

Trabajo Fin de Grado

Título del trabajo: Estudio para un despliegue
automático de red inalámbrica virtualizada

English title: Study for an automatic deployment
of virtualized wireless network

Autor/es

Carmen Arasanz Jordán

Director/es

María Canales Compés
Julián Fernández Navajas

Titulación del autor

Ingeniería de Tecnologías y Servicios de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

Estudio para un despliegue automático de red inalámbrica virtualizada

Resumen

Actualmente, las redes inalámbricas se encuentran en un periodo de gran expansión, debido a su bajo coste, su facilidad en el despliegue y a la libertad de movimiento que proporcionan. Otro factor muy importante ha sido la consolidación del estándar IEEE 802.11 como tecnología inalámbrica de mayor uso en interiores.

Desde su lanzamiento en 1997, los sucesivos estándares de Wi-Fi se han convertido en una de las principales tecnologías de comunicaciones inalámbricas, y se emplea en todos los ámbitos, tanto domésticos como públicos y empresariales. Para su integración como uno de los accesos en escenarios heterogéneos de redes 5G y 6G, el crecimiento exponencial del uso de redes móviles ha provocado la necesidad de implementaciones de redes escalables y manejables en las tecnologías inalámbricas actuales. Esta demanda crea nuevos requisitos para la arquitectura de la red, como la flexibilidad en la gestión y configuración y la adaptabilidad.

Para cumplir con estos requisitos, se ha propuesto una red inalámbrica definida por software (SDWN) como una solución rentable. SDWN desacopla el plano de datos del plano de control, lo que permite que ambos planos puedan ser diseñados, dimensionados y optimizados de forma independiente.

A su vez es necesario compartir los recursos de un servidor y optimizar el uso de sus procesadores. El concepto de virtualización crea un entorno informático virtual, en lugar de un entorno físico. Esto permite a las organizaciones particionar un equipo o servidor físico en varios sistemas virtuales. Cada sistema puede interactuar de forma independiente y ejecutar sistemas operativos o aplicaciones diferentes mientras comparten los recursos de una sola máquina host. En este documento analizamos las ventajas y desventajas del uso de máquinas virtuales y contenedores como métodos de virtualización, para aplicar el más conveniente a nuestra solución final.

La automatización permite que la red complete una sola tarea o función sin intervención humana. Mientras que la orquestación define la serie de pasos o tareas necesarias para desplegar nuestra infraestructura de red, para luego dar soporte a servicios. Juntos aseguran que los procesos específicos, como la asignación informática o el aprovisionamiento de recursos informáticos, se realicen cada vez de manera consistente con una mínima intervención humana. En este trabajo de fin de grado también estudiamos las herramientas existentes de orquestación y adoptamos la elegida en nuestra solución.

Índice de Contenidos

1. Introducción	8
1.1. Motivación	8
1.2. Objetivos	10
2. Planteamiento del problema	11
2.1. SDWN	11
2.2. Escenario de partida	12
2.2.1. Plano de control	14
2.2.2. Plano de datos	14
2.2.3. Plano de gestión	15
2.3. Solución virtualizada	16
3. Virtualización	17
3.1. Máquina virtual	17
3.2. Hipervisor	17
3.3. Contenedores	18
4. Herramientas	20
4.1. GNS3	20
4.2. Docker	20
4.2.1. Imagen Docker	21
4.2.1.1. Imagen padre	21
4.2.2. Contenedor	22
4.2.2.1. Modelo de red	23
4.2.2.2. Aislamiento de contenedor Docker	24
4.3. Kubernetes	25
4.3.1. Componentes de Kubernetes	25
4.3.1.1. Nodo master	25
4.3.1.2. Nodo worker	26
4.3.2. Modelo de red de Kubernetes	27
4.4. K3s	27
4.4.1. Flannel	28
5. Evaluación de la virtualización	30
5.1. Antenas	30
5.2. Procesado de los datos	31
5.3. Creación de contenedor	32
5.4. Resultados	33
6. Despliegue automatizado del escenario Wi-Fi	36

6.1. Prueba preliminar de virtualización en Docker	36
6.2. Elementos virtualizados en K3s	41
6.3. Estudio de conectividad	42
6.4. Despliegue de puntos de acceso	48
7. Conclusiones y Líneas Futuras	50
7.1. Conclusiones	50
7.2. Líneas futuras	51
Anexos	53
Anexo 1. Conexión remota a la VPN del laboratorio	53
Anexo 2: Despliegue inicial en GNS3	55
Anexo 3: Graphic Network Simulator-3	56
Anexo 4: Instalación Docker	58
Anexo 5: Repositorio Docker Hub	60
Anexo 6: Código de Python y Matlab utilizado en la evaluación de virtualización	61
Anexo 7: Creación de imágenes Docker	63
Anexo 8: Script punto de acceso	67
Anexo 9: Instalación de K3s	68

Glosario de términos

AP. Access Point

ARP. Address Resolution Protocol

BSSID. Basic Service Set Identifier

CNI. Container Network Interface

CPU. Central Processing Unit

CSA. Channel Switch Announcement

DHCP. Dynamic Host Configuration Protocol),

GNS3. Graphic Network Simulator-3

HostAPD. Host Access Point Daemon

LVAP. Light Virtual Access Point

MAC. Media Access Control

NAT. Network Address Translation

NFV. Network Function Virtualization

SDN. Software Defined Networking

SDWN. Software Defined Wireless Networking

SO. Sistema Operativo

UDP. User Datagram Protocol

VM. Virtual Machine

VMM. Virtual Machine Monitor

VXLAN. Virtual Extensible LAN

1. Introducción

La virtualización es una tecnología que permite crear servicios de tecnología de la información útiles, con recursos que tradicionalmente están limitados al hardware. Esta tecnología distribuye las funciones de una máquina física entre varios usuarios o entornos, posibilitando el uso de toda la capacidad de la máquina host.

Existen distintos tipos de virtualización como la virtualización de los datos, de escritorios, de servidores, de los sistemas operativos y de las funciones de red. En este proyecto trabajamos con dos servidores físicos remotos y nos centramos en la virtualización de los sistemas operativos y de las funciones de red.

Los servidores físicos tienen el sistema operativo (SO) Ubuntu y los sistemas operativos que virtualizamos tienen el SO Debian. Para la virtualización de estos sistemas exploramos el uso de máquinas virtuales y de contenedores, adoptando finalmente la que mejor se adecúe a nuestro propósito.

La integración de esta tecnología nos proporciona numerosas ventajas como la reducción del coste de mantenimiento de hardware, el aumento de la eficacia al aprovechar mejor los recursos físicos, la facilidad de implementación sin la necesidad de aumentar el equipo físico y el ahorro de tiempo, gracias a la facilidad de creación de entornos virtuales y la posibilidad de guardarlos para futuras ocasiones.

A su vez diseñamos una red inalámbrica definida por software (SDWN), lo que nos permite separar las funciones de envío de datos de las funciones de control, lo que permite crear una red virtual que puede ser programada y gestionada de forma centralizada.

Por último, el estudio de la orquestación del despliegue de la infraestructura nos permitirá automatizar el despliegue.

1.1. Motivación

El incremento del uso de smartphones y el efecto multiplicador de los dispositivos de vídeo ha tenido como consecuencia el aumento del tráfico de datos enviados a través de redes inalámbricas. La Figura 1 muestra la estimación de la evolución del tráfico IP global según el dispositivo utilizado, publicada por Cisco en 2019. En ella los smartphones representan un 23% en 2017 frente a un 50% en 2022.

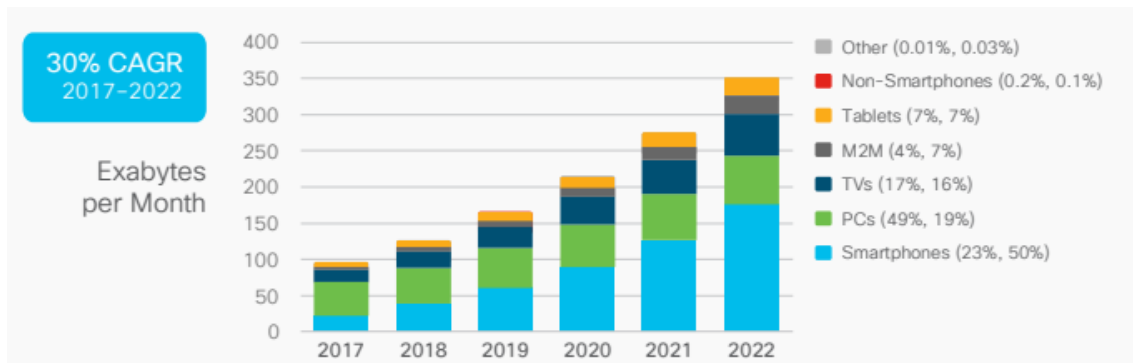


Figura 1: Previsiones de crecimiento de tráfico IP según dispositivo [1].

En la Figura 2 observamos que el porcentaje de generación de tráfico de vídeo en 2022 apunta a un 79%, suponiendo casi cuatro quintas partes del tráfico de datos móviles.

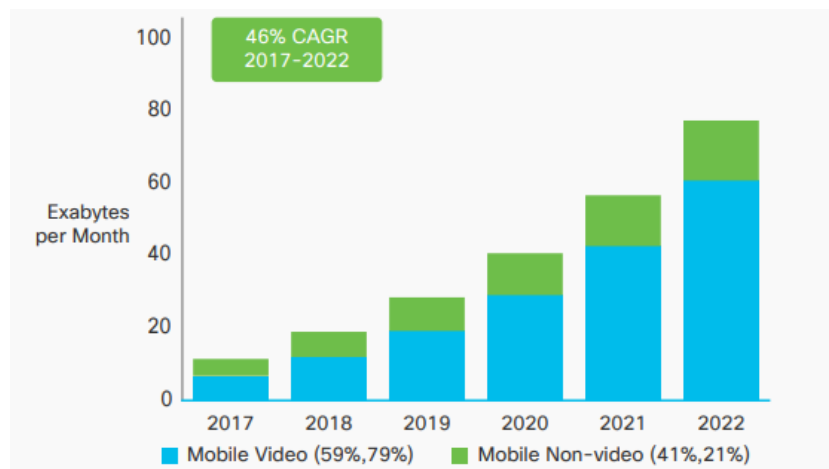


Figura 2: Previsiones de crecimiento de tráfico de datos de vídeo [2].

La Figura 3 muestra que tanto la tecnología Wi-Fi como las redes móviles tienen un mayor crecimiento que las redes fijas (Ethernet). El tráfico fijo cae del 48% del tráfico IP total en 2017 al 29% en 2022. El tráfico móvil y la descarga desde dispositivos móviles representa el 48% del tráfico IP total en 2022, lo que demuestra el importante crecimiento y el impacto de los dispositivos móviles en el tráfico global. Por último, el tráfico Wi-Fi procedente tanto de dispositivos móviles como de dispositivos sólo Wi-Fi representa en conjunto más de la mitad (51%) del tráfico IP total en 2022, frente al 43% en 2017.

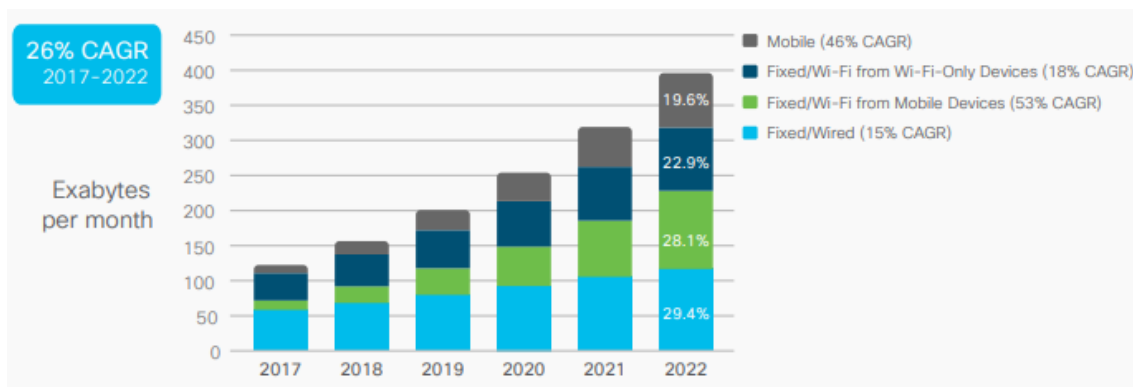


Figura 3: Previsiones de crecimiento tecnología Wi-Fi [1].

Debido al crecimiento del tráfico de datos y a la consolidación y el crecimiento de las redes inalámbricas es necesario el despliegue de un mayor número de puntos de acceso para poder dar conectividad a los usuarios. En consecuencia, nace la necesidad de despliegue de redes robustas, flexibles y escalables.

En este trabajo estudiamos cómo desplegar la red inalámbrica de forma eficiente haciendo uso de la tecnología de virtualización de sistemas operativos y de funciones de red, de la SDN para Wi-Fi (SDWN) y finalmente de la orquestación del escenario a desplegar.

1.2. Objetivos

El objetivo de este proyecto es el estudio para un despliegue automático de red inalámbrica virtualizada haciendo uso de la tecnología de virtualización, de la SDWN y el estudio de la orquestación del mismo despliegue.

Partimos de una red SDWN previamente implementada (Anexo 2) con un simulador gráfico de red y la estudiamos para después desplegarla sin utilizar dicho simulador. El objetivo de este trabajo es la selección del método de virtualización más adecuado para el despliegue de dicha red y el estudio de la automatización de su despliegue.

2. Planteamiento del problema

2.1. SDWN

SDN significa Redes Definidas por Software y es un enfoque de arquitectura de red que separa el plano de control de la red del plano de datos. En una arquitectura de red tradicional, el plano de control y el plano de datos están integrados en los dispositivos de red, como switch y router. Sin embargo, en una arquitectura SDN, el plano de control se implementa en software y se ejecuta en un controlador centralizado, que puede programar dinámicamente el comportamiento de los dispositivos de red [3].

Gracias a la centralización del control en un equipo, la infraestructura de la red subyacente queda abstraída de las aplicaciones [4].

SDN permite a los administradores de red administrar la red de manera más flexible y dinámica, lo que les permite configurar, monitorizar y administrar los recursos y flujos de tráfico de la red con mayor facilidad. Proporciona un alto nivel de programabilidad y automatización, lo que facilita la adaptación a las necesidades cambiantes y los requisitos de la red.

Una SDN presenta tres planos: el plano de control, que se encuentra centralizado en el controlador, el de datos, conformado por los equipos encargados de la transmisión de los flujos de información y el plano de gestión, que proporciona al administrador de la red un mecanismo de monitorización, análisis y despliegue de la red [5].

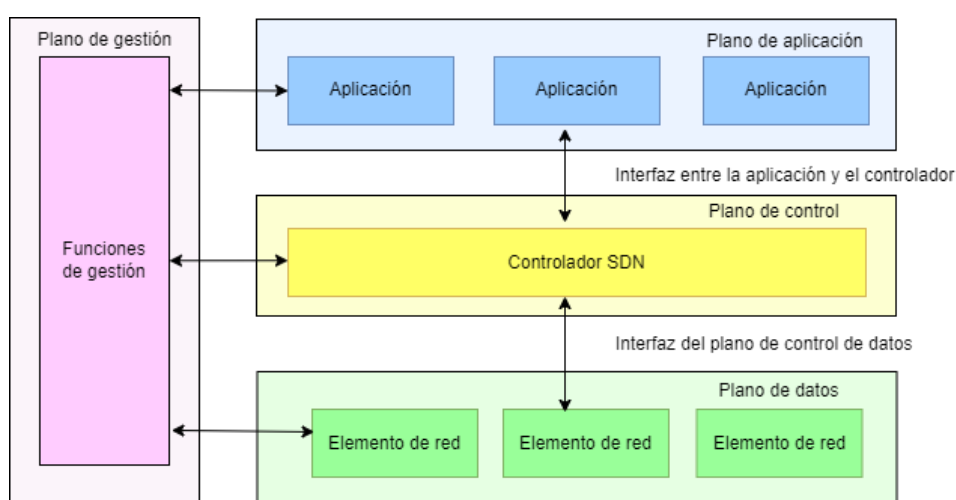


Figura 4: Componentes de SDN.

Algunas de las funciones clave de un controlador de red SDN incluyen:

- **Programación de la red:** configuración de la lógica de enrutamiento y la política de seguridad para la red.

- **Monitorización de la red:** recopilación de información sobre el tráfico de red y el estado de los dispositivos de red.
- **Control de la red:** gestión de la configuración y el comportamiento de los dispositivos de red, como la asignación de ancho de banda y el balanceo de carga.

El controlador de red SDN proporciona una vista global y unificada de la red, lo que permite a los administradores de red controlar y administrar la red de manera más eficiente y con menos errores. Además, la programabilidad del controlador de red SDN permite una mayor flexibilidad y capacidad de adaptación a las necesidades cambiantes de la red y de los usuarios de la red.

Algunos de los beneficios de SDN incluyen una mejora en el rendimiento de la red, una simplificación en la administración de la red y una reducción en los costos operativos.

A partir de SDN, se desarrolló el concepto de Redes Inalámbricas Definidas por Software (SDWN), como una aplicación específica de SDN a las redes inalámbricas, que presentan desafíos y requisitos específicos, como la gestión de la movilidad de dispositivos, la gestión del espectro inalámbrico y la mitigación de interferencias.

Mientras que SDN se centra en la centralización y automatización de la gestión de red, SDWN se enfoca, también, en la gestión de la infraestructura inalámbrica, incluyendo la configuración, asignación de recursos, movilidad de los dispositivos y aplicaciones de políticas de seguridad específicas para redes inalámbricas [6].

Las ventajas del uso de una SDWN son variadas: permite el control de la red de acceso Wi-Fi, mejora el rendimiento en los procesos de asociación entre dispositivos y puntos de acceso, reduce el impacto negativo de los handoff y gestiona el ancho de banda disponible de una manera más eficiente [5].

2.2. Escenario de partida

En este trabajo partimos de la red SDWN presentada en la Figura 5 [3] [6], previamente desplegada en el entorno de Graphic Network Simulator-3 (Anexos 2 y 3). Esta red está compuesta por los planos de gestión, de control y de datos, que detallaremos más adelante. A su vez incluye dos puntos de acceso que alojan múltiples LVAP (Light Virtual Access Point), cada uno asociado a un único terminal inalámbrico. Desplegamos dos puntos de acceso para ser capaces de realizar un handoff o handover. Ambos términos hacen referencia a la transferencia de una conexión de un punto de acceso a otro.

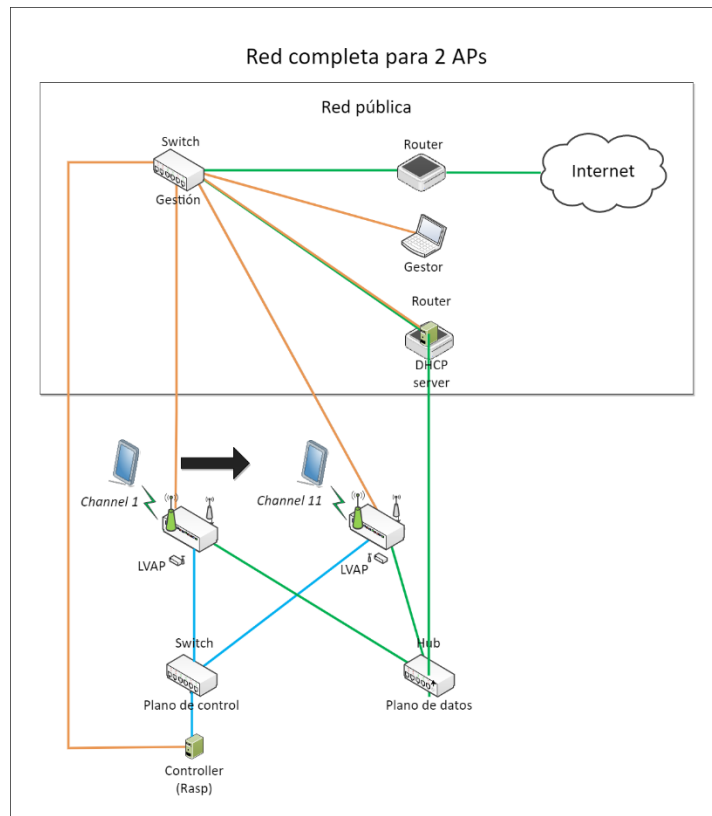


Figura 5: Red completa para 2 APs.

Wi-Fi debe dar solución al enorme crecimiento del número de dispositivos con capacidad de conexión debido, por ejemplo, a la expansión de Internet of Things (IoT) o a la demanda de mayores tasas de transmisión por parte de los servicios multimedia en tiempo real.

Estas circunstancias han empujado a los operadores hacia el desarrollo de redes con un mayor número de puntos de acceso especializados en la prestación de servicios determinados. Esta solución pasa por la implementación de LVAP que puedan ser orquestados, encendidos y apagados, en función de la demanda [5]. A su vez pueden ser administrados centralmente por el controlador, lo que facilita la expansión de la red inalámbrica sin tener que agregar puntos de acceso físicos adicionales.

Cuando un dispositivo se asocia por primera vez con un punto de acceso físico, el controlador crea un LVAP para él. Este LVAP contiene parámetros de comunicación exclusivos para el terminal asociado. Cada punto de acceso tendrá un LVAP para cada dispositivo asociado a él. El LVAP puede ser traspasado entre distintos puntos de acceso físicos. De esta forma, aunque el terminal efectúe desplazamientos, siempre verá el mismo punto de acceso, lo que evita la necesidad de asociación a un nuevo punto de acceso físico [6].

2.2.1. Plano de control

Nuestro plano de control está constituido por dos controladores y los LVAP. En nuestro caso, los dos controladores funcionan en la misma máquina física, que en este caso es la Raspberry pi 3, pero escuchan en puertos distintos y tienen diferentes funciones.

El primero de los controladores que se presenta es Odin Controller, cuya función es monitorizar el rendimiento de los APs físicos, crear los LVAPs y ajustar dinámicamente su configuración para optimizar el rendimiento y la calidad del servicio inalámbrico. Este controlador escucha en el puerto 6777.

El segundo controlador de red SDN utilizado es el Open Controller, y se comunica con los dispositivos de red mediante el protocolo estándar OpenFlow. Su funcionalidad es proporcionar aquellas funciones no específicas de Wi-Fi, y que no son controladas por Odin: como el reenvío de tramas ARP (Address Resolution Protocol) o la utilización del protocolo DHCP. Este controlador escucha en el puerto 6633.

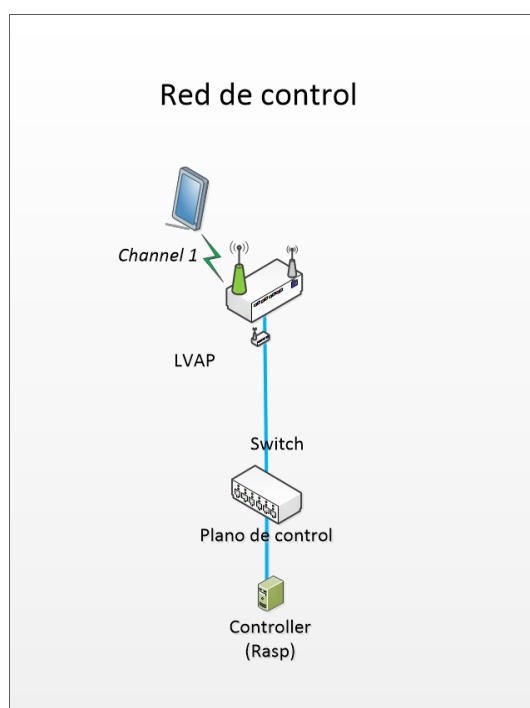


Figura 6: Plano de control.

2.2.2. Plano de datos

En una arquitectura de redes definidas por software (SDN), el plano de datos es la capa que maneja la transferencia de datos a través de la red. El plano de datos se refiere a la parte de la red que está encargada del reenvío y la entrega de los paquetes de datos desde un origen a un destino en la red.

El direccionamiento IP de esta red de datos es privado, por lo que la información de la red que sale hacia Internet a través de un router, debe pasar por un NAT (Network Address Translation). Al mismo tiempo este router aloja un servidor DHCP (Dynamic Host Configuration Protocol), encargado de la asignación de direcciones IP privadas a los terminales Wi-Fi.

En cada punto de acceso que desplegamos en nuestra red inalámbrica usamos dos antenas Wi-Fi. Una de ellas para la transmisión y recepción de los datos, y la otra para la monitorización de los canales Wi-Fi.

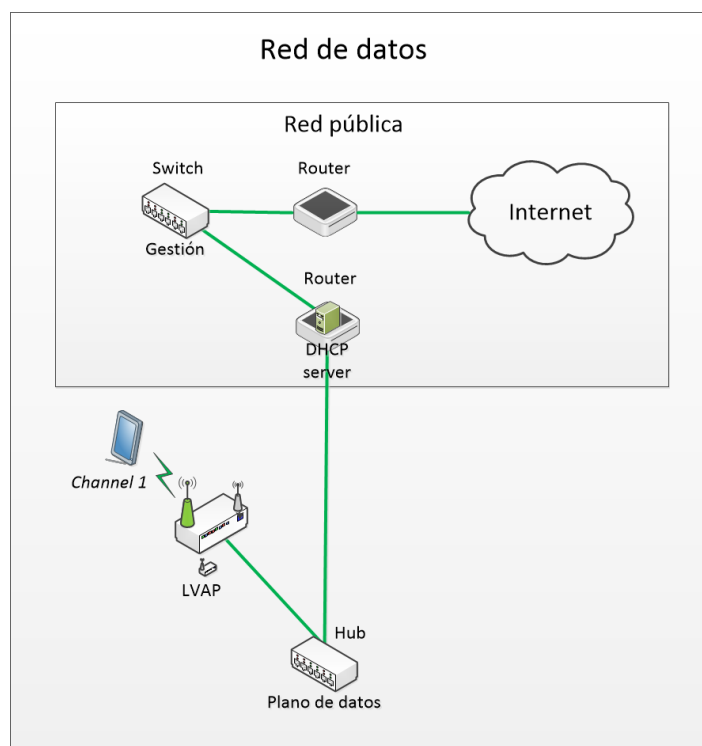


Figura 7: Plano de datos.

2.2.3. Plano de gestión

La red de gestión se utiliza para administrar y configurar los componentes de la red, como servidores, router, switch y otros dispositivos.

En nuestro caso el gestor está ubicado en la red pública del laboratorio y con acceso remoto gestiona el controlador, los AP y el router con NAT y servidor DHCP.

La red de gestión permite la configuración de la red y que sus dispositivos ofrezcan sus servicios con total disponibilidad.

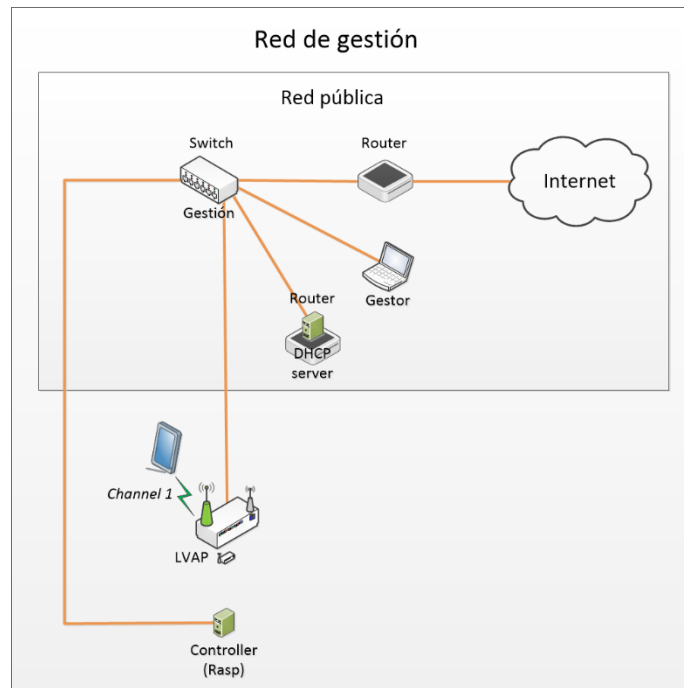


Figura 8: Plano de gestión.

2.3. Solución virtualizada

Trabajando sobre el escenario previo se quiere automatizar el proceso de despliegue y aprovechar las ventajas de la virtualización (que explicaremos con detalle en el próximo capítulo). Para ello, en los próximos capítulos detallaremos las herramientas utilizadas, el método de virtualización escogido y las pruebas realizadas.

3. Virtualización

3.1. Máquina virtual

Una máquina virtual es un software que emula a un ordenador y puede ejecutar programas como si fuese un ordenador real.

Las máquinas virtuales se pueden clasificar en dos grandes categorías según su funcionalidad y su grado de equivalencia a una verdadera máquina:

- **Máquinas virtuales de sistema:** permiten a la máquina física subyacente dividirse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo. El software que permite la virtualización es el hipervisor. Algunos ejemplos son VirtualBox, VMWare, QEMU, etc.
- **Máquinas virtuales de proceso:** se ejecuta como un proceso normal dentro de un sistema operativo y soporta un solo proceso. La máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se para cuando éste finaliza. El ejemplo más conocido actualmente de este tipo de máquina virtual es la máquina virtual de Java.

El tipo de máquina virtual que utilizamos en este proyecto es el de máquina virtual de sistema porque nos centramos en la virtualización de un punto de acceso y necesitamos distribuir los recursos físicos de un host (servidor remoto con tarjetas Wi-Fi) entre varias máquinas virtuales. A continuación, presentaremos el concepto de hipervisor, necesario en el tipo de máquina virtual que usamos.

3.2. Hipervisor

Introducimos el concepto de hipervisor, conocido también como monitor de máquina virtual (VMM), que es un software que crea y ejecuta máquinas virtuales (VM) dentro de un único servidor.

El hipervisor proporciona a cada máquina virtual los recursos que se le habían asignado, y gestiona la programación de ellos en función de los recursos físicos. Con un hipervisor, muchos sistemas operativos diferentes pueden funcionar a la par y compartir los mismos recursos de hardware virtualizados. Esta es una de las principales ventajas de la virtualización, ya que, sin ella, sólo se puede ejecutar un sistema operativo en el hardware.

El servidor en el que el hipervisor ejecuta una o más máquinas virtuales es conocido como “host machine”, mientras que cada VM se conoce como “guest machine”.

Existen dos tipos de hipervisores:

- **Tipo 1 (conocido como hipervisor nativo):** se ejecuta directamente en el hardware del host y gestiona los sistemas operativos guest. Ejemplos: VMware: ESX / vSphere Hypervisor , Microsoft Hyper-V Server o KVM.
- **Tipo 2 (conocido como hipervisor alojado):** se ejecuta como una capa de software superior al sistema operativo de la host machine. Se utilizan para abstraer los sistemas operativos guest del sistema operativo host. Ejemplos: VirtualBox, VMWare, ProxmoxVE o QEMU.

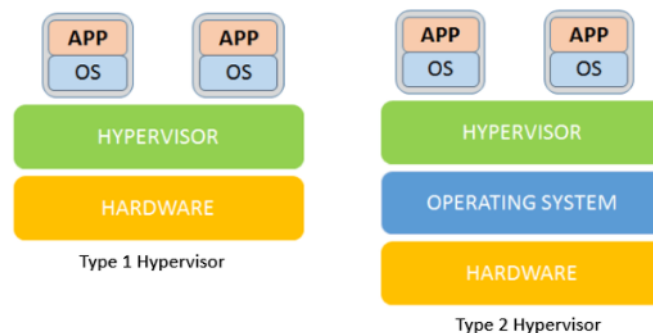


Figura 9: Tipos de hipervisores [7].

En este proyecto utilizamos el supervisor QEMU, que es de tipo 2, desde línea de comandos y ejecutamos la máquina virtual con sistema operativo Debian sobre una máquina real con S.O. Ubuntu.

La herramienta GNS3, que usaremos como sistema de gestión y que presentamos más adelante, utiliza QEMU junto a KVM, para aprovechar la virtualización asistida por hardware y mejorar el rendimiento de las máquinas virtuales.

3.3. Contenedores

El problema de los hipervisores y las máquinas virtuales es que cada máquina virtual es independiente de las demás. Al no reutilizarse ningún componente, se ocupa mucho espacio tanto en disco como en memoria y el tiempo de ejecución siempre será mayor que si sólo hubiera un sistema operativo.

Para resolver este problema se crearon los contenedores en los que se utilizan mecanismos existentes en el sistema operativo para aislar las aplicaciones, pero compartiendo el mayor número posible de componentes del sistema operativo o incluso de las aplicaciones.

La ventaja de los contenedores es que no tenemos que emular un sistema operativo completo y que comparten el máximo de componentes con el sistema operativo host.

Los contenedores suelen ser elementos efímeros. La facilidad con la que pueden crearse y ponerse en marcha hace que sea más fácil crear un nuevo contenedor que

modificar uno ya existente. Por ello, los datos generados por las aplicaciones no se suelen guardar en los contenedores, sino fuera de ellos. Su ligereza hace más fácil tener varios contenedores con una aplicación en cada uno de ellos que tener un único contenedor con varias aplicaciones en él. Por ello, un aspecto importante de los contenedores es su orquestación, es decir, la administración simultánea de muchos contenedores, incluyendo la comunicación entre ellos.

Los contenedores no necesitan un sistema operativo para funcionar y aprovechan el kernel del sistema operativo del host para ejecutarse.

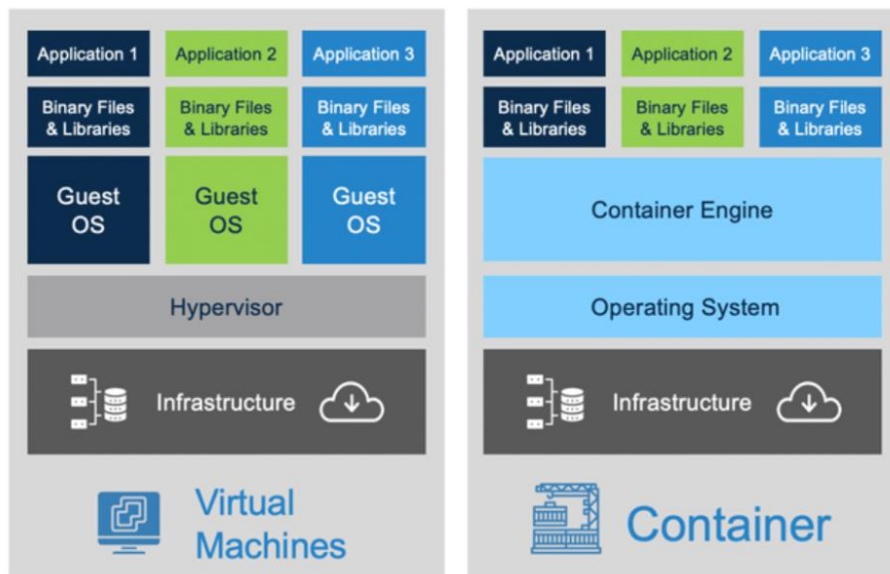


Figura 10: Estructura máquina virtual y contenedor [8].

En la siguiente tabla presentamos las principales diferencias entre los dos métodos de virtualización anteriormente introducidos:

Máquina virtual	Contenedor
Aislamiento completo: cada máquina virtual tiene un sistema operativo independiente	Aislamiento imperfecto: el contenedor comparte el sistema operativo de la máquina host
Proporciona un aislamiento completo del sistema operativo host y otras máquinas virtuales.	Por lo general, proporciona un aislamiento ligero del host y otros contenedores.
Las máquinas virtuales son de pocos GB	Los contenedores son ligeros (KB/MB)
Comienza en minutos	Comienza en segundos
Admite escalado, pero alcanzará los límites de recursos informáticos rápidamente	Fácil de escalar con velocidad y tamaño pequeño
Mayor seguridad al haber total aislamiento	Menor seguridad porque si el sistema operativo subyacente o el software en el contenedor es vulnerable a una amenaza, la seguridad de todo el sistema se ve comprometida

Tabla 1: Comparativa máquina virtual frente a contenedor.

4. Herramientas

4.1. GNS3

GNS3 (Graphic Network Simulator-3) es un simulador gráfico de red lanzado en 2008, que permite diseñar topologías de red complejas y poner en marcha simulaciones sobre ellas, permitiendo la combinación de dispositivos reales y virtuales. Es un software gratuito de código abierto.

Esta herramienta es importante para nuestro proyecto porque con ella fue desarrollado el escenario que nosotros hemos utilizado como punto de partida y que ilustramos en el Anexo 2. Iniciamos nuestro proyecto estudiando dicho despliegue y entendiendo esta herramienta, para posteriormente realizar los cambios convenientes con el fin de desarrollar nuestra propia solución. Detallamos la herramienta en el Anexo 3.

4.2. Docker

Docker es un software libre y de código abierto lanzado en 2013 y que permite a los desarrolladores crear, implementar, ejecutar, actualizar y administrar contenedores. Los contenedores virtuales de Docker se pueden ejecutar en ordenadores Linux, Windows o MacOS.

Según la encuesta de desarrolladores de Stack Overflow 2022 [9] Docker es la herramienta de desarrollo número uno más querida y también la más buscada.

En las siguientes gráficas correspondientes a esta encuesta, 59164 desarrolladores fueron encuestados. En primer lugar, se preguntó qué herramienta de desarrollo habían usado durante el último año, siendo Docker la más utilizada:

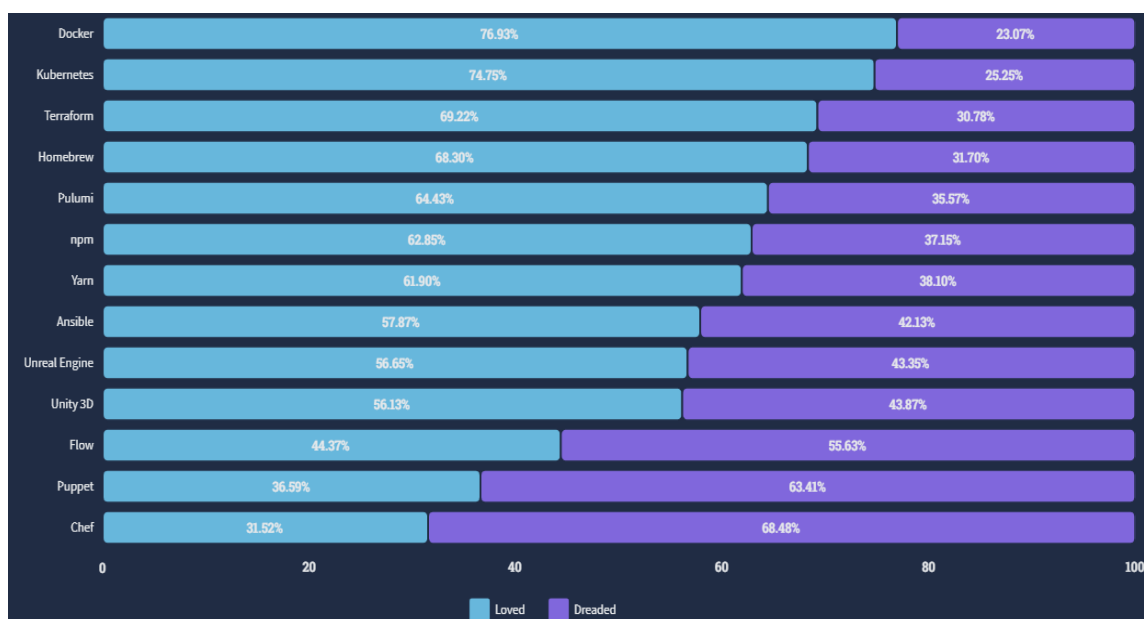


Figura 11: Encuesta uso de Docker en 2022.

A continuación, se preguntó en qué nueva herramienta tenían interés en desarrollar en el próximo año. Los resultados prevén que Docker será la que mayor desarrollo tenga en 2023:

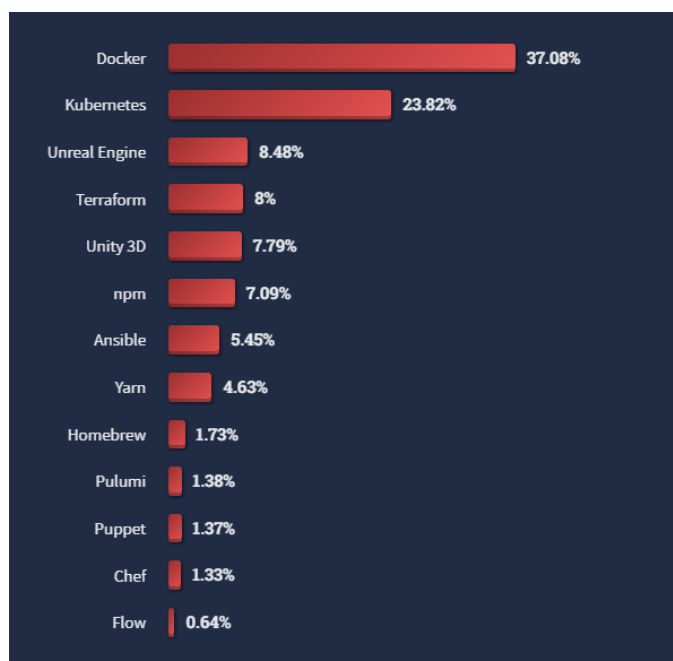


Figura 12: Encuesta evolución de Docker en 2023.

La creciente expansión de Docker y el hecho de que es una herramienta de código abierto han sido los motivos de nuestra decisión de elegirla para implementar nuestras funciones en contenedores.

4.2.1. Imagen Docker

Una imagen en Docker es un archivo compuesto de diversas capas y que se utiliza con el objetivo de ejecutar un código dentro de un contenedor de Docker. Las imágenes se utilizan para crear contenedores ya que éstas nunca van a cambiar, esto permite crear contenedores con diferentes capas de imágenes, las cuales se van a superponer sobre otras.

Normalmente, cuando creamos imágenes, partimos de una imagen padre sobre la que aplicamos adiciones y modificaciones. En nuestro caso partimos de una imagen Debian 10 a la que le añadiremos lo necesario para implementar el software de nuestra solución (puntos de acceso, controladores...).

4.2.1.1. Imagen padre

Docker Hub es un repositorio público de imágenes Docker, donde podemos encontrar multitud de imágenes gratuitas en repositorios públicos y privados, que se pueden descargar y usar. En el Anexo 5 explicamos cómo registrarnos en este repositorio.

En nuestro caso buscamos una imagen Debian y entre las imágenes disponibles elegimos la imagen Debian 10, que utilizaremos como imagen padre. La descargamos de este repositorio haciendo un *pull* (descarga) de dicha imagen.

Hay numerosos sistemas operativos de código abierto y basados en Linux entre los que podríamos haber elegido para desarrollar este proyecto (Ubuntu, Debian, Fedora, Arch Linux, openSUSE, etc). Sin embargo, escogimos desarrollar el proyecto en Debian porque es un sistema operativo libre, es decir, no pertenece a ninguna empresa (a diferencia de Ubuntu, Fedora u openSUSE) y es la distribución de Linux más genérica, basándose muchos sistemas operativos en ella.

4.2.2. Contenedor

Un contenedor de Docker es una unidad de software que encapsula una aplicación y sus dependencias en un entorno aislado y portátil. Los contenedores de Docker se utilizan para empaquetar aplicaciones y servicios de forma que se puedan mover entre diferentes entornos de ejecución, como un entorno de desarrollo, un entorno de pruebas o un entorno de producción, de manera rápida y sencilla.

Los contenedores de Docker se crean a partir de imágenes de contenedores, que contienen el código de la aplicación y sus dependencias. Las imágenes de contenedores se pueden compartir y distribuir entre diferentes entornos de ejecución, lo que hace que el proceso de desarrollo y despliegue de aplicaciones sea más rápido y eficiente.

A su vez, una vez realizadas modificaciones en nuestro contenedor, creamos una imagen a partir del mismo y la subimos al repositorio Docker Hub para que sea accesible desde cualquier equipo y bajo cualquier circunstancia. De esta forma se podrán crear nuevos contenedores a partir de la nueva imagen.

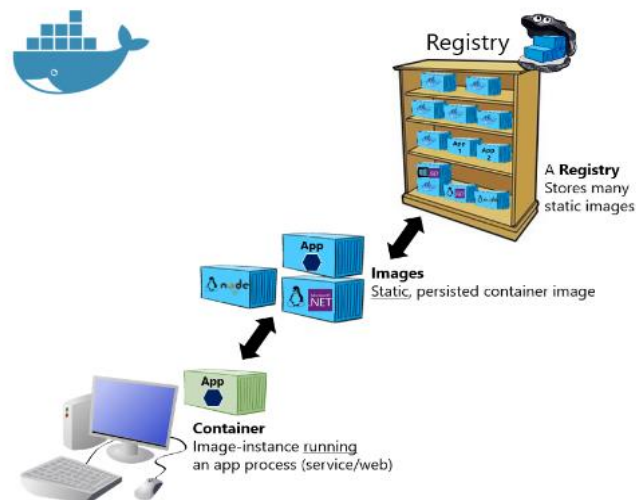


Figura 13: Contenedor, imagen y repositorio para Docker [10].

4.2.2.1. Modelo de red

Es muy importante que los contenedores puedan comunicarse entre sí, por ello Docker crea una red virtual por defecto llamada "bridge" o "docker0" al instalarse en un sistema. Esta red "docker0" actúa como una interfaz de red virtual que permite esta comunicación entre los contenedores Docker y el sistema host, así como la comunicación entre los propios contenedores en la misma máquina host.

La red "docker0" es una red privada que se crea automáticamente por Docker y se configura con un rango de direcciones IP predefinido. Docker utiliza NAT (Network Address Translation) para proporcionar conectividad a Internet a los contenedores de esta red. Por lo general, Docker asigna direcciones IP en el rango de 172.17.0.0/16 a los contenedores en esta red.

Cuando se crea un nuevo contenedor en Docker sin especificar una red específica, el contenedor se conecta automáticamente a la red "docker0" por defecto.

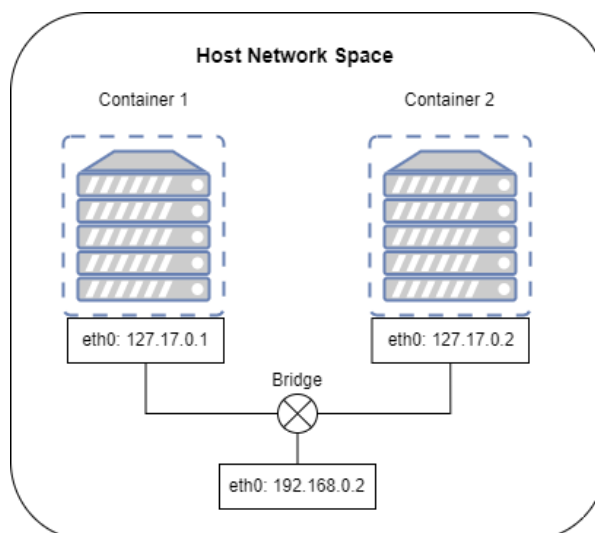


Figura 14: Contenedores en modo "bridge".

No obstante, hay otras configuraciones de red disponibles, entre las que nos interesa destacar la de tipo "host". En ella, los contenedores usan directamente la red del host, lo que significa que los contenedores utilizan la misma dirección IP y configuración de red que el host.

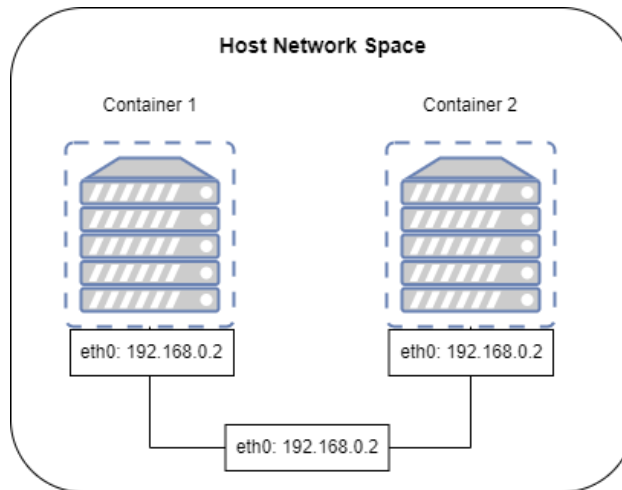


Figura 15: Contenedores en modo "host".

4.2.2.2. Aislamiento de contenedor Docker

A continuación, describiremos algunas formas de aislamiento un contenedor Docker:

- **Utilización de namespaces:** Docker utiliza namespaces del kernel de Linux para proporcionar aislamiento a nivel de sistema operativo para los contenedores. Los namespaces son una capa de abstracción de Linux que aísla los recursos del sistema (interfaces de red disponibles, lista de procesos, sistema de archivos, identificadores de usuario y grupo, etc). De esta forma dichos recursos aparecerán para los procesos ejecutándose en un namespace como los únicos disponibles. Esto permite que los procesos que se ejecutan dentro del contenedor vean y accedan a su propio conjunto de recursos aislados, lo que proporciona un nivel básico de aislamiento para los contenedores.
- **Configuración de cgroups:** utilización de cgroups (control groups) para limitar y gestionar los recursos del sistema que un contenedor puede utilizar, (CPU, memoria, almacenamiento, dispositivos, ancho de banda o procesos). Los cgroups permiten establecer límites y restricciones en el uso de recursos utilizados por los procesos que se ejecutan dentro del contenedor, lo que ayuda a prevenir que un contenedor consuma todos los recursos del sistema y afecte a otros contenedores o al sistema host.
- **Configuración de redes propias:** creación de redes propias para los contenedores, lo que permite establecer comunicaciones entre los contenedores de forma controlada y aislada del resto de la red del sistema host. Esto garantiza un nivel adicional de aislamiento y seguridad y facilita la escalabilidad de las aplicaciones en contenedores. Estas redes propias acceden a Internet haciendo uso de NAT.

4.3. Kubernetes

Si el número de aplicaciones crece en nuestro sistema, aumenta también la complejidad de gestión. Docker deja de ser suficiente, ya que necesitamos una coordinación para hacer el despliegue, la supervisión de servicios, el reemplazo, el escalado automático y, en definitiva, la administración de los distintos servicios que componen nuestra arquitectura distribuida.

Kubernetes [11] es una plataforma de orquestación de contenedores de código abierto. Se presentó en 2014 y fue desarrollada por Google. Esta herramienta gestiona cada uno de los ciclos de vida de los contenedores. Kubernetes permite automatizar la implementación, el escalado y la administración de aplicaciones en contenedores, lo que simplifica y acelera el proceso de desarrollo y despliegue de aplicaciones, es decir, se trata de un orquestador de contenedores.

4.3.1. Componentes de Kubernetes

A continuación, detallaremos los componentes de Kubernetes relevantes en este trabajo. Kubernetes trabaja en un entorno llamado clúster. El clúster es el nivel más alto de la jerarquía de Kubernetes y es un grupo de nodos virtuales o físicos, de forma que los nodos desarrollan una función conjunta. Se compone de un nodo master que controla y administra un conjunto de nodos worker.

Kubernetes soporta múltiples clústeres virtuales sobre el clúster físico. El clúster virtual se denomina namespace y permite establecer un nivel adicional de separación entre los contenedores que comparten los recursos de un clúster. Kubernetes proporciona dos namespaces por defecto: kube-system y default. Los objetos de sistema se crean en el espacio de nombres kube-system, mientras que los de usuario se encuentran en default.

Los pods son las unidades de computación desplegables más pequeñas que se pueden crear y gestionar en Kubernetes. Un pod puede alojar uno o más contenedores [12].

La estructura de Kubernetes se describe mediante objetos utilizando el lenguaje YAML.

4.3.1.1. Nodo master

El plano de control se aloja en el nodo máster, mantiene un registro de todos los objetos presentes y ejecuta bucles de control para gestionar el estado de los mismos [13]. Se compone de los siguientes elementos:

- **Etcd:** es una base de datos que almacena toda la información del clúster de Kubernetes.

- **Kube-apiserver:** es el front-end de Kubernetes, a través de él se reciben las peticiones del exterior y actualiza el estado en etcd.
- **Kube-scheduler:** su función es escoger el nodo en el que se deben ejecutar los nuevos pod.
- **Kube-controller-manager:** ejecuta los controladores de Kubernetes. Los controladores detectan y responden cuando un nodo deja de funcionar, mantienen el número adecuado de pods, construyen los endpoints (uniones entre pods y services) y crean tokens de acceso a la API.

4.3.1.2. Nodo worker

El nodo worker se componen de los siguientes elementos:

- **Kubelet:** es un agente que se ejecuta en cada nodo worker. Kubelet arranca los contenedores y monitoriza que sigan corriendo de manera normal. Esto lo realiza a través del reinicio del contenedor, en el caso de que este se acabe como consecuencia de una excepción fuera de control. También se comunica con el plano de control para el scheduling de nuevos pods.
- **Kube-proxy:** gestiona el tráfico de red dentro del clúster. Su principal función es el balanceo de carga [14].

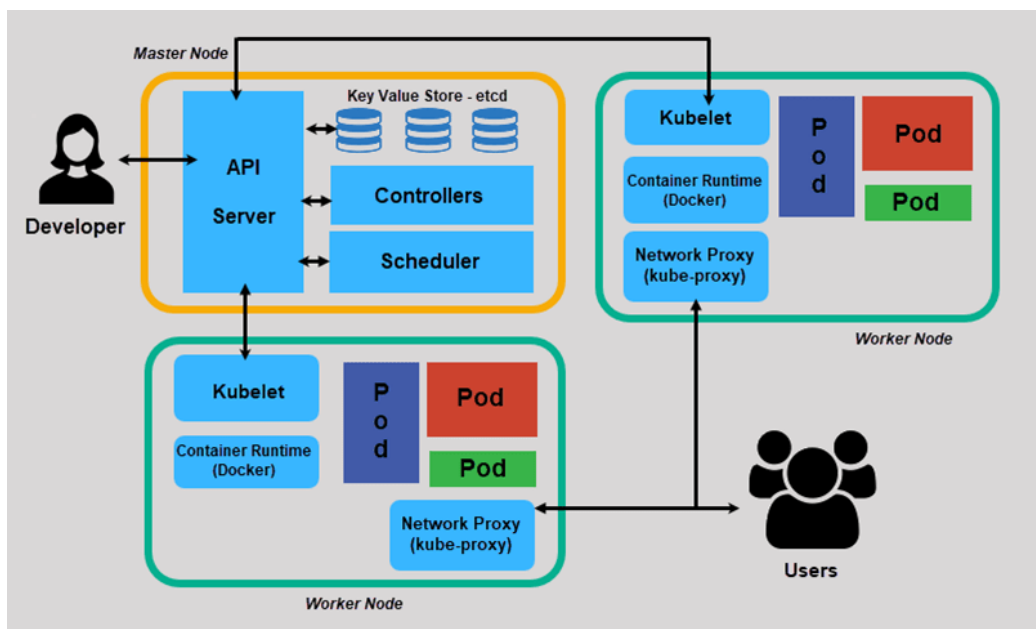


Figura 16: Arquitectura de Kubernetes.

4.3.2. Modelo de red de Kubernetes

Independientemente de sus nodos, se espera que los pods puedan comunicarse con otros pods en el clúster, todo eso sin necesidad de implementar NAT. Esto es un principio y requerimiento fundamental de cualquier implementación de redes en Kubernetes.

Los contenedores se integran con el modelo de red de Kubernetes mediante el uso de CNI (Container Network Interface) [15] es una especificación que sirve para configurar interfaces de red en contenedores de Linux. CNI se ocupa de la conectividad de red de los contenedores y de la eliminación de los recursos asignados cuando se elimina el contenedor.

Existen distintos tipos de CNIs:

- **Bridge:** es el CNI por defecto de Kubernetes y proporciona una red basada en puentes.
- **Host-local:** asocia una dirección IP a cada contenedor utilizando una subred predefinida.
- **Flannel:** utiliza una red superpuesta ("overlay network") para conectar los contenedores.
- **Calico:** utiliza una arquitectura basada en políticas de seguridad para aplicar reglas de seguridad de red en los contenedores.
- **Weave Net:** utiliza una red de superposición y admite múltiples modos de operación, incluyendo modo puente y modo enrutado.

En el siguiente apartado concretamos qué tipo de CNI utilizamos en nuestra solución.

4.4. K3s

Es una distribución ligera y certificada de Kubernetes creada por Rancher Labs. Entre sus ventajas está que es extremadamente liviano y que utiliza contenedores Docker. Es un producto diseñado para gestionar las cargas de trabajo en alta disponibilidad, en ubicaciones remotas desatendidas, con recursos limitados o dentro de dispositivos de IoT.

En la siguiente tabla comparamos la distribución K3s frente a K8s (abreviatura de Kubernetes):

K3s	K8s
Menor tamaño (menos de 200 MB)	Mayor tamaño (cientos de MB)
Menos dependencias	Más dependencias (incluidas etcd, kube-proxy, etc)
Requiere menos recursos (CPU, RAM, etc)	Utiliza más recursos, especialmente en clústeres grandes
Implementación y administración sencilla	Implementación y administración compleja
Escalabilidad limitada	Gran escalabilidad

Puede tener limitaciones para la alta disponibilidad	Opciones sólidas de alta disponibilidad, que incluyen redundancia a nivel de clúster, conmutación por error automática, etc
Menos superficies de ataque debido a una base de código más pequeña	Base de código más grande con más superficies de ataque potenciales

Tabla 2: Características de K3s y K8s [16].

K3s resulta ideal para pequeñas implementaciones con recursos limitados, como es nuestro caso. El uso de K8s sería más adecuado en implementaciones grandes y complejas con altos requerimientos de recursos (como por ejemplo en computación de alto rendimiento, aprendizaje automático o Big Data).

Resaltamos que el CNI integrado por defecto por K3s y, por lo tanto, utilizado en nuestra solución, es Flannel.

4.4.1. Flannel

Flannel [17] es una solución de red *overlay* para Kubernetes que permite a los contenedores en diferentes nodos de Kubernetes comunicarse entre sí. Una red *overlay* o red superpuesta, es una red construida sobre una red subyacente. En este caso Flannel crea una red virtual superpuesta sobre la red host.

Flannel utiliza por defecto el protocolo de encapsulación VXLAN (Virtual Extensible LAN) como backend, es decir, como mecanismo para la comunicación de red entre los nodos de un clúster. VXLAN se utiliza para encapsular tramas Ethernet dentro de datagramas UDP (User Datagram Protocol) y enviarlos a través de la red. Además, al añadir una etiqueta VXLAN es capaz de localizar redes distintas.

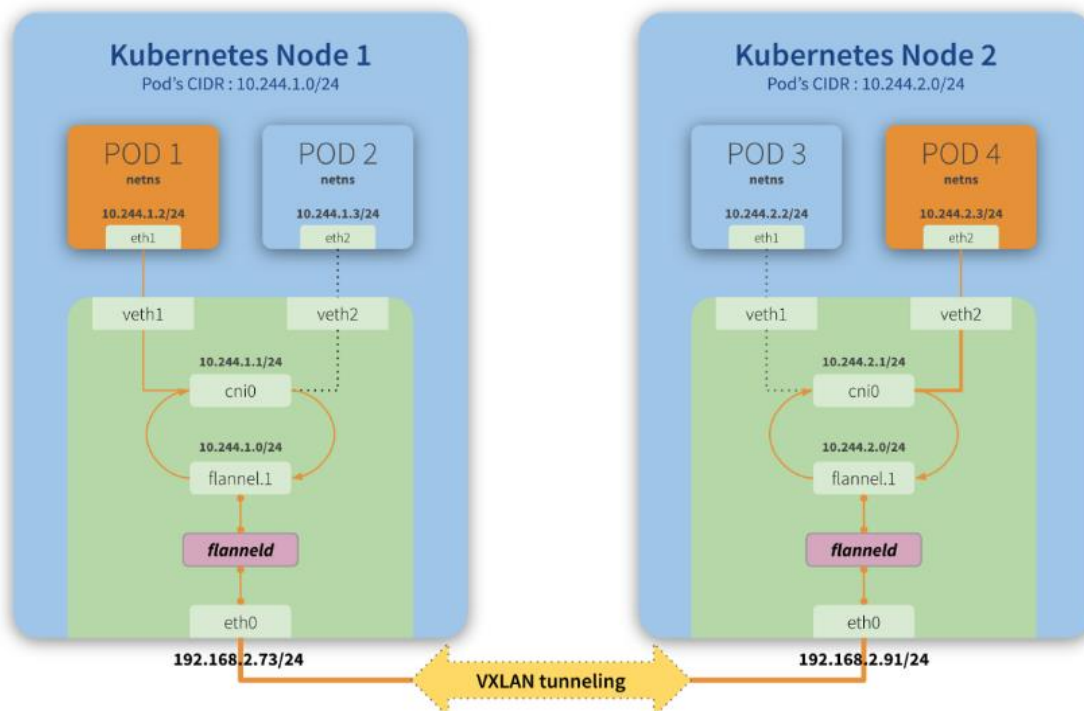


Figura 17: CNI Flannel en modo VXLAN [18].

En la Figura 17 representamos el modelo de red para la comunicación entre contenedores alojados en pods diferentes que se ubican en distintos nodos. Este escenario utiliza el CNI Flannel en modo VXLAN, como nosotros en nuestra solución.

Cada nodo tiene una interfaz de red principal (eth0) y cada pod también tiene una interfaz principal, por ejemplo, el pod1 tiene la interfaz eth1. Los contenedores dentro de un pod comparten dirección IP y puerto, y pueden accederse a través de localhost. Las interfaces veth son interfaces virtuales que vinculan las interfaces principales de cada pod con el cni0.

Cni0 es un dispositivo de puente de red de Linux, todos los dispositivos veth se conectarán a este puente, permitiendo la comunicación entre todos los pods del mismo nodo.

Al ejecutar Flannel en modo VXLAN se crea una red overlay que conecta los nodos del clúster mediante túneles de red virtuales entre los nodos. Estos túneles se establecen mediante la dirección IP de la interfaz flannel.1 de cada nodo.

Por último, flanneld es un demonio responsable de mantener actualizadas las rutas entre los nodos y entre los pods que contienen.

5. Evaluación de la virtualización

El objetivo de este trabajo es el estudio del despliegue de la SDWN virtualizada, por lo que hay que decidir qué componentes se virtualizan y evaluar si, como el estado del arte indica, es más ventajoso usar contenedores o no para la virtualización.

Para la evaluación ejecutamos un proceso de transmisión controlada de tramas Wi-Fi y medimos los recursos de la unidad central de procesamiento (CPU) que se utilizan, realizando esta prueba por separado sobre la máquina real, la máquina virtual y el contenedor Docker. Los AP se van a dedicar a transmitir y recibir tramas Wi-Fi, luego este tipo de proceso, si bien no es el AP real, nos puede dar una ligera idea del consumo de un componente principal del diseño.

De esta forma esta prueba nos permite establecer comparaciones entre los métodos de virtualización y familiarizarnos con el uso de los contenedores Docker. El proceso que ejecutamos requiere cómputo operacional y por lo tanto gastará recursos de la CPU, pero apenas consume recursos de memoria. El código del proceso se incluye en el Anexo 6.

El proceso consiste en la transmisión Wi-Fi de tramas previamente cifradas. El cifrado de las mismas es lo que requiere cómputo, nuevamente hacemos esto para emular un software con demanda de procesado. Mediante un sleep (intervalo de tiempo configurable en el que la ejecución del código no avanza) determinaremos cuánto tiempo transcurre antes de empezar a mandar cada trama. Es decir, si el sleep tiene un valor de 1 ms, esperaremos ese intervalo temporal antes de mandar cada trama. De esta forma podemos variar el número de tramas que transmitimos por segundo. Para las pruebas realizadas transmitimos mil tramas y no limitamos el uso de las CPUs disponibles en la máquina real. Nuestra máquina real dispone de 4 CPUs, por lo que tanto el contenedor Docker como la máquina virtual tendrán acceso a todas ellas para la ejecución del proceso.

5.1. Antenas

Para la inyección de tráfico Wi-Fi utilizamos la Antena RTL8812AU, que conectamos al servidor remoto mediante usb.



Figura 18: Antena RTL8812AU Realtek.

Escogemos esta antena porque es una antena que trabaja en el estándar de Wi-Fi AC, es decir, es capaz de inyectar tráfico en la banda de 5 GHz.

El comando `lsusb` nos muestra los dispositivos conectados a los puertos USB de nuestro servidor remoto:

```
proyecto@proyecto:~$ lsusb
Bus 001 Device 002: ID 8087:8001 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 002: ID 0b95:1790 ASIX Electronics Corp. AX88179 Gigabit Ethernet
Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 002 Device 005: ID 8087:0a2a Intel Corp.
Bus 002 Device 004: ID 0bda:8812 Realtek Semiconductor Corp. RTL8812AU 802.11a/b/g/n/ac 2T2R DB WLAN Adapter
Bus 002 Device 003: ID 0bda:8812 Realtek Semiconductor Corp. RTL8812AU 802.11a/b/g/n/ac 2T2R DB WLAN Adapter
Bus 002 Device 002: ID 03f0:164a HP, Inc HP SC Keyboard - Apollo (Liteon)
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figura 19: Dispositivos USB conectados.

Nos interesa observar a qué número de *bus* y *device* están conectadas nuestras antenas.

Para asignar la primera de ellas a nuestra instancia virtual ejecutamos la máquina virtual de la siguiente manera:

```
/usr/bin/qemu-system-x86_64 -hda /home/gns3/GNS3/images/QEMU/WIFI-
carmen-160.qcow2 -boot d -m 256M -smp cpus=4 -nographic
-device ich9-usb-ehci1,id=usb,bus=pci.0 -device usb-
host,hostbus=2,hostaddr=4,vendorid=0x0bda,productid=0x8812,id=hostdev0,bus=
usb.0
```

Así podemos asignar el número de tarjetas Wi-Fi que queramos a nuestra instancia virtual.

En nuestro caso este método de asignación es el utilizado en la máquina virtual. Una vez asignado el recurso, la máquina virtual se apropia de él, dejando de ser visible para la máquina host o cualquier otra máquina.

5.2. Procesado de los datos

Nuestro objetivo es obtener dos informaciones distintas:

- **Número de paquetes transmitidos/recibidos en cada segundo de ejecución:** podríamos calcular simplemente el valor promedio, pero realizamos un análisis más fino calculando el número exacto de paquetes transmitidos segundo a segundo. Para ello introducimos marcas temporales en las tramas transmitidas, que imprimimos por pantalla, para visualizar en qué instante de tiempo ha sido transmitida cada trama y así poder hacer un recuento en cada uno de los segundos. Para llevar esto a cabo utilizaremos las hojas de cálculo de Excel y un script de Matlab cuyo código adjuntamos en el Anexo 6.
- **Porcentaje de CPU y memoria utilizados por el proceso:** para conocer el porcentaje de recursos utilizados usamos el comando `top`. El comando `top`

permite ver las tareas del sistema que se ejecutan en tiempo real y el uso de la CPU que hace cada una de las tareas, entre otras informaciones. La frecuencia de refresco estándar del comando es de tres segundos. En nuestro caso nos interesa conocer esta medida cada segundo por lo que especificamos esta frecuencia de refresco de las medidas al ejecutar el comando de la siguiente manera: *top -d 1*.

5.3. Creación de contenedor

A partir de una imagen podemos crear un contenedor, hacer las modificaciones que deseemos en él y después crear una imagen a partir del mismo para que los cambios sean definitivos.

Hay distintas opciones:

- **docker create:** crea un nuevo contenedor a partir de una imagen. Pero no lo ejecuta.
- **docker start:** ejecuta un contenedor que previamente fue creado. Es decir, después de crear un contenedor con el comando `docker create` podemos ejecutarlo con este comando.
- **docker run:** crea un contenedor y, acto seguido, lo pone en funcionamiento. Es una combinación de los dos anteriores.
- **docker exec:** ejecuta dentro de un contenedor creado y en funcionamiento el comando que se le indique.

En nuestro caso crearemos y ejecutaremos el contenedor al mismo tiempo, por lo que utilizaremos el comando `docker run`. Este comando tiene numerosas opciones, pero únicamente nos centraremos en las que hemos utilizado para nuestro propósito. Creamos y ejecutamos un contenedor de la siguiente forma:

```
sudo docker run -i -t --privileged --net=host image_name
```

-i: Para adjuntar el standard input stream (STDIN) del proceso del contenedor con el terminal host. De esta forma podemos interactuar con el proceso como si lo ejecutásemos en el host.

-t: para asignar un pseudo-TTY y tener habilitada la funcionalidad completa del terminal.

-- privileged: para otorgar acceso a todos los dispositivos del host. Para evaluar la virtualización y desplegar los puntos de acceso queremos que el contenedor tenga acceso a las antenas Wi-Fi que utilizamos, pudiendo encenderlas, apagarlas y cambiar su canal y modo.

--net=host: con la red configurada en tipo host un contenedor compartirá la pila de red del host y todas las interfaces del host estarán disponibles para el contenedor. Le permite al contenedor un mayor acceso a la red de lo que normalmente puede obtener. Es necesario que el contenedor que utilizamos en la evaluación de la

virtualización pueda acceder a las interfaces inalámbricas del servidor porque queremos inyectar tráfico Wi-Fi. Cabe destacar que con esta configuración el recurso seguirá siendo visible para el host, es decir, el contenedor tiene acceso, pero no se apropia de él.

image_name: imagen a partir de la cual creamos el contenedor. La imagen la creamos en todos los casos a partir de una imagen padre Debian 10, como indicamos en el punto “4.2.1. Imagen Docker”. A partir de la imagen padre hemos realizado las modificaciones necesarias para obtener la imagen utilizada en la prueba de virtualización. Detallamos la creación de la imagen en el Anexo 7.

5.4. Resultados

En las siguientes gráficas presentamos el número de paquetes transmitidos por segundo (eje de abscisas) frente al porcentaje de CPU utilizado (eje de ordenadas).

En esta primera prueba ejecutamos el código directamente desde la máquina real, sin utilizar ningún método de virtualización.

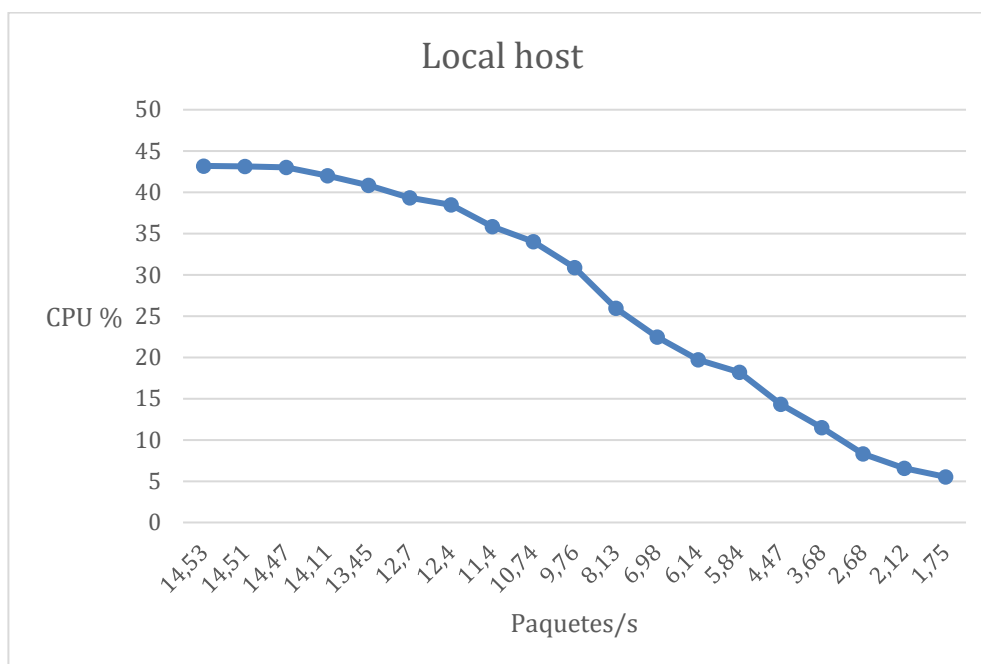


Figura 20: Prueba de carga realizada en el local host.

A continuación, realizamos la misma prueba sobre el contenedor Docker creado como hemos detallado con anterioridad:

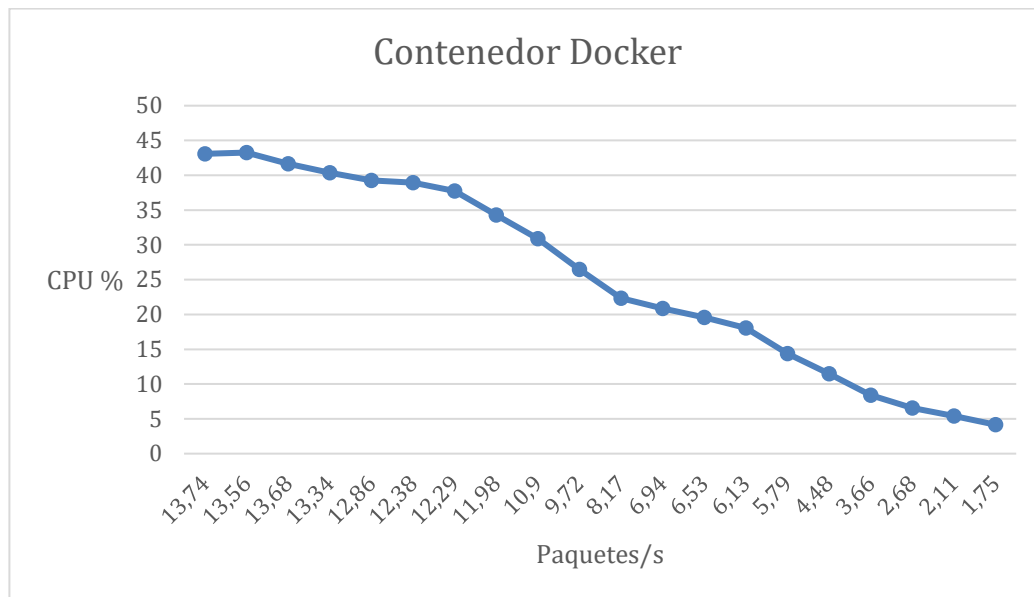


Figura 21: Prueba de carga realizada en el contenedor Docker.

Por último, obtenemos los resultados respectivos a la máquina virtual:

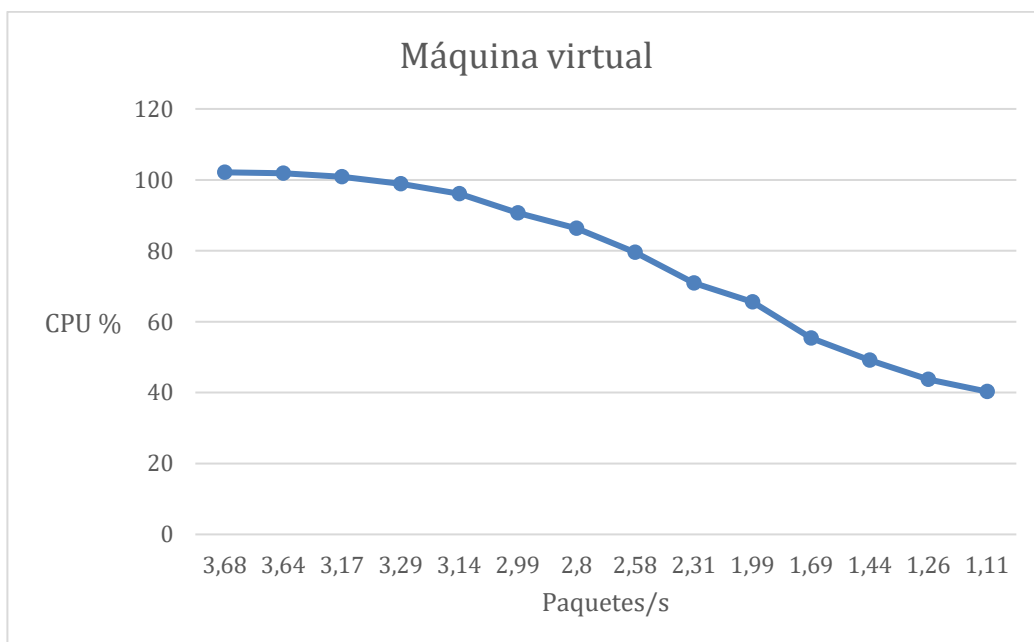


Figura 22: Prueba de carga realizada en la máquina virtual.

El máximo número de paquetes transmitidos por segundo corresponde con un time sleep de cero segundos, es decir, no esperamos ningún tiempo entre las transmisiones de paquetes.

En la máquina real el máximo de paquetes transmitidos por segundo es de 14,53, en el contenedor este valor desciende a 13,74 y por último en la máquina virtual únicamente transmitimos 3,68 paquetes cada segundo. La diferencia entre ambos

métodos de virtualización es grande, obteniendo mejores resultados en el contenedor Docker.

No obstante, también debemos observar el gasto de CPU, utilizando la máquina real el máximo consumo es de 43,19%, con el contenedor es de 43,49% y con la máquina virtual de 102,12%. Cabe señalar que el valor de CPU consumido supera el 100% porque hace referencia a las 4 CPUs disponibles en la máquina real, pudiendo llegar al 400%.

Resulta evidente que la virtualización mediante el contenedor resulta más eficiente y productiva que con la máquina virtual, que consume muchos más recursos para ejecutar el mismo código. Por último, superponemos las tres gráficas para obtener una comparativa visual de los resultados previamente comparados:

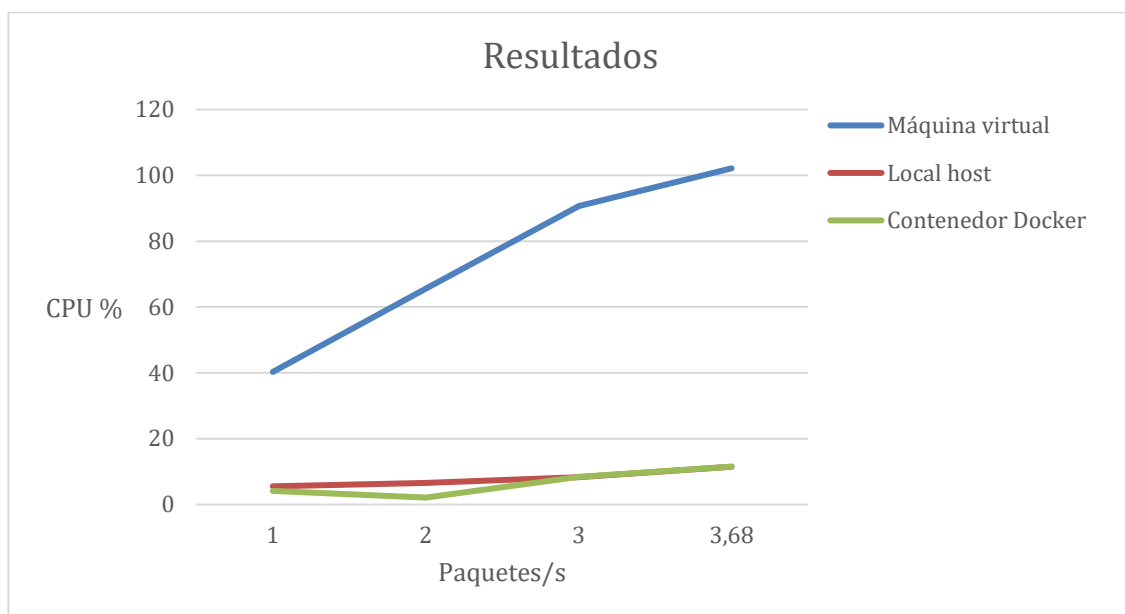


Figura 23: Superposición de resultados obtenidos.

6. Despliegue automatizado del escenario Wi-Fi

En este apartado estudiamos la automatización del despliegue de la red propuesta en la Figura 5. Nos centramos en la automatización del plano de control, concretamente, en los puntos de acceso y el controlador.

En la evaluación de virtualización hemos visto que utilizar contenedores para procesos que hacen uso de la tecnología Wi-Fi no solo es factible, sino ventajoso en utilización de recursos respecto a la máquina virtual. Por lo que la virtualización de los AP en contenedores resulta adecuada.

En el caso del controlador, el software consiste en procesos de aplicación que no requieren uso de Wi-Fi y que resultan más sencillos de virtualizar y responden al uso clásico de contenedores.

En cuanto al plano de datos, como tiene implicaciones con servicios de red (NAT, DHCP, encaminamiento...) se ha pospuesto su análisis ya que se ha considerado fundamental verificar que el plano de control, que es la clave de la red SDWN, se puede desplegar correctamente.

Para el despliegue utilizamos la herramienta K3s mencionada anteriormente y el modelo de red Flannel con VXLAN también detallado con anterioridad. Esta solución virtualizada es la que se estudia en este apartado.

6.1. Prueba preliminar de virtualización en Docker

En este apartado creamos los contenedores que alojan los puntos de acceso y el controlador con el objetivo de verificar la realización de un handoff. El handoff es la transferencia de una conexión de un punto de acceso a otro.

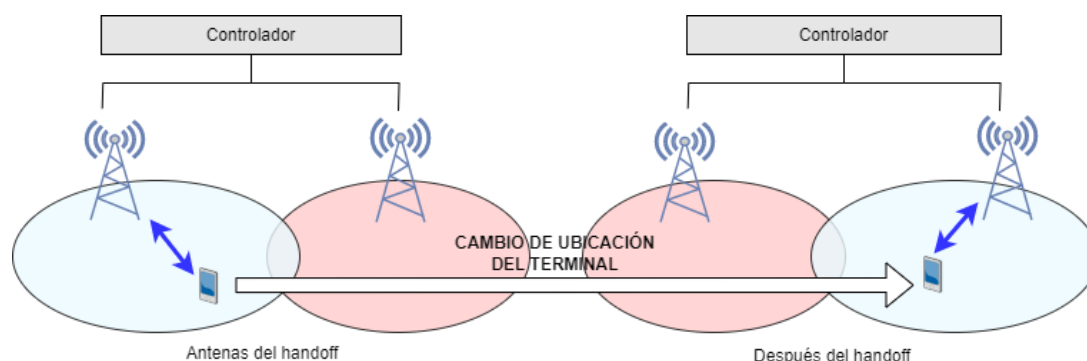


Figura 24: Esquema genérico de handoff.

Creamos el contenedor del punto de acceso de la misma forma que en el apartado “5.3. Creación del contenedor”. La imagen Docker del punto de acceso requiere los componentes detallados en el Anexo 7, donde también presentamos los comandos necesarios para su creación.

Además, debemos acceder al fichero de configuración para configurar las siguientes variables:

- La dirección IP del controlador.
- Las direcciones IP de las interfaces de control y de datos.
- Las direcciones MAC de las interfaces inalámbricas.

Esta configuración podemos hacerla al crear la imagen, creando una imagen para cada punto de acceso que despleguemos. Otra alternativa es crear una única imagen y realizar la configuración en cada contenedor que creamos a partir de dicha imagen. Nosotros hemos optado por la primera opción, porque nos permite subir las imágenes ya configuradas para cada punto de acceso al repositorio de Docker Hub. Esto nos aporta seguridad y rapidez si queremos desplegar dichos puntos de acceso en otras máquinas.

Por otra parte, el contenedor del controlador lo creamos de la siguiente manera:

```
sudo docker run -i -t controller_image
```

Observamos que, a diferencia del punto de acceso, el controlador no requiere privilegios para acceder a los dispositivos del host porque no accede a las antenas Wi-Fi. En consecuencia, tampoco requiere configuración de red en tipo host. En el Anexo 7 detallamos la creación de la imagen del controlador.

En el caso del controlador debemos acceder al archivo de configuración para indicar la dirección IP de la interfaz de control de cada punto de acceso. Igual que en el caso de los puntos de acceso, podemos hacer esta configuración antes de crear la imagen del controlador o posteriormente en el contenedor creado a partir de la imagen.

En este caso realizamos la configuración sobre el contenedor del controlador, porque al tratarse únicamente de un parámetro su configuración es rápida y puede interesarnos su modificación. No obstante, también subimos la imagen Docker del controlador al repositorio Docker Hub.

Una vez creadas ambas imágenes y a partir de ellas los contenedores, ponemos en funcionamiento los puntos de acceso. En el Anexo 8 detallamos el script ejecutado para poner en funcionamiento cada punto de acceso.

Pasados unos segundos, veremos el siguiente mensaje por pantalla: "Now you can launch the controller and press Enter". En ese momento ejecutamos los siguientes comandos en el contenedor del controlador:

```
1 cd Wi5/odin-wi5-controller-gui-development
2 ./initiate_odin_master
```

Estos comandos simplemente ejecutan el script "initiate_odin_master" ubicado en el directorio "/Wi5/odin-wi5-controller-gui-development".

Por último, ejecutamos "Enter" en cada punto de acceso.

A continuación, verificamos que se ejecuta un handoff cuando el terminal lo requiere, capturando el tráfico inalámbrico generado en el proceso de traspaso.

En la Figura 25 representamos el intercambio de tramas que intervienen en el proceso de handover:

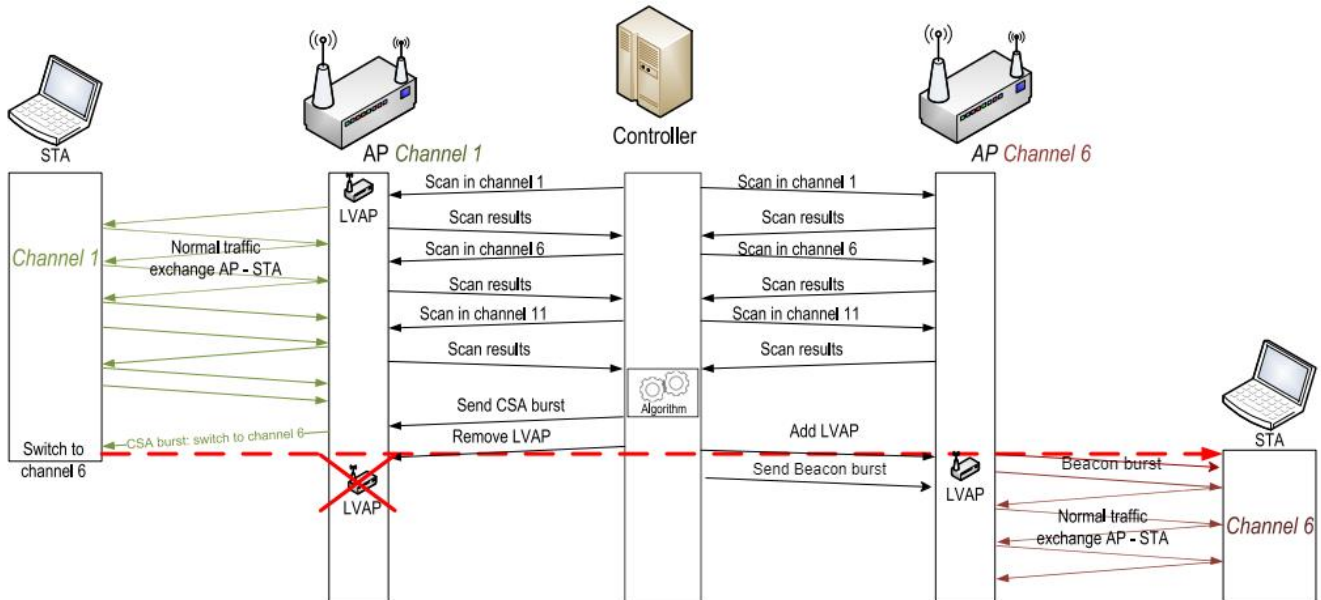


Figura 25: Intercambio de tramas en el handover [3].

En una red Wi-Fi, cada punto de acceso inalámbrico tiene un BSSID (Basic Service Set Identifier) único que lo identifica. Un BSSID es una dirección MAC única que se asigna a un punto de acceso inalámbrico (WAP) en una red Wi-Fi. Es utilizado por los dispositivos inalámbricos para identificar y conectarse a una red Wi-Fi específica.

También es posible que haya varios puntos de acceso que compartan el mismo BSSID. En nuestro caso implementamos esta última opción cambiando la dirección MAC de la antena principal de ambos puntos de acceso mediante el comando 'macchanger'. De esta forma el usuario siempre se comunicará con la misma MAC, siendo transparente para él el cambio de punto de acceso, como representamos en la Figura 26:

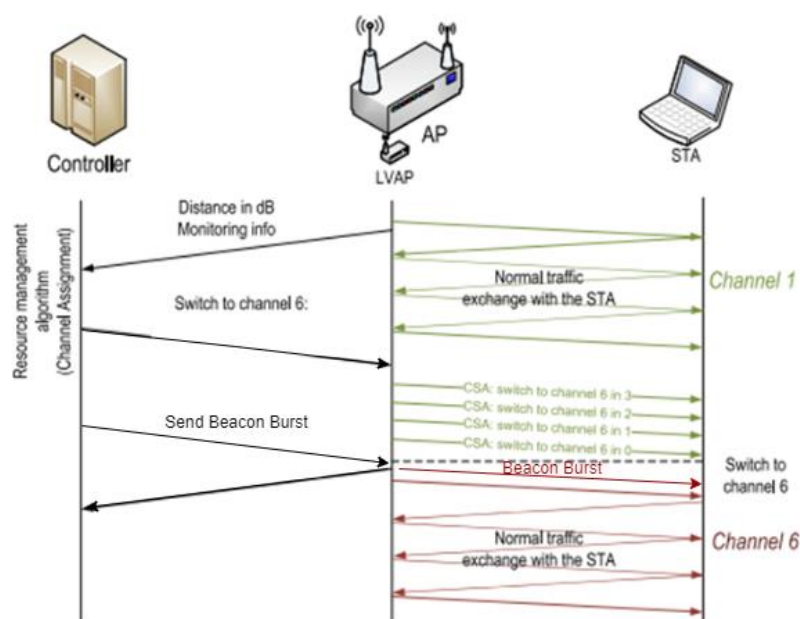


Figura 26: Intercambio de tramas desde la perspectiva del usuario [3].

Una de las tramas más relevantes en Wi-Fi es la trama beacon (Beacon Frame), transmitida periódicamente por un punto de acceso (AP) para anunciar la presencia de una red inalámbrica y proporcionar información básica sobre la misma. Estas tramas se utilizan para sincronizar los dispositivos inalámbricos con la red y permitirles conectarse a ella. La trama beacon contiene información como el SSID (nombre de la red), la dirección MAC del punto de acceso, el tipo de seguridad utilizado, el canal de frecuencia y otros datos de configuración importantes para que los dispositivos puedan unirse a la red de forma segura.

Los puntos de acceso de la red monitorizan la señal, mandan la información al controlador y este ordena la ejecución del handover (traspaso) cuando el nivel de señal es inferior a -40 dBm (parámetro configurable).

```
#####
17:13:29.009 [pool-3-thread-11] INFO n.f.o.applications.MobilityManager - [handler] Ordering the hand off of STA AE:0E:D7:58:92:86 to agent /192.168.1.104
17:13:29.010 [pool-3-thread-11] INFO n.f.o.applications.MobilityManager -
#####
```

Figura 27: Controlador ordena handover.

En ese momento el punto de acceso actual anuncia al móvil el cambio de canal o CSA (Channel Switch Announcement) y el número del nuevo canal. Esto permite que el móvil cambie al canal indicado sin necesidad de hacer una nueva asociación. El número de veces que se manda la trama CSA es configurable, en nuestro caso se envía en cuatro ocasiones.

```

> Frame 9881: 226 bytes on wire (1808 bits), 226 bytes captured (1808 bits)
> Radiotap Header v0, Length 41
> 802.11 radio information
✓ IEEE 802.11 Beacon frame, Flags: .....C
  Type/Subtype: Beacon frame (0x0008)
  > Frame Control Field: 0x8000
    .000 0000 0011 1100 = Duration: 60 microseconds
    Receiver address: ae:0e:d7:58:92:86 (ae:0e:d7:58:92:86)
    Destination address: ae:0e:d7:58:92:86 (ae:0e:d7:58:92:86)
    Transmitter address: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    Source address: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    BSS Id: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    .... 0000 = Fragment number: 0
    0111 0000 0101 .... = Sequence number: 1797
    Frame check sequence: 0x3699e5a8 [unverified]
    [FCS Status: Unverified]
✓ IEEE 802.11 Wireless Management
  > Fixed parameters (12 bytes)
  > Tagged parameters (145 bytes)
    > Tag: SSID parameter set: wi5orange
    > Tag: Supported Rates 1(B), 2(B), 5.5(B), 6, 9, 11(B), 12, 18, [Mbit/sec]
    > Tag: DS Parameter set: Current Channel: 1
    > Tag: Traffic Indication Map (TIM): DTIM 0 of 1 bitmap
    > Tag: Country Information: Country Code ES, Environment Any
    > Tag: QSS Load Element 802.11e CCA Version
    > Tag: ERP Information
    > Tag: HT Capabilities (802.11n D1.10)
    > Tag: Extended Supported Rates 24, 36, 48, 54, [Mbit/sec]
    > Tag: HT Information (802.11n D1.10)
    > Tag: Extended Capabilities (6 octets)
    > Tag: Vendor Specific: Microsoft Corp.: WMM/WME: Parameter Element
    > Tag: Channel Switch Announcement Mode: 0, Number: 11, Count: 4

```

Figura 28: Trama Beacon CSA unicast.

En esta captura observamos que en nuestro caso el envío de esta trama es unicast. En los campos de la trama observamos que el canal utilizado por el nuevo punto de acceso es el 11 y que esta trama tiene el subíndice cuatro. Normalmente se anuncia la trama cuatro veces.

Inmediatamente el segundo punto de acceso manda una ráfaga de beacon al móvil para asegurar el traspaso y evitar que el terminal pierda la asociación con el LVAP. El número de tramas que compone la ráfaga también es configurable, en nuestro caso mandamos cinco tramas.

```

> Frame 15232: 219 bytes on wire (1752 bits), 219 bytes captured (1752 bits)
> Radiotap Header v0, Length 39
> 802.11 radio information
✓ IEEE 802.11 Beacon frame, Flags: .....C
  Type/Subtype: Beacon frame (0x0008)
  > Frame Control Field: 0x8000
    .000 0000 0011 1100 = Duration: 60 microseconds
    Receiver address: ae:0e:d7:58:92:86 (ae:0e:d7:58:92:86)
    Destination address: ae:0e:d7:58:92:86 (ae:0e:d7:58:92:86)
    Transmitter address: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    Source address: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    BSS Id: Alfa_a4:7b:aa (00:c0:ca:a4:7b:aa)
    .... 0000 = Fragment number: 0
    0100 0010 1100 .... = Sequence number: 1068
    Frame check sequence: 0x2c3faf8a [unverified]
    [FCS Status: Unverified]
✓ IEEE 802.11 Wireless Management
  > Fixed parameters (12 bytes)
  > Tagged parameters (140 bytes)
    > Tag: SSID parameter set: wi5orange
    > Tag: Supported Rates 1(B), 2(B), 5.5(B), 6, 9, 11(B), 12, 18, [Mbit/sec]
    > Tag: DS Parameter set: Current Channel: 11
    > Tag: Traffic Indication Map (TIM): DTIM 0 of 1 bitmap
    > Tag: Country Information: Country Code ES, Environment Any
    > Tag: QSS Load Element 802.11e CCA Version
    > Tag: ERP Information
    > Tag: HT Capabilities (802.11n D1.10)
    > Tag: Extended Supported Rates 24, 36, 48, 54, [Mbit/sec]
    > Tag: HT Information (802.11n D1.10)
    > Tag: Extended Capabilities (6 octets)
    > Tag: Vendor Specific: Microsoft Corp.: WMM/WME: Parameter Element

```

Figura 29: Trama Beacon unicast.

6.2. Elementos virtualizados en K3s

La primera decisión de despliegue es definir la arquitectura física correspondiente, es decir, en qué nodos se va a configurar el clúster de kubernetes para ejecutar los pods correspondientes.

En nuestro caso, dada la necesidad de ubicar los AP en localizaciones separadas para dar la cobertura necesaria, es imprescindible, al menos dos nodos para ejecutar (nodos worker). La ubicación del nodo controlador no tiene exigencias de localización por lo que puede ejecutarse de manera separada de los AP o co-localizado con alguno de ellos.

Así, como muestra la Figura 30, una opción es desplegar tres nodos: uno master y dos nodos worker. Uno de los nodos para alojar el controlador y cada uno de los restantes un punto de acceso. Únicamente contemplamos dos puntos de acceso porque sería suficiente para poder realizar pruebas de movilidad entre ambos.

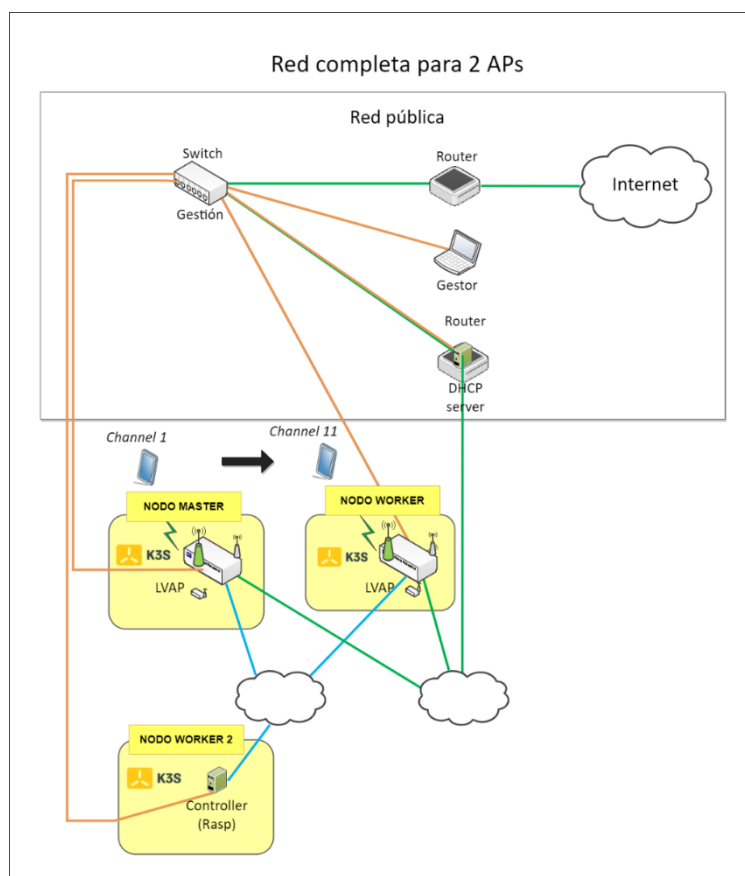


Figura 30: Red completa con KS3 en tres nodos.

Como segunda opción, como el controlador podría estar situado en la misma máquina que uno de los puntos de acceso, podemos utilizar únicamente un nodo master (con funcionalidad adicional de worker) y un nodo worker, dando lugar al siguiente escenario:

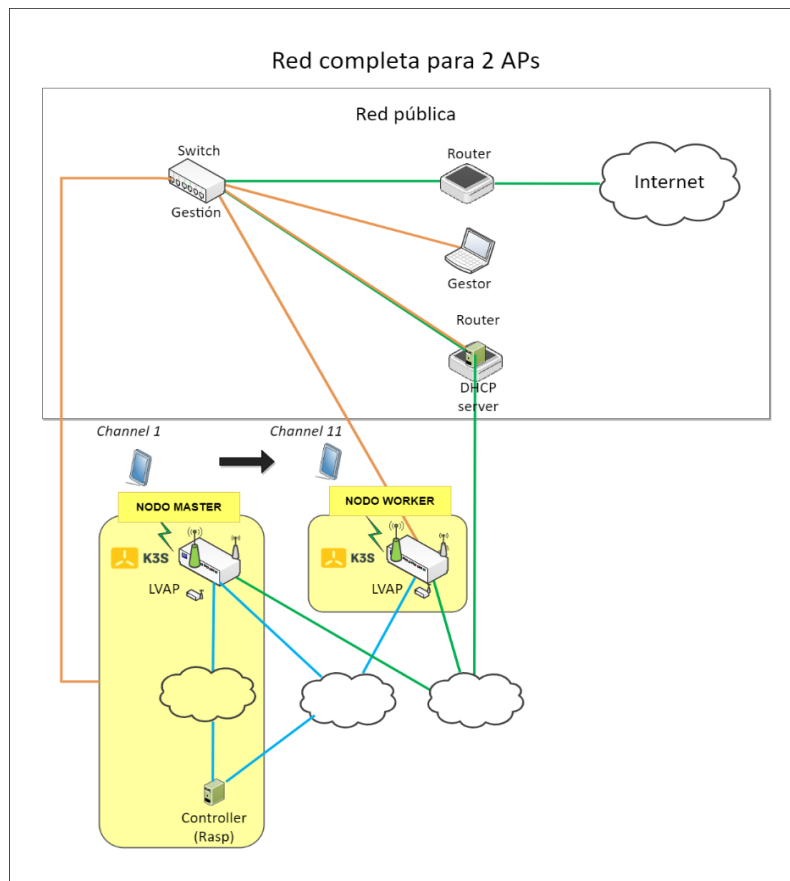


Figura 31: Red completa con KS3 en dos nodos.

6.3. Estudio de conectividad

Una vez planteados los dos escenarios posibles en las Figuras 30 y 31, en este apartado vamos a realizar un estudio de la conectividad entre los nodos involucrados en ambas soluciones.

De acuerdo con lo estudiado para el caso de Docker en el punto “4.4.1. Creación del contenedor”, en Kubernetes resulta necesario aplicar las mismas necesidades en los contenedores que alojan los puntos de acceso. Por lo tanto, los pods que desplegamos para alojar dichos contenedores deben tener acceso a las interfaces del host y privilegios para controlar las antenas Wi-Fi conectadas a dichas interfaces.

Por otra parte, los controladores Odin Controller y Open Controller, no requieren acceso a ningún recurso específico de la máquina host, por lo que el pod que aloja el contenedor correspondiente puede pertenecer a la red Flannel.

Alojamos en un contenedor diferenciado cada punto de acceso y el controlador, y únicamente ubicamos un contenedor por pod. De esta forma la IP de cada contenedor será distinta, teniendo una IP diferente para cada punto de acceso y para el controlador. Para estudiar la conectividad entre los distintos elementos, dadas las condiciones indicadas, planteamos diferentes escenarios para acercarnos

progresivamente a la solución deseada: tener conectividad entre dos pods en modo host (APs) y un pod perteneciente a la red Flannel (controlador).

Dichos pods podrán estar ubicados en dos o en tres nodos, como ya hemos planteado en el apartado anterior “6.2. Elementos virtualizados”. En primer lugar, estudiamos el escenario que contempla el despliegue de tres nodos y planteamos la primera aproximación de que todos los pods pertenecen a la red Flannel:

- **Tres nodos que contienen tres pods en la red Flannel:**

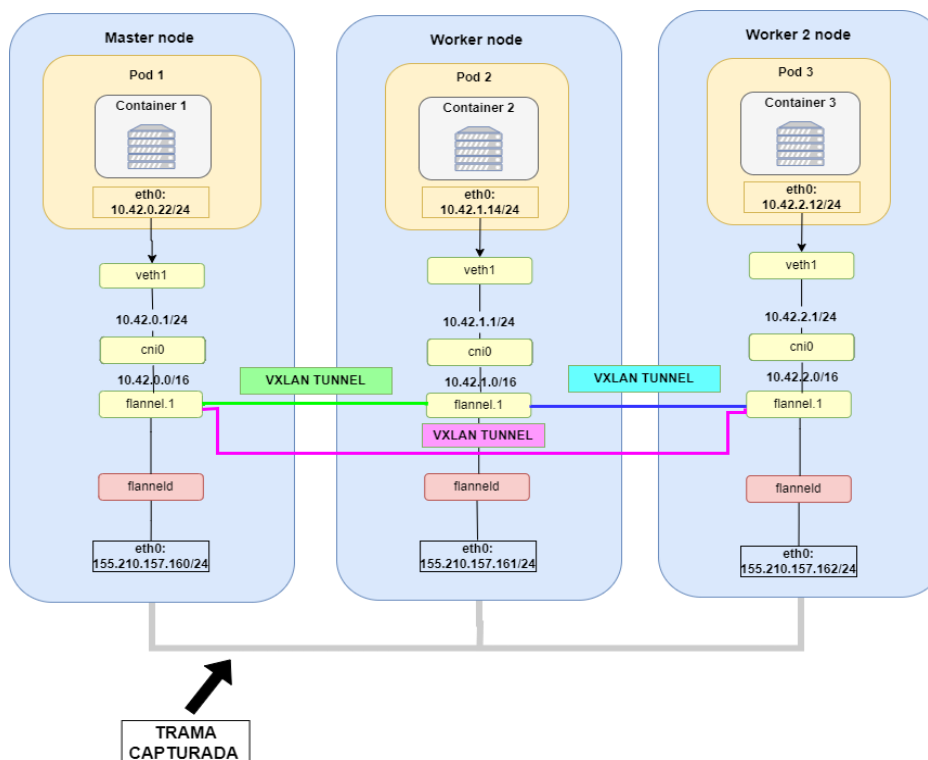


Figura 32: Tres pods en tres nodos en la red Flannel.

Observamos que la red Flannel es del rango 10.42.0.0/16 y cada nodo tiene una subred de dicho rango: 10.42.0.0/24 (nodo master), 10.42.1.0/24 (nodo worker) y 10.42.2.0/24 (nodo worker 2).

Cada pod se encuentra en un nodo diferente y tiene una dirección privada perteneciente a la subred de dicho nodo. La interfaz flannel.1 de cada nodo sirve de extremo de los túneles VXLAN que se establecen entre los diferentes nodos.

La Tabla 3 muestra las rutas del pod en el nodo master. Para el resto de los pod seguiría la misma filosofía:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	10.42.0.1	0.0.0.0	UG	0	0	0	eth0
10.42.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
10.42.0.0	10.42.0.1	255.255.0.0	UG	0	0	0	eth0

Tabla 3: Tabla de rutas del pod en la red Flannel.

La primera entrada indica que cualquier tráfico destinado a una red desconocida es enviado a través de `cni0` (10.42.0.1), que actúa como puerta de enlace predeterminada. Por otra parte, el tráfico destinado a la red 10.42.0.0/24 (red del pod) se enruta directamente a través de la interfaz de red `eth0`, sin necesidad de una puerta de enlace. La última entrada indica que el tráfico destinado a la red 10.42.0.0/16 (la que incluye los otros nodos, 10.42.1.0/24 y 10.42.2.0/24) se enviará a través de la puerta de enlace predeterminada.

En el nodo master tenemos la siguiente tabla de rutas, pero en el resto de nodos seguiría la misma lógica:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	155.210.157.254	0.0.0.0	UG	0	0	0	eth0
155.210.157.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
10.42.0.0	0.0.0.0	255.255.255.0	U	0	0	0	cni0
10.42.1.0	10.42.1.0	255.255.255.0	UG	0	0	0	flannel.1
10.42.2.0	10.42.2.0	255.255.255.0	UG	0	0	0	flannel.1

Tabla 4: Tabla de rutas del nodo master.

Cualquier paquete dirigido a una red desconocida se envía a través del router del nodo (155.210.157.254). Un paquete destinado a la red local 155.210.157.0/24 se encamina directamente a través de la interfaz `eth0`.

Cualquier paquete perteneciente a la red 10.42.0.0/24 (red del nodo master) se encamina a través del `cni0`.

No obstante, los paquetes dirigidos a las redes de los nodos worker 1 (10.42.0.1/24) y worker 2 (10.42.0.2/24) se encaminan a través de la interfaz `flannel.1` del respectivo nodo. Para entender la consulta de la dirección MAC de dicha IP hay que tener en cuenta que el registro ARP es agregado automáticamente en el nodo master por el demonio `flanneld`.

A continuación, ilustramos la estructura de la trama capturada en el enlace indicado en el escenario de la Figura 32. La trama se transmite desde el nodo master al nodo worker:

Cabecera externa Ethernet	IP Nodo Master (fuente)	IP Nodo Worker (destino)	MAC Nodo Master (fuente)	MAC Nodo Master (fuente)	Cabecera externa UDP	Datos UDP					
						Cabecera VXLAN	MAC Flannel.1 Nodo Master (fuente)	MAC Flannel.1 Nodo Worker (destino)	IP Pod 1 (fuente)	IP Pod 3 (destino)	Datos

Figura 33: Trama capturada en la Figura 32.

La trama se transmite con las IP de los pod y en la interfaz `flannel` se añade la MAC de la interfaz `flannel` del nodo destino, que actúa como gateway. Esta MAC se conoce debido a que el demonio `flanneld` se encarga de completar la tabla ARP de los nodos para las interfaces `flannel`. La trama tiene una cabecera VXLAN y se encapsula sobre UDP, siendo encaminada entre los nodos master (fuente) y worker (destino).

A continuación, debemos tener en cuenta que los puntos de acceso necesitan ser configurados en modo “host” por lo que adaptamos el escenario anterior a esta particularidad:

- **Tres nodos que contienen dos pods en la red host y un pod en la red Flannel:**

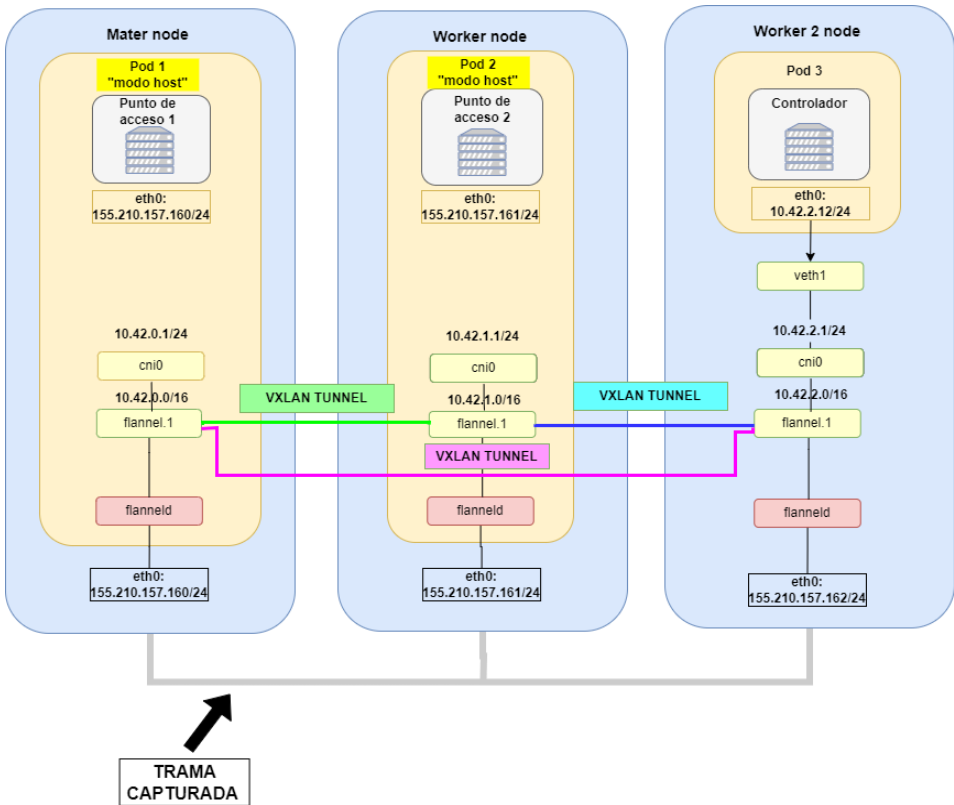


Figura 34: Tres nodos: dos pods en la red host y un pod en la red Flannel.

Como observamos en la Figura 34 los pods en “modo host” no tienen una IP asignada dentro de la red Flannel.

En este caso los pods en “modo host” tienen acceso directamente a la red del host por lo que la tabla de rutas del pod en el nodo master queda de la siguiente forma:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	155.210.157.254	0.0.0.0	UG	0	0	0	eth0
155.210.157.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
10.42.0.0	0.0.0.0	255.255.255.0	U	0	0	0	cni0
10.42.1.0	10.42.1.0	255.255.255.0	UG	0	0	0	flannel.1
10.42.2.0	10.42.2.0	255.255.255.0	UG	0	0	0	flannel.1

Tabla 5: Tabla de rutas del pod 1 en “modo host”.

Como podemos observar en este caso la tabla de rutas del pod coincide con la Tabla 4, que en el escenario anterior correspondía al nodo master. Esto es lógico porque al trabajar en “modo host” la tabla de rutas del pod es idéntica a la del nodo al que pertenece.

La trama capturada en este escenario entre los nodos master (origen) y worker 2 (destino) presenta la siguiente estructura:

Cabecera externa Ethernet	IP Nodo Master (fuente)	IP Nodo Worker 2 (destino)	MAC Flannel.1 Nodo Master (fuente)	MAC Flannel.1 Nodo Master (fuente)	Cabecera externa UDP	Datos UDP				
						Cabecera VXLAN	MAC Flannel.1 Nodo Master (fuente)	MAC Flannel.1 Nodo Worker 2 (destino)	IP Flannel.1 Nodo Master (fuente)	IP Pod 3 (destino)
										Datos

Figura 35: Trama capturada en la Figura 34.

Observamos que, aunque las interfaces `cni0` y `flannel.1` siguen presentes, el encapsulado es diferente. En este caso no se utiliza la IP origen del pod (dado que no se ha asignado ninguna IP del rango) sino que se añade la IP de la interfaz `flannel.1` del nodo master como origen.

A continuación, estudiamos el caso en que el despliegue únicamente contempla dos nodos. En primer lugar, consideraremos todos los pods pertenecientes a la red Flannel.

- **Dos nodos que contienen tres pods en la red Flannel:**

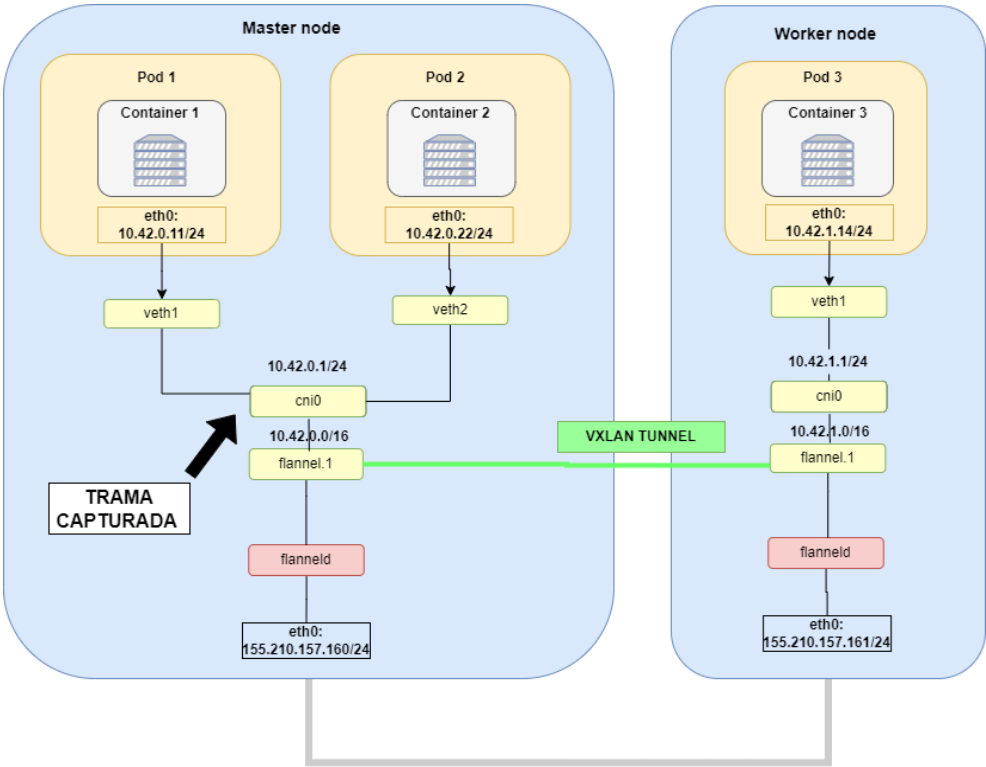


Figura 36: Dos nodos que contienen: tres pods en la red Flannel.

En este caso solo existe un único túnel VXLAN, establecido entre las interfaces `flannel.1` en cada uno. Cada pod del nodo master tienen una dirección IP perteneciente a la red `10.42.0.0/24` y acceso a la interfaz `cni0` del nodo master.

En la interfaz `cni0` capturamos la siguiente trama transmitida entre el pod 1 del nodo master y el pod 2 del mismo nodo:

MAC cni0	MAC Pod 2	IP Pod 1 (fuente)	IP Pod 2 (destino)	Datos
----------	-----------	-------------------	--------------------	-------

Figura 37: Trama capturada en la Figura 36.

La trama se encamina con las direcciones IP de los pod y a través de la interfaz interna cni0, sin necesidad de ser encapsulada sobre UDP.

- **Dos nodos que contienen dos pods en la red host y un pod en la red Flannel:**

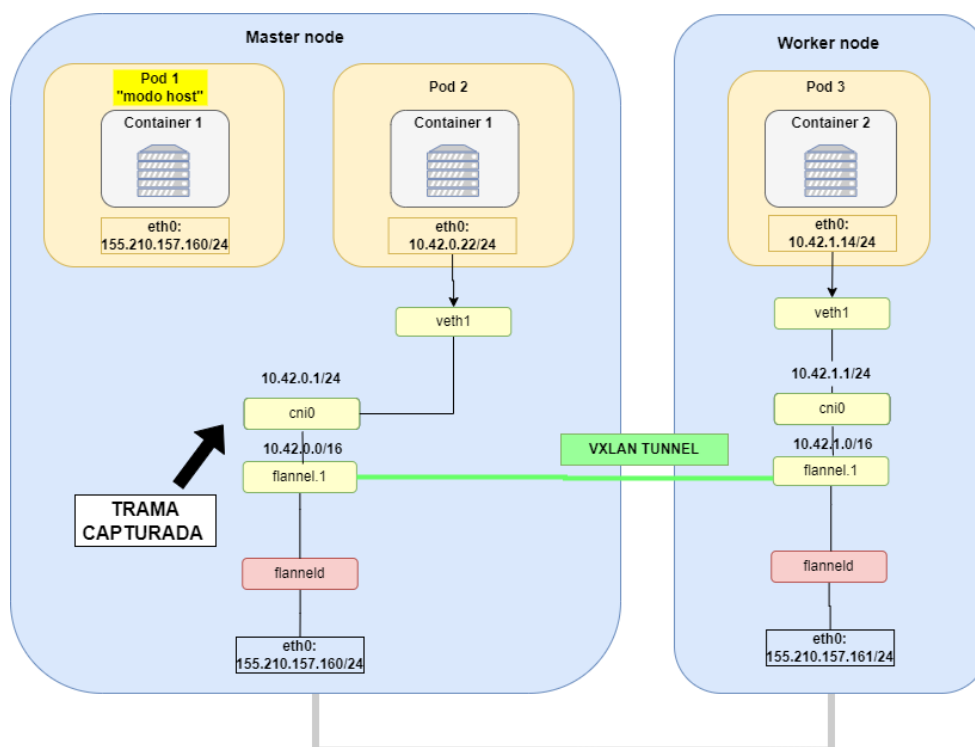


Figura 38: Dos nodos que contienen: dos pods en la red host y un pod en la red Flannel.

Analizamos la trama capturada en la interfaz cni0 con origen pod 1 del nodo master y destino pod 2 en el mismo nodo:

MAC cni0	MAC Pod 2	IP cni0 Nodo Master (fuente)	IP Pod 2 (destino)	Datos
----------	-----------	------------------------------------	-----------------------	-------

Figura 39: Trama capturada en la Figura 38.

En este caso observamos que cambia la IP origen asignando la IP del cni0 del nodo master. Esto se debe a que el pod 1 en “modo host” no tiene IP en la red Flannel luego se encamina con la IP de la interfaz cni0.

Realizamos algunas pruebas de conectividad realizando pings entre los equipos para verificar que las conexiones estudiadas en este apartado teórico efectivamente funcionan.

6.4. Despliegue de puntos de acceso

Desplegamos dos puntos de acceso, por lo que si utilizamos el nodo master para alojar aplicaciones de usuario es suficiente con un único nodo worker y un pod en cada uno de los nodos, donde se ejecuta el software de cada punto de acceso. Para nuestra aplicación es necesario conceder privilegios y que el tipo de red sea de host en cada host. De lo contrario no sería posible acceder a las antenas Wi-Fi. Por lo tanto, la estructura del archivo YAML para la creación de cada pod destinado a un punto de acceso es la siguiente:

```
apiVersion: v1
kind: Pod
metadata:
  name: ap
spec:
  hostNetwork: true
  containers:
  - image: ap_image
imagePullPolicy: Always
securityContext:
  privileged: true
  name: ap
  command: ["sh", "/home/ap"]
nodeSelector:
  disktype: master
```

Mediante el uso de la etiqueta `nodeSelector`¹ asignamos el pod al nodo que nos interesa, en el ejemplo ilustrado se crearía en el nodo master. En la línea de comandos indicamos el script que se debe ejecutar al crear el contenedor, para lanzar el software del punto de acceso. Detallamos dicho script en el Anexo 8.

Creamos los pods ejecutando los siguientes comandos con “kubectl”, que es la herramienta de línea de comandos oficial de Kubernetes que se utiliza para interactuar con el clúster.

```
kubectl create -f /home/proyecto/ap1.yaml
```

```
kubectl create -f /home/proyecto/ap2.yaml
```

-f: utilizado para especificar un archivo de configuración YAML que contiene la descripción del recurso que se va a crear.

“/home/proyecto/ap1.yaml” es la ruta del archivo YAML que contiene la definición del recurso que se desea crear. En este caso, para cada punto de acceso tenemos un archivo YAML porque, como indicamos en el apartado “6.1. Prueba preliminar de

¹ Etiqueta `nodeSelector`: <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

virtualización en Docker”, creamos una imagen Docker diferente para cada punto de acceso.

Al configurar la red como tipo host los pods adquieren la IP del nodo donde se crean de tal forma que la asignación de IPs es la siguiente:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
default	ap2	1/1	Running	0	2m22s	155.210.157.161	worker
default	ap1	1/1	Running	0	2m18s	155.210.157.160	master

Figura 40: Pods creados para los dos APs.

Verificamos el despliegue realizando de nuevo un handover entre los puntos de acceso desplegados en los pods. Capturamos las mismas tramas presentadas en el apartado “6.1. Prueba preliminar de virtualización en Docker”.

7. Conclusiones y Líneas Futuras

7.1. Conclusiones

En este Trabajo de Fin de Grado hemos estudiado las opciones para desplegar automáticamente una red SDWN virtualizada. Para ello, primero hemos estudiado la conveniencia de utilizar la virtualización basada en contenedores Docker para comunicaciones Wi-Fi.

En primer lugar, se ha establecido una comparación en el gasto de CPU de un proceso controlado de transmisión de tramas Wi-Fi en distintas tecnologías: máquinas virtuales y contenedores Docker. Los resultados obtenidos reflejan un mayor gasto de CPU en la máquina virtual concluyendo que el uso de contenedores es más eficiente.

En este trabajo también hemos estudiado el orquestador para contenedores Kubernetes, centrándonos en su aplicación en nuestro escenario particular de red SDWN. Después de analizar la estructura de nuestro escenario de red en los diferentes planos: control, datos y gestión, hemos estudiado las necesidades de despliegue. Tras documentarnos en cómo abordar y cubrir dichas necesidades con la tecnología Docker hemos conseguido desplegar los puntos de acceso y el controlador sobre contenedores Docker, efectuando pruebas de conectividad y movilidad entre dos puntos de acceso.

Por último, hemos estudiado la utilización de Kubernetes para realizar el despliegue, adaptando las necesidades previamente identificadas a esta herramienta de orquestación de contenedores. Hemos establecido la relación entre los contenedores implementados (puntos de acceso y controlador) y los componentes de Kubernetes identificando los nodos de la infraestructura, así como la asignación y despliegue de los pods correspondientes. Después de familiarizarnos con el despliegue de los diferentes elementos que componen el clúster de Kubernetes hemos realizado diferentes pruebas de conectividad, concluyendo que el modelo de red permite la correcta implementación del plano de control. Por último, se ha verificado específicamente el despliegue de nuestros puntos de acceso sobre Kubernetes, comprobando la correcta realización del handoff entre ellos.

En este trabajo hemos identificado la necesidad de eliminar el procedimiento manual de lectura de teclado para el despliegue del plano de control, descrito en el apartado “6.1. Prueba preliminar de virtualización en Docker”. Su eliminación es necesaria para posibilitar la automatización del despliegue. Para ello hemos introducido un sleep de 10 segundos en el código del punto de acceso, dando tiempo para lanzar el controlador sin necesidad de realizar ninguna lectura de teclado.

Por otra parte, en el código del controlador identificamos la necesidad de modificar la ejecución del programa OdinAP, que se realiza en segundo plano. Para la automatización del despliegue con Kubernetes, resulta más adecuado ejecutarlo en

primer plano, para evitar posibles problemas al ejecutar el controlador desde Kubernetes.

Además, identificamos la necesidad de modificación del código de los puntos de acceso con el objetivo de crear una única imagen Docker para todos los puntos de acceso. Esto requeriría la creación de un script cuyo objetivo sería la identificación y asignación de:

- Las direcciones IP de las interfaces de control y de datos de cada punto de acceso.
- Las direcciones MAC de las interfaces inalámbricas que requiere cada punto de acceso.

De esta forma el script asignaría las direcciones necesarias a cada punto de acceso, sin necesidad de configurarlas manualmente ni crear una imagen Docker diferente para cada punto de acceso, como hemos hecho en el apartado “6.1. Prueba preliminar de virtualización en Docker”.

A su vez sería necesario modificar el código del controlador para ser capaces de asignar automáticamente la dirección IP de la interfaz de control de los puntos de acceso. Hasta el momento esta configuración ha sido manual, como indicamos en el apartado “6.1. Prueba preliminar de virtualización en Docker”. Finalmente sería interesante incluir en la modificación del código la unificación de los dos controladores para tener uno único.

7.2. Líneas futuras

Una vez comentadas las conclusiones, los problemas y las necesidades detectadas durante el proyecto, planteamos las siguientes líneas futuras:

- Estudio de gasto de CPU sobre máquina virtual y contenedor para nuevos procesos que transmiten tramas Wi-Fi. Nuestro estudio sobre el consumo de CPU ha permitido valorar la conveniencia de utilizar contenedores de manera genérica. No obstante, se podría analizar con más detalle el impacto sobre los contenedores específicos del diseño (en función de los usuarios y por tanto según los LVAP, según la carga del controlador, etc).
- Modificación del código del punto de acceso con el objetivo de crear una única imagen Docker para todos los puntos de acceso. El nuevo script debe ser capaz de identificar y asignar automáticamente las direcciones IP y MAC involucradas en la configuración de cada punto de acceso.
- Modificación del código del controlador para que no sea necesario conocer a priori las IPs de los nodos donde se ubican los puntos de acceso y para unificar los dos controladores en uno único. Despliegue del software de los controladores en un nodo de Kubernetes y comprobación del funcionamiento de dicho escenario.

- Estudio de automatización del plano de datos sobre Kubernetes, identificando las nuevas necesidades de los elementos que componen dicho plano. En dicho estudio es necesario tener en cuenta la participación de elementos como el servidor DHCP (responsable de la asignación de direcciones) o el NAT.
- Estudio para la creación de una única imagen Docker para todos los puntos de acceso. uso de un único controlador, en lugar de la utilización de dos controladores (Odin Controller y Open Controller).

Anexos

Anexo 1. Conexión remota a la VPN del laboratorio

Para poder trabajar desde nuestro ordenador, ajeno a la red del laboratorio, debemos crear una VPN para ser capaces de acceder a la red del laboratorio y así poder conectarnos a nuestros servidores remotos.

Agregamos una conexión VPN con las siguientes características:

Agregar una conexión VPN

Proveedor de VPN
Windows (integrado)

Nombre de conexión
Conexión VPN

Nombre de servidor o dirección
155.210.158.30

Tipo de VPN
L2TP/IPsec con clave previamente compartida

Clave previamente compartida
[Masked]

Tipo de información de inicio de sesión
Nombre de usuario y contraseña

Nombre de usuario (opcional)
[Masked]

Contraseña (opcional)
[Masked]

☒ Recordar información de inicio de sesión

Guardar Cancelar

Figura 41: Configuración de la VPN.

No mostramos las claves porque es información confidencial.

Tras la configuración la conexión VPN aparece entre nuestras conexiones de red:

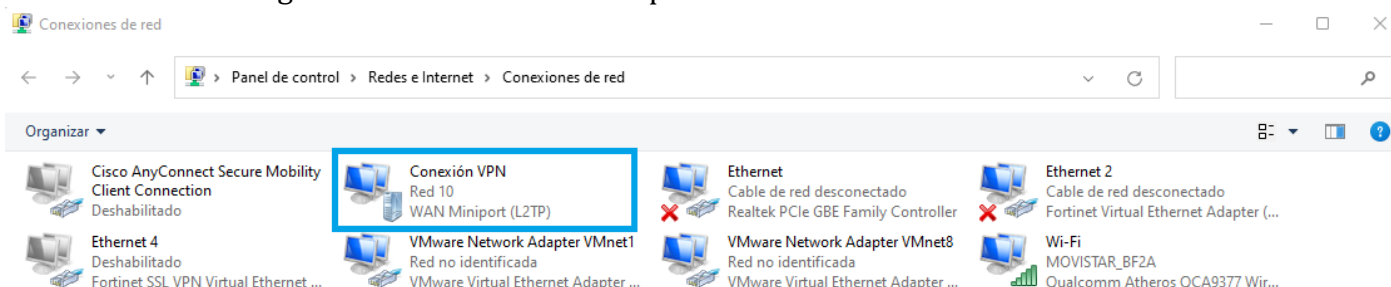


Figura 42: Conexión de red VPN configurada.

Accedemos a Propiedades > Funciones de red > Protocolo de Internet versión 4 (TCP/IPv4) > Propiedades > Opciones avanzadas y desactivamos la primera opción:

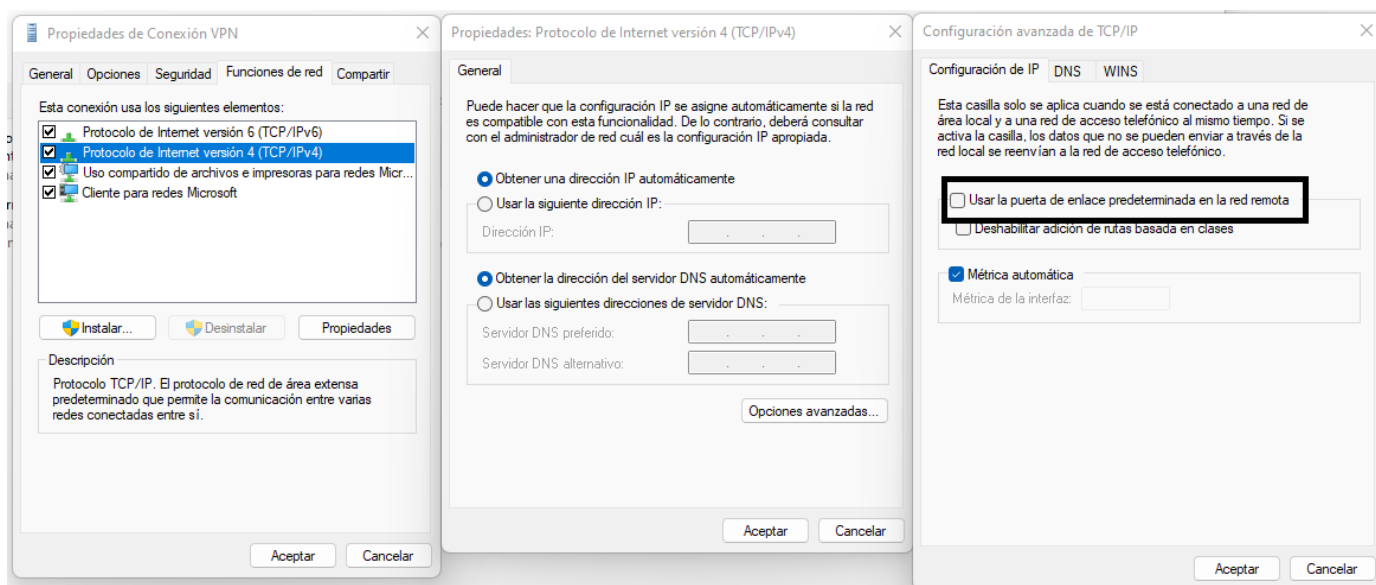


Figura 43: Configuración avanzada de la VPN.

De lo contrario no tendríamos acceso a ninguna conexión Wi-Fi mientras estuviésemos conectados a la VPN. De esta forma podemos tener ambas conexiones funcionando simultáneamente.

Nos conectamos a la VPN configurada y se nos asigna una dirección del laboratorio:

```
Adaptador PPP Conexión VPN:

Sufijo DNS específico para la conexión. . . :
Dirección IPv4. . . . . : 155.210.157.187
Máscara de subred . . . . . : 255.255.255.255
Puerta de enlace predeterminada . . . . . :
```

Figura 44: Dirección IPv4 asignada en la red del laboratorio.

En este caso se nos ha asignado la dirección 155.210.157.187.

Anexo 2: Despliegue inicial en GNS3

En este proyecto en primer lugar estudiamos el siguiente despliegue desarrollado con la herramienta GNS3 y utilizando máquinas virtuales Debian 10 Minimal. Su comprensión nos permitió entender las modificaciones que queremos implementar en nuestro proyecto con el fin de desarrollar nuestra propia solución.

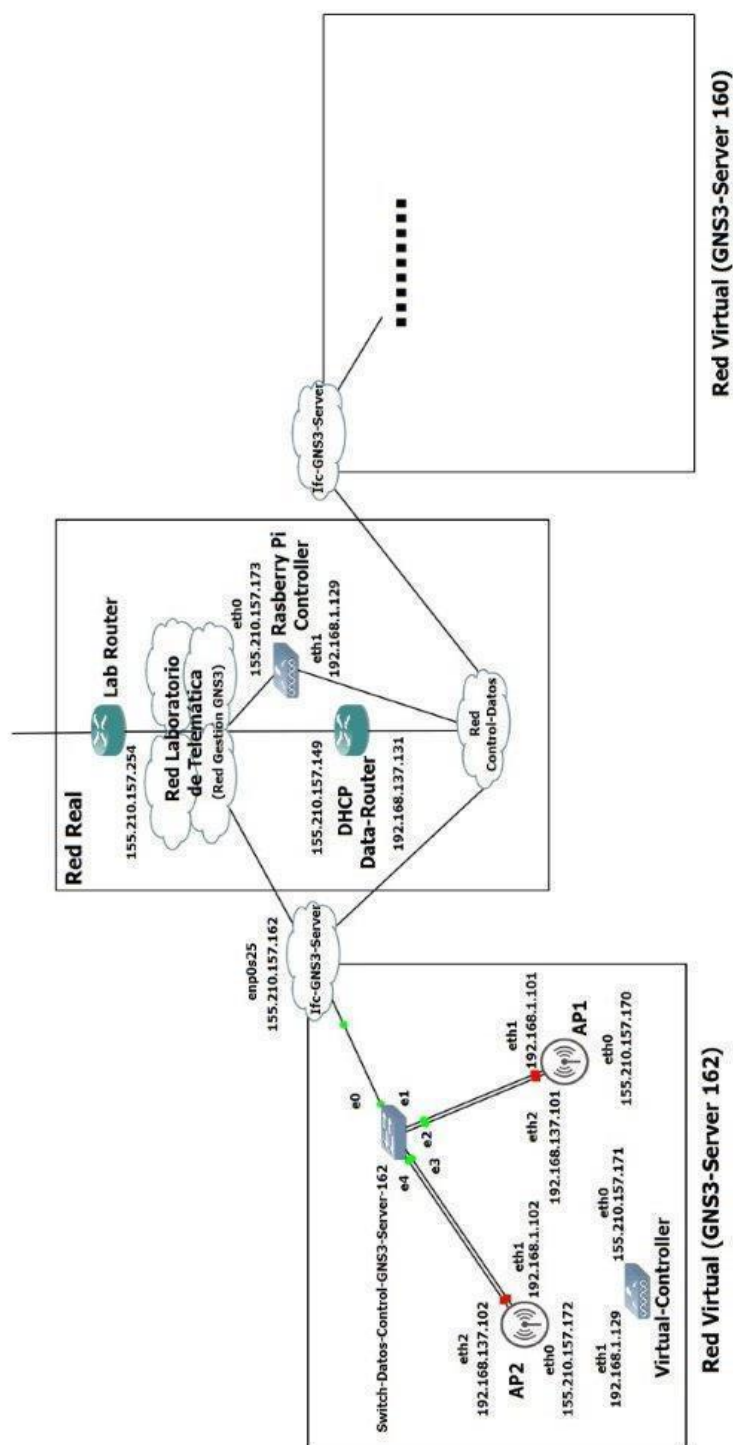


Figura 45: Despliegue inicial en GNS3.

Anexo 3: Graphic Network Simulator-3

La herramienta Graphic Network Simulator-3 (GNS3) consta de una arquitectura cliente-servidor. El cliente de GNS3 presenta la interfaz gráfica de usuario (GUI) y se conecta a uno o varios servidores en función de las necesidades.

A continuación, presentamos algunas características y conceptos de la herramienta:

Tipos de servidores: hay varias opciones para el servidor del software que pueden ser usadas simultáneamente si es necesario.

- **Servidor GNS3 local:** se ejecuta localmente en el mismo PC donde se instala el cliente de GNS3.
- **Servidor remoto:** se utiliza un servidor GNS3 en un equipo remoto, que puede ser Windows o GNU/Linux. Utilizar servidores remotos presenta la ventaja de distribuir la carga de procesamiento entre diferentes máquinas.
- **Servidor sobre máquina virtual GNS3:** se utiliza en caso de que el servidor local o remoto no disponga de hipervisor KVM, necesario para utilizar máquinas emuladas mediante QEMU de forma eficiente. Esta opción incluye un servidor sobre una máquina virtual Ubuntu con hipervisor KVM, ejecutada localmente sobre un software de virtualización como por ejemplo VMWare Workstation.

Template: permite la creación de nuevos equipos virtuales vinculando un archivo de imagen virtual con determinados parámetros de inicio para ser alojados y ejecutados en uno de los posibles servidores mencionados anteriormente.

En este proyecto creamos template de tipo QEMU para ejecutar máquinas virtuales y de tipo Docker para alojar contenedores virtuales Docker, ambas con S.O. Debian.

Apropiación interfaz física: en el apartado de configuración avanzada de un template encontramos la opción de asignarle un recurso físico real del servidor donde es ejecutado. Si asignamos el recurso a una máquina virtual, esta se apropia del mismo y ni la máquina anfitriona (servidor remoto) ni ningún otro dispositivo puede visualizarlo ni apropiarse de él.

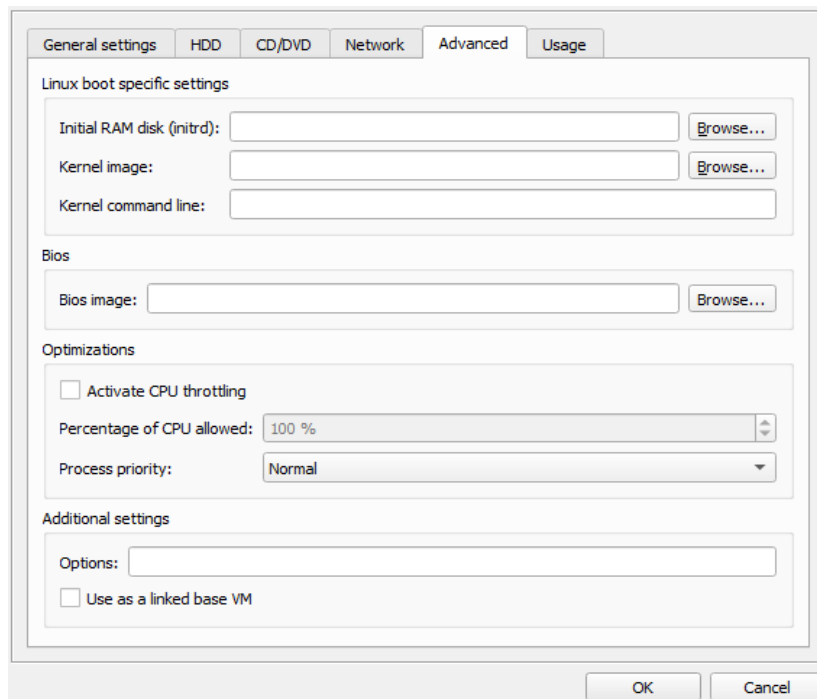


Figura 46. Configuración avanzada del template.

En la casilla “Options” podemos añadir un recurso físico de la siguiente manera:

*-device ich9-usb-ehci1,id=usb,bus=pci.0 -device usb-host,**hostbus=2,hostaddr=4**,vendorid=0x0bda,productid=0x8812,id=hostdev0,bus=usb.0*

De esta forma nos apropiamos del dispositivo que se encuentra en el hostbus número 2 con hostaddr número 4.

Anexo 4: Instalación Docker

En primer lugar, actualizamos la lista de paquetes existente:

```
sudo apt update
```

A continuación, instalamos algunos paquetes de requisitos previos que permitan a apt usar paquetes a través de HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl  
software-properties-common
```

Luego, añadimos la clave de GPG para el repositorio oficial de Docker en nuestro sistema:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |  
sudo apt-key add -
```

Agregamos el repositorio de Docker a las fuentes de APT:

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu focal stable"
```

A continuación, actualizamos el paquete de base de datos con los paquetes de Docker del repositorio recién agregado:

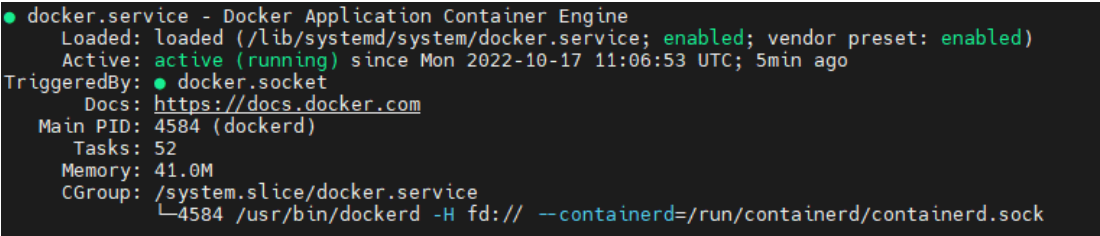
```
sudo apt update
```

Por último, instalamos Docker:

```
sudo apt install docker-ce
```

Con esto, Docker quedará instalado, el demonio se iniciará y el proceso se habilitará para ejecutarse en el inicio. Comprobamos que funciona:

```
sudo systemctl status Docker
```



```
● docker.service - Docker Application Container Engine  
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)  
   Active: active (running) since Mon 2022-10-17 11:06:53 UTC; 5min ago  
 TriggeredBy: ● docker.socket  
     Docs: https://docs.docker.com  
    Main PID: 4584 (dockerd)  
      Tasks: 52  
     Memory: 41.0M  
    CGroup: /system.slice/docker.service  
            └─4584 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Figura 47: Comprobación de Docker.

También podemos ejecutar el siguiente comando, que creará un contenedor que muestra por pantalla el siguiente mensaje: “Hello from Docker!”, seguido de información adicional que indica que la instalación ha sido satisfactoria:

```
docker run hello-world
```

```
proyecto@proyecto-NUC1:~$ sudo docker run hello-world
[sudo] password for proyecto:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:18a657d0cc1c7d0678a3fba8b7eb4918bba25968d3e1b0adebfa71caddbc346
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Figura 48: Ejecución contenedor hello-world.

Anexo 5: Repositorio Docker Hub

En primer lugar, debemos registrarnos accediendo a la siguiente página web desde nuestro navegador: <https://hub.docker.com/signup>

Rellenamos los siguientes campos requeridos:

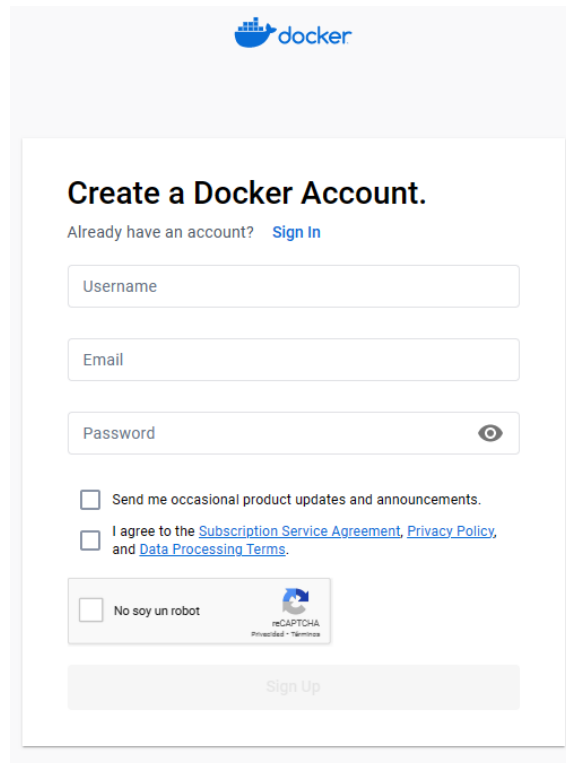
The image shows a web form titled 'Create a Docker Account.' with the Docker logo at the top. Below the title, there is a link 'Already have an account? Sign in'. The form contains three input fields: 'Username', 'Email', and 'Password' (with an eye icon for toggling visibility). Below these fields are three checkboxes: 'Send me occasional product updates and announcements.', 'I agree to the Subscription Service Agreement, Privacy Policy, and Data Processing Terms.', and 'No soy un robot' (which includes a reCAPTCHA logo). At the bottom of the form is a 'Sign Up' button.

Figura 49: Creación de Docker Account.

Después de confirmar nuestra identidad a través de un correo electrónico recibido con tal fin ya podemos acceder a nuestra cuenta a través de la línea de comandos para subir o descargarnos imágenes del repositorio Docker Hub:

`docker login -u username`

```
proyecto@proyecto:~$ sudo docker login -u username
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

Figura 50: Login Docker Account.

Podemos acceder a Docker Hub desde la versión web o instalar Docker Desktop en nuestro PC desde el siguiente enlace:

<https://www.docker.com/products/docker-desktop/>

Para ambas opciones necesitaremos acceder con el usuario previamente creado.

Anexo 6: Código de Python y Matlab utilizado en la evaluación de virtualización

Código de Python ejecutado para la transmisión de las tramas:

```
# -*- coding: utf- 8 -*-
import scapy.all as scapy
import datetime
import time
import sys
from scapy.layers.dot11 import *
from scapy.layers.dot11 import Dot11, LLC, RadioTap,
Dot11Beacon, Dot11Elt, Dot11ProbeResp
from scapy.layers.inet import UDP
from scapy.packet import Raw
from scapy.utils import hexdump, checksum
from datos import *
import pickle5 as pickle

IFACE = 'wlx00c0caa81a2c'          #Interfaz de la tarjeta de red
del transmisor
AP_MAC = '00:c0:ca:a8:1a:2c'       #Direccion mac del transmisor
AP_MAC_2= '00:c0:ca:a8:1a:36'     #Direccion mac del receptor()
rates=0                           #Velocidad de transmision
forma=0
num=1
iter=1
#key = secrets.token_bytes(16)    # Aquí debería coger una clave
asociada al receptor
key =
b'\x01\x0a\x03\x0a\x03\x0a\x03\x0a\x03\x0a\x03\x0a\x03\x0a\x03\x0a'

# Almacenamos el valor de la velocidad de transmision que se ha
elegido
rates=sys.argv[1]
forma=sys.argv[2]
for i in range(1000):

    iter=iter+1
    time.sleep(0.5) # Tiempo de espera antes de transmitir cada
trama
    print("Forma seleccionada: ",forma," . Envío del AMPDU
número: ",num)
    if int(forma) == 1:
        AMPDU, AMPDUint = cifrarNORMAL(key)
        packet = []
        qoscontrol=b'\x00\x00'
        packet.append(RadioTap() / Dot11(type=2, subtype=8,
addr1=AP_MAC_2, addr2=AP_MAC, addr3=AP_MAC)/qoscontrol/AMPDU)
        sendp(packet, iface=IFACE, verbose=False)
    if int(forma) == 2:
        packet = []
        qoscontrol = b'\x80\x00'
```

```

    Hdr, mpduCi_Hex, mpduCi, ps, xs = cifrarCRT()    # Cifrado de
la trama
    packet.append(RadioTap() / Dot11(type=2, subtype=0,
addr1=AP_MAC_2, addr2=AP_MAC,addr3=AP_MAC) / qoscontrol /
mpduCi_Hex)
    sendp(packet, iface=IFACE, verbose=False)    # Transmision de
la trama
    time2 = datetime.now().time().strftime('%H:%M:%S:%f')    #
Marca temporal de la transmision
    print("Enviado paquete AMPDU con addr1: ",AP_MAC_2," addr2:
",AP_MAC," y addr3: ",AP_MAC," Iteracion: ",iter,"      ", time2)

    if int(forma) == 3:
        calc_primos(3,128)

```

A continuación, indicamos el código de Matlab para obtener el número de paquetes transmitidos cada segundo. En cada fila del fichero de entrada se indica el minuto y el segundo en que ha sido transmitido cada paquete. En el fichero de salida indicamos cada segundo y el número de paquetes transmitidos en dicho segundo:

```

M = importdata('p3_16_TX.txt');
m0 = M(1,1);    % Minuto inicial
s0 = M(1,2);    % Segundo inicial
c = length(100);    % Vector que creo. Cada posicion es para un
segundo.
f = length(M);    % Numero de segundos
ind=1;

for i=2:f    % Bucle para recorrer las filas
    valor = c(ind);
    m = M(i,1);
    s = M(i,2);

    if (m==m0 && s==s0) % Una trama mas en el mismo segundo
        c(ind)=valor+1; % Sumo un paquete transmitido
    else
        % Cambio de segundo o de minuto
        ind=ind+1;    % Cambio al siguiente segundo
        valor=0;
        c(ind) = valor+1;    % Sumo un paquete transmitido
        m0=m;    % Actualizo referencias
        s0=s;
    end
end
c=c';    % Vector columna

paquetes_totales = sum(c)    % Comprobacion

Segundos = (1:length(c))';
Paquetes = c;
T = table(Segundos, Paquetes)

writetable(T,'tabledata_p3_16.txt');

```

Anexo 7: Creación de imágenes Docker

Detallamos los componentes incluidos en cada imagen Docker creada en este trabajo y presentamos los comandos ejecutados para su creación.

Imagen prueba de virtualización: incluye Python 3.7.3, instalador de paquetes para Python Pip 18.1, biblioteca Scapy, 2.4.5, driver de las antenas RTL8812AU y código necesario para inyectar tráfico, incluido en el Anexo 5.

Presentamos los comandos utilizados para crear la imagen Docker utilizada en la prueba de virtualización.

En primer lugar, descargamos la imagen padre Debian 10:

```
sudo docker search debian
sudo docker pull dokken/debian-10
```

A continuación, creamos un contenedor Docker a partir de la imagen padre:

```
sudo docker run -it dokken/debian-10 bash
```

Actualizamos los paquetes instalados en el contenedor Debian:

```
sudo apt update
sudo apt upgrade
```

Instalamos Python, el instalador de paquetes para Python Pip 18.1 y la biblioteca Scapy:

```
sudo apt-get install python3
sudo apt-get install python3-pip
sudo apt-get install python3-scapy
```

Instalamos el driver de las antenas RTL8812AU:

```
sudo apt-get install dkms
git clone -b v5.2.20 https://github.com/aircrack-ng/rtl8812au.git
cd rtl8812au/
sudo ./dkms-install.sh
```

Salimos del contenedor Docker:

```
exit
```

Incluimos el código necesario para inyectar tráfico:

```
docker cp RUTA_LOCAL_CODIGO_INYECCION_TRAFICO
NOMBRE_CONTENEDOR:RUTA_DEL_CONTENEDOR
```

Creamos la nueva imagen a partir del contenedor:

```
sudo docker commit -m " Imagen Docker utilizada en la
prueba de virtualización " NOMBRE_CONTENEDOR
NOMBRE_USUARIO_DOCKER_HUB/prueba_virtualizacion_image
```

Subimos la imagen creada al repositorio Docker Hub:

```
sudo docker login -u NOMBRE_USUARIO_DOCKER_HUB

sudo docker push
NOMBRE_USUARIO_DOCKER_HUB/prueba_virtualizacion_image
```

Imagen del punto de acceso: incluye driver de las antenas RTL8812AU, OpenvSwitch, HostAPD, Macchanger, compilador GCC y el software del punto de acceso, ejecutado como código C.

- **OpenvSwitch:** implementación de código abierto de un conmutador multicapa virtual distribuido . El objetivo principal de OpenvSwitch es proporcionar una pila de conmutación para entornos de virtualización. Al mismo tiempo admite múltiples protocolos y estándares.
- **HostAPD:** es un demonio de software que permite a una computadora con una tarjeta de red inalámbrica actuar como un punto de acceso Wi-Fi.
- **Macchanger:** herramienta de software diseñada para cambiar la dirección MAC (Media Access Control) de una interfaz de red.

A continuación, presentamos los comandos ejecutados para crear la imagen Docker del punto de acceso:

En primer lugar, descargamos la imagen padre Debian 10:

```
sudo docker search debian
sudo docker pull dokken/debian-10
```

A continuación, creamos un contenedor Docker a partir de la imagen padre:

```
sudo docker run -it dokken/debian-10 bash
```

Actualizamos los paquetes instalados en el contenedor Debian:

```
sudo apt update
sudo apt upgrade
```

Instalamos el driver de las antenas RTL8812AU:

```
sudo apt-get install dkms
git clone -b v5.2.20 https://github.com/aircrack-ng/rtl8812au.git
cd rtl8812au/
sudo ./dkms-install.sh
```

Instalamos OpenvSwitch:

```
sudo apt-get install openvswitch-switch
```

Instalamos HostAPD:

```
sudo apt-get -y install hostapd
```

Instalamos el compilador GCC:

```
sudo apt-get install build-essential
```

Salimos del contenedor:

```
exit
```

Incluimos el código del punto de acceso:

```
docker cp RUTA_LOCAL_CODIGO_ACCESS_POINT  
NOMBRE_CONTENEDOR:RUTA_DEL_CONTENEDOR
```

Creamos la nueva imagen a partir del contenedor:

```
sudo docker commit -m " Imagen Docker utilizada para el  
punto de acceso " NOMBRE_CONTENEDOR  
NOMBRE_USUARIO_DOCKER_HUB/access_point_image
```

Subimos la imagen creada para el punto de acceso al repositorio Docker Hub:

```
sudo docker login -u NOMBRE_USUARIO_DOCKER_HUB  
  
sudo docker push  
NOMBRE_USUARIO_DOCKER_HUB/access_point_image
```

Imagen del controlador: incluimos OpenvSwitch y el software del controlador Odin, que se ejecuta sobre Java.

Por último, presentamos los comandos utilizados para crear la imagen Docker del controlador:

En primer lugar, descargamos la imagen padre Debian 10:

```
sudo docker search debian  
sudo docker pull dokken/debian-10
```

A continuación, creamos un contenedor Docker a partir de la imagen padre:

```
sudo docker run -it dokken/debian-10 bash
```

Actualizamos los paquetes instalados en el contenedor Debian:

```
sudo apt update  
sudo apt upgrade
```

Instalamos OpenvSwitch:

```
sudo apt-get install openvswitch-switch
```

Salimos del contenedor:

```
exit
```

Incluimos el código del controlador:

```
docker cp RUTA_LOCAL_CODIGO_ODIN_CONTROLLER  
NOMBRE_CONTENEDOR:RUTA_DEL_CONTENEDOR
```

Creamos la nueva imagen a partir del contenedor:

```
sudo docker commit -m " Imagen Docker utilizada para el  
punto de acceso " NOMBRE_CONTENEDOR  
NOMBRE_USUARIO_DOCKER_HUB/controller_image
```

Subimos la imagen creada para el controlador al repositorio Docker Hub:

```
sudo docker login -u NOMBRE_USUARIO_DOCKER_HUB  
  
sudo docker push NOMBRE_USUARIO_DOCKER_HUB/controller_image
```

Anexo 8: Script punto de acceso

En el primer punto de acceso ejecutamos los siguientes comandos (en el segundo punto de acceso ejecutamos lo mismo, teniendo en cuenta que los nombres de las interfaces inalámbricas cambian):

```
1  ifconfig wlx00c0caa81a2c up
2  ifconfig wlx00c0caa4737c up
3  ifconfig wlx00c0caa81a2c down
4  macchanger wlx00c0caa81a2c -s
5  macchanger wlx00c0caa81a2c -b -m 00:11:11:11:11:11
6  ifconfig wlx00c0caa81a2c up

7  echo 1 > /proc/sys/net/ipv4/conf/eth1/arp_announce
8  echo 1 > /proc/sys/net/ipv4/conf/eth2/arp_announce
9  echo 2 > /proc/sys/net/ipv4/conf/eth1/arp_ignore
10 echo 2 > /proc/sys/net/ipv4/conf/eth2/arp_ignore

11 cd /OdinAPLVAP-GNS3
12 ./start.sh
```

- **Líneas 1 y 2:** encendemos las dos antenas del punto de acceso.
- **Línea 3:** apagamos la antena principal.
- **Líneas 4:** mostramos la dirección MAC de la antena principal.
- **Línea 5:** modificamos la dirección MAC de la antena principal.
- **Línea 6:** encendemos la antena principal.
- **Líneas 7 y 8:** establecemos el parámetro "arp_announce" a 1 en las interfaces de datos y de control. El valor 1 indica que el sistema enviará respuestas ARP con la dirección MAC de la interfaz de salida en lugar de la dirección MAC de origen original. Esta configuración se utiliza en casos en los que la máquina responde en nombre de otras máquinas virtuales o interfaces de red.
- **Líneas 9 y 10:** este comando establece el parámetro "arp_ignore" a 2 para las interfaces de datos y control. El valor 2 indica que el sistema ignorará las solicitudes ARP que lleguen a la interfaz si la dirección IP de destino no está configurada en esa interfaz.
- **Líneas 11 y 12:** ejecutamos el script "start.sh" ubicado en el directorio "/OdinAPLVAP-GNS3" para poner en marcha el punto de acceso.

Anexo 9: Instalación de K3s

A continuación, detallamos la instalación de K3s en los diferentes nodos.

En el nodo master ponemos en marcha la distribución de Kubernetes con el siguiente comando:

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="server  
--node-ip 155.210.157.160 --node-name master --token  
wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy  
5OZ/5" sh -s
```

Este comando descarga el script de instalación de K3s y lo ejecuta en un solo paso, instalando un nodo K3s en el servidor especificado con una dirección IP y un nombre de nodo, utilizando un token de acceso específico.

En cada nodo agente ejecutamos el siguiente comando, indicando su propia dirección IP, nombre y el token del clúster:

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="agent  
--server=https://155.210.157.160:6443 --node-ip  
155.210.157.161 --node-name worker --token  
wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy  
5OZ/5" sh -s
```

Bibliografía

- [1] Cisco. (2019). Cisco Visual Networking Index: Forecast and Trends, 2017–2022. Obtenido de <https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf>
- [2] Cisco. (2019). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022. Obtenido de <http://media.mediapost.com/uploads/CiscoForecast.pdf>
- [3] Saldana, J., Munilla, R., Eryigit, S., Topal, O., Ruiz Mas, J., Fernández Navajas, J., & Sequeira, L. (2018). Unsticking the Wi-Fi Client: Smarter Decisions Using a Software Defined Wireless Solution. IEEE Access.
- [4] Foundation, O. N. (2014). SDN Architecture. Open Networking. Obtenido de https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf
- [5] Roca, D. G. (2021). Gestión y análisis de un escenario virtualizado de control para una red inalámbrica basado en tecnología Wi-Fi. Trabajo Fin de Grado, Universidad de Zaragoza.
- [6] Saldana, J., Ruiz Mas, J., Fernández Navajas, J., Salazar, L., Javaudin, J., Bonnamy, J., & Le Dizes, M. (2017). Attention to Wi-Fi Diversity: Resource Management in WLANs with Heterogeneous APs. IEEE Access.
- [7] Maurya, R. (Febrero de 2021). Obtenido de <https://www.how2shout.com/tools/8-free-best-open-source-bare-metal-hypervisors-foss.html>
- [8] Arrington, E. (Mayo de 2019). Obtenido de <https://akfpartners.com/growth-blog/vms-vs-containers>
- [9] Overflow, S. (2022). 2022 Developer Survey. Obtenido de <https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-other-tools>
- [10] Microsoft. (Noviembre de 2022). Obtenido de <https://learn.microsoft.com/es-es/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries>
- [11] The Kubernetes Authors. (s.f.). Obtenido de <https://kubernetes.io/>
- [12] The Kubernetes Authors. (s.f.). Obtenido de <https://kubernetes.io/docs/concepts/workloads/pods/>
- [13] The Kubernetes Authors. (s.f.). Obtenido de <https://kubernetes.io/docs/concepts/overview/components/>

- [14] Jain, H. (Junio de 2020). Obtenido de <https://medium.com/devops-and-sre-learning/quick-learning-guide-for-kubernetes-part-ii-f5336d7993>
- [15] The CNI Authors. (s.f.). Obtenido de <https://www.cni.dev/>
- [16] Pathak, S. (2023). CIVO. Obtenido de <https://www.civo.com/blog/k8s-vs-k3s>
- [17] Buil, M. (s.f.). Flannel. Obtenido de <https://github.com/flannel-io/flannel>
- [18] Vallim, M. T. (s.f.). Flannel. Obtenido de <https://mvallim.github.io/kubernetes-under-the-hood/documentation/kube-flannel.html>