# Practice M4: Network. Software. Services (Ubuntu)

\* NOTE: Most of the exercises included in this practice guide are not suitable for execution in WSL or Docker environment. It is recommended a virtual infrastructure to be used instead.

## Part 1: Network

### Preparation

For this part we will use an Ubuntu VM with two network adapters – one in **NAT** mode (or **Bridged** if you prefer), and another in **Internal Network** (set the name to **MyNet**). We can either reuse the machine used in previous practices or import / create new one.

For the next steps we will assume that we have imported a new machine and configured it according to the above requirements.

### General

No matter what distribution we use, there is a set of commands that are available. For example, for all distributions that adopted **systemd**, we can change the host's name with:

lsauser@ubuntu:~$ **hostnamectl**

Static hostname: ubuntu

...

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname jupiter.lsa.lab**

lsauser@ubuntu:~$

We can add also the so-called pretty name with:

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname --pretty 'Jupiter Server'**

lsauser@ubuntu:~$

Our prompt stays the same. Let's check some of the related configuration files:

lsauser@ubuntu:~$ **cat /etc/hostname**

jupiter.lsa.lab

lsauser@ubuntu:~$ **cat /etc/machine-info**

PRETTY\_HOSTNAME="Jupiter Server"

lsauser@ubuntu:~$

The second one may exist or may not exist

We must take care for one more thing in order to guarantee that our hostname will be preserved (during reboot):

lsauser@ubuntu:~$ **sudo vi /etc/cloud/cloud.cfg**

…

preserve\_hostname: true

…

lsauser@ubuntu:~$

In order the changes to be reflected in the prompt, we must **close the session**, and open a new one.

Now, that we are back in, let's ask for the network links:

lsauser@jupiter:~$ **ip link show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP mode DEFAULT group default qlen 1000

link/ether 08:00:27:68:e4:3a brd ff:ff:ff:ff:ff:ff

3: enp0s8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether 08:00:27:da:28:ad brd ff:ff:ff:ff:ff:ff

lsauser@jupiter:~$

We can use even shorter commands by skipping characters, as long as the given arguments can be translated into a valid combination. For example, the above command can be written like **ip l sh**, or even shorter, if we know that the default action is show, it can become **ip l**.

Now, we can bring our second interface up:

lsauser@jupiter:~$ **sudo ip link set dev enp0s8** **up**

We can see current IP addresses with:

lsauser@jupiter:~$ **ip address show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP group default qlen 1000

link/ether 08:00:27:68:e4:3a brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3

valid\_lft 86274sec preferred\_lft 86274sec

inet6 fe80::a00:27ff:fe68:e43a/64 scope link

valid\_lft forever preferred\_lft forever

3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP group default qlen 1000

link/ether 08:00:27:da:28:ad brd ff:ff:ff:ff:ff:ff

inet6 fe80::a00:27ff:feda:28ad/64 scope link

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~$

The same can be achieved with just **ip a**. As we can see, our **enp0s8** adapter does not have any address assigned. We can add one with:

lsauser@jupiter:~$ **sudo ip address add 192.168.200.1/24 dev enp0s8**

lsauser@jupiter:~$

lsauser@jupiter:~$ **ip address show enp0s8**

3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:2c:b7:e0 brd ff:ff:ff:ff:ff:ff

inet 192.168.200.1/24 scope global enp0s8

valid\_lft forever preferred\_lft forever

inet6 fe80::9bed:86ab:7f9d:1fc9/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~$

Now, let's try this:

lsauser@jupiter:~$ **ping -c 3 -q 192.168.200.1**

PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data.

--- 192.168.200.1 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 1998ms

rtt min/avg/max/mdev = 0.040/0.043/0.045/0.002 ms

lsauser@jupiter:~$

Without further actions, the changes will be lost on system restart. We don't want to make them permanent, at least not now, so let's tweak the last **ip** command to delete the address:

lsauser@jupiter:~$ **sudo ip address del 192.168.200.1/24 dev enp0s8**

lsauser@jupiter:~$

lsauser@jupiter:~$ **ip address show enp0s8**

3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:2c:b7:e0 brd ff:ff:ff:ff:ff:ff

lsauser@jupiter:~$

With the same tool we can manage the routing table:

lsauser@jupiter:~$ **ip route show**

default via 10.0.2.2 dev enp0s3 proto dhcp src 10.0.2.15 metric 100

10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15

10.0.2.2 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 100

lsauser@jupiter:~$

### NetworkD and Netplan

We can use the **networkctl** tool to query for network information:

lsauser@jupiter:~$ **networkctl list**

IDX LINK TYPE OPERATIONAL SETUP

1 lo loopback carrier unmanaged

2 enp0s3 ether routable configured

3 enp0s8 ether degraded unmanaged

3 links listed.

lsauser@jupiter:~$

If want to see general information about all managed interfaces, we can execute:

lsauser@jupiter:~$ **networkctl status**

● State: routable

Address: 10.0.2.15 on enp0s3

fe80::a00:27ff:fe68:e43a on enp0s3

fe80::a00:27ff:feda:28ad on enp0s8

Gateway: 10.0.2.2 on enp0s3

DNS: 192.168.1.5

lsauser@jupiter:~$

Alternatively, for a particular interface:

lsauser@jupiter:~$ **networkctl status enp0s8**

● 3: enp0s8

Link File: /lib/systemd/network/99-default.link

Network File: n/a

Type: ether

State: degraded (unmanaged)

Path: pci-0000:00:08.0

Driver: e1000

Vendor: Intel Corporation

Model: 82540EM Gigabit Ethernet Controller (PRO/1000 MT Desktop Adapter)

HW Address: 08:00:27:da:28:ad (PCS Systemtechnik GmbH)

Address: fe80::a00:27ff:feda:28ad

lsauser@jupiter:~$

Network settings can be controlled via Netplan YAML file in the **/etc/netplan** folder. Check the content of the folder and open the only file there in **vi**, add the second interface (**bold text**) and set it to **dhcp**:

lsauser@jupiter:~$ **sudo vi /etc/netplan/50-cloud-init.yaml**

...

network:

ethernets:

enp0s3:

dhcp4: true

**enp0s8:**

**dhcp4: true**

version: 2

...

lsauser@jupiter:~$ **sudo netplan --debug apply**

The file name may vary. For example, depending on the version of Ubuntu, yours may be named **00-installer-config.yaml**

Be very careful with the number of spaces (NOT tabs, but spaces). If in your file the indent is two spaces, then stick to it.

You may check again with **networkctl list**

If our second interface was configured in a network with DHCP server it would have received an IP address. This is not our case, at least not yet.

Let's open the file again, and set static address to the second network interface (we may omit the **nameservers** part):

lsauser@jupiter:~$ **sudo vi /etc/netplan/50-cloud-init.yaml**

...

**addresses: [192.168.200.1/24]**

**nameservers:**

**addresses: [8.8.8.8]**

**dhcp4: no**

...

lsauser@jupiter:~$ **sudo netplan apply**

lsauser@jupiter:~$ **ip a**

...

lsauser@ubuntu:~$ **networkctl status**

● State: routable

Address: 10.0.2.15 on enp0s3

192.168.200.1 on enp0s8

fe80::a00:27ff:fe68:e43a on enp0s3

fe80::a00:27ff:feda:28ad on enp0s8

Gateway: 10.0.2.2 on enp0s3

DNS: 192.168.1.5

8.8.8.8

lsauser@ubuntu:~$

We are ready to move on. We have one interface connected to the external world with dynamic IP, and another one set with static IP, that is connected to an internal network.

## Part 2: Software and Services

### DEB

Let's first install a local package. In order to achieve this, we must download it first. Let's execute:

lsauser@ubuntu:~$ **wget https://zahariev.pro/linux/hello-lsa/releases/hello-lsa-1.0\_amd64.deb**

...

lsauser@ubuntu:~$

List what we have so far:

lsauser@jupiter:~$ **ls -al hello\***

-rw-rw-r-- 1 lsauser lsauser 4108 Jan 11 2019 hello-lsa-1.0\_amd64.deb

lsauser@jupiter:~$

Now, let's install the packages:

lsauser@jupiter:~$ **sudo dpkg -i hello\***

...

lsauser@jupiter:~$

It seems that all went well. Let's test it:

lsauser@jupiter:~$ **hello-lsa**

Once we are done, we can remove it with:

lsauser@jupiter:~$ **sudo dpkg -r hello-lsa**

### APT

Before installing packages from remote repositories, we must refresh or update the information about them:

lsauser@jupiter:~$ **sudo apt update**

Hit:1 http://bg.archive.ubuntu.com/ubuntu disco InRelease

Hit:2 http://bg.archive.ubuntu.com/ubuntu disco-updates InRelease

Hit:3 http://bg.archive.ubuntu.com/ubuntu disco-backports InRelease

Hit:4 http://bg.archive.ubuntu.com/ubuntu disco-security InRelease

Reading package lists... Done

Building dependency tree

Reading state information... Done

59 packages can be upgraded. Run 'apt list --upgradable' to see them.

lsauser@jupiter:~$

Now, we will install something funny:

lsauser@jupiter:~$ **sudo apt install -y cowsay**

Reading package lists... Done

Building dependency tree

Reading state information... Done

Suggested packages:

filters cowsay-off

The following NEW packages will be installed:

cowsay

...

Processing triggers for man-db (2.8.5-2) ...

lsauser@jupiter:~$

Let's test our new piece of software:

lsauser@jupiter:~$ **cowsay -b Hello human!**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

< Hello human! >

--------------

\ ^\_\_^

\ (==)\\_\_\_\_\_\_\_

(\_\_)\ )\/\

||----w |

|| ||

lsauser@jupiter:~$

We can do few more experiments, and finally remove it by executing:

lsauser@jupiter:~$ **sudo apt purge -y cowsay**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages will be REMOVED:

cowsay\*

...

Processing triggers for man-db (2.8.5-2) ...

lsauser@jupiter:~$

We can upgrade the installed packages. Before doing it, we can check the list of upgradeable packages:

lsauser@jupiter:~$ **sudo apt list --upgradable**

Listing... Done

apparmor/disco-updates 2.13.2-9ubuntu6.1 amd64 [upgradable from: 2.13.2-9ubuntu6]

apt-utils/disco-updates 1.8.3 amd64 [upgradable from: 1.8.0]

...

lsauser@jupiter:~$

Now, we can issue the upgrade process:

lsauser@jupiter:~$ **sudo apt upgrade -y**

...

lsauser@jupiter:~$

If we need a package, which name, we don't know, we can search for it with:

lsauser@jupiter:~$ **apt search "web browser"**

...

w3m/cosmic 0.5.3-36build2 amd64

WWW browsable pager with excellent tables/frames support

...

lsauser@jupiter:~$ **sudo apt install -y w3m**

Reading package lists... Done

...

lsauser@jupiter:~$

Now, we can open a site of our choice.

We can activate other repositories, for example **Canonical Partners** repository. For this purpose, we must uncomment one or two lines in:

lsauser@jupiter:~$ **sudo vi /etc/apt/sources.list**

...

deb http://archive.canonical.com/ubuntu focal partner

...

lsauser@jupiter:~$ **sudo apt update**

...

Not there are that many packages here, but anyway. We can examine them with:

lsauser@jupiter:~$ **grep Package /var/lib/apt/lists/archive.canonical.com\_ubuntu\_dists\_focal\_partner\_binary-amd64\_Packages**

Package: adobe-flashplugin

Package: adobe-flash-properties-gtk

Package: adobe-flash-properties-kde

lsauser@jupiter:~$

There are other third party repositories as well. We can check this site: <https://www.ubuntuupdates.org/ppas>

Let's follow the procedure for installing the repository for **Node.js 12**:

lsauser@jupiter:~$ **curl -s https://deb.nodesource.com/gpgkey/nodesource.gpg.key | sudo apt-key add -**

OK

lsauser@jupiter:~$ **sudo sh -c "echo deb https://deb.nodesource.com/node\_12.x focal main \**

**> /etc/apt/sources.list.d/nodesource.list"**

lsauser@jupiter:~$ **sudo apt-get update**

...

lsauser@jupiter:~$ **sudo apt-get install -y nodejs**

...

lsauser@jupiter:~$

We can check what version of node.js is installed:

lsauser@jupiter:~$ **node --version**

v12.22.1

lsauser@jupiter:~$

Let's test our newly installed **node.js**. Don't worry we wont's start developing an app, we will copy the hello world example from the official site (<https://nodejs.org/en/docs/guides/getting-started-guide/>) and paste it (change the address to 0.0.0.0) in an **app.js** file in our home directory:

lsauser@jupiter:~$ **vi app.js**

const http = require('http');

**const hostname = '0.0.0.0';**

const port = 3000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello World\n');

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

Save and close the file.

Now, we can run our application with:

lsauser@jupiter:~$ **node app.js**

Server running at http://0.0.0.0:3000/

...

We can stop it any time by pressing **Ctrl+C**. In order to check the application output, we can open second SSH session and assuming that we have the **w3m** text browser, we can execute:

lsauser@jupiter:~$ **w3m http://localhost:3000**

...

We are ready to continue with the next step.

### Dependencies Exploration

We can ask for the dependencies between binary and all shared libraries that it may need. This way, we can find any unmet dependencies. For this, we can use the **ldd** command:

lsauser@jupiter:~$ **ldd /bin/ls**

linux-vdso.so.1 (0x00007ffecc940000)

libselinux.so.1 => /lib/x86\_64-linux-gnu/libselinux.so.1 (0x00007f21c8924000)

libc.so.6 => /lib/x86\_64-linux-gnu/libc.so.6 (0x00007f21c873a000)

libpcre.so.3 => /lib/x86\_64-linux-gnu/libpcre.so.3 (0x00007f21c86c6000)

libdl.so.2 => /lib/x86\_64-linux-gnu/libdl.so.2 (0x00007f21c86c0000)

/lib64/ld-linux-x86-64.so.2 (0x00007f21c8b78000)

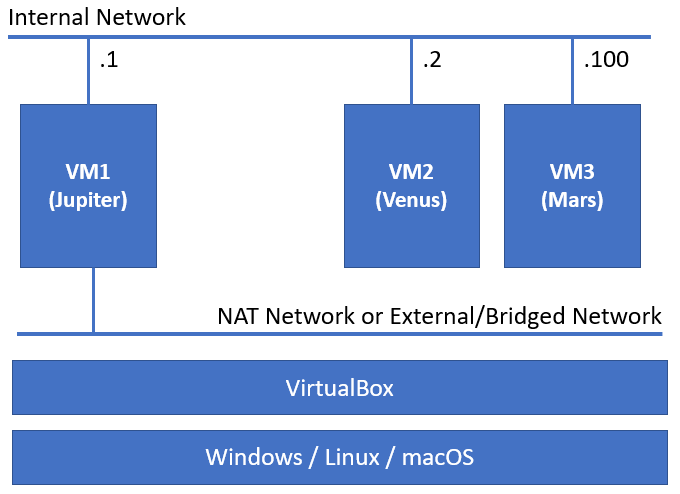
libpthread.so.0 => /lib/x86\_64-linux-gnu/libpthread.so.0 (0x00007f21c869f000)

lsauser@jupiter:~$

## Part 3: Network Services

### Goal

Our goal is to create a setup like the one bellow by applying what we learned so far:



We can assume that we have:

* Server – **Ubuntu VM #1** – machine with two network adapters (we can reuse the one prepared during part 2) that will act as a router. In addition, it will run services like DHCP, FTP, and NTP. Until the end of the practice it will be referred as **jupiter**;
* Stations – **Ubuntu VM #2** and **#3** – those are standard VMs with just one network adapted. They will be connected only to the internal network (as shown on the picture). Until the end of the document they will be referred as **venus** and **mars**;

If you are short on resources, you can use just the server and one station. You can also lower the RAM of the VMs to **512 MB**.

### Preparation

As we will continue with the machine used in the previous two parts, we do not have to do anything to prepare it.

We must focus on the workstations. We must import one or more, but do not forget during the import process to activate the option for **MAC** re-initialization. In addition, as post-import step, we must switch the network adapter of each workstation to **Internal Network** mode and select the name used for the second card of the server.

### (Station 1) Configure with Static IP Address

Start the station #1, login as **lsauser** and execute:

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname venus.lsa.lab**

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname --pretty 'Venus Station'**

lsauser@ubuntu:~$ **hostnamectl**

Static hostname: venus.lsa.lab

Pretty hostname: Venus Station

...

lsauser@ubuntu:~$

Now, we must set the host name to be persistent:

lsauser@ubuntu:~$ **sudo vi /etc/cloud/cloud.cfg**

…

preserve\_hostname: true

…

lsauser@ubuntu:~$

Next, we must set the network adapter with a static IP address (your YAML file may be with different name):

lsauser@ubuntu:~$ **sudo vi /etc/netplan/50-cloud-init.yaml**

...

addresses: [192.168.200.2/24]

gateway4: 192.168.200.1

nameservers:

addresses: [8.8.8.8]

dhcp4: no

...

lsauser@ubuntu:~$ **sudo netplan apply**

...

lsauser@ubuntu:~$ **ip a**

...

lsauser@ubuntu:~$ **ip r**

...

lsauser@ubuntu:~$ **ping -c 4 192.168.200.1**

We can restart the machine.

### (Station 2) Configure with Dynamic IP Address

Start the station #2, login as **lsauser** and execute:

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname mars.lsa.lab**

lsauser@ubuntu:~$ **sudo hostnamectl set-hostname --pretty 'Mars Station'**

lsauser@ubuntu:~$ **hostnamectl**

Static hostname: mars.lsa.lab

Pretty hostname: Mars Station

...

lsauser@ubuntu:~$

Now, we must set the host name to be persistent:

lsauser@ubuntu:~$ **sudo vi /etc/cloud/cloud.cfg**

…

preserve\_hostname: true

…

lsauser@ubuntu:~$

Next, we check settings of the network adapter (your YAML file may be with different name):

lsauser@ubuntu:~$ cat **/etc/netplan/50-cloud-init.yaml | grep dhcp**

dhcp4: true

lsauser@ubuntu:~$

If we receive the above, it means that the adapter is set to receive an IP address dynamically. This is what we want, and we will leave it as it is.

We can restart the machine.

**(Server) Install and Connect via SSH**

Start the server if not running and log in as **lsauser**.

If we did a default installation and did not forget to select **Install OpenSSH Server**, we should have a working SSH service and we can skip this section.

In case, due to some reason, we do not have an **SSH** installed and working on the server, we can do it by:

lsauser@jupiter:~$ **sudo apt install openssh-server**

...

lsauser@jupiter:~$ **sudo systemctl start sshd**

lsauser@jupiter:~$ **sudo systemctl enable sshd**

lsauser@jupiter:~$ **sudo ufw enable**

lsauser@jupiter:~$ **sudo ufw allow 22/tcp**

lsauser@jupiter:~$ **sudo ufw status verbose**

Assuming that we have a working **SSH**, we can connect to our server and start having a better experience. We will be able to copy and paste commands and exchange files between the host and the **VM**.

If our host is **Windows** based, we have two options:

* If we have recent and updated **Windows 10**, we can enable the **OpenSSH Client** optional feature;
* No matter which version of **Windows** we use, we can install one of the popular tools, for example **PuTTY** for handling **SSH** connections, and perhaps **WinSCP** for moving files between our host and the **VMs**;

Now on, we will assume that we have **OpenSSH Client** installed. This will make all steps that follow the same for **Windows**, **Linux**, or **macOS** host.

If our **VM's** network adapter is set to **NAT**, we must create a forwarding rule in our virtualization solution. For **VirtualBox**, we must do:

* Open the **VM** settings;
* Go to **Network**;
* Select the **NAT** adapter and open **Advanced** section;
* Click on **Port Forwarding**;
* Create a rule – for **host port** set for example **20022**, and for VM port – **22**. Then click **OK**
* Close the **Setting** window by clicking **OK**

Now, we can open a terminal window and enter the following to establish a session:

**ssh -p 20022 lsauser@localhost**

We can copy files to the **VM** by executing:

**scp -P 20022 myfile-local.txt lsauser@localhost:/tmp/myfile-remote.txt**

Alternatively, we can download files from the **VM** to the host (save the file in the current folder on the host) with:

**scp -P 20022 lsauser@localhost:/some-folder/myfile-remote.txt .**

Please note, that the above paths and filenames are arbitrary, you must substitute them with ones appropriate in your situation.

If our **VM's** network adapter is set to **Bridge**, we can omit rule creation and both **-p 20022** and **-P 20022** and change the **localhost** to the **VM's** **IP** address (which we can get with the **ip** command) in the above commands.

**(Server) Install and Configure DHCP**

The needed package is named **isc-dhcp-server** and we can install it with:

lsauser@jupiter:~$ **sudo apt install -y isc-dhcp-server**

...

Then we can modify the configuration of the service:

lsauser@jupiter:~$ **sudo vi /etc/dhcp/dhcpd.conf**

...

Enter in edit/insert mode, modify the first two **option** lines in the file to match those listed below, and then enter the **subnet** block as it is:

option domain-name "lsa.lab";

option domain-name-servers 8.8.8.8;

subnet 192.168.200.0 netmask 255.255.255.0 {

range 192.168.200.100 192.168.200.120;

option routers 192.168.200.1;

option broadcast-address 192.168.200.255;

default-lease-time 600;

max-lease-time 7200;

}

Save the file and quit the editor (if using **vi**, then press **Esc**, and type **:wq** and hit **Enter**).  
Now, it is good time to test, if we have a good configuration file. Execute:

lsauser@jupiter:~$ **sudo dhcpd -t**

...

If we see the word **error** in the output, we must go back and correct the file.

With this, the process of base DHCP configuration is over. We must start and enable the service:

lsauser@jupiter:~$ **sudo systemctl restart isc-dhcp-server**

lsauser@jupiter:~$ **systemctl status isc-dhcp-server**

...

The red text should not scare us, because it states that there is no definition for our external adapter. This is exactly what we want.

Now our workstations (at least those with network adapters set to DHCP mode) will receive their IP addresses. We can go on **Mars** station and check if it got its IP address:

lsauser@mars:~$ **ip a s enp0s3**

...

lsauser@mars:~$ **ip r**

...

lsauser@mars:~$ **ping -c 4 192.168.200.1**

...

We can return on the server (**Jupiter**) and check the leases database with:

lsauser@jupiter:~$ **cat /var/lib/dhcp/dhcpd.leases**

...

Now, we should have a working **DHCP** service, but our stations cannot access Internet yet.

### (Server) Manage the Firewall

Return to the server. Open the main configuration file for **ufw** and change the **DEFAULT\_FORWARD\_POLICY** to **ACCEPT**:

lsauser@jupiter:~$ **sudo vi /etc/default/ufw**

...

**DEFAULT\_FORWARD\_POLICY="ACCEPT"**

...

There is one more configuration file to change - **/etc/ufw/sysctl.conf**. We must uncomment **net/ipv4/ip\_forward=1** line:

lsauser@jupiter:~$ **sudo vi /etc/ufw/sysctl.conf**

...

**net/ipv4/ip\_forward=1**

#net/ipv6/conf/default/forwarding=1

...

lsauser@jupiter:~$

If we want to have **IPv6** forwarding, we must uncomment the corresponding line(s) as well.

In addition, the final configuration step is to add the corresponding rule in the **/etc/ufw/before.rules** file. Insert the following block just after the header comments (on row 10):

lsauser@jupiter:~$ **sudo vi /etc/ufw/before.rules**

...

**# NAT table rules**

**\*nat**

**:POSTROUTING ACCEPT [0:0]**

**# Forward traffic from enp0s8 (internal) through enp0s3 (external).**

**-A POSTROUTING -s 192.168.200.0/24 -o enp0s3 -j MASQUERADE**

**# Don't delete the 'COMMIT' line or these NAT table rules won't be processed**

**COMMIT**

...

Comments are not obligatory and can be skipped but make the file more readable.

In order to apply the changes, we must restart the firewall:

lsauser@jupiter:~$ **sudo ufw disable && sudo ufw enable**

Now, we can go to one of the workstations and check for Internet connectivity.

Let's first go to the one with the dynamic address (**Mars**):

lsauser@mars:~$ **ping -c 1 softuni.bg**

PING softuni.bg (217.174.159.195) 56(84) bytes of data.

64 bytes from softuni.bg (217.174.159.195): icmp\_seq=1 ttl=63 time=1.20 ms

--- softuni.bg ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 1.205/1.205/1.205/0.000 ms

lsauser@mars:~$

Now, let's go to the one with the static address (**Venus**), and execute the same. It should work as well. If it is not working, then return to the beginning of this section and check the settings we did.

If everything is working as expected, we could update the packages on both stations.

For the sake of testing (is not mandatory), let's modify the settings of the **sshd** service running on **Mars** host. We will change the port to **50022** for example:

lsauser@mars:~$ **sudo systemctl stop sshd**

lsauser@mars:~$ **sudo vi /etc/ssh/sshd\_config**

...

**Port 50022**

...

lsauser@mars:~$ **sudo systemctl start sshd**

lsauser@mars:~$ **systemctl status sshd**

...

The firewall is not enabled by default. Let's add a rule and enable it. But before this, we can test the rule, and if satisfied by the result, add it permanently:

lsauser@mars:~$ **sudo ufw --dry-run allow 50022/tcp**

...

lsauser@mars:~$ **sudo ufw allow 50022/tcp**

...

lsauser@mars:~$ **sudo ufw enable**

...

Now, return to the **Jupiter** server and try to connect to **Mars** via **ssh**:

lsauser@jupiter:~$ **ssh -p 50022 lsauser@192.168.200.100**

The authenticity of host '[192.168.200.100]:50022 ([192.168.200.100]:50022)' can't be established.

ECDSA key fingerprint is SHA256:xrG/XbjJzZynS3RuYERsiqzJgLWthpuU8Wd0yROnNSE.

Are you sure you want to continue connecting (yes/no)? **yes**

Warning: Permanently added '[192.168.200.100]:50022' (ECDSA) to the list of known hosts.

lsauser@192.168.200.100's password:

Welcome to Ubuntu 19.04 (GNU/Linux 5.0.0-29-generic x86\_64)

...

lsauser@mars:~$ **hostnamectl**

Static hostname: mars.lsa.lab

Pretty hostname: Mars Station

Icon name: computer-vm

Chassis: vm

Machine ID: f6ae34a562e54412bce078824fc71836

Boot ID: 119f2cec7d5a45ccb58c9ccfade1f37a

Virtualization: oracle

Operating System: Ubuntu 19.04

Kernel: Linux 5.0.0-29-generic

Architecture: x86-64

lsauser@mars:~$ **exit**

logout

Connection to 192.168.200.100 closed.

lsauser@jupiter:~$

Of course, we can revert the changes in the **sshd** configuration and in the firewall if we want to.

### (Server) Install and Configure NTP

First, we must install the package:

lsauser@jupiter:~$ **sudo apt install -y ntp ntpstat ntpdate**

...

lsauser@jupiter:~$ **sudo vi /etc/ntp.conf**

...

# If you want to provide time to your local subnet, change the next line.

# (Again, the address is an example only.)

**broadcast 192.168.200.255**

...

lsauser@jupiter:~$ **sudo systemctl restart ntp**

...

lsauser@jupiter:~$ **systemctl status ntp**

...

lsauser@jupiter:~$ **ntpq -p**

Now, we must open the appropriate port in the firewall for requests coming from the internal network:

lsauser@jupiter:~$ **sudo ufw allow proto udp from 192.168.200.0/24 to any port 123**

Rule added

lsauser@jupiter:~$ **sudo ufw disable && sudo ufw enable**

...

lsauser@jupiter:~$

Recent versions of **Ubuntu** have **NTP** client installed and active by default. Let's change its configuration and restart it. We will do this on **Mars** only:

lsauser@mars:~$ **sudo vi /etc/systemd/timesyncd.conf**

...

[Time]

**NTP=192.168.200.1**

...

lsauser@mars:~$ **sudo systemctl restart systemd-timesyncd**

lsauser@mars:~$ **systemctl status systemd-timesyncd**

...

Now, our stations can ask the Jupiter server for an accurate time. We can ask for detailed information with:

lsauser@mars:~$ **timedatectl timesync-status**

Server: 192.168.200.1 (192.168.200.1)

Poll interval: 2min 8s (min: 32s; max 34min 8s)

Leap: normal

Version: 4

Stratum: 2

Reference: 4D4E992D

Precision: 1us (-22)

Root distance: 18.424ms (max: 5s)

Offset: -8.338ms

Delay: 949us

Jitter: 3.152ms

Packet count: 2

Frequency: +101.766ppm

lsauser@mars:~$

**Install and Configure FTP**

First, we will install all necessary packages on the server:

lsauser@jupiter:~$ **sudo apt install -y vsftpd**

...

Next, we can explore and modify the configuration file if needed. In general, it should be okay for our purposes:

lsauser@jupiter:~$ **sudo vi /etc/vsftpd.conf**

...

anonymous\_enable=NO

...

write\_enable=YES

...

This is by default set to **NO**. If we made any changes, we must restart the service.

Of course, we must modify the firewall rules. We will access the **FTP** only from the internal network:

lsauser@jupiter:~$ **sudo ufw allow proto tcp from 192.168.200.0/24 to any port 21**

Rule added

lsauser@jupiter:~$ **sudo ufw disable && sudo ufw enable**

Firewall stopped and disabled on system startup

Command may disrupt existing ssh connections. Proceed with operation (y|n)? **y**

Firewall is active and enabled on system startup

lsauser@jupiter:~$ **sudo ufw status numbered**

Status: active

To Action From

-- ------ ----

[ 1] 22/tcp ALLOW IN Anywhere

[ 2] 123/udp ALLOW IN 192.168.200.0/24

[ 3] 21/tcp ALLOW IN 192.168.200.0/24

[ 4] 22/tcp (v6) ALLOW IN Anywhere (v6)

lsauser@jupiter:~$

The last step is to test the **FTP** from one of our stations. The ftp client is installed by default. So we can try to connect:

[lsauser@venus ~]$ **ftp 192.168.200.1**

We will be asked for a username and password. By default, it will assume that we want to use the current user. In the current situation, we have the same user with the same password on both the server and the stations, so we can continue with it.

Files are uploaded with **put** and downloaded with **get**. Remote folder listing can be seen with **ls**. To execute command locally, for example local file listing, prefix it with an exclamation mark: **!ls**

We can exit the **FTP** client with **quit**.