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

Autor: Enrique Fernández Sánchez
Tutor: José María Malgosa Sanahuja

Universidad Politécnica de Cartagena (UPCT)

4 de abril de 2022

Índice

Introducción

- *Abstract*: Guía ejemplificada para comprender los conceptos de virtualización ligera, contenedores y namespaces.

Objetivos del proyecto

- Aprender **conceptos básicos de NFV**.
- **Diferenciar entre virtualización** "dura" y "ligera".
- Estudiar soluciones, dentro del sistema operativo Linux, que permitan realizar soluciones NFV.
- **Definir namespaces**.
- Desgranar el **concepto de contenedor**.
- **Ejemplos de uso** de la virtualización "ligera" para la simulación de redes de telecomunicaciones.

Virtualización de funciones de red

- *Nacimiento de NFV*: *white paper* en octubre de 2012 (WG de la ITU, 13 operadoras)
- NFV como **solución para los problemas de las operadoras de red**

Problemas actuales de las operadoras de red

- Saturación general en la red.
- Dispositivos de red que requieren una instalación o intervención manual.
- Problemas de gestión de las operadoras de red (ej: reducción de costes).

Virtualización de funciones de red

- Comparación entre redes clásicas y redes virtualizadas:

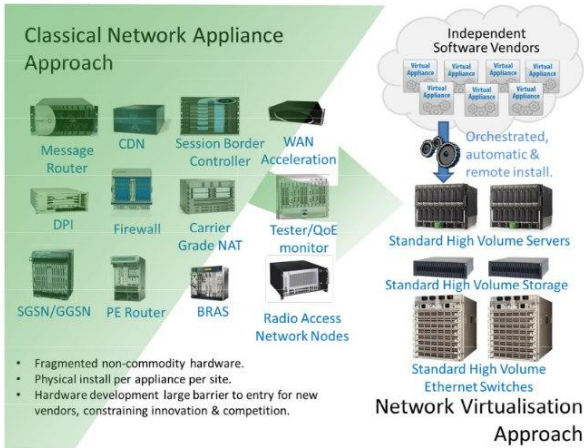
Classic networks

- Hardware embebido.
- Cada **dispositivo** de red asociado a una **única función** de red (firewall, balanceador de carga, router...)
- Software dependiente del "vendor" (cisco, jupyter...)
- Gestión de recursos software/hardware limitada.

Network Functions Virtualization (NFV)

- Hardware de carácter general.
- Un **mismo hardware** puede tener asociadas **más de una función** de red.
- Independencia del "vendor".
- Gestión de recursos software/hardware total.

Virtualización de funciones de red



Ventajas de NFV

- **Desacoplar funciones de red** de hardware específico.
- Acelerar desarrollo de servicios para los operadores de red.
- Reducción de costes.

Figura: Comparativa enfoque clásico contra enfoque virtualizado

SDN & NFV

Software Defined Networks (SDN)

- NFV depende de SDN, y viceversa.
- SDN desacopla el plano de control y el plano de datos de hardware de red.
- El **controlador SDN** es el encargado de gestionar el plano de control de los dispositivos.
- Un router solo sabe encaminar paquetes de la forma que el controlador SDN le ha programado.

APIs implicadas en SDN

Para programar bajo el paradigma SDN utilizamos:

- `Southbound`
- `Northbound`

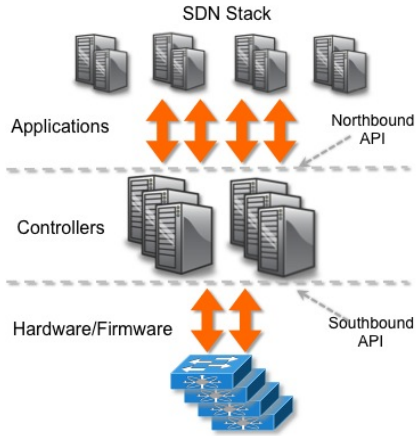


Figura: Esquema y ámbito de aplicacion de las APIs

Virtualización “ligera”

etsit escuela técnica superior de ingeniería de telecomunicación

Lightweight virtualization

- El propio `kernel` del SO realiza la virtualización.
- **Espacios** de usuarios **aislados** entre sí.
- Conocida como `containers`.

Comparativa virt. “dura”

- Eliminamos el *overhead*, al tener un único kernel.
- Menor consumo de recursos.
- Facilidad a la hora de manipular las instancias.

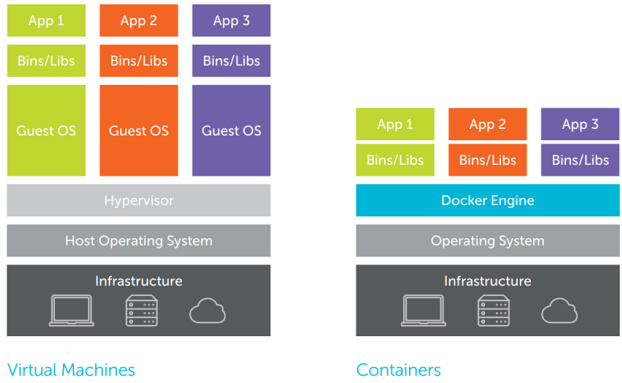


Figura: Comparativa entre virt. dura (izq.) y virt. ligera (der.)

Interfaces de red virtuales en Linux

- Linux permite una amplia **selección de interfaces virtuales de red**. Algunas de ellas son realmente interesantes **para la implementación de virtualización ligera**, ya que permiten comunicar las instancias aisladas.

Selección de interfaces de red virtuales en Linux

- Bridge
- MAC compartida
- VLAN 802.1q
- VLAN 802.1ad
- **Pares ethernet virtuales**
- **TUN/TAP**

Pares ethernet virtuales

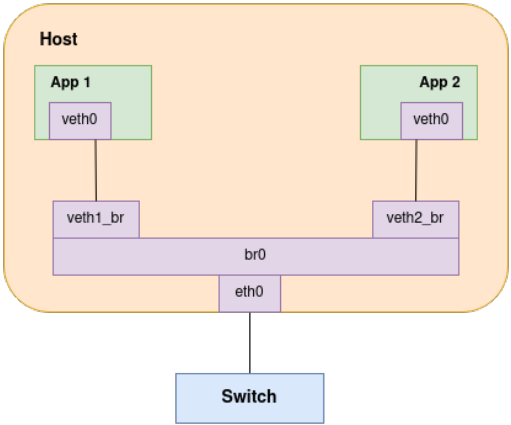


Figura: Diagrama de ejemplo de uso de pares ethernet virtuales

Ejemplo: topología usando veth

- **Dispositivos** virtuales **creados a pares**.
- Los paquetes transmitidos en un extremos se reciben automáticamente en el otro extremo.
- Útiles para asignar un extremo de la pareja a un **namespace**.
- `ip link add ... type veth peer ...`

TUN/TAP

Permiten conectar aplicaciones a través de un socket de red.

- TUN. Transporta paquetes IP
- TAP. Transporta *frames* Ethernet.

Crear interfaz TUN/TAP

- `tunctl`
- `ip tuntap`

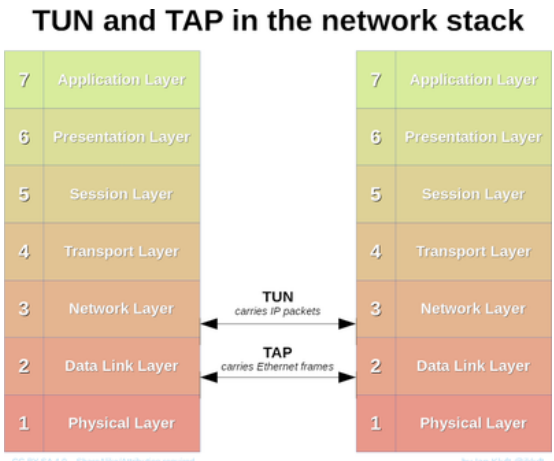


Figura: Comparativa en capa OSI de TUN/TAP

Ejemplos interfaces de red virtuales

etsit escuela técnica superior de ingeniería de telecomunicación

Ejemplos realizados en esta sección de la memoria

- Creación de cada una de las interfaces de red virtuales.
- Aplicación que crea una interfaz TUN/TAP.
- Aislamiento entre puertos usando interfaz TUN/TAP.

Namespaces

etsit escuela técnica superior de ingeniería de telecomunicación

Definición namespace

Característica del kernel de Linux que permite **gestionar** los recursos, **pudiendo limitarlos** a uno (o varios) **procesos**.

- *Objetivo*: adquirir una característica del sistema como una abstracción. Dentro del namespace tienen su propia **instancia aislada** del recurso global.

Crear/acceder namespace

- `unshare`
- `nsenter`

¿Cuantos namespaces hay?

1. UTS (`hostname`)
2. **Mount**
3. Process ID
4. **Network**
5. User ID
6. Interprocess Communication
7. **Control groups**
8. **Time**

Mount namespace

- Primer namespace en aparecer en el kernel.
- *Objetivo*: **restringir la visualización** de la jerarquía **global** de archivos. Cada namespace tiene su **propio "set"** de puntos de montaje.

Shared subtrees

Por defecto, los mount namespaces tienen activada la opción de `shared subtree`. Esta opción determina **cómo se van a propagar los montajes** a otros puntos de montaje.

- `shared mount`. Replicado, todas las copias iguales.
- `slave mount`. Montaje tipo esclavo, solo se propagan hacia él.
- `private mount`. No permite propagación.
- `unbindeable mount`. No permite ser asociado.

Mount namespace

- Un mount namespace nos **permite modificar un sistema de archivos** en concreto, **sin que otros procesos puedan ver y/o acceder** a dicho sistema de archivo.
- Permite tener **diferentes vistas para cada namespace**.

Montaje tipo bind

Opción de `mount` (`--bind`) que nos permite realizar un **montaje** de un dispositivo **sobre un directorio** de nuestro sistema de archivos.

Sistema de archivos temporal (`tmpfs`)

Funcionalidad que permite crear un **sistema de archivos temporal** dentro de nuestro sistema. El sistema creado se aloja en memoria.

Network namespace

- Permite **aislar la red de un proceso**.
- El namespace crea una interfaz virtual, con un **stack de red completo**.

Creación de network namespace

- Para que el network namespace sea persistente, lo tenemos que crear con: `ip netns add <nombre>` o `unshare --net=<file>`
- Si quisiéramos comprobar los network namespaces existentes, ejecutamos: `ip netns` o `lsns`
- Para tener conectividad con el exterior, tenemos que añadirle una interfaz virtual: `veth`, `ip link set <veth link> netns <nombre>`

Control groups & time

Control groups (`cgroups`)

- **Mecanismos de control** para los recursos del sistema.
- Funcionalidades que puede realizar:
  - Limitar recursos
  - Priorizar tareas
  - Monitorización
  - Control
- **Configurado utilizando sistema de archivos virtual**: `/sys/fs/cgroup`
- Disponibles dos versiones: cada vez más aceptada la `v2`

Time

- Último namespace añadido al kernel
- Permite **crear desfases entre los relojes del sistema**.
- Cambiar la fecha y hora dentro del namespace sin modificar la del host.

Ejemplos namespaces

etsit escuela técnica superior de ingeniería de telecomunicación

Ejemplos realizados en esta sección de la memoria

- Uso de UTS namespace
- **Persistencia de un namespace**
- Mount namespace con dispositivo físico
- **Casos de uso de montaje tipo bind**
- **Casos de uso de shared subtree**
- **Topología de red usando network namespaces y NAT**
- Uso de process ID namespace
- Uso de UID namespace
- **Limitar memoria RAM con cgroups**
- **Limitar uso de CPU con cgroups**
- Topología de red usando comando `ip`
- Topología de red usando comando `unshare`

Virtualización ligera y contenedores

etsit escuela técnica superior de ingeniería de telecomunicación

Definición virt. ligera

Tipo de virtualización que se realiza a **nivel de sistema operativo**, permitiendo la **coexistencia de diferentes espacios aislados** entre sí. Todas las instancias aisladas **utilizan el mismo kernel**.

Definición contenedor

Abstracción alto nivel de un sistema aislado creado utilizando `namespaces` y `cgroups`. Además, proporcionan unas **APIs de alto nivel** para interactuar con el contenedor. Algunos ejemplos de sistemas de contenedores son:

- `LXC`, LinuX Containers.
- `Docker`
- `Podman`

Contenedor DIY

- Fundación *Linux Foundation's Container Initiative* (`OCI`) recoge las directivas para crear `images` y `runtimes`.
- Diferentes `runtimes` en función de la solución de contenedores a utilizar.

`Images`

- Contiene un **sistema de archivos root** con las dependencias de la aplicación.
- Imágenes diferenciadas en capas, que serán montadas por el `runtime` usando *montajes de unión*. (`overlayfs`)
- Capa inferior inmutable y capa superior con todo lo referido a la aplicación.

`Runtimes`

- Dada una imagen, **`runtime` la ejecutará**.
- Creación y configuración de los namespaces (netns, pid, ipc, uts, etc).
- Configuración de cgroups para monitorización y limites de recursos.
- Montaje de las "capas" del sistema de archivos.

LXC

- Contenedores de **bajo nivel**. Uso de `mount namespaces` para conseguir una **estructura de directorios** propia de una **distribución de Linux**.
- Virtualización de una **distribución completa**.
- Permite crear contenedores sin privilegios, es decir, no pueden realizar acciones sobre el host.
- Apto para la ejecución de entornos gráficos.
- Galería de imágenes: `https://uk.lxd.images.canonical.com/`

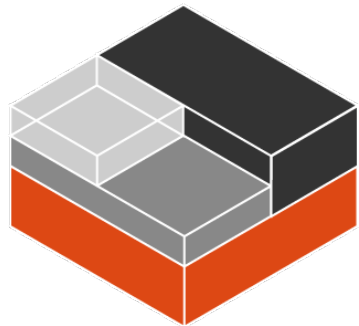


Figura: Logotipo `LXC`

Docker

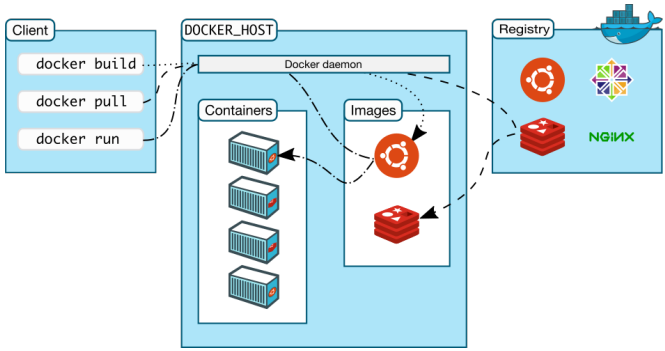


Figura: Arquitectura Docker

- Solución muy famosa de contenedores. Escrita en `Go`. Uso de `namespaces`.
- **Arquitectura cliente-servidor**. Cliente se comunica con "servicio" para crear, ejecutar y distribuir los contenedores.

Ejemplos virtualización ligera

etsit escuela técnica superior de ingeniería de telecomunicación

Ejemplos realizados sobre contenedores DIY

- Análisis de imagen basada en `alpine`.
- Análisis del `runtime` para ejecutar aplicación usando imagen basada en `alpine`.

Ejemplos realizados sobre LXC

- **Comprobar los namespaces de un contendor LXC en Ubuntu**

Ejemplos realizados sobre Docker

- Primeros pasos
- Crear un Dockerfile
- **Comprobar los namespaces asociados a un contenedor Docker**

Virtualización para simulación de redes

- *Objetivo*: evaluar y/o **simular situaciones** concretas de nuestras **redes**. Modelamos los nodos utilizando contenedores.

Opciones para la evaluación y/o simulación de redes

- `Shell script` para desplegar los namespaces necesarios para la topología. Uso de comandos `nsenter` y `unshare`.
- Desplegar e interconectar contenedores LXC. Cada contenedor con una función de red asociada.
- Utilizar un **simulador de redes basado en namespaces**. **Mininet** es un simulador que permite crear una red virtual realista utilizando namespaces. Permite un **sencillo uso** y un **despliegue rápido**.

Mininet

- Permite customizar y **desplegar topologías de red fácilmente**.
- Python API muy bien documentada.
- **Herramienta muy utilizada** en el campo de la investigación, sobre todo en el ámbito de **SDN**.
- Permite la integración de controladores SDN externos.

Ejemplos realizados con mininet

- Primeros pasos
- Topología simple sin mininet
- Topología simple con mininet
- Limitación de recursos, tanto en nodos como en enlaces
- Controlador SDN externo: `Ryu`

Containernet

- Fork de `mininet`.
- Permite añadir a la topología **contenedores Docker como hosts**.
- Cambios dinámicos a la topología (añadir, conectar y eliminar contenedores).
- Ejecución de comandos dentro de los contenedores usando Python API.
- Limitar uso de recursos de los contenedores.

Ejemplos realizados con containernet

- Instalación y primeros pasos
- Ejecución topología simple (dos contenedores, dos switchs y un controlador)

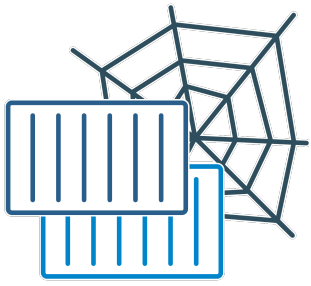


Figura: Logotipo `containernet`

Conclusiones

- Alcanzados todos los objetivos propuestos
- Se ha profundizado en el **concepto** de **contenedor** y **namespaces**
- Se han realizado **28 ejemplos** sobre técnicas de **virt. ligera**
- Se han aportado dos soluciones para la evaluación de redes de comunicaciones

Propuestas futuras

- Profundizar en las diferencias de rendimiento entre una virt. "ligera" y virt. "dura".
- Investigar sobre modificaciones del protocolo OpenFlow. Además, indagar sobre **DPDK** y sus ventajas.
- Pruebas de concepto del **lenguaje de programación P4**.

Bibliografía

- La contenida en la memoria del proyecto: páginas 100 - 102

Muchas gracias por su atención

¿Preguntas?

Enlace al proyecto:
`https://github.com/Raniita/lightweight-virtualization-nfv`