

Integración de DevOps

Laura Cruz, Andres Saldaña, Jonny Vargas

24 de octubre de 2025

1. Introducción

Mediante este proyecto se analizarán las competencias prácticas en el uso, configuración e integración de sistemas operativos dentro de entornos virtualizados y contenerizados, aplicando conceptos de DevOps y NetOps. A través del desarrollo de una infraestructura tecnológica simulada, se busca reflejar el desempeño de un ingeniero DevOps con capacidades de administración de redes, encargado de implementar, monitorear y documentar la comunicación entre diferentes dependencias de una empresa tecnológica.

Cada dependencia contará con máquinas virtuales y contenedores específicos, interconectados mediante subredes independientes y supervisados desde un servidor central administrado por Debian. Además, se emplearán herramientas de análisis, monitoreo y gestión.

Se busca integrar los principios de la automatización, el despliegue eficiente y la supervisión continua de servicios, comprendiendo la arquitectura de redes modernas y la administración avanzada de sistemas Linux, pilares fundamentales para la ingeniería DevOps actual.

<https://github.com/Patito83/Integracion-de-DevOps>

2. Gestión de la Empresa

Para la gestión de las dependencias solicitadas por la empresa tecnológica A, se conformo una máquina virtual central con Debian, que cumple la función de administrador y monitor general de todas las dependencias. Luego cada dependencia fue desplegada de acuerdo con lo requerido por la empresa

- Recursos Humanos: Máquina virtual con Arch Linux.
- Tecnología: Máquina virtual con Rocky Linux.
- Financiera: Contenedor con Garuda Linux.
- Comercial y Ventas: Contenedor con Fedora Linux.

Para realizar bien la practica se instalo cada maquina virtual, incluyendo la configuración de red, asignación de IP estática, instalación de dependencias y pruebas de conectividad. Se validó que cada dependencia únicamente pudiera comunicarse con sus propios equipos, mientras que el administrador Debian podía acceder a todas las redes mediante un puente configurado en QEMU.

2.1. Desarrollo de Sistema de gestión Debian

Para la instalación de Debian mediante una Máquina Virtual (QEMU), se siguio los siguientes pasos. 1. Primero debemos mediante la consola instalar

el motor que crea las maquinas virtuales.

```
apt update
apt install -y qemu-kvm libvirt-daemon-system libvirt-
  clients virt-manager bridge-utils # Esto instala el
  Motor de KVM / QEMU - el motor que crea las maquinas
  virtuales.

libvirt - el sistema que las gestiona (crea, apaga,
  configura).

virt-manager - programa grafico (si tu Debian tiene
  entorno grafico).

bridge-utils - para configurar redes virtuales.
```

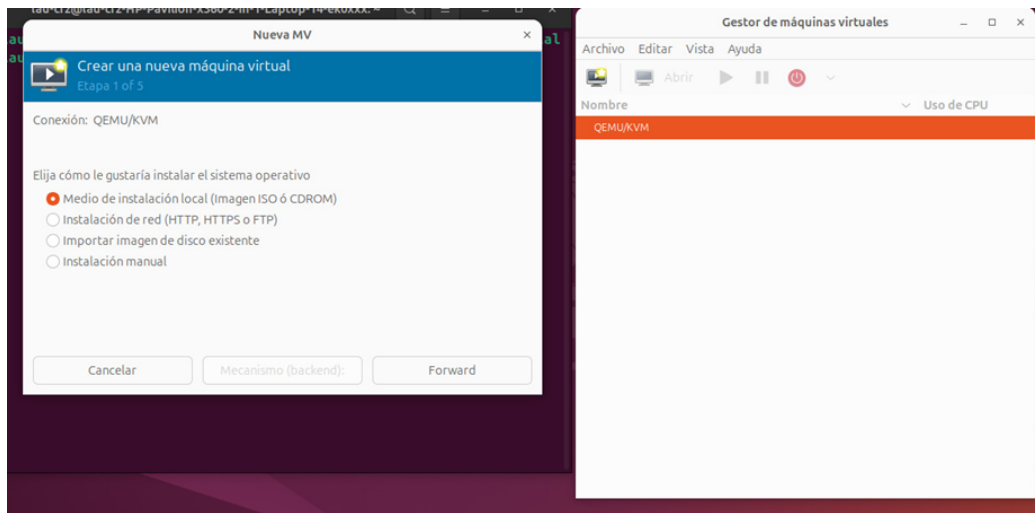


Figura 1: Creación Máquina Virtual

2. Luego de Descargar QEMU, debemos instalar el ISO o la imagen de Debian. Esto para poder crear la Máquina virtual mediante el Motor de QEMU

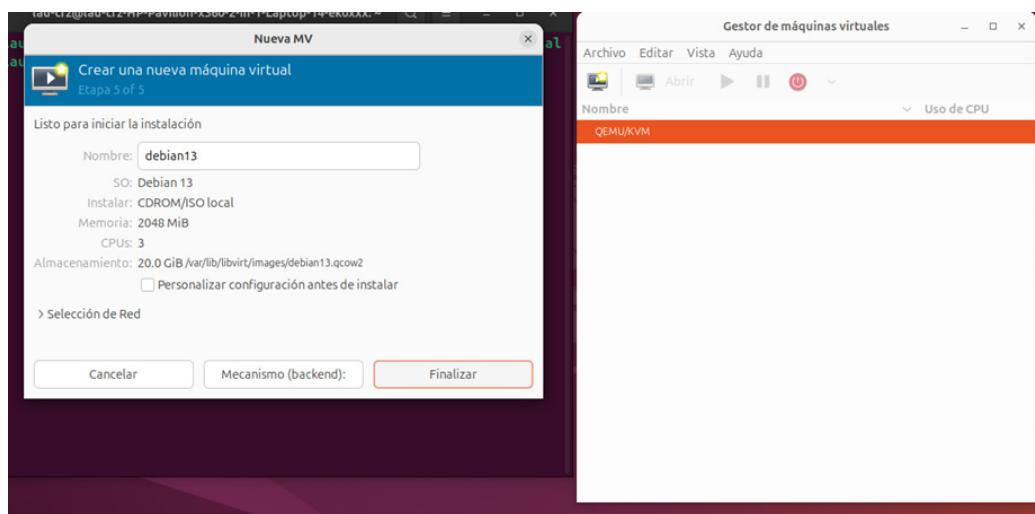


Figura 3: Creacion Maquina 2

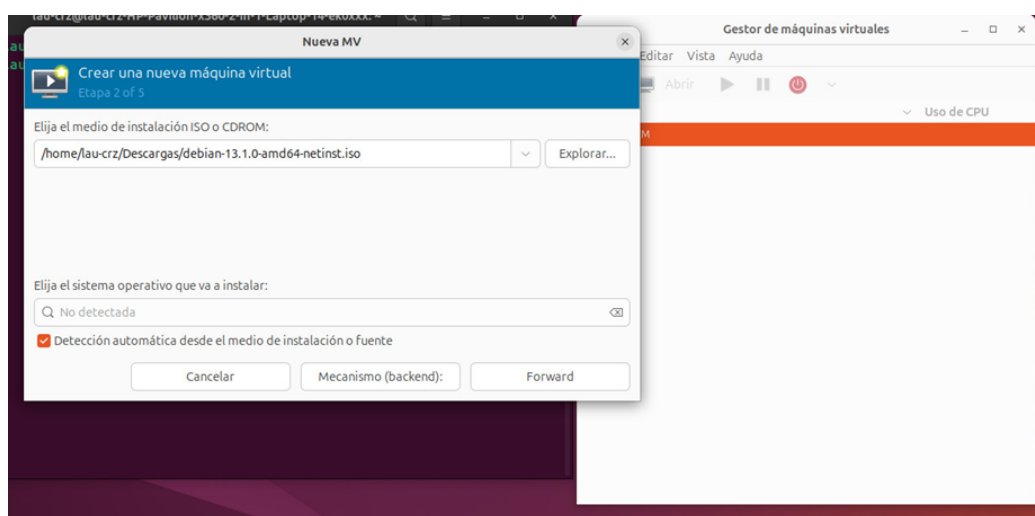


Figura 2: Debian ISO

3. Ya con Debian instalado, se gestiono la configuración de las redes. Esto para que la Maquina pudiera iniciar correctamente y para luego gestionarla como la red principal.

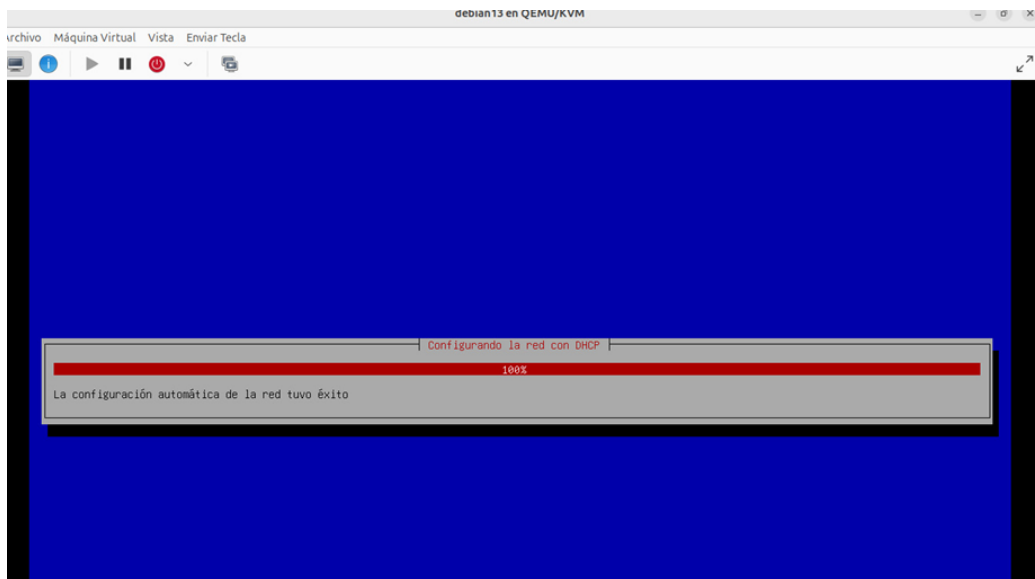


Figura 4: Gestión de Red Debian

2.2. Gestión de Red Principal Debian

Para que Debian sea el sistema principal de toda la red se siguieron los siguientes pasos. Estos pasos los gestionamos fuera de la Máquina virtual, se deben realizar en el terminal principal.

1. Lo primero es convertirse en el administrador del Host y verificar el libvirtd.

```
sudo -i
systemctl status libvirtd

# si esta inactive => arrancarlo:
sudo systemctl start libvirtd
sudo systemctl enable libvirtd
```

2. Luego para arrancar la Máquina debemos ver el estado y los nombre de está.

```
virsh list --all # muestra nombres y estado

Arrancar / detener la VM Debian

sudo virsh start <debian-name>
sudo virsh shutdown <debian-name>
sudo virsh destroy <debian-name>
```

```
Conectarte a la consola serial de la VM

sudo virsh console <debian-name> # para salir de la
    consola: Ctrl+]

virt-manager          # abrir GUI y seleccionar la VM
virt-viewer <debian-name>
```

```
Hacer que la VM arranque automaticamente al arrancar el
    host

sudo virsh autostart <debian-name>
# para desactivar:
sudo virsh autostart --disable <debian-name>

Ver logs si algo falla

# logs de libvirtd
sudo journalctl -u libvirtd -e

# logs de la maquina virtual (domxml)
sudo virsh dominfo <debian-name>
sudo virsh domxml-to-native qemu-argv <debian-name> #
    informacion avanzada
```

2.3. Acceso de Administrador dentro de la Máquina Virtual

Es importante darle el acceso a Debian para que este pueda ser el host principal de la red.

1. Desde la cuenta con permisos de root debemos crear un nuevo usuario y habilitar el SSH.

```
sudo -l

usermod -aG sudo tu_usuario

adduser nuevo_usuario
usermod -aG sudo nuevo_usuario
```

```
sudo passwd root
# ahora puedes usar su - para cambiar a root con la
  contraseña nueva

Instalar y habilitar SSH (para acceder remotamente)

sudo apt update
sudo apt install -y openssh-server
sudo systemctl enable --now ssh # comprobar estado
sudo systemctl status ssh
```

2. Luego para verificar la correcta instalación del SSH desde otro equipo configuramos la seguridad de este.

```
ssh tu_usuario@IP_debian_vm

En tu equipo local:
ssh-keygen -t ed25519
ssh-copy-id tu_usuario@IP_debian_vm # ahora ssh
  tu_usuario@IP_debian_vm te dejara entrar sin
  contraseña

Seguridad basica dentro de la VM

sudo apt update && sudo apt upgrade -y
editar /etc/ssh/sshd_config PermitRootLogin no y sudo
  systemctl restart ssh

Habilita firewall:

sudo apt install -y ufw
sudo ufw allow OpenSSH
sudo ufw enable
```

2.4. Administración para las otras Maquinas

Para crear y instalar las demás Maquinas en un mismo host debemos tener sus ISO's en el host principal.

```
echo 'net.ipv4.ip_forward=1' | sudo tee -a /etc/sysctl.
  conf
```

```
sudo sysctl -p

VERIFICACION FINAL:

echo "=== INTERFACES ==="
ip addr show | grep "inet " | grep -v "127.0.0.1"

echo "=== RUTAS ==="
ip route show

echo "=== FORWARDING ==="
cat /proc/sys/net/ipv4/ip_forward
```

Si todo esta correcto cuando veamos las redes conectadas a Debian deben estar todas las redes de los demás sistemas

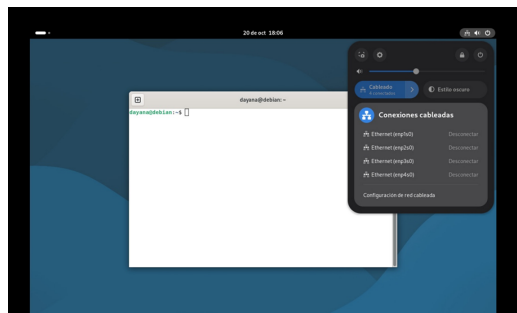


Figura 5: Visualización de Redes

2.5. Creación de las Máquinas para las dependencias faltantes

Para instalar las otras dependencias se realizó los mismos pasos para Debian, con diferencia de la red de host. De igual manera los ISO's de las máquinas restantes deben estar en el PC principal.

1. ArchLinux - RRHH

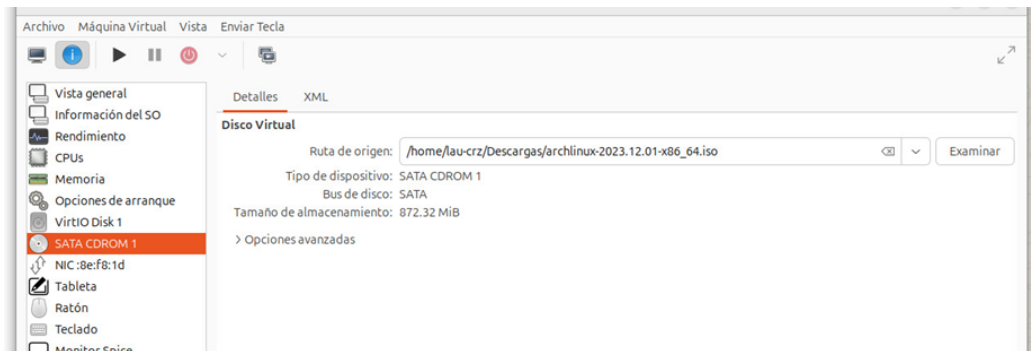


Figura 6: Disco Virtual Arch



Figura 7: Arranque Arch

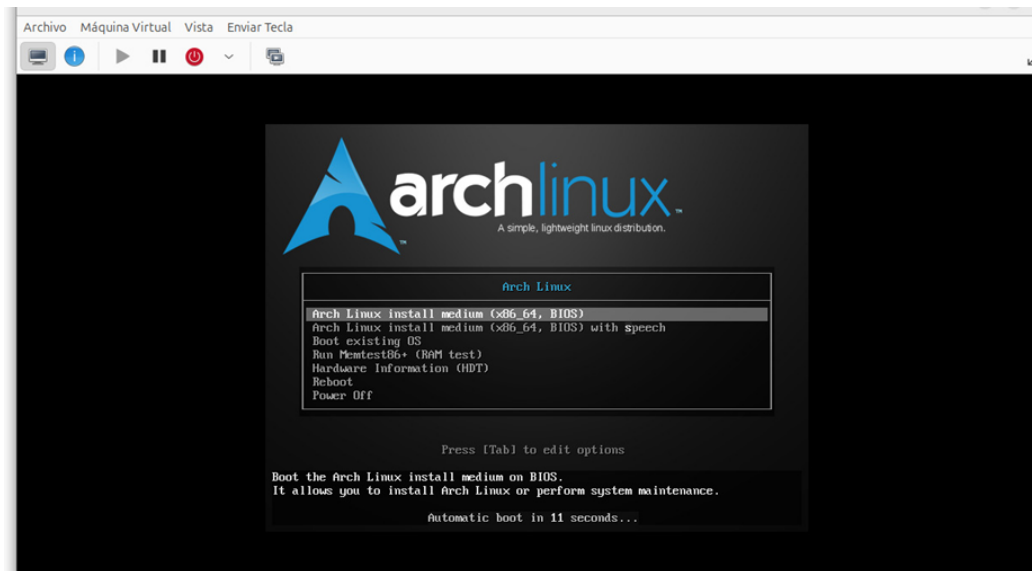


Figura 8: Instalador de Arch

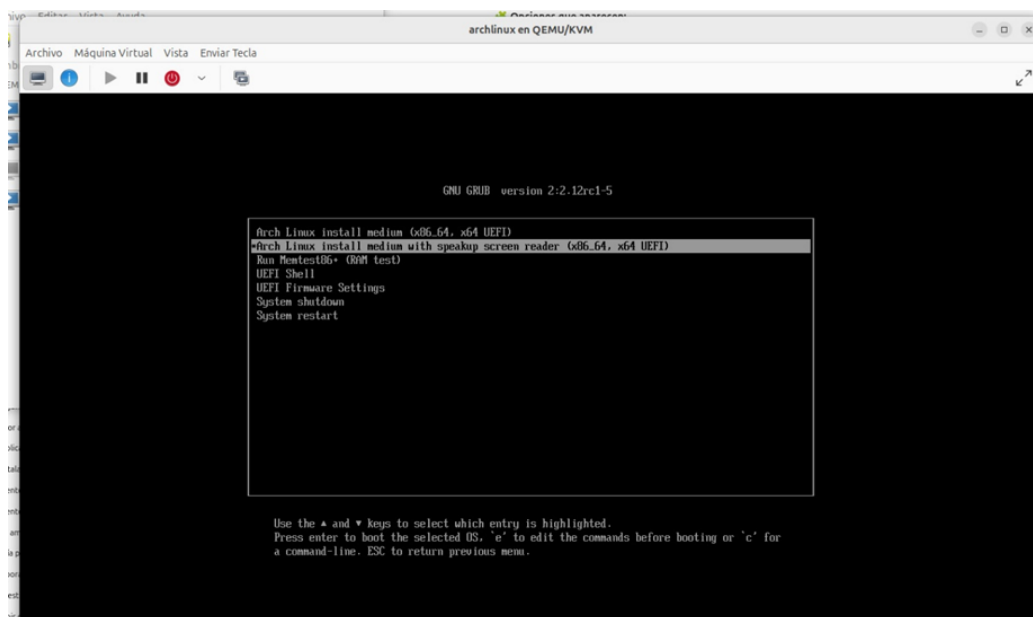


Figura 9: Arch

2. Rocky Linux

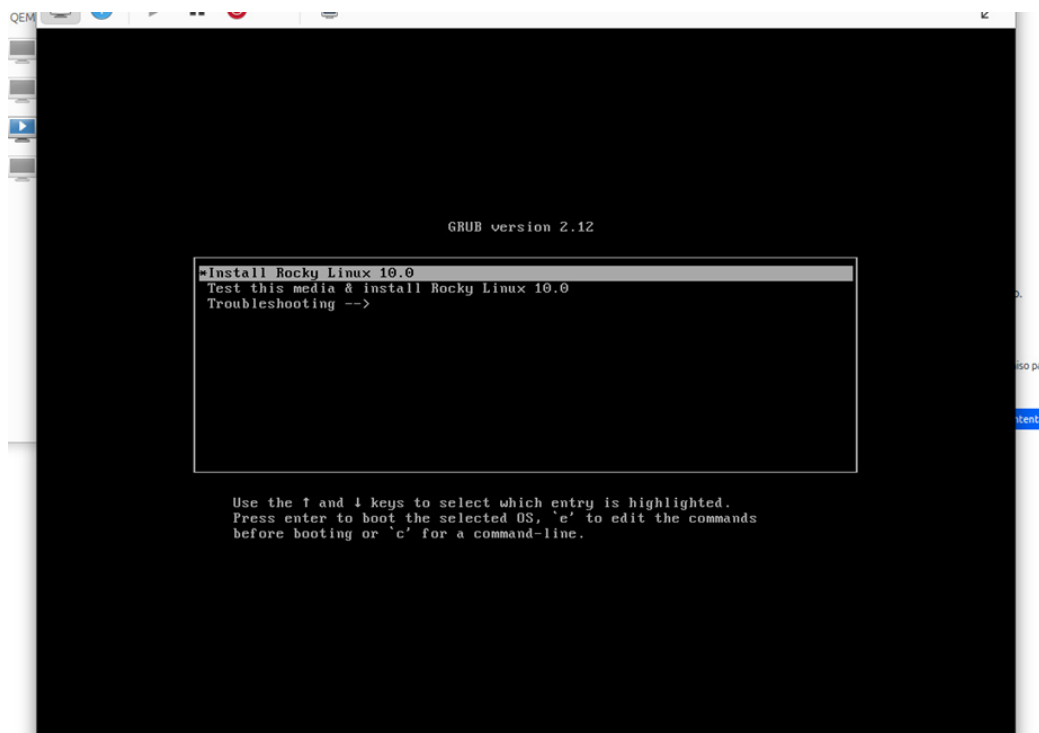


Figura 10: Rocky Instalador

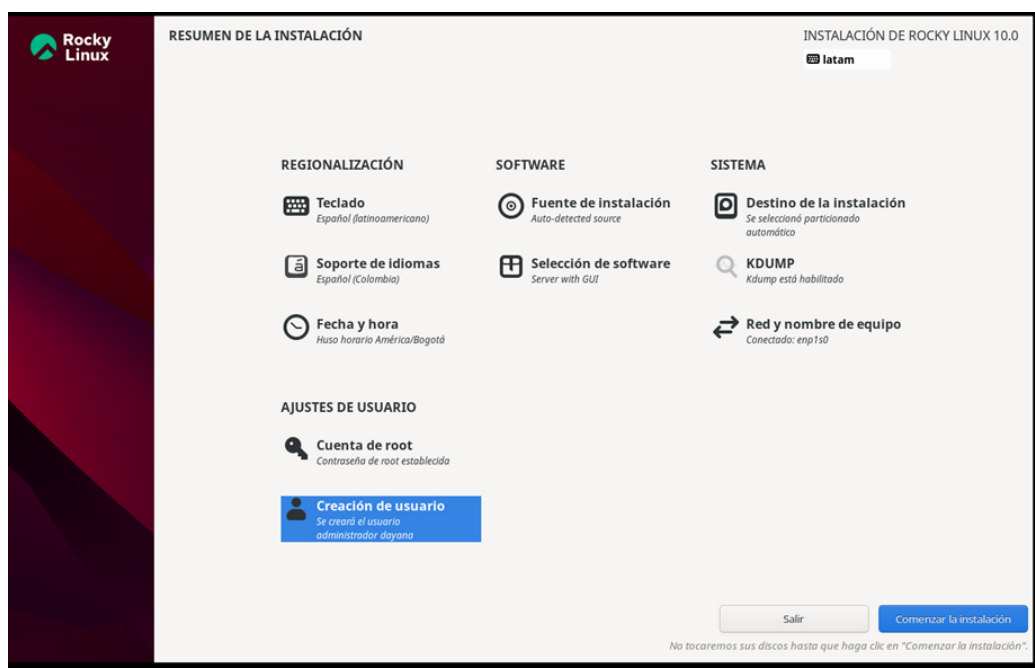


Figura 11: Rocky Instalado

3. Verificación de conexión de Red - Debian A Rocky

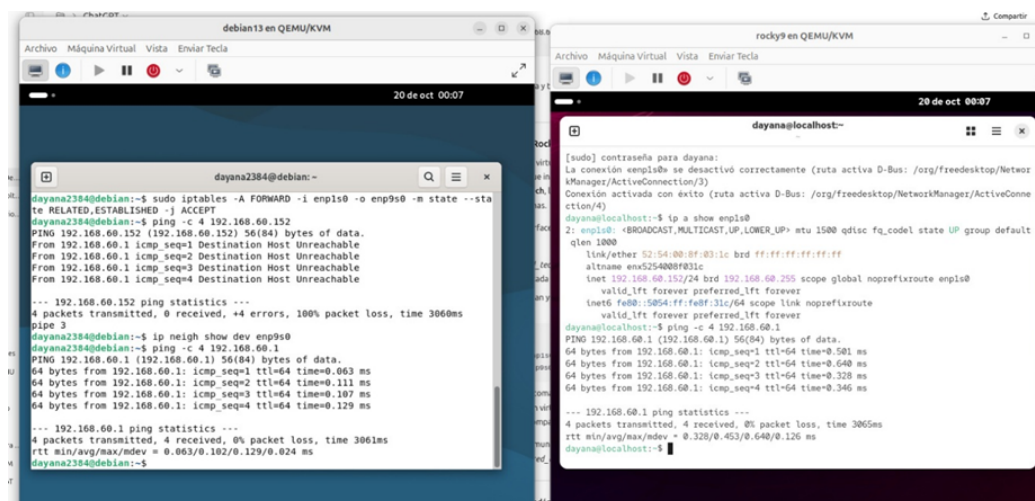


Figura 12: Conexion DYR

3. Gestión de la Empresa con Contenedores

3.1. Creación de contenedor Debian

Para crear el contenedor debemos crear el archivo de docker

```
FROM archlinux:latest
LABEL maintainer="dayana2384@debian"

RUN pacman -Sy --noconfirm archlinux-keyring && \
    pacman -Syu --noconfirm && \
    pacman -S --noconfirm base-devel vim git sudo fish
    fastfetch

RUN useradd -m garuda && echo "garuda:garuda" | chpasswd
    && usermod -aG wheel garuda
RUN chsh -s /usr/bin/fish garuda

CMD ["fastfetch"]
```

Luego construimos la imagen y corremos el contenedor

```
contruir la imagen
sudo docker build -t garuda-financiera .

correr contenedor
sudo docker run -it --name contenedor-financiero garuda-
financier

Esto te abrir una consola interactiva dentro del
    contenedor con el shell fish que configuraste.
Si todo va bien deber as ver un prompt como:

[garuda@<container_id> ~]$

Puedes probar que el entorno funcione con:

fastfetch
```

```

dayana2384@debian:~/garuda-container$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
garuda-financiera    latest         c67f87b4453a   4 minutes ago  1.17GB
dayana2384@debian:~/garuda-container$ sudo docker run -it --name contenedor-financiero garuda-financiera
root@c8479645baf7
-----
OS: Arch Linux x86_64
Host: KVM/QEMU Standard PC (Q35 + ICH9)
Kernel: Linux 6.12.48+deb13-amd64
Uptime: 34 mins
Packages: 158 (pacman)
Shell: bash 5.3.3
Display (QEMU Monitor): 1280x800 in 15z
Terminal: xterm
Terminal Font: fixed (8.0pt)
CPU: 3 x 12th Gen Intel(R) Core(TM) i5z
GPU: RedHat Virtio 1.0 GPU
Memory: 840.43 MiB / 1.93 GiB (43%)
Swap: 257.84 MiB / 1.08 GiB (23%)
Disk (/): 8.05 GiB / 18.46 GiB (44%) -y
Local IP (eth0): 172.17.0.2/16
Locale: C.UTF-8

```

Figura 13: Docker Debian

```

dayana@debian:~$ sudo docker exec -it financiera bash
[root@b024741e91bd /]# ping -c 3 172.18.0.1
PING 172.18.0.1 (172.18.0.1) 56(84) bytes of data:
64 bytes from 172.18.0.1: icmp_seq=1 ttl=64 time=0.229 ms
64 bytes from 172.18.0.1: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 172.18.0.1: icmp_seq=3 ttl=64 time=0.124 ms

--- 172.18.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.121/0.158/0.229/0.050 ms
[root@b024741e91bd /]# exit
exit
dayana@debian:~$

```

Figura 14: Docker Debian

3.2. Creación de contenedor Fedora y Kali

Para estos dos manejamos los mismos pasos que con Debian, debemos crear el docker y correrlo.

```

FROM fedora:latest

# Mantenedor
LABEL maintainer="Dependencia Comercial y Ventas"

# Actualizamos e instalamos utilidades b sicas (sin
neofetch)
RUN dnf -y update --setopt=install_weak_deps=False && \

```

```

dnf -y install vim git sudo fish util-linux && \
dnf clean all

# Instalamos fastfetch (reemplazo moderno de neofetch)
RUN dnf -y install fastfetch || echo "fastfetch no
disponible, continuando..."

# Cambiamos el shell por defecto a fish
RUN chsh -s /usr/bin/fish ventas

# Definimos el usuario por defecto
USER ventas

# Definimos el shell por defecto
CMD ["/usr/bin/fish"]
construir la imagen de fedora

sudo docker build -t fedora-comercial .
correr el contenedor

sudo docker run -it --name contenedor-comercial fedora-
comercial

```

```

$ sudo docker exec -it ventas bash
[root@986f8eeb8e06 /]# ping -c 3 172.19.0.1
PING 172.19.0.1 (172.19.0.1) 56(84) bytes of data.
64 bytes from 172.19.0.1: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 172.19.0.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 172.19.0.1: icmp_seq=3 ttl=64 time=0.147 ms

--- 172.19.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2040ms
rtt min/avg/max/mdev = 0.044/0.104/0.147/0.043 ms
[root@986f8eeb8e06 /]#

```

Figura 15: Fedora



Figura 16: Kali

3.3. Interfaz de Arch Manager

Para la gestión visual de las máquinas virtuales y contenedores se utilizó la herramienta **Arch Manager** (Virt-Manager), la cual permite observar el estado, consumo de recursos y conexión de red de cada entorno virtualizado.

Desde esta interfaz se pudieron controlar las máquinas Debian, Arch y Rocky, además de gestionar los contenedores implementados posteriormente en Manjaro. Su uso facilitó la supervisión del sistema completo sin necesidad de depender exclusivamente de comandos en terminal.

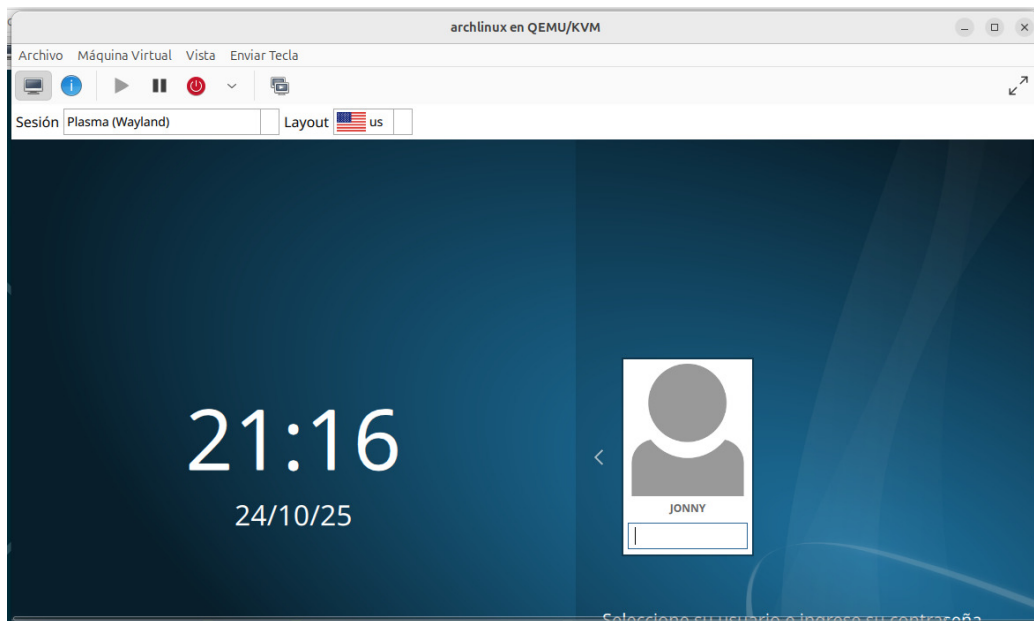


Figura 17: Interfaz de Arch Manager mostrando las máquinas activas

3.4. Instalación de Manjaro

Dentro del entorno DevOps, se realizó la instalación de **Manjaro Linux** como una plataforma adicional de administración y prueba. Esta distribución basada en Arch Linux fue elegida por su ligereza, entorno gráfico moderno y compatibilidad con Docker y LXC.

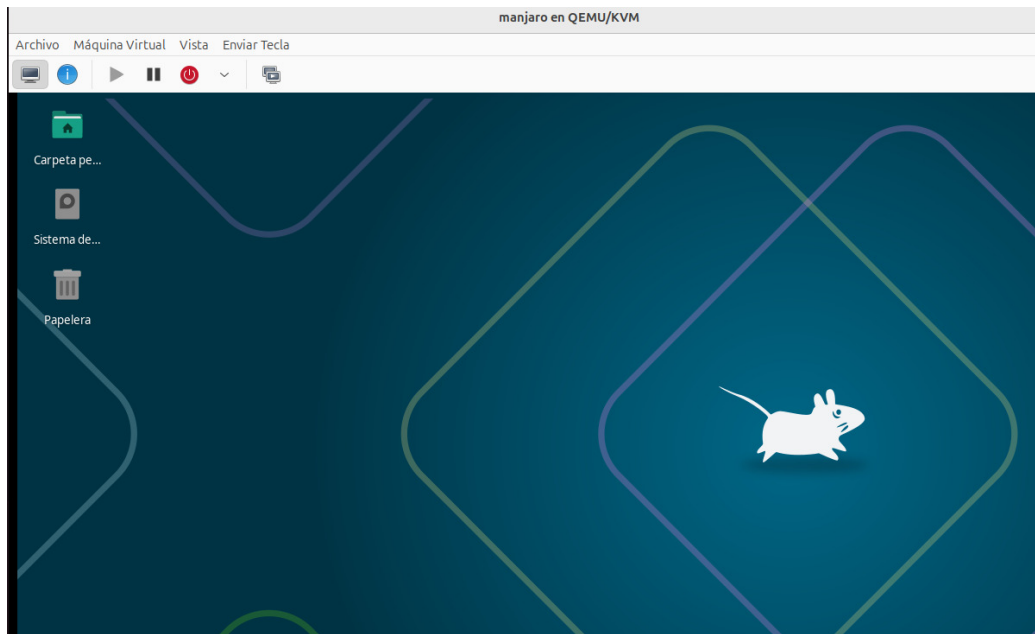


Figura 18: Descarga de la imagen ISO de Manjaro

Una vez completada la instalación, se actualizaron los repositorios y se habilitó Docker para la gestión de contenedores:

```
sudo pacman -Syu
sudo pacman -S docker
sudo systemctl enable --now docker
```

3.5. Creación del Contenedor Centenario (CentOS)

Dentro de Manjaro, se procedió a crear el contenedor **Centenario** u

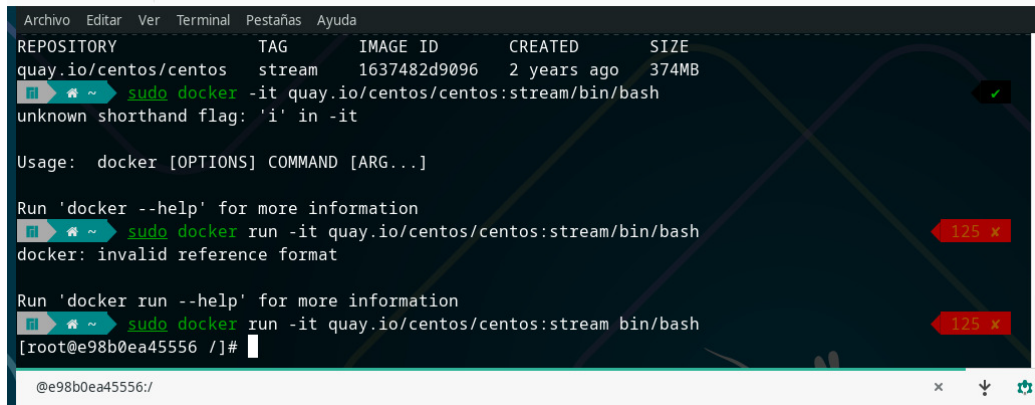
3.6. Creación del Contenedor Centenario (CentOS)

Dentro del sistema **Manjaro**, se procedió a crear el contenedor denominado **Centenario**, basado en la imagen oficial de **CentOS**. Este contenedor tuvo como objetivo comprobar la integración entre entornos de distintas familias de Linux (Arch y Red Hat), además de validar la comunicación con el servidor Debian.

```
sudo docker pull centos:8
sudo docker run -it --name centenario centos:8 /bin/bash
```

Una vez iniciado el contenedor, se configuraron las interfaces de red y las rutas de comunicación, asegurando su conexión con la red general.

```
ip addr add 192.168.200.10/24 dev eth0
ip route add default via 192.168.200.1
```



```
Archivo Editar Ver Terminal Pestañas Ayuda
REPOSITORY          TAG             IMAGE ID        CREATED        SIZE
quay.io/centos/centos  stream         1637482d9096   2 years ago   374MB
[~] [~] [~] ~ sudo docker run -it quay.io/centos/centos:stream/bin/bash
unknown shorthand flag: 'i' in -it

Usage: docker [OPTIONS] COMMAND [ARG...]

Run 'docker --help' for more information
[~] [~] [~] ~ sudo docker run -it quay.io/centos/centos:stream/bin/bash
docker: invalid reference format

Run 'docker run --help' for more information
[~] [~] [~] ~ sudo docker run -it quay.io/centos/centos:stream bin/bash
[root@e98b0ea45556 /]#
```

Figura 19: Contenedor Centenario (CentOS) ejecutándose dentro de Manjaro

Con esta implementación se logró validar la comunicación entre Manjaro y el contenedor Centenario, demostrando la correcta integración de los entornos contenerizados bajo una arquitectura DevOps híbrida.