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

\* 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.

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 an **AlmaLinux OS 9.x** VM(or **CentOS Stream 9**, or **Oracle Linux 9.x**, or **Rocky Linux 9.x**) 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

### General

In 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@almalinux ~]$ **hostnamectl**

Static hostname: almalinux.lsa.lab

...

[lsauser@almalinux ~]$ **sudo hostnamectl set-hostname jupiter.lsa.lab**

[lsauser@almalinux ~]$

*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@almalinux ~]$ **sudo hostnamectl set-hostname --pretty 'Jupiter Server'**

[lsauser@almalinux ~]$

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

[lsauser@almalinux ~]$ **cat /etc/hostname**

jupiter.lsa.lab

[lsauser@almalinux ~]$ **cat /etc/machine-info**

PRETTY\_HOSTNAME="Jupiter Server"

[lsauser@almalinux ~]$

The second one may exist or may not exist

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 pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 08:00:27:fd:b7:a1 brd ff:ff:ff:ff:ff:ff

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

link/ether 08:00:27:2c:b7:e0 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, let's turn off the connection on one of the interfaces:

[lsauser@jupiter ~]$ **sudo ip link set dev enp0s8** **down**

[lsauser@jupiter ~]$ **ip l**

...

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

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

[lsauser@jupiter ~]$

If we are not connected through the same interface, we can bring it up with:

[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 qlen 1

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 qlen 1000

link/ether 08:00:27:c6:c0:c7 brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3

valid\_lft 86341sec preferred\_lft 86341sec

inet6 fe80::f00f:bbaa:c8c4:341a/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 08:00:27:2c:b7:e0 brd ff:ff:ff:ff:ff:ff

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

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

[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 static metric 100

10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100

[lsauser@jupiter ~]$

### Network Manager

We have two utilities to manage the network – one for the command line – **nmcli** and one interactive tool – **nmtui**. In fact, **nmtui** is more like a shell, that combines three tools – **nmtui-connect**, **nmtui-edit** and **nmtui-hostname**.

Let's work with **nmcli**. We can ask for general information about the network connectivity:

[lsauser@jupiter ~]$ **nmcli general status**

STATE CONNECTIVITY WIFI-HW WIFI WWAN-HW WWAN

connected full enabled enabled enabled enabled

[lsauser@jupiter ~]$

The above command can be executed as **nmcli general** as well because **status** is the default action. It can be shortened even further to **nmcli g**. Different options can be seen by executing **nmcli help** or **nmcli *command* help**.

Information about the statuses of the existing connections can be received by:

[lsauser@jupiter ~]$ **nmcli connection show**

NAME UUID TYPE DEVICE

enp0s3 ad155b9a-8ea6-413b-bf9c-adfaf1ba9b49 ethernet enp0s3

enp0s8 daf45ae0-f7a4-472e-914f-28f32364f20e ethernet enp0s8

Wired connection 1 defacd16-83ce-3986-93e2-c024b042bafb ethernet --

[lsauser@jupiter ~]$

The output may vary. Let's remove all connections except the one for **enp0s3** (repeat command as many times as needed).

For example, delete the **Wired connection 1** (if exists):

[lsauser@jupiter ~]$ **sudo nmcli connection del "Wired connection 1"**

Connection 'Wired connection 1' (defacd16-83ce-3986-93e2-c024b042bafb) successfully deleted.

[lsauser@jupiter ~]$

Then delete the **enp0s8** connection (if exists):

[lsauser@jupiter ~]$ **sudo nmcli connection del "enp0s8"**

Connection 'enp0s8' (daf45ae0-f7a4-472e-914f-28f32364f20e) successfully deleted.

[lsauser@jupiter ~]$

At the end, we should have just a single one, for example:

[lsauser@jupiter ~]$ **nmcli conn**

NAME UUID TYPE DEVICE

enp0s3 ad155b9a-8ea6-413b-bf9c-adfaf1ba9b49 ethernet enp0s3

[lsauser@jupiter ~]$

There are many reasons for having multiple profiles or connection for the same adapter. We can do:

[lsauser@jupiter ~]$ **sudo nmcli connection add type ethernet ifname enp0s8 con-name static-internal**

Connection 'static-internal' (4abf5881-6234-4aad-8b65-ddf49046a202) successfully added.

[lsauser@jupiter ~]$

Now, depending on the version either check here (default for **branch 8.x**):

[lsauser@jupiter ~]$ **ls -al /etc/sysconfig/network-scripts/ifcfg-static-internal**

-rw-r--r--. 1 root root 284 12 дек 10,35 /etc/sysconfig/network-scripts/ifcfg-static-internal

[lsauser@jupiter ~]$

Or here (default for **branch 9.x**)\*:

[lsauser@almalinux ~]$ **ls -al /etc/NetworkManager/system-connections/static-internal.nmconnection**

-rw-------. 1 root root 200 Sep 23 14:18 /etc/NetworkManager/system-connections/static-internal.nmconnection

[lsauser@almalinux ~]$

*\* More info:* [*https://www.redhat.com/en/blog/rhel-9-networking-say-goodbye-ifcfg-files-and-hello-keyfiles*](https://www.redhat.com/en/blog/rhel-9-networking-say-goodbye-ifcfg-files-and-hello-keyfiles)

We can always use the **nmcli** command:

[lsauser@jupiter ~]$ **nmcli connection show**

NAME UUID TYPE DEVICE

enp0s3 ad155b9a-8ea6-413b-bf9c-adfaf1ba9b49 ethernet enp0s3

static-internal 4abf5881-6234-4aad-8b65-ddf49046a202 ethernet enp0s8

[lsauser@jupiter ~]$

Because of the above, we have our second connection, but again with dynamic address. We can examine its configuration file (output may vary) if on the **8.x branch** with:

[lsauser@jupiter ~]$ **cat /etc/sysconfig/network-scripts/ifcfg-static-internal**

TYPE=Ethernet

PROXY\_METHOD=none

BROWSER\_ONLY=no

BOOTPROTO=dhcp

DEFROUTE=yes

IPV4\_FAILURE\_FATAL=no

IPV6INIT=yes

IPV6\_AUTOCONF=yes

IPV6\_DEFROUTE=yes

IPV6\_FAILURE\_FATAL=no

IPV6\_ADDR\_GEN\_MODE=stable-privacy

NAME=static-internal

UUID=4abf5881-6234-4aad-8b65-ddf49046a202

DEVICE=enp0s8

ONBOOT=yes

[lsauser@jupiter ~]$

Or, if on the **9.x branch** with:

[lsauser@almalinux ~]$ **sudo cat /etc/NetworkManager/system-connections/static-internal.nmconnection**

[connection]

id=static-internal

uuid=68cd4874-4bce-4396-99bc-c50935dc05ce

type=ethernet

interface-name=enp0s8

[ethernet]

[ipv4]

method=auto

[ipv6]

addr-gen-mode=stable-privacy

method=auto

[proxy]

[lsauser@almalinux ~]$

Let's modify its parameters and assign a static IP address:

[lsauser@jupiter ~]$ **sudo nmcli connection modify static-internal ipv4.addresses 192.168.200.1/24 ipv4.method manual**

[lsauser@jupiter ~]$

[lsauser@jupiter ~]$ **sudo nmcli connection down static-internal; sudo nmcli connection up static-internal**

...

[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 brd 192.168.200.255 scope global noprefixroute enp0s8

valid\_lft forever preferred\_lft forever

inet6 fe80::85b:d5f5:9070:7c85/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

[lsauser@jupiter ~]$ **nmcli connection show**

NAME UUID TYPE DEVICE

enp0s3 ad155b9a-8ea6-413b-bf9c-adfaf1ba9b49 ethernet enp0s3

static-internal 4abf5881-6234-4aad-8b65-ddf49046a202 ethernet enp0s8

[lsauser@jupiter ~]$

Let's check the configuration file now (output may vary) if on the **8.x branch** with:

[lsauser@jupiter ~]$ **cat /etc/sysconfig/network-scripts/ifcfg-static-internal**

TYPE=Ethernet

PROXY\_METHOD=none

BROWSER\_ONLY=no

BOOTPROTO=none

DEFROUTE=yes

IPV4\_FAILURE\_FATAL=no

IPV6INIT=yes

IPV6\_AUTOCONF=yes

IPV6\_DEFROUTE=yes

IPV6\_FAILURE\_FATAL=no

IPV6\_ADDR\_GEN\_MODE=stable-privacy

NAME=static-internal

UUID=4abf5881-6234-4aad-8b65-ddf49046a202

DEVICE=enp0s8

ONBOOT=yes

IPADDR=192.168.200.1

PREFIX=24

[lsauser@jupiter ~]$

Or, if on the **9.x branch** with:

[lsauser@almalinux ~]$ **sudo cat /etc/NetworkManager/system-connections/static-internal.nmconnection**

[connection]

id=static-internal

uuid=68cd4874-4bce-4396-99bc-c50935dc05ce

type=ethernet

interface-name=enp0s8

[ethernet]

[ipv4]

address1=192.168.200.1/24

method=manual

[ipv6]

addr-gen-mode=stable-privacy

method=auto

[proxy]

[lsauser@almalinux ~]$

Detailed information about connections and their parameters can be received by:

[lsauser@jupiter ~]$ **nmcli connection show static-internal**

connection.id: static-internal

connection.uuid: 4abf5881-6234-4aad-8b65-ddf49046a202

connection.stable-id: --

connection.type: 802-3-ethernet

connection.interface-name: enp0s8

...

ipv4.method: manual

ipv4.dns: --

...

ipv4.addresses: 192.168.200.1/24

ipv4.gateway: --

...

[lsauser@jupiter ~]$

In general, all parameters written in lower case can be changed on the command line.

In a similar way, we can ask for information about the network devices:

[lsauser@jupiter ~]$ **nmcli device status**

DEVICE TYPE STATE CONNECTION

enp0s3 ethernet connected enp0s3

enp0s8 ethernet connected static-internal

lo loopback unmanaged --

[lsauser@jupiter ~]$

[lsauser@jupiter ~]$ **nmcli device show enp0s8**

GENERAL.DEVICE: enp0s8

GENERAL.TYPE: ethernet

GENERAL.HWADDR: 08:00:27:1C:76:31

GENERAL.MTU: 1500

GENERAL.STATE: 100 (connected)

GENERAL.CONNECTION: static-internal

GENERAL.CON-PATH: /org/freedesktop/NetworkManager/ActiveConnection/63

WIRED-PROPERTIES.CARRIER: on

IP4.ADDRESS[1]: 192.168.200.1/24

IP4.GATEWAY: --

IP4.ROUTE[1]: dst = 192.168.200.0/24, nh =

IP6.ADDRESS[1]: fe80::d0bd:af79:f9cc:fc56/64

IP6.GATEWAY: --

IP6.ROUTE[1]: dst = fe80::/64, nh = ::, mt

IP6.ROUTE[2]: dst = ff00::/8, nh = ::, mt =

[lsauser@jupiter ~]$

We can ask again for the routing table with:

[lsauser@jupiter ~]$ **ip r**

default via 10.0.2.2 dev enp0s3 proto dhcp metric 100

10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100

192.168.200.0/24 dev enp0s8 proto kernel scope link src 192.168.200.1 metric 101

[lsauser@jupiter ~]$

Now, let's add a default gateway to our **static-internal** connection, restart it, and check back again:

[lsauser@jupiter ~]$ **sudo nmcli connection modify static-internal ipv4.gateway 192.168.200.1**

[lsauser@jupiter ~]$

[lsauser@jupiter ~]$ **sudo nmcli connection down static-internal; sudo nmcli connection up static-internal**

...

[lsauser@jupiter ~]$ **ip r**

default via 10.0.2.2 dev enp0s3 proto dhcp metric 100

default via 192.168.200.1 dev enp0s8 proto static metric 101

10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100

192.168.200.0/24 dev enp0s8 proto kernel scope link src 192.168.200.1 metric 101

[lsauser@jupiter ~]$

For our purposes, we don't need a default gateway on this connection, so we will delete it. We can either manually edit the configuration file or do it on the command line. Let's do in on the command line with:

[lsauser@jupiter ~]$ **sudo nmcli connection modify static-internal ipv4.gateway 0.0.0.0**

[lsauser@jupiter ~]$

[lsauser@jupiter ~]$ **sudo nmcli connection down static-internal; sudo nmcli connection up static-internal**

...

[lsauser@jupiter ~]$ **ip r**

default via 10.0.2.2 dev enp0s3 proto dhcp metric 100

10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100

192.168.200.0/24 dev enp0s8 proto kernel scope link src 192.168.200.1 metric 101

[lsauser@jupiter ~]$

We can achieve everything we have done so far with the other available tool – **nmtui**. Explore it with:

[lsauser@jupiter ~]$ **sudo nmtui**

## Part 2: Software and Services

### RPM

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 ~]$ **curl -O https://zahariev.pro/linux/hello-lsa/releases/hello-lsa-1.0-1.el9.x86\_64.rpm**

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

[lsauser@jupiter ~]$ **curl -O https://zahariev.pro/linux/hello-lsa/releases/hello-lsa-1.0-1.el9.aarch64.rpm**

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

[lsauser@jupiter ~]$ **ls -l hello-lsa-1.0-1.el9.x86\_64.rpm**

-rw-r--r--. 1 lsauser lsauser 10899 Mar 27 14:20 hello-lsa-1.0-1.el9.x86\_64.rpm

[lsauser@jupiter ~]$

We can ask for detailed information with:

[lsauser@jupiter ~]$ **rpm -qip hello-lsa-1.0-1.el9.x86\_64.rpm**

Name : hello-lsa

Version : 1.0

Release : 1.el9

Architecture: x86\_64

Install Date: (not installed)

Group : Unspecified

Size : 16273

License : GPLv3+

Signature : (none)

Source RPM : hello-lsa-1.0-1.el9.src.rpm

Build Date : Thu Mar 27 13:51:13 2025

Build Host : almalinux

URL : https://www.zahariev.pro/linux/hello-lsa

Summary : Hello LSA greeting utility

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 ~]$ **rpm -qlp hello-lsa-1.0-1.el9.x86\_64.rpm**

/usr/bin/hello-lsa

...

[lsauser@jupiter ~]$

Now that we have all the information, we can proceed with package installation:

[lsauser@jupiter ~]$ **sudo rpm -ivh hello-lsa-1.0-1.el9.x86\_64.rpm**

Preparing... ################################# [100%]

Updating / installing...

1:hello-lsa-1.0-1.el9 ################################# [100%]

[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 rpm -e hello-lsa**

### DNF (YUM)

Let's continue with **dnf**.

Now we can try to update all installed packages, for which there is an update, on our system:

[lsauser@jupiter ~]$ **sudo dnf upgrade**

...

Upgrade has an alias **up** and a deprecated alias **update**. Both still work.

If there aren't any candidates to be updated, we will see:

Nothing to do.

Complete!

[lsauser@jupiter ~]$

If there are packages ready to be updated, after all checks for additional dependencies are made, we will see a list:

Resolving Dependencies

--> Running transaction check

---> Package GeoIP.x86\_64 0:1.5.0-13.el7 will be updated

---> Package GeoIP.x86\_64 0:1.5.0-14.el7 will be an update

...

Total download size: 258 M

Is this ok [y/d/N]:

We can either confirm (type **y**) or deny (type **n**) the proposed list and hit the **Enter** key to set our choice. If we decided to go for an update, then it could happen that we need to answer one more question:

...

Is this ok [y/N]:

We must accept the key import. Then the update process will continue.

Let's search for a tool called **wget** (a tool for retrieving files via HTTP and FTP):

[lsauser@jupiter ~]$ **dnf search wget**

...

wget.x86\_64 : A utility for retrieving files using the HTTP or FTP protocols

[lsauser@jupiter ~]$

Now, that we are sure about the name of the package, we can continue with the installation:

[lsauser@jupiter ~]$ **sudo dnf install wget**

...

Is this ok [y/d/N]: **y**

...

[lsauser@jupiter ~]$

It is time to check what repositories we have installed on our system:

[lsauser@jupiter ~]$ **dnf repolist**

repo id repo name

appstream AlmaLinux 9 - AppStream

baseos AlmaLinux 9 - BaseOS

extras AlmaLinux 9 - Extras

[lsauser@jupiter ~]$

These are the default and enabled repositories on a typical **AlmaLinux** system. Should we want to see all installed repositories, including the disabled ones, we can execute:

[lsauser@jupiter ~]$ **dnf repolist --all**

To enable or disable a registered repository, we must use the **config-manager** sub-command. For example, to enable the **Plus** repository, we can execute:

[lsauser@jupiter ~]$ **sudo dnf config-manager --enable plus**

Of course, we can end up in a situation in which we must add one or more additional repositories. Different repositories can be installed by following different approaches. There are three ways in general:

* installing a repository as a regular package
* using a special sub-command (**config-manager**) or in case of YUM - tool (**yum-config-manager**)
* download the configuration file and store it in a special folder – **/etc/yum.repos.d/**

Let's first install one of the most popular additional repositories for **RedHat**-based distributions – **EPEL** which stands for **Extra Packages for Enterprise Linux**. It is available as a regular package:

[lsauser@jupiter ~]$ **sudo dnf install -y epel-release**

...

Other **AlmaLinux** repositories can be seen here: <https://wiki.almalinux.org/repos/AlmaLinux.html>

We can check the **/etc/yum.repos.d/** folder and pick up one of the **repo** files there and explore its content.

*Please note that the following section may behave differently between the various distributions in the family and their versions (****8.x branch*** *vs* ***9.x branch****).*

Let’s explore the modules feature. First, check the list of available modules with:

[lsauser@jupiter ~]$ **dnf module list**

Now let’s utilize this functionality and install the **Node.js** platform. We can refer to this URL: <https://nodejs.org/en/download/package-manager/#centos-fedora-and-red-hat-enterprise-linux>

According to the procedure, we must execute the following command *(on* ***AlmaLinux 9.x****)*:

[lsauser@jupiter ~]$ **sudo dnf module install nodejs:18**

Before the installation, we could have checked what the module contains with:

[lsauser@jupiter ~]$ **dnf module info nodejs:18**

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 listening address as shown below) 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 forward port **3000** to let's say port **8080** on the host.

Open a browser tab and navigate to <http://localhost:8080>

If nothing happens, return on the console session and press **Ctrl+C** to stop the application

Now, check if the **firewalld** service is running:

[lsauser@jupiter ~]$ **systemctl status firewalld**

If it is running (as we do not know yet how to manage it), stop it with:

[lsauser@jupiter ~]$ **sudo systemctl stop firewalld**

Start the application again and check in the browser window that it is working as expected.

We are ready to continue with the next step.

Before we close this section, we can install the missing legacy networking (**arp**, **ifconfig**, **route**, etc.) tools. First, we need to know how the package is named. We can use the **provides** sub-command, as we know how the tools are named:

[lsauser@jupiter ~]$ **dnf provides netstat**

...

***net-tools-2.0-0.64.20160912git.el9.x86\_64 : Basic networking tools***

...

[lsauser@jupiter ~]$ **sudo dnf install -y net-tools**

...

[lsauser@jupiter ~]$

Now, we can experiment with commands like **netstat**, **ifconfig**, and **route**.

And a few more tricks. Let’s check what packages we can install from the **baseos** repository:

[lsauser@jupiter ~]$ **dnf repoquery --repo baseos**

...

We can use the same against a repository which is not enabled by default, like:

[lsauser@jupiter ~]$ **dnf repoquery --repo crd**

...

In addition to modules, we have the concept of groups. We can list available groups with:

[lsauser@jupiter ~]$ **dnf group list**

...

Then, we can examine information for one of them, for example:

[lsauser@jupiter ~]$ **dnf group info "System Tools"**

...

And finally, install it (skip this step) with:

[lsauser@jupiter ~]$ **sudo dnf group install "System Tools"**

...

### Dependencies Exploration

We can ask for the dependencies between a 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 => (0x00007fff1d572000)

libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f9f8236f000)

libcap.so.2 => /lib64/libcap.so.2 (0x00007f9f8216a000)

libacl.so.1 => /lib64/libacl.so.1 (0x00007f9f81f61000)

libc.so.6 => /lib64/libc.so.6 (0x00007f9f81b94000)

libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f9f81932000)

libdl.so.2 => /lib64/libdl.so.2 (0x00007f9f8172e000)

/lib64/ld-linux-x86-64.so.2 (0x00007f9f82596000)

libattr.so.1 => /lib64/libattr.so.1 (0x00007f9f81529000)

libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f9f8130d000)

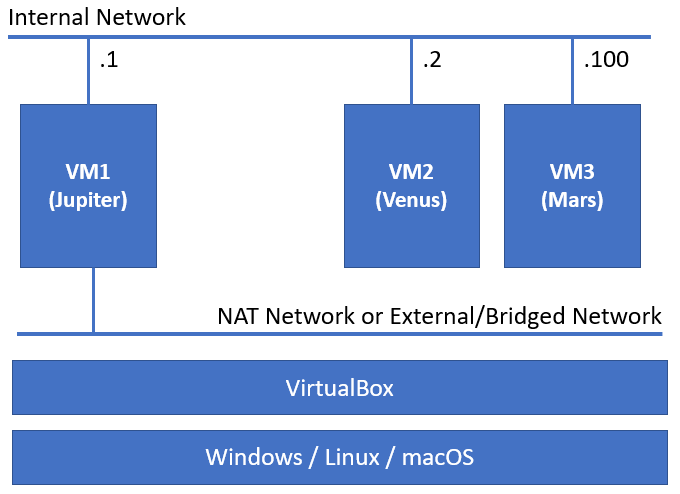
[lsauser@jupiter ~]$

## Part 3: Network Services

### Goal

Our goal is to create a network consisting of three machines. One acting as a server or a router and facing the external world and two machines acting as clients that are behind the router.

For this, we will create a setup like the one bellow by applying what we learned so far:



We can assume that we have:

* Server – **AlmaLinux 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 – **AlmaLinux VM #2** и **#3** – those are standard VMs with just one network adapter. 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. Also, 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 a Static IP Address

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

[lsauser@almalinux ~]$ **sudo hostnamectl set-hostname venus.lsa.lab**

[lsauser@almalinux ~]$ **sudo hostnamectl set-hostname --pretty 'Venus Station'**

[lsauser@almalinux ~]$ **hostnamectl**

Static hostname: venus.lsa.lab

Pretty hostname: Venus Station

...

[lsauser@almalinux ~]$

Next, we check settings of the network adapter (on **branch 8.x**):

[lsauser@almalinux ~]$ **cat /etc/sysconfig/network-scripts/ifcfg-enp0s3 | grep BOOT**

BOOTPROTO="dhcp"

ONBOOT="yes"

[lsauser@almalinux ~]$

If nothing appears or you are on **branch 9.x** check here as well:

[lsauser@almalinux ~]$ **cat /etc/NetworkManager/system-connections/enp0s3.nmconnection | grep -i method**

method=auto

[lsauser@almalinux ~]$

If we receive any of the above, it means that the adapter is set to receive an IP address dynamically. For this station, we do not want this behavior, so we will modify it with:

[lsauser@almalinux ~]$ **sudo nmcli connection modify enp0s3 ipv4.addresses 192.168.200.2/24 ipv4.gateway 192.168.200.1 ipv4.method manual**

[lsauser@almalinux ~]$ **sudo nmcli conn down enp0s3; sudo nmcli conn up enp0s3**

...

[lsauser@almalinux ~]$ **ip a**

...

[lsauser@almalinux ~]$ **ip r**

...

[lsauser@almalinux ~]$ **ping 192.168.200.1**

The same can be done using the **nmtui** utility.

We can restart the machine.

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

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

[lsauser@almalinux ~]$ **sudo hostnamectl set-hostname mars.lsa.lab**

[lsauser@almalinux ~]$ **sudo hostnamectl set-hostname --pretty 'Mars Station'**

[lsauser@almalinux ~]$ **hostnamectl**

Static hostname: mars.lsa.lab

Pretty hostname: Mars Station

...

[lsauser@almalinux ~]$

Next, we check settings of the network adapter:

[lsauser@almalinux ~]$ **cat /etc/sysconfig/network-scripts/ifcfg-enp0s3 | grep BOOT**

BOOTPROTO="dhcp"

ONBOOT="yes"

[lsauser@almalinux ~]$

If nothing appears or you are on **branch 9.x** check here as well:

[lsauser@almalinux ~]$ **cat /etc/NetworkManager/system-connections/enp0s3.nmconnection | grep -i method**

method=auto

[lsauser@almalinux ~]$

If we receive any of the above, it means that the adapter is set to receive an IP address dynamically. This is exactly what we want for this station, so 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 with the **Minimal** profile, 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 dnf install -y openssh**

...

[lsauser@jupiter ~]$ **sudo systemctl start sshd**

[lsauser@jupiter ~]$ **sudo systemctl enable sshd**

[lsauser@jupiter ~]$ **sudo firewall-cmd --add-service=ssh --permanent**

[lsauser@jupiter ~]$ **sudo firewall-cmd --reload**

If 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 on the host to establish a session to the server:

**ssh -p 20022 lsauser@localhost**

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

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

Or 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 **dhcp** and we can install it with:

[lsauser@jupiter ~]$ **sudo dnf install -y dhcp-server**

...

Then we can modify the configuration of the service:

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

...

Enter in edit/insert mode and type the following:

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;

}

Normally the option **domain-name-servers** must contain the **DNS** server for our domain or our local **DNS** server. Currently, we do not have one, so in order for the internal station to have a valid **DNS** record and working name resolution, we will set the address to one of the public well-known **DNS** servers.

Save the file and quit the editor (if using **vi**, then press **Esc**, and type **:wq** and hit **Enter**).

Now, it is a 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.

Assuming that our configuration is correct, we must start and enable the service:

[lsauser@jupiter ~]$ **sudo systemctl start dhcpd**

[lsauser@jupiter ~]$ **sudo systemctl enable dhcpd**

Created symlink from /etc/systemd/system/multi-user.target.wants/dhcpd.service to /usr/lib/systemd/system/dhcpd.service.

[lsauser@jupiter ~]$ **systemctl status dhcpd**

...

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 will receive their IP addresses. If we don't want to wait, we can log to each one of them (at least those set to **DHCP** mode) and execute:

[lsauser@mars ~]$ **sudo nmcli connection up id enp0s3**

...

[lsauser@mars ~]$ **ip a**

...

[lsauser@mars ~]$ **cat /etc/resolv.conf**

# Generated by NetworkManager

search lsa.lab

nameserver 8.8.8.8

We can return on the server and check the leases database with:

[lsauser@jupiter ~]$ **cat /var/lib/dhcpd/dhcpd.leases**

...

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

### (Server) Manage the Firewall

Return to the server (**Jupiter**) and check the status of the firewall:

[lsauser@jupiter ~]$ **systemctl status firewalld**

If it is stopped, start it. Then list the active zones with:

[lsauser@jupiter ~]$ **sudo** **firewall-cmd --get-active-zones**

public

interfaces: enp0s3 enp0s8

[lsauser@jupiter ~]$

*Please note that on* ***9.x branch*** *you may need to use* ***sudo*** *even when you are reading the configuration.*

If we did not change anything, both our interfaces would be in the **public** zone. We must change their zones. List available zones with:

[lsauser@jupiter ~]$ **sudo** **firewall-cmd --get-zones**

Modify both adapter’s zones by executing the following commands.

Put the first network adapter, the one that will act as an external, in the **external** zone:

[lsauser@jupiter ~]$ **sudo nmcli connection modify enp0s3 connection.zone external**

[lsauser@jupiter ~]$

And put the second network adapter, the one connected with the other two stations, in the internal zone:

[lsauser@jupiter ~]$ **sudo nmcli connection modify static-internal connection.zone internal**

[lsauser@jupiter ~]$

*Please note that for* ***the 9.x branch*** *you may need to use the* ***trusted*** *zone instead of the* ***internal*** *one:*

*[lsauser@jupiter ~]$* ***sudo nmcli connection modify static-internal connection.zone trusted***

If we ask again for zones of the interfaces:

[lsauser@jupiter ~]$ **sudo firewall-cmd --get-active-zones**

internal

interfaces: enp0s8

external

interfaces: enp0s3

[lsauser@jupiter ~]$

We will see that now everything is how it should be.

Now we must enable check and enable the **NAT** functionality if not enabled already:

[lsauser@jupiter ~]$ **cat /proc/sys/net/ipv4/ip\_forward**

0

If the above command returned **1**, we should skip the commands that follow. The changes that we made to the connection profiles should be enough to have a working Internet connection for the stations.

If not, then we should execute these ones as well:

[lsauser@jupiter ~]$ **sudo firewall-cmd --zone=external --add-masquerade --permanent**

success

[lsauser@jupiter ~]$ **sudo firewall-cmd --reload**

success

[lsauser@jupiter ~]$ **cat /proc/sys/net/ipv4/ip\_forward**

1

[lsauser@jupiter ~]$

Now, we should be ready.

### (Stations) Adjust Network Configuration

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

...

[lsauser@mars ~]$

Now, let's go to the one with the static address (**Venus**), and execute the same:

[lsauser@venus ~]$ **ping -c 1 softuni.bg**

ping: softuni.bg: Name or service not known

[lsauser@venus ~]$

We will notice that we do not have a working name resolution. If we try to ping **softuni.bg** by its **IP** address, we will see that we have a working Internet communication:

[lsauser@venus ~]$ **ping -c 1 217.174.159.195**

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

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

...

[lsauser@venus ~]$

Let's check the **/etc/resolv.conf** file:

[lsauser@venus ~]$ **cat /etc/resolv.conf**

# Generated by NetworkManager

search lsa.lab

[lsauser@venus ~]$

There isn't any record for a name server. Because our network settings are in manual mode, we must do it explicitly with the following command:

[lsauser@venus ~]$ **sudo nmcli connection modify enp0s3 ipv4.dns 8.8.8.8**

[lsauser@venus ~]$ **sudo nmcli connection down enp0s3;sudo nmcli connection up enp0s3**

Now, if we try to ping an Internet resource by name (for example, softuni.bg), it must work.

If on both stations everything is working as expected, we could update them with the package manager.

Return to the **Mars** station. Check if you have **SSH** service installed and running. If you don't have, then install one.

Assuming that we have a running **SSH** service, we will stop it first in order to modify its configuration. Then, we will change the port from the standard **22** to **50022**:

[lsauser@mars ~]$ **sudo systemctl stop sshd**

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

...

**Port 50022**

...

*Take a moment and explore the file.*

Next, we will test the configuration changes:

[lsauser@mars ~]$ **sudo sshd -t**

Now, we can start the service:

[lsauser@mars ~]$ **sudo systemctl start sshd**

[lsauser@mars ~]$ **systemctl status sshd**

Finally, modify the firewall rules to allow communication on this port:

[lsauser@mars ~]$ **sudo firewall-cmd --add-port=50022/tcp --permanent**

[lsauser@mars ~]$ **sudo firewall-cmd --reload**

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

[lsauser@jupiter ~]$ **ssh -p 50022 lsauser@192.168.200.100**

...

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:

...

[lsauser@mars ~]$ **hostnamectl**

...

[lsauser@mars ~]$ **exit**

logout

Connection to 192.168.200.100 closed.

[lsauser@jupiter ~]$

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

Should we want to, we can add port forwarding on our server to access stations from the outside (for example from our host):

[lsauser@jupiter ~]$ **sudo firewall-cmd --zone=external --add-forward-port=port=12345:proto=tcp:toport=50022:toaddr=192.168.200.100 --permanent**

success

[lsauser@jupiter ~]$ **sudo firewall-cmd --reload**

success

[lsauser@jupiter ~]$

Don't forget to add the appropriate rule for the NAT interface of the server VM if applicable.

### (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@venus ~]$

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

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