



UNIVERSIDAD SANTO TOMÁS

Facultad de Ingeniería Electrónica

Proyecto Final: Infraestructura de Red y Virtualización

Estudiantes: Yojan Contreras - Cristian Losada

Docente: Ing. Diego Alejandro Barragán Vargas

Bogotá D.C., Octubre 2025

1. Introducción del proyecto

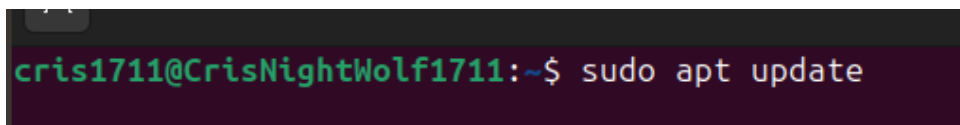
Una empresa tecnológica A contrata a dos estudiantes de la Universidad Santo Tomás como ingenieros DevOps que tengan habilidades también en redes, es decir con cualidades de NetOps para la creación de la infraestructura que requieren en la conexión y comunicación de sus diferentes dependencias, dentro de las que se encuentran:

- Dependencia de Recursos Humanos
- Dependencia de Tecnología
- Dependencia Financiera
- Dependencia Comercial y Ventas

2. Preparación Inicial

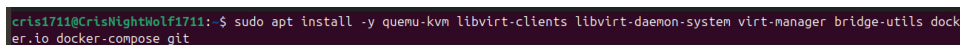
2.1. Instalación de herramientas básicas

Hacemos el proceso de instalación de las herramientas básicas (qemu) para poder instalar lo que necesitaremos



```
cris1711@CrisNightWolf1711:~$ sudo apt update
```

Figure 1: Actualización de recursos del computador



```
cris1711@CrisNightWolf1711:~$ sudo apt install -y qemu-kvm libvirt-clients libvirt-daemon-system virt-manager bridge-utils docker.io docker-compose git
```

Figure 2: Instalación de herramientas básicas como qemu y Docker

3. Máquinas Virtuales y contenedores centrales

3.1. Objetivo

La empresa tecnológica A le solicita al ingeniero los siguientes requerimientos:

3.1.1 Crear la máquina virtual (Debian) como máquina del Administrador

- Ejecutamos el virt manager y lo abrimos

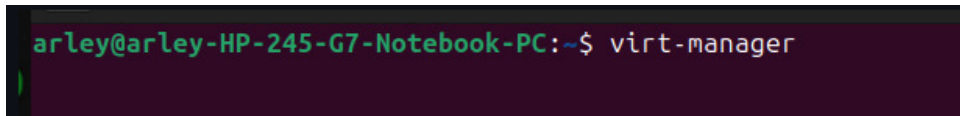


Figure 3: Interfaz del virt-manager para las maquinas virtuales

- Seleccionamos la seccion de archivo para crear una nueva maquina virtual

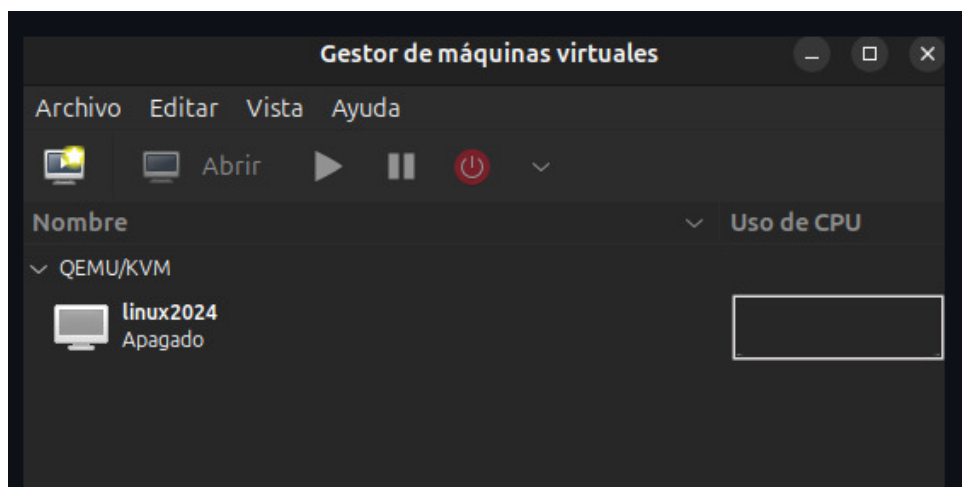


Figure 4: Damos archivo para crear maquina virtual

- Seleccionamos nueva maquina virtual

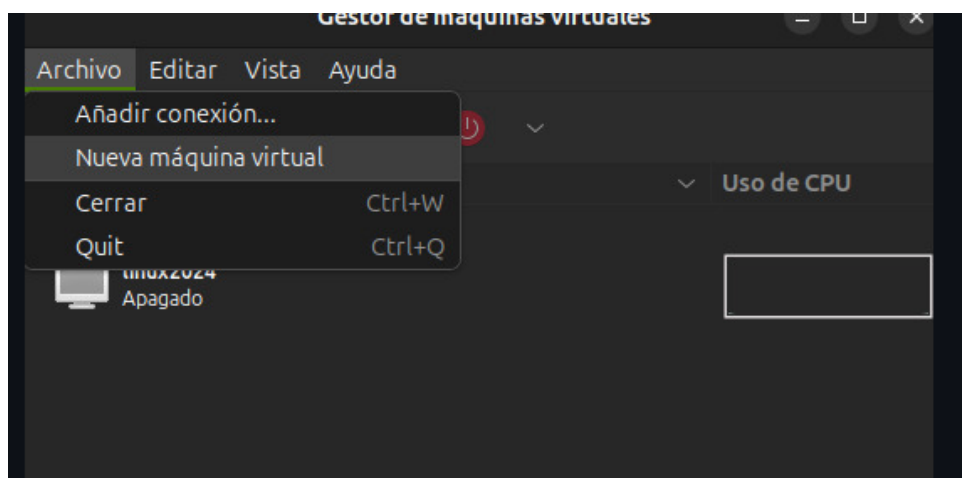


Figure 5: Creamos la máquina virtual

- Seleccionamos crear una nueva maquina virtual

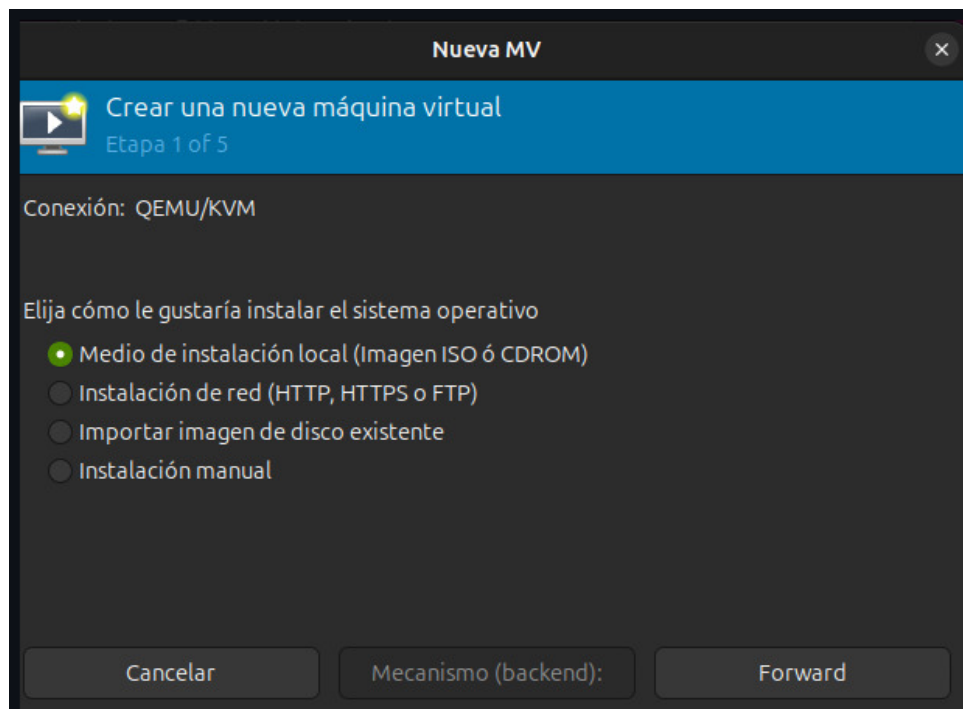


Figure 6: Seleccionamos Forward

- Le damos cuanta memoria queremos que tenga la maquina virtual y cuanta cpu nos va a consumir

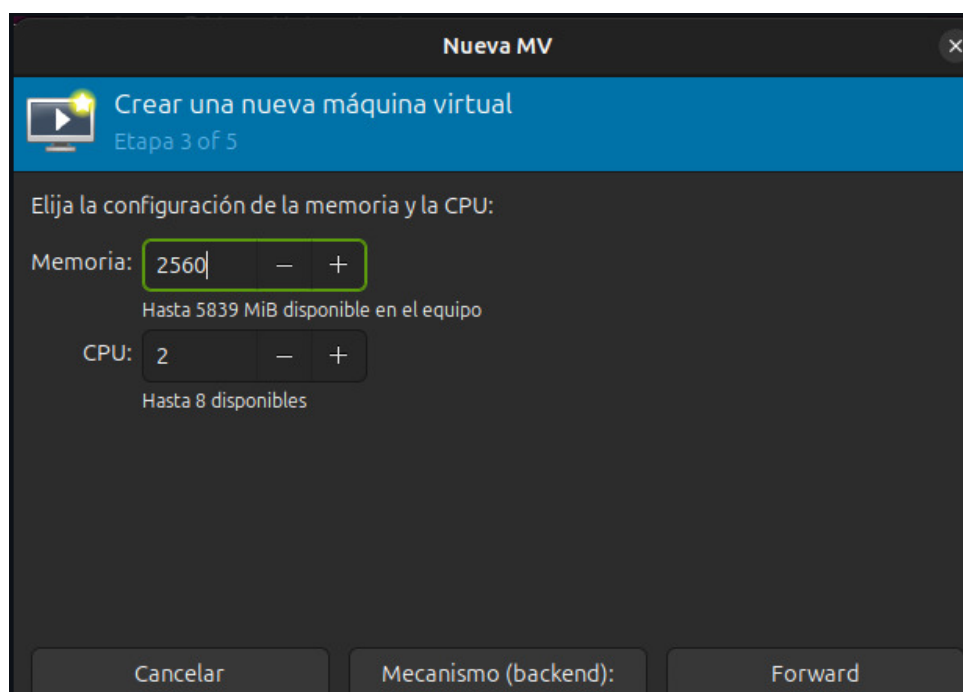


Figure 7: Le asignamos memoria y CPU

- Le añadimos tambien cuanto espacio en el disco queremos que consuma la VM

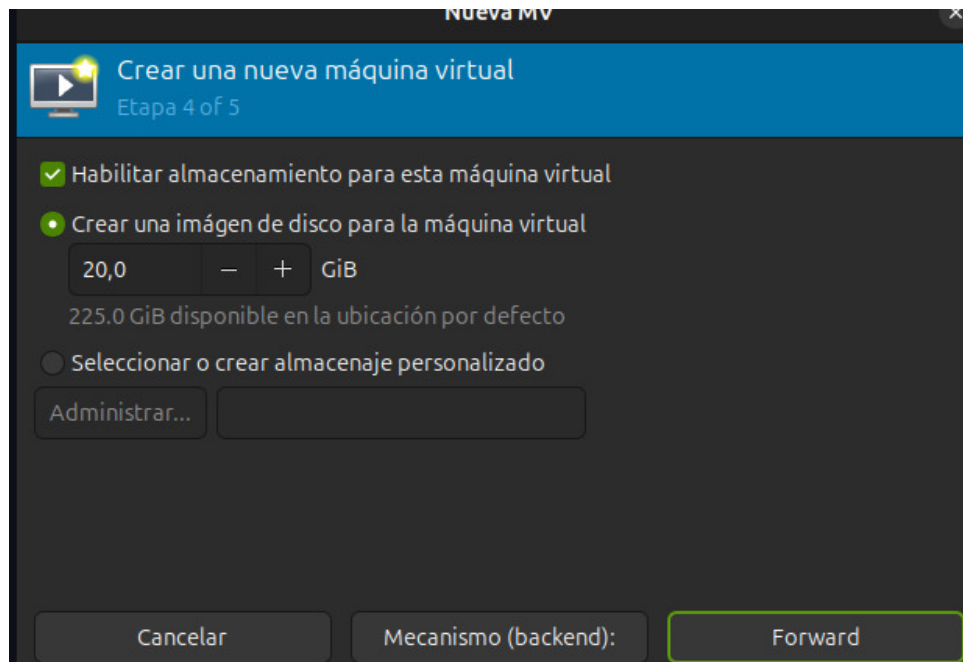


Figure 8: Le asignamos espacio en el disco

- Comprobamos que todo haya quedado bien.

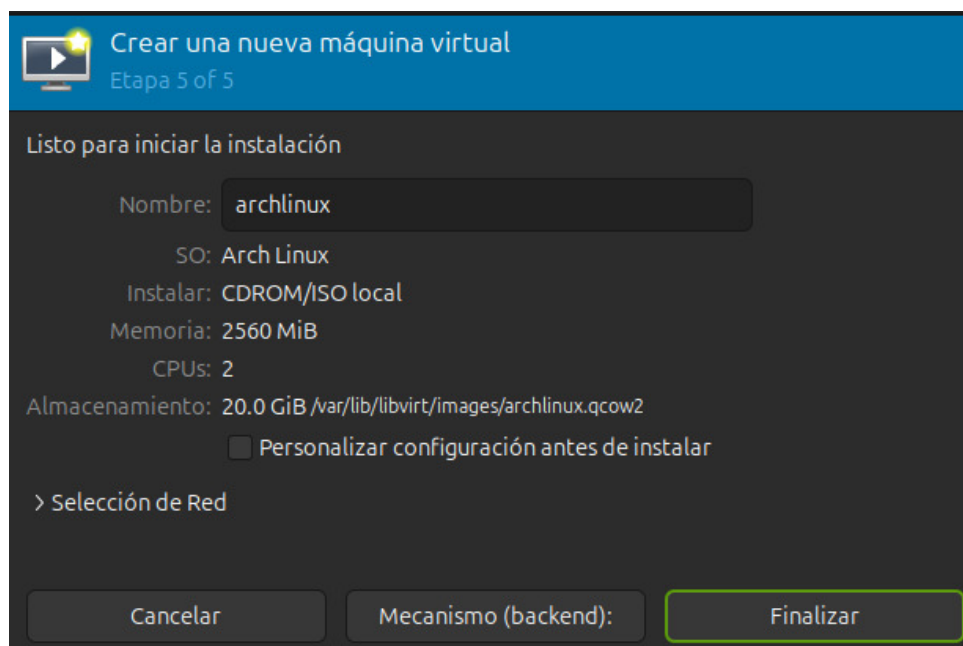


Figure 9: Comprobamos y aceptamos

- Después del proceso de instalación ya tenemos la VM de Debian funcionando

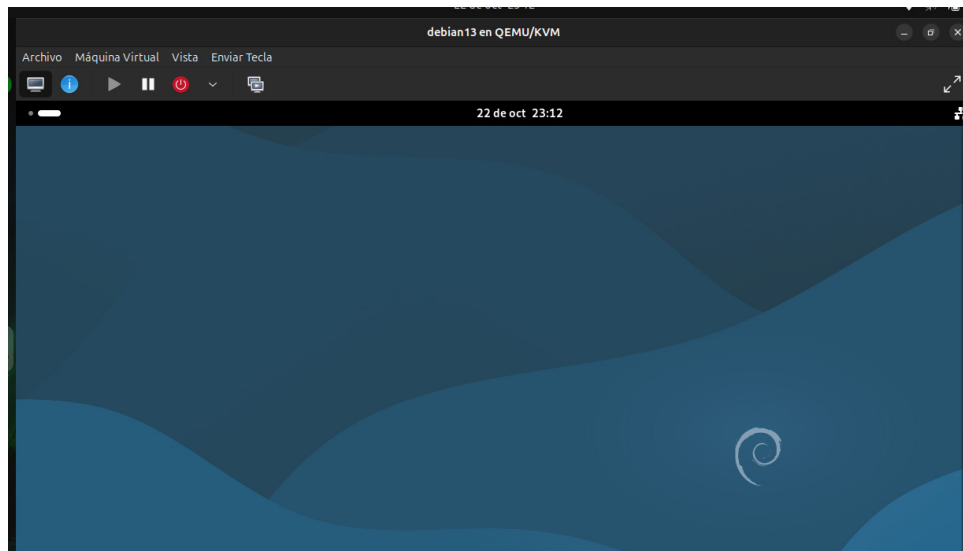


Figure 10: VM Debian creada exitosamente

3.1.2 Máquina Virtual Arch Linux (nodo Recursos Humanos)

Es el mismo proceso de instalacion para todas las VM en virt manager. Por lo que el proceso sera el mismo para las demas.

- VM de Arch Linux funcionando correctamente

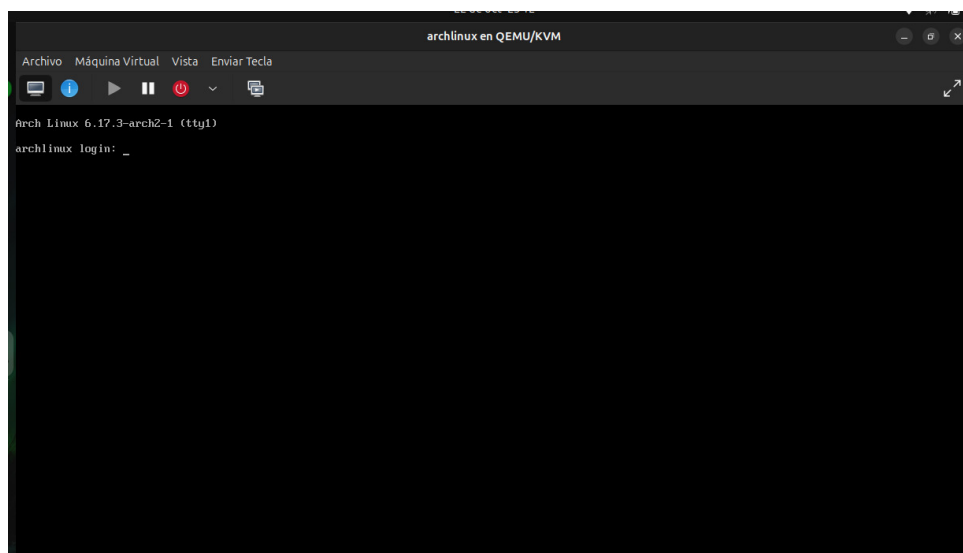


Figure 11: VM ArchLinux configurada

3.1.3 Contenedor Fedora (dependencia Comercial y Ventas)

- Se realiza el proceso de descargar la imagen de fedora en docker

```
arley@arley-HP-245-G7-Notebook-PC:~/pybullet_docker/baxter$ docker pull fedora
Using default tag: latest
latest: Pulling from library/fedora
30f8cfaf47ed: Pull complete
Digest: sha256:aa7befe5cfd1f0e062728c16453cd1c479d4134c7b85eac00172f3025ab0d522
Status: Downloaded newer image for fedora:latest
docker.io/library/fedora:latest
```

Figure 12: Descargamos la imagen de fedora

- Realizamos la creacion del contenedor con fedora y entramos al contenedor para visualizarlo

```
arley@arley-HP-245-G7-Notebook-PC:~/pybullet_docker/baxter$ docker run -it --name fedora_rh fedora bash
[root@2345f5f4772e /]#
```

Figure 13: Creamos y entramos al contenedor

- Realizamos la verificacion de que estamos dentro del contenedor

```
arley@arley-HP-245-G7-Notebook-PC:~/pybullet_docker/baxter$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PO
RTS		NAMES			
2345f5f4772e	fedora	"bash"	31 seconds ago	Exited (127) 2 seconds ago	
		fedora_rh			

Figure 14: Verificamos

3.1.4 Contenedor Garuda (dependencia financiera)

- Garuda no es oficial den Docker Hub usaremos una imagen basada en Arch Linux
- Descargamos la imagen basica de Arch Linux en Docker

```
cris1711@CrisNightWolf1711:~$ sudo docker pull archlinux:latest
```

Figure 15: Descargar imagen base

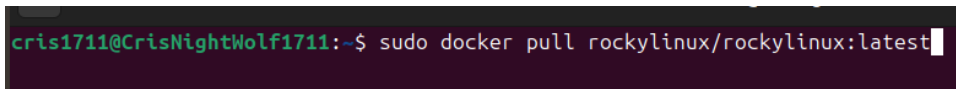
- Ejecutamos un contenedor interactivo en Docker

```
cris1711@CrisNightWolf1711:~$ sudo docker run -it --name cont-garuda archlinux:latest /bin/bash
```

Figure 16: Descargar imagen base

3.1.5 VM Rocky Linux (dependencia Tecnologia)

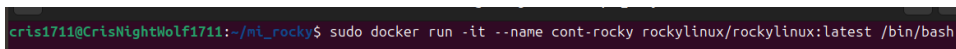
- Para esta maquina virtual debido a que no presentamos buenos computadores para realizar el proceso de instalacion, lo vamos a realizar como maquina virtual.
- Descargamos la ultima version estable de Rocky.



```
cris1711@CrisNightWolf1711:~$ sudo docker pull rockylinux/rockylinux:latest
```

Figure 17: Descargar imagen de Rocky Linux

- Iniciamos el contenedor de Rocky Linux en Docker



```
cris1711@CrisNightWolf1711:~/ni_rocky$ sudo docker run -it --name cont-rocky rockylinux/rockylinux:latest /bin/bash
```

Figure 18: Ejecutamos el contenedor con la imagen de Rocky Linux

4. Instalaciones en cada dependencia (maquinas/contenedores adicionales)

Tenemos en cuenta la arquitectura anterior que designamos con vm y contenedores la empresa nos solicita lo siguiente:

4.1. Recursos HUmanos

- VM Manjaro

El proceso de instalacion de la maquina virtual de Manjaro es la misma realizada anteriormente en el virt-manager

- Maquina vm Manjaro funcionando

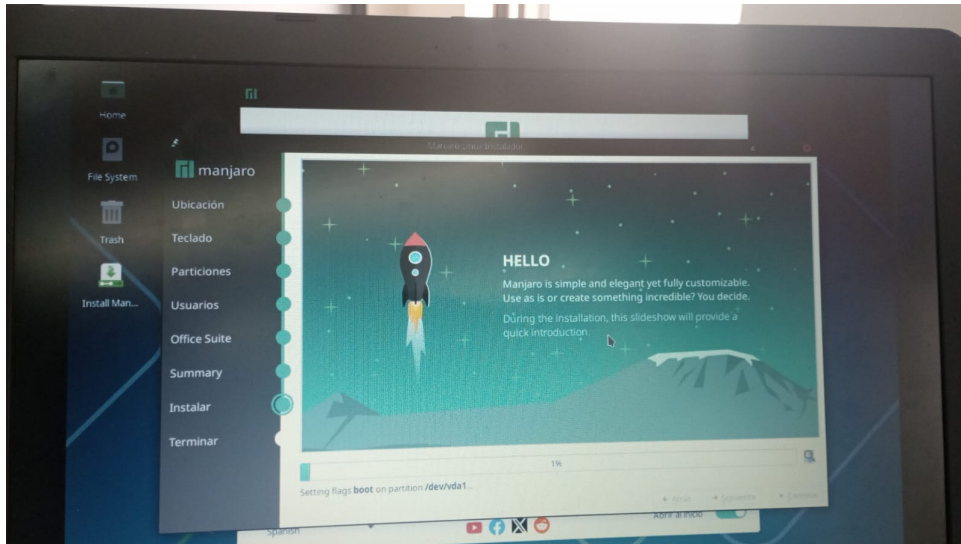


Figure 19: Ejecutamos el contenedor con la imagen de Rocky Linux

- Contenedor CenTOS

Descargamos la imagen de Centos 8 en Docker

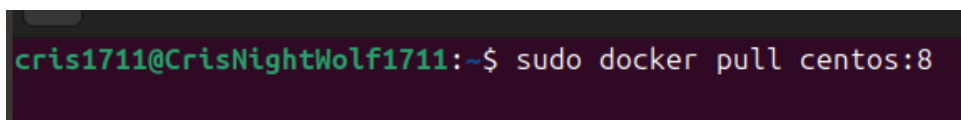


Figure 20: Descargamos imagen de Centos 8

- Iniciamos el contenedor con Centos 8

4.2. Tecnología

- Maquina Virtual Kali

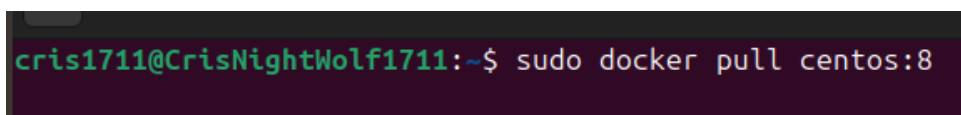


Figure 21: Vm de Kali Linux

5. Configuración de Red y Conectividad

5.1. Configuración de interfaces de red

Se procedió a configurar las interfaces de red para cada máquina virtual y contenedor, asegurando la comunicación entre todas las dependencias.

Ya que por motivos de RAM de nuestros pcs se realiza la conexion de dos computadores mediante switch para la realizacion del proyecto

```
Switch(config)#vlan 10
Switch(config-vlan)#name RH
Switch(config-vlan)#vlan 20
Switch(config-vlan)#name TEC
Switch(config-vlan)#vlan 30
Switch(config-vlan)#name FIN
Switch(config-vlan)#vlan 40
Switch(config-vlan)#name SALES
Switch(config-vlan)#vlan 99
Switch(config-vlan)#name MANAGEMENT
Switch(config-vlan)#
```

Figure 22: Configuracion inicial del switch

```
Switch(config-if)#interface FastEthernet0/1
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 99
Switch(config-if)#switchport trunk allowed vlan 10,20,30,40
Switch(config-if)#switchport trunk native vlan 99
Switch(config-if)#spanning-tree portfast trunk
```

Figure 23: Configuracion Laptop 1

```
Switch(config-if)#interface FastEthernet0/2
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 10,20,30,40,99
Switch(config-if)#switchport trunk native vlan 99
Switch(config-if)#spanning-tree portfast trunk
```

Figure 24: Configuracion laptop 2

```
Switch#show vlan brief
```

VLAN	Name	Status	Ports
1	default	active	Fa0/3, Fa0/4, Fa0/5, Fa0/6 Fa0/7, Fa0/8, Fa0/9, Fa0/10 Fa0/11, Fa0/12, Fa0/13, Fa0/14 Fa0/15, Fa0/16, Fa0/17, Fa0/18 Fa0/19, Fa0/20, Fa0/21, Fa0/22 Fa0/23, Fa0/24, Gi0/1, Gi0/2
10	RH	active	
20	TEC	active	
30	FIN	active	
40	SALES	active	
99	MANAGEMENT	active	
1002	fddi-default	act/unsup	
1003	token-ring-default	act/unsup	
1004	fddinet-default	act/unsup	
1005	trnet-default	act/unsup	

```
Switch#
```

Figure 25: Comprobamos que quedaron bien

```
Switch#show interfaces trunk
```

Port	Mode	Encapsulation	Status	Native vlan
Fa0/1	on	802.1q	trunking	99
Fa0/2	on	802.1q	trunking	99

```
Port Vlan allowed on trunk
Fa0/1 10,20,30,40,99
Fa0/2 10,20,30,40,99

Port Vlan allowed and active in management domain
Fa0/1 10,20,30,40,99
Fa0/2 10,20,30,40,99

Port Vlan in spanning tree forwarding state and not pruned
Fa0/1 10,20,30,40,99
Fa0/2 10,20,30,40,99
Switch#
```

Figure 26: Verificamos

```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo apt update && sudo apt install -y vlan bridge-utils
[sudo] contraseña para arley:
```

Figure 27: Instalamos Paquetes

```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo modprobe 8021q
arley@arley-HP-245-G7-Notebook-PC:~$
```

Figure 28: Activamos las VLANS

```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add link eno1 name eno1.10 type vlan id 10
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add link eno1 name eno1.20 type vlan id 20
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add link eno1 name eno1.30 type vlan id 30
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add link eno1 name eno1.40 type vlan id 40
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add link eno1 name eno1.99 type vlan id 99
arley@arley-HP-245-G7-Notebook-PC:~$
```

Figure 29: Creamos las interfases VLAN

```

arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add name br10 type bridge
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add name br20 type bridge
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add name br30 type bridge
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add name br40 type bridge
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link add name br99 type bridge
arley@arley-HP-245-G7-Notebook-PC:~$

```

Figure 30: Se crean los bridges

```

arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set eno1.10 master br10
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set eno1.20 master br20
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set eno1.30 master br30
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set eno1.40 master br40
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set eno1.99 master br99
arley@arley-HP-245-G7-Notebook-PC:~$

```

Figure 31: Se unen las VLANs a sus bridges

```

arley@arley-HP-245-G7-Notebook-PC:~$ for i in eno1 eno1.10 eno1.20 eno1.30 eno1.40 eno1.99 br10 br20 br30 br40 br99; do sudo ip
link set dev $i up || true
> done

```

Figure 32: Se activan todas las interfases que tenemos

```

arley@arley-HP-245-G7-Notebook-PC:~$ ip -d link show type vlan
9: eno1.10@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br10 state UP mode DEFAULT group default qlen 1
000
    link/ether f8:0d:ac:49:b9:b0 brd ff:ff:ff:ff:ff:ff promiscuity 1 allmulti 1 minmtu 0 maxmtu 65535
    vlan protocol 802.1Q id 10 <REORDER_HDR>
    bridge_slave state forwarding priority 32 cost 19 hairpin off guard off root_block off fastleave off learning on flood on po
rt_id 0x8001 port_no 0x1 designated_port 32769 designated_cost 0 designated_bridge 8000.66:7b:ab:fb:e5:2a designated_root 8000.6
6:7b:ab:fb:e5:2a hold_timer 0.00 message_age_timer 0.00 forward_delay_timer 0.00 topology_change_ack 0 config_pending 0
    proxy_arp off proxy_arp_wifi off mcast_router 1 mcast_fast_leave off mcast_flood on bcst_flood on mcast_to_unicast off neigh_s
uppress off group_fwd_mask 0 group_fwd_mask_str 0x0 vlan_tunnel off isolated off locked off addrngmode eui64 numtxqueues 1 numr
xqueues 1 gso_max_size 64000 gso_max_segs 64 tso_max_size 64000 tso_max_segs 64 gro_max_size 65536
10: eno1.20@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br20 state UP mode DEFAULT group default qlen
1000
    link/ether f8:0d:ac:49:b9:b0 brd ff:ff:ff:ff:ff:ff promiscuity 1 allmulti 1 minmtu 0 maxmtu 65535
    vlan protocol 802.1Q id 20 <REORDER_HDR>
    bridge_slave state forwarding priority 32 cost 19 hairpin off guard off root_block off fastleave off learning on flood on po
rt_id 0x8001 port_no 0x1 designated_port 32769 designated_cost 0 designated_bridge 8000.d6:91:d5:e1:93:f6 designated_root 8000.d
6:91:d5:e1:93:f6 hold_timer 0.00 message_age_timer 0.00 forward_delay_timer 0.00 topology_change_ack 0 config_pending 0
    proxy_arp off proxy_arp_wifi off mcast_router 1 mcast_fast_leave off mcast_flood on bcst_flood on mcast_to_unicast off neigh_s
uppress off group_fwd_mask 0 group_fwd_mask_str 0x0 vlan_tunnel off isolated off locked off addrngmode eui64 numtxqueues 1 numr
xqueues 1 gso_max_size 64000 gso_max_segs 64 tso_max_size 64000 tso_max_segs 64 gro_max_size 65536
11: eno1.30@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br30 state UP mode DEFAULT group default qlen
1000
    link/ether f8:0d:ac:49:b9:b0 brd ff:ff:ff:ff:ff:ff promiscuity 1 allmulti 1 minmtu 0 maxmtu 65535
    vlan protocol 802.1Q id 30 <REORDER_HDR>
    bridge_slave state forwarding priority 32 cost 19 hairpin off guard off root_block off fastleave off learning on flood on po
rt_id 0x8001 port_no 0x1 designated_port 32769 designated_cost 0 designated_bridge 8000.a6:50:eb:71:91:ec designated_root 8000.a
6:50:eb:71:91:ec hold_timer 0.00 message_age_timer 0.00 forward_delay_timer 0.00 topology_change_ack 0 config_pending 0
    proxy_arp off proxy_arp_wifi off mcast_router 1 mcast_fast_leave off mcast_flood on bcst_flood on mcast_to_unicast off neigh_s

```

Figure 33: Verificamos que hayan quedado bien creadas

```

arley@arley-HP-245-G7-Notebook-PC:~$ bridge link
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding priority 32 cost 100
9: eno1.10@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br10 state forwarding priority 32 cost 19
10: eno1.20@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br20 state forwarding priority 32 cost 19
11: eno1.30@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br30 state forwarding priority 32 cost 19
12: eno1.40@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br40 state forwarding priority 32 cost 19
13: eno1.99@eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br99 state forwarding priority 32 cost 19

```

Figure 34: Verificamos


```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo tcpdump -i eno1.10 -c 10
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eno1.10, link-type EN10MB (Ethernet), snapshot length 262144 bytes
10:14:46.286565 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:48.291151 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:50.299929 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:52.300742 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:54.304882 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:56.310603 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:14:58.314962 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:15:00.320106 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:15:02.325252 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10:15:04.329999 STP 802.1d, Config, Flags [none], bridge-id 800a.4c:71:0c:bd:55:00.8001, length 42
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

Figure 35: Se confirma que se recibe trafico del switch

5.2. Pruebas de conectividad

Se realizaron pruebas de ping entre todos los dispositivos para verificar la conectividad de la red.

Se realiza una configuracion de Vms enlazandolas con su respectivo bridge

- Mostramos el inicio de la prueba de conexion entre las maquinas virtuales. Aca estamos en el administrador de maquinas KVM/QEMU y se selecciona la maquina Debian 13, que sere encendida para comenzar las pruebas de red.

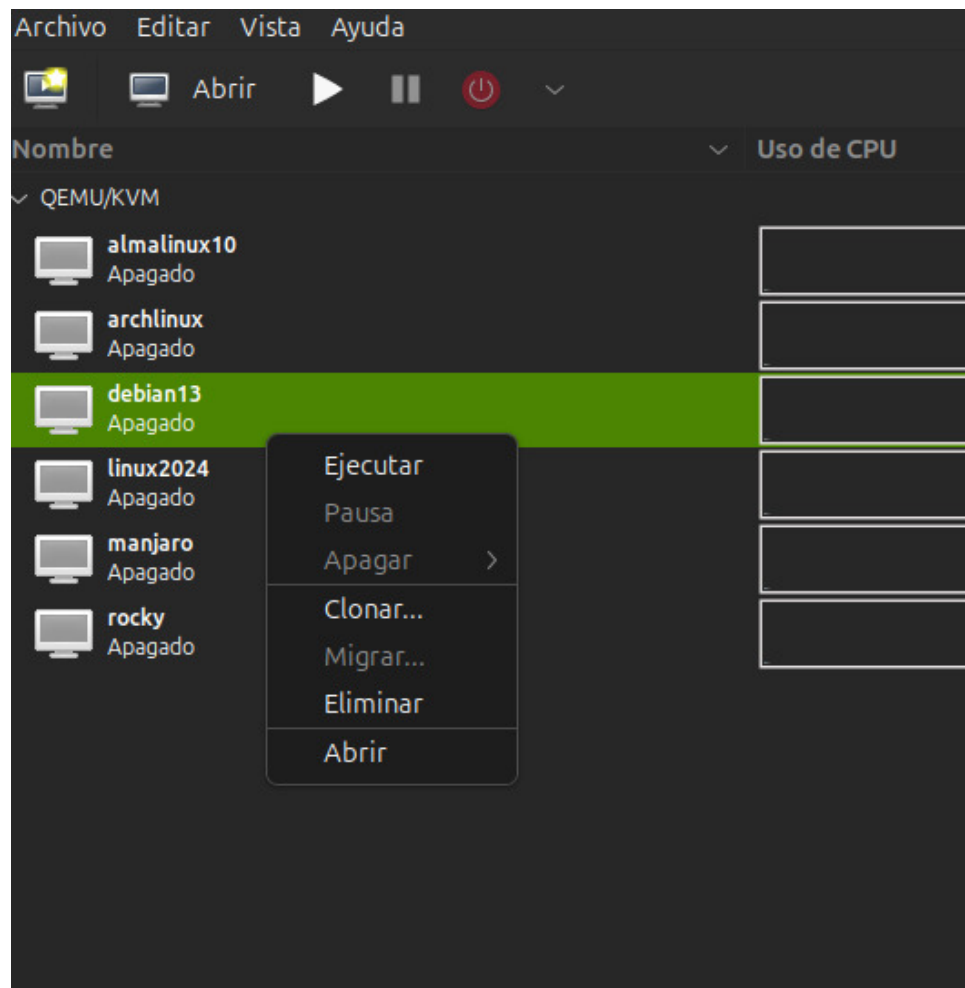


Figure 36: Click derecho abrir

- En esta parte se configura la conexión de red de la máquina virtual. Se selecciona la fuente de red como "Dispositivo de puente" y se elige el bridge previamente creado.

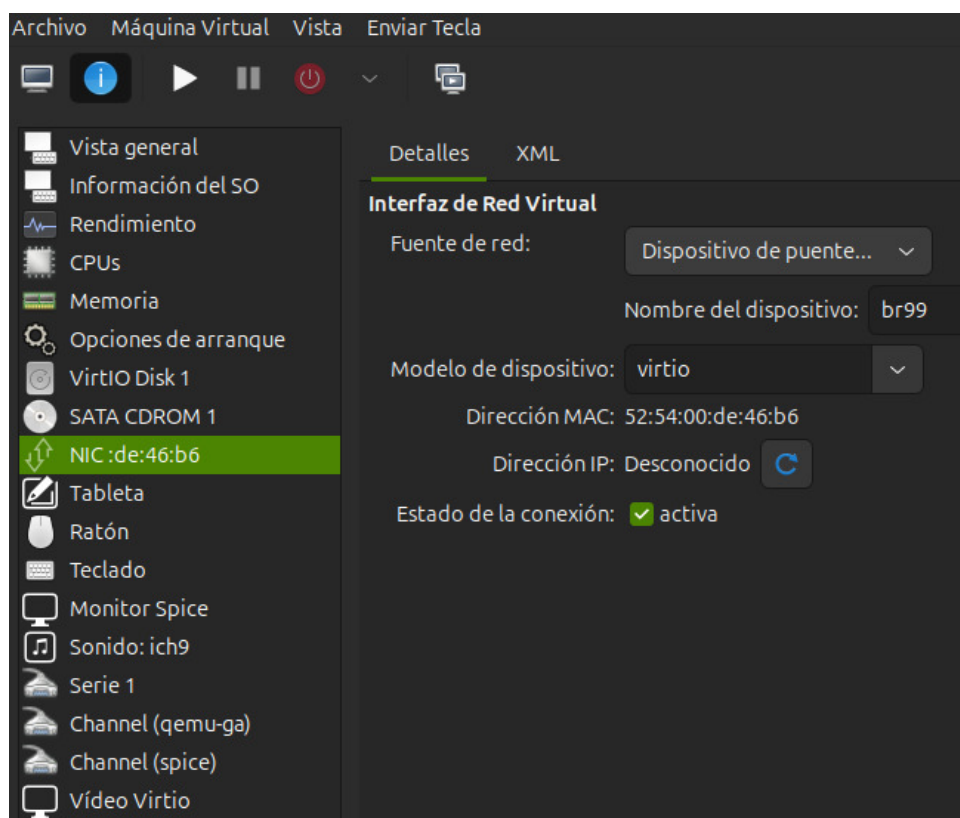


Figure 37: Configuramos fuentes de red

Aplicamos y hacemos el mismo proceso en las otras VM.

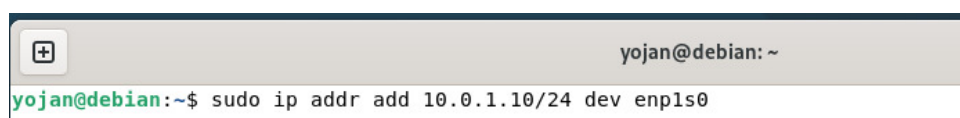


Figure 38: Dentro de Debian le asignamos una ip

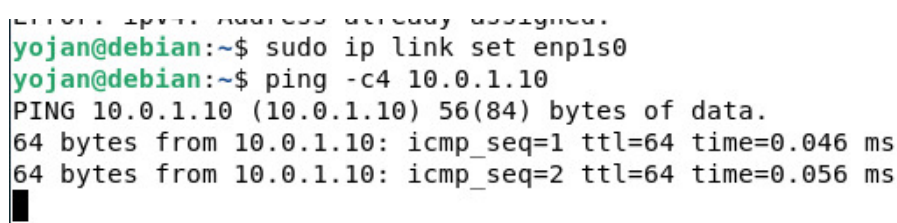


Figure 39: Activamos y enviamos ping para comprobar su conexion

- Configurar ip en Arch Linux

Se edita el archivo de configuracion de red en Arch Linux que esta en dicha ubicacion. Definimos los parametros de la interfaz, como el nombre del adaptador, la direccion ip, la mascara y la puerta de enlace.

```
[root@archlinux ~]# sudo nano /etc/systemd/network/20-wired.network
```

Figure 40: Editamos el archivo

```
GNU nano 8.6 /etc/systemd/network/20-wired.network
[Match]
Name=enp1s0

[Network]
Address=10.0.10.11/24
DNS=8.8.8.8
EOF'
```

Figure 41: Asignamos la ip

```
[root@archlinux ~]# sudo systemctl restart systemd-networkd
```

Figure 42: Aplicamos los cambios

```
[root@archlinux ~]# ip addr show dev enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:49:3f:3b brd ff:ff:ff:ff:ff:ff
    altname enx525400493f3b
    inet 10.0.10.11/24 brd 10.0.10.255 scope global enp1s0
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe49:3f3b/64 scope link proto kernel_ll
        valid_lft forever preferred_lft forever
[root@archlinux ~]#
```

Figure 43: Verificamos que la ip haya quedado bien

- Comprobamos que la ip quede asignada realizando un ping a si misma

```
valid_lft forever preferred_lft forever
inet6 fe80::5054:ff:fe49:3f3b/64 scope link proto kernel_ll
valid_lft forever preferred_lft forever
[root@archlinux ~]# ping 10.0.10.11
PING 10.0.10.11 (10.0.10.11) 56(84) bytes of data:
64 bytes from 10.0.10.11: icmp_seq=1 ttl=64 time=0.083 ms
64 bytes from 10.0.10.11: icmp_seq=2 ttl=64 time=0.041 ms
64 bytes from 10.0.10.11: icmp_seq=3 ttl=64 time=0.054 ms
64 bytes from 10.0.10.11: icmp_seq=4 ttl=64 time=0.108 ms
```

Figure 44: Comprbamos que quedo asignada

- Vamos a asignar direcciones IP a cada bridge del host, cada bridge representa una red diferente por dependencia.

```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip addr add 10.0.1.1/24 dev br99
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip addr add 10.0.10.1/24 dev br10
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip addr add 10.0.20.1/24 dev br20
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip addr add 10.0.30.1/24 dev br30
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip addr ad 10.0.40.1/24 dev br40
arley@arley-HP-245-G7-Notebook-PC:~$ █
```

Figure 45: Crear ips en los bridges del host


```
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set br99 up
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set br10 up
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set br20 up
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set br30 up
arley@arley-HP-245-G7-Notebook-PC:~$ sudo ip link set br40 up
arley@arley-HP-245-G7-Notebook-PC:~$
```

Figure 46: Se activan por si acaso

- Se configura la ruta por defecto (gateway) en la vm Arch Linux

```
[root@archlinux ~]# sudo ip route add default via 10.0.10.1
```

Figure 47: En Arch Linux se configuran las rutas del gateways en cada vm

En debian tambien se realiza la misma configuracion.

```
yojan@debian:~$ sudo ip route add default via 10.0.1.1
```

Figure 48: En Debian hacemos lo mismo

```
[root@archlinux ~]# ping -c 3 10.0.10.1
PING 10.0.10.1 (10.0.10.1) 56(84) bytes of data.
64 bytes from 10.0.10.1: icmp_seq=1 ttl=64 time=0.460 ms
64 bytes from 10.0.10.1: icmp_seq=2 ttl=64 time=0.204 ms
64 bytes from 10.0.10.1: icmp_seq=3 ttl=64 time=0.469 ms

--- 10.0.10.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2056ms
rtt min/avg/max/mdev = 0.204/0.377/0.469/0.122 ms
[root@archlinux ~]#
```

Figure 49: Desde arch ping al gateway de vlan

```
[root@archlinux ~]# ping -c 3 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=1 ttl=63 time=0.977 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=63 time=0.788 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=63 time=0.598 ms

--- 10.0.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.598/0.787/0.977/0.154 ms
[root@archlinux ~]#
```

Figure 50: Desde Arch ping a Debian admin

Ahora haremos la misma prueba de ping desde Debian

```

yojan@debian:~$ ping -c 3 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.233 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.290 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.514 ms

--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2030ms
rtt min/avg/max/mdev = 0.233/0.345/0.514/0.121 ms
yojan@debian:~$

```

Figure 51: Dede Debian ping al gateway de vlan

```

yojan@debian:~$ ping -c 3 10.0.10.11
PING 10.0.10.11 (10.0.10.11) 56(84) bytes of data.
64 bytes from 10.0.10.11: icmp_seq=1 ttl=63 time=0.856 ms
64 bytes from 10.0.10.11: icmp_seq=2 ttl=63 time=0.767 ms
64 bytes from 10.0.10.11: icmp_seq=3 ttl=63 time=0.385 ms

--- 10.0.10.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2018ms
rtt min/avg/max/mdev = 0.385/0.669/0.856/0.204 ms
yojan@debian:~$ █

```

Figure 52: Desde Debian ping a Arch

- Ahora asignaremos la ip en Rocky Linux

```

yojan@localhost:~$ sudo nmcli con add type ethernet ifname ens3 con-name static-tec ipv4.addresses 10.0.20.11/24 ipv4.gateway 10.0.20.1 ipv4.dns 8.8.8.8 ipv4.method manual autoconnect yes
Conexión «static-tec» (34770833-0c50-461b-8338-c291f03e04fc) añadida con éxito.

```

Figure 53: Asignamos ip en Rocky

Tambien realizamos comprobacion mediante un pign realizado a si mismo

```

yojan@localhost:~$ sudo nmcli con up static-tec
Conexión activada con éxito (ruta activa D-Bus: /org/freedesktop/NetworkManager/ActiveConnection/11)
yojan@localhost:~$ ip addr show dev ens3
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:1f:96:d1 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altnamex enx5254001f96d1
    inet 10.0.20.11/24 brd 10.0.20.255 scope global noprefixroute ens3
        valid_lft forever preferred_lft forever
    inet6 fe80::e308:511b:15ad:d16f/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

Figure 54: Activamos la ip y comprobamos

Realizamos tambien ping al gateway del host y a Debian

```

yojan@localhost:~$ ping -c 3 10.0.20.1
PING 10.0.20.1 (10.0.20.1) 56(84) bytes of data.
64 bytes from 10.0.20.1: icmp_seq=1 ttl=64 time=1.57 ms
64 bytes from 10.0.20.1: icmp_seq=2 ttl=64 time=29.0 ms
64 bytes from 10.0.20.1: icmp_seq=3 ttl=64 time=0.486 ms

```

Figure 55: Ping al gateway del host

```
yojan@localhost:~$ ping -c 3 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=1 ttl=63 time=0.737 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=63 time=0.569 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=63 time=0.661 ms

--- 10.0.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 0.569/0.655/0.737/0.068 ms
```

Figure 56: Ping al admin Debian

5.3. Configuración de servicios

Se instalaron y configuraron los servicios necesarios para el funcionamiento óptimo de cada dependencia.

6. Trabajo pendiente y pasos detallados para completar la implementación

Durante la ejecución práctica del proyecto se instalaron y configuraron las piezas fundamentales (switch, bridges, Debian administrador, Arch y Rocky). Sin embargo hubo varios elementos que no se llegaron a completar por tiempo y limitaciones de hardware. En esta sección se documenta en detalle el **paso a paso** que se debe seguir para terminar el proyecto y las pruebas solicitadas en el enunciado. Cada bloque incluye comandos concretos, comprobaciones y la evidencia que debe guardarse en Overleaf.

6.1. Resumen de elementos pendientes

- Completar creación y conexión de **7 contenedores** indicados en el proyecto (Fedora, Garuda, CentOS, Ubuntu, Alpine, y los contenedores padre/hijos adicionales).
- Instalar agentes de monitoreo (**node_exporter**) en todas las máquinas/contenedores faltantes y registrar los targets en Prometheus.
- Completar la configuración de Netdata, captura Wireshark, smartctl y herramientas de análisis de logs en la dependencia financiera.
- Desplegar la aplicación **Streamlit** en la VM de Ventas (o en contenedor) y auditarla con **lynis** y **nmap**.
- Automatizar reglas de NAT/forwarding y persistencia (iptables/nftables) para permitir acceso a repositorios y pruebas exteriores.

- Crear y completar dashboards de Grafana con métricas de CPU, memoria, I/O, procesos y contenedores.

6.2. 1. Contenedores faltantes — creación, red y verificación

A continuación se describen los contenedores que faltaron y los pasos exactos para crearlos y conectarlos a las VLAN correctas (suponemos que los bridges en el host ya existen: br10, br20, br30, br40, br99).

6.2.1 Red Docker con macvlan (recomendado para VLANs reales)

ejemplo: crear red macvlan para VLAN 30 (Finanzas)

```
sudo docker network create -d macvlan \
  --subnet=10.0.30.0/24 \
  --gateway=10.0.30.1 \
  -o parent=br30 \
  garuda-net
```

Nota: use parent=brXX con el bridge concreto. Si Docker emite errores, borrar y recrear la red y reintentar.

6.2.2 Crear cada contenedor (comandos ejemplares)

1) CentOS (RRHH)

```
sudo docker run -dit --name centos_rh --network some-net --ip 10.0.10.12 centos:8 bash
```

2) Ubuntu (Finanzas)

```
sudo docker run -dit --name ubuntu_fin --network garuda-net --ip 10.0.30.12 ubuntu:22
```

3) Alpine (Finanzas)

```
sudo docker run -dit --name alpine_fin --network garuda-net --ip 10.0.30.13 alpine:la
```

4) Fedora (Ventas) -> si falla con mirrors use 'debian' como alternativa

```
sudo docker run -dit --name fedora_sales --network sales-net --ip 10.0.40.11 fedora:l
```

6.2.3 Verificaciones dentro del contenedor

Entrar y probar (ejemplo Ubuntu):

```
sudo docker exec -it ubuntu_fin bash
apt update
```

```
apt install -y iputils-ping iproute2 curl
ip addr
ping -c 3 10.0.30.1    # gateway host
ping -c 3 10.0.1.10    # admin Debian
```

Si ping falla:

1. verificar que la red Docker apunta a la `parent` correcta:

```
sudo docker network inspect garuda-net | grep Parent
```

2. si se usan `macvlan`, crear una interfaz `macvlan` en el host para hablar con los contenedores:

```
sudo ip link add macvlan-g30 link br30 type macvlan mode bridge
sudo ip addr add 10.0.30.1/24 dev macvlan-g30
sudo ip link set macvlan-g30 up
```

6.3. 2. Instalación de Node Exporter en todas las VM/Contenedores

En cada nodo (VMs Arch, Rocky y contenedores que soporten `systemd` o ejecución directa) se instalará `node_exporter`.

VER=1.6.1

```
curl -LO https://github.com/prometheus/node_exporter/releases/download/v${VER}/node_e-
tar xvf node_exporter-*.tar.gz
sudo mv node_exporter-*/node_exporter /usr/local/bin/
sudo useradd -rs /bin/false node_exporter || true
```

Crear servicio `systemd` (si está disponible)

```
sudo tee /etc/systemd/system/node_exporter.service <<'EOF'
```

```
[Unit]
```

```
Description=Node Exporter
```

```
After=network.target
```

```
[Service]
```

```
User=node_exporter
```

```
ExecStart=/usr/local/bin/node_exporter
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
EOF
```

```
sudo systemctl daemon-reload
sudo systemctl enable --now node_exporter
```

Si el contenedor no tiene systemd: ejecutar el binario en background:

```
/usr/local/bin/node_exporter &> /var/log/node_exporter.log &
```

6.4. 3. Configuración de Prometheus (Debian administrador)

Editar `prometheus.yml` en `/monitoring` y añadir todos los targets:

```
scrape_configs:
  - job_name: 'node_exporters'
    static_configs:
      - targets:
        - '10.0.1.10:9100'    # admin
        - '10.0.10.11:9100'   # arch
        - '10.0.10.12:9100'   # centos (rh)
        - '10.0.20.11:9100'   # rocky
        - '10.0.30.11:9100'   # garuda/ubuntu_fin
        - '10.0.30.12:9100'   # alpine_fin (si aplica)
        - '10.0.40.11:9100'   # fedora_sales (si aplica)
```

Luego reiniciar Prometheus (docker-compose):

```
cd ~/monitoring
docker compose restart prometheus
```

6.5. 4. Grafana — importar dashboards y visualizaciones

1. Ingresar a Grafana: `http://10.0.1.10:3000` (admin:admin).
2. Data Source: Prometheus → URL: `http://prometheus:9090` o `http://10.0.1.10:9090`.
3. Importar dashboard: *Node Exporter Full* (ID 1860) y un dashboard para cAdvisor/containers.
4. Capturar pantallazos de CPU, memoria, I/O y contenedores para anexar al informe.

6.6. 5. Dependencia Financiera: Netdata, wireshark, smartctl y logs

Para la dependencia financiera (contenedores Ubuntu / Alpine) se solicita análisis de archivos, logs y visualización con Netdata.

```
# Netdata (en Debian admin o en nodo financiero)
sudo docker run -d --name=netdata \
  -p 19999:19999 \
  --cap-add SYS_PTRACE --cap-add SYS_ADMIN \
  netdata/netdata

# Wireshark: captura desde host (o usar tcpdump)
sudo tcpdump -i br30 -w ~/proyecto_red/outputs/finanzas_capture.pcap
# o usar tshark dentro del host/VM
\textbf{smartctl:} (ejecutar en host o VM que tenga acceso a disco físico)
\begin{verbatim}
sudo apt install smartmontools
sudo smartctl -a /dev/sdX > ~/proyecto_red/outputs/smartctl_sdX.txt
```

6.7. Ejemplos

```
ncdu /var/log lnav /var/log/syslog goaccess access.log -o report.html
```

Guardad capturas de Netdata, salida de tcpdump, smartctl y reportes de goaccess.

6.8. 6. Dependencia Comercial y Ventas: Streamlit y auditoría

1. Crear app streamlit (archivo app.py):

```
import streamlit as st
st.title("Empresa Tech A")
st.write("Página informativa | servicios, contacto, métricas")
```

2. Dockerizar app (Dockerfile):

```
FROM python:3.11-slim
WORKDIR /app
COPY app.py requirements.txt ./
RUN pip install -r requirements.txt
```

```
EXPOSE 8501
```

```
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

3. Construir y publicar en Docker Hub (opcional).

4. Instalar `lynis` y ejecutar auditoría:

```
sudo apt install -y lynis
sudo lynis audit system > ~/proyecto_red/outputs/lynis_audit.txt
```

5. Escanear con `nmap` desde admin:

```
nmap -sT -p- 10.0.40.11 -oN ~/proyecto_red/outputs/nmap_fedora.txt
```

6.9. 7. Persistencia de configuración y automatización

Para que todo persista tras reinicios:

- Guardar reglas iptables:

```
sudo apt install -y iptables-persistent
sudo netfilter-persistent save
```

- Crear scripts de arranque en `/usr/local/bin` o systemd unit files que creen las docker networks y bridges si faltan:

```
#!/bin/bash
# /usr/local/bin/setup_networks.sh
ip link add name br30 type bridge || true
ip link set br30 up
docker network create -d macvlan --subnet=10.0.30.0/24 --gateway=10.0.30.1 -o pa
```

- Crear un systemd service que ejecute el script al boot.

7. Resultados

El desarrollo del proyecto permitió la creación de una infraestructura virtualizada que integra conceptos de redes, sistemas operativos y tecnologías de contenedores, cumpliendo parcialmente los requerimientos establecidos para la empresa tecnológica A. La arquitectura implementada se basó en un entorno híbrido compuesto por máquinas virtuales y contenedores Linux interconectados mediante bridges y VLANs configuradas sobre un switch Cisco 2960, logrando segmentación lógica de las distintas dependencias.

7.1. Resultados obtenidos

- Se implementaron correctamente las máquinas virtuales **Debian** (Administrador), **Arch Linux** (Recursos Humanos) y **Rocky Linux** (Tecnología), configuradas con direcciones IP estáticas y comunicación bidireccional verificada mediante pruebas de conectividad **ping**.
- Se establecieron bridges de red (**br10**, **br20**, **br30**, **br40**, **br99**) asociados a VLANs independientes, garantizando la separación de tráfico y la correcta correspondencia con los enlaces físicos y lógicos del switch.
- Se logró la comunicación entre las VLAN 10, 20 y 99, asegurando la interoperabilidad entre las dependencias a través del host configurado como router.
- Se instaló y configuró con éxito el entorno de monitoreo en la máquina **Debian**, implementando los servicios **Prometheus** y **Grafana** para la recolección y visualización de métricas del sistema.
- Se implementó el agente **Node Exporter** en las máquinas Arch y Rocky, logrando la supervisión remota de recursos como CPU, memoria, almacenamiento y red.
- Se verificó la detección de los nodos en el panel de **Prometheus** y la representación gráfica en los dashboards de **Grafana**, confirmando la correcta integración de los sistemas de monitoreo.
- Se configuró la topología de red sobre el switch Cisco 2960, con puertos asignados a VLANs específicas y enlaces troncales para la transmisión simultánea de múltiples redes virtuales.

7.2. Resultados pendientes

Aunque se obtuvo una infraestructura base completamente operativa, algunas fases quedaron pendientes de ejecución por limitaciones de hardware y tiempo. Entre ellas:

- Configuración definitiva de los contenedores **Fedora**, **Garuda**, **CentOS**, **Ubuntu** y **Alpine**, los cuales debían representar las dependencias Comercial, Financiera y nodos de respaldo.
- Despliegue de servicios adicionales como **Netdata**, **cAdvisor**, **Wireshark**, **smartctl** y herramientas de auditoría (**Lynis**, **nmap**) para la capa de seguridad y diagnóstico.
- Implementación de la aplicación **Streamlit** en el contenedor de Ventas, junto con la generación de imágenes personalizadas y su publicación en **Docker Hub**.

- Ejecución de auditorías de desempeño y seguridad, generación de reportes automáticos en **Grafana** y análisis de métricas históricas.

A pesar de lo anterior, los resultados alcanzados consolidaron una infraestructura funcional, reproducible y técnicamente documentada, capaz de escalar y adaptarse a la incorporación de los elementos faltantes sin modificar la base ya implementada.

7.3. Impacto técnico

El proyecto demostró la aplicabilidad de las metodologías DevOps y NetOps dentro de un entorno educativo, integrando prácticas de virtualización, automatización, despliegue de servicios y monitoreo en una topología multi-nodo. Se evidenció la capacidad de los estudiantes para configurar redes VLAN reales, interconectarlas mediante bridges y establecer un entorno de observabilidad completo mediante software libre. El diseño adoptado reproduce fielmente las condiciones de una infraestructura corporativa moderna, distribuida y segmentada por áreas funcionales.

8. Conclusiones

El desarrollo del proyecto de infraestructura de red y virtualización permitió comprender, integrar y aplicar conceptos avanzados de administración de sistemas, redes y automatización de entornos híbridos. A partir de la implementación, se destacan las siguientes conclusiones principales:

1. La combinación de máquinas virtuales y contenedores proporciona una arquitectura flexible y eficiente, capaz de aislar servicios y garantizar la escalabilidad horizontal dentro de una red corporativa.
2. La configuración de bridges, VLANs y enlaces troncales en un switch Cisco permitió segmentar el tráfico, mejorar la seguridad y facilitar el control de cada dependencia, replicando la estructura de una red empresarial real.
3. El uso de herramientas como **Prometheus** y **Grafana** demostró la importancia de los sistemas de monitoreo centralizado para la gestión proactiva de recursos y la detección temprana de fallas en entornos distribuidos.
4. A pesar de las limitaciones de hardware y tiempo, se logró una infraestructura estable y funcional, con comunicación efectiva entre nodos y servicios básicos de observabilidad implementados correctamente.

5. Las prácticas realizadas fortalecen competencias en administración Linux, virtualización (KVM/QEMU), gestión de contenedores (Docker), configuración de redes (VLAN, bridges) y monitoreo de sistemas, consolidando una visión integral del perfil profesional DevOps–NetOps.
6. El trabajo colaborativo y la distribución de responsabilidades entre los integrantes del equipo permitió simular un entorno real de trabajo en TI, donde la cooperación y la documentación precisa son esenciales para la continuidad operativa del proyecto.
7. Finalmente, el proyecto evidencia que, incluso con recursos limitados, es posible diseñar, desplegar y documentar infraestructuras complejas y escalables, reafirmando la importancia del conocimiento técnico, la planificación estructurada y la automatización en los procesos de ingeniería de redes y sistemas.

Conclusión general: El proyecto integró de manera exitosa los principios de virtualización, segmentación de redes y observabilidad de sistemas, sentando las bases para una infraestructura DevOps completamente funcional. El trabajo desarrollado constituye un prototipo sólido que puede ser ampliado con nuevas dependencias, políticas de seguridad y herramientas de gestión, demostrando la capacidad de los autores para diseñar y administrar entornos tecnológicos complejos bajo estándares profesionales de ingeniería.