SWitCH – Computing systems and networks (SCOMRED)

2019/2020

Laboratory Script – Week 02

Installing Ubuntu Server 18.04 LTS in a virtual machine (Oracle Virtual Box). SSH server. Apache server. Managing Linux Containers (LXC). Creating a Linux Container on the DEI private servers cloud.

IMPERIOUS: After each practical activity (in yellow), students are supposed to reflect on results, and wait for the teacher's acknowledgment before progressing to the next practical activity in this script.

1. Installing Ubuntu Server 18.04 LTS on a Virtual Machine (VirtualBox)

If not already done:

- Download and install Oracle VM VirtualBox.
- Download the Ubuntu Server 18.04 LTS install image (CD/DVD ISO file).

PRACTICE:

a) Start **Oracle VM VirtualBox** and create a new Virtual Machine.

b) Give the new Virtual Machine a suitable name.

For type select **Linux**, and for version **Ubuntu (64 bits)**.

c) Accept the suggested memory (RAM) size for the Virtual Machine (1 GB).

Create a new disk with the suggested size, it will be stored in an image file. An existing disk (image file) could be used, for instance copied from another Virtual Machine.

VirtualBox supports several different image file formats for disks, keep the default/selected format for the new disk's image file to be created.

Select **dynamically allocated**, but first read the included explanation about what it stands for.

Accept the suggested disk size.

Finally click **Create**.

You now have a fresh new Virtual Machine ready to receive an operating system.

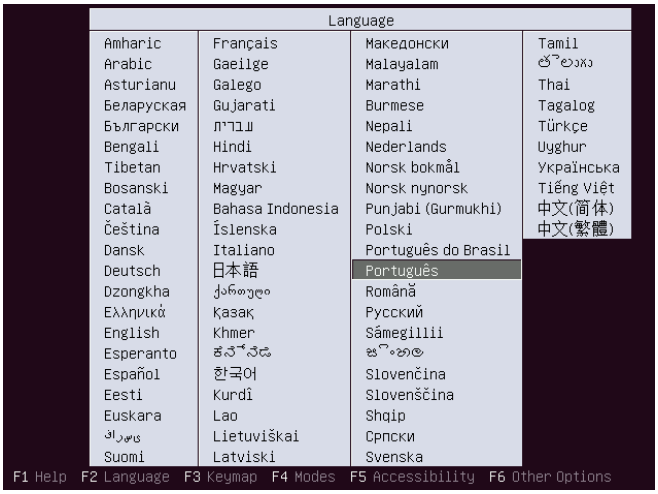
PRACTICE:

The operating system's installation goes the same as if it was a physical machine: insert the installation CD/DVD (an image file in this case) and power on the machine. VirtualBox shows a window to be used as **console**.

Mind the installation of Linux may have significant variations depending on the Linux distribution and version.

We are proceeding with the installation of Linux Ubuntu Server 18.04 LTS.

a) Once the system boots and the operating system's installation starts, at the **console**, select the desired language to be used and press Enter:



b) On the next screen select the first option to **Install Ubuntu Server** and press Enter.

c) Avoid the "Detect keyboard layout" feature, we will simply select it from a list.

Select the keyboard's country and layout, use the first layout presented for the country.

The installation system will now take a while to prepare things.

d) After a while you will be prompted for a **name to be given to the new computer**.

You can give it any name you which, for personal use computers, names are arbitrary. Also, it can be changed later.

If this virtual machine was to become a network server, then the computer should have a name matching it´s future DNS name.

e) Now it's time to establish a new **user who will be able to login to the server and administrate it**.

First you are asked for a text describing the new user, if you which you can leave it blank.

Second you are asked for the username, also known as login name because it must be provided when logging in into the system.

Third you must establish a secret password to be used for authentication. It will have to be entered twice, just to be sure there´re no mistypes.

You should memorize the password, if you forget it, later you won't be able to login.

f) The next enquiry is about this new user's **home directory**.

Multiuser operating systems and user accounts

Linux, and many others, are multiuser operating systems, this means they recognise different users and they will handle each in a different personalized way.

To recognise different users, each user identifies himself to the system through his **username**. The secret **password**, together with the username proves to the system it's really that user.

Internally, operating systems often identify users by other means, for instance in Linux each user has a unique UID and that´s a number. Of course, usernames themselves must be unique within each system.

Each user has a set of properties, including the username, password. The operating system stores that information in the user's database, every valid user has a record there holding that user's properties (attributes). This record that holds a user's properties is called the **user account**.

As seen so far, in a Linux system, the user account has a username, a password and an UID. One other attribute a user account usually has is the user's **home directory name**.

The home directory name identifies, within the filesystem, a folder for private use for the user, outstandingly, the corresponding user is the only user that will be able to write and manage the folder.

To ensure only the correct user has total access his personal folder (home directory), in the filesystem, the folder has specific properties, the most important is: it's owned by the user.

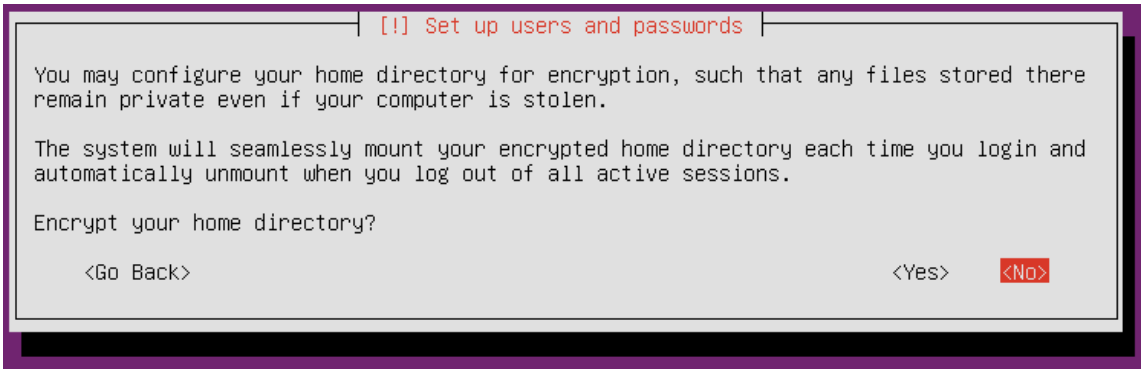
Security and privacy in the home directory are easily enforced by ownership and permissions.

However, that's effective only if the access to the filesystem is controlled by the operating system, it will not work if the disk where the filesystem is stored is simply stolen.

To overcome this kind of scenario, some recent operating system offer the option of encrypting the filesystem. Consequently, only with the secret key/password used in encryption it'll possible to decrypt and understand what's written on the disk.

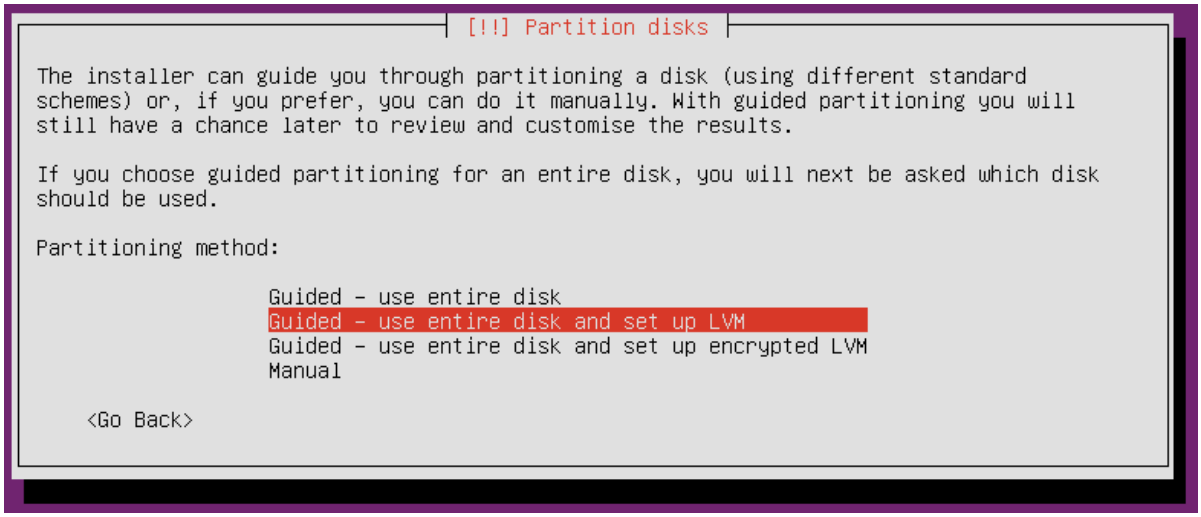
Filesystem encryption can be deployed to an entire filesystem or only some parts like files and folders, in this case the new user's home directory.

When using filesystem's encryption, there's always one drawback to point out. If the key/password is lost, stored date will be unrecoverable.



Having in mind the mentioned drawback, we won´t encrypt the home folder, so keep **No** selected.

g) The next step regards establishing how the disk is going to be used to store the required filesystems, including the operating system itself.



Some options are given, if not aiming a special configuration, **Manual** shouldn't be used. That would also require some expertise.

Guided options will automatically define partitions using the whole disk.

- **Guided – use entire disk**

  This option deploys standard partitions, partitions are disks subdivisions into areas, these partitions are static continuous areas. A partition of this type can´t be spread through different areas of the disk.

- **Guided – use entire disk and set up LVM**

  LVM stands for Logical Volume Manager this allows a much more flexible management of disks. Volumes will be made available instead of partitions, but volumes very flexible, a volume can encompass several partitions even in different disks. Volumes are not static, as far as there's free space in the disk, the volume can always be expanded.

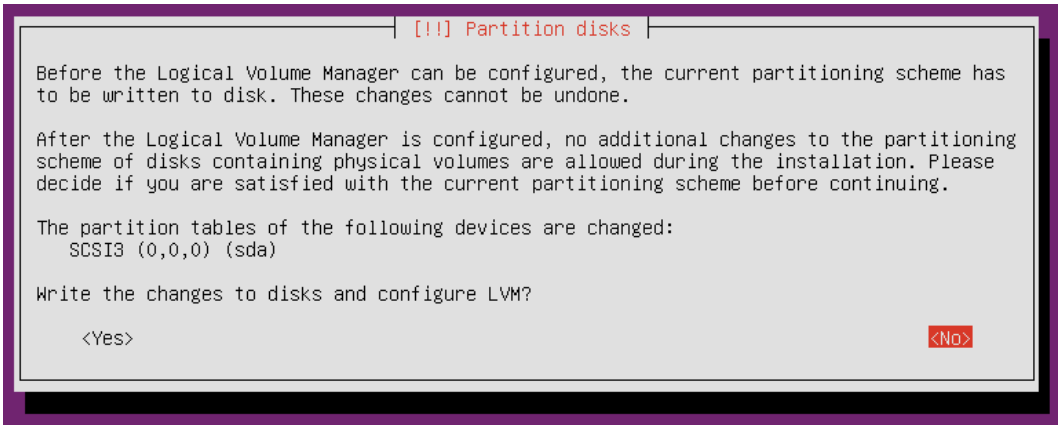
- **Guided – use entire disk and set up encrypted LVM**

  The same as before but the volume will be encrypted by LVM.

Choose option: **Guided – use entire disk and set up LVM**.

Then select the disk where the LVM partition is to be created. No alternatives here, the Virtual Machine has only one disk.

Now, because changes to the disk are eminent a final confirmation is asked:



Select **Yes**.

h) Use the whole disk's space for the new volume.

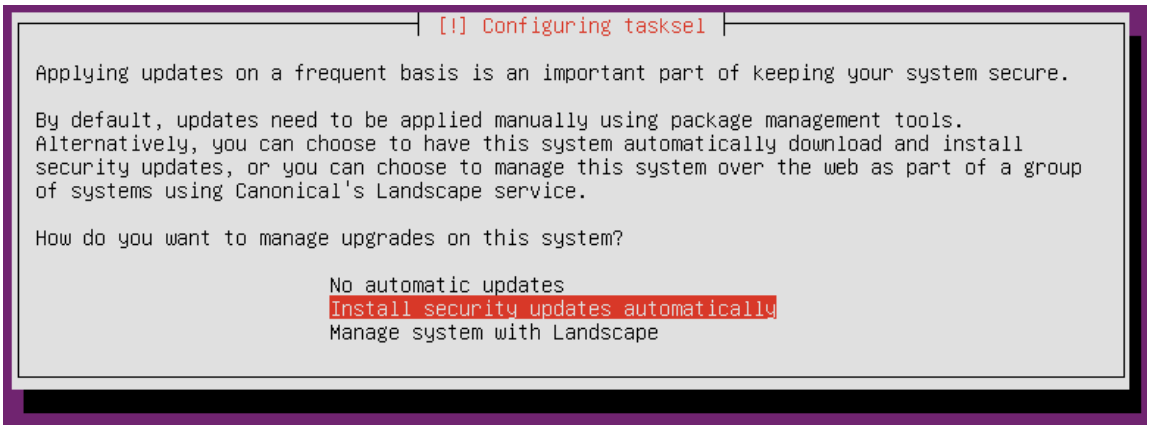
Additional confirmations may be required before finally creating the partitions on the disk.

i) Leave **proxy information** blank and continue

The installation system is now going to download updated software packages from internet repositories maintained for this Linux distribution. Some local networks may deny direct access to the internet and force the use of proxies. This is not the case of ISEP nor DEI networks.

A proxy is a special server that receives requests from local computers, for instance a request for some web page on the internet, the proxy fetches the page and then hands it over to the local computer. So in this scenario local computers don't access the internet directly, they will always ask the proxy.

j) Operating System and applications updates



Keeping the software updated is most important, particularly for security reasons. This may be done manually at the server's command line or automatically for security updates.

Select the second option, **Install security updates automatically**.

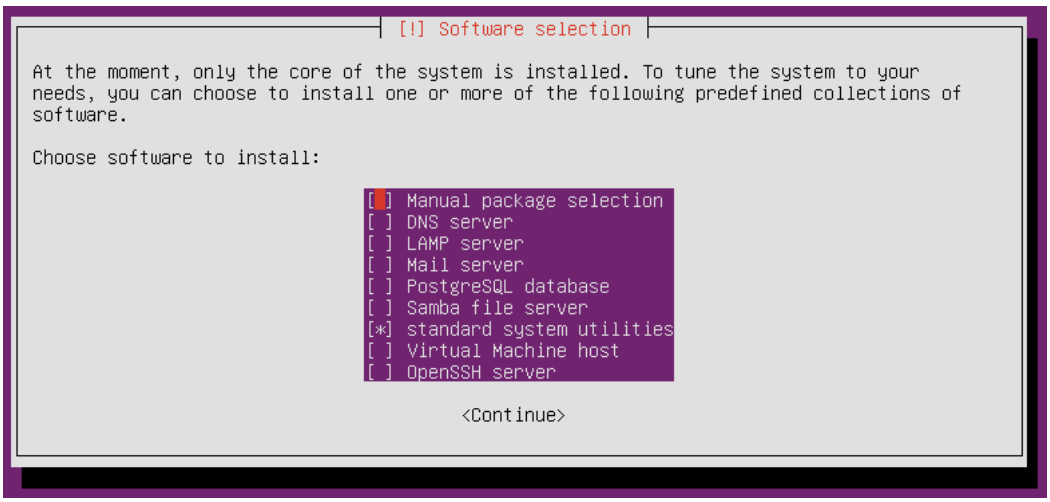
Both manual and automatic updates are simple because there's a repository on the internet with updated packages for the distribution, yet repositories are not maintained forever. For Ubuntu LST (Long Term Support) distributions, they are maintained during five years since launching date, a longer period than for non LTS versions.

Regarding Landscape, it's a network service that allows the management of a set of several Linux servers in a single central server, one thing Landscape can cope with is keeping all software updated on those servers. Imagine you manage 30 Linux servers and you want to install a new application package on each and every one. If those 30 Linux servers are under Landscape control you can do that just once on Landscape and not 30 times, once a time for each server.

k) Next it´s time to select applications and services to install

It's not mandatory all necessary software is installed now. Once the server starts, at any time the administrator will be able to add or remove software packages.

Add to selected packages **DNS server** and **OpenSSH server**, beware this screen is tricky, to select a package use the space bar, not Enter, only when all required packages are selected, then press Enter to processed to the installation.

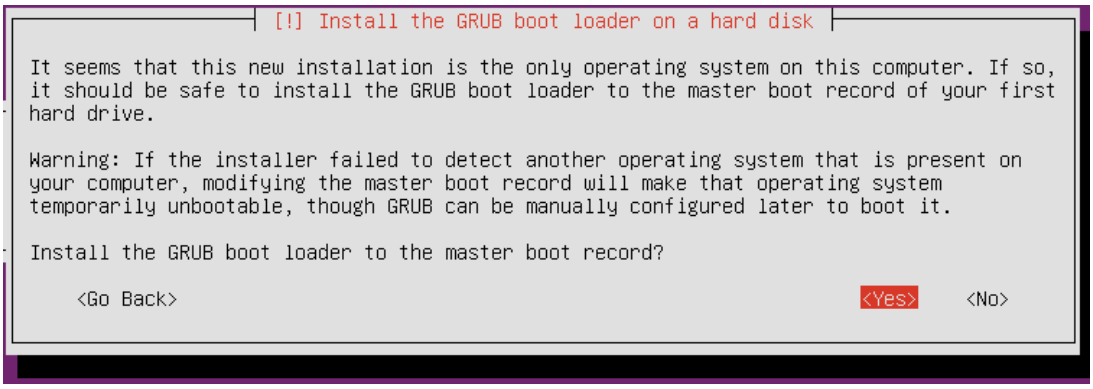


l) GRUB Install

GRUB stands for GNU GRand Unified Bootloader. It's a boot loader and boot manager, when the computer boot it's up to the bootloader to load the operating system into RAM and passing the control to it.

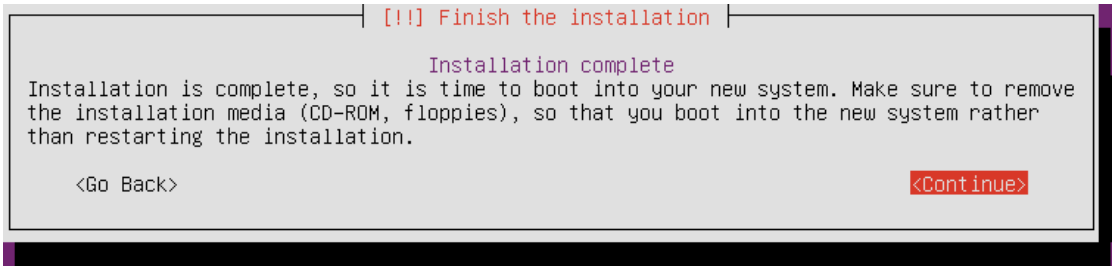
GRUB is also a boot manager, there may be several operating systems on different partitions and volumes of the disk, in that case GRUB presents a menu for the user to select which Operating System he wants to boot, usually there's a default, if none is select within a time period, then the default is booted.

In this case it has been detected there're no other operating systems on the disk, so it's suggested for GRUB to be installed on the disk's MBR, this means this disk boot is going to be managed by GRUB and not any other boot loader or boot manager.



So, on the screen above select **Yes**. It's almost done.

m) Finally, it's ready.



If the computer's BIOS settings had the CD/DVD reader as first boot device you should now remove the CD (image file). For this virtual machine is not required because with default BIOS settings the CD/DVD reader is used for boot only if it can´t boot from the disk, and now the disk has GRUB and Linux.

PRACTICE:

Once the Operating System boot finishes, at the Virtual Machine's console Linux will be prompting for the user's login information.

a) Before logging in, let's check some details.

On the screen, we can see the computer name, the operating system version and the terminal name (**tty1**), in fact the information displayed before the user's login is configurable, it´s known as issue.

There is only one console device but several terminals exist on it. This is most useful because several independent user sessions may be kept at the console by using different terminals. You will have at least six terminals available at the console, tty1 up to tty6, you can switch to another of those terminals just by pressing correspondingly ALT+F1 up to ALT+F6.

Try it please, check that they are different terminals.

b) Now login at tty1 with the username you have created during the operating system installation.

Every multiuser system distinguishes between common users that are allowed to use but not change the system and system's administrator with different levels of permissions to change and configure the system. Usually there´s a top administrator with absolute permission on the whole system called super user. In Linux systems the super user has the username **root**.

The username you are using is not root, it has permissions to administrate the system, but not directly. Among other permissions you can run the **sudo** command. This permission is granted by being a member of a group of users called **sudo**.

The **sudo** command name stands for **super user do**, meaning do something a super user (root).

Often, when called, the sudo command will ask for your password. This is a security feature, if you leave the terminal with an open session, someone taking advantage of that will not be able to use sudo without knowing your password.

On the terminal prompt, enter:

```
who am i
```

Don't be misled, the system doesn´t understand English, the command **who** is being executed and that command recognises the **am** and **i** arguments. More detailed information about your username can be seen with the command:

```
id
```

Now you can see your internal number (UID) and groups of users you belong to, groups have their own internal numbers called Group Identifier (GID).

To prove the point on sudo, now let's run the **id** command as root:

```
sudo id
```

As mentioned, from time to time, sudo asks for your password to be sure it's really you.

2. The Linux Command Line and Bash

The command-line prompt is an essential tool in Linux, often used to automate and troubleshoot tasks. Linux relies heavily on the abundance of command line tools, and this interface provides some interesting advantages:

- Reduced overhead, when compared to a graphical interface;
- Virtually every task in Linux can be accomplished using the command line;
- You can script tasks and series of procedures;
- You can sign into remote machines anywhere on the Internet;
- You can initiate graphical applications directly from the command line.

The program that interprets the commands entered in the command-line is called a *shell* (sometimes *command shell*, or *shell prompt*). There are many different shells, each with slightly different features and we will use one of the most popular shells: the **bash shell**.

Most input lines in the bash have three basic elements:

Command: The name of the program you want to execute

Options: A list of options that modify what the program does and usually start with one or two dashes, e.g. -p or --p. Most commands will accept the option --help, that will output a description of what the command does and options and arguments it takes.

Arguments: One or more arguments of the program, for example a list of input files

You have seen some examples of commands above (who, id, sudo are commands). Here is a list of some other common commands.

Command	Meaning
ls	displays a list of files in the current working directory, like the **dir** command in DOS
cd ***directory***	change directories
mkdir ***directory***	create a directory
passwd	change the password for the current user
file ***filename***	display file type of file with name **filename**
cat ***textfile***	prints the content of **textfile** on the screen
less ***textfile***	prints the content of **textfile** on the screen, page by page
pwd	display present working directory
exit or **logout**	leave this session
man ***command***	read man pages on **command**
head ***textfile***	prints the first lines (by default, the first 10 lines) of **textfile** on the screen
tail ***textfile***	prints the last lines (by default, the first 10 lines) of **textfile** on the screen
adduser ***username***	add a new user called **username**
echo text	prints the given **text**
chmod ***permissions files***	change **permissions** of one or more **files**
ps	display a list of active processes

You can find out the shell you are using with the command (we will see exactly what $SHELL means later):

```
echo $SHELL
```

PRACTICE:

Login using the username and password you defined. Perform the following tasks on your command line:

a) Read the output of **`ls --help`**.

b) List the files in the current directory.

c) Print the current working directory.

d) Read the man page for the command **`wget`**.

e) Download the file http://www.gutenberg.org/cache/epub/174/pg174.txt

f) See the contents of the file **`pg174.txt`**.

g) See the contents of the file **`pg174.txt`**, page by page.

h) List the first 20 lines of the file **`pg174.txt`**.

i) List the last 20 lines of the file **`pg174.txt`**.

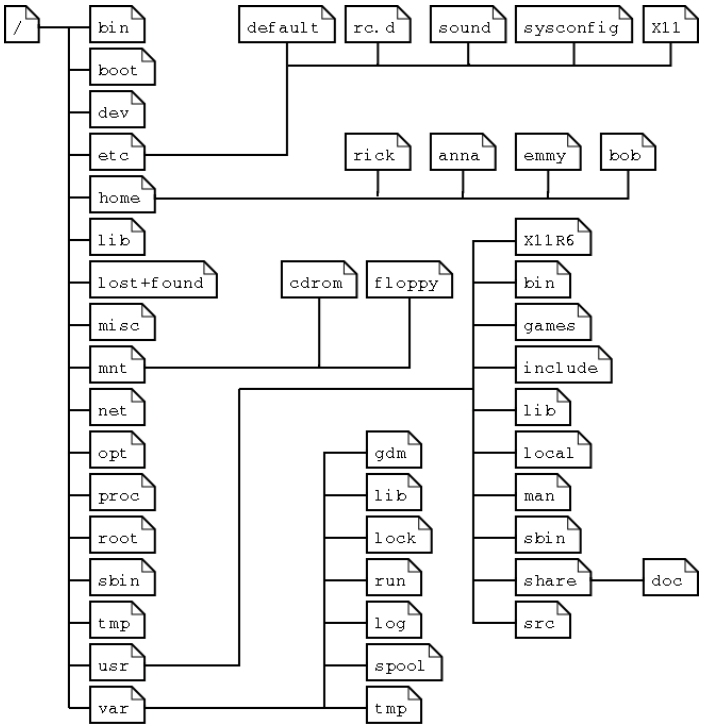
j) Add a new user called **`switch2018`**.

k) Change the password of the user **`switch2018`**.

l) Logout.

3. The Linux Filesystem

In Linux, the character '/' is used to separate different directories (as opposed to '\' in Windows). Here is a common filesystem hierarchy:



The directories in Linux follow a defined hierarchy, where they have specific roles[1].

[1] https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

Here is a list of some important directories of the Linux filesystem and a description of their contents.

/: The filesystem root;

/boot: The startup files and the kernel;

/dev: Contains references to all the CPU peripheral hardware, which are represented as files with special properties;

/bin: Common programs, shared by the system, the system administrator and the users;

/usr: Programs, libraries, documentation, etc. for all user-related programs;

/etc: System configuration files;

/home: Home directories of the users;

/media: Mount point for removable media (e.g. USB drives);

/var: Variable data files. Includes spool (data intended for printing) directories and files, logs and other transient and temporary files;

/tmp: Directory used by programs that need to store temporary files.

Relative and absolute paths

To navigate the filesystem, we can use **relative paths**, which do not start with a '**/**' and are in reference to the current working directory and absolute paths, which start with a '**/**' and are always in reference to the root of the filesystem.

Home Directory

Each user of a Linux system has a directory just for himself, where he can store his own files. This directory is also the default location where new sessions for that user are started.

The home directory is referred using the '**~**'. When the bash finds a '**~**' in your input commands, it replaces it with the location of your home folder. Find the location of your home directory using:

```
echo ~
```

If you enter "**cd**" with no arguments, or "**cd ~**", the current working directory will be changed to your home directory.

PRACTICE:

a) List the contents of **/usr/bin** using the command "**ls .**" (which lists the contents of the current working directory). Use an absolute path to navigate to **/usr/bin.**

b) List the contents of **/usr/bin** using the command "**ls .**". Use only relative path(s) to navigate to **/usr/bin.**

c) If you insert a USB drive on the virtual computer, where would you expect to find its files?

d) Can you tell two ways of printing your home directory?

e) How many users with home directories exist in the system?

4. `/etc` - Important Linux Configuration Files

A very important folder in a Linux system is the /etc. This is where most system configuration files can be found. Here is a list of some relevant configuration files.

File	Information/service
`default`	Default options for certain commands, such as **`useradd`**.
`fstab`	Lists partitions and their *mount points*.
`group`	Configuration file for user groups. Use the command **`groupadd`**, **`groupmod`** and **`groupdel`** to edit this file. Edit manually only if you really know what you are doing.
`hosts`	A list of machines that can be contacted using the network, but without the need for a domain name service. This has nothing to do with the system's network configuration, which is done in **`/etc/sysconfig`**.
`inittab`	Information for booting: mode, number of text consoles etc.
`issue`	Information about the distribution (release version and/or kernel info).
`motd`	Message Of The Day: Shown to everyone who connects to the system (in text mode), may be used by the system admin to announce system services/maintenance etc.
`mtab`	Currently mounted file systems. It is advised to never edit this file.
`passwd`	Lists local users. Use the shadow utilities **`useradd`**, **`usermod`** and **`userdel`** to edit this file. Edit manually only when you really know what you are doing.
`profile`	System wide configuration of the shell environment: variables, default properties of new files, limitation of resources etc.

This is only a small sample of the configuration files you can find in **`/etc`**. You can have a look at the contents of these files using the command **`cat`**. For example:

```
cat /etc/hosts
```

prints the contents of the **`hosts`** file.

PRACTICE:

a) How many user groups are configured in the system?

b) How many users are configured in the system?

c) What is the message shown to everyone that connects to the system?

d) Which partition are mounted when your server boots?

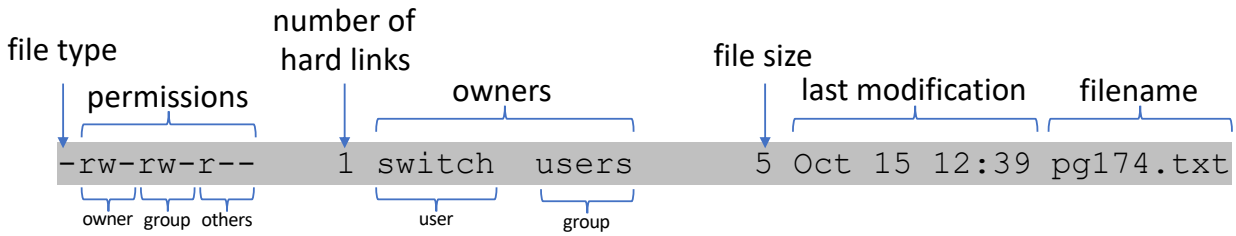
5. File Access Rights

The Linux file permissions defines three categories of users: **owner**, **group** and **others**. All files have an **owner** user and **group**. Permissions of users that are not owners of the file are defined in the **others** category. Each category has three permissions: **read (r)**, **write (w)** and **execute (x)**. Read and write should be self-explanatory, execute only makes sense for programs that you want to execute (Linux will refuse to execute programs that do have execute permissions) and also for directories. You can only enter a directory to which you have execute permissions.

Use the long option **"-l"** of the command **ls** to see the file permissions of a file. Try:

```
ls -l pg174.txt
```

to see the permissions of **pg174.txt.** For example (in your system the output will be slightly different):



The permissions for the file are defined in the string **rw-rw-r--**, and can be read as follows:

- The first set of three characters (**rw-**) is for the **owner** permissions, indicating read (**r**) and write (**w**) permissions over the file. The **-** indicates no execute permissions.
- The second set of three characters (**rw-**) is for the **group** permissions, indicating read (**r**) and write (**w**) permissions over the file. The **-** indicates no execute permissions.
- The third set of three characters (**r--**) is for **other** users permissions, indicating read permissions over the file (no write or execute permissions, indicated by **--**).

After the permissions, **ls** lists the owners of the file. In this case, the owner is the user **switch** and the group is **users**.

File permissions can be manipulated using the command **chmod**. For example:

```
chmod g-w pg174.txt
```

Will remove write permissions (**-w**) for the owner group (**g**). Similarly:

```
chmod o+w pg174.txt
```

Will add write permissions (**+w**) for **other** users (**o**). You can also set permissions for a category:

```
chmod u=rx pg174.txt
```

Will set read and execute permissions for the owner user (**u**).

This is a simple summary of **chmod**. Remember you can do **man chmod** to have a look at its manual pages.

To change the owner of a file use **chown *owner:group file***. You can specify the new owner or the new group, or both. For example:

```
chown switch2018 pg174.txt
```

Will change the owner of the file **pg174.txt** to the user **switch2018**. To specify a group, precede the group name with ':'

```
chown :adm pg174.txt
```

Will change the owner group of the file **pg174.txt** to the group **adm**. You can specify both owner and group at the same time:

```
chown switch2018:adm pg174.txt
```

PRACTICE:

a) Change the owner the file **pg174.txt** to the group **adm**.

b) Change the permissions of the file **~/.bashrc** in your home folder so that only you can read and write to it.

c) See the permissions of the files in **/etc**.

6. Linux processes

A **process** is a very important abstraction used in Linux (and other operating systems). When you run a program, the operating system creates an entity, *called a process*, that provides the program with the illusion that it has exclusive access of the CPU and memory. This is an important abstraction that simplifies the life of programmers. Managing processes is one of the most important roles of an operating system.

In Linux, every process has a unique identifier, called the **Process ID**, or **PID**. You can view the list of active processes in a Linux using the **ps** command. We will use the option **-e** to print all processes in the system:

```
p#ps -e
  PID TTY          TIME CMD
    1 ?        00:00:08 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:02 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    8 ?        00:07:02 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:02:02 migration/0
   11 ?        00:00:03 watchdog/0
   12 ?        00:00:03 watchdog/1
   13 ?        00:01:35 migration/1
   14 ?        00:00:02 ksoftirqd/1
   16 ?        00:00:00 kworker/1:0H
   17 ?        00:00:02 watchdog/2
   18 ?        00:02:01 migration/2

 (continues)
```

Using **ps** with no arguments will print a list of the processes associated with the current session.

Another interesting aspect of Linux is that all processes are created by a parent process (a process is said to be a child of the parent process), forming a tree of parent/child processes. There is a special process that has no parent, called the **`init`** process or, in more modern systems, the **`systemd`**. You can see the process tree using **`pstree`**:

```
#pstree
systemd─┬─ModemManager─┬─{gdbus}
        │              └─{gmain}
        ├─NetworkManager─┬─{gdbus}
        │                └─{gmain}
        ├─accounts-daemon─┬─{gdbus}
        │                 └─{gmain}
        ├─acpid
        ├─agetty
        ├─at-spi-bus-laun─┬─dbus-daemon
        │                 ├─{dconf worker}
        │                 ├─{gdbus}
        │                 └─{gmain}
        ├─at-spi2-registr─┬─{gdbus}
        │                 └─{gmain}
        ├─avahi-daemon───avahi-daemon
        ├─avahi-dnsconfd
        ├─cgmanager
        ├─colord─┬─{gdbus}
        │        └─{gmain}
        ├─console-kit-dae─┬─62*[{console-kit-dae}]
        │                 ├─{gdbus}
        │                 └─{gmain}

 (continues)
```

Linux also allows the user to manipulate processes. The most important process manipulation command is **`kill`**. Despite its name, related to the most common usage of **`kill`** – killing processes – technically, **`kill`** uses a very limited communication primitive, called a signal, that can be used to communicate with processes.

Usage: **`kill`** ***`signal PID`***, where PID is the process identifier of the process to which you want to send the signal. If no signal is indicated, by default **`kill`** will send a TERM signal, which requests the program to exit, allowing it to release resources and do some processing before exiting. For example:

```
kill 18
```

Will send the TERM signal to the process with PID 18[2]. In contrast, for example, you can send the KILL signal, which terminates the program immediately[2]:

```
kill -KILL 18
```

The above (**`kill -KILL`**) can be useful to terminate processes that otherwise refuse to terminate.

It is also useful to know that you can use the keyboard to send certain signals, namely:

- **Ctrl-C** sends an INT signal, which, by default, this causes the process to terminate.
- **Ctrl-Z** sends a TSTP signal ("terminal stop"); by default, this causes the process to suspend execution and go to background (more on this next).

[2] If you try this command, it will fail because process 18 was created by the administrator user root, and you do not have permissions over this process.

Background and Foreground processes

Linux processes can be categorized in two main types:

- **foreground processes** (or interactive processes) which are the processes that are usually initiated by the user and with whom they interact, and
- **background processes** (also referred to as non-interactive/automatic processes), which are processes that don't expect any user input and the most common type of non-interactive processes are those related to system services, such as a secure shell server (SSH) or a webserver.

When the user starts a new process, by default, this process is executing in the foreground, and will expect interaction from the user (e.g. mouse clicks or keyboard inputs). However, you can start any process in the background using the '&' at the end of the command.

A command line popular text editor in **nano**. You can start **nano** in the foreground using:

```
nano &
```

This will have an output similar to this:

```
#nano &
[1] 23827
```

When you start a program from a shell, it is called a job, and the number **[1]** above indicates the **job ID** and the number next to it (23827) is the PID (the PID will very likely be different in your system).

The command **fg** brings to the foreground a previously started process. If you want to put **nano** in the foreground, so that you can interact with it, type, where '**1**' is the job id of **nano**:

```
fg 1
```

Now that you are interacting with **nano**, you can send a TSTP signal ("terminal stop") signal with the key combination **Ctrl-Z**. This will put **nano** back in the background.

PRACTICE:

a) Start two nano processes in the background. One with the file **pg174.txt** loaded and another with a new file called **newfile.txt**:

```
1 nano pg174.txt &
2 nano newfile.txt &
```

b) What is the PID of each **nano** process (use **ps** with no arguments to confirm) ?

c) Put the **nano** process with the **newfile.txt** in the foreground and add the text "Title of the book:". Now, put **nano** back into the background.

d) Put the **nano** with **pg174.txt** in the foreground and copy/see the title of the book. Exit **nano** using the program's option Ctrl-x.

e) Check the list of active processes (use **ps** with no arguments).

f) Put the **nano** with **newfile.txt** back in the foreground and add the tile to the text. Exit **nano** using the program's option Ctrl-x and save the changes.

g) Check if **nano** is no longer in the list of active processes (use **ps** with no arguments).

7. Networking and network services

Now let's check the networking environment around your Linux Virtual Server running in Virtual Box virtual machine.

PRACTICE:

a) On the Linux command shell.

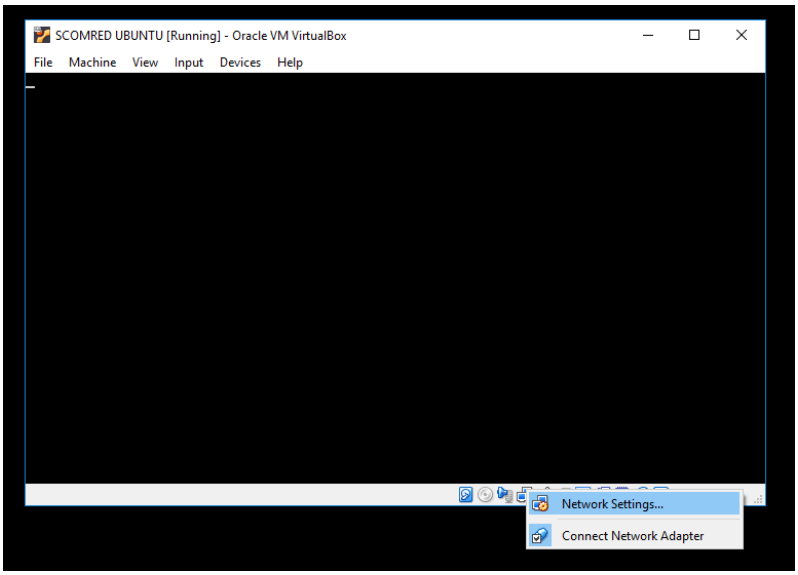
Enter the following command line:

```
ip addr show
```

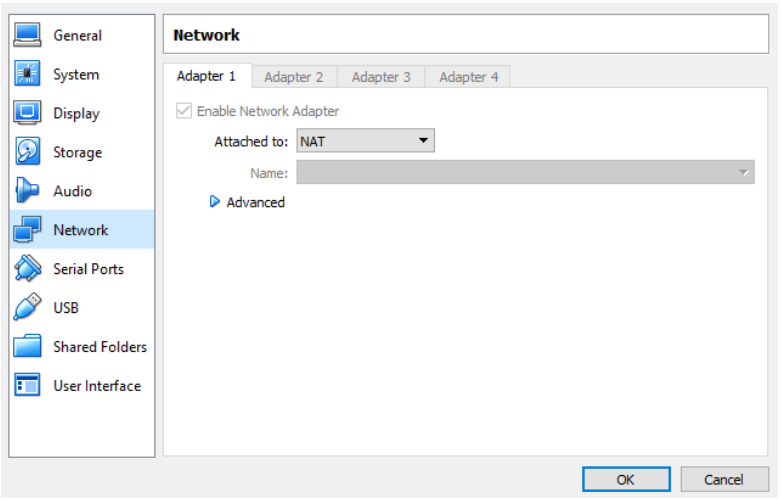
This shows all existing network interfaces' configuration. Check the IPv4 node address you server is using, confront it with the IPv4 node address your laptop is using in the real network.

They don't belong to the same network. This means your server's virtual network interface must be connected to the real network by using NAT. Let's check it.

On the console window's bottom bar use the mouse's right button over the network icon to view Network Settings.



As expected, NAT is being used:



By default, when the Virtual Machine was first created, VirtualBox assumed it would be appropriate to have a single network interface, connected in NAT mode to the real network.

In this specific NAT configuration there´s no Virtual Switch, each Virtual Machine has its own independent connection to the real network. A virtual switch would allow direct communications between Virtual Machines connected to it, this configuration does not.

Using NAT means that direct access from the real (public) network to the virtual (private) network is impossible, it only works the other way around.

This means network services running in your server are not available from the real network.

We already have one important network service running, the SSH (Secure Shell) server. Now we are going to add another.

b) Install the software package named **apache2**.

This package contains a sophisticated web server, once installed it will be started automatically.

Run the command line:

```
sudo apt install apache2
```

The **apt** command is the software packages manager used in this Linux distribution.

Notice the **apt** command is run by user root. That is required because only root is able to manage installed software packages.

Despite using a NAT connection, we have already learned that with some special configurations, it's possible to access network services behind NAT from the public network.

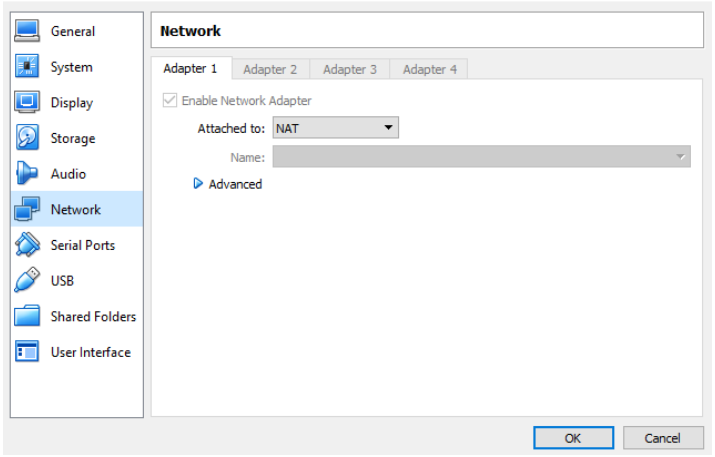
This special configuration is often known as port redirection and it must be deployed on the device performing NAT, in our case the VirtualBox software.

Port redirections are implemented case by case for each network service, so the first thing to know is the port numbers of services to be made accessible. In our case they are going to be:

- Web service on TCP port number 80
- SSH service on TCP port number 22

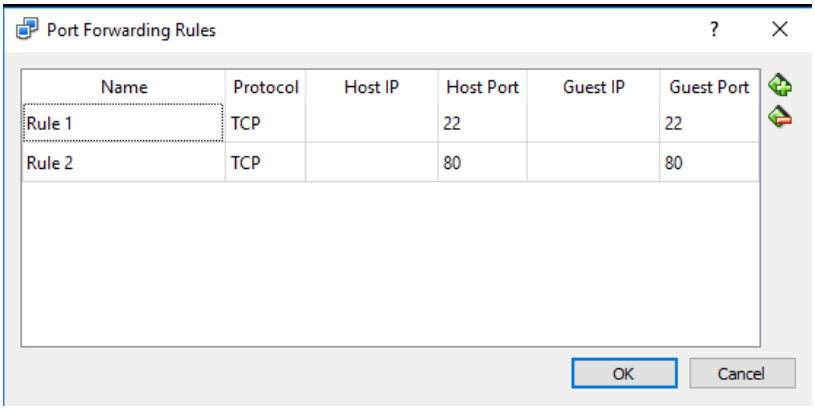
c) In Virtual Box, let´s enforce port redirections for your server's services.

Back on **Network Settings** for your Virtual Machine:



Select **Advanced**, and then, **Port Forwarding**.

Add a rule for each service as presented on this image:

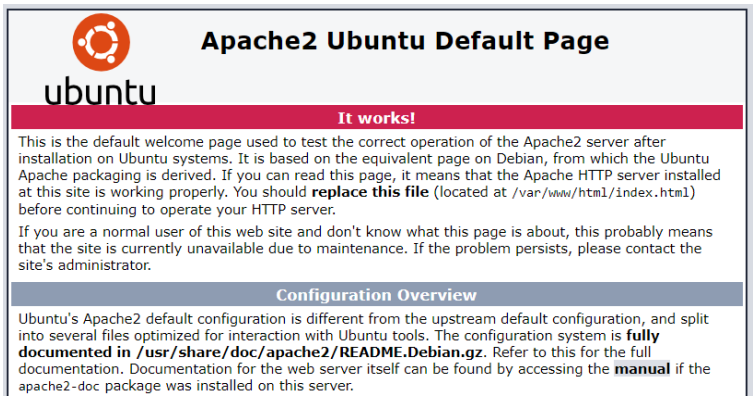


d) Testing access to your server's WEB service (Apache).

Your server's services are now indirectly available by sending requests to the host's IP address (of course the host is your laptop). When running tests on your laptop itself, requests may be sent to the local loopback address 127.0.0.1, usually also recognized by the name **localhost**.

Use your personal web browser on your laptop to open either **localhost** or address **127.0.0.1**.

If everything went ok, you should see something like:



e) Testing access to your server's SSH service (SSH server).

You have already installed an SSH client on your laptop, now use it to connect to you Linux server inside VirtualBox, again, it should be available on address **127.0.0.1** or **localhost**.

Use you SSH client to connect and login to your Linux server.

8. Linux Containers

Linux containers (LXC) are so light they can be run inside a Virtual Machine. Your previously created Linux server in a VirtualBox virtual machine already has support for LXC.

PRACTICE:

Before carrying on, let's take some time to update our operating system by using the apt packages manager.

Start with updating the packages list stored on your server:

```
sudo apt update
```

Now let's update all updatable software packages installed on your server:

```
sudo apt dist-upgrade
```

Now, back to containers.

LXD the **Container Hypervisor Daemon**, is running on your server, you can check that by typing the following command:

```
service lxd status
```

(Use lower case q letter to quit)

At the command line, the **lxc** command can be used to manage LXC containers under control of LXD. Because LXD is a network service, this means the **lxc** command can be used to manage both local containers and remote containers as well. This is achieved by interacting, correspondingly with the local LDX or the remote LXD, through the network.

The first argument following the lxc command is an LXC-COMMAND, lxc command usage:

lxc <LXC-COMMAND> [options]

A list of available command is shown by running:

```
lxc --help
```

This shows the most common lxc commands, a complete list is shown by running:

```
lxc --help --all
```

For help regarding a specific command you can use --help after the command, for instance try:

```
lxc list --help
```

a) Starting to use lxc.

The first time lxc command is used, some initializations must be done, including generating a public key certificate to be used when contacting LXD services.

Request a listing of local containers:

```
lxc list
```

A client certificate has been generated, and you get an empty list. There are no local containers yet.

As advised, run **lxd init** as root:

```
sudo lxd init
```

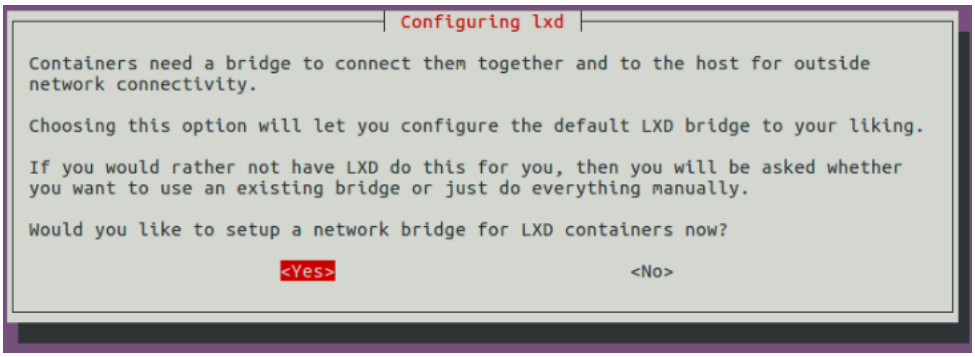
This will be done only once, it will establish the LXD service's configuration.

Accept all defaults:

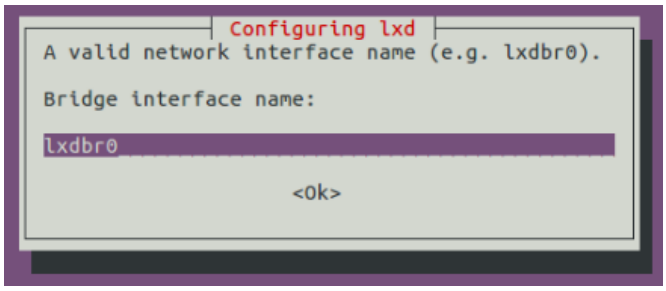
```
sudo lxd init
Do you want to configure a new storage pool (yes/no) [default=yes]?
Name of the storage backend to use (dir or zfs) [default=dir]:
Would you like LXD to be available over the network (yes/no) [default=no]?
Do you want to configure the LXD bridge (yes/no) [default=yes]?
```

The LXD Bridge (Linux Software Bridge) is a virtual switch, we are going to add a virtual DHCP server to it and connect it to the real network by NAT.

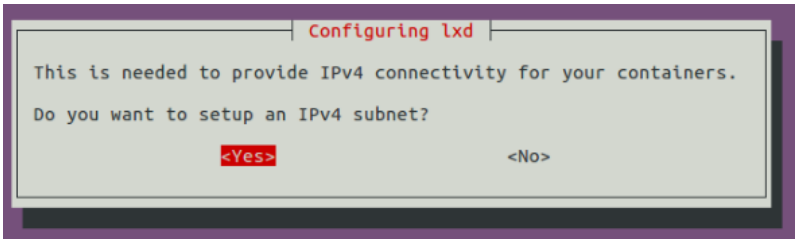
On next screens use the following options:



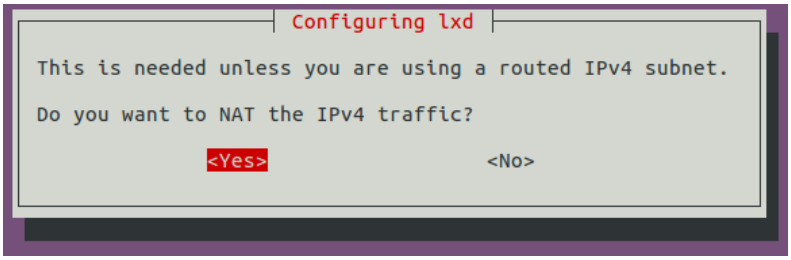
Accept the default bridge name (virtual switch):



Add IPv4 support for the containers, a virtual DHCP server will assign IPv4 configurations within the virtual switch:

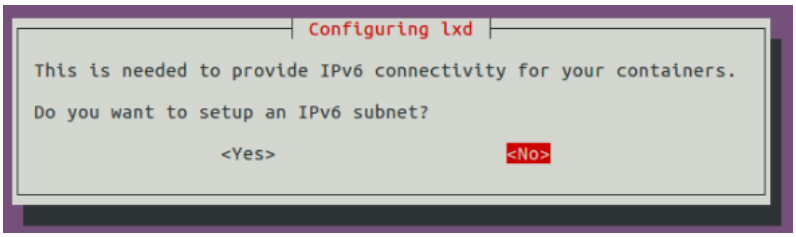


Connect the virtual switch to the real network through NAT:



Accept all defaults, except for IPv6 support.

Don't support IPv6 for the containers:



b) Creating a container

Containers are created from images, many are available on the internet through image servers, your local LXD service already knows some of those images servers, and others could be added.

List image servers being used:

```
lxc remote list
```

One images server is named **ubuntu**, list available images there:

```
lxc image list ubuntu:
```

Get details about one image, named **18.04**, on that server:

```
lxc image info ubuntu:18.04
```

Create a container from that image. You can either use the **launch** command or the **init** command, they both create a new container from a provided image name. The launch command also starts the container upon creating it.

Create a new container named C1 and start it once created.

Run the command:

```
lxc launch ubuntu:18.04 C1
```

This will take a while mainly because the image has to be downloaded and stored locally.

Create a second container named C2 and start it once created:

```
lxc launch ubuntu:18.04 C2
```

Now it's faster because the image is already on your local server.

Create a third container named C3 without starting it:

```
lxc init ubuntu:18.04 C3
```

List the local containers:

```
lxc list
```

Start container C3:

```
lxc start C3
```

List local containers again and again, you may see it takes a while but ultimately they'll all get their IPv4 addresses, assigned by the virtual DHCP server.

Take note of IPv4 addresses being used by each container for testing ahead.

c) Running commands inside a container.

A single command line may be run inside a container, to run the **ls -la /** command inside container C3, simply run:

```
lxc exec C3 -- ls -la /
```

The double minus prevents the lxc command interpreting further options started by minus.

We can also run a shell, enter:

```
lxc exec C2 bash
```

Now you are in a shell inside container C2 (check the prompt). Notice you are root (super user) in the container. You may leave to the host OS any time by typing **exit**.

Within C2 container, try sending ICMP echo requests to the other two containers' IPv4 addresses:

```
ping …
```

Yet within C2 container, request an update of software packages with apt:

```
apt update
```

The **sudo** command is not required because you are already root. You can see, thanks to NAT, you have internet access from within the container.

This looks pretty like a standard operating system running.

Now leave the C2 container (just enter the **exit** command).

Let's compare processes running on the host operating system with those running inside container C1 (guest OS).

Enter the following commands and compare results:

```
ps xa


lxc exec C1 -- ps xa
```

As you may see inside the container there's only a small subset of the processes and services running in a standard operating system, just those needed for the container's specific purposes. In this container you may see there's already an SSH server running.

Let's check the network configuration on the host OS.

Run a command to show every network interface configuration:

```
ip addr show
```

You can see the Virtual Switch (name **lxdbr0**) also has an IPv4 address, so from the host OS you can also reach the containers, try pinging then.

You could even access the container through SSH (the guest OS has a SSH server running), try accessing the container through SSH by using the command line SSH client (ssh) and requesting a session into the IP address being used by the container (GUEST-IP-ADDRESS):

```
ssh GUEST-IP-ADDRESS
```

You won't be able to login due to the guest OS configuration of its SSH server, but that could be easily changed by reconfiguring the SSH service from within the container.

d) Copying files to and from a container.

From the host OS it's possible to copy files to and from the filesystem inside the container, this is possible whether the container is running or stopped. The **file push** command is used to copy files from the host OS to the container's filesystem and the **file pull** command the other way around.

To copy the file **/etc/passwd** from the host OS into the root of the C1 container's filesystem, just run the following command:

```
lxc file push /etc/passwd C1/
```

This may be used either C1 is running or not.

Now we can see the copied file from within the container:

```
lxc exec C1 -- ls -la /passwd
```

Of course, now the container must be running for this to work.

9. Using Linux Containers in DEI cloud

Virtualization concepts, embracing hardware virtualization (Virtual Machines), operating system virtualization (Containers) and network virtualization (Virtual Switches), are used together to establish what is known as cloud computing.

Cloud computing platforms provide a user friendly way to create and manage virtual servers and virtual networks at a high level without handling on how they are implemented.

Several platforms are publicly available, among others: Amazon Web Services (AWS), Google Cloud Platform, Microsoft Azure, IBM Bluemix, and Alibaba Cloud.

At DEI, an experimental cloud computing platform it available, the front end web page is:

https://vs-ctl.dei.isep.ipp.pt/

Warning: this service is available within DEI networks only, if you are somewhere on the internet or connected to a wireless network in ISEP, first you need to establish a VPN connection with DEI.

PRACTICE:

a) Use your DEI credential to login, on the start page you have some documentation to read later.

b) Create your own virtual server.

Go to **Available Virtual Server Templates**, find template number **6**

The template is named **Apache on Ubuntu 18.04 LTS - VNET1** it's an LXC container.

Click **Create a new VS from this template.**

This will take a while, it's an experimental platform mostly implemented over Virtual Machines itself.

While your virtual server is being created and configured, read the available documentation about the platform.

In this platform, users have limited access to virtual networks management, they are limited to use four existing networks named VNET1, VNET2, VNET3, and VNET4. From those, VNET1 is mandatory because it's through it the virtual server will be accessed for administration. Other networks can be freely used.

Of key benefit is every virtual server has a unique static network configuration assigned to it on VNET1, in addition, each virtual server's dedicated IPv4 address is mapped in DNS to the virtual server's name.

The selected template has a single network connection to VNET1, internally named eth0.

c) Start and manage the Virtual Server

Hopefully the virtual server will now be ready. Click on the virtual server name to manage it, initially the same as the template name.

Now you are on **Virtual Server Details** page for your virtual server, click the START button to boot your Virtual Server, once started new options are available.

The root user (super user) is allowed to access through SSH, a random password was established and is available on the web page, it can be changed.

The Virtual Server may be accessed by using a standard SSH client, it can also be accessed through by SSH using a standard web browser (Shell in a box service).

By default **Shell In A Box Direct Access** is disabled, but you can enable it. This allows SSH access without the need to provide a password, it presents some security issues, and anyone with the provided browser link will be able to access your server as root (super user).

Try **Shell In A Box Direct Access**, first enable it, and then click the provided link.

You now have a terminal session on your browser, let's do somethings.

List the virtual server's network interfaces:

```
ip addr show
```

Let's check the networking is in bridge mode. First, we will install a special web browser that is able to run in a terminal, it's called lynx. Use apt to install it:

```
apt install lynx
```

Now use it to access the web page https://rede.dei.isep.ipp.pt/myip

Just type:

```
lynx rede/myip
```

It´s enough because the default DNS domain in your virtual server is **dei.isep.ipp.pt**, and by default lynx adds the **http://** prefix. This web server itself redirects any http request to https, so the browser ends up with **https://rede.dei.isep.ipp.pt/myip**.

We can see your requests, as seen by server **rede.dei.isep.ipp.pt,** seem to be coming from an address that is exactly the IP address your virtual server has. This proves NAT is not being used, also you can see your server is reachable from server **rede.dei.isep.ipp.pt**, your virtual server has replied to the ICMP echo request sent by server **rede.dei.isep.ipp.pt**.

d) The web server

This Virtual Server also has the Apache web server installed and running, the most basic operation of a web server is providing clients (web browsers) the content of static files.

Those static files are referred by clients following the web server name, if no filename is provided by the client, the server assumes a default filename, usually **index.html**.

Go to your **Virtual Server Details** page and click the link to access your HTTP server, you'll get an already familiar Apache Default Web Page, for now this is the default page of your web server.

The web server searches for filenames referred by clients in a base folder, often called the **document root**, in this specific apache configuration the document root is **/var/www/html**. The **index.html** for the Apache Default Web Page should be there.

Let's create our own default web page, we can do that on Shell in a Box.

Jump to the apache document root:

```
cd /var/www/html
```

List the folder's content:

```
ls -l
```

There it is.

Let's replace it by our own. First rename it:

```
mv index.html oldindex.html
```

Now create a new one:

```
nano index.html
```

The suggested text editor is **nano**, if you prefer you may use another.

Define the following content for the text file:

```
<html><head>
<title>My Web server default page</title>
</head>
<body bgcolor=green>
<h1>My Web server default page</h1>
<hr>
<p>The original Apache Default Web Page is available <a href=oldindex.html>here</a>
<hr>
</body>
</html>
```

Save the file, in nano use [CTRL]+[X], then [Y], then [ENTER].

This file uses HTML tags, the main purpose of HTML tags is establishing presentation formatting, so it's mostly about text fonts and colours. The <a> tag is used to create a clickable live link, in this case pointing to the old HTML page.

Now, go back to the default page of your web server and reload it. Match what you see with the HTML content above, test the link to the old page.

Soon in this course we will be talking about HTML (Hyper Text Mark-up Language) in more detail.