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

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

This covers both **Debian** and **Ubuntu**. There are points on which they differ. It will be stated clearly.

Please note that for each practice we start with a fresh machine or set of machines.

## Part 1: Network

### Preparation

For this part we will use either **Debian** or **Ubuntu** VM *(stick to the one you prefer)* with two network adapters – one in **NAT** mode (or **Bridged** if you prefer), and another in **Internal Network** (set the name to **MyNet**, for example).

For the next steps we will assume that we have imported a new machine and configured it according to the above requirements. In general, this is what we should have:

A screenshot of a computer

Description automatically generated

### Warning

If you are using **Ubuntu** template installed with the **minimal profile**, you should **unminimize** it first by executing the following command:

lsauser@ubuntu:~$ **sudo unminimize**

And follow the prompts. Note that the machine should have a working Internet connection as the process involves package download.

### 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@debian:~$ **hostnamectl**

Static hostname: debian

...

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

lsauser@debian:~$

*Note that you can substitute* ***set-hostname*** *with* ***hostname*** *and the result will be the same. In fact, exactly* ***hostname*** *can be seen as a sub-command in the man page about* ***hostnamectl***

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

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

lsauser@debian:~$

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

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

jupiter.lsa.lab

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

PRETTY\_HOSTNAME="Jupiter Server"

lsauser@debian:~$

The second one may exist or may not exist.

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

*This is NOT applicable to* ***Debian*** *and is applicable on* ***Ubuntu******ONLY if CloudInit is in use****.*

*Make sure to change the following file:*

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

*…*

***preserve\_hostname: true***

*…*

*lsauser@ubuntu:~$*

In order for 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 (Ubuntu)

*This is applicable* ***ONLY*** *to the recent versions of* ***Ubuntu****. For* ***Debian****, skip to the next section.*

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 files there:

lsauser@jupiter:~$ **ls -al /etc/netplan/**

total 12

drwxr-xr-x 2 root root 4096 Sep 23 13:41 .

drwxr-xr-x 84 root root 4096 Sep 23 13:37 ..

-rw-r--r-- 1 root root 117 Sep 1 07:04 00-installer-config.yaml

lsauser@ubuntu:~$

Usually, the file is named **00-installer-config.yaml** or **50-cloud-init.yaml**

Whichever you find, open it with a text editor, 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**

*Again, 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**

...

enp0s8:

**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, which is connected to an internal network.

### Classic Network Scripts (Debian)

*This is applicable* ***ONLY*** *to* ***Debian****. For* ***Ubuntu****, go to the previous section.*

The network configuration files could be found here:

lsauser@jupiter:~$ **ls -al /etc/network/**

total 32

drwxr-xr-x 7 root root 4096 Aug 31 14:27 .

drwxr-xr-x 68 root root 4096 Sep 23 16:21 ..

drwxr-xr-x 2 root root 4096 Nov 4 2020 if-down.d

drwxr-xr-x 2 root root 4096 Nov 4 2020 if-post-down.d

drwxr-xr-x 2 root root 4096 Nov 4 2020 if-pre-up.d

drwxr-xr-x 2 root root 4096 Nov 4 2020 if-up.d

-rw-r--r-- 1 root root 317 Aug 31 14:27 interfaces

drwxr-xr-x 2 root root 4096 Nov 4 2020 interfaces.d

lsauser@jupiter:~$

The one we are interested in is the **/etc/network/interfaces**. So, let’s open it and add the second adapter with static network settings:

lsauser@jupiter:~$ **sudo vi /etc/network/interfaces**

Make sure to add the following block\* at the end:

**# Add the second network interface**

**allow-hotplug enp0s8**

**iface enp0s8 inet static**

**address 192.168.200.1/24**

*\* Do not forget to adjust the name of the interface to match your situation. You can check, for example, with* ***ip a***

Save and close the file. Restart the machine to apply the changes.

Alternatively, we can restart just the interface:

lsauser@jupiter:~$ **sudo ifdown enp0s8 && sudo ifup enp0s8**

Once back in, check the result with:

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

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 pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:22:89:b9 brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3

valid\_lft 86354sec preferred\_lft 86354sec

inet6 fe80::a00:27ff:fe22:89b9/64 scope link

valid\_lft forever preferred\_lft forever

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

link/ether 08:00:27:d8:e0:7a brd ff:ff:ff:ff:ff:ff

inet 192.168.200.1/24 brd 192.168.200.255 scope global enp0s8

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fed8:e07a/64 scope link

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~$

More information about this file could be found by executing:

lsauser@jupiter:~$ **man 5 interfaces**

## Part 2: Software and Services

### DEB

Let's first install a local package. In order to achieve this, we must download it first. Of course, we should get the right package. You can check by visiting <https://zahariev.pro/linux/hello-lsa>

On **AMD64**-based hardware, execute:

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

On **ARM64**-based hardware, execute:

lsauser@jupiter:~$ **wget https://zahariev.pro/linux/hello-lsa/releases/hello-lsa-1.0\_arm64.deb**

List what we have so far. It should display the downloaded package. For example, for **AMD64**, we should see:

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

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

lsauser@jupiter:~$

We can ask for detailed information with:

lsauser@jupiter:~$ **dpkg --info hello-lsa-1.0\_amd64.deb**

new Debian package, version 2.0.

size 4108 bytes: control archive=388 bytes.

234 bytes, 9 lines control

Package: hello-lsa

Version: 1.0

Section: custom

Priority: optional

Architecture: amd64

Essential: no

Installed-Size: 10812

Maintainer: Dimitar Zahariev <dimitar@zahariev.pro>

Description: Simple utility to display the text Hello LSA.

lsauser@jupiter:~$

In a similar way, we can find all files that will be installed by the package:

lsauser@jupiter:~$ **dpkg --contents hello-lsa-1.0\_amd64.deb**

drwxrwxr-x lsauser/lsauser 0 2021-05-19 14:41 ./

drwxrwxr-x lsauser/lsauser 0 2021-05-19 14:41 ./usr/

drwxrwxr-x lsauser/lsauser 0 2021-05-19 14:42 ./usr/bin/

-rwxrwxr-x lsauser/lsauser 19224 2021-05-19 14:42 ./usr/bin/hello-lsa

lsauser@jupiter:~$

Now, let's install the packages:

lsauser@jupiter:~$ **sudo dpkg -i hello-lsa-1.0\_amd64.deb**

Selecting previously unselected package hello-lsa.

(Reading database ... 35182 files and directories currently installed.)

Preparing to unpack hello-lsa-1.0\_amd64.deb ...

Unpacking hello-lsa (1.0) ...

Setting up hello-lsa (1.0) ...

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://security.debian.org/debian-security bookworm-security InRelease

Hit:2 http://deb.debian.org/debian bookworm InRelease

Hit:3 http://deb.debian.org/debian bookworm-updates InRelease

Reading package lists... Done

Building dependency tree... Done

Reading state information... Done

44 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

base-files/stable 12.4+deb12u7 amd64 [upgradable from: 12.4+deb12u5]

bash/stable 5.2.15-2+b7 amd64 [upgradable from: 5.2.15-2+b2]

...

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/stable 0.5.3+git20230121-2 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, by either editing the **/etc/apt/sources.list** file or by adding a file to the **/etc/apt/sources.list.d/** folder.

There are many additional and / or third-party repositories.

For **Ubuntu**, we can check this site: <https://www.ubuntuupdates.org/ppas>

And for **Debian**, we can check here: <https://wiki.debian.org/DebianRepository/Unofficial>

We can see the files that are keeping track of the available packages for the installed/activated repositories:

lsauser@jupiter:~$ **ls -al /var/lib/apt/lists/\*Packages**

...

For example, to see the contents of one of them, we can use:

lsauser@jupiter:~$ **grep "Package: " /var/lib/apt/lists/deb.debian.org\_debian\_dists\_bookworm\_non-free-firmware\_binary-amd64\_Packages**

Package: amd64-microcode

Package: atmel-firmware

...

We can easily count the packages that are offered by the respective repository with:

lsauser@jupiter:~$ **grep "Package: " /var/lib/apt/lists/deb.debian.org\_debian\_dists\_bookworm\_non-free-firmware\_binary-amd64\_Packages | wc -l**

**38**

lsauser@debian:~$

Now, what about installing completely external/separate software? Like, for example, NodeJS?

Let's follow the procedure (<https://github.com/nodesource/distributions#debian-and-ubuntu-based-distributions>) for installing the repository for **Node.js 18**.

First, arrange to meet the prerequisites by installing any potentially missing packages:

lsauser@jupiter:~$ **sudo apt-get update && sudo apt-get install -y ca-certificates curl gnupg**

**...**

Then prepare the target folder:

lsauser@jupiter:~$ **sudo mkdir -p /etc/apt/keyrings**

And download and install the key:

lsauser@jupiter:~$ **curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg**

Now, we are ready to create the repository:

lsauser@jupiter:~$ **echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node\_18.x nodistro main" | sudo tee /etc/apt/sources.list.d/nodesource.list**

*For other versions (for example,* ***16*** *or* ***20****), change the stated number* ***18****. Check the documentation.*

We are ready to install the desired version of **Node.js**:

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

...

We can check what version of **Node.js** is installed:

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

v18.20.6

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)

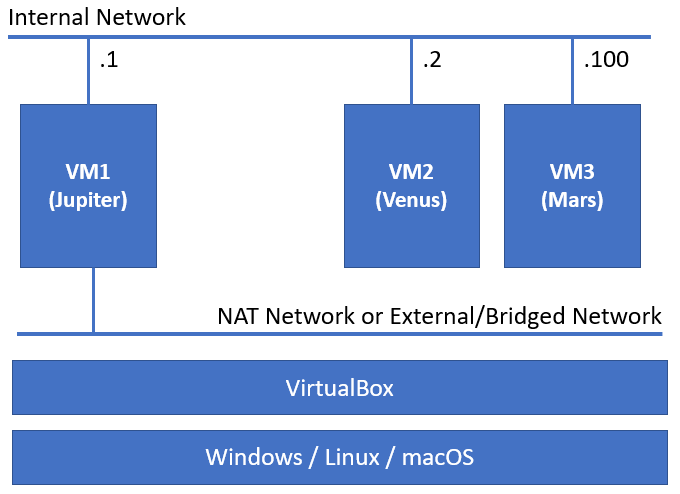
libpthread.so.0 => /lib/x86\_64-linux-gnu/libpthread.so.0 (0x00007f21c869f000)

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 – **Debian / 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 the DHCP service. Until the end of the practice, it will be referred to as **jupiter**;
* Stations – **Debian / 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 in 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 a 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@debian:~$ **sudo hostnamectl set-hostname venus.lsa.lab**

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

lsauser@debian:~$ **hostnamectl**

Static hostname: venus.lsa.lab

Pretty hostname: Venus Station

...

lsauser@debian:~$

#### On Ubuntu

**Skip for Debian and move to the next section**

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

...

Save and close the file. Then check the configuration with:

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

If you see a depreciation warning about the **gateway4** field, change the block to:

...

addresses: [192.168.200.2/24]

routes:

- to: default

via: 192.168.200.1

nameservers:

addresses: [8.8.8.8]

dhcp4: no

...

Finally, apply the new configuration and run some additional checks:

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.

#### On Debian

Let’s open the respective configuration file and change the first adapter with static network settings:

lsauser@jupiter:~$ **sudo vi /etc/network/interfaces**

Make sure to adjust the block to look like this:

**allow-hotplug enp0s3**

**iface enp0s3 inet static**

**address 192.168.200.2/24**

**gateway 192.168.200.1**

*\* Do not forget to adjust the name of the interface to match your situation. You can check, for example, with* ***ip a***

Save and close the file. Restart the machine to apply the changes.

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

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

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

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

lsauser@debian:~$ **hostnamectl**

Static hostname: mars.lsa.lab

Pretty hostname: Mars Station

...

lsauser@debian:~$

#### On Ubuntu

**Skip for Debian and move to the next section**

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.

#### On Debian

Let’s open the respective configuration file and change the first adapter with static network settings:

lsauser@jupiter:~$ **sudo vi /etc/network/interfaces**

Make sure to adjust the block to look like this:

**allow-hotplug enp0s3**

**iface enp0s3 inet dhcp**

*\* Do not forget to adjust the name of the interface to match your situation. You can check, for example, with* ***ip a***

Save and close the file. Restart the machine to apply the changes.

### (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**

*If we are working on* ***Ubuntu*** *and have the firewall active, we must execute the following as well:*

*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** or newer, 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**

...

*Please note that you may see an error which will cause the service to enter a failed state. This is due to an incorrect (not applicable to our case) or missing configuration (missing network adapter configuration). We will take care of this in the following lines.*

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.

If, however, the service refuses to start, then we should check the reason for this by executing:

lsauser@jupiter:~$ **sudo journalctl -xeu isc-dhcp-server.service**

Here we may see an error similar to this one:

...

**Not configured to listen on any interfaces!**

...

This related to the note that we made earlier during the installation. Now, open the following file for editing:

lsauser@jupiter:~$ **sudo vi /etc/default/isc-dhcp-server**

And make sure that you change this:

INTERFACESv4=""

INTERFACESv6=""

To this:

INTERFACESv4="**enp0s8**"

INTERFACESv6=""

*Note that the name of the adapter should match your situation.*

Save and close the file. Now, restart the service again and check its status:

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

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

...

All should be “green” now and we should be able to continue.

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 the Internet yet.

### (Server) Manage the Firewall (Debian)

Please note that **UFW** is **NOT** installed by default on **Debian**. So, it should be enough to change just the following file:

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

And make sure that there is the following:

...

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

...

Or this one:

...

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

...

*Which one to use will depend on the version of the distribution. You can easily decide by taking a look at the other similar settings that you can find the file. In addition, it is usually a matter of uncommenting the line as it is already there. If not, just add it.*

And then apply the changes with:

lsauser@jupiter:~$ **sudo sysctl -p**

In addition, you may need to install **iptables** and create the following rules:

lsauser@jupiter:~$ **sudo iptables -t nat -A POSTROUTING -s 192.168.200.0/24 -o enp0s3 -j MASQUERADE**

...

And then save the rule:

lsauser@jupiter:~$ **sudo iptables-save | sudo tee /etc/iptables/rules.v4**

### (Server) Manage the Firewall (Ubuntu)

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 *(skip these steps for* ***Debian****, where the* ***UFW*** *is* ***not installed*** *by default)*:

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) SSH Keys \*

We can use key instead of the traditional password. Let’s try it.

Return on the server amd generate a pair of private and public key (hit the **Enter** key four times):

lsauser@jupiter:~$ **ssh-keygen**

Generating public/private rsa key pair.

Enter file in which to save the key (/home/lsauser/.ssh/id\_rsa):

Created directory '/home/lsauser/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/lsauser/.ssh/id\_rsa

Your public key has been saved in /home/lsauser/.ssh/id\_rsa.pub

The key fingerprint is:

SHA256:t/SnKWCyqJtdJ4y34ixGTFgxybMx7ZjRzLrBhSkMVu8 lsauser@jupiter.lsa.lab

The key's randomart image is:

+---[RSA 3072]----+

|+o+@ |

|o.@.B |

| = % . |

|. O o |

| o o E S o |

| + o. oo o |

| . ..=+... . . |

| o+oo.+ . + |

| .+=+.. .o |

+----[SHA256]-----+

lsauser@jupiter:~$

By default this will create a pair of keys using the RSA algorythm which are not password protected and will store them in the .ssh subforlder of the home folder of our user:

lsauser@jupiter:~$ **ls -al ~/.ssh**

total 8

drwx------. 2 lsauser lsauser 38 Sep 23 15:19 .

drwx------. 3 lsauser lsauser 128 Sep 23 15:19 ..

-rw-------. 1 lsauser lsauser 2610 Sep 23 15:19 id\_rsa

-rw-r--r--. 1 lsauser lsauser 577 Sep 23 15:19 id\_rsa.pub

lsauser@jupiter:~$

Should we want to change this behavior, we can add the appropriate options on the command line.

Now, how do we use these keys? We must copy the public key to all stations that we want to use it on.

For example, let’s copy it on the Venus machine by executing:

lsauser@jupiter:~$ **ssh-copy-id 192.168.200.2**

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/lsauser/.ssh/id\_rsa.pub"

The authenticity of host '192.168.200.2 (192.168.200.2)' can't be established.

ED25519 key fingerprint is SHA256:2n9I5AgmewInhFyLfhsvMSGHgETgg6i62dm5lyhTR8E.

This key is not known by any other names

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

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed

/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys

lsauser@192.168.200.2's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh '192.168.200.2'"

and check to make sure that only the key(s) you wanted were added.

lsauser@jupiter:~$

You will be asked to enter the password for the lsauser on the other machine (in our case it is existing there and has the same password).

Now, we can try and do the SSH session just like:

lsauser@jupiter:~$ **ssh 192.168.200.2**

Last login: Fri Sep 23 14:04:13 2022

lsauser@jupiter:~$

And bam, without being asked for a password we managed to establish a session.

Close the session and return to the **Jupiter** machine.