# Aplicación de técnicas de virtualización ligera para la evaluación de redes de comunicaciones

Trabajo Final de Estudios
Ingeniería Telemática

Enrique Fernández Sánchez
Universidad Politécnica de Cartagena

Revisión 25 Mayo 2021

# Índice

# 1 Introduction

# 2 Interfaces de red en *Linux*

Linux dispone de una selección muy diferente de interfaces de red que nos permiten, de manera sencilla, el uso de máquinas virtuales o contenedores. En este apartado vamos a mencionar las interfaces más relevantes de cara a la virtualización ligera que proponemos para el despliegue de una red virtualizada. Para obtener una lista completa de las interfaces disponibles, podemos ejecutar el siguiente comando `ip link help`.

En este trabajo, vamos a comentar la siguientes interfaces:

- eth0:{0,1,2...}
- eth0.{0,1,2...} VLAN
- eth0.{0,1,2...} VLAN 802.1ad
- VETH pairs
- TUN/TAP

## 2.1 Nombres predecibles interfaces de red. `systemd networkd v197`

Para la explicación de las diferentes interfaces, vamos a suponer que estamos utilizando el nombrado de interfaces de red antiguo. Esto proviene de que en la últimas versiones del kernel, se ha cambiado la forma en la que las interfaces de red son nombradas por Linux. Es por esto por lo que antes podíamos tener interfaces tal que `eth0` y ahora nos encontramos con la siguiente nomenclatura `enps30`. Este cambio surge ya que anteriormente se nombraban las diferentes interfaces conforme el propio ordenador estaba en la etapa de `boot`, por lo que podría pasar que a lo que nosotros entendíamos como `eth1`, en el próximo arranque fuera `eth0`, dando lugar a incontables errores en el sistema. Es por esto por lo que se empecó a trabajar en soluciones alternativas. Por ejemplo, la que hemos utilizado de ejemplo, utiliza la información aportada por la BIOS del dispositivo para catalogarlo en diferentes categorías, con su formato de nombre para cada categorías. Dichas clasificación corresponde con las siguientes:

1. Nombres incorporando Firmware/BIOS que proporcionan un número asociado a dispositivos en la placa base. (ejemplo: `eno1`)
2. Nombres incorporando Firmware/BIOS proveniente de una conexión PCI Express hotplug, con número asociado al conector. (ejemplo: `ens1`)
3. Nombres que incorporan una localización física de un conector hardware. (ejemplo: `enp2s0`)
4. Nombres que incorporan una la MAC de una interfaz. (ejemplo: `enx78e7d1ea46da`)
5. Sistema clásico e impredecible, asignación de nombres nativa del kernel. (ejemplo: `eth0`)

## 2.2 `enp2s0:{0,1,2...}`

Todas las interfaces asociadas comparten la misma dirección MAC. Cada una de ellas, recibe el nombre de *aliases*. La funcionalidad principal que tienen este tipo de interfaces es la de asignar varias direcciones de IP a una misma interfaz de red.

```
$ ip addr add 192.168.56.151/24 broadcast 192.168.56.255 dev enp2s0
label enp2s0:1
```

Sin embargo, el comando `iproute2` admite esta misma funcionalidad sin tener que crear interfaces de red extra. Para ello, solo tenemos que asociar cada IP con la interfaz de red deseada.

```
$ ip addr add 192.168.56.151/24 dev enp2s0
$ ip addr add 192.168.56.251/24 dev enp2s0
```

## 2.3 VLAN. `enp2s0.{0,1,2...}`

Mismo concepto que la interfaz anterior, sin embargo, utilizamos el estandar 802.1q, que permite etiquetar las tramas, para crear una red lógica independiente. Es necesario que la interfaz a la que estemos asignando, sea un puerto trunk, o bien sea tagged para una VLAN específica.

```
$ ip link add link enp2s0 name enp2s0.{num} type vlan id {num}
$ ip addr add 192.168.100.1/24 brd 192.168.100.255 dev enp2s0.{num}
$ ip link set dev enp2s0.{num} up
```

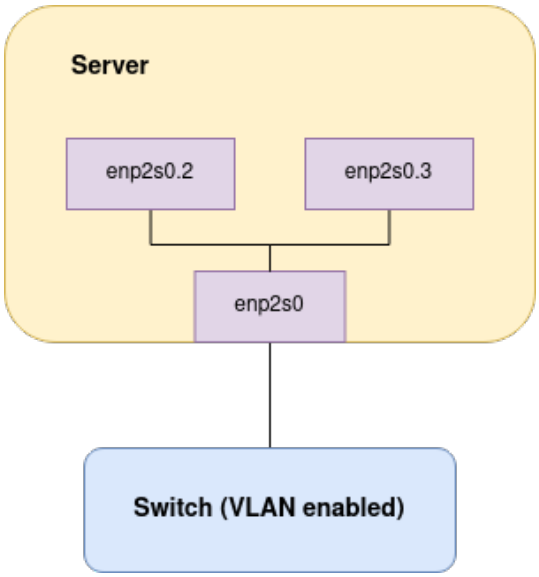


Figura 1: Diagrama conexión VLANs

## 2.4 VLAN 802.1ad. `enp2s0.{0,1,2...}`

## 2.5 Pares VETH.

Los VETH (Ethernet virtuales) son un dispositivo que forman un túnel local ethernet. El dispositivo se crea en parejas.

Los paquetes transmitidos por un extremo del VETH se reciben inmediatamente en el otro extremo. Si alguno de ellos se encuentra apagado, decimos que el link de la pareja esta también apagado.

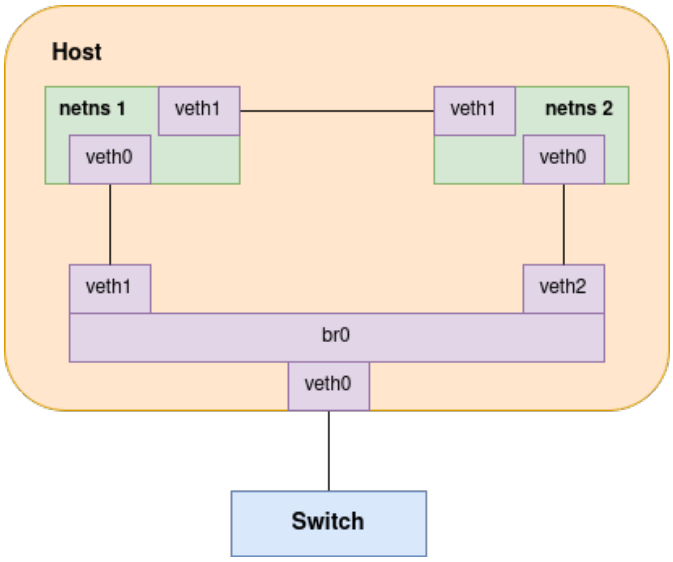


Figura 2: Diagrama VETH y netns

Es necesario utilizar VETH cuando un namespace necesita comunicarse con el host, o viceversa.

La configuración que tenemos que realizar para tener el diagrama de la Figura 2.

```
$ ip netns add net1
$ ip netns add net2
$ ip link add veth1 netns net1 type veth peer name veth1 netns net2
```

De esta manera, tendríamos creados los namespaces `net1` y `net2`, que estarían interconectados con las interfaces `veth1` y `veth2`. Nos quedaría asignar una IP a cada interfaz.

```
$ ip netns exec net1 ip addr add 10.1.1.1/24 dev veth1
$ ip netns exec net1 ip link set veth1 up
$ ip netns exec net2 ip addr add 10.1.1.2/24 dev veth1
$ ip netns exec net2 ip link set veth1 up
```

Ahora nos quedaría la parte de crear un bridge entre las diferentes interfaces virtuales que hemos creado. Para esto, podemos utilizar una interfaz tipo *bridge* de Linux, o bien podemos configurar un *bridge* usando *Open VSwitch*.

```
$ ip link add veth1_br type veth peer name veth0 netns net1
$ ip link add veth2_br type veth peer name veth0 netns net2
$ ovs-vsctl add-br ovsbr0
$ ovs-vsctl add-port ovsbr0 veth1_br
$ ovs-vsctl add-port ovsbr0 veth2_br
$ ovs-vsctl add-port ovsbr0 <outbound interface>
```

Añadimos las direcciones IP que nos faltan:

```
$ ip addr add 10.1.1.10/24 dev veth1_br
$ ip link set veth1_br up
$ ip addr add 10.1.1.20/24 dev veth2_br
$ ip link set veth2_br up
$ ip netns exec net1 ip addr add 10.1.1.15/24 dev veth0
$ ip netns exec net1 ip link set veth0 up
$ ip netns exec net2 ip addr add 10.1.1.25/24 dev veth0
$ ip netns exec net2 ip link set veth0 up
$ ip netns exec net1 ip link set lo up
$ ip netns exec net2 ip link set lo up
```

## 2.6 TUN/TAP

Permite conectar una aplicación en específico con el kernel, utilizando un dispositivo Ethernet. Una aplicación puede acceder al tuntap o bien crearlo ella misma.

TUN trabaja a nivel IP (capa 3), tiene por función principal definir túneles. No entiende nada de ARP (ni nada de nivel 2).

TAP trabaja solo a nivel de MAC (entiende ARP, etc.) y sirve directamente para un bridge virtual. Los túneles los puede crear directamente el comando ip, por lo que la interfaz TUN apenas se utiliza. Sin embargo, TAP es muy utilizado para crear un *bridge*.

Un *bridge* puede incorporar interfaces TAP. En este caso, la aplicación que lo monitoriza puede ser un hipervisor (Virtualbox, Qemu, libvirt), es decir, el *bridge* está dentro del kernel.

# 3 Espacio de nombres en *Linux*

## 3.1 ¿Qué es un *espacio de nombres*?

Los *espacios de nombres*, o también llamados, *namespaces*, son una característica del kernel de Linux que permite gestionar los recursos del kernel, pudiendo limitarlos a un proceso o grupo de procesos. Suponen una base de tecnología que aparece en las técnicas de virtualización más modernas (como puede ser Docker, Kubernetes, etc). A un nivel alto, permiten aislar procesos respecto al resto del kernel.

El objetivo de cada *namespaces* es adquirir una característica global del sistema como una abstracción que haga parecer a los procesos de dentro del *namespace* que tienen su propia instancia aislada del recurso global.

## 3.2 ¿Cúantos *namespaces* hay?

El kernel ha estado en contaste evolución desde que 1991, cuando Linus Torvalds comenzó el proyecto, actualmente sigue muy activo y se siguen añadiendo nuevas características. El origen de los namespaces se remonta a la versión del kernel 2.4.19, lanzada en 2002. Conforme fueron pasando los años, más tipos diferentes de namespaces se fueron añadiendo a Linux. El concepto de *User namespaces*, se consideró terminado con la versión 3.9.

Actualmente, tenemos 8 tipos diferentes de namespaces, siendo el último añadido en la versión 5.8 (lanzada el 2 de Agosto de 2020).

1. UTS (hostname)
2. Mount (mnt)
3. Process ID (pid)
4. Network (net)
5. User ID (user)
6. Interprocess Communication (ipc)
7. Control group (cgroup)
8. Time

### 3.2.1 UTS namespace

El tipo más sencillo de todos los namespaces. La funcionalidad consiste en controlar el hostname asociado del ordenador, en este caso, del proceso o procesos asignados al namespace. Existen tres diferentes rutinas que nos permiten obtener y modifcar el hostname:

- *sethostname()*
- *setdomainname()*
- *uname()*

En una situación normal sin namespaces, se modificaría una String global, sin embargo, si estamos dentro de un namespace, los procesos asociados tienen su propia variable global asignada.

Un ejemplo muy basico de uso de este namespaces podría ser el siguiente:

Listado 1: Example usage UTS namespace

```
$ sudo su                          # super user
$ hostname                         # current hostname
> arch-linux
$ unshare -u /bin/sh               # shell with UTS namespace
$ hostname new-hostname            # set hostname
$ hostname                         # check hostname of the shell
> new-hostname
$ exit                             # exit shell and namespace
$ hostname                         # original hostname
> arch-linux
```

En el ejemplo planteado, vemos que utilizamos el comando `unshare`. Utilizando la documentación de dicho comando, `man unshare`. Podemos deducir los siguiente:

- Ejecuta un programa con algunos namespaces diferentes del host.
- En los parametros podemos especificar cual o cuales namespaces queremos desvincular.
- Tenemos que especificar la ruta del ejecutable que queremos aislar
- La sintaxis sería tal que: `unshare [options] <program>[<argument>...]`

### 3.2.2 Mount namespace

Un *mount namespace (mnt)* supone otro tipo de espacio de nombres, en este caso relacionado con los *mounts* de nuestro sistema. Lo primero es entender a que nos referimos cuando hablamos de *mount*. *Mount*, o montaje, hace referencia a conectar un sistema de archivos adicional que sea accesible para el sistema de archivos actual de un ordenador. Un mount, tiene asignado lo que se llama *mount point*, que corresponde con el directorio en el que está accesible el sistema de archivo que previamente hemos montado.

Por lo tanto, un namespace de tipo *mount* nos permite modificar un sistema de archivos en concreto, sin que el host pueda ver y/o acceder a dicho sistema de archivos. Un ejemplo básico de esta funcionalidad podría ser la siguiente:

Listado 2: Example of usage mount namespace

```
$ sudo su                    # run a shell in a new mount namespace
$ unshare –m /bin/sh
$ mount —bind /usr/bin/ /mnt/
$ ls /mnt/cp
> /mnt/cp
$ exit                       # exit the shell and close namespace
$ ls /mnt/cp
> ls: cannot access '/mnt/cp': No such file or directory
```

Como vemos en el ejemplo, dentro del namespaces lo que hacemos es crear un *mount* de tipo *bind*, que tiene por función que un archivo de la máquina host se monte en un directorio en específico, en este caso, un directorio unicamente del programa que hemos asignado al namespace. Otro ejemplo de uso de estos namespaces es crear un sistema de archivos temporal que solo sea visible para ese proceso.

### 3.2.3 Process ID namespace

Para entender en que consiste este namespace, primero tenemos que conocer la definición de *process id* dentro del Kernel. En este caso, *process id* hace referencia a un número entero que utiliza el Kernel para identificar los procesos de manera unívoca.

Concretando, aísla el namespace de la ID del proceso asignado, dando lugar a que, por ejemplo, otros namespaces puedan tener el mismo PID. Esto nos lleva a la situación de que un proceso dentro de un *PID namespace* piense que tiene asignado el ID "1", mientras que en la realidad (en la máquina host) tiene otro ID asignado.

Listado 3: Example of usage process id namespace

```
$ echo $$                  # PID de la shell
$ ls -l /proc/$$/ns        # ID espacios de nombres
$ sudo unshare -f --mount-proc -p /bin/sh
$ echo $$                  # PID de la shell dentro del ns
$ ls -l /proc/$$/ns        # nuevos ID espacio de nombres
$ ps

$ ps -ef                   # ejecutar en una shell fuera del ns. Comparar PID
$ exit
```

Si ejecutamos el ejemplo, lo que podemos comprobar es que el ID del proceso que está dentro del namespaces (`echo $$`), no coincide con el proceso que podemos ver de la máquina host (`ps -ef | grep /bin/sh`). Más concretamente, el primer proceso creado en un PID namespace recibirá el pid número 1, y además de un tratamiento especial ya que supone un `init process` dentro de ese namespace.

### 3.2.4 Network namespace

Este namespaces nos permite aislar la parte red de una aplicación o proceso que nosotros elijamos. Con esto conseguimos que el *stack* de red de la máquina host sea diferente al que tenemos en nuestro namespace. Debido a esto, el namespace crea una interfaz virtual, conjunto con el resto de necesidades para conformar un stack de red completo (tabla de enrutamiento, tabla ARP, etc...).

Para crear un *namespace* de tipo *network*, y que este sea persistente, utilizamos la *tool* ip (del *package* iproute2).

Listado 4: Creation persistent network namespace

```
$ ip netns add ns1
```

Este comando creará un network namespace llamado ns1. Cuando se crea dicho namespace, el comando ip realiza un montaje tipo bind en la ruta /var/run/netns, permitiendo que el namespace sea persistente aún sin tener un proceso asociado.

Listado 5: Comprobar network namespaces existentes

```
$ ls /var/run/netns
or
$ ip netns
```

Como ejemplo, podemos proceder a añadir una interfaz de *loopback* al namespace que previamente hemos creado:

Listado 6: Asignar interfaz loopback a un namespace

```
$ ip netns exec ns1 ip link dev lo up
$ ip netns exec ns1 ping 127.0.0.1
> PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
> 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.115 ms
```

La primera línea de este ejemplo, corresponde con la directiva que le dice al namespace que "levante"la interfaz de loopback. La segunda línea, vemos como el namespace ns1 ejecuta el ping a la interfaz de loopback (el loopback de ese namespace).

Es importante mencionar, que aunque existen más comandos para gestionar las redes dentro de linux (como pueden ser ifconfig, route, etc), el comando ip es el considerado sucesor de todos estos, y los anteriores mencionados, dejarán de formar parte de Linux en versiones posteriores. Un detalle a tener en cuenta con el comando ip, es que es necesario tener privilegios de administrador para poder usarlo, por lo que deberemos ser root o utilizar sudo.

Por lo tanto, utilizando el comando ip, podemos recapitular que si utilizamos la siguiente directiva, podemos ejecutar el comando que nosotros indiquemos, pero dentro del network namespace que previamente hemos creado.

Listado 7: Ejecutar cualquier programa con un network namespace

```
$ ip netns exec <network-namespace> <command>
```

#### 3.2.4.1 Ejemplo práctico

Una de las problemáticas que supone el uso de los network namespaces, es que solo podemos asignar **una interfaz** a **un namespace**. Suponiendo el caso en el que el usuario root tenga asignada la interfaz eth0 (identificador de una interfaz de red física), significaría que solo los programas en el namespace de root podrán acceder a dicha interfaz. En el caso de que eth0 sea la salida a Internet de nuestro sistema, pues eso conllevaría que no podríamos tener conexión a Internet en nuestros namespaces. La solución para esto reside en los **veth-pair**.

Un veth-pair funciona como si fuera un cable físico, es decir, interconecta dos dispositivos, en este caso, interfaces virtuales. Consiste en dos interfaces virtuales, una de ellas asignada al root namespace, y la otra asignada a otro network namespace diferente. Si a esta arquitectura le añadimos una configuración de IP válida y activamos la opción de hacer NAT en el eth0 del host, podemos dar conectividad de Internet al network namespace que hayamos conectado.

Listado 8: Ejemplo configuración de NAT entre eth0 y veth

```
# Remove namespace if exists
$ ip netns del ns1 &>/dev/null

# Create namespace
$ ip netns add ns1

# Create veth link
$ ip link add v-eth1 type veth peer name v-peer1

# Add peer-1 to namespace.
$ ip link set v-peer1 netns ns1

# Setup IP address of v-eth1
$ ip addr add 10.200.1.1/24 dev v-eth1
$ ip link set v-eth1 up

# Setup IP address of v-peer1
$ ip netns exec ns1 ip addr add 10.200.1.2/24 dev v-peer1
$ ip netns exec ns1 ip link set v-peer1 up
# Enabling loopback inside ns1
$ ip netns exec ns1 ip link set lo up

# All traffic leaving ns1 go through v-eth1
$ ip netns exec ns1 ip route add default via 10.200.1.1
```

Siguiendo el ejemplo propuesto, llegamos hasta el punto en el que el tráfico saliente del namespace ns1, será redirigido a v-eth1. Sin embargo, esto no es suficiente para tener conexión a Internet. Tenemos que configurar el NAT en el eth0.

Listado 9: Configuración de NAT para dar Internet a un network namespace

```
# Share internet access between host and NS

# Enable IP-forwarding
$ echo 1 > /proc/sys/net/ipv4/ip_forward

# Flush forward rules, policy DROP by default
$ iptables -P FORWARD DROP
$ iptables -F FORWARD

# Flush nat rules.
$ iptables -t nat -F

# Enable masquerading of 10.200.1.0 (ip of namespaces)
$ iptables -t nat -A POSTROUTING -s 10.200.1.0/255.255.255.0 -o eth0
        -j MASQUERADE

# Allow forwarding between eth0 and v-eth1
$ iptables -A FORWARD -i eth0 -o v-eth1 -j ACCEPT
$ iptables -A FORWARD -o eth0 -i v-eth1 -j ACCEPT
```

Si todo lo hemos configurado correctamente, ahora podríamos realizar un ping hacia Internet, y este nos debería resultar satisfactorio.

```
$ ip netns exec ns1 ping google.es
> 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=50 time=48.5ms
> 64 bytes from 8.8.8.8: icmp_seq=2 ttl=50 time=58.5ms
```

Aún así, no resulta muy cómodo el utilizar `ip netns exec` seguido de la aplicación a utilizar. Es por esto por lo que es común ejecutar dicho comando para asignar el network namespace a una shell. Esto sería tal que así:

```
$ ip netns exec ns1 /bin/bash
```

Utilizaremos `exit` para salir de la shell y abandonar el network namespace.

### 3.2.5 User ID (user)

Cada sistema dispone de una manera de monitorizar que usuario es el dueño de cada archivo. Esto permite al sistema restringir el acceso a aquellos archivos que consideramos sensibles. Además, bloquea el acceso entre diferentes usuarios dentro del mismo sistema. Para el usuario, este identificador de usuarios se muestra como el usuario que en ese momento está conectado, sin embargo, para nuestro sistema, el identificador de usuario esta compuesto por una combinación arbitraria de caracteres alfanuméricos. Con el fin de mantener el monitoreo correctamente, hay un proceso encargado de transformar esos caracteres a un número específico de identificación (UID), como por ejemplo sería 1000. Es este valor el que se asocia con los archivos creados por este usuario. Esto nos aporta la ventaja de que, si un usuario cambia su nombre, no es necesario reconstruir el sistema de archivos, ya que su UID sigue siendo 1000.

Si por ejemplo queremos ver el UID del usuario que estamos usando en este momento, podemos ejecutar: *echo $UID*, el cual nos devolverá el número asociado a nuestro usuario, en mi caso es el 1000.

Además de diferenciar entre los IDs de usuarios (UID), también se nos permite separar entre IDs de grupos (GID). En linux, un grupo sirve para agrupar usuarios de modo que un grupo puede tener asociado un privilegio que le permite usar un recurso o programas.

Por lo tanto, el namespace de UID, lo que nos permite es tener un UID y GID diferente al del host.

Listado 10: Ejemplo de uso UID namespace

```
$ ls -l /proc/$$/ns                # espacios de nombres originales
$ id
> uid=1000(user) gid=1000(user) groups=1000(user), ...
$ unshare -r -u bash     # Crea un namespace de tipo usuario, programa bash
$ id
> uid=0(root) gid=0(root) groups=0(root),65534(nobody)
$ cat /proc/$$/uid_map
>          0       1000          1
$ cat /etc/shadow                  # No nos deja acceder
> cat: /etc/shadow: Permission denied
$ exit
```

Como vemos en el ejemplo, el UID de usuario difiere de la máquina host. Dentro del namespace, tenemos UID 0, sin embargo, eso no significa que podamos acceder a los archivos con UID 0 de la máquina host, ya que en verdad lo que hace el namespace es *mapear* el UID 1000 al 0.

### 3.2.6 Interprocess Communication namespace (IPC)

Este namespace supone uno de los más técnicos, complicados de entender y explicar. IPC (Interprocess comunication) controla la comunicación entre procesos, utilizando zonas de la memoría que están compartidas, colas de mensajes, y semáforos. La aplicación más común para este tipo de gestión es el uso en bases de datos.

### 3.2.7 Control group (cgroup)

Los grupos de control, cgroups, de Linux suponen un mecanismo para controlar los diferentes recursos de nuestro sistema. Cuando CGroups están activos, pueden controlar la cantidad de CPU, RAM, acceso I/O, o cualquier faceta que un proceso puede consumir. Además, permiten definir jerarquías en las que se agrupan, de manera en la que el administrador del sistema puede definir como se asignan los recursos o llevar la contabilidad de los mismos.

Por defecto, los CGroups se crean en el sistema de archivos virtual /sys/fs/cgroup. Si creamos un namespace de tipo CGroups, lo que estamos haciendo es mover el espacio de archivos virtual de dicho CGroup. Un ejemplo de esto sería, creamos un CGroup namespace en el directorio /sys/fs/cgroup/mycgroup. El host verá lo siguiente /sys/fs/cgroup/mycgroup/{group1, group2, group3}, sin embargo, el namespace solo verá {group1, group2, group3}. Esto es así ya que aporta seguridad a un namespace ya que los procesos del namespace solo pueden acceder a su sistema de archivos.

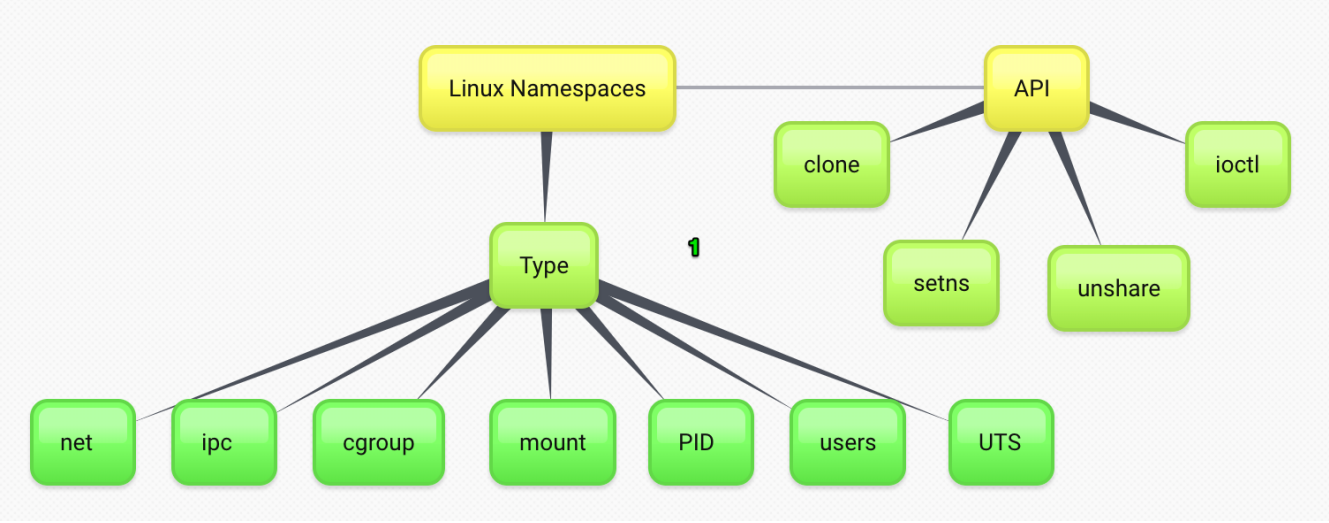


Figura 3: Diferentes namespaces en Linux y su API de acceso. (1)

### 3.2.8 Time

Por último, nos queda el namespaces asociado al tiempo. Este namespace fue propuesto para que se incorporara al kernel de Linux en 2018 y en enero de 2020 fue añadido a la versión mainline de Linux. Apareció en la release 5.6 del kernel de Linux.

El namespace time, permite que por cada namespace que tengamos, podamos crear desfases entre los relojes monotónicos (CLOCK_MONOTONIC) y de boot (CLOCK_BOOTTIME), de la máquina host. Esto permite que dentro de los contendores se nos permita cambiar la fecha y la hora, sin tener que modificar la hora del sistema host. Además, supone una capa más de seguridad, ya que no estamos vinculando directamente la hora a los relojes físicos de nuestro sistema.

Un namespace de tipo time, es muy similar al namespace de tipo PID en la manera de como lo creamos. Utilizamos el comando unshare -T, y mediante una systemcall se nos creará un nuevo time namespace, pero no lo asocia directamente con el proceso. Tenemos que utilizar setns para asociar un proceso a un namespace, además todos los procesos dependientes también tendrán asignado dicho namespace.

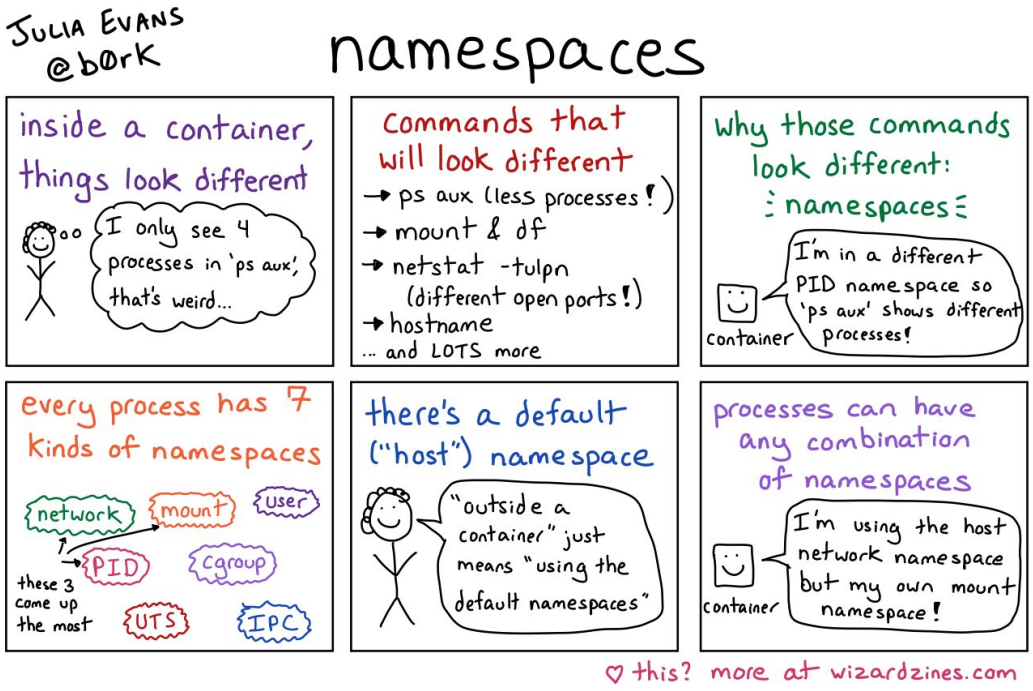


Figura 4: Como funcionan los contenedores. (2)

## 3.3 Ejemplo de uso de 'netns' usando comando `ip`

En este apartado, vamos a detallar un ejemplo de como funciona el comando IP manejando los 'network namespaces'.

Creamos los *network namespaces*, en este caso, con nombre h1 y h2. El sistema no crea directamente el namespace, lo que en realidad hace es definirlos en el sistema. El *network namespace* se crea cuando una aplicación se asocia a el.

```
$ ip netns add h1
$ ip netns add h2
```

Si utilizamos el comando `ip netns`, nos mostrará los netns existentes. Como es un sub-comando de `ip`, muestra los ns que el comando `lsns` no muestra.

Procedemos a asociar una aplicación a cada netns. Utilizamos `bash`.

```
$ ip netns exec h1 bash
$ ip netns exec h2 bash
```

Ahora, si que podemos utilizar el comando `lsns`. Comprobamos que si nos aparecen los ns que hemos creado, cosa que antes de asociar una aplicación al ns, no pasaba.

El comando `ip` crea automaticamente un *nsfs* para poder colocar los archivos de configuración del *netns*. Para ello, debe crearse el directorio `/etc/netns/h1` y poner en el los archivos de configuración de la red.

```
$ mkdir /etc/netns/h1
$ echo "nameserver 8.8.8.8" > /etc/netns/h1/resolv.conf
```

En este momento, tenemos el ns configurado con DNS. Nos quedaría realizar la conexión entre el ethernet físico de nuestro *host* y las interfaces de nuestros namespaces. Para ello, vamos a utilizar un conmutador virtual, en este caso Open vSwitch.

```
$ systemctl enable --now openvswitch.service
```

Creamos un *brige* utilizando el OpenvSwitch.

```
$ $ ovs-vsctl add-br s1
```

Utilizando el comando `ip`, creamos las interfaces virtuales de ethernet y las asignamos a sus namespaces.

```
$ ip link add h1-eth0 type veth peer name s1-eth1
$ ip link add h2-eth0 type veth peer name s1-eth2
$ ip link set h1-eth0 netns h1
$ ip link set h2-eth0 netns h2
```

Utilizando el comando `ovs-vsctl`, asignamos al *bridge* el otro par ethernet que hemos creado para cada namespace.

```
$ ovs-vsctl add-port s1 s1-eth1
$ ovs-vsctl add-port s1 s1-eth2
```

Verificamos que el controlador sea *standalone*, así el switch se comportará como un *learning-switch*.

```
$ ovs-vsctl set-fail-mode br0 standalone
```

Como la conexión es desde localhost al exterior, entendemos que es una conexión fuera de banda.

```
$ ovs-vsctl set controller br0 connection-mode=out-of-band
```

En este momento, tenemos todos configurado a falta de habilitar las diferentes interfaces de nuestra topología.

```
$ ip netns exec h1 ip link set h1-eth0 up
$ ip netns exec h1 ip link set lo up
$ ip netns exec h1 ip add add 10.0.0.1/24 dev h1-eth0
$ ip netns exec h2 ip link set h2-eth0 up
$ ip netns exec h2 ip link set lo up
$ ip netns exec h2 ip add add 10.0.0.2/24 dev h2-eth0
$ ip link set s1-eth1 up
$ ip link set s1-eth2 up
```

Ahora tenemos todas las interfaces configuradas, el switch activado y el sistema interconectado, por lo que podemos ejecutar un ping en una de las terminales de los namespaces para verificar la topología.

```
$ ip netns exec h1 ping -c4 10.0.0.2
```

Si queremos revertir todas las configuraciones que hemos hecho, lo que tenemos que hacer es ejecutar los siguientes comandos:

```
$ ovs-vsctl del-br s1
$ ip link delete s1-eth1
$ ip link delete s1-eth2
$ ip netns del h1
$ ip netns del h2
```

## 3.4 Ejemplo de uso de 'netns' usando comando `unshare`

En el ejemplo anterior, utilizabamos el comando `ip` para manejar los netns, sin embargo, eso nos limitaba los tipos namespaces que queriamos asignar a nuestro namespace. En contra partida a esto, el comando `unshare` nos da más libertad a la hora de crear los namespaces.

Utilizando `unshare`, no podemos ponerle un nombre, pero sí que nos permite asociarlo a un archivo, que montará con tipo bind. Esto nos permitirá utilizar en namespace aunque no haya ningún proceso corriendo en el, para ello podemos utilizar el comando `nsenter`.

```
$ touch /var/net-h1
$ touch /var/uts-h1
$ unshare --net=/var/net-h1 --uts=/var/uts-h1 /bin/bash
```

Utilizando el comando `nsenter` podemos ejecutar comandos dentro del namespace.

```
$ nsenter --net=/var/net-h1 --uts=/var/uts-h1 hostname h1
$ nsenter --net=/var/net-h1 --uts=/var/uts-h1 ip address
```

Para destruir el namespace, lo que tendremos que hacer es desmontar los archivos asignados a dicho namespace.

```
$ umount /var/net-h1
$ umount /var/uts-h1
```

Como será común necesitar más de un namespace, de ahora en adelante tendremos que utilizar los comandos `unshare` y `nsenter`.

# 4 Containers LXC

# 5 Docker

# 6 Evaluación de prestaciones

# 7 Interconexión física de diferentes red virtuales

# 8 Openflow OKO

# 9 Anexo 1. Instalación de `ansible` para automatizar una VM

En el caso de que queramos utilizar la herramienta ansible para configurar una VM, tendremos que seguir los siguientes pasos.

1. Asegurarnos de que tenemos instalado el programa en el host. La principal dependencia de `ansible` es `Python`.
   Para instalar `ansible` en diferentes distribuciones sería tal que:

   - Ubuntu 21.04:

     ```
     $ sudo apt install -y software-properties-common
     $ sudo add-apt-repository --yes --update ppa:ansible/ansible
     $ sudo apt update
     $ sudo apt install -y ansible
     ```

   - Fedora 32:

     ```
     $ sudo dnf update
     $ sudo dnf install ansible
     ```

   - OpenSUSE:

     ```
     $ zypper in ansible
     ```

   - Arch Linux:

     ```
     $ sudo pacman -S ansible
     ```

2. Creamos una clave SSH para utilizarla como método de autentificación con la maquina virtual.

   ```
   (host)$ ssh-keygen -t ed25519 -C "Host Ansible ssh key"
   ```

3. Utilizando nuestro gestor de máquinas virtuales, encendemos la VM. Es importante que nos aseguremos que tiene internet, por lo que configuramos que la interfaz de red sea de tipo *NAT*. Dentro de la máquina, buscaremos que ip tiene asignada con el comando `ip address`. Desde este momento, dejaremos la máquina virtual encendida.

4. Procedemos a copiar la clave pública SSH que hemos generado en el paso dos. Utilizamos el siguiente comando, sustituyendo con el usuario e IP específico de la máquina.

   ```
   (host)$ ssh-copy-id -i $HOME/.ssh/id_ed25519.pub <user>@<IP VM>
   ```

# 10 Glosario de términos

- Namespace. *Espacio de nombres*
- Linux. *Sistema operativo tipo UNIX, de codigo abierto, multiplataforma, multiusuario y multitarea*
- Kernel de linux. *Núcleo del sistema operativo Linux*
- PID. *Process Identifier*
- root. *Cuenta superusuario del sistema operativo Linux*
- veth-pair. *Virtual Ethernet Pair*
- UID. *Identificador de usuario*
- GID. *Identificador de grupo*
- ns. *Network namespace*

# 11 Bibliografía

## 11.1 Enlaces y referencias


1. *Namespaces*
2. Tutorial: Espacio de nombres en Linux
3. *Time namespaces coming to linux*
4. *Container is a lie. Namespaces*
5. *Namespaces.* Uso de *cgroups.*
6. *Introduction to Network Namespaces*
7. *Build a container by hand: the mount namespace*
8. Indentificador de procesos (*process id*)
9. *Linux PID namespaces work with containers*
10. *Network Namespaces*
11. *Introduction to Linux interfaces for virtual networking*
12. *Introducción a los grupos de control (cgroups) de Linux*
13. *Time namespaces*
14. *Network namespaces. Assign and configure*
15. Fundamentos de Docker

## 11.2 Images

1. Namespaces y API de acceso.
2. How containers work.