LABORATORY "0A". CREATING YOUR COURSE VIRTUAL MACHINE

DEPARTMENT OF COMPUTER SCIENCE. UNIVERSITY OF OVIEDO

Computer Security | 2022 – 2023 (V2.7 *"S-81 Isaac Peral"*)

CONTENT

BEFORE YOU START

Why all this?

The COVID-19 health crisis has put us in a situation in which most of our teaching activities have to be made online or at home. This causes lots of problems, and one of the most important ones is that you must use your home hardware to run the software required to take this course. We know that the hardware capabilities can be widely different in this scenario. For that reason, all the laboratories in the course have been completely redesigned to dramatically decrease the hardware requirements: now you will only need to use a single virtual machine to run all of them. We are now using innovative infrastructure building and provisioning technologies, although studying them is out of scope of this security course. However, to counteract this, **we have provided you with scripts that allow you to work with them without needing to understand the underlying technology** (unless you are curious, as everything is commented in case you want to learn 😉)

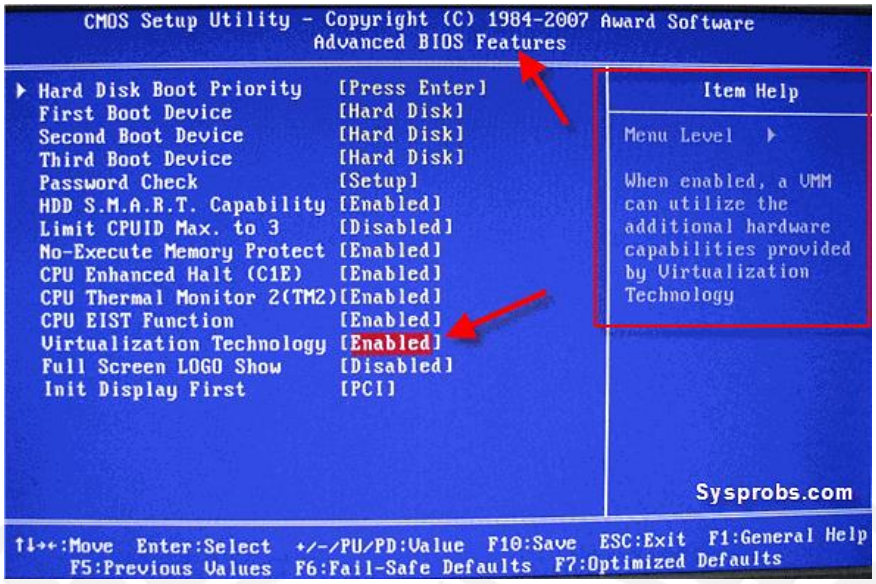
This document has only one goal: to give you options to **deploy the VM you need to follow the course at home**, no matter your virtualization system, OS, or hardware capabilities. You still need a bare minimum to follow the course:

- Any 64-bit CPU with hardware virtualization extensions (VT-X, AMD-V) (every CPU from 2012 onwards should fulfill this requirement)
- Possibility to assign 2 cores to the virtual machine (1 should also be possible, but at a substantial performance cost). We recommend not to use a GUI with 1 core only.
- Possibility to assign 2Gb of RAM to the virtual machine (1Gb might be viable for most labs, but it has not been thoroughly tested). **We strongly recommend not to use a GUI with 1Gb only.**
- A maximum of 80Gb for the VM hard disk (VM disk is dynamic, it will only take space if it is needed. Reaching 80Gb is highly unlikely)

We will provide you with three options, each one with advantages and disadvantages: please pick up the one that benefit you the most, although **we recommend the *Vagrant* one**.

Virtualization support in the host BIOS

In order to be able to virtualize machines it is necessary to **enable hardware virtualization support** in the BIOS of your computer. This option is located in different places depending on the BIOS manufacturer, but usually is in a group of advanced options and is described with tags similar to "*Intel Virtualization Technology*", "*Virtualization Technology*", "*Enable Virtualization Technology*", "*Enable VT-X*" or "*Enable AMD-V*". **We need you to make sure that this option is active**, especially since many manufacturers disable it by default in BIOS, and if you have not used it previously, it could be disabled. Without this option **you cannot run most modern virtualization software**. Fortunately, all *Intel* and *AMD* CPUs from 2010 onwards implement this technology.



Example of BIOS and hardware virtualization technology support (Source: https://support.bluestacks.com/hc/es/articles/115003910391--C%C3%B3mo-puedo-habilitar-la-virtualizaci%C3%B3n-VT-en-mi-PC-**)**

To GUI or not to GUI

The VM we supply or the one you will build has a lightweight GUI environment to facilitate your work in the course. This GUI is accessible typing `startx` in the command line. The decision of using a GUI or not is yours: you may be able to save time, but at a resource cost. If you prefer not to use a GUI (or your hardware does not allow you to have one) we give you a different option: using advanced terminal features. We are going to detail it more here, because they can be used no matter the option you choose, and most of these options are also present in the *Docker* containers we supply for each lab.

Using terminals

Even if we have servers without GUI to minimize resource usage, which does not mean that we will not be able to do multiple tasks at the same time. A user, once logged in to the system, can open multiple console terminals simply by pressing Alt-F1, Alt-F2, etc. If this is the first time a terminal associated with a function key has been opened, the user and password will be asked again. Next terminal changes will continue with the activity that was being done on it without having to be revalidated. This is especially useful, since it allows to perform tasks in parallel, for example, to edit and save a configuration on one terminal leaving it opened and have another to do restarts of the associated service.

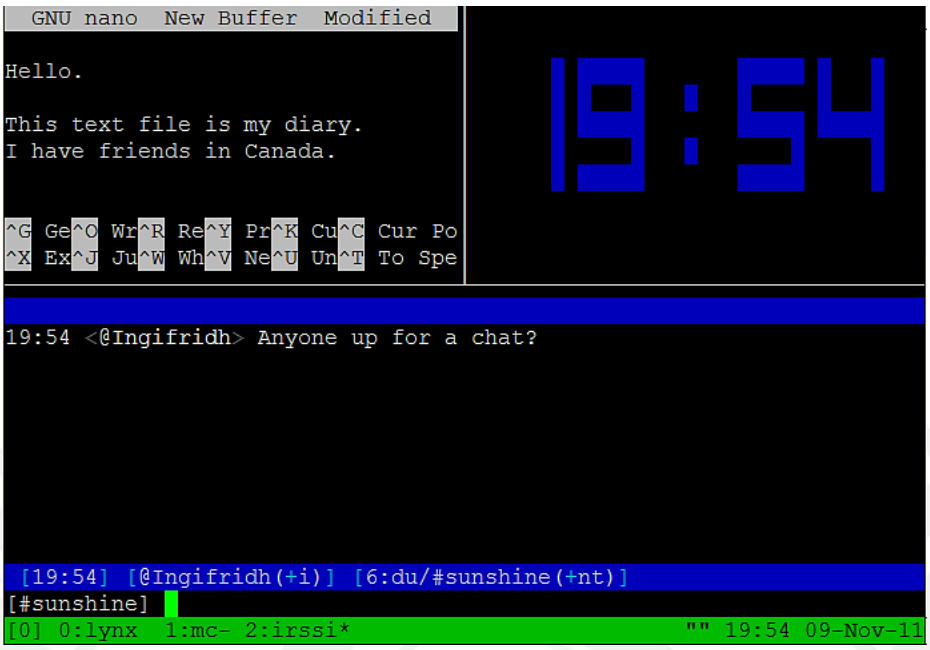
```
Ubuntu 18.04.5 LTS ssiserver tty1

ssiserver login: ssiuser_
```

New terminal, resulting from the Alt-F2 key sequence

TMux: Terminals and panels

`tmux` is a package that allows you to have multiple terminal windows in the same screen and divide a window in different panes that may work concurrently. `tmux` is preinstalled, with mouse integration (`gpm` module) and terminal color support, in the *Vagrant* machine, in the `.ova` one, and in every interactive container of the labs. It could be a nice feature to add if you want more agility and productivity without installing or going to a GUI,

as it gives you part of the features of a GUI with far less resource consumption. It may be also useful in other courses that force you to heavily use the terminal.



Tmux running three panels with different applications (Source: *Wikipedia*)

This cheatsheet shows you the most typical options of `tmux`. If you decide to create the VM yourself, the files of the *Vagrant* VM contain a `tmux.conf` file with another nice starting configuration that you can copy to `/etc/tmux.conf` (it also provides a nice default `nano` configuration that you can copy to `<your home directory>/.nanorc`)

TMux 3.1 Cheatsheet (ingenieriainformatica.uniovi.es)

Console enhancer and multi-pane support tool

https://github.com/tmux/tmux/wiki

GENERAL USAGE

```
tmux
```

NOTES

Press Ctrl + B, release them, and then press the corresponding key
exit on the last pane exits tmux

OPTIONS

Window handling

New window: CTRL+b, release and then c
Next window: CTRL+b, release and then n
List all windows: CTRL+b, release and then w
Change window name: CTRL+b, release and then ,

Panel management

Vertical: CTRL+b, release and then %
Horizontal: CTRL+b, release and then " (press Uppercase key)
Hide panel: CTRL+b, release and then x
Show panel number: CTRL+b, release and then q
Change panel: CTRL+b, release and then the corresponding arrow key
Close current panel : exit

Copy and paste text

- **CTRL+b, release and then [**: Enter "copy" mode.
- Move to start/end of text to highlight.
- **CTRL + space** and start highlighting text. Selected text change color. Move to opposite end of text to copy.
- **ALT + w**: Copies selected text into tmux clipboard. (On Mac use Esc+w.)
- Go to the destination pane and windows and put the cursor where you want to paste the text.
- **CTRL+b, release and then]**: Paste copied text from tmux clipboard

```
:00:00 UTC; path=/;";document.cookie="_gat=; expires=Thu, 01 Jan 1970 00:00:00 U
TC; path=/;"}window.cookieconsent.initialise({palette:{popup:{background:"#edeff
5",text:"#838391"},button:{background:"#4b81e8"}},type:"opt-in",content:{message
:"Nuestra web utiliza cookies propias y de terceros para analizar sus h bitos de
 navegaci n y elaborar estad sticas sobre su interacci n con nuestro sitio web,
as_ como mostrar botones de compartici n de contenido en redes sociales. Puede o
btener m s informaci n en nuestra Pol tica de Cookies.",dismiss:"Descartar",allo
w:"Aceptar",deny:"Rechazar",link:"M s informaci n",href:"/politicacookies",polic
y:"Pol tica de Cookies"}})});/*]]>*/</script> </body> </html> root@e75b0892bfc
1:/#
root@e75b0892bfc1:/#

64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=50 ttl=113 time=18.4 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=51 ttl=113 time=18.9 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=52 ttl=113 time=18.6 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=53 ttl=113 time=19.0 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=54 ttl=113 time=18.2 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=55 ttl=113 time=19.0 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=56 ttl=113 time=18.6 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=57 ttl=113 time=18.8 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=58 ttl=113 time=18.1 ms
64 bytes from 172.217.17.4 (172.217.17.4): icmp_seq=59 ttl=113 time=18.3 ms

[0] 0:bash- 1:bash*                                  "e75b0892bfc1" 10:43 23-Dec-20
```

MORE INFORMATION

http://www.sromero.org/wiki/linux/aplicaciones/tmux
https://www.hostinger.es/tutoriales/usar-tmux-cheat-sheet/

Nice default configuration kindly provided by Alejandro Saez Morollón (put this in /etc/tmux.conf to apply it globally):

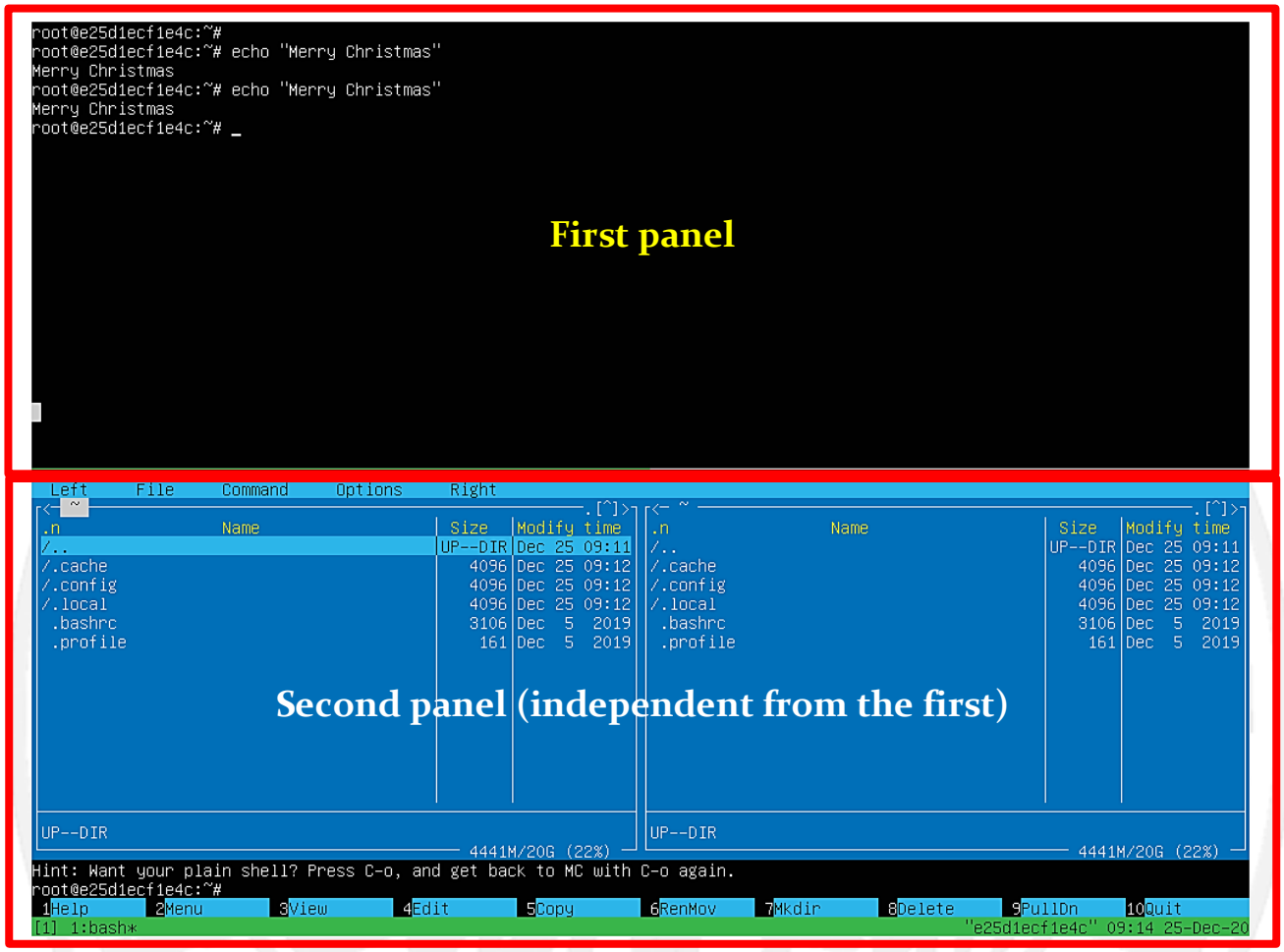
```
# Start windows and panes at 1, not 0
set -g base-index 1
setw -g pane-base-index 1
# Tmux fix for terminal colours
set -g default-terminal "screen-256color"
# Open new pane or window in the current directory
bind '"' split-window -c "#{pane_current_path}"
bind % split-window -h -c "#{pane_current_path}"
bind c new-window -c "#{pane_current_path}"
# Avoid rename
set -g allow-rename off
# if run as "tmux attach", create a session if one does not already exist
new-session -A -s main
```

Another **tmux** cheatsheet is this one:

Tmux Cheatsheet. Source: https://acloudguru.com/blog/engineering/tmux-cheat-sheet

Midnight Commander: File Browser

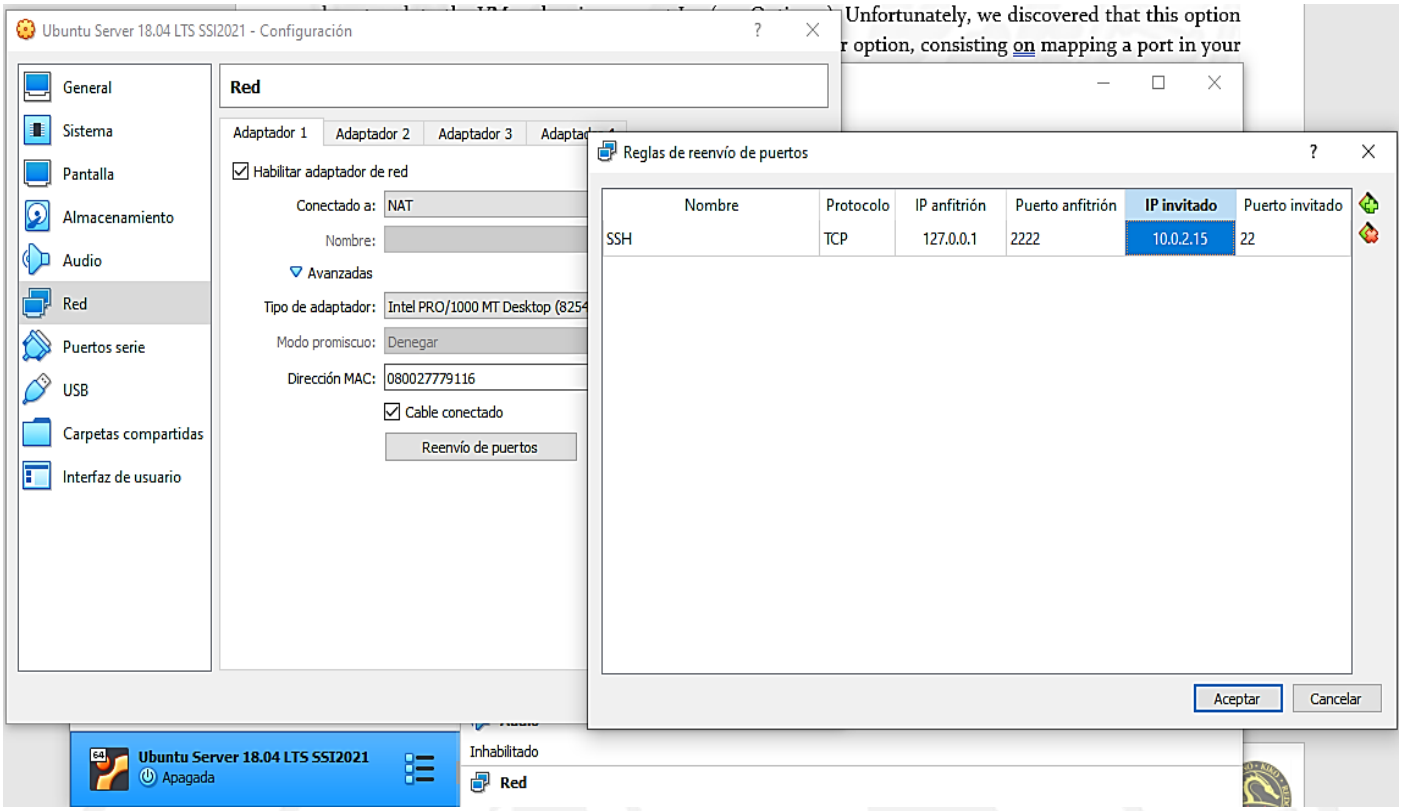
One of the most missed options when you do not have a GUI is to be able to easily browse and view files in the hard disk. We have provided you with this popular console-based file browser to give you this ability and increase your productivity. You can run it just typing `mc`. As this image shows, it can be integrated with `tmux` and has mouse support. This is also installed in all interactive *Docker* containers that we are going to use in the course.



SSH to the VM

If you use a GUI with our *Vagrant* or supplied `.ova` VM, it is not needed to have an SSH connection as the VM fully supports copy and paste text to/from your PC. However, a direct way of connecting to your machine with advanced SSH client such as *MobaXTerm* is to create a host-only network to the VM and assign correct IPs (see Option 3). VMs obtained from option 1 and 2 map the `2222` port of your PC to the `ssh` port of the VM, so if your *MobaXTerm* / *Putty* / any SSH client connects to `127.0.0.1:2222`, it will connect to your VM instead without requiring more steps.

Only manually created machines require to do the following redirection in *VirtualBox*. You need to access the VM configuration, go to *Network*, choose the *NAT* interface, click on *Advanced*, port redirection and configure the settings you see in the image. If you do this for multiple VM, please use different host ports.



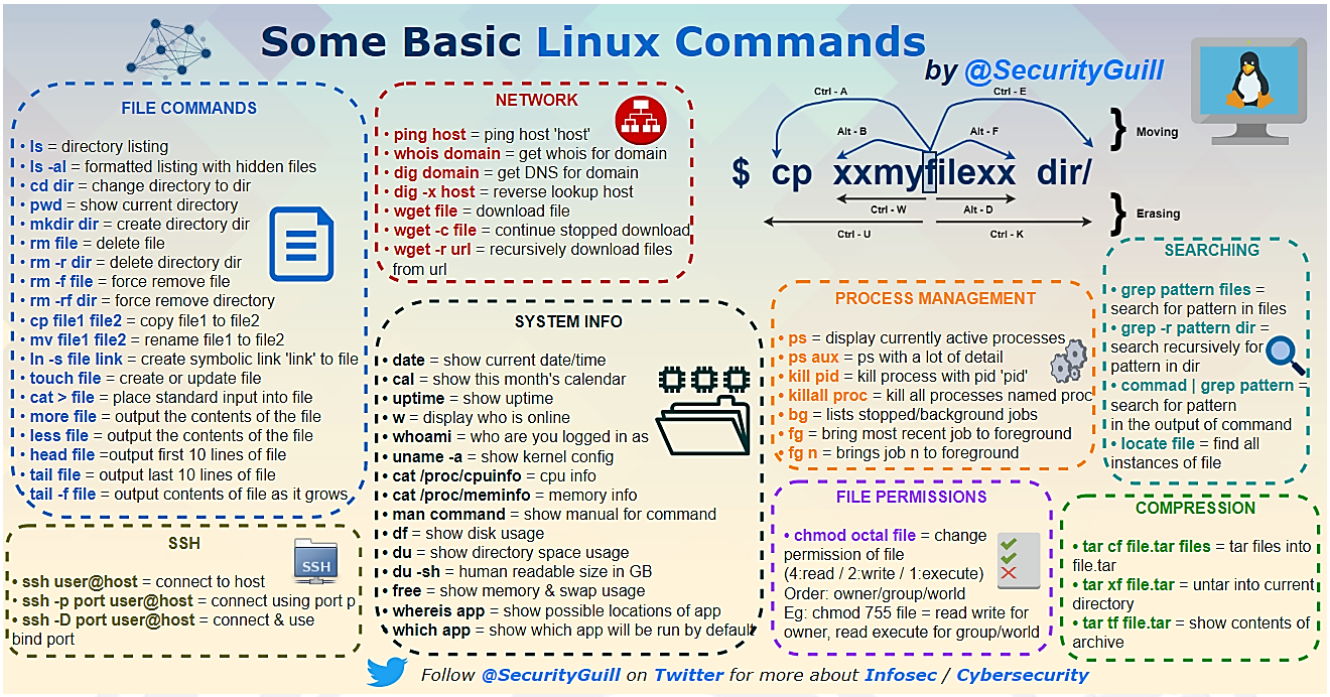
Editing text files

You will need to edit files in the console multiple times unless you decide to use the GUI. To facilitate your work, we have created a special `nano` configuration that facilitates this type of work, converting tabs to whitespaces, using better colors, displaying the cursor position, line numbers, and performing automatic file backups of the ones you edit. You can find this configuration in the `.nanorc` file of the default user profile of the prebuild VM or as part of the *Vagrant* files if you want to use it in other courses or for your personal use. Machines of options 1 and 2 include the `mousepad` and the *Visual Studio Code* editors (recommended option) to edit files via GUI.

Linux Knowledge

Yes, you require *Linux* command-line knowledge to deal with this course. No more than the one you have acquired through the operating systems course though. However, we know that you may have forgot some of them. Do not worry, we assume this and implemented a slow (but progressive) start in this matter. To help you we recommend the following:

- Use this cheatsheet as a guideline to remember the typical commands you will need. Do not forget that the command `ifconfig` gives you your current network cards names and IPs:



- If you have absolutely no clue about the command you should use, run `apropos -a <text to search>` to let the system suggest you commands that may do what you want. For example:

```
redondo@redondo-VirtualBox:~$ apropos -a create directory
docker-plugin-create (1) - Create a plugin from a rootfs and configuration. P...
hpmkdir (1)          - create a directory on an HFS+ volume
mkdir (2)            - create a directory
mkdirat (2)          - create a directory
mkdtemp (3)          - create a unique temporary directory
mkfontdir (1)        - create an index of X font files in a directory
mklost+found (8)     - create a lost+found directory on a mounted Linux secon...
mktemp (1)           - create a temporary file or directory
pam_mkhomedir (8)    - PAM module to create users home directory
update-info-dir (8)  - update or create index file from all installed info fi...
```

OPTIONS FOR CREATING THE COURSE VIRTUAL MACHINE

Option 1: Importing the supplied .ova image (easy)

Why picking up this method?

Recommended if: You have *VirtualBox* installed, a stable *Internet* connection, do not want to test *Infrastructure as Code* technologies to experiment what they can do for you in the future.

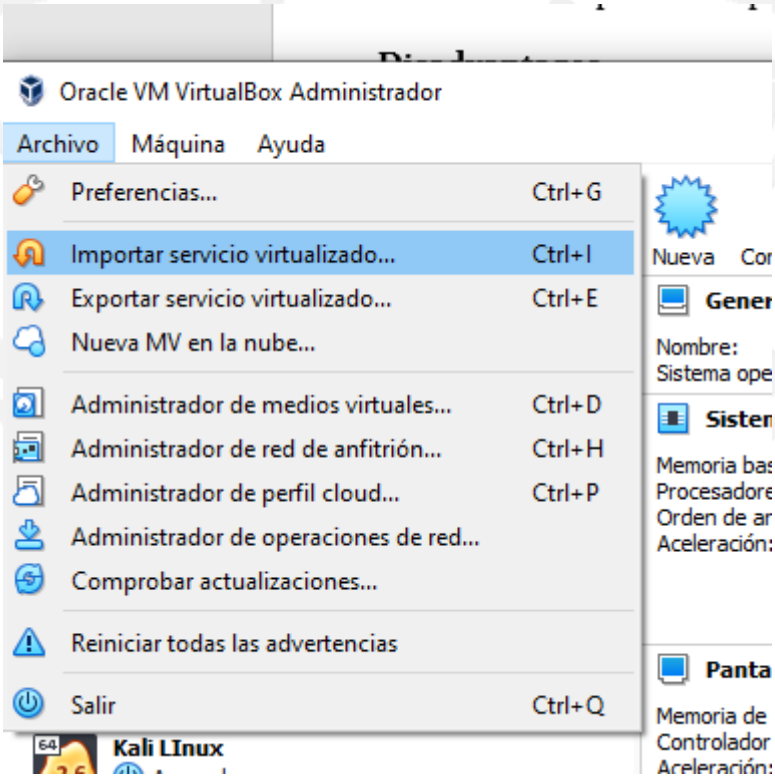
Advantages

- Ready to use preconfigured VM with all the packages installed to make the course laboratories.
- One-step VM set up (**user**: `ssiuser`, **password**: `test123…` (three dots at the end)).

Disadvantages

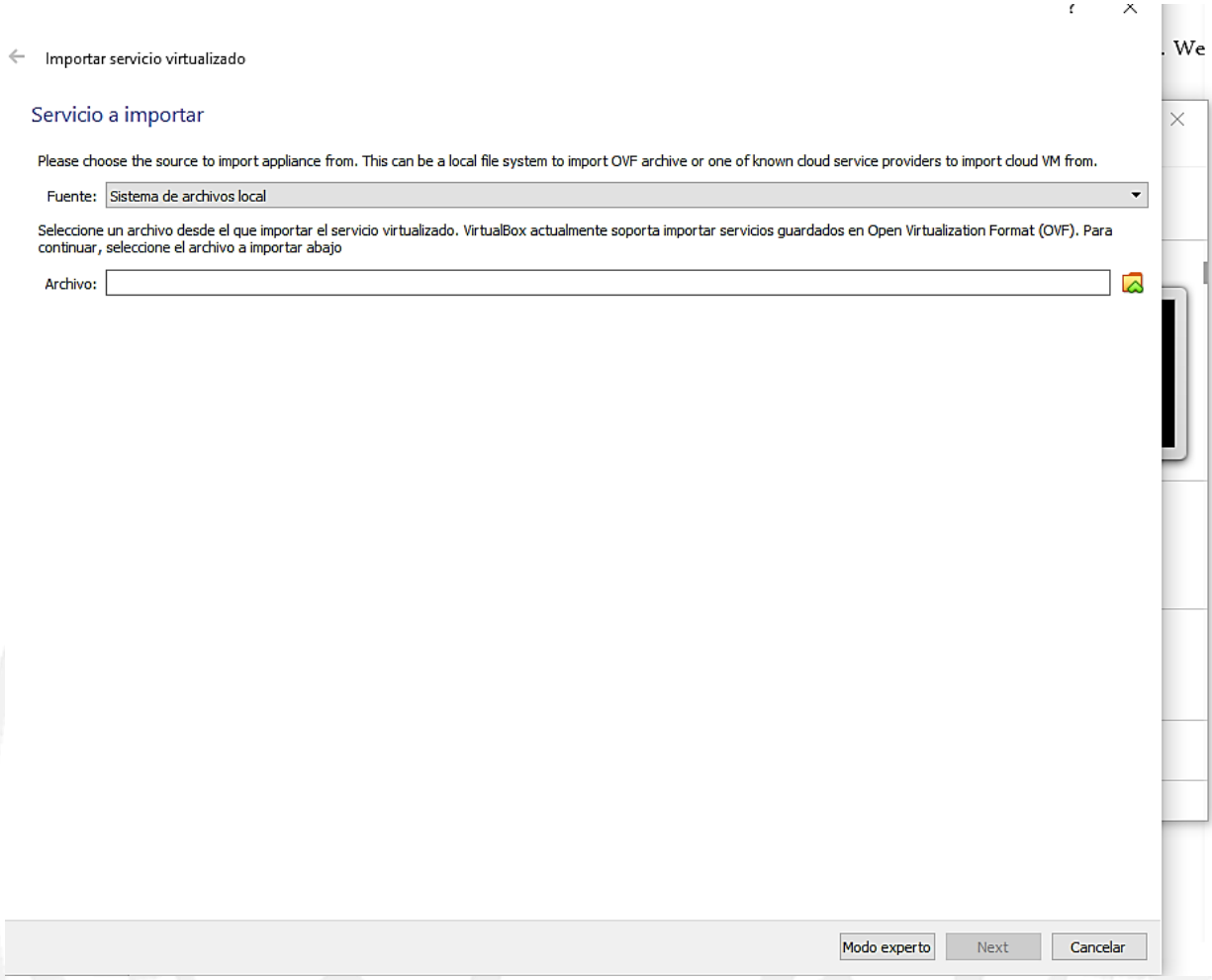
- Large download that may not be possible in certain situations. The VM has been specially shrank to minimize its size, tough
- You must manually create a shared folder with the host (consult the last section of this document).
- Prepared to run only with *VirtualBox*, other virtualization systems may not be able to use `.ova` files. We provide you with two import examples, but they are not guaranteed to work:
  - **Importing a `.ova` with *VMWare***: https://docs.vmware.com/es/VMware-Fusion/11/com.vmware.fusion.using.doc/GUID-275EF202-CF74-43BF-A9E9-351488E16030.html
  - **Importing a `.ova` with *Hyper-V***: https://superuser.com/questions/1133256/convert-ova-to-vhd-for-usage-in-hyper-v (please go to the Option 3 to know the VM characteristics you must create).

To facilitate your work during the whole course, we provided you **a prebuilt virtual machine**. You can download it from a URL that your teacher will provide. The machine is a console-based (with optional GUI) **Ubuntu Server 18.04 LTS** system. To import it, download the `.ova` file and use this VirtualBox option:



Importing an `.ova` file option

Choose the downloaded **.ova** file, click next, and follow the instructions to import the VM. It will take a while. An SSD is recommended to dramatically decrease import times, boot, and general responsiveness, but it is not required.



Selection of the .ova file screen

Further options to manage the imported VM will be detailed in the next section, as this machine is identical as the one provided through *Vagrant*.

Option 2: Use *Vagrant* to Auto-Build a Virtual Machine (easy-medium, recommended)

Recommended if: You have *VirtualBox, Hyper-V* or *VMWare Desktop* installed, want to test *Infrastructure as Code* technologies to experiment what they can do for you in the future.

Why picking up this method?

Advantages

- Machine download, configuration, creation, and setup is fully automated, no interaction is needed.
- It normally creates a VM which uses resources more optimally than a default one.
- Uses advanced virtualization techniques not available through the *VirtualBox GUI* or that require additional advanced configuration not easily done unless you have some experience.
- Runs in any platform compatible with *Vagrant* (but requires *VirtualBox, Hyper-V* or *VMWare Desktop* installed, although the virtualization system will be automatically chosen)
- Machine configuration can be distributed (and deployed to different systems) in small text files, being always the same and very easily updatable.
- A shared folder is automatically created in the same directory where the machine is built.

Disadvantages

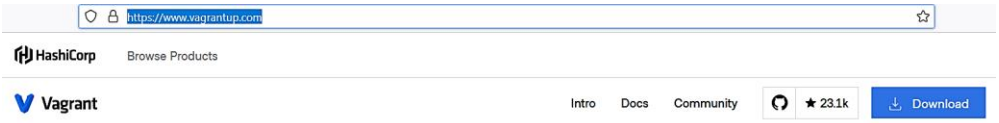
- Using *VirtualBox* is preferred, as it is the main virtualization system supported by the vendor. However, *Vagrant*, the base official *Ubuntu* box we use, and the files we provide do also support *Hyper-V*)[1]
- Requires a long initialization time (only in its first execution)
- Requires you to install extra software in your computer (*Vagrant*). Lab computers in our school have *Vagrant* installed.

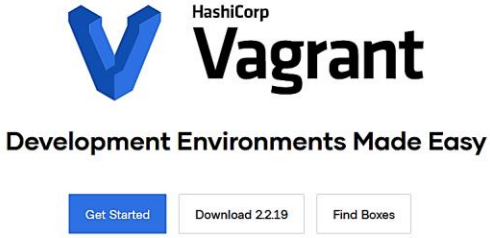
How to use the Vagrant files

VM building preparation

- Download the `.zip` file from the virtual campus with the *Vagrant* files
- Ensure that the virtualization platform you are going to use is correctly installed
- Install the most up to date *Vagrant* version for your OS: https://www.vagrantup.com/downloads
    - For *Ubuntu*, please use the downloaded package from the official webpage, *Ubuntu* repositories have an outdated *Vagrant* version that does not work with the provided files: https://linuxize.com/post/how-to-install-vagrant-on-ubuntu-18-04/

[1] The `Vagrantfile` has an entry for a 3rd provider, `vmware_desktop`. This technically works, but it was not tested and has no optimization. We only provide this because the base *Vagrant* box supports it, but we strongly recommend you to use other virtualization options





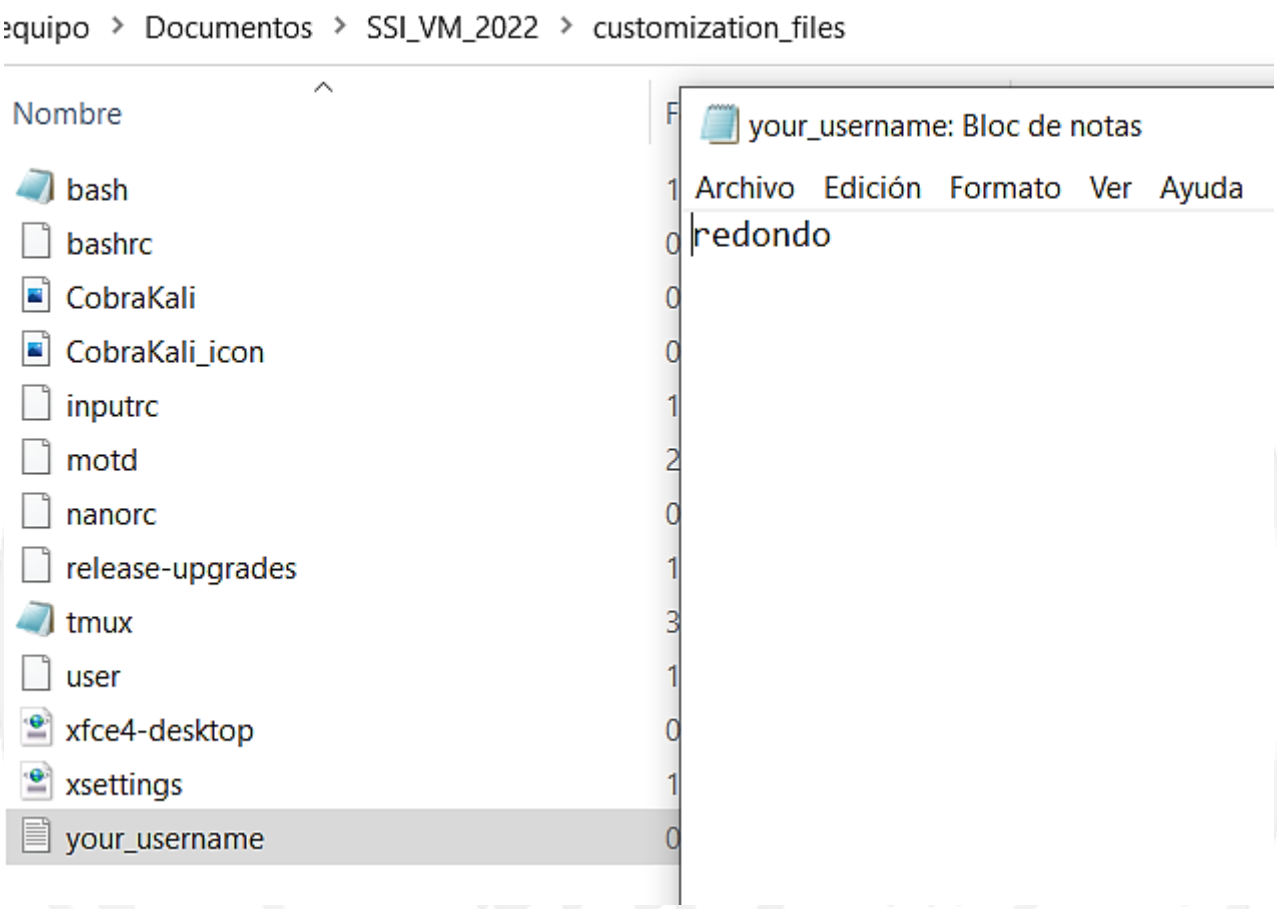
- Unzip the file and place their contents in a folder you create (for example, `ssiubuntu`). These contents are:

.vagrant	08/12/2021 12:49	Carpeta de archivos	
customization_files	28/12/2021 7:33	Carpeta de archivos	
customization_scripts	10/12/2021 9:55	Carpeta de archivos	
provisioners	08/12/2021 12:41	Carpeta de archivos	
shared	21/12/2021 13:41	Carpeta de archivos	
buildvm	13/11/2021 10:10	Archivo por lotes ...	1 KB
buildvm	13/11/2021 10:10	Archivo SH	1 KB
destroyvm	13/11/2021 10:10	Archivo por lotes ...	1 KB
destroyvm	13/11/2021 10:10	Archivo SH	1 KB
runvm	25/10/2021 9:54	Archivo por lotes ...	1 KB
runvm	25/10/2021 9:54	Archivo SH	1 KB
updatevm	13/11/2021 10:11	Archivo por lotes ...	1 KB
updatevm	13/11/2021 10:11	Archivo SH	1 KB
Vagrantfile	28/12/2021 7:41	Archivo	8 KB

  - **The `Vagrantfile`**: With all the configuration of the VM you are about to create.
  - **Shell scripts to manage *Vagrant*** as standard *Linux* bash files. A *Windows*-compatible `.bat` version is also provided.
    - `buildvm.sh`: Build the VM from the `Vagrantfile`. Please note that this should be done only once.
    - `runvm.sh`: Runs the previously build VM from the command line if from some reason you cannot run it from the *VirtualBox* GUI.
    - `updatevm.sh`: If you update the `Vagrantfile` (for example install more packages by default, increase the available RAM, number of cores...) this script will update the VM configuration without performing a full rebuilt (much faster). Note that you do not need to do that in this course, it is just provided for your convenience.
    - `destroyvm.sh:` Deletes a previously build VM, freeing all the space and lose all changes and the results of your operations. If you want to use the VM, you need to build it again.

- Rest of the files and folders are used by the previous one and you are not required to understand them to follow this course (although if you are curious…they are fully commented 😊)

- Once you have all files unzipped, go to the `customization_files` subfolder and locate the `your_username.txt` file. Change its contents so you put your own username (UO or whatever you like), **provided it is a valid *Linux* username**. An account for this user (password `text123...`) will be created for you in the VM. An additional `vagrant` user (also with password `test123…`) exist in the VM that will be created.



- You can now proceed to build the virtual machine. However, if your PC has enough power, you can increase the VM resources to work more efficiently. Open the `Vagrantfile` and locate this section in the file. 2 cores and 2Gb of RAM are the minimum requirements, but if you can give at least 4Gb of RAM it will be nice.

```
## Machine Characteristics:
## ------------------------

# In Mb, if you can afford it, more memory is always welcome :). Be careful not to push this too hard, and turn yourself into a
Chrome browser :P
RAM = 2048

# If you can afford it, more cores are always welcome :). Be careful not to push this too hard, and let your host breathe :P
CORES = 2

# Size of the box disk
DISK_SIZE = "80GB"

# Amount of available VRAM. Consider if your VM is going to have a GUI or not to choose this.
VRAM = 128
```

  - **NOTE FOR MAC USERS ONLY**: One of the advanced features of this `Vagrantfile`, *Page Fusion*, may not supported in MAC hosts. Please, if you use a MAC to deploy the VM via *Vagrant*, open

de `Vagrantfile` and comment or delete the line that has the `modifyvm` option with the `--pagefusion` parameter. No other change should be necessary.

Vagrantfile: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
    # Use VBoxManage to customize the VM for high-performance options and other things
    vb.customize ["modifyvm", :id, "--acpi", "on"]
    vb.customize ["modifyvm", :id, "--accelerate3d", VIDEO_3D]
    vb.customize ["modifyvm", :id, "--apic", "on"]
    vb.customize ["modifyvm", :id, "--biosapic", "x2apic"]
    vb.customize ["modifyvm", :id, "--cpu-profile", "host"]
    vb.customize ["modifyvm", :id, "--graphicscontroller", "vmsvga"]
    vb.customize ["modifyvm", :id, "--hpet", "on"]
    vb.customize ["modifyvm", :id, "--hwvirtex", "on"]
    vb.customize ["modifyvm", :id, "--largepages", "on"]
    vb.customize ["modifyvm", :id, "--longmode", "on"]
    vb.customize ["modifyvm", :id, "--nestedpaging", "on"]
    vb.customize ["modifyvm", :id, "--pae", "on"]
    # If you have a MAC host and get an error, comment this line, as this feature is not currently supported on MAC (yet)
    vb.customize ["modifyvm", :id, "--pagefusion", "on"]
    vb.customize ["modifyvm", :id, "--rtcuseutc", "on"]
    vb.customize ["modifyvm", :id, "--spec-ctrl", "off"]
    vb.customize ["modifyvm", :id, "--x2apic", "on"]
    vb.customize ["modifyvm", :id, "--vram", VRAM]
    vb.customize ["modifyvm", :id, "--vtxux", "on"]
    vb.customize ["modifyvm", :id, "--vtxvpid", "on"]
    vb.customize ["modifyvm", :id, "--vrde", VRDE]
    vb.customize ["storagectl", :id, "--name", HD_CONTROLLER, "--hostiocache", "on"]

    # Shared folder (host path, guest path)
    config.vm.synced_folder "./shared", "/host_data", create: true, automount: true
  end
```

VM Building process

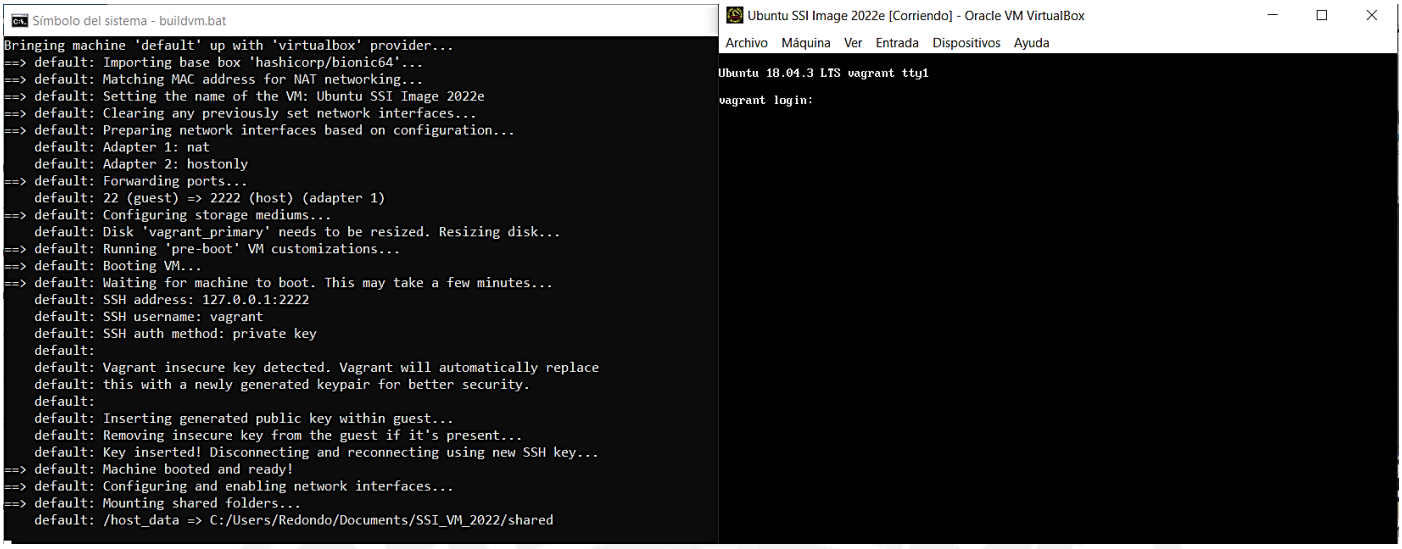
Once these preparations are finished, you only need to run `buildvm.bat` (or `buildvm.sh` if you are not in a *Windows* OS) and wait for the VM to be built.

Símbolo del sistema - buildvm.bat

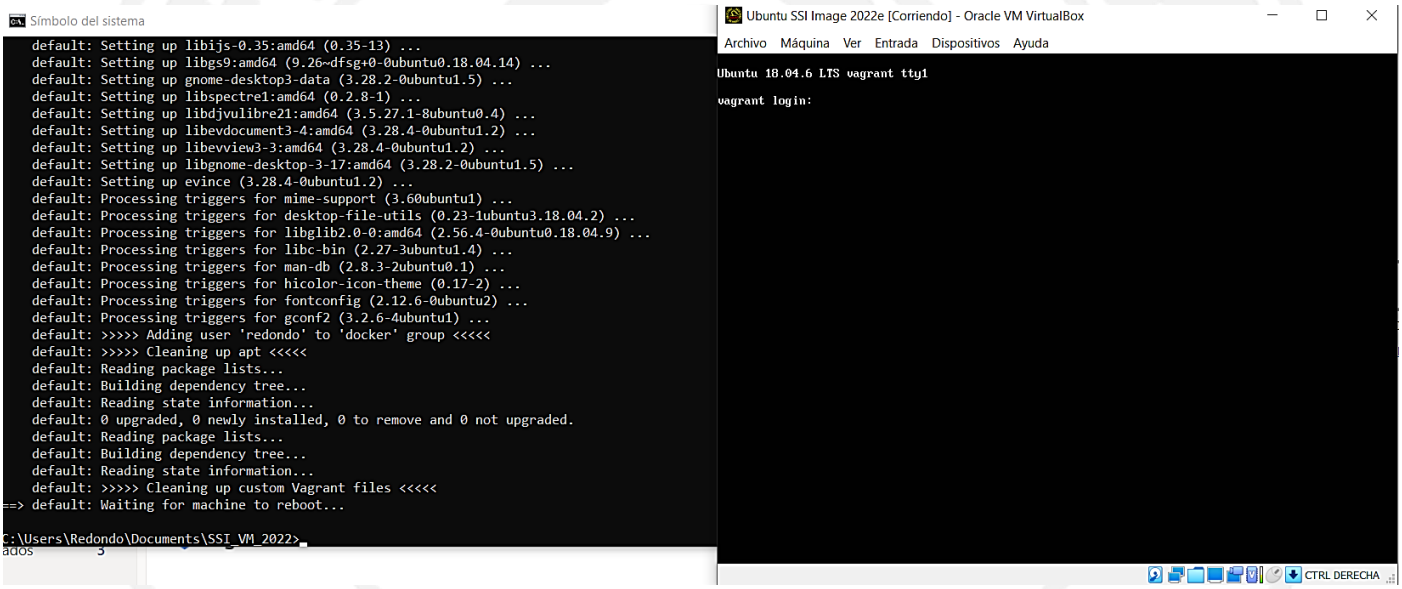
```
C:\Users\Redondo\Documents\SSI_VM_2022>buildvm.bat

C:\Users\Redondo\Documents\SSI_VM_2022>vagrant up
==> vagrant: You have requested to enabled the experimental flag with the following features:
==> vagrant:
==> vagrant: Features:  disks
==> vagrant:
==> vagrant: Please use with caution, as some of the features may not be fully
==> vagrant: functional yet.
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'hashicorp/bionic64'...
```

Make sure you have network connection during the entire process. Do not open the VirtualBox GUI while doing this!

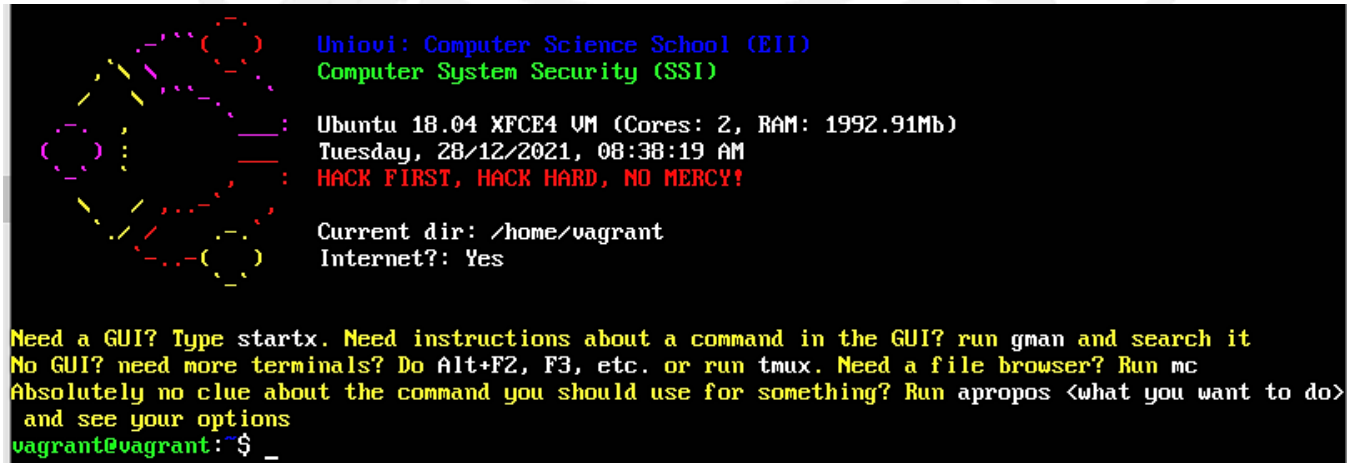
The VM will start almost immediately, **but please wait for the script to finish and do not login yet** (the VM will reboot automatically). You can use the VM right now, but it lacks all the features you need. The VM updates in the background, installing all necessary software and updates while allowing you to use it, but as it needs to reboot at the end, you may lose your work. Additionally, *Docker*, *Docker-Compose* and other necessary software will not be available unless you let the build process to finish.



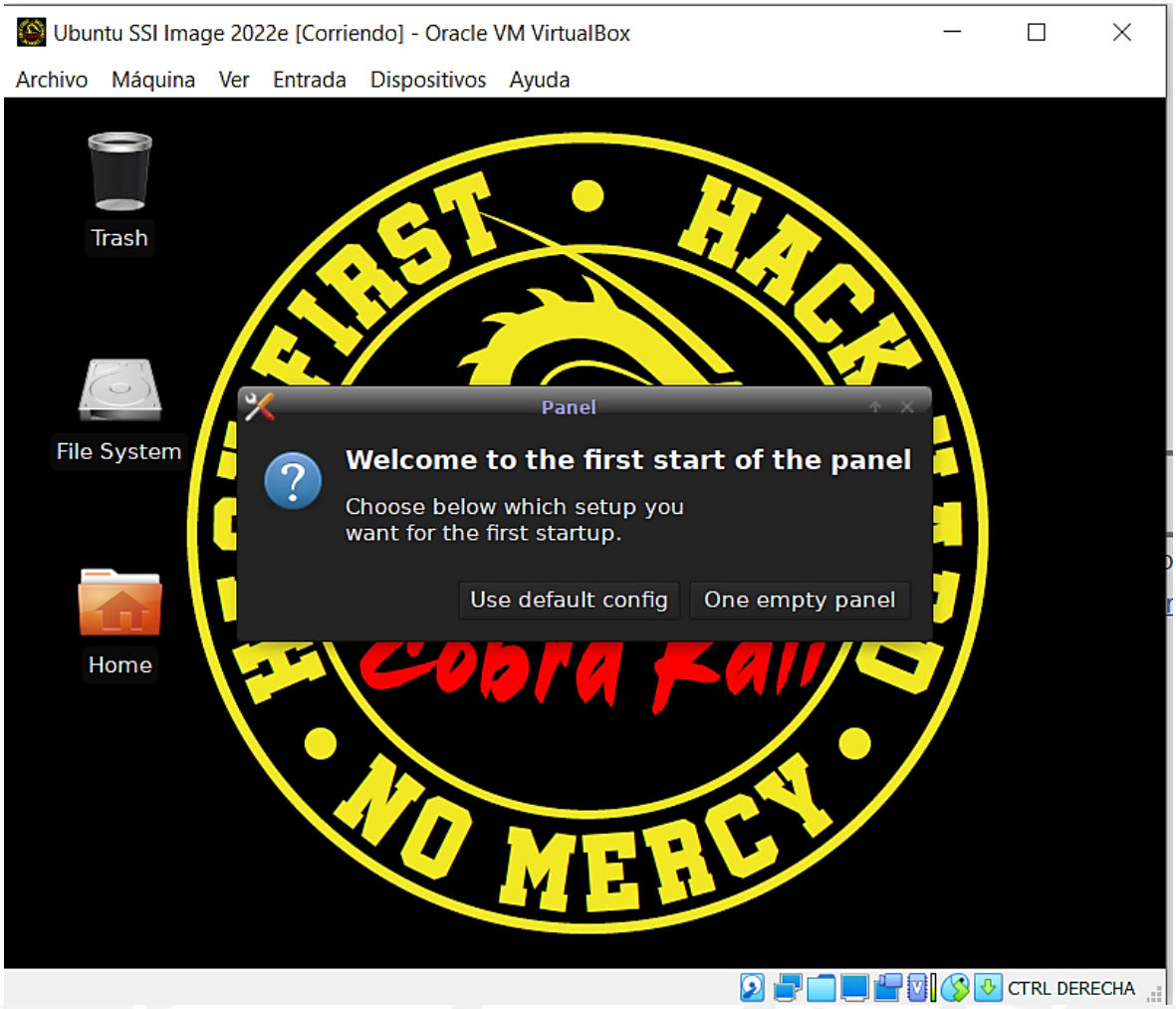
Once the script finishes the machine reboots and you are greeted with the login screen (please note the *Ubuntu* version update from the previous screen). Now you can use the VM with your user or with the `vagrant` user (both have the same `test123...` password).



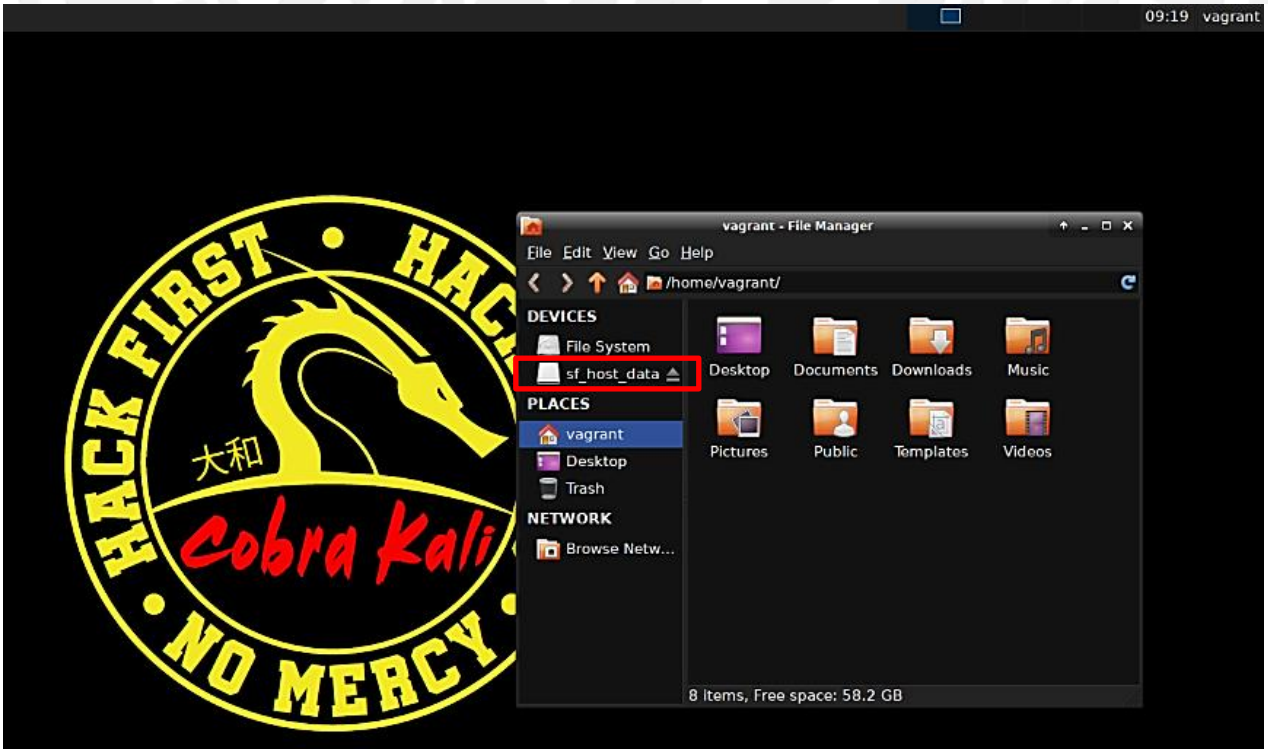
Successful login will present you this welcome screen with basic instructions (a reminder of what we said in the first section of this document for your convenience)



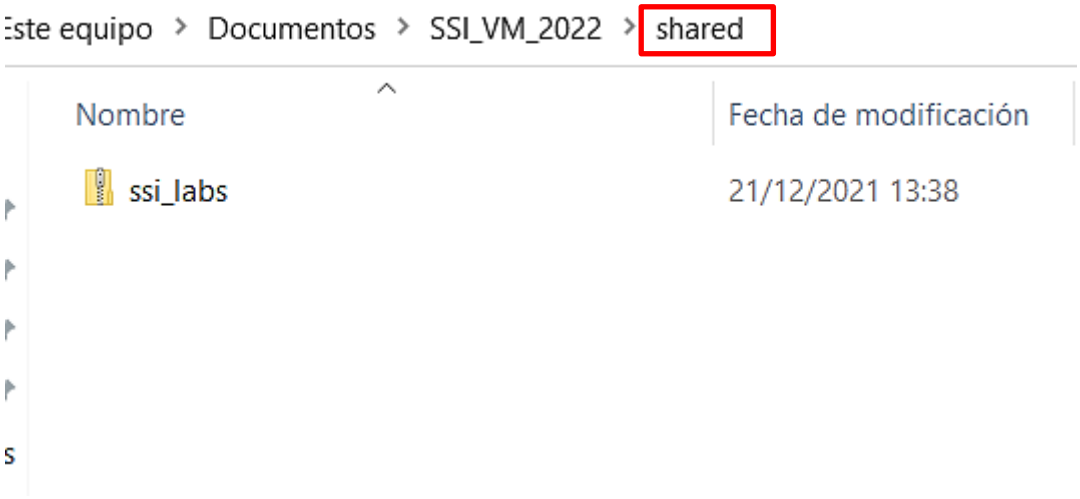
If you like to use a GUI, run `startx` as indicated in the welcome screen and accept the default config. You can now maximize the screen to better use the VM:



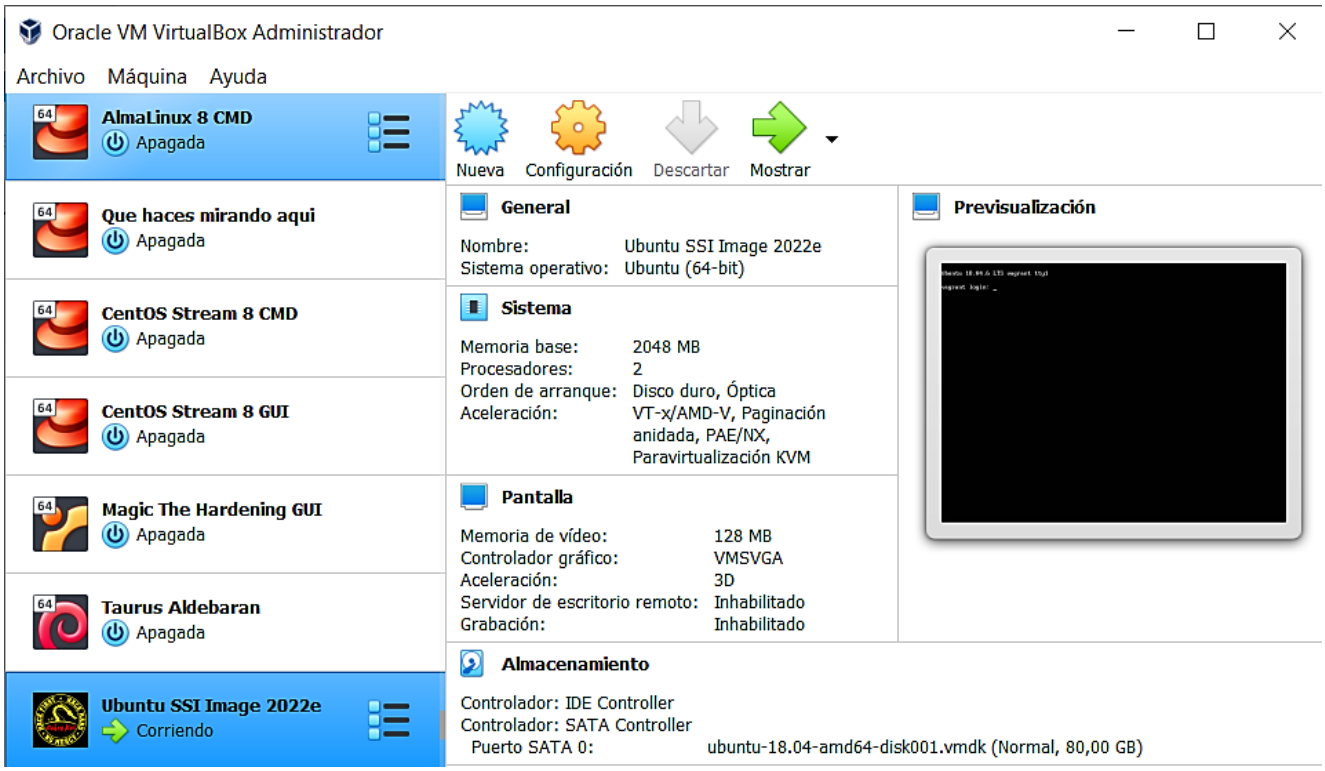
If now you open a file explorer, you will notice that there is a special folder called `sf_host_data`:



Everything you write here will be written in this subfolder of the path where you unzipped the VM files in your PC (and vice versa!). This ensures that transferring files between the VM and your PC is quite easy!



As you notice, the lab materials are already present there, you need to follow the instructions of the next document "*LabooB. SSI Automated Lab Infrastructures*" to know how to use them. Also, note that booting the VM again does not require following this process. The VM will appear in your virtualization system GUI as a "normal" one! You can also obtain SSH access from the host to the running VM with `ssh vagrant@127.0.0.1 -p 2222`.

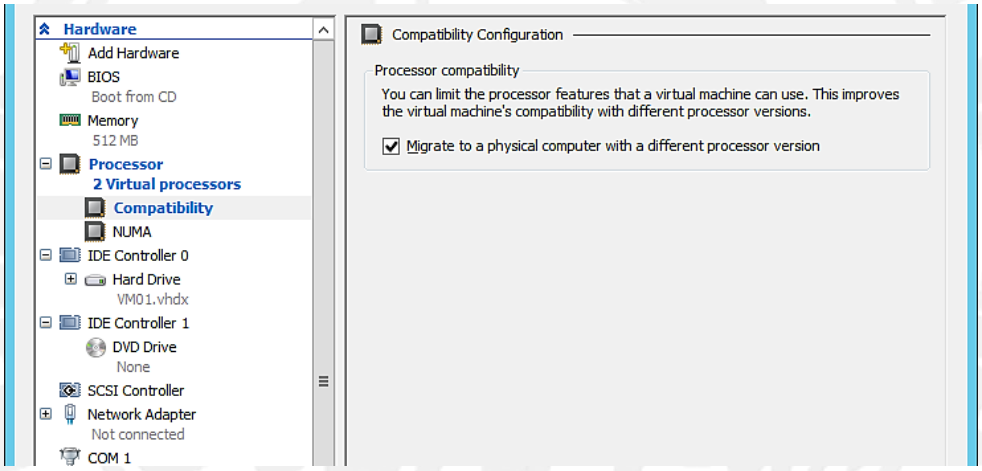


Troubleshooting

Additional notes if you run Vagrant on Windows OS and use Hyper-V

Using *Vagrant* and *VirtualBox* should work straightforward from the provided `Vagrantfile`. However, you could have *Hyper-V* installed in *Windows* instead of *VirtualBox*. The provided `Vagrantfile` is created to automatically detect this and use *Hyper-V* custom configuration instead of the default *VirtualBox* one. Nonetheless, *Hyper-V* support in *Vagrant* is not as mature as the *VirtualBox* one, and the following problems are likely to happen. Below, we show you how to address them:

- You need to run the `buildvm.bat`, etc scripts from a command line that has been **opened as an Administrator.**
- Once you launch the build process, it will likely fail complaining from two things. If that is your case, run it again and the process should complete successfully without further error. These things are:
  - Unable to resize the VM disk.
  - Specify a password for the SMB shared folders even if they have been disabled (in this case just provide an empty user and password until the process finishes and run the build again).
- Once the VM is launched, the mouse may be slow, or its response may lag. If that is you case, please check this option in the VM *Configuration* option while the VM is shut down on the *Hyper-V* GUI. This should fix the problem permanently.



***Hyper-V* CPU compatibility option to ensure proper VM behavior**

You are running VirtualBox, but the provided Vagrantfile gives error during machine build

If your system is unable to run the `Vagrantfile` due to errors related to *VirtualBox* configuration, you can do these steps:

- Ensure you have the latest *Vagrant* version installed.
- **Update your *VirtualBox*** to the newest version: We are using advanced *VirtualBox* options to ensure the best possible performance for the VM, and some may be incompatible with old versions. Last year this solved almost 100% of the problems.
- If the error shows the *VirtualBox* option that gives the error, you can open the `Vagrantfile` and comment this option only, trying to build the VM again.
- If this also does not work, we recommend you to follow the other options to create the VM.
- There is a known incompatibility between the *Kaspersky* antivirus and the latest *Vagrant* version, as the antivirus SSL monitoring module interferes with the connection to the *Vagrant Cloud* to extract base VM images and throws a certificate validation-related error. The only known solution is to make an exception in the antivirus to the *Vagrant* address shown in the error, or just using the Option 1.

- Some *VirtualBox* installations updated from a previous versions experience problems configuring host-only interfaces. In rare occasions you may find *Vagrant* telling you this error

```
There was an error while executing `VBoxManage`, a CLI used by Vagrant
for controlling VirtualBox. The command and stderr is shown below.

Command: ["startvm", "d04f893a-4165-426f-b94c-3599a1471dc6", "--type", "gui"]

Stderr: VBoxManage.exe: error: Failed to open/create the internal network
'HostInterfaceNetworking-VirtualBox Host-Only Ethernet Adapter' (VERR_INTNET_FLT_IF_NOT_FOUND).
VBoxManage.exe: error: Failed to attach the network LUN (VERR_INTNET_FLT_IF_NOT_FOUND)
VBoxManage.exe: error: Details: code E_FAIL (0x80004005), component ConsoleWrap, interface
IConsole
```

The only known solution is to open the `Vagrantfile` and remove or comment this line:

```
config.vm.network "private_network", ip: "192.168.56.10", adapter: 2
```

Even if you lose the host-only connection doing this, you can connect to the VM using SSH anyway by doing `ssh localhost:2222`.

Other questions

- **I only have *VMWare Desktop* as a virtualization software**: Sorry, but this software cannot be obtained for free, so we do not provide a tested `Vagrantfile` for it. However, as the base *Vagrant* box we use is compatible with this virtualization software, in theory you can use `Vagrantfile` we provide with it. If you want to do it because you do not want to install *VirtualBox*, and succeed optimizing the resulting VM, donations of your configuration will be appreciated 😊. However, in this case we strongly recommend other VM creation options.
- **I have a MAC and *Parallels* only, I do not want to install any more virtualization software**: unfortunately, the base *Vagrant* box we use is not compatible with *Parallels* now. There are other similar (*Ubuntu 18.04*) *Vagrant* boxes available compatible with *Parallels* (example: https://app.vagrantup.com/generic/boxes/ubuntu1804) and a guide on how to use *Parallels* with *Vagrant* (https://kb.parallels.com/en/122843), but this configuration could not be tested and hence is not recommended for this course. If you want to do it because you do not want to install *VirtualBox*, and succeed creating and optimizing the resulting VM, donations of your configuration will be appreciated 😊. However, in this case we strongly recommend other VM creation options.
- **I only have *libvirt* as a virtualization provide and I don not want to install any other**: The same as *Parallels*. Support for *libvirt* is not official and still experimental, but maybe it is doable. We do not recommend following this route. Once again, if you want to do it because you do not want to install *VirtualBox*, and succeed optimizing the resulting VM, donations of your configuration will be appreciated 😊. However, in this case we strongly recommend other VM creation options.

Option 3: Creating your own VM (Hard, not recommended)

Recommended if: You have any other hypervisor from the ones we mentioned, you want full control of the VM creation process, you have experience dealing with VMs.

Why picking up this method?

Advantages

- You have the control of everything you create
- You can use any virtualization system on any OS. It is advisable (but not required) to use the same credentials than the preconfigured VM (**user**: `ssiuser`, **password**: `test123…`) to avoid confusions.

Disadvantages

- Takes much more time to complete the set up.
- Requires technical knowledge, and you need to follow the steps on this section. It is not automatable.

Configuring a Virtualization Platform to install a Linux Server

Virtual hardware overview

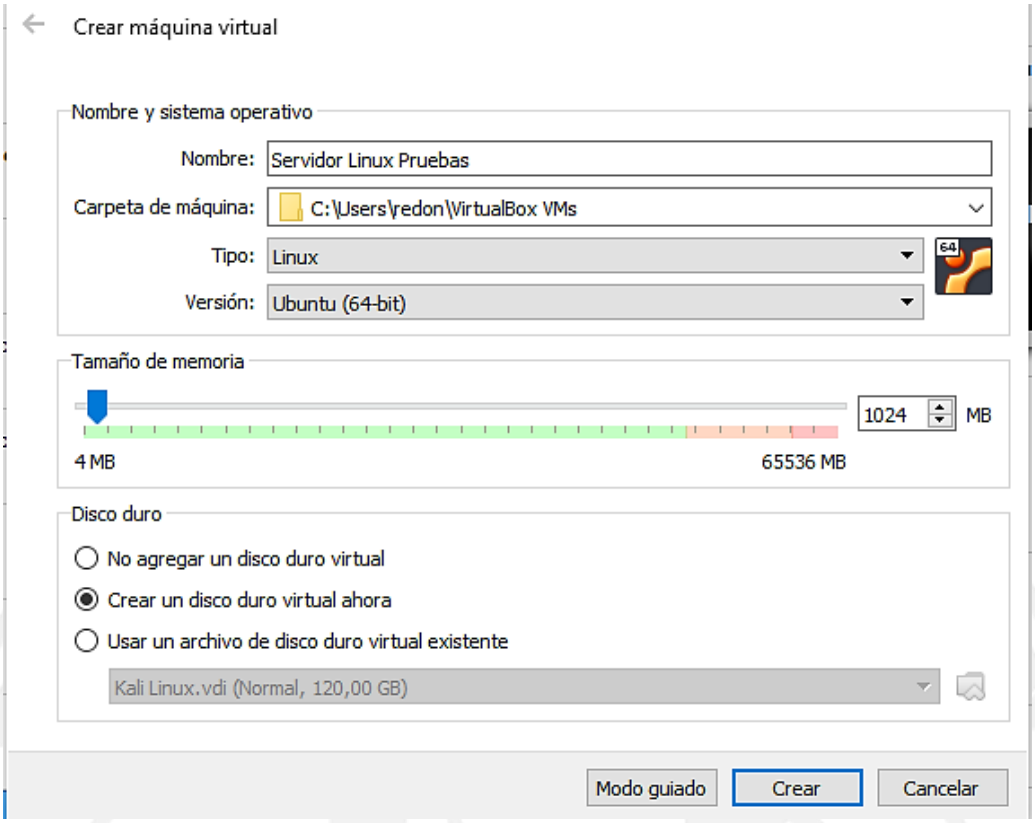
The prebuilt VM has the following characteristics you should be able to reproduce in your virtualization system and OS (please ensure that you have enough space available). We provide here the characteristics of the VM using *VirtualBox* features as an example. You should search for equivalent characteristic in your virtualization system

- 2Gb RAM (or more if you can afford it 😊)
- 2 CPU cores (1 could be also possible, but at a performance cost)
- 128Mb of video memory (maximum) with 3D acceleration enabled.
- 80Gb HD of dynamic storage. Using an SSD if available is recommended, but not mandatory. It is recommended to check the "Use the host I/O cache option".
- No audio is required
- A NAT network card (first one), configured to obtain its IP through DHCP
- A host-only network card (second one), with the static IP `192.168.56.33` (this is **optional**, and only if you are planning to use SSH clients such as *MobaXTerm* to interact with the machine and do not want to use port redirection as explained at the beginning of the document for some reason).

Rest of the settings can be left at their default values. These options will be explained in the following sections using *VirtualBox* https://www.virtualbox.org/ as virtualization software. Note that *VirtualBox* is available to all major OS for free, so you should not have any problem working with it. Also note that if you have *Hyper-V* enabled in a *Windows* machine, *VirtualBox* will be unable to operate, **as both are mutually incompatible if installed**.

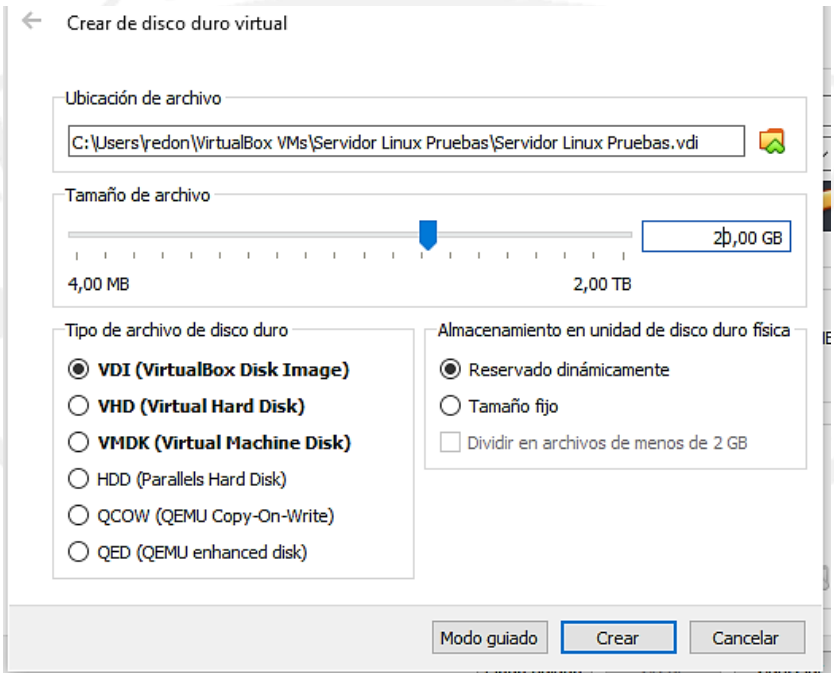
VM creation

The first step is to create a new machine profile suitable for the OS that we are going to install, in our case it will be a 64-bit *Ubuntu 18.04 LTS Server* you can download from here: https://ubuntu.com/download/server



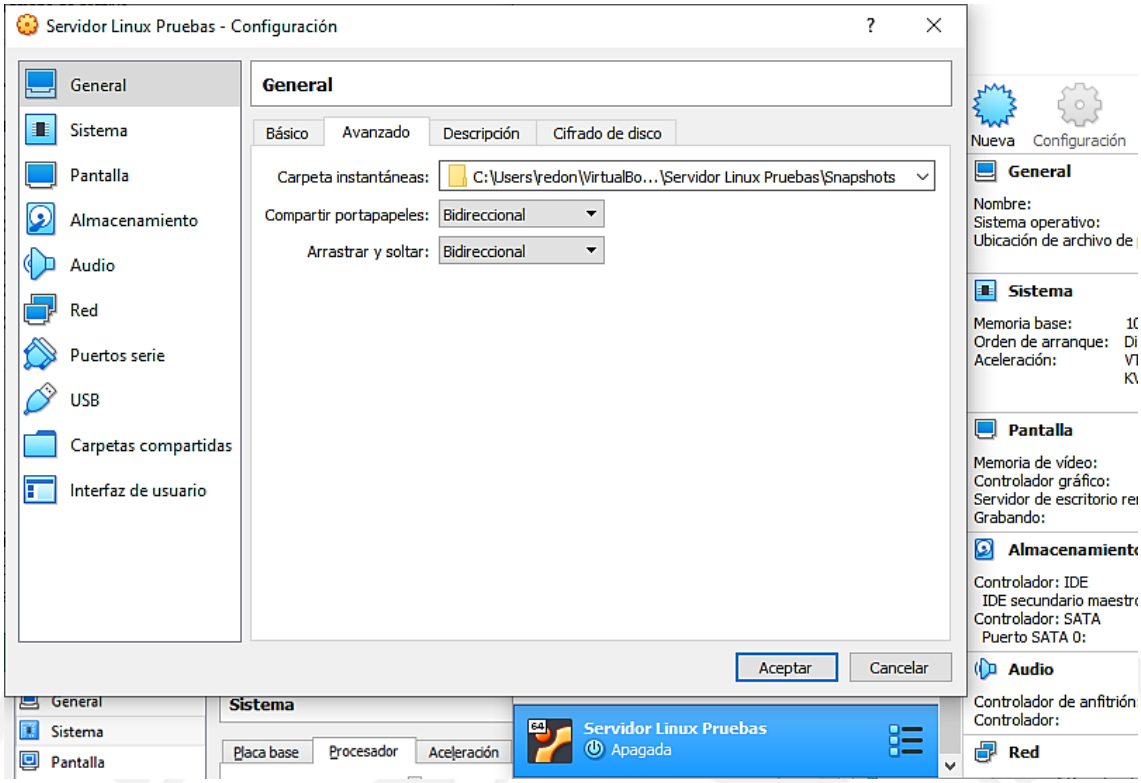
Type of virtual machine to create and RAM

We proceed now to assign the RAM and HD space we previously mentioned. The format can be *VirtualBox* native (VDI) or *VMDK* for portability with *VMWare*. For resource efficiency we will create it **dynamic.**



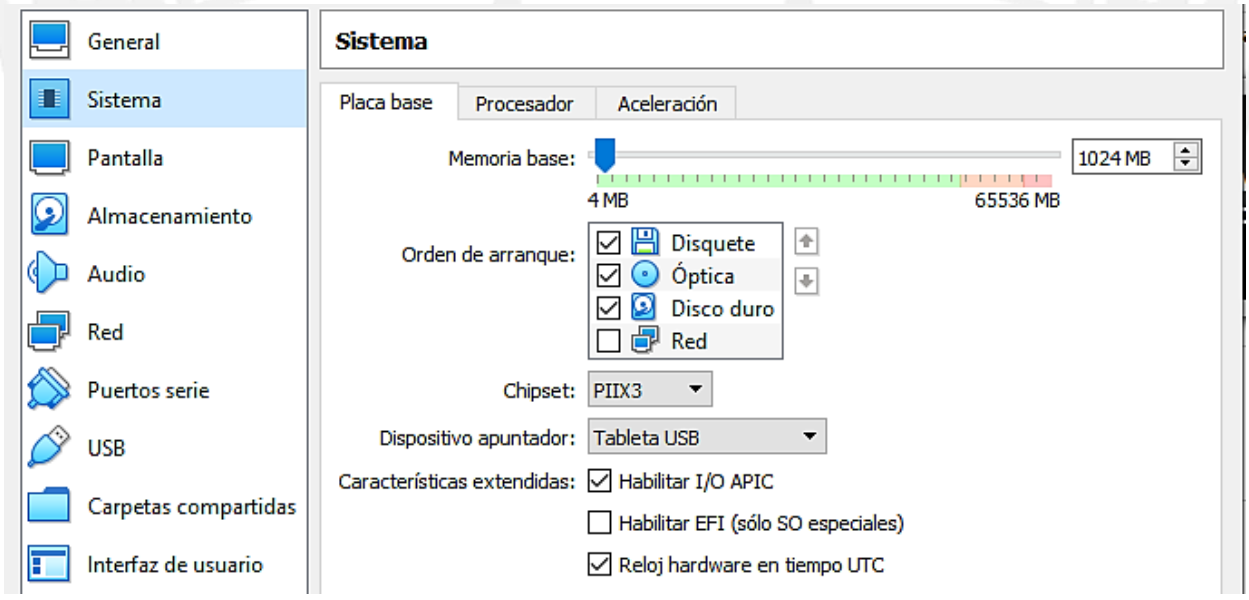
Virtual machine hard drive

Once done this, the new machine template will be created, and we can improve its configuration by editing it. The first step is the *General* category, where we can enable the *Clipboard* and the ability to drag and drop between the host and the virtual machine and vice versa (**Bidirectional**). However, this will only be effective if we have a GUI.



General Category of a VirtualBox

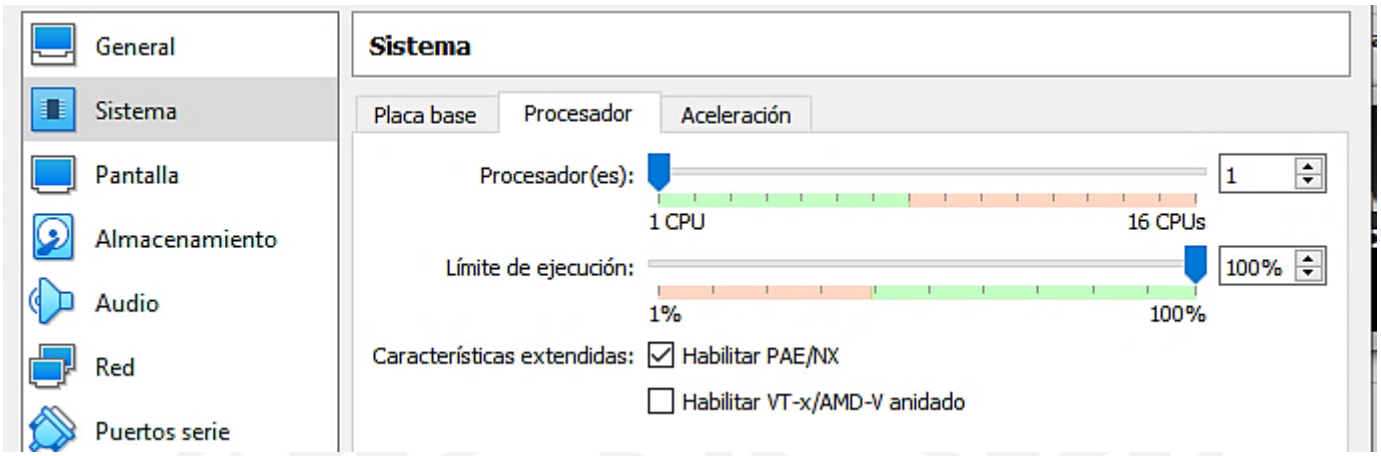
Later, in the *System* category we enable the IO/APIC and UTC clock options. While it is convenient to enable EFI for modern *Windows* or *Linux* operating systems, we found that if the machine is exported to **.ova**, during the re-import it may suffer boot issues, so we choose not to enable it to avoid them. It should be noted that, once the system is installed, this option cannot be changed: a system installed without UEFI mode will not boot if it is activated and vice versa.



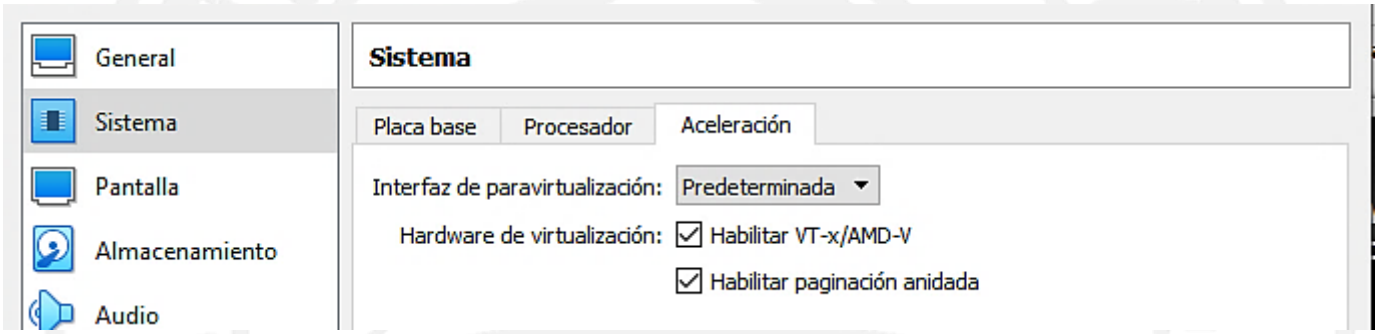
Category System of a VirtualBox Virtual Machine

Then we will assign 1 or 2 CPUs and proceed to activate the PAE/NX, VT-x or AMD-V CPU instruction set, and nested paging features (if available on our base hardware) for better virtualization performance. Nested VT-x /AMD-V may not be available depending on the CPU, but it is not required for this course. Simply put, you would allow virtual machines to run within virtual machines (with the non-trivial performance cost that this

entails 😊 😊). The paravirtualization interface does not need to be changed unless we have *Hyper-V* installed on a Windows. Since *Hyper-V* and *VirtualBox* cannot operate normally if installed on the same machine, this way we should be able to make use of both (at a substantial performance cost):

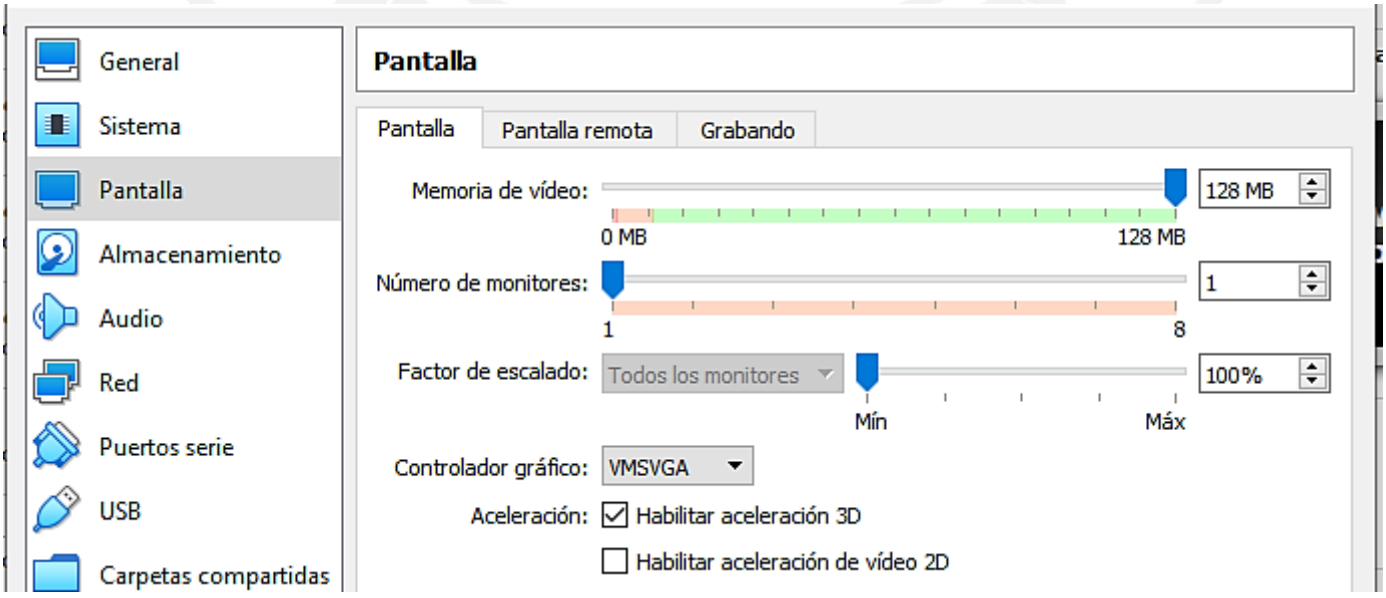


No of CPUs of the machine to be created



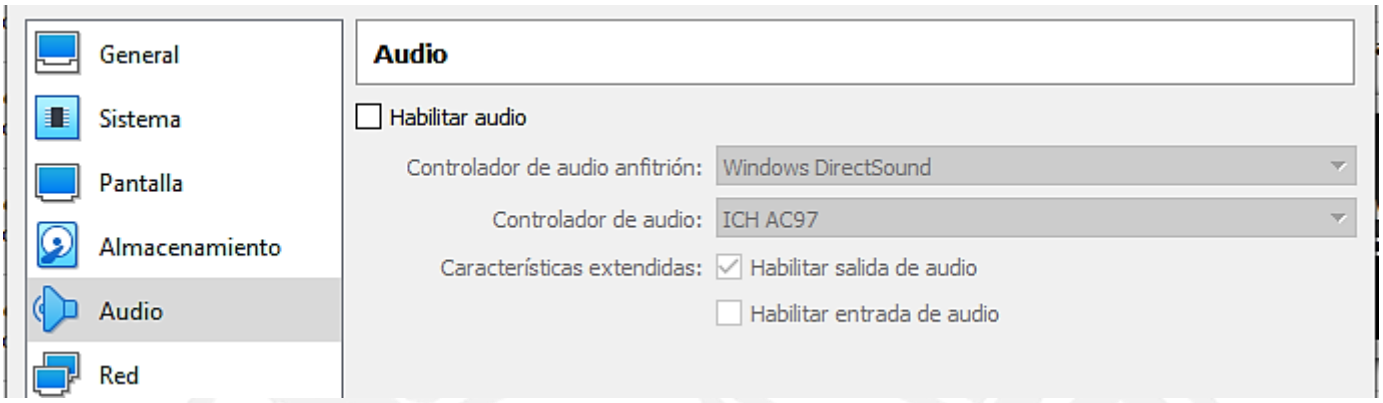
Acceleration for the VM

As for the selection of video memory, and 2D and 3D acceleration capabilities, it all depends on whether the machine to be installed will have a GUI. In a real scenario, as a server should be oriented to use the less resources possible, it should not have it, but this is not the case in our course. It should be noted that currently *VirtualBox* does not allow 2D acceleration on Linux platforms (3D is supported). However, as we plan is to install a GUI later, you should choose the maximum available video memory (128Mb). As graphical controller we leave the default configuration:



Video memory allocated to the machine

The machine profile we created will not use *Audio*, so we chose to disable it and thus do not install extra drivers or services.

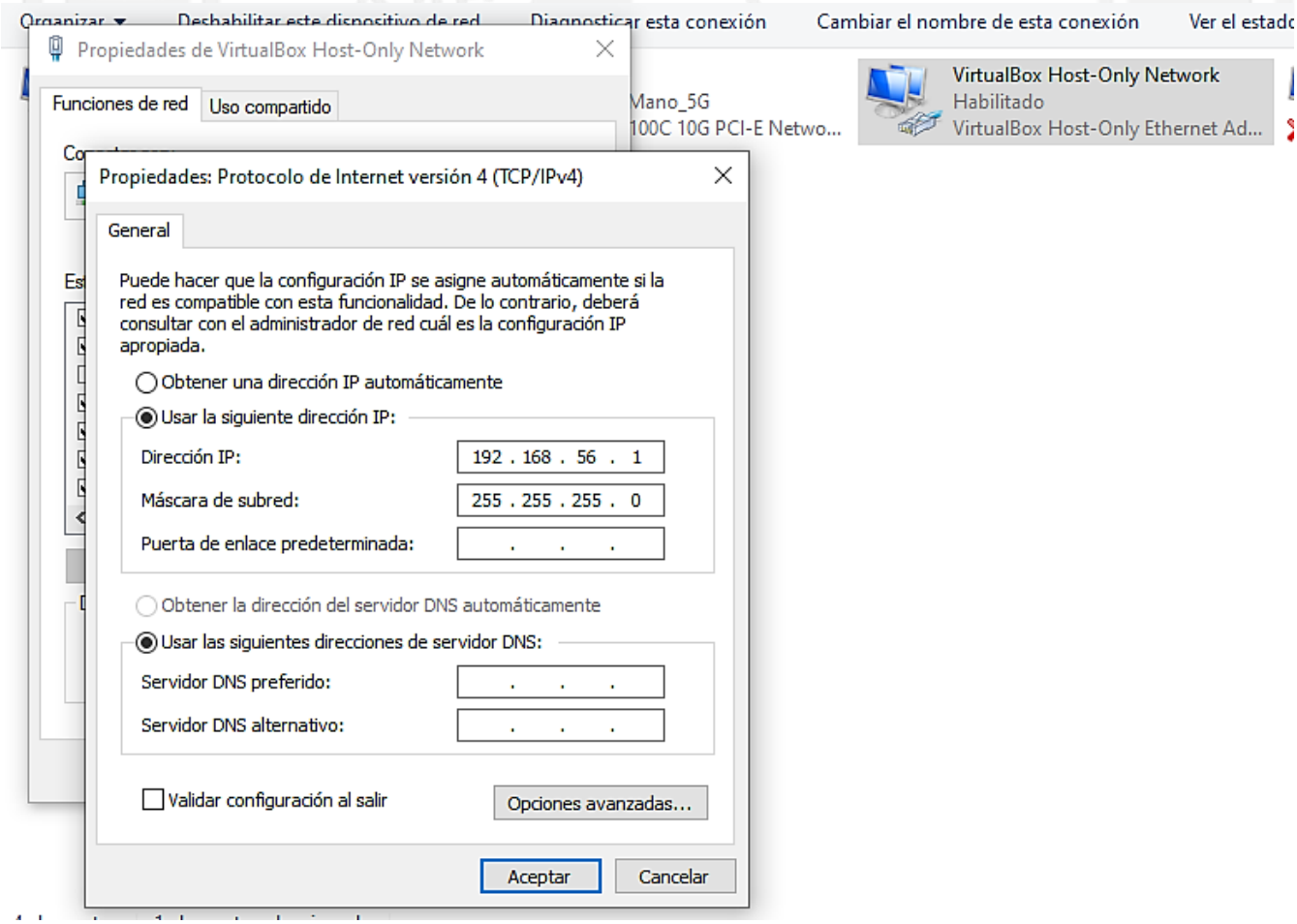


Audio settings

VirtualBox Networks

Virtual machine networks are more complex and require additional attention. The main types of networks that a *VirtualBox* machine may have are:

- **Host-only**: Allows our host machine to communicate directly with the virtual machine using a local network created for this purpose. In that case your PC should have a static IP assigned a network virtual adapter that is created when you install *VirtualBox*. It is usually the one seen in this image:



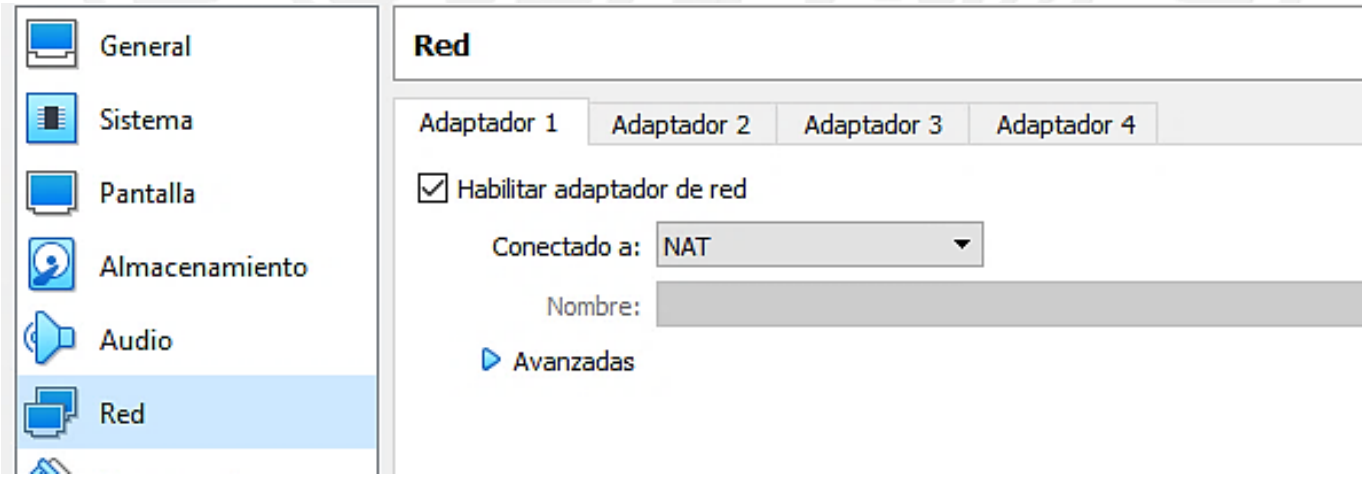
It would therefore be enough to configure the host-only adapter on the virtual machine, so that it had a fixed IP on that subnet (56 or the one used in this installation) to enable connectivity. This is the method that usually gives less problems: you can only see your local machines and the IP never changes. In school labs this may not be feasible because *VirtualBox* might have been installed without a network card that supports this feature and must be created by local administrator at each machine.

- "**Bridge**": This is typically used if your PC gets its IP via DHCP. The effect is that the created virtual machine will also do so, that is, it will be automatically assigned an IP on the same subnet as the host, being accessible from the host (and from outside the host) using it. The disadvantage is that it is not a fixed IP, and the virtual machine is visible to external agents. However, it is a very viable option in typical home network installations where a router or cable modem serves DHCP-dependent machines and routes all traffic out by a single external IP. Using this mode in labs can cause problems, because the IP pool available by DHCP for a lab can be exhausted if everyone uses a bridge network. If DHCP is not used, you need to manually assign to this network card an IP in the same network than the host.
- **Private internal network**: If you have two virtual machines, you can make one the client of the other by assigning both a new internal network interface with the same name assigned to them. Once this is done, you only need to assign each one a different fixed IP on the same subnet with the same network mask, so they can communicate with each other. The disadvantage is that you need both VMs running to do the tests, which means greater resource consumption.
- **NAT**: This interface is responsible for providing output to the Internet through the host connection automatically. It should be active on the first card to avoid problems and not take care of it anymore.

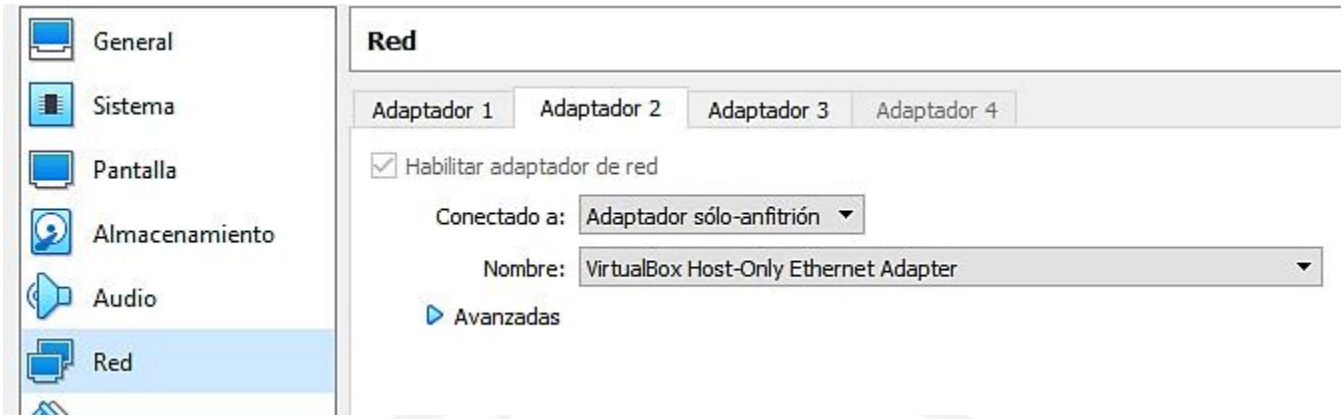
The *Linux* machine to be created should have two network cards:

- **Adapter 1**: By *NAT*. It allows the Internet connection through the host and will be the one used for updates and software download from *Ubuntu* repositories and third-party servers.
- **Adapter 2**: *Host-only*. The purpose of this network is that the host can be a client of the created machine (**optional**).

In the following images we see the configuration of the two adapters:



Network adapter 1



Network adapter 2

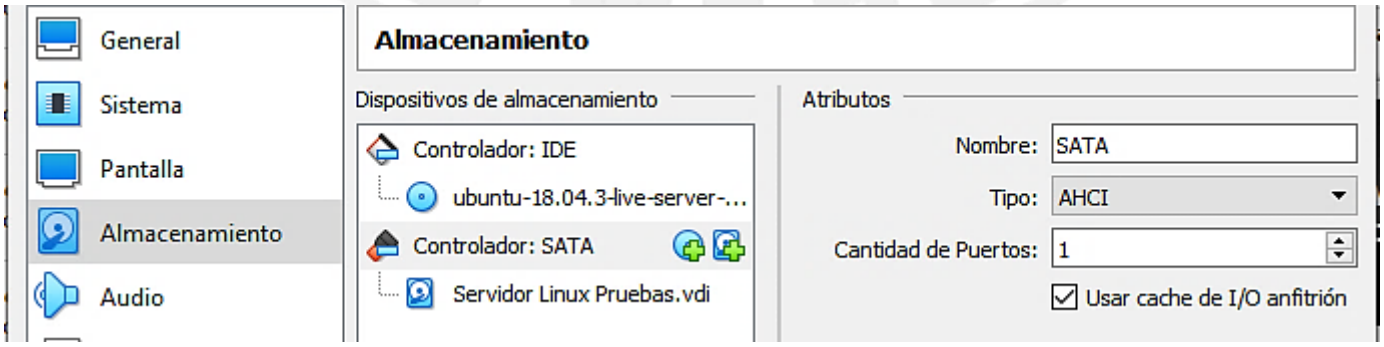
USB

Only USB 1.1 support should be enabled on the machines because any subsequent USB revision requires the installation of the *VirtualBox Extension Pack*, which might be problematic when reimporting the machine in other PCs. This limits the use of USBs to keyboard and mouse, as the connection of pens or hard drives to the virtual machine will be slow. If you run this at home or do not use *VirtualBox*, you can enable the best USB support that your virtualization system allows.

USB virtual machine support

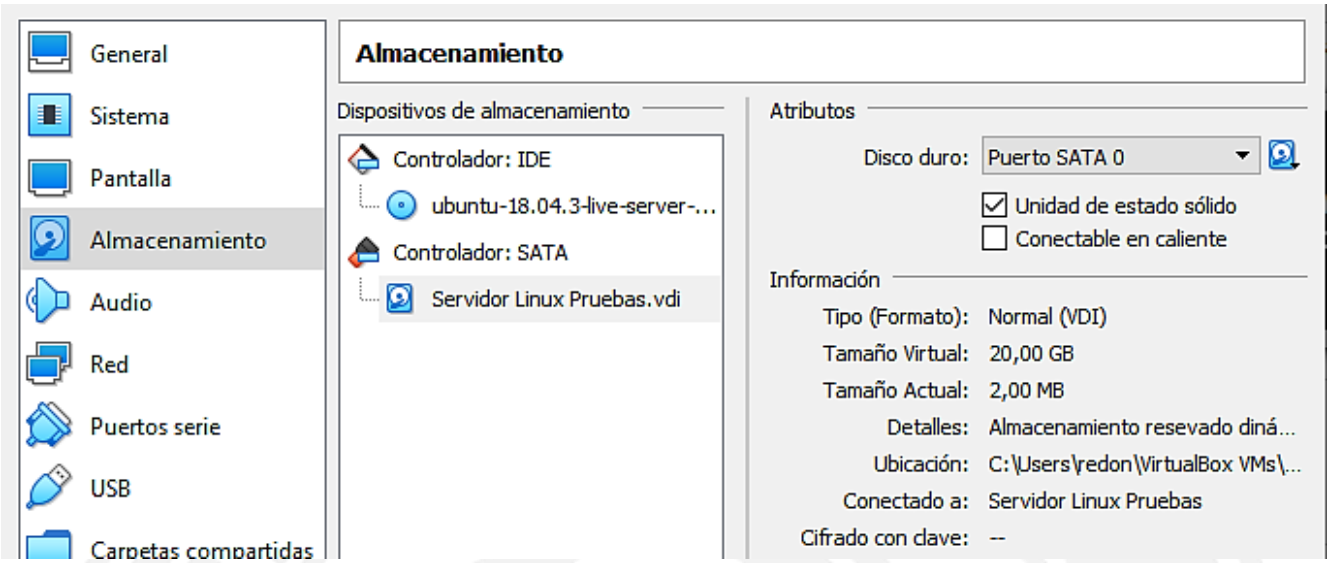
Storing and selecting the OS ISO

Regarding storage, for optimal virtualization we can enable the host I/O cache for the SATA hard drive controller



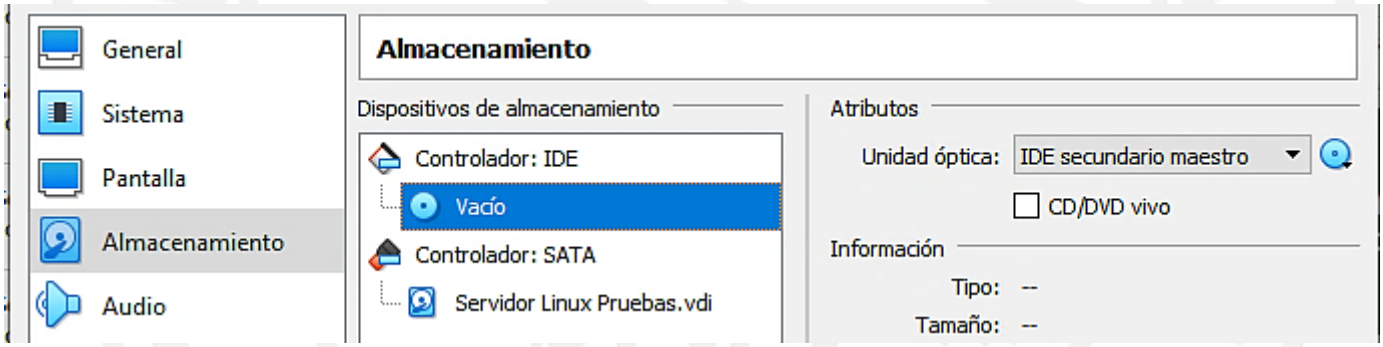
SATA controller configuration

In addition, if the machine is saved on an SSD, we can tell *VirtualBox* to make the necessary optimizations. You do not want to check that the drive is hot-pluggable unless it really is (that is, it is a secondary hard drive that is on a USB pen or something similar):

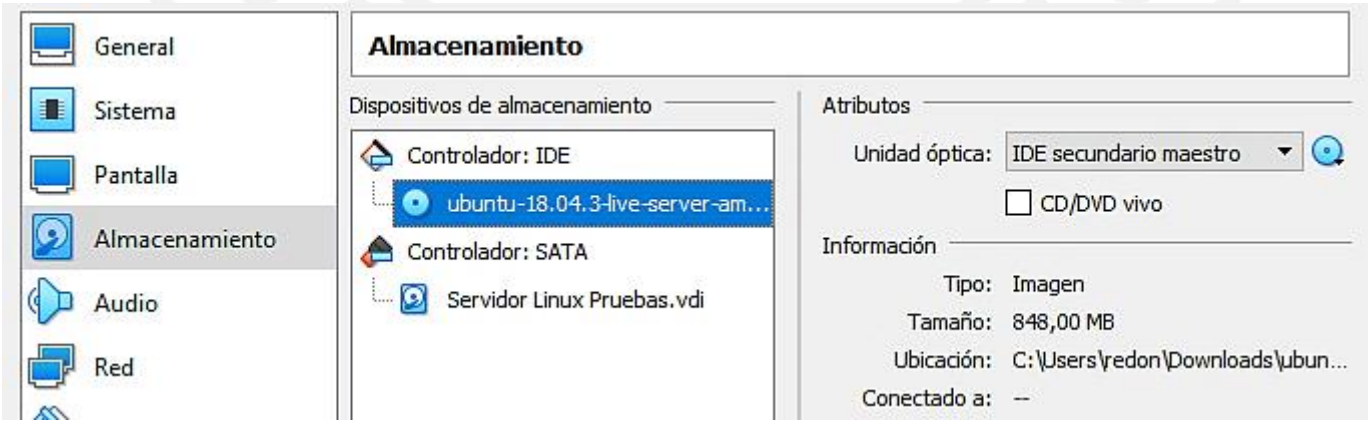


Disk configuration

Finally, we have configured the machine to start the installation, and we must only select the ISO of the OS that we previously downloaded. We proceed to assign it to the corresponding CD/DVD drive.



Assigning the Installation ISO (I)



Assignment of the installation ISO (II)

Now we can start the machine and the installation

Install an Ubuntu Linux Server

OS installation

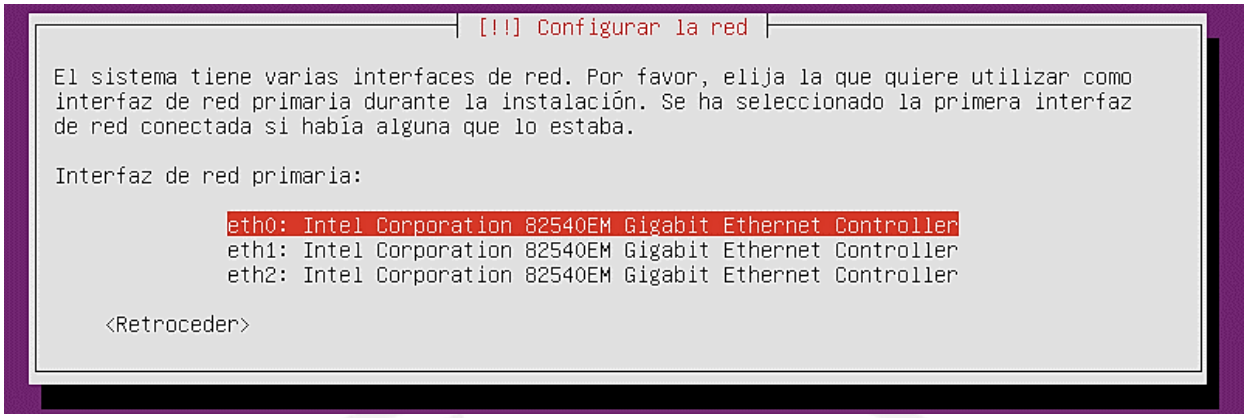
Once the virtual machine is configured, it is time to install the *Ubuntu Server*. The installation screens may have changed slightly with successive versions, but they are the same as the ones listed here. In any case, we should start the installation using the simple installation option of *Ubuntu*:

Start of Ubuntu installation

Hardware discovery is typically a time-consuming process, but with *VirtualBox* virtual hardware it should not be a problem:

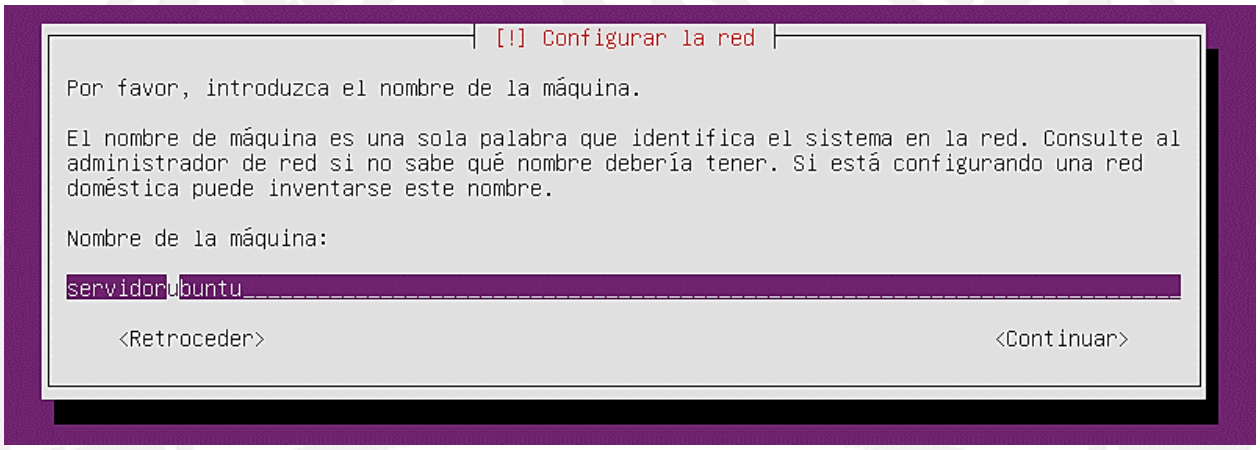
Virtual hardware detection

Selecting the main network card is important because it will determine the one the system uses to connect to the Internet for installations/updates. In our case it is the first, as we saw above:



Network configuration

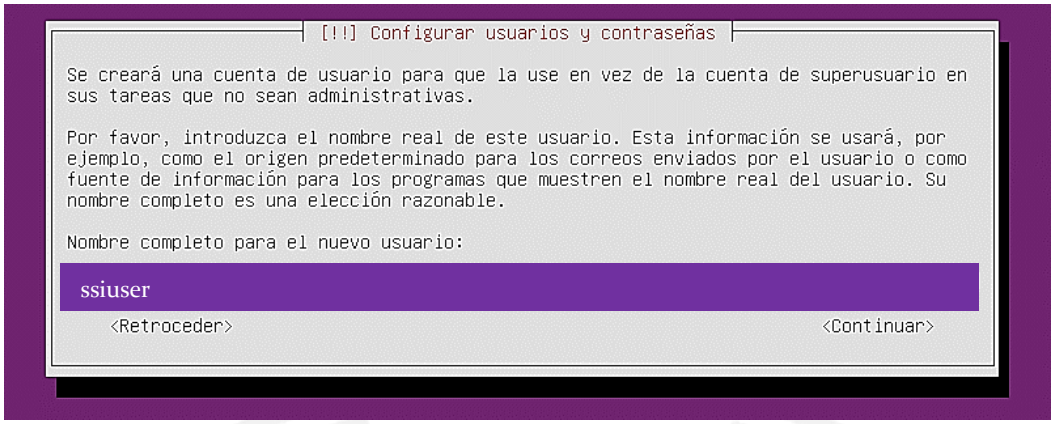
Since this machine is going to be cloned, the name we give it is a little irrelevant at this point, because every clone we make must have its own.



Assigning a name to the machine

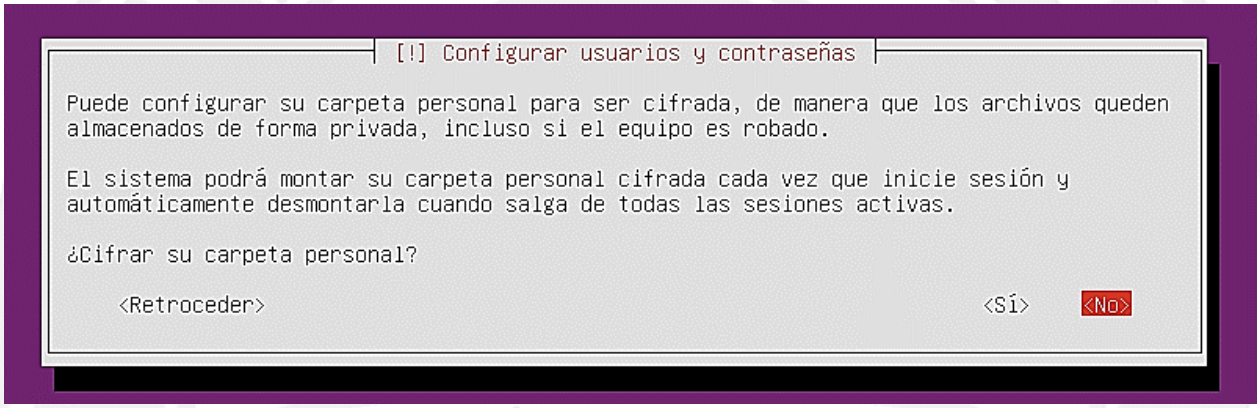
Creating an initial user is a crucial point in installing an operating system. There are several reasons:

- Management operations and "normal" use of the machine from the `root` account should not be done. This is because this account has an exceedingly high set of privileges and can do virtually anything. Any action we do as `root` can be very harmful if we do not do it correctly, which is why Ubuntu does not let us access directly with this user via standard login.
- As we obviously need a user to work, the installation asks to create one, which in this example will be called "`ssiuser`".
- This user is "special", since it belongs to the *sudoers* group. What does this mean? that even if we do not access as `root` directly, we still need elevated privileges to make installations, updates, etc. as necessary. This cannot be done with a "normal" user (such as the one we created), but if that user is *sudoer*, you can use the `sudo <command that we want to run>` command to get, after entering the *sudoer* user's own *password,* that command executed with `root` privileges and therefore can do the administration/installation/etc. The user created in the installation belongs by default to that group and is the one we will use to manage instead of `root`.
- Future users that can be created in the system are not *sudoers* by default unless we need it.
- This way only one user has elevated privileges and only uses them when you explicitly invoke a command interactively. This prevents potentially harmful software from doing a lot of damage to the system, since the privileges of a normal user are far from those of `root`. This is not valid, of course, if we run harmful software with `sudo`...



Creating a default user

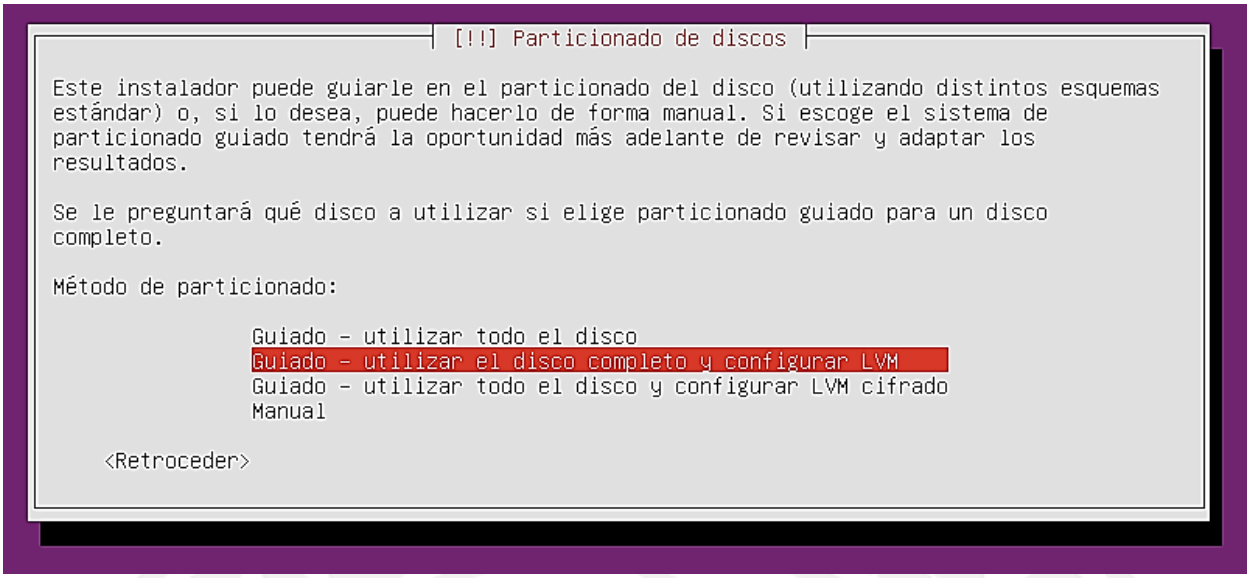
The use of personal file encryption is optional, but convenient for physical security reasons. However, in this course we will not use it, but it is good to know that it can be activated here by the time we talk about this option in theory.



Encrypting users' personal folder

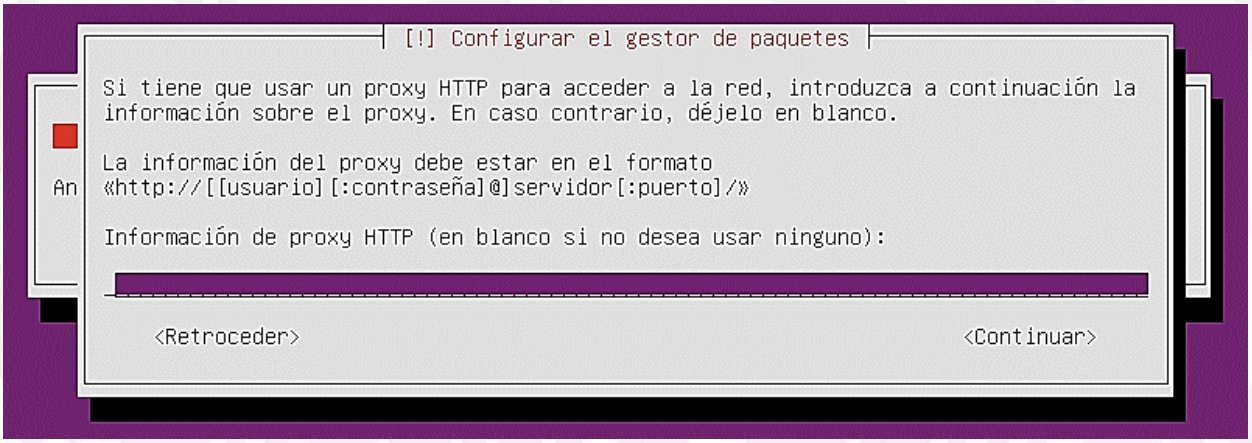
Disk partitioning is a sensitive topic that may even require pre-planning, especially on multi-disk systems. Partitioning must be tailored to the needs of the server. If we have several disks of different technologies and speeds, we can decide to start the file system so that each disk is occupied by a specific part of the storage system (user files, web files, *uploads*...). It is also common to have a `/home` partition with user files independent of the other files. In any case, we will not insist on this aspect, and we will use the simple model proposed by the installer "*Use the full disk*". Other partitioning options can be found at the following address: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/s2-diskpartrecommend-x86.html

However, it is good to talk about the other option: "Use *the full disk and configure LVM*". LVM stands for *Logical Volume Manager* and is a utility that allows you to create logical volumes from physical hard disks. For example, if we have two physical hard disks and we want the operating system to see them as a single partition, LVM can be used to logically "join" both and to appear to have a single hard disk installed whose size is the sum of both. Therefore, it is a very appropriate way to manage the disks of a PC by grouping *physical volumes* (PV) into virtual groups of disks (*volume groups*, VG) so that you can then create partitions or *logical volumes* (LV). For practical purposes, if you have multiple physical hard drives you want to join in a single space, it is convenient to use LVM. For a single physical disk is not necessary, but if convenient if the space we have turns out to be very small and we want to "hot" expand it with new physical hard disks. We can find more information about this and the use of encrypted LVM at the following link: http://diegosamuel.blogspot.com.es/2014/01/particiones-con-lvm-y-cifrado.html.



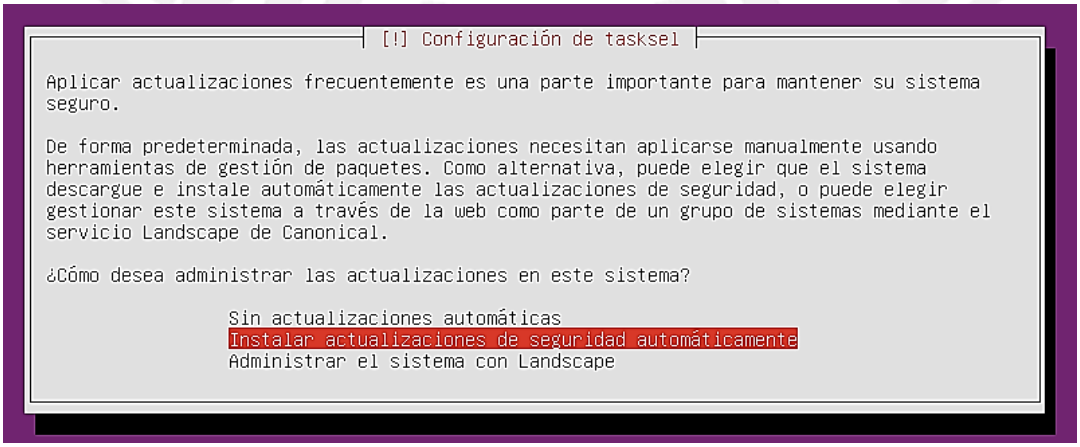
Disk partitioning

After configuring the hard drive partitioning, we must specify if we want to use an HTTP *proxy* to connect to the Internet. In our case it will not be necessary:



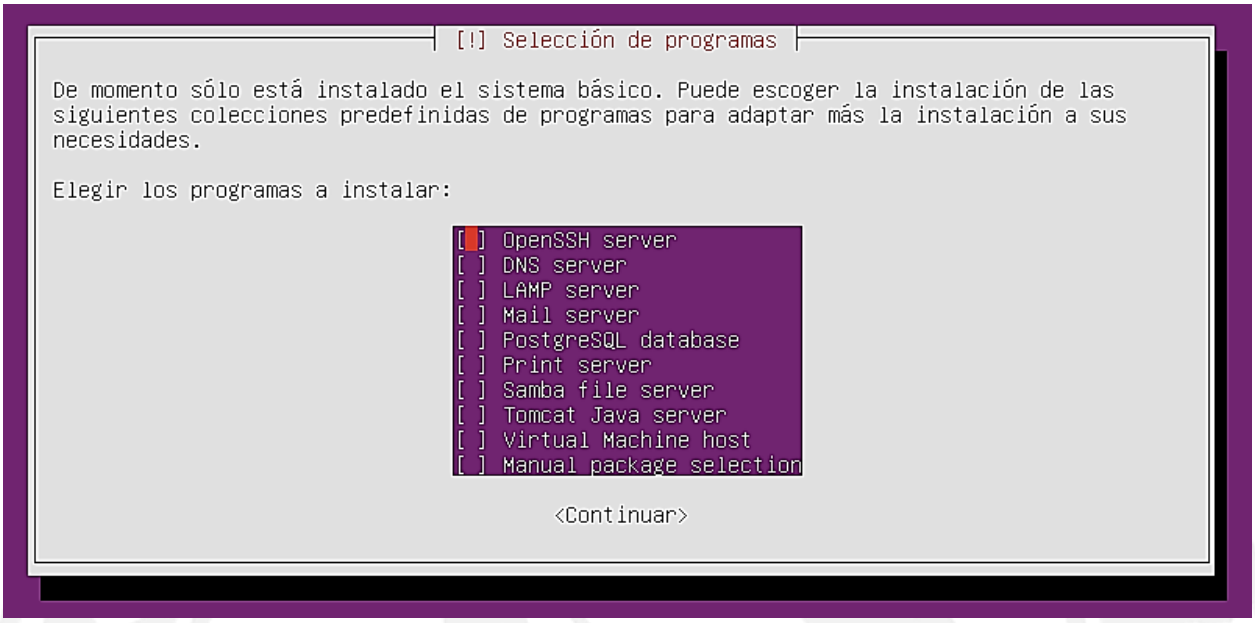
Network proxy mapping

When it comes to updates, keeping the operating system up to date is especially important for machine security, which is why we will enable automatic installation of security updates. However, it is advisable to get used to updating the OS manually ourselves on a regular basis.



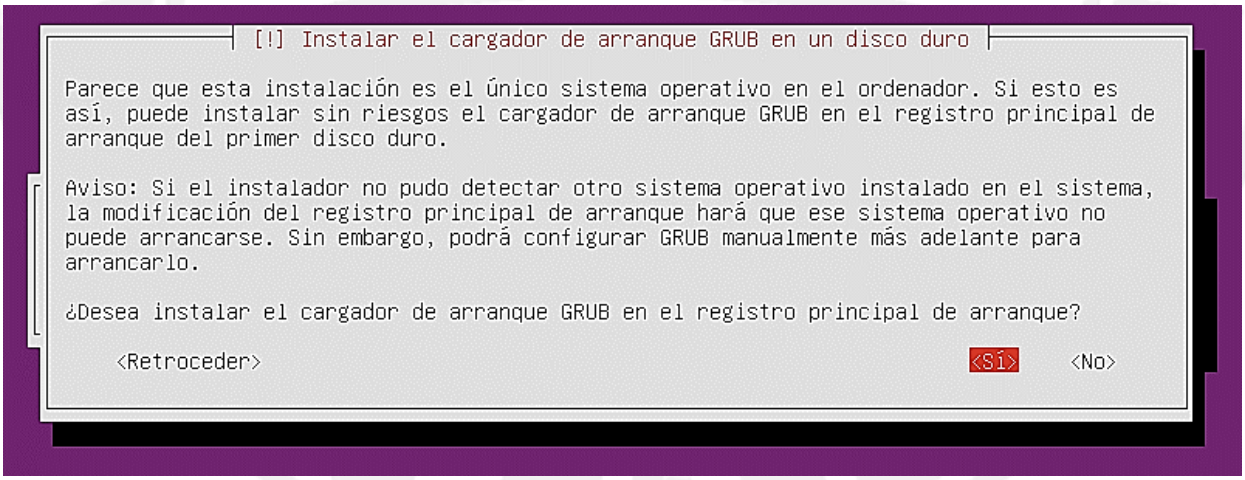
Managing security updates

When almost finishing the installation, the installer asks us if we want to perform installations of package groups that configure the server with a particular usage profile (DNS server, SSH server, etc.). Since we are installing a base machine, in this case it is best not to install anything by default and leave the system as simple as possible (**NOTE**: This screen usually displays *Docker* as an installation option. This version is not the latest one but should be enough for the course. You can check *Docker* here and skip its installation later).



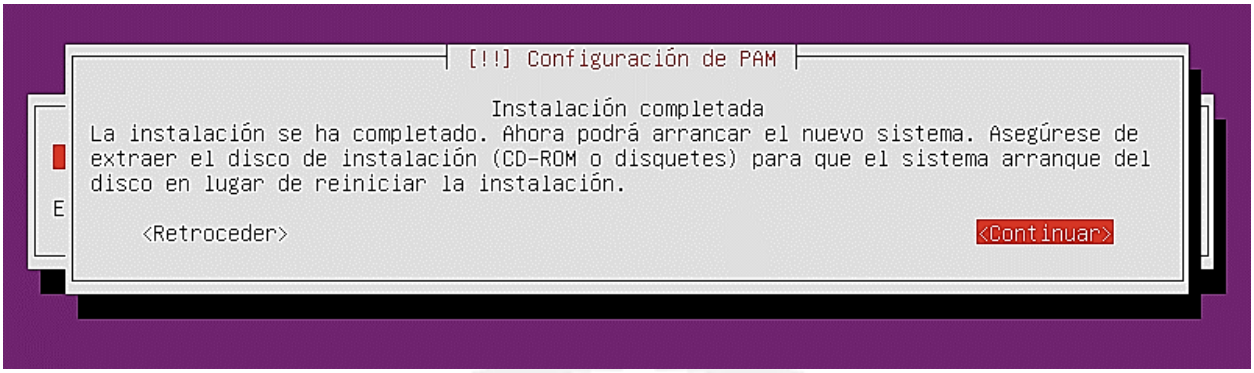
Selecting a package installation profile

After the choice of installation options, since this is going to be the only OS on the machine and we only have one HD, we can install GRUB without problems.



End of installation setup process

We then proceed to the automatic installation until we reach the completion and restart message. You should remove the operating system ISO from the machine's virtual drive before rebooting.



End of installation and restart

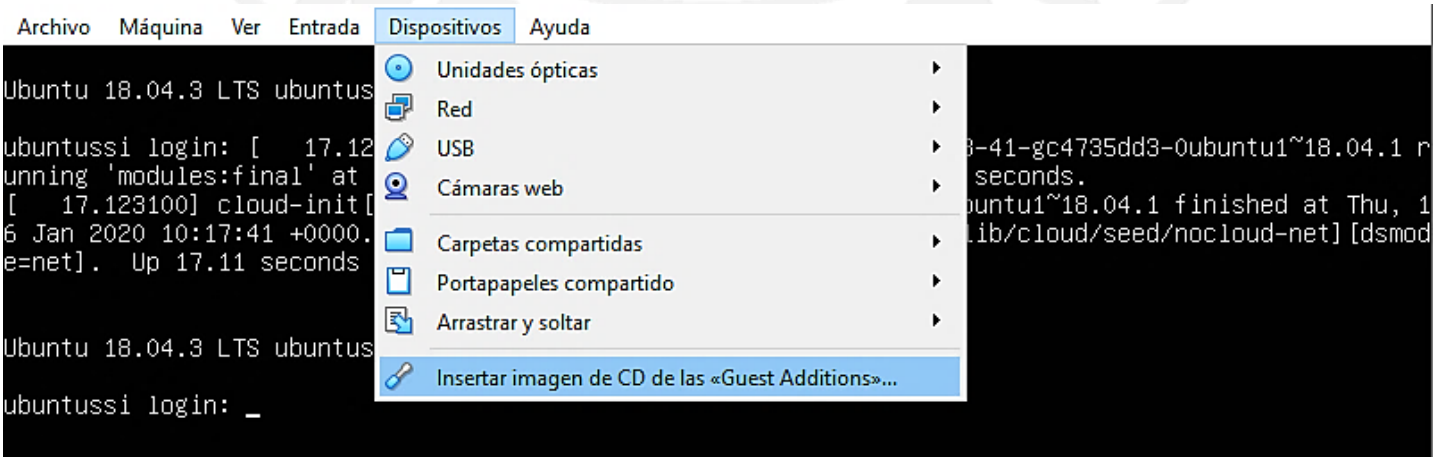
Having done this, we can already start working with the Linux installed, making use of the only user we have created for it:

First login

Configuring the machine to improve virtualization

Once the machine is operational, and before we leave it ready to work, one of the things we should do is improving its installation by taking advantage of the *"Guest Additions"* of the used virtualization platform (many well-known virtualization products have similar modules of this class). The use of these *guest additions* is convenient, since it will improve the overall performance of the machine, installing more suitable drivers. In the case we are dealing with *VirtualBox,* the instructions to install them can be found at the following link: https://www.tecmint.com/install-virtualbox-guest-additions-in-ubuntu/. However, we will reproduce them below.

Keep in mind that *guest additions* need a set of Linux packages not installed as standard for their proper installation, such as `dkms`, `build-essential` and `linux-headers-generic`. It is also convenient to install the kernel headers of the specific architecture by adding the `linux-headers-$(uname -r)` package. With these packages installed we can insert the CD image of the guest *additions* and proceed to the execution of their installer:



Installing guest additions

To do this we first mount the CD, once inserted in the `/media/cdrom` directory

```
operario@servidorubuntu:~$ sudo mount /dev/sr0 /media/cdrom
mount: dispositivo de bloques /dev/sr0 está protegido contra escritura; se monta
 como sólo lectura
operario@servidorubuntu:~$
```

Access to *guest additions* files

We access your content and run `./VBoxLinuxAdditions.run` to select the executable of the guest *additions* most suitable for the machine we have and proceed to its installation:

```
operario@servidorubuntu:/media/cdrom$ dir
32Bit         cert                    VBoxSolarisAdditions.pkg
64Bit         OS2                     VBoxWindowsAdditions-amd64.exe
AUTORUN.INF   runasroot.sh            VBoxWindowsAdditions.exe
autorun.sh    VBoxLinuxAdditions.run  VBoxWindowsAdditions-x86.exe
operario@servidorubuntu:/media/cdrom$ sh ./VBoxLinuxAdditions.run
Verifying archive integrity... All good.
Uncompressing VirtualBox 4.3.28 Guest Additions for Linux............
This program must be run with administrator privileges.  Aborting
operario@servidorubuntu:/media/cdrom$ sudo sh ./VBoxLinuxAdditions.run
```

Running the *guest additions* installer

To continue working with the machine it is necessary to do a restart.

At this point it is STRONGLY RECOMMENDED to save the machine as it is, and work with **clones of it from now on. This way, if we make a mistake in the configuration of the machine in some activity and its solution is not trivial, we will be able to return to the starting machine without wasting too much time. Options 1 and 2 allow to rebuild the machine easily, but it is not a bad idea to create a clone in these cases.**

Software to install in your custom VM

After you have the corresponding hardware specifications configured, and the OS installed, following the steps on the next sections, you must ensure to install the following software in the virtual machine to be able to correctly follow the laboratories

- Adding *Docker* repository and updating `apt`

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo apt-get -y update
```

- Upgrading packages

```
sudo apt-get -y full-upgrade
```

- Installing necessary programs

```
sudo apt-get install -y --no-install-recommends apt-utils apt-transport-https ca-certificates
gnupg-agent software-properties-common unzip curl wget tmux preload gpm mc virt-what nano grep
firefox iputils-ping net-tools less openssl bash-completion
```

- Installing the XCFE4 GUI

```
sudo apt-get -y install -y xfce4
```

- Installing *VirtualBox Additions* (**only if you use VirtualBox as your virtualization platform and you did not install them using the installation CD as previously mentioned**)

```
sudo apt-get -y install -y virtualbox-guest-dkms virtualbox-guest-utils virtualbox-guest-x11
sudo VBoxClient --clipboard
sudo VBoxClient --draganddrop
sudo VBoxClient --vmsvga
sudo VBoxClient --checkhostversion
sudo VBoxClient --seamless
sudo adduser ssiuser vboxsf
```

- Installing *Docker* and making your current user a *Docker* user without needing `sudo`

```
sudo apt-get -y install docker-ce docker-ce-cli containerd.io
sudo groupadd docker
sudo usermod -aG docker ssiuser
sudo systemctl enable docker
```

- Installing *Docker Compose*

```
sudo apt-get -y install docker-compose
```

- Change keyboard layout to Spanish (**not necessary if you choose Spanish during the installation of the OS**)

```
loadkeys es
sudo dpkg-reconfigure console-setup
sudo sed -i 's/XKBLAYOUT=\"\w*"/XKBLAYOUT=\"es\"/g' /etc/default/keyboard
```

- Install `tmux` to have multiple terminal windows and panes (optional). You can check the `tmux` cheatsheet on a previous section to find a nice starting configuration. Also, the files to work with *Vagrant* contain a `tmux.conf` file with another nice starting configuration that you can copy to `/etc/tmux.conf`

```
sudo apt-get -y install tmux
```

- Clean up

```
sudo apt-get -y autoremove
sudo apt-get -y autoclean
```

- Reboot

```
sudo reboot
```

We also strongly recommend installing now every software present in the `customization_scripts\software` folder of the *Vagrant* VM files you can download on the virtual campus. You only need to copy and paste the provided scripts.

Special steps in Hyper-V virtualization platforms

If instead of *VirtualBox* you are using *Hyper-V*, the recommended software installation procedure changes in just one step. Replace the step "*Installing VirtualBox Additions*" by this, ensuring the best possible performance of the VM in this virtualization platform:

- Enabling *Hyper-V Linux Integration Services* (LIS)

```
    sudo sed -i '$a hv_vmbus' /etc/initramfs-tools/modules
    sudo sed -i '$a hv_storvsc' /etc/initramfs-tools/modules
    sudo sed -i '$a hv_blkvsc' /etc/initramfs-tools/modules
    sudo sed -i '$a hv_netvsc' /etc/initramfs-tools/modules
```

- Replace default kernel with `linux-virtual` for the best LIS experience

```
    sudo apt-get -y install linux-virtual linux-cloud-tools-virtual linux-tools-virtual
```

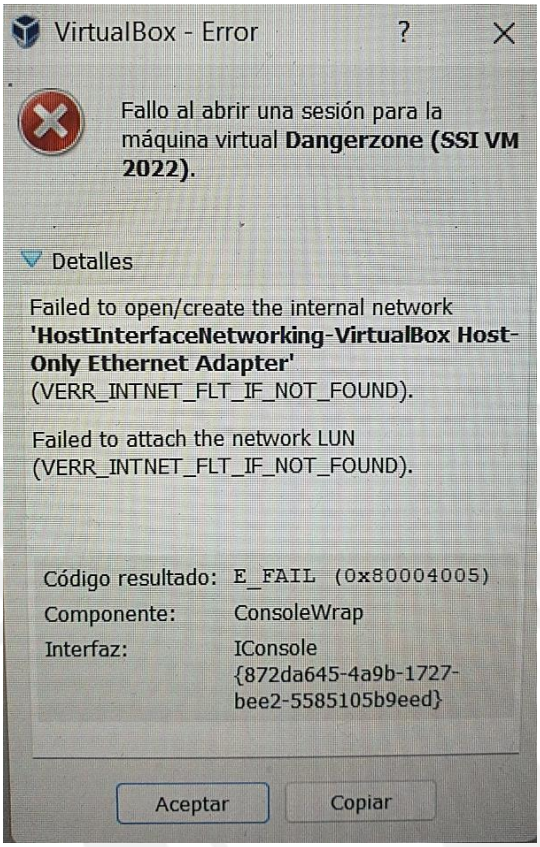
- Update `initramfs`

```
    sudo update-initramfs -u
```

General Troubleshooting (any Option)

When I boot the VM, I have the VERR_INTNET_FLT_IF_NOT_FOUND error

This error (see image) happens when the current installation of *VirtualBox* does not configure a Host-Only adapter and, therefore, every VM using one will fail on boot. The fastest solution is simple, deactivate such adapter in the "Network" configuration tab of the machine that gives this error.



My VM boots, but I only obtain a black screen and no output at all

If your VM does not boot and it just freezes with a blank screen and no output at all, the cause is that if you have *Hyper-V* enabled in a *Windows* machine, *VirtualBox* will be unable to operate, **as both are mutually incompatible if installed**. The solution is either uninstall *Hyper-V* (if you do not really need it but be careful because things like *Docker Desktop* and other programs require it, so check first if you do not really need it for other courses) or uninstall *VirtualBox*, and let *Vagrant* detect *Hyper-V* and try to install the VM on it. Additional information about the problem can be found here: https://forums.virtualbox.org/viewtopic.php?f=6&t=92673

Of course, **MAKE A BACKUP OF ANYTHING VALUABLE FIRST**! :)

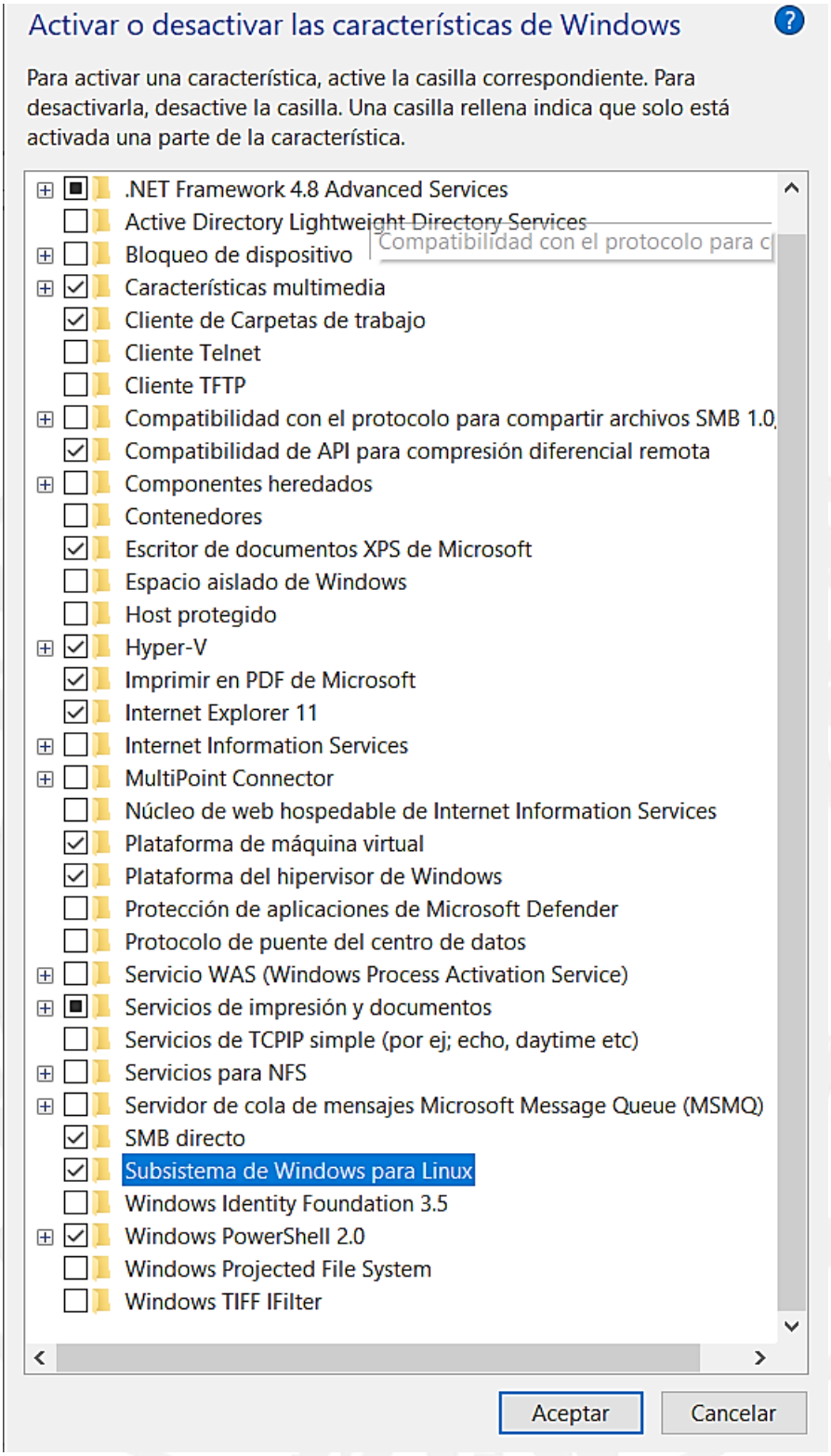
I still need Hyper-V for other courses / my work, so disabling it every time I boot the VM is cumbersome

Some of you have told us that the software used in other courses interferes with the SSI virtual machine on some occasions. Specifically, one of the main "culprits" is *Docker Desktop*, which uses *Hyper-V* underneath. While a solution is to disable it every time our VM is going to boot or make an installation via *Vagrant* for *Hyper-V* instead of *VirtualBox*, this can be cumbersome.

For that reason, we have checked our virtual machine and found that the reason for this incompatibility ("green turtle" icon, black screen without any output) is in one of the optimizations we have made to ensure that the machine goes as fast as possible and occupies the least amount of RAM, so that it works better on any host.

For that reason, we have created a new version of the virtual machine (Options 1 and 2) that does not use these optimizations, and that we have verified that it works with the following active services:

1. Hyper-V
2. Virtual Machine Platform
3. Windows Hypervisor Platform
4. Windows Subsystem for Linux

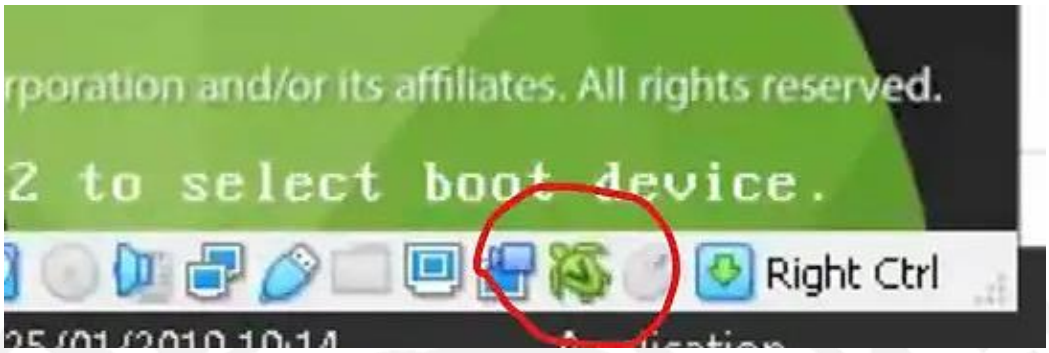


It is possible that the activation of services 2 and 3 is necessary for a smooth operation of this new version, which you can find just below the existing one in the virtual campus. The new version of the MV does not change anything from the old one in terms of software and features. We have verified that it loses around 15% of performance, but it works in an environment where we have reproduced the problematic scenario that some of you have found, in which the original was not able to boot. Please note that the "green turtle" icon still appears, only this time the machine can start and fulfill its mission.

GENERAL VM MANAGEMENT OPERATIONS

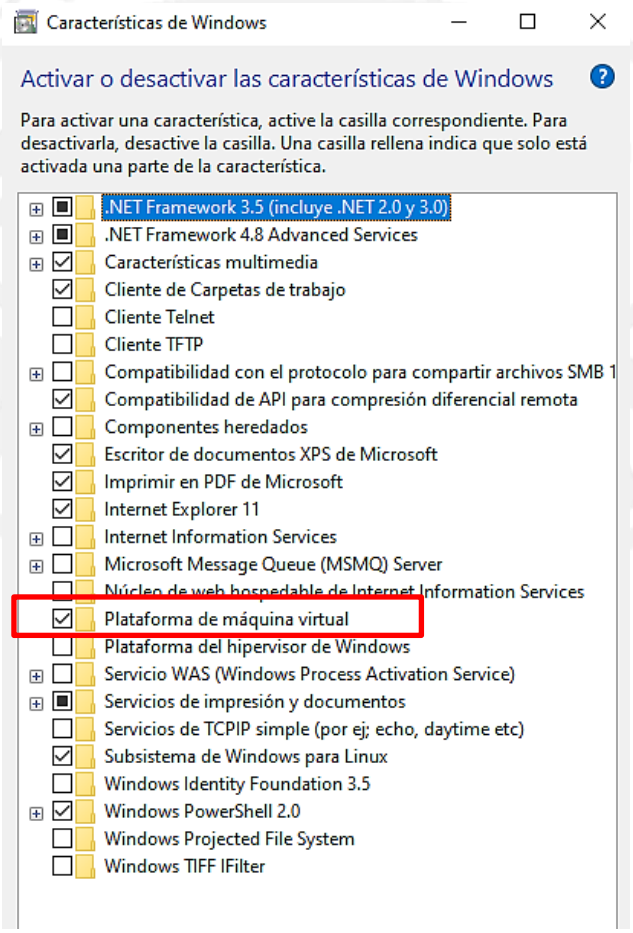
Resolving incompatibilities between VirtualBox and Hyper-V

Some students have experienced the dreadful "Green turtle problem", which means that *VirtualBox* is running slow due to something blocking hardware virtualization acceleration (this is indicated with a green turtle icon in the lower bar of the VM screen window). The main indicator of suffering this problem **is a black screen with no output at all**.



Typically, this is caused by Hyper-V blocking this feature, but on rare occasions, a few of you have something that shadowy uses virtualization and is harder to locate, giving you the slowness problem (the turtle). Someone put in the VirtualBox forums the list of problematic software that causes this problem: https://forums.virtualbox.org/viewtopic.php?f=6&t=92673

However, we found that most of you have the same culprit, apart from Hyper-V: the "*Plataforma de máquina virtual*" (*Virtual Machine Platform*) *Windows* feature enabled.



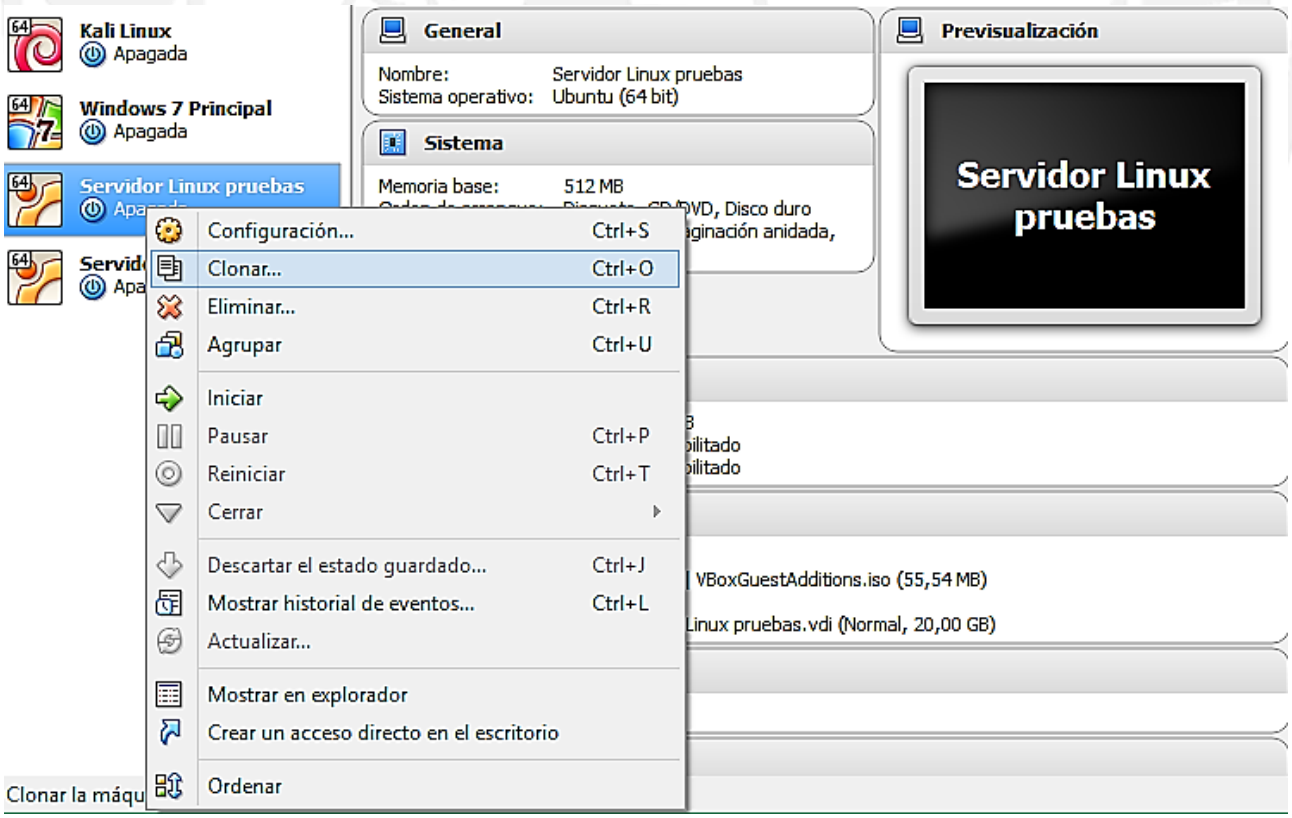
If you disable that, your *VirtualBox* starts running at normal speed again (no more turtle just a blue v icon) and stop slowing down your machine too much. This happens no matter the option you choose to create the VM, as it is a Microsoft-VirtualBox issue.

Cloning and configuring machines

If we must install N *Linux* machines, repeating an installation process for each of them is tedious, error prone (especially if you do not automate it), and therefore inadequate. Normally what is done in these cases is to prepare a "model" machine, like the one we have built, and once optimized, clone it as many times as necessary to create a base for machines for different purposes, or have more copies of it available. **Another option is to work with snapshots** of the base machine, something we will later. Let us see how to perform the cloning process.

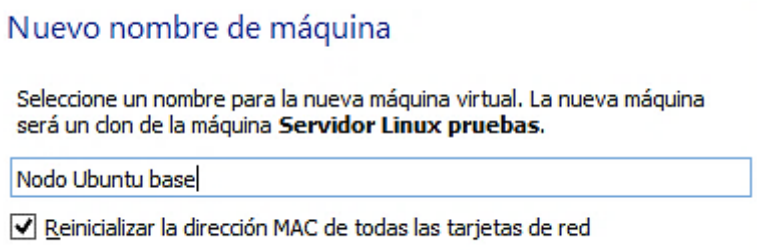
The first thing to do before cloning it is to study if it is worth doing an HD compaction. This is a process that examines the hard disk of the virtual machine and optimizes it so that the file representing that hard disk on the host occupies as little as possible without breaking the machine. For a machine that has been running for a long time and/or has installed or uninstalled multiple applications is an interesting process since it can decrease the size it takes. However, in our case, being a newly installed machine, we will not do it as it is not worth it. However, instructions on how to perform this process can be found here: https://www.virtualbox.org/manual/ch08.html or at: https://www.howtogeek.com/312883/how-to-shrink-a-virtualbox-virtual-machine-and-free-up-disk-space/. This makes it easier to download and distribute the machine.

Once everything is ready, we can take any machine and proceed to clone it with the associated *VirtualBox* option:



Cloning a machine in VirtualBox

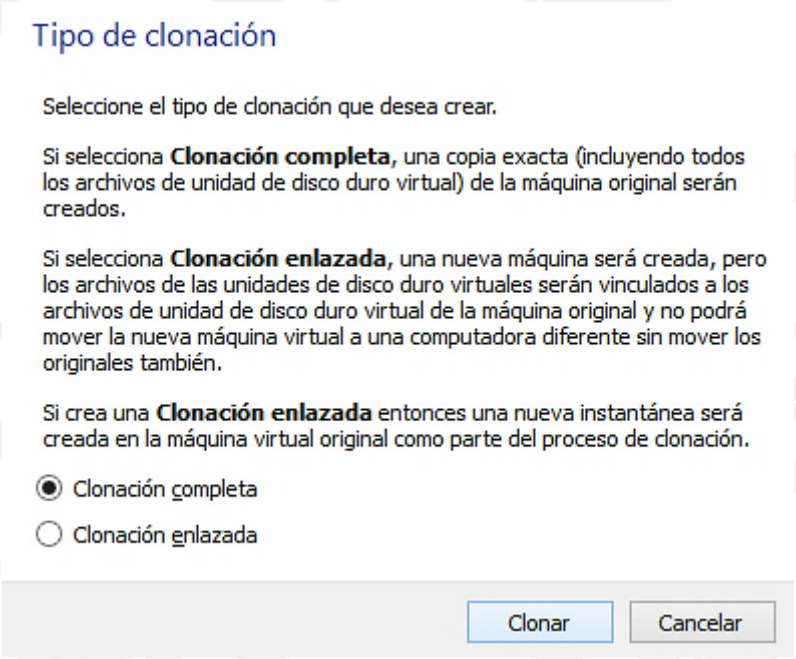
The cloned machine will appear in the *VirtualBox* inventory, so we need to give it a name that does not exist on it already:



Naming a new clone of a machine

It is important to change the MACs of the network cards because the cloned machines are likely to be connected via network with their "original" and between them, and we want to avoid communication errors that will occur if they try to communicate two machines with cards who have the same MAC.

Having done this, we select the type of cloning that we want, which in this case will be complete to be able to move it without restrictions to where we want:



Selecting the type of cloning to be performed

Now we have two different machines, the client machine, and the server, but both have the same internal name or *hostname* (`ubuntuserver` the one we gave it in the installation). It needs to be changed to at least one of them to avoid potential network problems if both work at the same time. To do this we edit the `/etc/hostname` file:

```
ssiuser@ssiserver:~$ sudo nano /etc/hostname
[sudo] password for ssiuser:
```

Renaming the clone machine (I)

And you also must edit the `/etc/hosts` file to change the machine name that appears there.

Creating Working Machine Snapshots

Snapshots are a "cheap" way of having a machine in which we can do operations and roll back to its previous state if something went wrong. Cloning a machine allows you to create a separate machine, but identical to the one you already have. This is perfectly valid, **but it consumes a lot** of **resources** because it duplicates all its contents. In certain contexts, such as our course labs, it may be more convenient to use *snapshots,* because the space they consume is much smaller, and therefore they are created much more quickly.

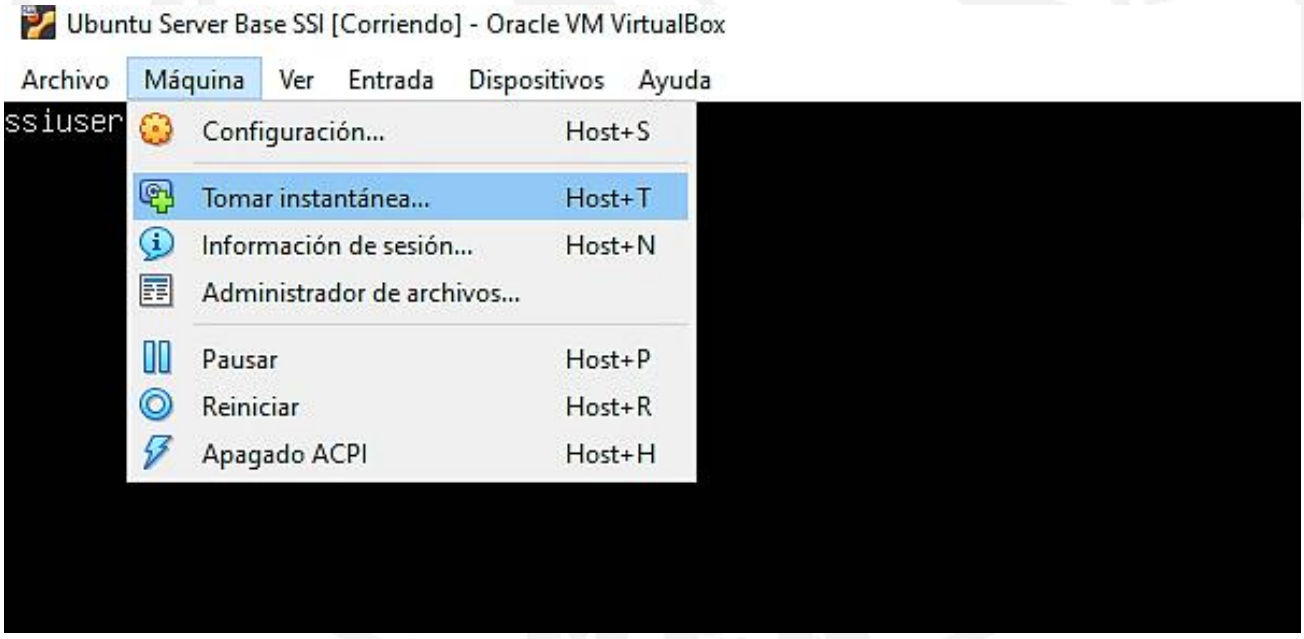
With snapshots, you can save a particular state of a virtual machine for later use. In any time, you can go back to any of those states, even if the VM has changed since then. Therefore, **a snapshot of a virtual machine is like a machine in the *Saved* state**, but there can be many of them and these saved states are preserved.

To view snapshots of a virtual machine, click the machine name in *VirtualBox Manager*. Then click the List icon next to the computer name and select **Snapshots**. Until a snapshot of the machine is taken, the snapshot list will be empty, except for the **Current Status** element, which represents the virtual machine as it is now.

Take, restore, and delete snapshots

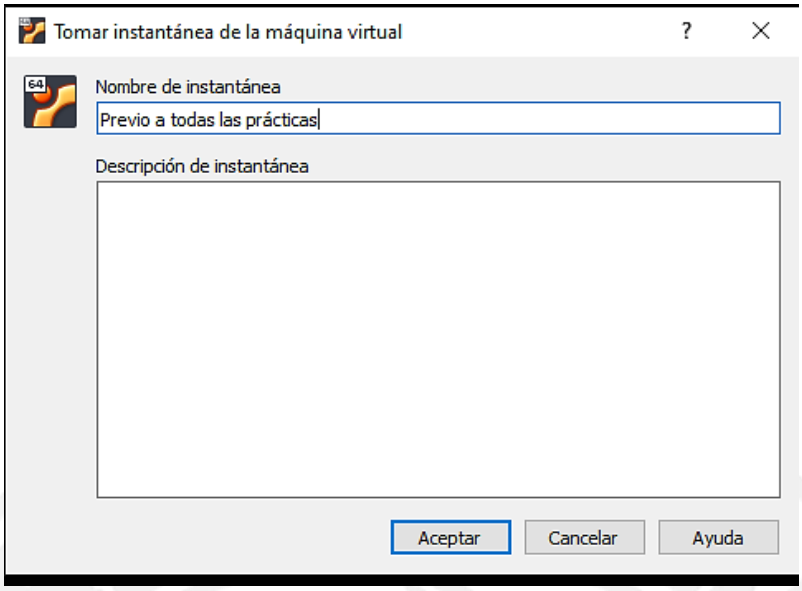
Take a snapshot

This makes a copy of the current state of the machine that can be returned to at any later time. **If the virtual machine is** running, select *Take Snapshot* from the *Machine* drop-down menu in the Virtual Machine window.



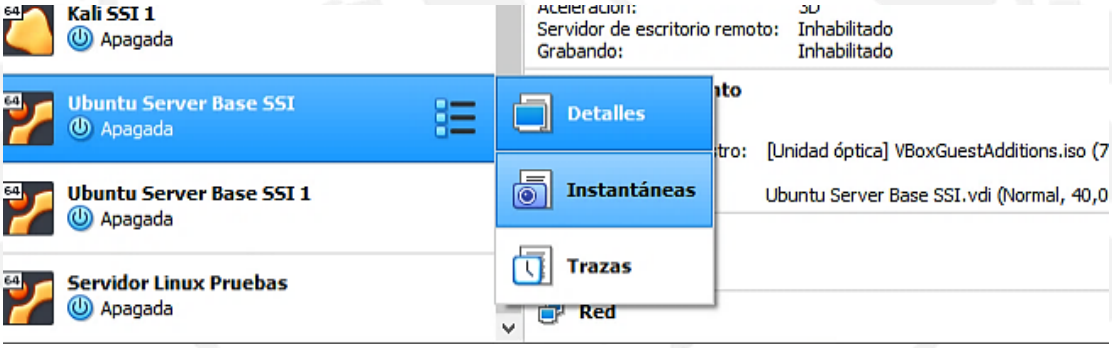
Take a snapshot of a working machine

This displays a window that requests a snapshot name. This name is purely for reference purposes, so it helps remember the state of the snapshot**. In our course it makes sense to generate states for each laboratory**, so that you can return to each of them independently and be able to start from the operations performed in each one if something goes wrong. You should also create a base state of the current machine to return to "zero state" if necessary. You can add a longer text in the *Description* field.



Name of a snapshot

If the virtual machine is in the *Saved* or *Off* state, in the name of the virtual machine in the VirtualBox main window, the List icon next to the machine name can be clicked and we can select under *Snapshots*. This shows the snapshot window where you can take a *snapshot* of the current state of the machine or return to one of the previously saved ones.



Take a snapshot of a stopped or saved machine

In any case, a snapshot list will appear. If below a snapshot an element called *Current State* appears, it means that the current state of the virtual machine is a variation of that snapshot. If another snapshot is taken later, they are displayed in sequence, and each subsequent snapshot is derived from a previous snapshot.

List of snapshots for a virtual machine

VirtualBox has no limits on the number of snapshots it can take beyond disk space on the host. Each snapshot stores the state of the virtual machine and therefore takes up disk space. However, it is much lower than an entire virtual machine (or one of its clones). For example, the *Ubuntu* base of the course may take about 4Gb

Logs	16/01/2020 11:55	Carpeta de archivos	
Snapshots	16/01/2020 12:03	Carpeta de archivos	
Ubuntu Server Base SSI	16/01/2020 12:03	VirtualBox Machin...	22 KB
Ubuntu Server Base SSI.vbox-prev	16/01/2020 12:03	Archivo VBOX-PREV	22 KB
Ubuntu Server Base SSI	16/01/2020 11:59	Virtual Disk Image	3.784.704 KB

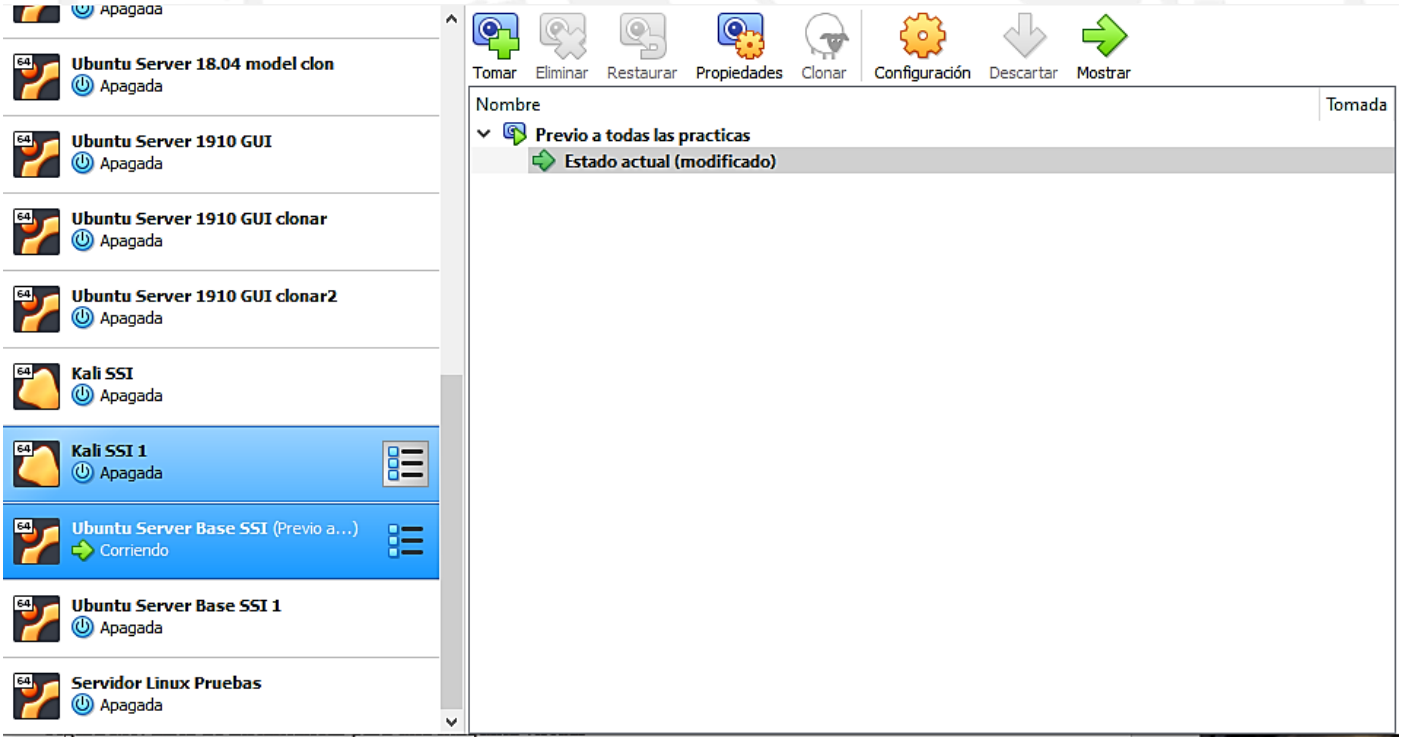
Space occupied by the *Ubuntu* VM or one of its clones

However, each snapshot of the machine we take (which is in the *Snapshots* folder of the path where the machine is saved) takes up significantly less space.

Nombre	Fecha de modificación	Tipo	Tamaño
{2b77b0ae-93b3-4952-82ae-3cd7a7abefaf}	16/01/2020 12:03	Virtual Disk Image	27.648 KB
{3be97ea4-94ba-4854-aedf-5f1bd4ac4cf9}	16/01/2020 12:05	Virtual Disk Image	19.456 KB
2020-01-16T10-59-52-226259100Z.sav	16/01/2020 11:59	Archivo SAV	588.905 KB
2020-01-16T11-03-25-769435400Z.sav	16/01/2020 12:03	Archivo SAV	588.969 KB

Space occupied by the *Ubuntu* VM Snapshots

The machine running from the base state will thus appear in the list of snapshots

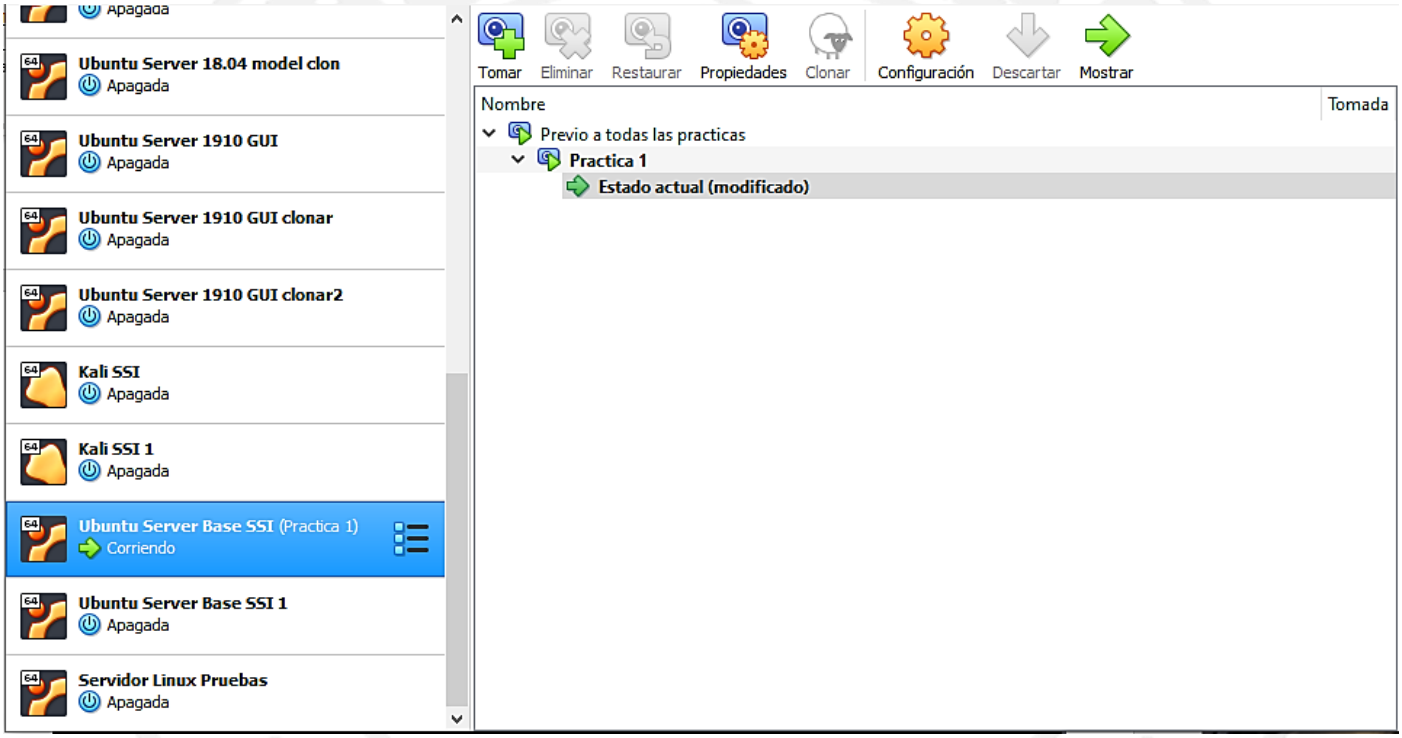


List of snapshots

If we now write a text file on the hard drive and take a snapshot, the new state of the machine will contain this file, while the base state will not.

```
ssiuser@ubuntussi:~$ ls
ficheroprac1.txt
ssiuser@ubuntussi:~$ ls -la
total 40
drwxr-xr-x 5 ssiuser ssiuser 4096 Jan 16 11:02 .
drwxr-xr-x 3 root    root    4096 Jan 13 19:29 ..
-rw------- 1 ssiuser ssiuser   94 Jan 13 19:55 .bash_history
-rw-r--r-- 1 ssiuser ssiuser  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 ssiuser ssiuser 3771 Apr  4  2018 .bashrc
drwx------ 2 ssiuser ssiuser 4096 Jan 13 19:34 .cache
-rw-rw-r-- 1 ssiuser ssiuser   10 Jan 16 11:02 ficheroprac1.txt
drwx------ 3 ssiuser ssiuser 4096 Jan 13 19:34 .gnupg
drwxrwxr-x 3 ssiuser ssiuser 4096 Jan 16 11:02 .local
-rw-r--r-- 1 ssiuser ssiuser  807 Apr  4  2018 .profile
-rw-r--r-- 1 ssiuser ssiuser    0 Jan 13 19:35 .sudo_as_admin_successful
ssiuser@ubuntussi:~$ _
```

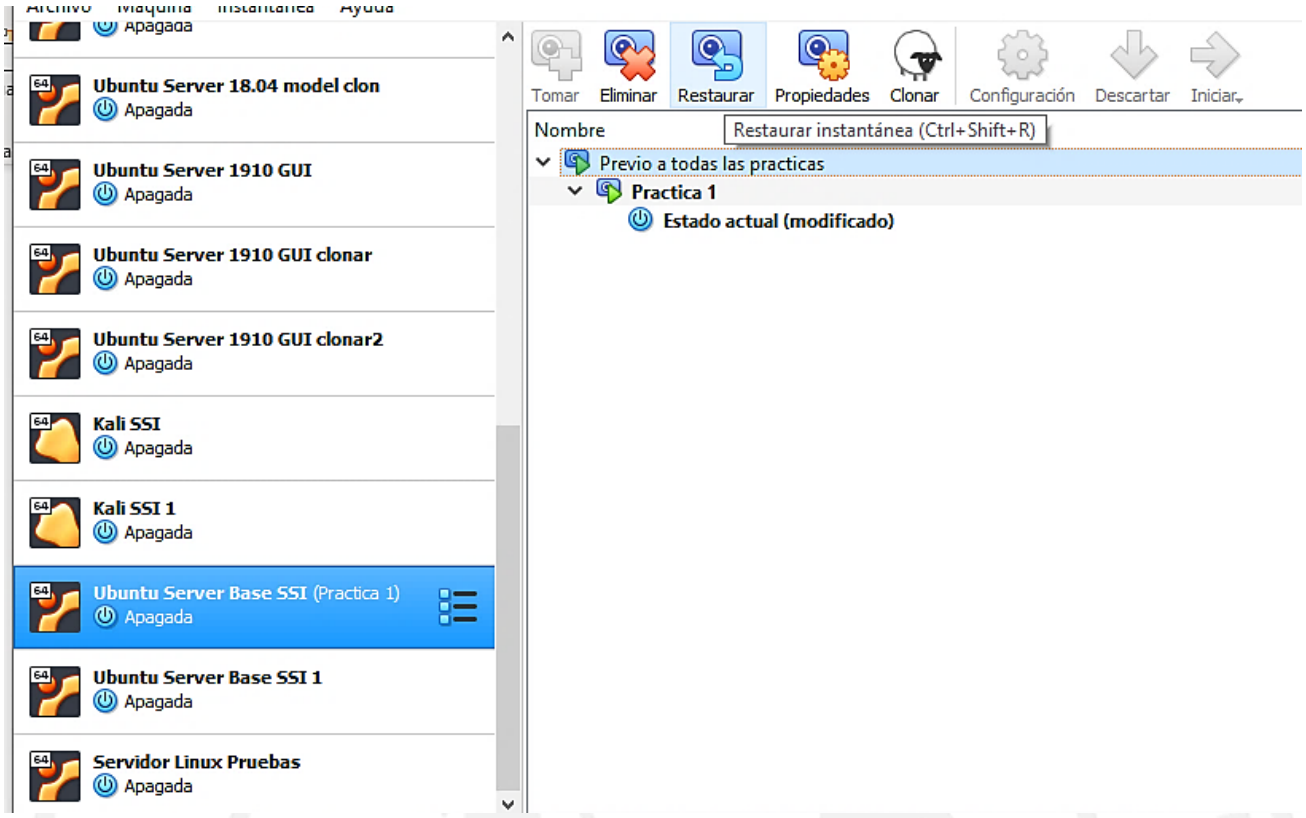
We see that when taking the snapshot, the current state appears below it.



New snapshot in snapshot list

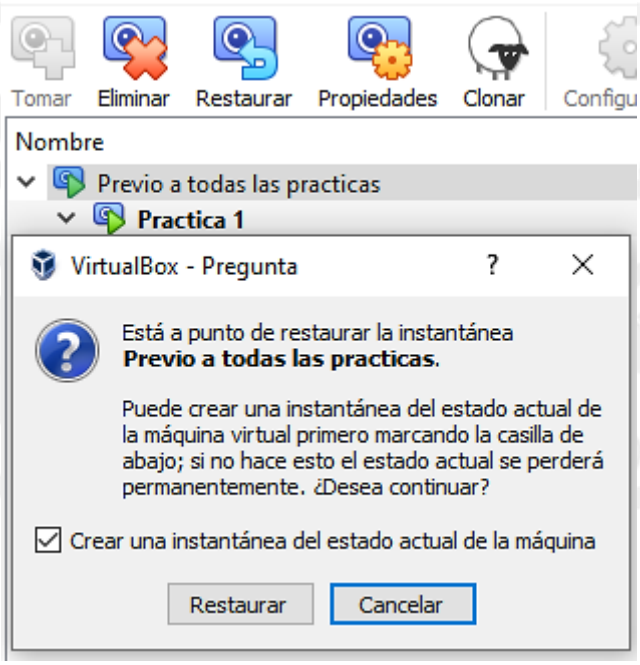
Restore a snapshot

With the machine saved or turned off we can go to the list of snapshots, right-click on any snapshot and select *Restore*. When you restore a snapshot, it goes back or forward in time depending on when it has been taken against the current state. The current state of the machine is lost, and the machine is restored to the exact state it was in when the snapshot was taken.



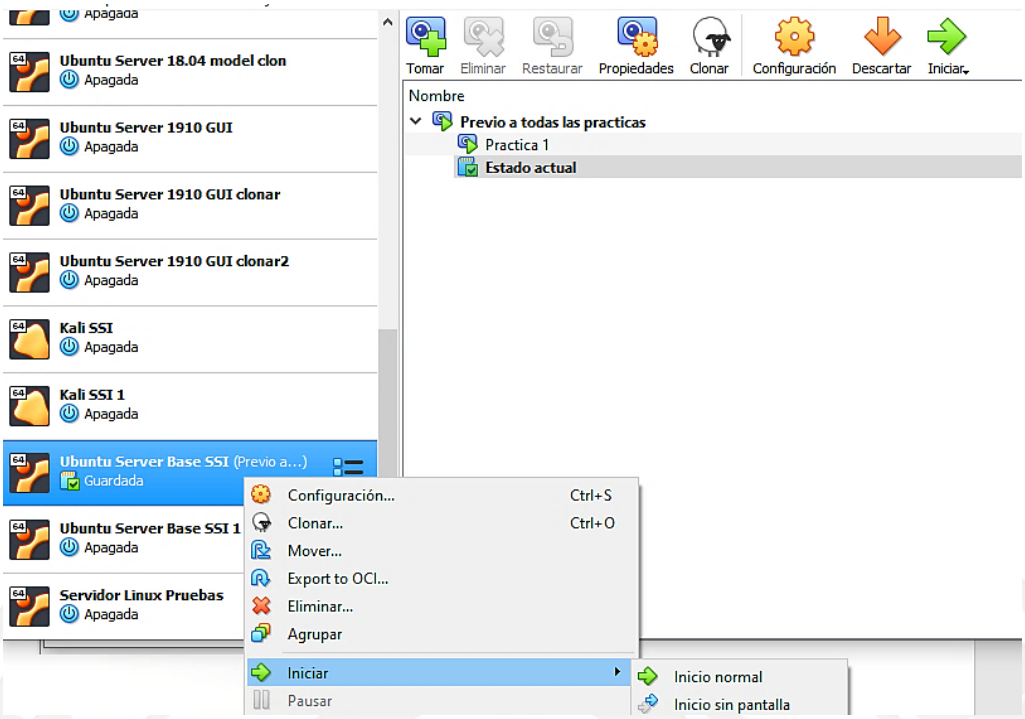
Restoring a snapshot

This loss of the current state of the machine can be a problem in some contexts, and therefore *VirtualBox* asks first if you want to take a snapshot of the current state before restoring the one, we have selected. In this case we will say no.



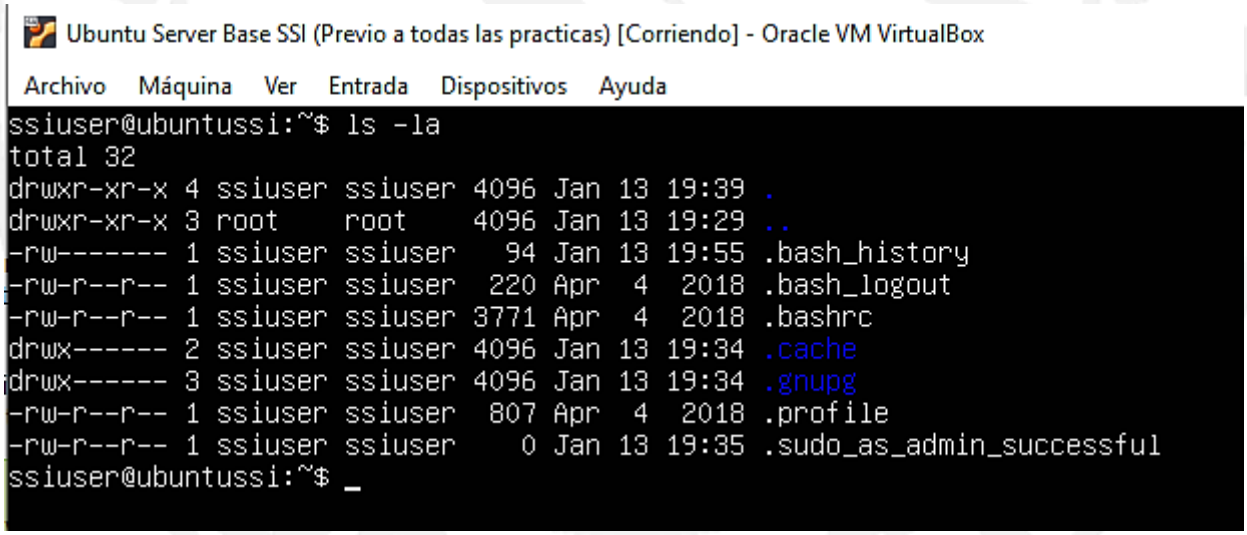
Notice of loss of the current state of the machine by restoration

In this example we have returned to the initial state (it appears in bold) and once this is done, we can start the machine again.



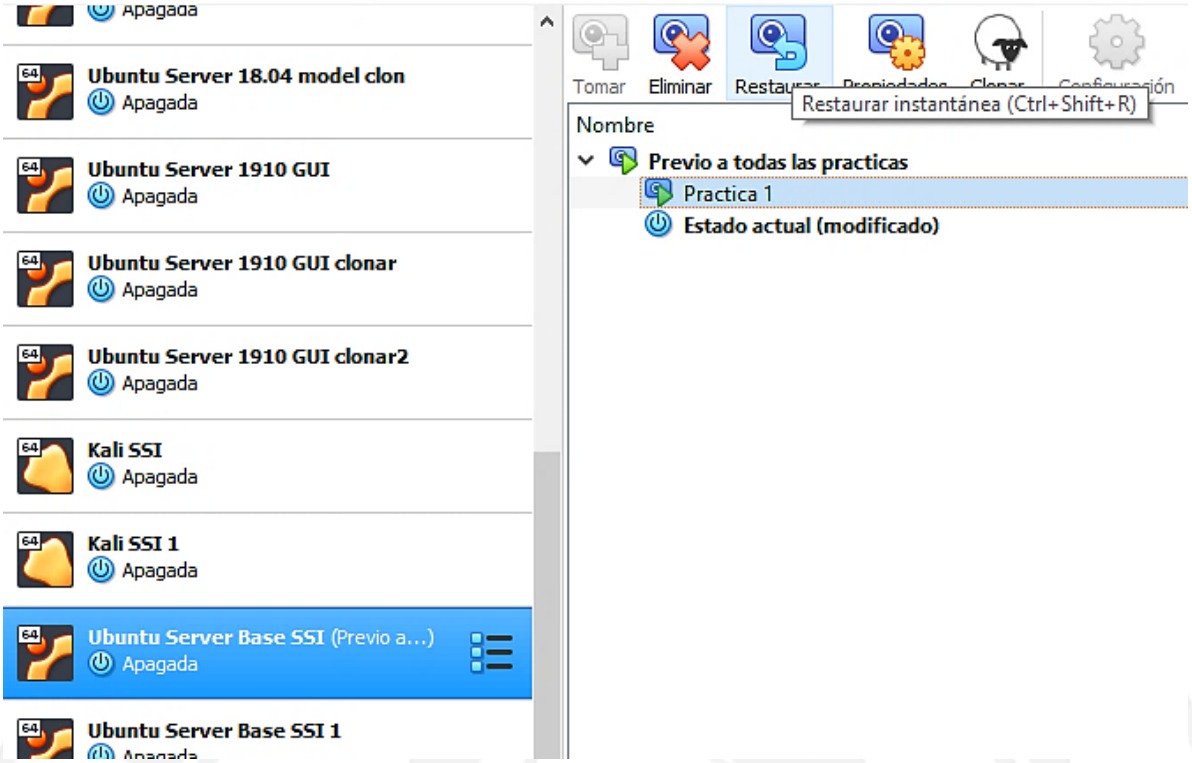
Starting the machine in the base snapshot

When we start it, we see that the machine no longer has the file that we had created, because that was only done in the Laboratory 1 state.

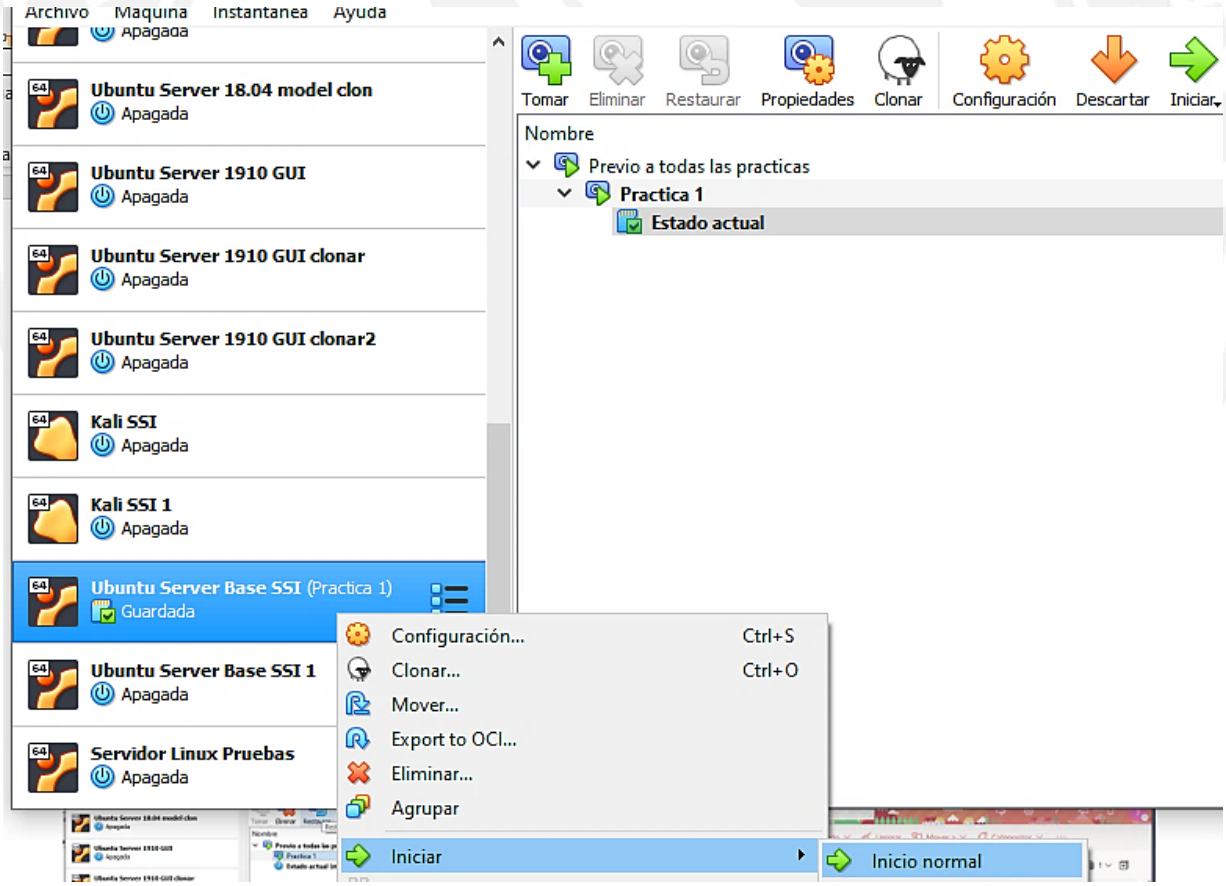


Machine in its original state

Likewise, we can restore laboratory 1 again by repeating the process and recovering the changes of the state (in this case our file)



Change to later status

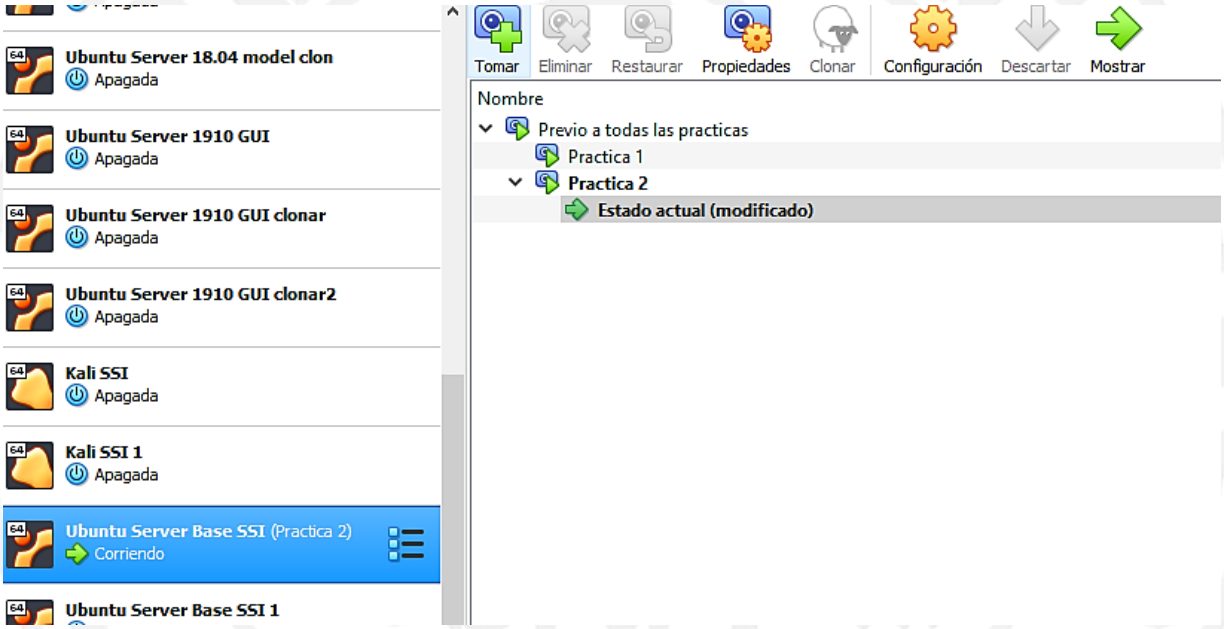


Place to restore a later state

```
ssiuser@ubuntussi:~$ ls -la
total 40
drwxr-xr-x 5 ssiuser ssiuser 4096 Jan 16 11:02 .
drwxr-xr-x 3 root    root    4096 Jan 13 19:29 ..
-rw------- 1 ssiuser ssiuser   94 Jan 13 19:55 .bash_history
-rw-r--r-- 1 ssiuser ssiuser  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 ssiuser ssiuser 3771 Apr  4  2018 .bashrc
drwx------ 2 ssiuser ssiuser 4096 Jan 13 19:34 .cache
-rw-rw-r-- 1 ssiuser ssiuser   10 Jan 16 11:02 ficheroprac1.txt
drwx------ 3 ssiuser ssiuser 4096 Jan 13 19:34 .gnupg
drwxrwxr-x 3 ssiuser ssiuser 4096 Jan 16 11:02 .local
-rw-r--r-- 1 ssiuser ssiuser  807 Apr  4  2018 .profile
-rw-r--r-- 1 ssiuser ssiuser    0 Jan 13 19:35 .sudo_as_admin_successful
ssiuser@ubuntussi:~$ _
```

Subsequent status with changes made

Now we could repeat the same process for the rest of the laboratories and proceed in the same way, making a list of *snapshots* that help us to follow the course.



List of snapshots by laboratory

To learn more about the contents of a *snapshot* and other details, we recommend consulting: https://www.virtualbox.org/manual/ch01.html#snapshots

Shared folders

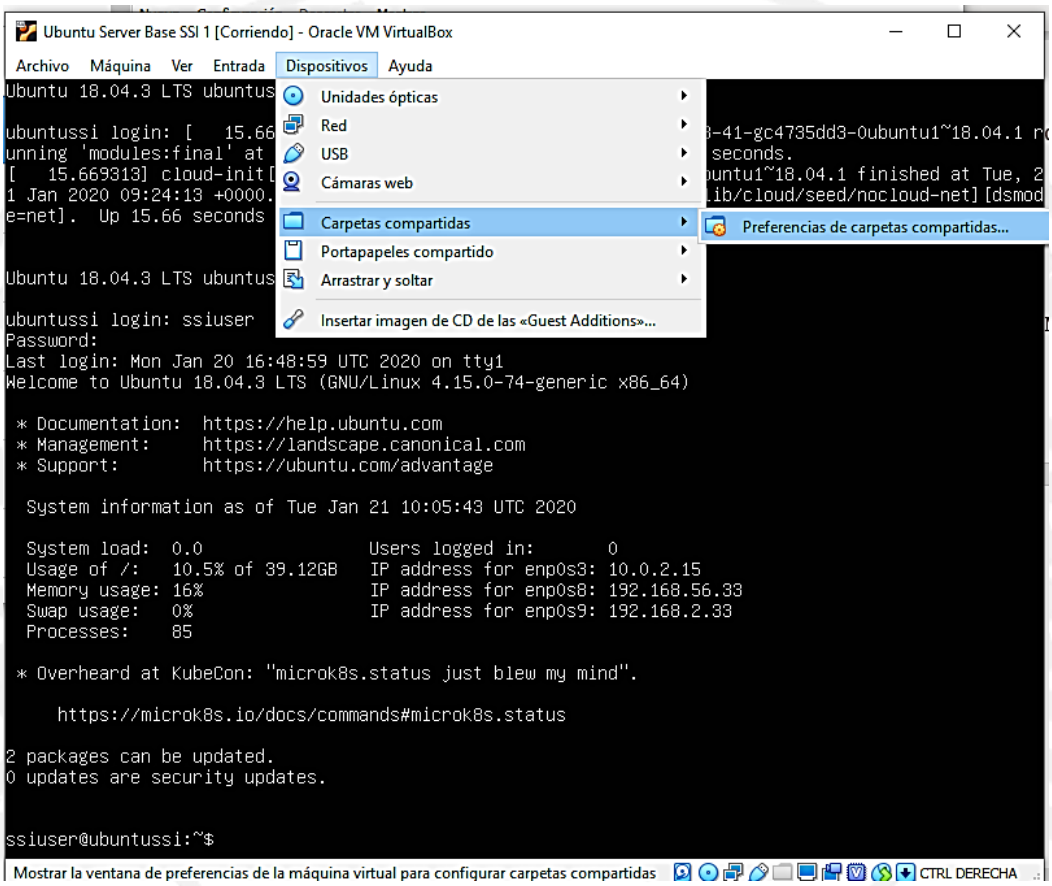
NOTE: This is not necessary if you obtained your VM using Option 1, as it has shared folders already configured.

One of the most typical ways to share information between a virtual machine and its host is by creating shared folders. This mechanism allows the creation of a folder on the virtual machine, so its contents are shared with the host:

- Everything on the host that is copied to or created in that folder will be readable, editable, and copiable from the virtual machine

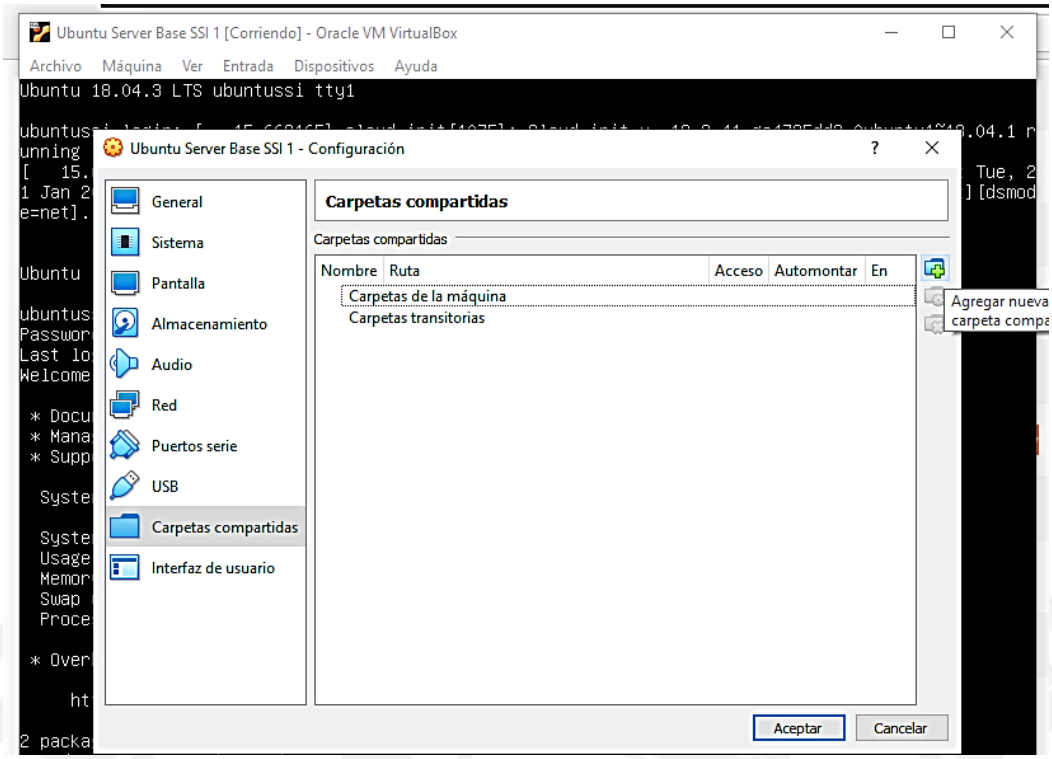
- Consequently, everything we copy or create on the virtual machine in that folder will be readable, editable, and copiable from the host

This is therefore a two-way file transfer mechanism suitable for many uses. It has only two conditions: **have the *Guest Additions* previously installed** and create and map the shared folder appropriately. The latter depends on the virtual machine and what you have installed. If the machine has GUI, as soon as we create the folder, we can access it from the host's file explorer and work with it as if it is a local folder. However, if we do not have GUI, we must do several additional steps to achieve access to it. In any case, the first thing is to create the shared folder, and for this we will go to "*Devices – Shared Folders – Shared Folder Preferences*" (it does not matter if the machine is powered on or not).



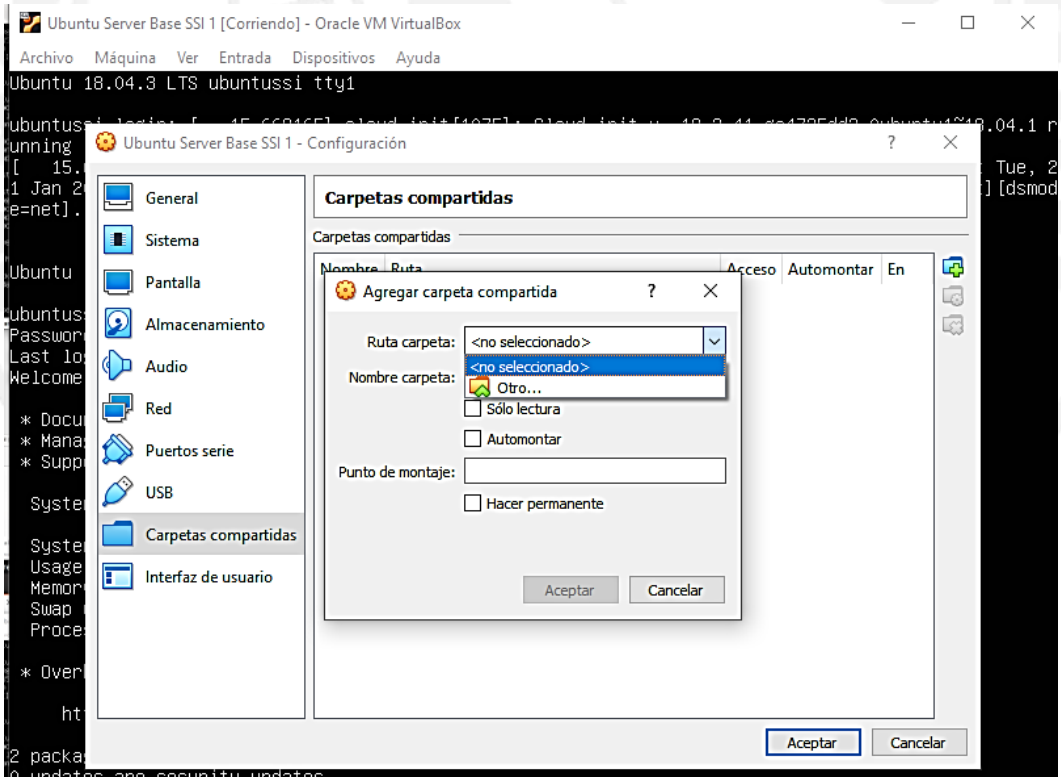
Access to a *VirtualBox* machine's shared folder interface

This shows a screen that ask us to create machine folder (potentially permanent) or a transient one (not permanent). We chose the first option



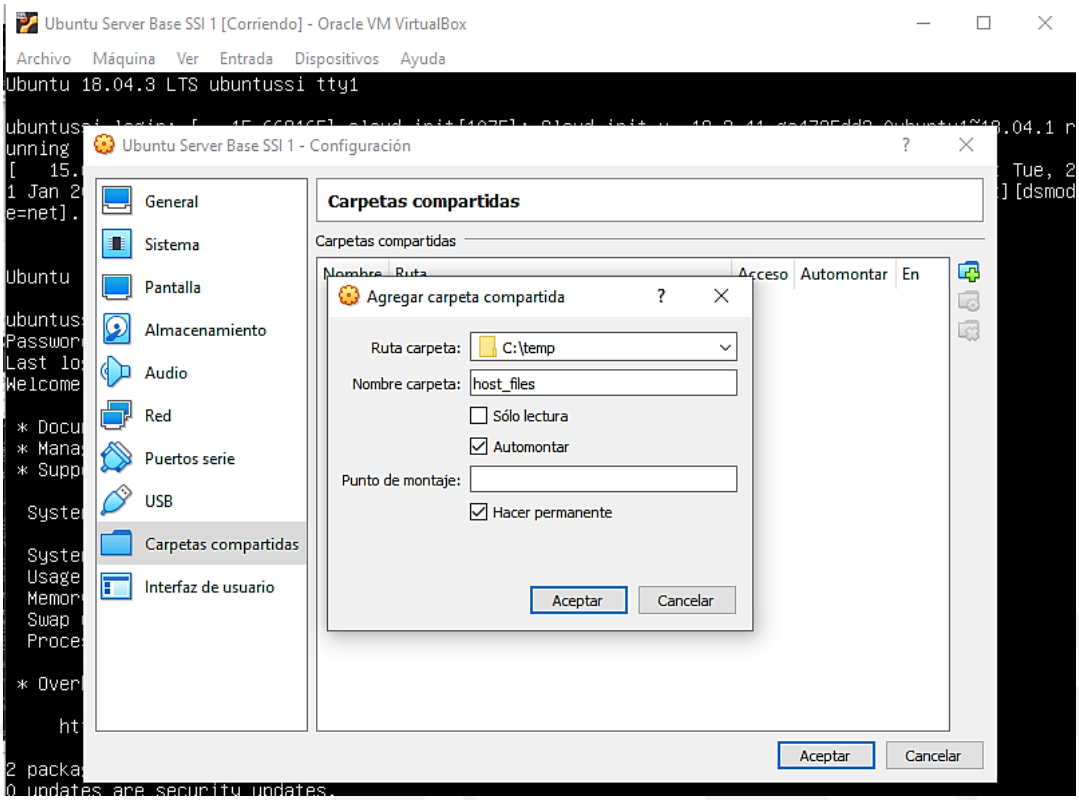
Folders currently shared on a VirtualBox machine

Folder path is the host folder that we want to use to write or read content, that is, which place on our hard drive we will use for files that go to or from the machine.



Selecting a host folder

We select "*Other*" and an existing folder (`c:/temp` in the example). We name the shared folder (`host_files`, name to be used on the VM side) and check the "*Automount*" options and "*Make permanent*" for easier handling. Once this is done, we accept.

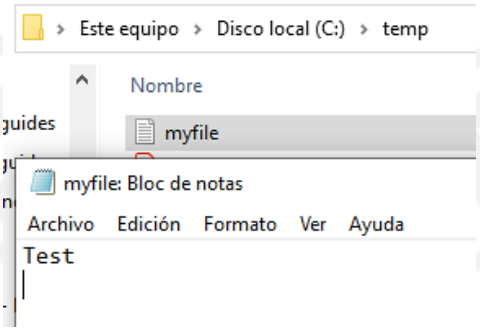


Full configuration of a VirtualBox shared folder

The next step is to create a directory on the virtual machine where the host contents will be viewed. We create the "`shared`" directory in the current user's home with `mkdir ~/shared` and then mount the shared folder created earlier over the directory we just created in the MV (here we use the name specified in "Folder Name" in the VirtualBox interface: `sudo mount -t vboxsf host_files ~/shared` (`host_data` if using the *Vagrant* machine)

```
ssiuser@ubuntussi:~$ sudo mount -t vboxsf host_files ~/shared
```

Now the host folder will be fully accessible from the VM, and you can copy or extract the files using it. For example, if we edit a text file on the virtual machine with `nano myfile.txt` inside that folder and put the text "`Test`", the *Windows* host will display it as well.



Shared folder viewed from a Windows host

In case the folder is not permanently mounted on the machine, we can use the technique explained here to make it persistent: https://gist.github.com/estorgio/1d679f962e8209f8a9232f7593683265