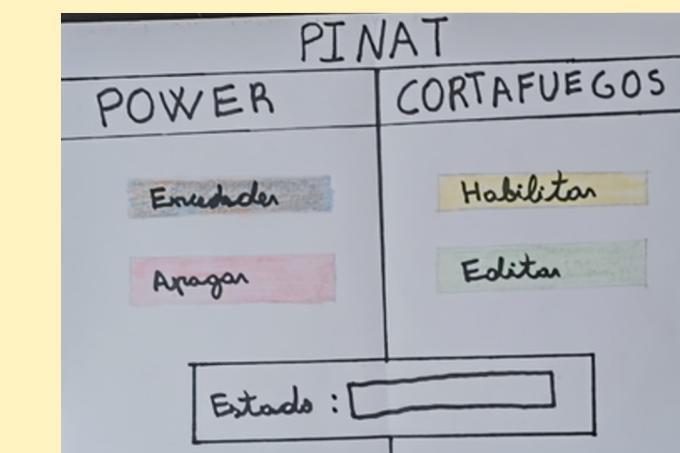


Interfaz gráfica



Esquema de PINAT entorno de Docker y API para segmentación por VLAN

Álex Torresano Organero



ÍNDICE



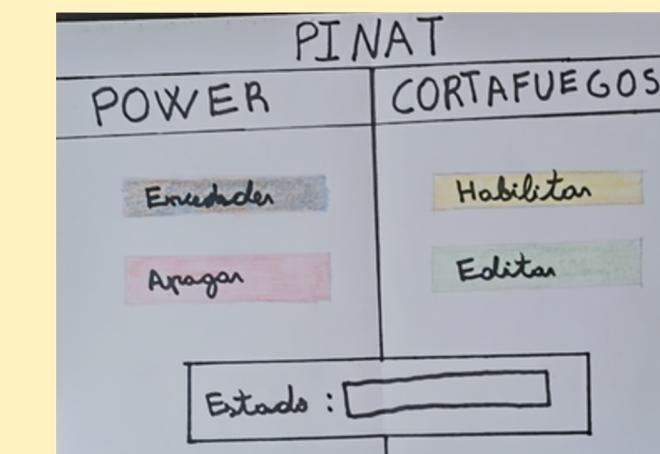


1 . INTRODUCCIÓN

¿Qué es?

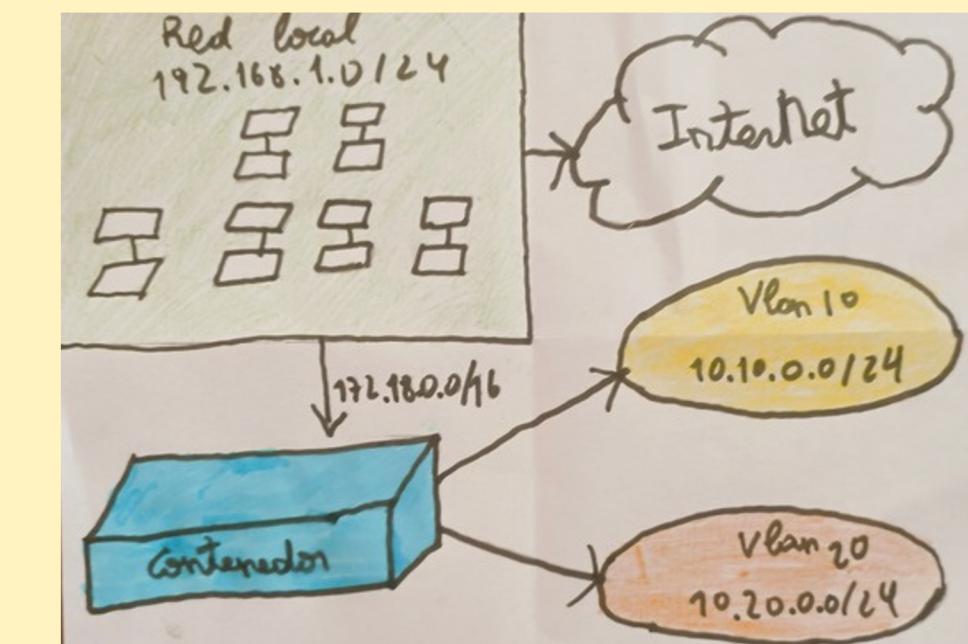
PINAT es una solución de red en contenedores que permite crear y gestionar VLANs de forma automatizada. Utiliza Docker, redes macvlan e iptables para enrutar tráfico entre redes virtuales y físicas. Incluye un servidor DHCP y una interfaz gráfica en Python para facilitar su administración. Permite mezclar redes Docker con VirtualBox y dispositivos reales. Está diseñado para entornos de laboratorio, pruebas y formación en networking.

Interfaz gráfica



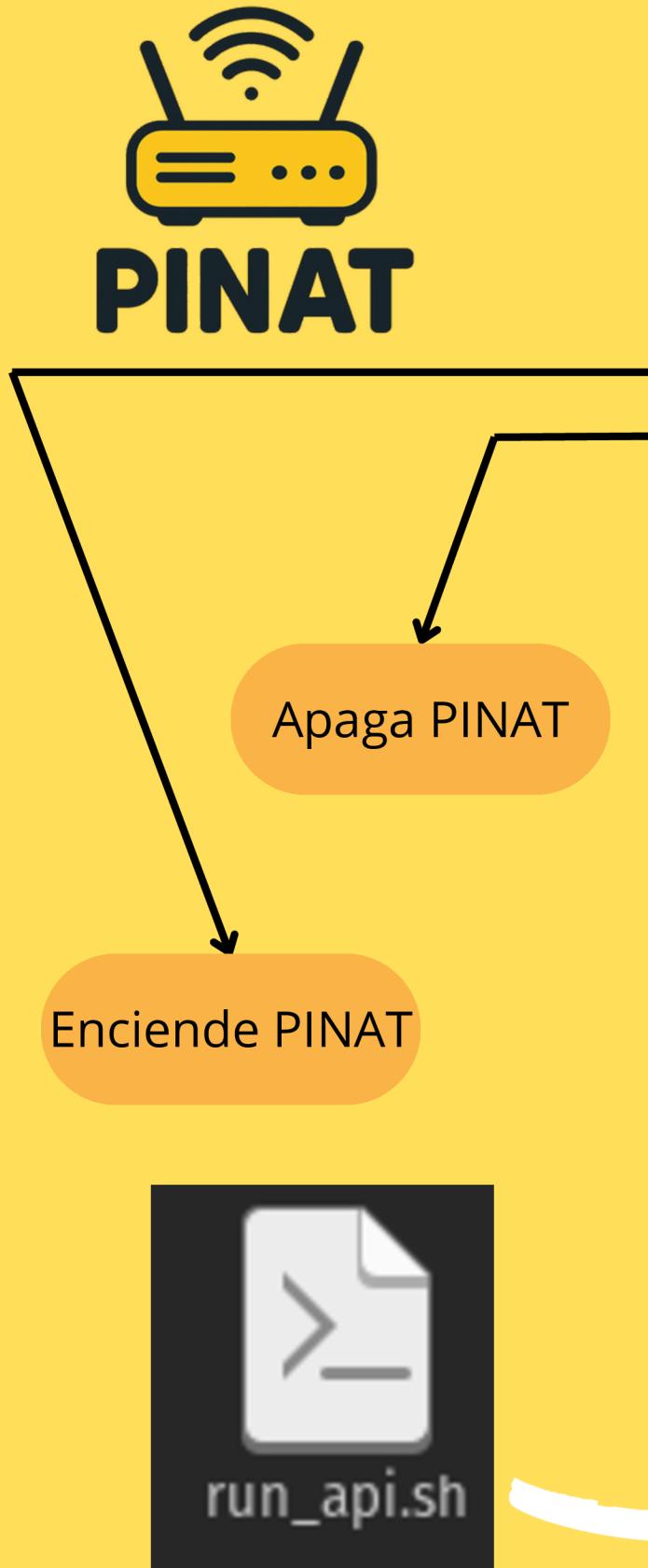
Bocetos del proyecto

Diagrama de red



Una VLAN es una red lógica creada dentro de una red física para segmentar el tráfico y mejorar el rendimiento, la seguridad y la administración de la red

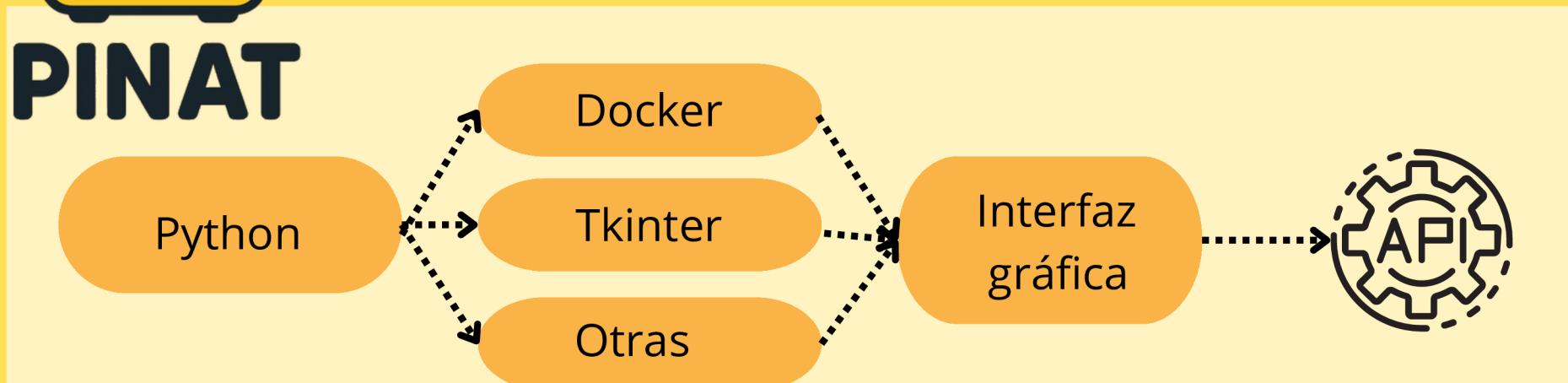
2. Instrucciones de uso





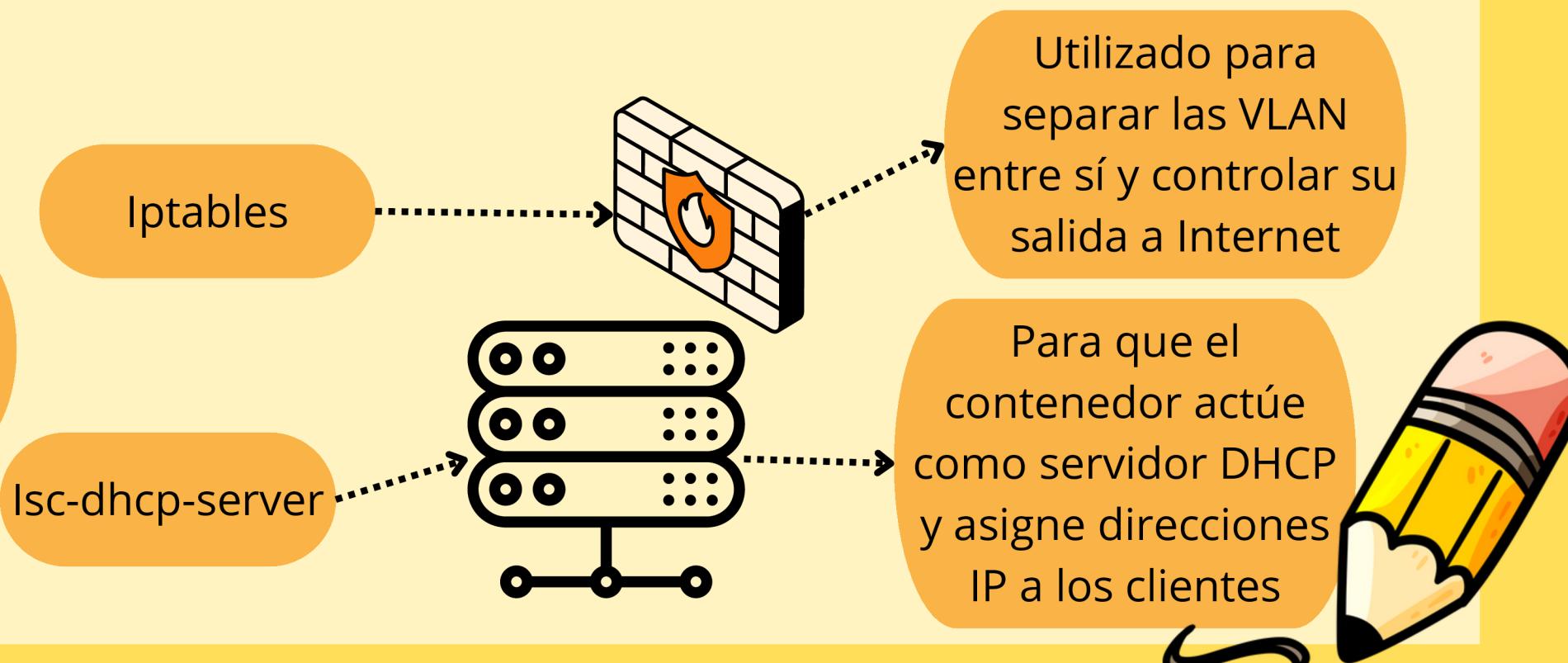
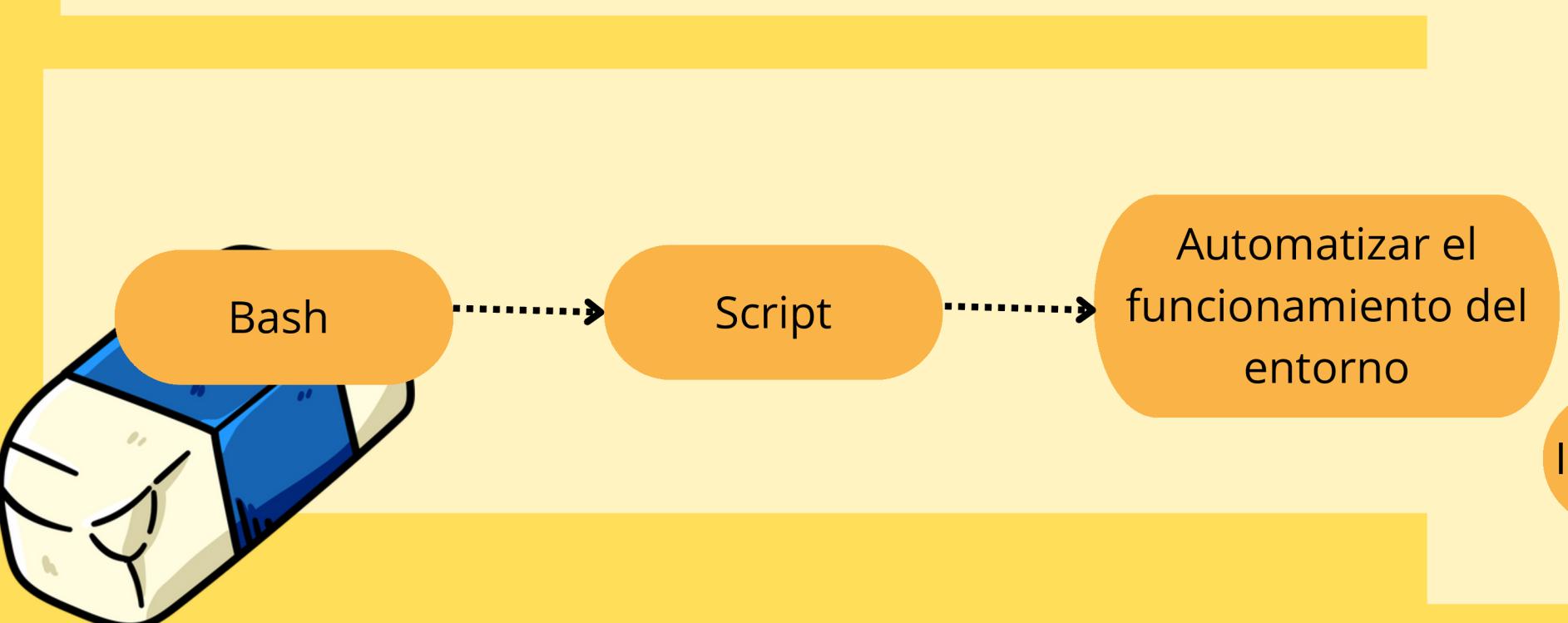
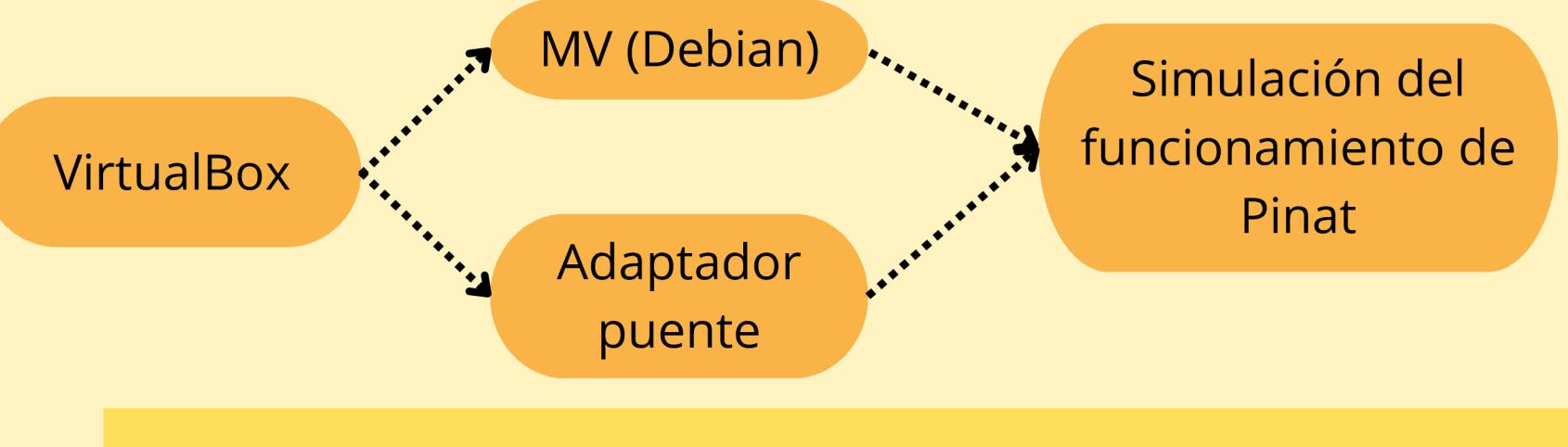
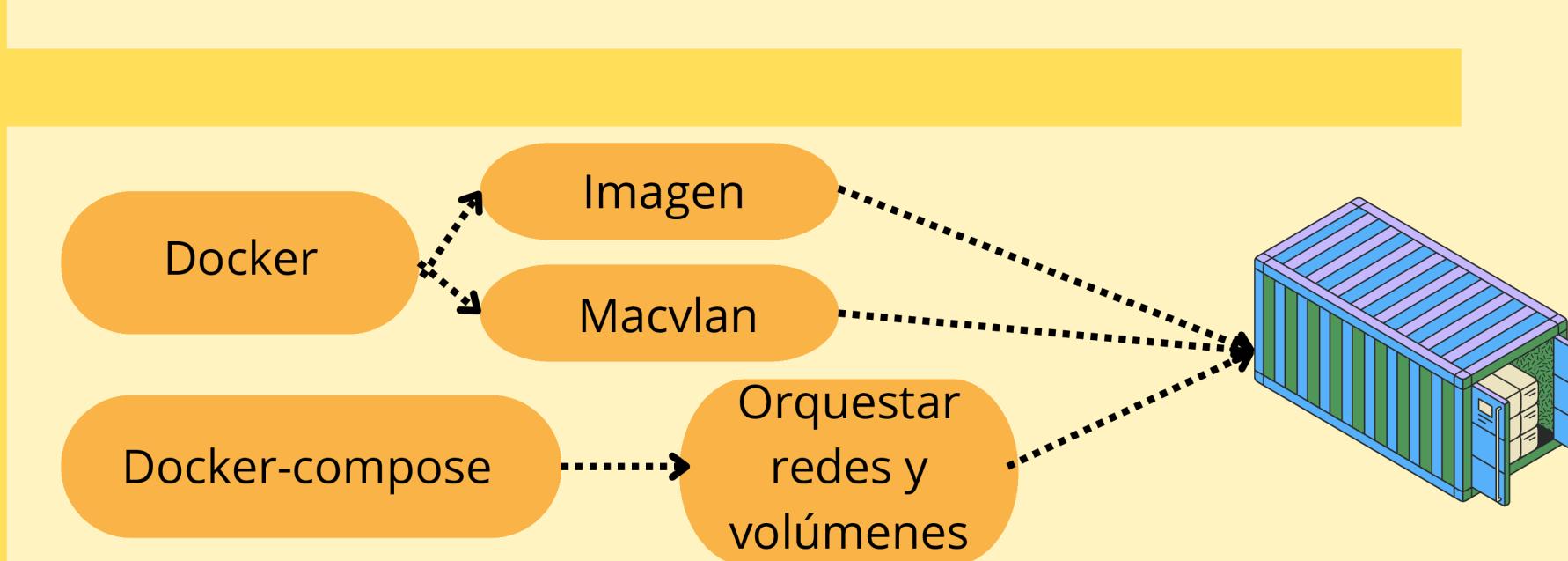
PINAT

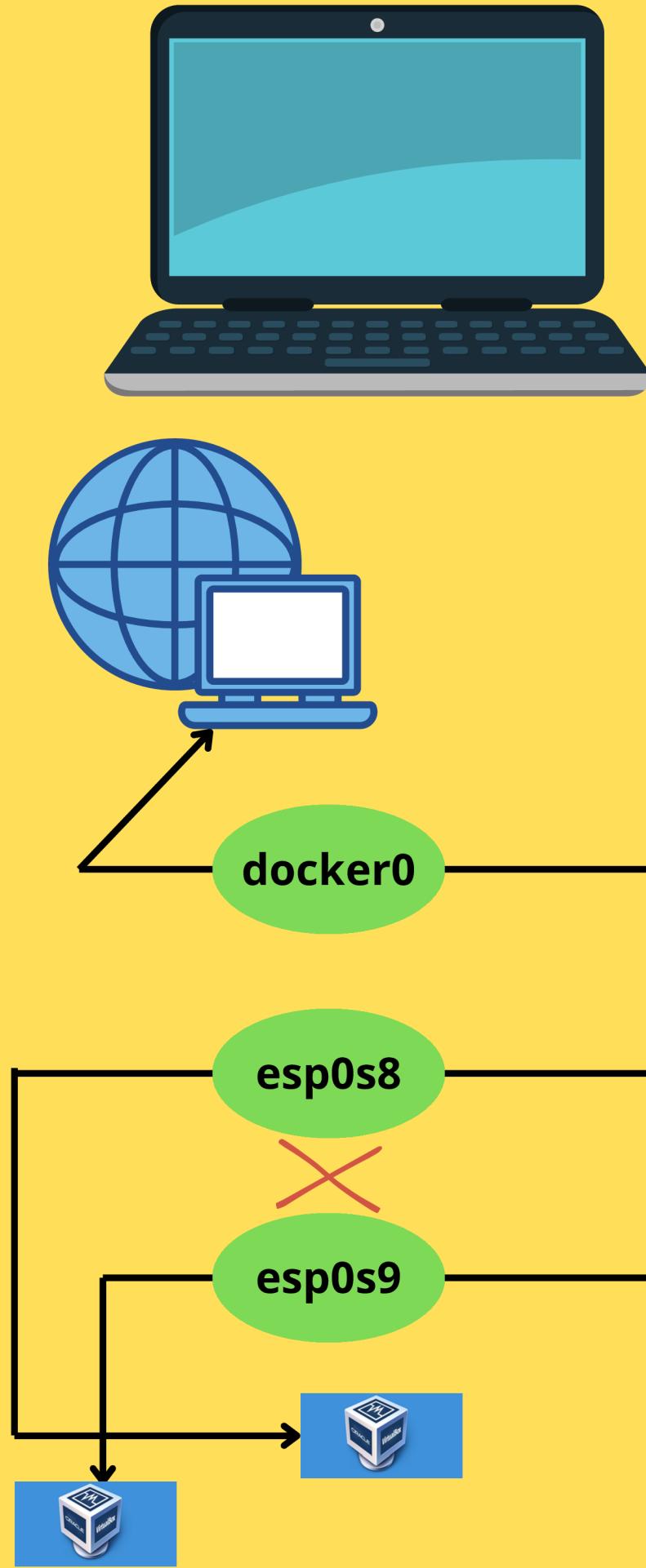
3. EXPLICACIÓN TÉCNICA



3.1

Esquemas de herramientas
y funciones





3.2

Redes entre host y contenedor



- 1 `docker0` y `eth0` forman parte de la red por defecto de Docker; `eth0` es la interfaz dentro del contenedor que le da acceso a Internet.
- 2 `enp0s8` en el host y `eth1` en el contenedor pertenecen a la red `macvlan_10`, una red de tipo macvlan que conecta directamente ambos.
- 3 `enp0s9` en el host y `eth2` en el contenedor pertenecen a la red `macvlan_20`, una red de tipo macvlan que conecta directamente ambos.



 La red por defecto de Docker es una red virtual tipo bridge que conecta los contenedores entre sí y con el host, usando NAT para permitirles acceder a Internet de forma segura y aislada.

 La red macvlan asigna a cada contenedor una interfaz de red propia con su dirección MAC, integrándolo directamente en la red física, como si fuera otro dispositivo conectado al mismo switch, lo que facilita la comunicación directa sin necesidad de NAT.

Cortafuegos (iptables)

3.3

```
# Limpiar todas las reglas existentes
iptables -F
iptables -X
iptables -Z
iptables -t nat -F
```

Elimina las reglas existentes para preparar la creación de nuevas reglas de firewall.

```
# Permitir reenvío desde eth1 hacia eth0
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT

# Permitir reenvío desde eth2 hacia eth0
iptables -A FORWARD -i eth2 -o eth0 -j ACCEPT

# (Opcional) Permitir el tráfico de respuesta de eth0 hacia eth1 y eth2
iptables -A FORWARD -i eth0 -o eth1 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth2 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Configuración para permitir el tráfico entre las interfaces VLAN y la interfaz eth0, asegurando el acceso controlado a Internet a través de eth0

```
# Habilitar NAT (mascaradeo) para que las redes internas salgan por eth0
iptables -t nat -A POSTROUTING -s 10.10.0.0/24 -o eth0 -j MASQUERADE
iptables -t nat -A POSTROUTING -s 10.20.0.0/24 -o eth0 -j MASQUERADE
```

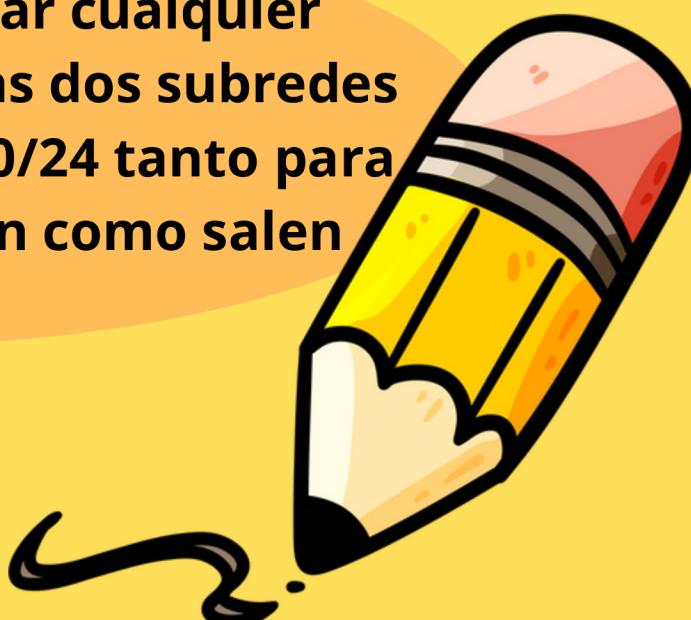
```
# Activar el reenvío IP en el kernel
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Habilita el reenvío de paquetes IP en el kernel para permitir que el host enrute tráfico entre interfaces

```
# Cortar el tráfico entre las VLAN
iptables -A OUTPUT -s 10.10.0.0/24 -d 10.20.0.0/24 -j DROP
iptables -A OUTPUT -s 10.20.0.0/24 -d 10.10.0.0/24 -j DROP
iptables -A INPUT -s 10.10.0.0/24 -d 10.20.0.0/24 -j DROP
iptables -A INPUT -s 10.20.0.0/24 -d 10.10.0.0/24 -j DROP
```

Reglas para bloquear cualquier comunicación entre las dos subredes 10.10.0.0/24 y 10.20.0.0/24 tanto para paquetes que entran como salen

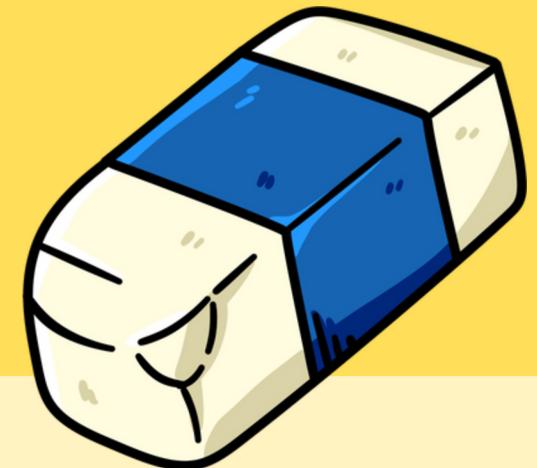
Permite que las subredes internas 10.10.0.0/24 y 10.20.0.0/24 usen la IP de la interfaz eth0 para salir a otras redes o Internet.





3.4

Servidor DHCP (isc-dhcp-server)



subnet 10.10.0.0 netmask 255.255.255.0 {
range 10.10.0.100 10.10.0.200;
option domain-name-servers 8.8.8.8;
option routers 10.10.0.2;
}

subnet 10.20.0.0 netmask 255.255.255.0 {
range 10.20.0.100 10.20.0.200;
option domain-name-servers 8.8.8.8;
option routers 10.20.0.2;
}

He creado dos subredes, una para cada VLAN. El cliente necesita una IP dinámica, así que pregunta a la red si hay un servidor disponible. Si las interfaces están conectadas, el contenedor responde y, según la configuración de la izquierda, el servidor DHCP le asigna una dirección IP al cliente





4 . Pruebas



4.1

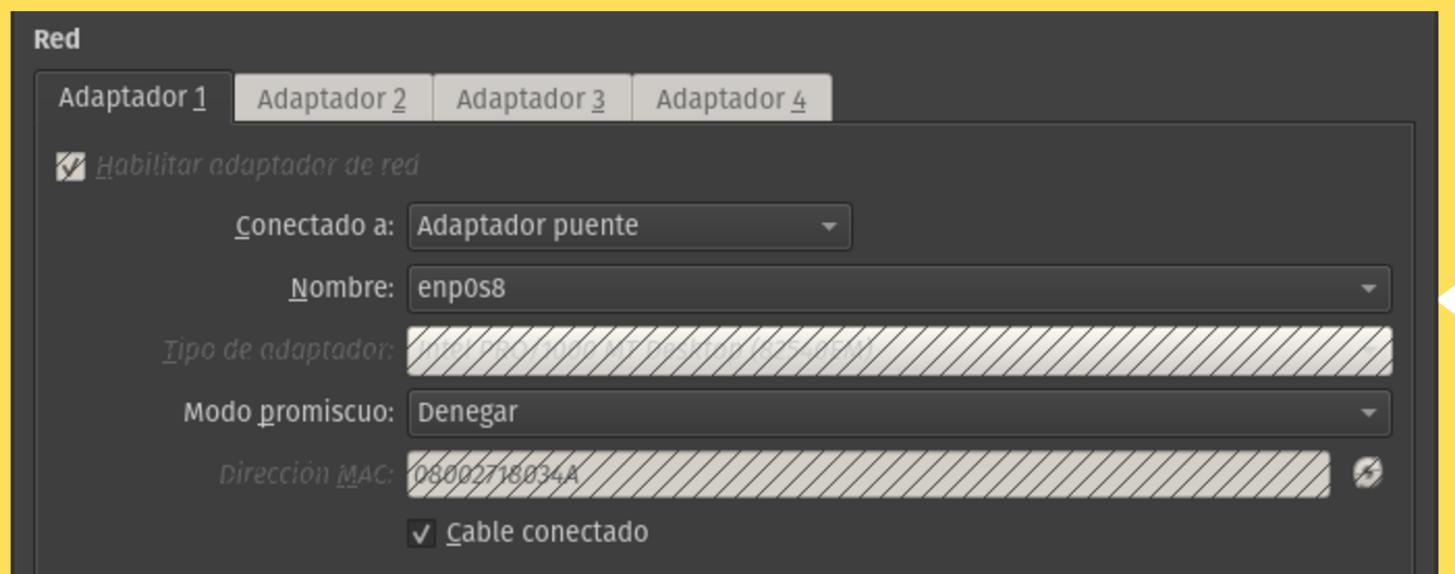
Prueba de DHCP

```
root@debian12:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

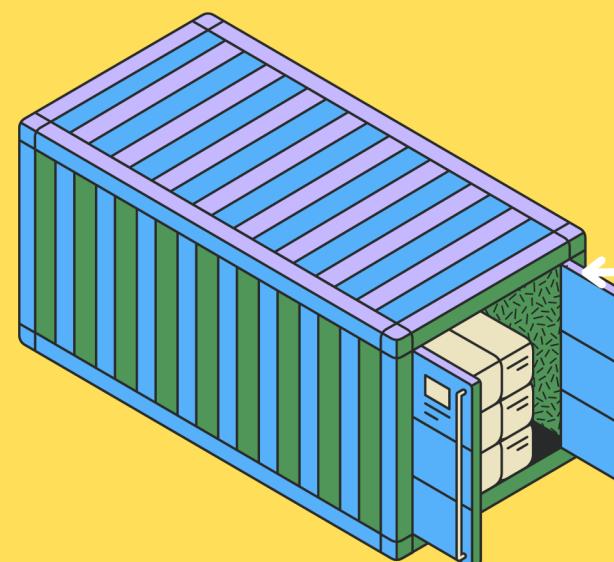
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet dhcp
root@debian12:~# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:18:03:4a brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.100/24 brd 10.10.0.255 scope global dynamic enp0s3
        valid_lft 578sec preferred_lft 578sec
    inet6 fe80::a00:27ff:fe18:34a/64 scope link
        valid_lft forever preferred_lft forever
root@debian12:~# _
```



eth1





```
root@debian12:~# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:18:03:4a brd ff:ff:ff:ff:ff:ff
    inet 10.20.0.100/24 brd 10.20.0.255 scope global dynamic enp0s3
        valid_lft 423sec preferred_lft 423sec
    inet6 fe80::a00:27ff:fe18:34a/64 scope link
        valid_lft forever preferred_lft forever
root@debian12:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=13.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=11.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=11.6 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 11.615/12.585/13.505/0.892 ms
root@debian12:~#
```

4.2

Prueba del cortafuegos (Firewall)

```
root@debian12:~# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:18:03:4a brd ff:ff:ff:ff:ff:ff
    inet 10.20.0.100/24 brd 10.20.0.255 scope global dynamic enp0s3
        valid_lft 367sec preferred_lft 367sec
    inet6 fe80::a00:27ff:fe18:34a/64 scope link
        valid_lft forever preferred_lft forever
root@debian12:~# ping 10.10.0.100
PING 10.10.0.100 (10.10.0.100) 56(84) bytes of data.
^C
--- 10.10.0.100 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5123ms
```





5. Conclusión

La idea de este proyecto surge de la integración de varios conocimientos adquiridos durante el curso de ASIR, complementados con contenidos del CCNA relacionados con redes. Esto me permitió entender conceptos fundamentales como el enrutamiento, redes lógicas, la configuración de NAT con iptables y la implementación de un servidor DHCP utilizando ISC-DHCP-Server.

Durante las prácticas también trabajé con VPNs mediante SonicWall, y mi interés por DevOps me llevó a aplicar estos conceptos en entornos Docker gestionados mediante un panel de control. Así, pensé en crear una especie de router capaz de manejar múltiples VLAN; hasta ahora se han creado dos, aunque es posible administrar muchas más.

Una vez definidas las redes lógicas, el siguiente reto fue segmentarlas y controlar su salida a Internet utilizando iptables. Para hacer el sistema más dinámico, incorporé un servidor DHCP. En el proceso descubrí las redes macvlan, lo que me permitió crear interfaces virtuales en Linux para aislar las redes y conectar máquinas virtuales mediante adaptadores en modo puente.

Una vez definidas las redes lógicas, el siguiente reto fue segmentarlas y controlar su salida a Internet utilizando iptables. Para hacer el sistema más dinámico, incorporé un servidor DHCP. En el proceso descubrí las redes macvlan, lo que me permitió crear interfaces virtuales en Linux para aislar las redes y conectar máquinas virtuales mediante adaptadores en modo puente.

