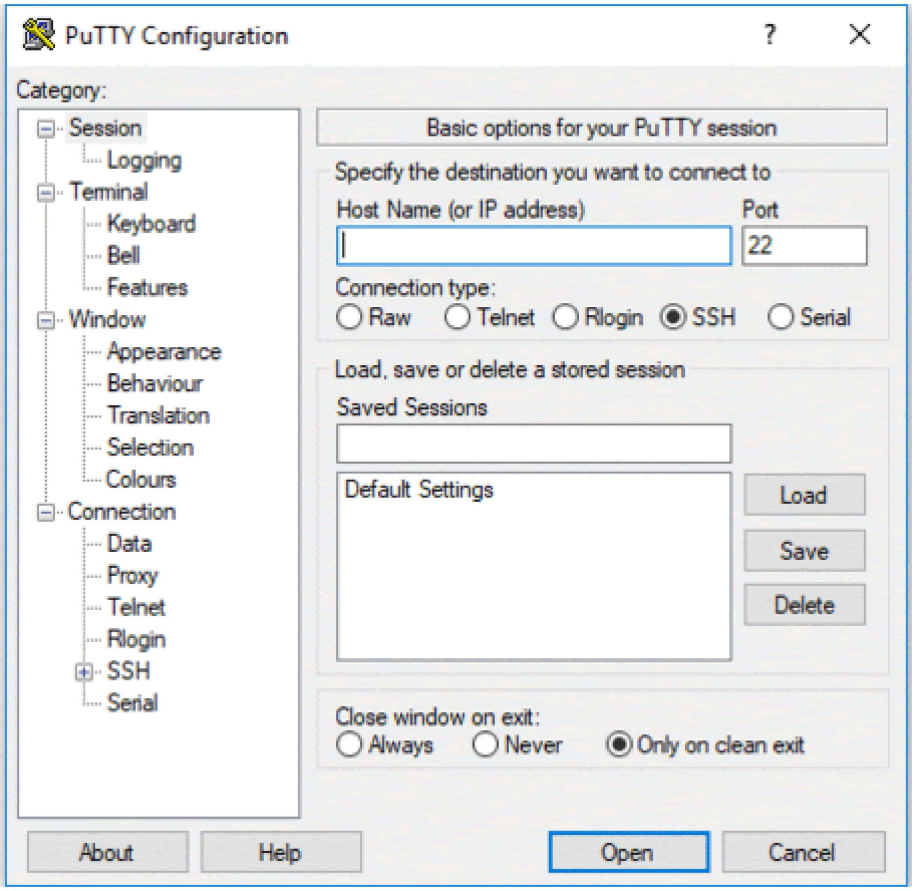
Download your PPK key file

You can download the VM's private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.

Connect to your VM using SSH and PuTTY

1. You can download Putty from here
2. In the **Host Name (or IP address)** box, enter username@external_ip_address.

Note: Replace **username** and **external_ip_address** with values provided in the lab.



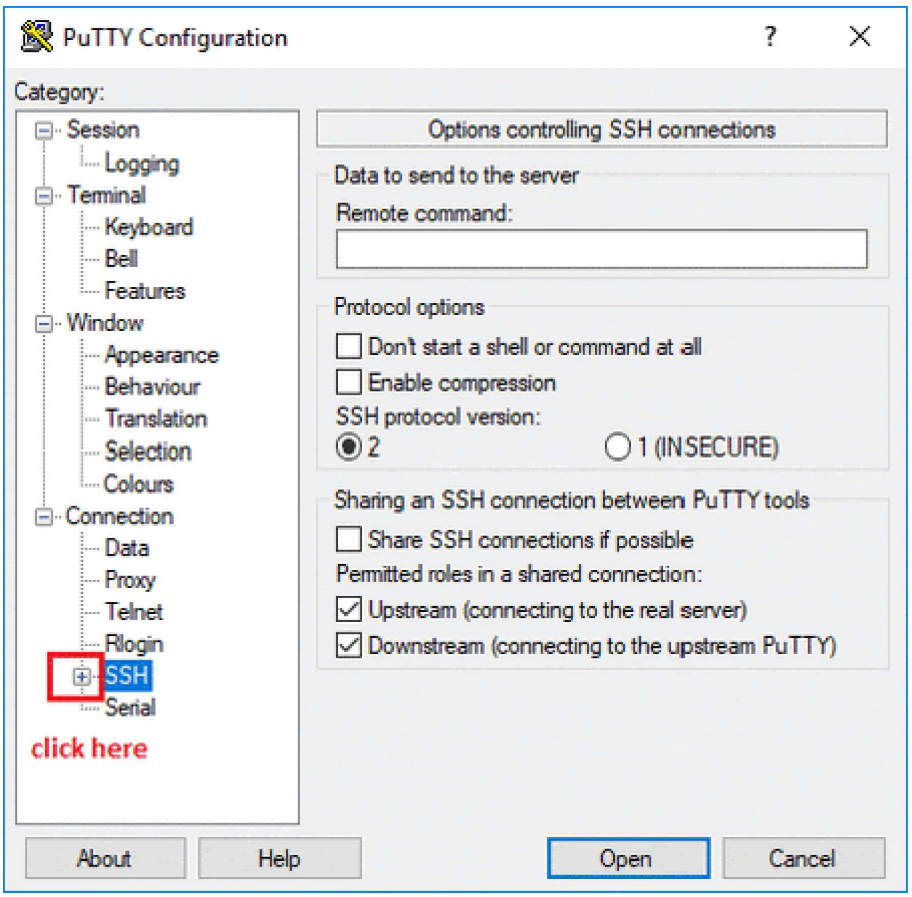
3. In the **Category** list, expand **SSH**.

4. Click **Auth** (don't expand it).

5. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.

6. Click on the **Open** button.

Note: PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



7. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

Common issues

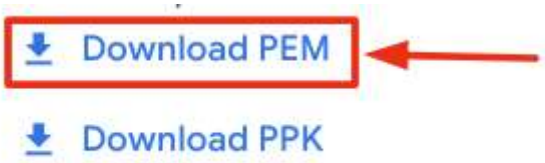
If PuTTY fails to connect to your Linux VM, verify that:

- You entered **<username>@<external ip address>** in PuTTY.
- You downloaded the fresh new PPK file for this lab from Qwiklabs.
- You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

Download your VM's private key file.

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



Connect to the VM using the local Terminal application

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
    - To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
    - To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

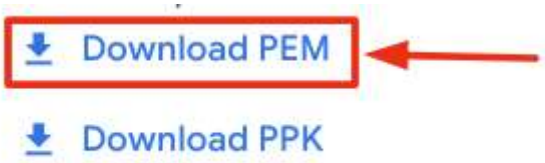
Note: Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.
You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

```
chmod 600 ~/Downloads/qwikLABS-XXXXX.pem
ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address
```

Option 3: Chrome OS users: Connecting to your VM via SSH

Note: Make sure you are not in **Incognito/Private mode** while launching the application.
Download your VM's private key file.

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



Connect to your VM

1. Add Secure Shell from here to your Chrome browser.

2. Open the Secure Shell app and click on **[New Connection]**.

3. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.

4. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

Note: If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

5. Once your key is uploaded, click on the **[ENTER] Connect** button below.

6. For any prompts, type **yes** to continue.

7. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

Go ahead and connect to **linux-instance** by following the instructions given in the section `Accessing the virtual machine`. Click on `Accessing the virtual machine` from the navigation pane on the right side.

Linux commands reminder

In this lab, we'll use a number of Linux commands that were explained during Course 3. This is a reminder of what these commands do:

- `sudo <command>`: executes a command with administrator rights
- `ls <directory>`: lists the files in a directory
- `cat <file>`: prints the whole contents of a file
- `nano <file>`: opens a text editor to edit the file

We will also be using the `service` command, shown in a previous lab, and we will present a number of new commands like `ip`, `dig` and `dhclient`. We will briefly explain what these commands do when they are shown. Remember that you can always read the manual page using `man <command_name>` to learn more about a command.

While you can copy and paste the commands presented in this lab, we recommend typing them out manually, to help with understanding and remembering these commands.

The scenario

The company that you are working for has dnsmasq set up to manage the DNS and DHCP needs of the network.

Currently, almost all of the DHCP range is being used to serve dynamic IPs. A number of servers will be added to the network and those should be configured

with known IPs.

Your task in this lab is to modify the configuration of this dnsmasq setup so that those servers always have the same IPs. To do this, not only will you need to give those servers the requested IPs, but also reduce the range for the dynamic IPs.

Network setup

Because you follow the rule of never testing in production, you will experiment with the necessary changes in a machine that is simulating the network. You should do this instead of trying these commands out in the actual DNS server.

In real life, after you were done experimenting you would then apply the same changes that you tried in the test instance to the production instance that is running dnsmasq on the network.

Let's have a look at this simulated network configuration.

For our test instance, we have configured a virtual network interface (called `eth_srv`) which the DNS and DHCP server will listen on. We can see the state of that interface with the `ip` command. The following will show information about the network configuration:

```
ip address show eth_srv
```

```
gcpstaging23816_student@linux-instance:~$ ip address show eth_srv
4: eth_srv@eth_cli: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether be:9c:7e:71:31:e7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 scope global eth_srv
       valid_lft forever preferred_lft forever
    inet6 fe80::bc9c:7eff:fe71:31e7/64 scope link
       valid_lft forever preferred_lft forever
```

We can see that this interface is configured to have the `192.168.1.1` IPv4 address in a network with a /24 or 255.255.255.0 netmask.

Additionally, we have another virtual network interface that we will use to simulate a client talking to the server and requesting DNS or DHCP traffic. This interface is called `eth_cli` and we can see the state using an equivalent command as the one mentioned above:

```
ip address show eth_cli
```

```
gcpstaging23816_student@linux-instance:~$ ip address show eth_cli
3: eth_cli@eth_srv: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether fe:cf:56:0d:a7:5d brd ff:ff:ff:ff:ff:ff
    inet6 fe80::fccf:56ff:fe0d:a75d/64 scope link
       valid_lft forever preferred_lft forever
gcpstaging23816_student@linux-instance:~$
```

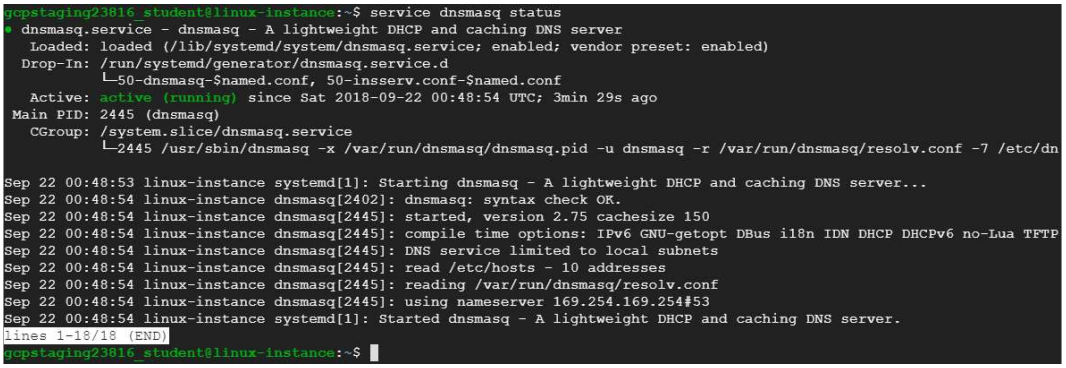
In this case we can see that the interface does not have an IPv4 address (yet).

Seeing debug output

When experimenting with changes in a service, it's a good idea to enable debug output so that we can understand what's going on and why.

Currently, dnsmasq is running as a daemon in the background. We can query the status using the service command learned in the previous lesson:

```
sudo service dnsmasq status
```



We can see that the service is running and that the last lines were generated by it.

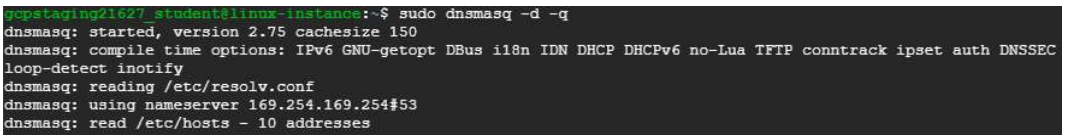
Note: If you get a prompt of the format `lines 1-18/18 (END)`, this means you can scroll to the right to see any additional output that is currently not visible. You can quit this view by pressing `q`.

In order to see the debug output we are going to stop the running service and start it manually as a foreground process. To do that, first stop it:

```
sudo service dnsmasq stop
```

Then start it manually, with debug flags:

```
sudo dnsmasq -d -q
```



If you are curious about all those parameters, the `-d` flag means "no daemon," which means run in foreground instead of background; the `-q` flag means "log queries," which means show the interactions with the clients. Since it's running in the foreground, you won't be able to run other commands in the terminal until you stop it manually by pressing "Ctrl+C". Don't stop it now though, since we'll need it to stay running for the next step.

Experimenting with DNS queries

In order to check that this is working properly, we will need to have a **second connection** to the machine. We will use it to simulate the client while your first connection terminal is running as the server. To do this, follow the instructions given in the section `Accessing the virtual machine`. Click on `Accessing the virtual machine` from the navigation pane at the right side and connect to the machine for the second time.

In this second terminal, let's start by asking some simple DNS queries and seeing what dnsmasq does with them. To do this, we will use the command `dig`, which allows us to request the IP address of a certain hostname. Let's try getting the IP address of example.com:

Second terminal (client):

```
dig example.com @localhost
```

```
gcpstaging21627_student@linux-instance:~$ dig example.com @localhost

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.com @localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35782
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            21599   IN      A       93.184.216.34

;; Query time: 29 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Aug 23 21:43:54 UTC 2018
;; MSG SIZE  rcvd: 56
```

By using the `@localhost` parameter, we told `dig` that we wanted to use the running machine as the DNS server. In other words, we queried our running `dnsmasq` and we got a reply from it. The reply contains the IPv4 address for the domain we requested.

Switch back to the first connection terminal, where you can see what the running dnsmasq said about the query:

First terminal (server):

```
dnsmasq: query[A] example.com from 127.0.0.1
dnsmasq: forwarded example.com to 169.254.169.254
dnsmasq: reply example.com is 93.184.216.34
```

This is telling us that it forwarded the request to another DNS server. This is the normal behavior for a caching DNS service like dnsmasq. Let's see what happens if we make the same request again. Go back to your second terminal and run the same `dig` command. Hint: You can press the up-arrow to avoid having to type it again.

Second terminal (client):

```
dig example.com @localhost
```

```
gcpstaging21627_student@linux-instance:~$ dig example.com @localhost

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.com @localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35782
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            21599   IN      A       93.184.216.34

;; Query time: 29 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Aug 23 21:43:54 UTC 2018
;; MSG SIZE  rcvd: 56
```

We got the same reply. What did dnsmasq say?

First terminal (server):

```
dnsmasq: query[A] example.com from 127.0.0.1
dnsmasq: cached example.com is 93.184.216.34
```

Notice how instead of forwarding the query to the nameserver of the machine, it has used the value that it had already cached.

Let's see what happens if we request an IP address of a domain name that doesn't exist.

Second terminal (client):

```
dig example.local @localhost
```

```
gcpstaging21627_student@linux-instance:~$ dig example.local @localhost

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.local @localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 16777
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;example.local.                 IN      A

;; AUTHORITY SECTION:
.                       86399   IN      SOA     a.root-servers.net. nstld.verisign-grs.com. 2018082301 1800 900 604
800 86400

;; Query time: 31 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Aug 23 21:47:22 UTC 2018
;; MSG SIZE  rcvd: 117
```

The reply we get indicates that the name doesn't have an associated IP address. Going back to the running dnsmasq, we can see what it said.

First terminl (server):

```
dnsmasq: forwarded example.local to 169.254.169.254
dnsmasq: reply example.local is NXDOMAIN
```

Because dnsmasq didn't have any information associated with that domain, it forwarded the query to the configured DNS server. The reply was that the domain didn't exist.

You can keep experimenting, querying for other domain names that you are interested in. Once you are done, in the terminal where you are running dnsmasq, stop it by pressing "Ctrl-C."

Current configuration setup

As mentioned, your company already has dnsmasq deployed on the network. In this lab we are going to make a few changes to the existing config. Let's look first at the current config.

The current config is stored in `/etc/dnsmasq.d/mycompany.conf`. Let's check out its contents with the `cat` command:

First terminal (server):

```
cat /etc/dnsmasq.d/mycompany.conf
```

```
gcpstaging23816_student@linux-instance:~$ cat /etc/dnsmasq.d/mycompany.conf
# This is the interface on which the DHCP server will be listening to.
interface=eth_srv

# This tells this dnsmasq to only operate on that interface and not operate
# on any other interfaces, so that it doesn't interfere with other running
# dnsmasq processes.
bind-interfaces

# Domain name that will be sent to the DHCP clients
domain=mycompany.local

# Default gateway that will be sent to the DHCP clients
dhcp-option=option:router,192.168.1.1

# DNS servers to announce to the DHCP clients
dhcp-option=option:dns-server,192.168.1.1

# Dynamic range of IPs to use for DHCP and the lease time.
dhcp-range=192.168.1.2,192.168.1.254,24h
```

Explanation of the meanings of these settings:

- **interface** is the name of the interface that will be used for listening to DHCP requests and serving replies; we are setting it to our virtual eth_srv interface.
- **bind-interfaces** means that dnsmasq will operate on that interface only and ignore the others.
- **domain** is the domain name used on the network.
- **dhcp-option** allows us to give the DHCP clients optional additional information. In this case we are setting the **router** (also known as default gateway) and the **dns-server**. When the clients receive the DHCP reply they will also receive and apply this configuration.

- **dhcp-range** indicates the range of IPs that are available to be used for dynamic IP assignment as well as the length of the lease. In this case, the whole network (except for the DHCP server) is currently available as part of the dynamic range. We know that we need to change this. The lease is currently set to 24h, which might not be a great idea if our network has a lot of devices that are only visible for short periods of time.

Now that we've had a look at this config, let's start dnsmasq again, but this time telling it the configuration file that we want it to use:

First terminal (server):

```
sudo dnsmasq -d -q -C /etc/dnsmasq.d/mycompany.conf
```

```
gcpstaging21627_student@linux-instance:~$ sudo dnsmasq -d -q -C /etc/dnsmasq.d/mycompany.conf
dnsmasq: started, version 2.75 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua TFTP conntrack ipset auth DNSSEC
loop-detect inotify
dnsmasq-dhcp: DHCP, IP range 192.168.1.2 -- 192.168.1.254, lease time 1d
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface eth_srv
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 169.254.169.254#53
dnsmasq: read /etc/hosts - 10 addresses
```

This time when dnsmasq starts, we see that it's currently serving with the configuration parameters that are specified in the config file.

Experimenting with a DHCP client

Now that we know that DNS is working properly, let's experiment with the DHCP configuration. In the second terminal, we will run `dhclient`, which is the most common DHCP client on Linux. As mentioned, we will run this on the `eth_cli` interface. Additionally, we will tell it to run in verbose mode and run a debugging script that we provide:

Second terminal (client):

```
sudo dhclient -i eth_cli -v -sf /root/debug_dhcp.sh
```

```
gcpstaging23816_student@linux-instance:~$ sudo dhclient -i eth_cli -v -sf /root/debug_dhcp.sh
Internet Systems Consortium DHCP Client 4.3.3
Copyright 2004-2015 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth_cli/fe:cf:56:0d:a7:5d
Sending on   LPF/eth_cli/fe:cf:56:0d:a7:5d
Sending on   Socket/fallback
Created duid \000\001\000\001#8P\213\376\317V\015\247].
DHCPDISCOVER on eth_cli to 255.255.255.255 port 67 interval 3 (xid=0x38baa831)
DHCPDISCOVER on eth_cli to 255.255.255.255 port 67 interval 4 (xid=0x38baa831)
DHCPREQUEST of 192.168.1.77 on eth_cli to 255.255.255.255 port 67 (xid=0x31a8ba38)
DHCPOFFER of 192.168.1.77 from 192.168.1.1
DHCPACK of 192.168.1.77 from 192.168.1.1
Received variables:
  Hostname: linux-instance
  IP address: 192.168.1.77
  Network: 192.168.1.0
  Netmask: 255.255.255.0
  Router: 192.168.1.1
  Domain Name: mycompany.local
  DNS server: 192.168.1.1
bound to 192.168.1.77 -- renewal in 33660 seconds.
```

The debugging script that we are passing with the `-sf` parameter is so that instead of actually modifying the whole network settings in this machine, we can see what information was received from the server.

In the debugging output we see that the options set in the configuration file were correctly sent to the client.

If you go back to the terminal where dnsmasq is running, you will see these additional lines:

First terminal (server):

```
dnsmasq-dhcp: DHCPDISCOVER(eth_srv) 0a:73:86:7b:69:3f
dnsmasq-dhcp: DHCPOFFER(eth_srv) 192.168.1.12 0a:73:86:7b:69:3f
dnsmasq-dhcp: DHCPREQUEST(eth_srv) 192.168.1.12 0a:73:86:7b:69:3f
dnsmasq-dhcp: DHCPACK(eth_srv) 192.168.1.12 0a:73:86:7b:69:3f linux-instance
dnsmasq-dhcp: not giving name linux-instance to the DHCP lease of 192.168.1.12 because the name ex
ists in /etc/hosts with address 10.0.0.2
```

So, we see that the server saw the request and replied with an address.

We also see that it knows that the machine with the MAC address of `eth_cli` is called linux-instance. The domain of the network is set to `mycompany.local`. This means that we can query the IP address of `linux-instance.mycompany.local` and it should give us the address that was received via DHCP:

Second terminal (client):

```
dig linux-instance.mycompany.local @localhost
```

```
gcpstaging21631_student@linux-instance:~$ dig linux-instance.mycompany.local @localhost

; <<>> DiG 9.10.3-P4-Ubuntu <<>> linux-instance.mycompany.local @localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65279
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;linux-instance.mycompany.local.        IN      A

;; ANSWER SECTION:
linux-instance.mycompany.local. 0 IN    A       192.168.1.196

;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Aug 23 22:39:06 UTC 2018
;; MSG SIZE  rcvd: 75
```

When dnsmasq assigns the dynamic addresses it also keeps track of the names of the machines in the network. This is so if other machines request those names, dnsmasq will recognize the associated addresses.

Changing the configuration

Alright, so we've now seen what dnsmasq is doing and experimented obtaining an IP address from it. Our company's request to assign fixed IPs to the new server still needs to be fulfilled. This kind of assignment is done using the MAC address

of the network card. We need to know the MAC addresses of the servers and which IP addresses to assign to them:

- aa:bb:cc:dd:ee:b2 - 192.168.1.2
- aa:bb:cc:dd:ee:c3 - 192.168.1.3
- aa:bb:cc:dd:ee:d4 - 192.168.1.4

Additionally, we know that we will be setting up more servers in the future, so we want to leave the first 20 IP addresses outside of the dynamic range.

We will use the dhcp-host configuration option to assign a specific address to a machine. The format of this option is to list the MAC address of the machine, followed by a comma, and then the IP address being assigned to it. For example, for the first server in the list, it would look like this:

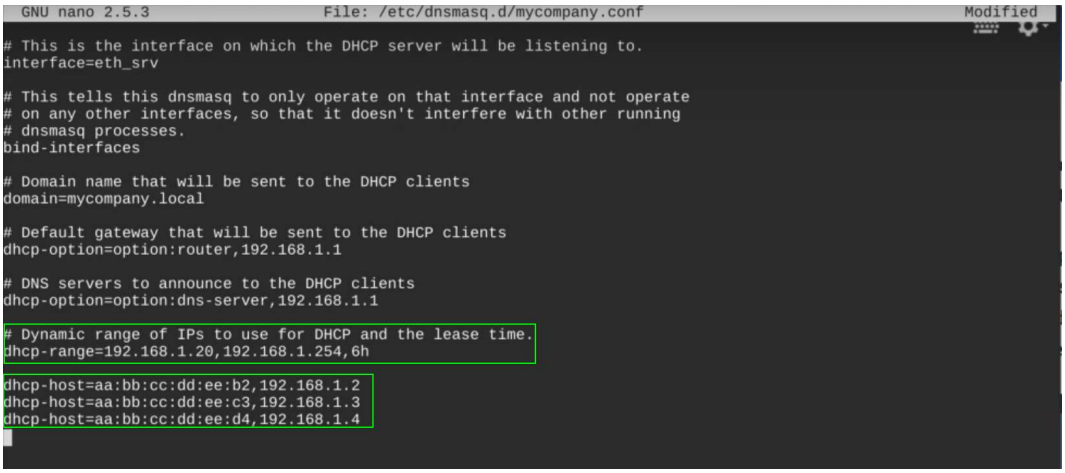
```
dhcp-host=aa:bb:cc:dd:ee:b2,192.168.1.2
dhcp-host=aa:bb:cc:dd:ee:c3,192.168.1.3
dhcp-host=aa:bb:cc:dd:ee:d4,192.168.1.4
```

To edit the configuration, stop the server (if you haven't already) with "Ctrl+C". Then let's open the configuration file using a text editor called *nano*.

```
sudo nano /etc/dnsmasq.d/mycompany.conf
```

In this file, we need to change the dhcp-range option and add dhcp-host lines:

- Change the dhcp-range option so that it starts with 192.168.1.20 instead of 192.168.1.2
- Change the 24h lease time to 6h
- At the end of this file, add three dhcp-host lines, one for each of the servers that need to be configured, as defined above.



Once you've done this, press "Ctrl-X" to exit the editor. It will ask you to save your changes. Press "Y" for yes and then enter at the filename prompt.

Click Check my progress to verify the objective.

Reduce Dynamic IP Range

We have done the change. Now we need to verify that it does what we intend it to do. The first step is to verify the syntax of the file. To do this, we can use a parameter to dnsmasq: `--test`

First terminal (server):

```
sudo dnsmasq --test -C /etc/dnsmasq.d/mycompany.conf
```

```
dnsmasq: syntax check OK.
```

If instead you get an error like: "`dnsmasq: bad hex constant at line 21 of /etc/dnsmasq.d/mycompany.conf`", it means that you have an error in the line indicated. Open the file with nano once again, and review and fix whatever is wrong in that line.

Click Check my progress to verify the objective.

New Servers have Fixed Addresses

Once we are sure that the configuration has the right syntax, we can start the service back up with the same command as before:

First terminal (server):

```
sudo dnsmasq -d -q -C /etc/dnsmasq.d/mycompany.conf
```

```
gcpstaging21631_student@linux-instance:~$ sudo dnsmasq -d -q -C /etc/dnsmasq.d/mycompany.conf
dnsmasq: started, version 2.75 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua TFTP conntrack
 ipset auth DNSSEC loop-detect inotify
dnsmasq-dhcp: DHCP, IP range 192.168.1.20 -- 192.168.1.254, lease time 6h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface eth_srv
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 169.254.169.254#53
dnsmasq: read /etc/hosts - 10 addresses
dnsmasq-dhcp: not giving name linux-instance to the DHCP lease of 192.168.1.196 because the na
me exists in /etc/hosts with address 10.0.0.2
```

We see it's now using the new range, starting from `192.168.1.20` and that the lease time is now 6h instead of 24h.

Now, we want to verify that the lines we added to set specific IP addresses are in effect. Given that we are using a test instance with a simulated network environment, we can test this change by setting the MAC address of our virtual network interface to match one of the addresses specified in the config file.

Important: You shouldn't change the MAC address of actual machines on the network, as this could lead to many networking problems that are hard to debug. This technique can be used here because it's a testing simulation.

Let's go back to our second terminal and change the MAC address of `eth_cli` by using the `ip link` command.

Second terminal (client):

```
sudo ip link set eth_cli address aa:bb:cc:dd:ee:c3
```

This command, as so many other Linux commands, doesn't generate any output if it succeeded. In order to see what it did, we need to look at the interface again, as we have done before.

Second terminal (client):

```
sudo ip address show eth_cli
```

```
gcpstaging21631_student@linux-instance:~$ sudo ip address show eth_cli
3: eth_cli@eth_srv: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether aa:bb:cc:dd:ee:c3 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8c93:63ff:fe8e:3951/64 scope link
       valid_lft forever preferred_lft forever
gcpstaging21631_student@linux-instance:~$
```

We see that the MAC address (shown in the link/ether line) is now set to the one that corresponds to one of the new servers. Let's see if dnsmasq gives us the right IP address.

Second terminal (client):

```
sudo dhclient -i eth_cli -v  -sf /root/debug_dhcp.sh
```

```
gcpstaging23816_student@linux-instance:~$ sudo dhclient -i eth_cli -v  -sf /root/debug_dhcp.sh
Internet Systems Consortium DHCP Client 4.3.3
Copyright 2004-2015 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth_cli/aa:bb:cc:dd:ee:c3
Sending on   LPF/eth_cli/aa:bb:cc:dd:ee:c3
Sending on   Socket/fallback
DHCPREQUEST of 192.168.1.77 on eth_cli to 255.255.255.255 port 67 (xid=0x2ec7640d)
DHCPREQUEST of 192.168.1.77 on eth_cli to 255.255.255.255 port 67 (xid=0x2ec7640d)
DHCPREQUEST of 192.168.1.77 on eth_cli to 255.255.255.255 port 67 (xid=0x2ec7640d)
DHCPDISCOVER on eth_cli to 255.255.255.255 port 67 interval 3 (xid=0x22709321)
DHCPREQUEST of 192.168.1.3 on eth_cli to 255.255.255.255 port 67 (xid=0x21937022)
DHCPOFFER of 192.168.1.3 from 192.168.1.1
DHCPACK of 192.168.1.3 from 192.168.1.1
Received variables:
  Hostname: linux-instance
  IP address: 192.168.1.3
  Network: 192.168.1.0
  Netmask: 255.255.255.0
  Router: 192.168.1.1
  Domain Name: mycompany.local
  DNS server: 192.168.1.1
bound to 192.168.1.3 -- renewal in 10737 seconds.
```

In this case, we see that as a first step the client tried to request the same IP address that it had before, (the `DHCPREQUEST` lines) but it got no reply. Eventually, it tried to get a new address, by doing a `DHCPDISCOVER` and for that it did get a reply, with the desired address.

What did dnsmasq say in the first connection terminal?

First terminal (server):

```
dnsmasq-dhcp: DHCPDISCOVER(eth_srv) 192.168.1.196 aa:bb:cc:dd:ee:c3
dnsmasq-dhcp: DHCPOFFER(eth_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3
dnsmasq-dhcp: DHCPREQUEST(eth_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3
dnsmasq-dhcp: DHCPACK(eth_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3 linux-instance
dnsmasq-dhcp: not giving name linux-instance to the DHCP lease of 192.168.1.3 because the name
 exists in /etc/hosts with address 10.0.0.2
```

We've made the requested change to the DHCP configuration and verified that it works as expected.

If you were doing this in real life, now is when you would copy the configuration file to the production server.

Conclusion

Congrats! You've successfully experimented with DNS queries and modified the configuration of an existing DHCP server.

This was not a simple task and you've put in practice all your service management knowledge as well as your networking knowledge to be able to achieve this. Awesome!

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.