**Practice M4: Network. Software. Services (openSUSE)**

\* 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 **openSUSE Leap 15.x** VM 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

*Please note that during the import every adapter for which a new MAC address is generated may not be recognized correctly (initially will be in down state without a valid configuration) by the network stack (if* ***wicked*** *is in use) and additional steps (described below) have to be taken.*

### 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@opensuse:~> **hostnamectl**

Static hostname: opensuse.lsa.lab

...

lsauser@opensuse:~> **sudo hostnamectl set-hostname jupiter.lsa.lab**

lsauser@opensuse:~>

*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@opensuse:~> **sudo hostnamectl set-hostname --pretty 'Jupiter Server'**

lsauser@opensuse:~>

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

lsauser@opensuse:~> **cat /etc/hostname**

jupiter.lsa.lab

lsauser@opensuse:~> **cat /etc/machine-info**

PRETTY\_HOSTNAME="Jupiter Server"

lsauser@opensuse:~>

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.

Let's assume the worst-case scenario – new MAC addresses generated during the import and second network adapted added before the first start. If your situation is different, then please skip some steps that are irrelevant.

If we ask for the network links, we will see something like:

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: eth2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether 08:00:27:18:d1:aa brd ff:ff:ff:ff:ff:ff

3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

lsauser@jupiter:~>

As we can see, all adapters are **in a down** state, which means that they are not connected. We cannot make an SSH connection to our machine even if the SSH service is installed and running.

Depending on how the machine was imported and what is the active network management solution, the situation with you may vary. If there is no **eth0** interface, you must continue with the following steps (**Repair network interfaces**). Otherwise, skip to the **Playing with network interfaces** section.

#### Repair network interfaces

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 bring both interfaces up:

lsauser@jupiter:~> **sudo ip link set dev eth1** **up**

lsauser@jupiter:~> **sudo ip link set dev eth2** **up**

We can see the current IP address 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: eth2: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:18:d1:aa brd ff:ff:ff:ff:ff:ff

inet6 fe80::a00:27ff:fe18:d1aa/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

inet6 fe80::a00:27ff:fe4c:d3b4/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~>

If neither of the network adapters is assigned a valid IP, we can either set manually one or work out the configuration files. Let's first go with the second option.

Create two copies of the existing configuration file and delete the original:

lsauser@jupiter:~> **sudo cp /etc/sysconfig/network/ifcfg-eth0 /etc/sysconfig/network/ifcfg-eth1**

lsauser@jupiter:~> **sudo cp /etc/sysconfig/network/ifcfg-eth0 /etc/sysconfig/network/ifcfg-eth2**

lsauser@jupiter:~> **sudo rm /etc/sysconfig/network/ifcfg-eth0**

Alternatively, delete the following:

lsauser@jupiter:~> **sudo rm /etc/udev/rules.d/70-persistent-net.rules**

lsauser@jupiter:~>

This will remove the remembered interface which had the old MAC address, and which is not available now, after we re-generated it during the import.

In either case, **restart** the system.

*The main difference between the two methods is that in the first, we will end up with* ***eth2*** *(in NAT) and* ***eth1*** *(in internal network). And in the second approach, the names will be* ***eth0*** *(in NAT) and* ***eth1*** *(in internal network). So, depending on what you did, adjust the following sections accordingly.*

#### Playing with network interfaces

If yours are with different indexes (**eth0** and **eth1** instead of **eth1** and **eth2**), you must adjust the commands that follow. Please note that if you came directly to this section, you may need to bring up the second interface.

Once the system has booted, let's login and check the situation with the IP addresses:

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: eth2: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:18:d1:aa brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global eth2

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fe18:d1aa/64 scope link

valid\_lft forever preferred\_lft forever

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

inet6 fe80::a00:27ff:fe4c:d3b4/64 scope link

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~>

As we can see, our **eht1** adapter still does not have any address assigned. We can add one manually with:

lsauser@jupiter:~> **sudo ip address add 192.168.200.1/24 dev eth1**

lsauser@jupiter:~>

lsauser@jupiter:~> **ip address show eth1**

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

inet 192.168.200.1/24 scope global eth1

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fe4c:d3b4/64 scope link

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, related to the manually set static address, will be lost on system restart. We don't want to make them permanent, at least not now, so let's tweak the command to delete the address:

lsauser@jupiter:~> **sudo ip address del 192.168.200.1/24 dev eth1**

lsauser@jupiter:~>

lsauser@jupiter:~> **ip address show eth1**

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

inet6 fe80::a00:27ff:fe4c:d3b4/64 scope link

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~>

With the same tool, we can manage the routing table:

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

default via 10.0.2.2 dev eth2 proto dhcp metric 101

10.0.2.0/24 dev eth2 proto kernel scope link src 10.0.2.15 metric 101

**Check Which One is Used**

**openSUSE** can run either one of the two network management software solutions - **wicked** and **Network Manager**. Normally you would see one of them up and running. So, how we can check which one is running? There are multiple ways of doing this check.

**Option 1** is to execute the following command:

lsauser@jupiter:~> **systemctl show -p Id network.service**

Id=NetworkManager.service

lsauser@jupiter:~>

**Option 2** is to check which service is installed and running by executing known commands:

lsauser@jupiter:~> **systemctl status wickedd**

● wickedd.service - wicked network management service daemon

Loaded: loaded (/usr/lib/systemd/system/wickedd.service; indirect; vendor preset: disabled)

Active: inactive (dead)

lsauser@jupiter:~> **systemctl status NetworkManager**

● NetworkManager.service - Network Manager

Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; vendor preset: disabled)

Drop-In: /usr/lib/systemd/system/NetworkManager.service.d

└─NetworkManager-ovs.conf

Active: active (running) since Sun 2019-03-03 16:39:28 EET; 15h ago

...

lsauser@jupiter:~>

**Option 3** is to use the all-in-one interactive system management utility – **YaST**. It can be started with:

lsauser@jupiter:~> **sudo yast2**

Then you must navigate to **System > Network Settings**

**How to Do the Switch**

***You are not obliged to do it. The following steps are included for completeness only. Feel free to skip them.***

***First, you should make sure that you have them both installed and one of them in use.***

***For NetworkManager this is done via:***

*lsauser@jupiter:~>* ***sudo zypper install NetworkManager***

***And for wicked via:***

*lsauser@jupiter:~>* ***sudo zypper install wicked***

Easiest option is to use **YaST** and change the active service from the **System > Network Settings**

Alternative approach is to execute a set of commands. If want to switch from **Network Manager** to **wicked**, we must execute:

lsauser@jupiter:~> **sudo systemctl stop network.service**

...

lsauser@jupiter:~> **sudo systemctl disable NetworkManager.service**

...

lsauser@jupiter:~> **sudo systemctl enable wicked.service**

...

lsauser@jupiter:~> **sudo systemctl start network.service**

In a similar fashion we can switch from **wicked** to **Network Manager**:

lsauser@jupiter:~> **sudo systemctl stop network.service**

...

lsauser@jupiter:~> **sudo systemctl disable wicked.service**

...

lsauser@jupiter:~> **sudo systemctl enable NetworkManager.service**

...

lsauser@jupiter:~> **sudo systemctl start network.service**

Switching between the two stacks can cause connectivity issues and may require additional configuration.

When switching from **Network Manager** to **wicked** you may want to save and convert the existing configuration to the new service. For this purpose, you must execute:

lsauser@jupiter:~> **cd /etc/wicked/ifconfig**

lsauser@jupiter:/etc/wicked/ifconfig> **sudo wicked show-config compat: | sudo tee all.xml**

...

lsauser@jupiter:~> **cd /etc/sysconfig/network ; sudo mkdir save**

lsauser@jupiter:/etc/sysconfig/network> **mv ifcfg-\* save/**

You may still need to adjust the configuration. **YaST** can help you and save you time.

Should you need to re-read the network interface configurations, execute:

lsauser@jupiter:~> **sudo systemctl restart network**

Both **xml** and **ifcfg** configurations can be mixed, but it is recommended to stick to one of them, it will make things easier and clearer.

### Wicked

***If your installation is not using wicked, feel free to skip this section.***

We can see the status of all network adapters with:

lsauser@jupiter:~> **sudo wicked show all**

lo up

link: #1, state up

type: loopback

config: compat:suse:/etc/sysconfig/network/ifcfg-lo

leases: ipv4 static granted

leases: ipv6 static granted

addr: ipv4 127.0.0.1/8 [static]

addr: ipv6 ::1/128 [static]

eth2 up

link: #2, state up, mtu 1500

type: ethernet, hwaddr 08:00:27:18:d1:aa

config: compat:suse:/etc/sysconfig/network/ifcfg-eth2

leases: ipv4 dhcp granted

leases: ipv6 dhcp requesting

addr: ipv4 10.0.2.15/24 [dhcp]

route: ipv4 default via 10.0.2.2 proto dhcp

eth1 setup-in-progress

link: #3, state up, mtu 1500

type: ethernet, hwaddr 08:00:27:4c:d3:b4

config: compat:suse:/etc/sysconfig/network/ifcfg-eth1

leases: ipv4 dhcp requesting

leases: ipv6 dhcp requesting

lsauser@jupiter:~>

From the above output, we can see that we have two network adapters. Furthermore, we can see where the configuration for each interface is coming from.

Similar result we can achieve by executing **sudo wicked ifstatus all**. If we want information for a specific adapter, we can substitute the word **all** with the adapter's name - **eth0**.

For compatibility reasons a valid and preferred way of managing configurations is to use the **ifcfg-xxx** files in **/etc/sysconfig/network**.

Let's examine the contents of the configuration file for adapter that has a valid IP address (**eth2**):

lsauser@jupiter:~> **cat /etc/sysconfig/network/ifcfg-eth2**

BOOTPROTO='dhcp'

BROADCAST=''

ETHTOOL\_OPTIONS=''

IPADDR=''

MTU=''

NAME=''

NETMASK=''

NETWORK=''

REMOTE\_IPADDR=''

STARTMODE='auto'

DHCLIENT\_SET\_DEFAULT\_ROUTE='yes'

Let's create a copy of the file for our other network adapter (**eth1**) under different name if haven't created one earlier:

lsauser@jupiter:~> **sudo cp /etc/sysconfig/network/ifcfg-eth2 /etc/sysconfig/network/ifcfg-eth1**

We are going to use the new file to configure our network adapter #2 (**eth1**). Let's open it for editing (set the **bold** lines):

lsauser@jupiter:~> **sudo vi /etc/sysconfig/network/ifcfg-eth1**

**BOOTPROTO='static'**

BROADCAST=''

ETHTOOL\_OPTIONS=''

**IPADDR='192.168.200.1/24'**

MTU=''

NAME=''

NETMASK=''

NETWORK=''

REMOTE\_IPADDR=''

STARTMODE='auto'

**DHCLIENT\_SET\_DEFAULT\_ROUTE='no'**

Save and close the file (**:wq**).

If yours is with fewer lines, you may execute **man 5 ifcfg** to learn more about the possible options and values.

Next, we must restart the network service and check that everything is set as expected:

lsauser@jupiter:~> **sudo systemctl restart network**

lsauser@jupiter:~> **ip a s eth1**

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:4c:d3:b4 brd ff:ff:ff:ff:ff:ff

inet 192.168.200.1/24 brd 192.168.200.255 scope global eth1

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fe4c:d3b4/64 scope link

valid\_lft forever preferred\_lft forever

lsauser@jupiter:~> **ip r**

default via 10.0.2.2 dev eth2 proto dhcp

10.0.2.0/24 dev eth2 proto kernel scope link src 10.0.2.15

192.168.200.0/24 dev eth1 proto kernel scope link src 192.168.200.1

lsauser@jupiter:~>

Now, we are ready to move forward.

### Network Manager

When using **Network Manager**, we can follow the same techniques as described in the corresponding section of the Practice for **AlmaLinux**.

## 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 users 10899 Mar 27 15:00 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**

### Zypper

Let's continue with **zypper**. We can refresh the information for the installed repositories:

lsauser@jupiter:~> **sudo zypper refresh**

...

All repositories have been refreshed.

lsauser@jupiter:~>

Now we can force the update of all installed packages. If there are any updates ready, they will be installed:

lsauser@jupiter:~> **sudo zypper update -y**

...

lsauser@jupiter:~>

We can skip the **-y** option in order to be able to check what will be updated and to decide shall we continue or not.

We can use the shorter alias **up** instead.

After an upgrade, there could be running programs that use or refer to files that have been changed or removed. You can execute a command to check for those, and eventually restart either the processes or the system:

lsauser@jupiter:~> **sudo zypper ps -s**

The following running processes use deleted files:

PID | PPID | UID | User | Command | Service

-----+------+------+------------+--------------------------+----------------

336 | 1 | 0 | root | lvmetad | lvm2-lvmetad

...

Let's search for a tool command line web browser. We will search not only in the name, but in the description also:

lsauser@jupiter:~> **zypper search -d browser**

Loading repository data...

Reading installed packages...

S | Name | Summary | Type

---+-----------------------------------------+----------------------------+---------

...

| lynx | A Text-Based WWW Browser | package

...

lsauser@jupiter:~>

Now, that we know the name of the package, we can continue with the installation:

lsauser@jupiter:~> **sudo zypper install lynx**

...

Loading repository data...

Reading installed packages...

Resolving package dependencies...

The following 2 NEW packages are going to be installed:

lynx xli

2 new packages to install.

Overall download size: 1,7 MiB. Already cached: 0 B. After the operation, additional 7,7 MiB will be used.

Continue? [y/n/...? shows all options] (y): **y**

...

lsauser@jupiter:~>

We can test our new browser by executing:

lsauser@jupiter:~> **lynx cnn.com**

We must confirm that we accept cookies with **A** for always. Once the page is loaded, we can explore a bit, and then quit with the **q** key.

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

lsauser@jupiter:~> **zypper repos --show-enabled-only**

Repository priorities are without effect. All enabled repositories share the same priority.

# | Alias | Name | Enabled | GPG Check | Refresh

---+---------------------+-----------------------------------+---------+-----------+------------

6 | repo-non-oss | Non-OSS Repository | Yes | (r ) Yes | Yes

7 | repo-oss | Main Repository | Yes | (r ) Yes | Yes

10 | repo-update | Main Update Repository | Yes | (r ) Yes | Yes

11 | repo-update-non-oss | Update Repository (Non-Oss) | Yes | (r ) Yes | Yes

lsauser@jupiter:~>

These are the default repositories on a typical **openSUSE** system. Of course, we can end up in a situation in which we must add a repository. Different repositories can be installed by following different approaches. There are multiple ways:

* using the **yast2** tool
* using the **zypper** tool
* creating or copying a **repo** file in **/etc/zypp/repos.d/** and etc.

Other **openSUSE** repositories can be found on the following places <https://en.opensuse.org/Package_repositories> and <https://en.opensuse.org/Additional_package_repositories>

Now let’s utilize package management and install the **Node.js** platform. We can refer to this URL: <https://nodejs.org/en/download/package-manager/all#opensuse-and-sle>

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

lsauser@jupiter:~> **sudo zypper install nodejs20**

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

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

v20.18.2

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 to 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 tools. First, we need to know how the package is named. We can use a special tool for this purpose:

lsauser@jupiter:~> **cnf ifconfig**

The program 'ifconfig' can be found in following packages:

\* net-tools-deprecated [ path: /bin/ifconfig, repository: zypp (repo-oss) ]

\* net-tools-deprecated [ path: /usr/bin/ifconfig, repository: zypp (repo-oss) ]

Try installing with:

sudo zypper install net-tools-deprecated

lsauser@jupiter:~>

Now, we know what package to install in order to have it. We can do it now, or test another approach for the same:

lsauser@jupiter:~> **zypper search -d ifconfig**

Loading repository data...

Reading installed packages...

S | Name | Summary | Type

--+----------------------+----------------------------------------------+--------

| gnome-nettool | GNOME Interface for Various Networking Tools | package

| moreutils | Additional Unix Utilities | package

i | net-tools | Important Programs for Networking | package

| net-tools-deprecated | Deprecated Networking Utilities | package

| python2-sh | Python subprocess interface | package

| python3-sh | Python subprocess interface | package

lsauser@jupiter:~>

We can ask for detailed information:

lsauser@jupiter:~> **zypper info net-tools-deprecated**

Loading repository data...

Reading installed packages...

Information for package net-tools-deprecated:

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

Repository : Main Repository

Name : net-tools-deprecated

Version : 2.0+git20170221.479bb4a-lp151.4.3

Arch : x86\_64

Vendor : openSUSE

Installed Size : 460.1 KiB

Installed : No

Status : not installed

Source package : net-tools-2.0+git20170221.479bb4a-lp151.4.3.src

Summary : Deprecated Networking Utilities

Description :

This package contains the deprecated network utilities arp, ifconfig, netstat and route,

which have been replaced by tools from the iproute2 package:

\* arp -> ip [-r] neigh

\* ifconfig -> ip a

\* netstat -> ss [-r]

\* route -> ip r

lsauser@jupiter:~>

As usual, there are many ways to achieve something. We can install the package and experiment with those tools or with their replacements as suggested by the last few lines above.

Last but not least, should we want to see which packages are installed from particular repository, we can execute:

lsauser@jupiter:~> **zypper search -i -r 14**

...

S | Name | Summary | Type

--+----------------------------+----------------------------------------------------------+--------

i | gstreamer-plugins-bad | GStreamer Streaming-Media Framework Plug-Ins | package

i | gstreamer-plugins-bad-lang | Translations for package gstreamer-plugins-bad | package

...

lsauser@jupiter:~>

By changing the above command a bit, we can see all the available packages (and patterns) in a repository:

lsauser@jupiter:~> **zypper search -r 14**

...

S | Name | Summary | Type

--+----------------------------+----------------------------------------------------------+--------

| 32bit | 32-Bit Runtime Environment | pattern

v | apparmor | AppArmor | pattern

...

lsauser@jupiter:~>

### 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 (0x00007ffea0700000)

libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f7ff12f4000)

libcap.so.2 => /usr/lib64/libcap.so.2 (0x00007f7ff10ef000)

libc.so.6 => /lib64/libc.so.6 (0x00007f7ff0d35000)

libpcre.so.1 => /usr/lib64/libpcre.so.1 (0x00007f7ff0aa8000)

libdl.so.2 => /lib64/libdl.so.2 (0x00007f7ff08a4000)

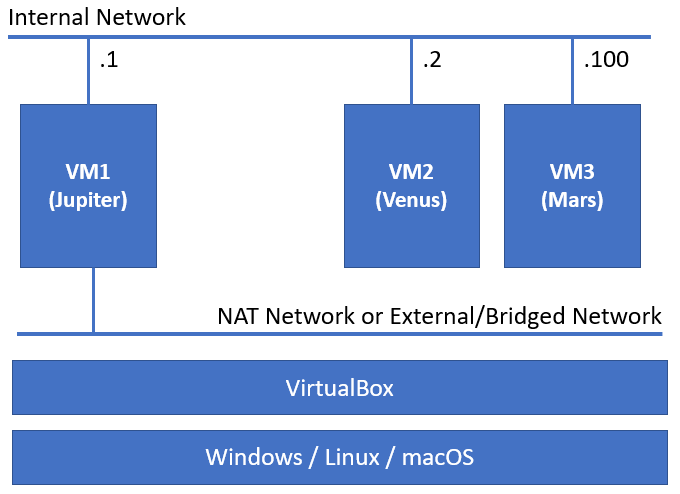
/lib64/ld-linux-x86-64.so.2 (0x00007f7ff173f000)

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 – **openSUSE 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 – **openSUSE 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.

Depending on the network stack in use, we may have to take care of different things.

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

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

lsauser@opensuse:~> **sudo hostnamectl set-hostname venus.lsa.lab**

lsauser@opensuse:~> **sudo hostnamectl set-hostname --pretty 'Venus Station'**

lsauser@opensuse:~> **hostnamectl**

Static hostname: venus.lsa.lab

Pretty hostname: Venus Station

...

lsauser@opensuse:~>

Next, we must check which network platform is running our network and act accordingly.

We must set our adapter to be with static IP **192.168.200.2/24** with gateway set to **192.168.200.1** (in case of **wicked** create or modify the file **/etc/sysconfig/network/routes** (or **/etc/sysconfig/network/ifroute-eth1**)and make sure that the following line exists **default 192.168.200.1 - eth1**) and DNS set to **8.8.8.8** (in case of **wicked** modify the file **/etc/sysconfig/network/config** by altering the parameter **NETCONFIG\_DNS\_STATIC\_SERVERS**).

Depending on the network stack, we can use either **nmtui** or **yast2** to ease our job.

Once done, we can restart the machine.

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

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

lsauser@opensuse:~> **sudo hostnamectl set-hostname mars.lsa.lab**

lsauser@opensuse:~> **sudo hostnamectl set-hostname --pretty 'Mars Station'**

lsauser@opensuse:~> **hostnamectl**

Static hostname: mars.lsa.lab

Pretty hostname: Mars Station

...

lsauser@opensuse:~>

By default, the network adapter is set to DHCP, so we don't need to change anything.

We can restart the machine.

### (Server) Install and Connect Via SSH

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

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 zypper 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**

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**

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.

### Install and Configure DHCP

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

lsauser@jupiter:~> **sudo zypper install -y dhcp-server**

...

Our first task is to configure the network interface (set it to listen on the one with the static IP address):

lsauser@jupiter:~> **sudo vi /etc/sysconfig/dhcpd**

...

# Examples: DHCPD\_INTERFACE="eth0 eth1 eth2"

# DHCPD\_INTERFACE="ANY"

#

**DHCPD\_INTERFACE="eth1"**

...

Then we can modify the configuration of the service:

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

...

Enter in edit/insert mode and type the following (let's say just after the header):

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 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.

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

lsauser@jupiter:~> **sudo systemctl start dhcpd**

lsauser@jupiter:~> **sudo systemctl enable dhcpd**

Created symlink /etc/systemd/system/dhcp-server.service → /usr/lib/systemd/system/dhcpd.service.

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

lsauser@jupiter:~> **systemctl status dhcpd**

...

Now our workstations will receive their IP addresses. We can log to the one (**Mars**) that is set to receive its IP address via DHCP and check if it got new address:

lsauser@mars:~> **ip a**

...

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

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

...

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 and current configuration of the firewall:

lsauser@jupiter:~> **sudo firewall-cmd --get-active-zones**

public

interfaces: eth2 eth1

lsauser@jupiter:~> **sudo firewall-cmd --get-zones**

block dmz drop external home internal public trusted work

lsauser@jupiter:~> **sudo firewall-cmd --get-zone-of-interface=eth2**

public

lsauser@jupiter:~> **sudo firewall-cmd --get-zone-of-interface=eth1**

public

lsauser@jupiter:~>

Now, let's set each interface in separate zone:

lsauser@jupiter:~> **sudo firewall-cmd --zone=external --change-interface=eth2 --permanent**

success

lsauser@jupiter:~> **sudo firewall-cmd --zone=internal --change-interface=eth1 --permanent**

success

lsauser@jupiter:~> **sudo firewall-cmd --reload**

success

lsauser@jupiter:~> **sudo firewall-cmd --get-active-zones**

external

interfaces: eth2

internal

interfaces: eth1

lsauser@jupiter:~>

### (Stations) Adjust Network Configuration

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

lsauser@venus:~> **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@venus:~>

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

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 vi /etc/ssh/sshd\_config**

...

**Port 50022**

...

lsauser@mars:~> **sudo systemctl restart sshd**

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

...

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**

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

ECDSA key fingerprint is SHA256:wQ+AJGr5G24fMloLq65Wwn77STLUMDCyc8E9nyaNZRE.

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.

Password:

Last login: Tue Sep 24 00:43:04 2019

Have a lot of fun...

lsauser@mars:~> hostnamectl

Static hostname: mars.lsa.lab

Transient hostname: mars

Icon name: computer-vm

Chassis: vm

Machine ID: 16333b0f57864155997c10082338fe24

Boot ID: d851f57c83a944ad84f7a53049d9f10c

Virtualization: oracle

Operating System: openSUSE Leap 15.1

CPE OS Name: cpe:/o:opensuse:leap:15.1

Kernel: Linux 4.12.14-lp151.27-default

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 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:m+6fKWLNGcPc71FLu1/rZwA7tR2UeauN67cfF+YyyNE lsauser@jupiter

The key's randomart image is:

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

| o|

| +.|

| . o|

| o .o |

| oS. . E\*=.|

| =oo ===+o|

| oo+ +.+++o|

| o.= o.oooB|

| . ooo+.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 users 38 Sep 23 16:04 .

drwxr-xr-x 8 lsauser users 184 Sep 23 16:04 ..

-rw------- 1 lsauser users 2602 Sep 23 16:04 id\_rsa

-rw-r--r-- 1 lsauser users 569 Sep 23 16:04 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.