ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SERVICIOS BASADOS EN UAVs Y VIRTUALIZACIÓN

SERVICIOS DE RED BASADOS EN CONTENEDORES

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN TELECOMUNICACIONES

ANDRÉS LENIN YAZÁN ENDARA
andres.yazan@epn.edu.ec

DIRECTOR: PHD. CHRISTIAN JOSÉ TIPANTUÑA TENELEMA
christian.tipantuna@epn.edu.ec

DMQ, octubre 2023

CERTIFICACIONES

Yo, YAZÁN ENDARA ANDRÉS LENIN declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

YAZÁN ENDARA ANDRÉS LENIN

Certifico que el presente trabajo de integración curricular fue desarrollado por YAZÁN ENDARA ANDRÉS LENIN, bajo mi supervisión.

PhD. CHRISTIAN JOSE TIPANTUÑA TENELEMA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

YAZÁN ENDARA ANDRÉS LENIN
PHD. CHRISTIAN JOSÉ TIPANTUÑA TENELEMA

DEDICATORIA

A mis padres, Pepita y Oscar, quienes me supieron guiar a lo largo de este camino. Gracias a su apoyo y sacrificio, estoy aquí.

A mis hermanos, Stefy y Stalin, quienes siempre estuvieron a mi lado brindándome fuerzas para salir adelante.

AGRADECIMIENTO

Agradezco a toda mi familia por el apoyo recibido en el camino que me trajo hasta aquí. En especial, a mis abuelitas Carmen y Esthela. Gracias a ustedes por estar siempre a mi lado y apoyarme. No fue fácil, pero siempre hay que seguir adelante.

De igual manera, agradezco a mis maestros, quienes me han inculcado conocimientos, tanto en lo profesional como en lo personal. Nunca dejaré de aprender gracias a sus enseñanzas.

Un agradecimiento especial al Dr. Christian Tipantuña por su ayuda en el desarrollo de este proyecto. A pesar de todo, siempre se puede mejorar.

También, quiero expresar mi agradecimiento a la Escuela Politécnica Nacional, que me vio crecer como profesional y persona.

Finalmente, mi gratitud a todos aquellos que me han ayudado en momentos de duda y cuyos consejos han iluminado mi camino. Les envío mis mejores deseos y gratitud.

ÍNDICE DE CONTENIDO

RESUMEN

En los últimos años, la virtualización de funciones de red ha sido fundamental para el despliegue de servicios de alto rendimiento al permitir la integración de funciones y servicios de red previamente limitados a dispositivos físicos. Esto ofrece múltiples servicios virtualizados al usuario. Actualmente, la forma predominante de implementar servicios y funciones de red virtualizadas es a través de máquinas virtuales. Aunque esta virtualización brinda varios beneficios en comparación con las implementaciones físicas, también presenta desventajas en términos de rendimiento en comparación con un despliegue no virtualizado. En este contexto, en los últimos años se han introducido varias tecnologías alternativas de virtualización, entre ellas los contenedores.

Los contenedores permiten crear entornos virtuales aislados en una misma infraestructura física, utilizando funciones propias del sistema operativo, como los namespaces y cgroups. A este respecto, en el presente trabajo de integración curricular se implementa el despliegue de servicios y funciones de red utilizando contenedores, centrándose en una solución basada en contenedores Docker. Para ello, se inicia revisando los conceptos generales de los contenedores, su arquitectura y los tipos de despliegue utilizando la tecnología Docker. Posteriormente, se diseña e implementa una solución multicontenedor utilizando una herramienta de Docker llamada Docker Compose, para el despliegue conjunto de los servicios de red implementados mediante Docker previamente. Finalmente, se lleva a cabo un análisis de la virtualización de servicios de red en el entorno de contenedores, para comprender cómo pueden ser utilizados de manera efectiva para crear y gestionar entornos aislados que faciliten la implementación de servicios de red, optimizando así los recursos y garantizando un despliegue eficiente y seguro.

PALABRAS CLAVE: Virtualización, máquinas virtuales, contenedor, docker, docker compose.

ABSTRACT

In recent years, the virtualization of network functions has been fundamental to deploying high-performance services by enabling the integration of network functions and services previously limited to physical devices. This offers multiple virtualized services to the user. Currently, virtual machines are the predominant way to deploy virtualized network functions and services. Although this virtualization provides several benefits compared to physical deployments, it also has disadvantages in terms of performance compared to non-virtualized deployment. In this context, several alternative virtualization technologies have been introduced in recent years, including containers. Containers allow the creation of isolated virtual environments in the same physical infrastructure, using the operating system's functions, such as namespaces and cgroups. In this regard, this curricular integration work implements the deployment of network services and functions using containers, focusing on a solution based on Docker containers. To do so, we start by reviewing the general concepts of containers, their architecture, and the types of deployment using Docker technology. Subsequently, a multi-container solution is designed and implemented using a Docker tool called Docker Compose, which allows the joint deployment of network services previously implemented using Docker. Finally, an analysis of the virtualization of services and network functions in the container environment is carried out, in order to understand how containers, especially Docker, can be used effectively to create and manage isolated environments that facilitate the deployment of services. network, thus optimizing resources and ensuring efficient and secure deployment.

KEYWORDS: Virtualization, virtual machines, container, docker, docker-compose.

1 INTRODUCCIÓN

Actualmente, la demanda creciente de servicios, aplicaciones y recursos de red por parte de los usuarios finales ha generado limitaciones en la capacidad de los proveedores de servicios para satisfacer estas necesidades debido a la escasez de recursos de hardware necesarios para escalar proporcionalmente a las demandas. Por lo tanto, los proveedores de servicios han tenido que adoptar nuevas tecnologías para mantenerse al día con las demandas actuales, maximizar la eficiencia de los recursos y ofrecer una alta calidad de servicio (QoS: Quality of Service) a los usuarios finales. En este contexto, las tecnologías de virtualización desempeñan un papel fundamental en las actuales tecnologías de la información.

Aunque existen tecnologías de virtualización como las máquinas virtuales (VM: Virtual Machines) que proporcionan servicios virtualizados, estas presentan importantes problemas de rendimiento y eficiencia de recursos. Por esta razón, las tecnologías de virtualización basadas en contenedores se han convertido en la mejor opción para superar las desventajas y limitaciones que presentan las VMs al virtualizar servicios [1].

Las tecnologías de virtualización basadas en contenedores aprovechan las características del sistema operativo para ofrecer servicios virtualizados altamente eficientes, eliminando la necesidad de capas adicionales y trabajando directamente sobre la infraestructura de software nativa de un dispositivo. Estas características incluyen los llamados “namespaces” y “cgroups”, que son componentes del kernel de un sistema operativo y se utilizan para proporcionar un entorno aislado e independiente dentro de la infraestructura nativa en la que se ejecutan [2].

A este respecto, el objetivo del presente trabajo de integración curricular es describir, implementar y analizar soluciones de servicios de red de basadas en contenedores Docker, una tecnología que ha ganado gran impulso desde su lanzamiento. Para lograr esto, el trabajo se estructura de la siguiente manera: En el Capítulo I se proporciona una breve descripción de los conceptos y características necesarios para comprender la estructura de un contenedor. Además, se realiza una comparación entre las tecnologías basadas en máquinas virtuales (VMs) y las soluciones basadas en contenedores. También se aborda el concepto de microservicio y su relación con las tecnologías de contenerización. En el Capítulo II se profundiza en los conceptos, herramientas y procesos utilizados para diseñar e implementar los servicios de red utilizando contenedores Docker. En el Capítulo III se analizan los resul-

tados obtenidos en el Capítulo II para evaluar el funcionamiento y rendimiento del sistema de contenerización implementado. A continuación, se establecen los objetivos principales, los objetivos específicos y el alcance del presente trabajo.

1.1 OBJETIVO GENERAL

Desplegar servicios de red implementados a través de contenedores.

1.2 OBJETIVOS ESPECÍFICOS

- ❒ Describir los fundamentos teóricos de los entornos de contenerización.
- ❒ Diseñar una solución para el despliegue de servicios de red basada en la tecnología de contenerización Docker.
- ❒ Implementar servicios de red sobre la tecnología de contenerización Docker.
- ❒ Analizar las implementaciones de los servicios de red.

1.3 ALCANCE

En el presente trabajo de integración curricular se realiza la implementación de servicios de red sobre la tecnología de contenerización Docker. Inicialmente, se describirán los conceptos y características asociadas a las tecnologías basadas en contenedores, haciendo énfasis en la tecnología Docker. Posteriormente, se describirán cada uno de los servicios de red a desplegar y se realizará un breve resumen de los microservicios, puntualizando su correlación con las tecnologías de contenerización. Finalizada la parte teórica, se realizará el análisis de los requerimientos de los servicios, además de profundizar en el entorno Docker, para implementar estos servicios en dos escenarios.

El primer escenario considera la implementación individual de cada servicio en contenedores Docker. Luego, se procede a una implementación conjunta utilizando Docker CLI. Todo montado sobre una placa de desarrollo Raspberry Pi. En el segundo escenario, se implementa una solución conjunta para los servicios contenerizados en el escenario anterior. En esta solución, se utiliza Docker Compose como herramienta para construir y coordinar los servicios, montados nuevamente en una placa Raspberry Pi. Además, para ambos escenarios se utilizarán múltiples dispositivos de cómputo, como laptops, para que realicen la función de clientes de estos servicios.

Una vez desplegados cada uno de los escenarios descritos, se procederá a realizar pruebas de rendimiento basadas en parámetros como: uso de CPU, memoria y parámetros de red.

Finalmente, se analizan los resultados obtenidos de las soluciones implementadas, para concluir si los servicios contenerizados presentan beneficios o desventajas, con respecto a las soluciones tradicionales.

1.4 MARCO TEÓRICO

En esta sección se presentan los conceptos básicos sobre la virtualización de servicios basadas en contenedores. Para lo cual, se parte con la descripción de los conceptos básicos asociados a un sistema operativo, en particular, sobre las definiciones, como el kernel, *namespaces* y *cgroups*. Igualmente, se abarcan los conceptos de virtualización sobre VMs, para entender el funcionamiento y características de un entorno virtualizado basado en contenedores. Finalmente, se presenta una descripción breve de los servicios de red, mismos que se presentarán en detalle en las secciones posteriores.

1.4.1 Sistema Operativo y Kernel

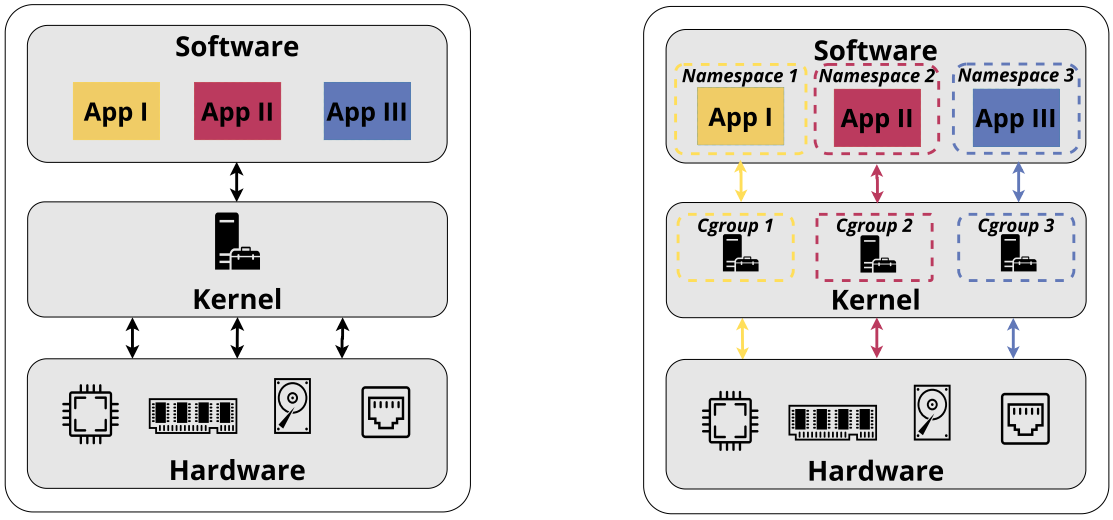
El sistema operativo es la base de cualquier sistema computacional, ya que proporciona los recursos y características necesarios para la ejecución de cualquier software o programa de computadora. Esto se logra a través de un conjunto de módulos, entre los cuales se encuentra el kernel. El kernel es el componente principal de un sistema operativo, encargado de administrar los recursos del hardware, como el almacenamiento, memoria, Unidad Central de Proceso (CPU: Central Processing Unit), red, entre otros; para satisfacer los requisitos del software. De esta manera, el kernel actúa como una interfaz que permite la comunicación entre hardware y software [3]. El Kernel de Linux cuenta con características para la administración de recursos, entre ellas se encuentran los *namespaces* y *cgroups*, introducidos en 2002 en la versión de Linux 2.4.19 [1].

1.4.1.1 Namespace

Son una característica por defecto de Linux, que manejan las propiedades de aislamiento de los contenedores permitiendo obtener un nivel de abstracción en los procesos dentro del kernel de Linux. En este sentido, los *namespaces* permiten ver los recursos del sistema, como sistema de archivos, red, procesos, entre otros, y posteriormente segmentarlos de manera virtual para una aplicación. Esto permite aislar los recursos computacionales por procesos o grupos de procesos para que tengan una visión única del sistema operativo [1].

1.4.1.2 Control Groups (cgroups)

Al igual que los *namespaces*, los *cgroups* son otra característica de Linux, que limitan y controlan la cantidad de recursos que se utiliza para un proceso en particular [4]. Los *namespaces* y los *cgroups* funcionan conjuntamente para aislar procesos y gestionar los recursos asignados a cada uno de ellos [4]. En la Figura 1.1 se presenta una comparación entre la arquitectura computacional tradicional basada en kernel y la arquitectura basada en *namespaces* y *cgroups*, para la administración de los recursos del hardware y del software que se proporcionan a las aplicaciones. Aunque el enfoque para el despliegue en el presente trabajo se basa en Linux, se puede abstraer los mismos conceptos para incorporarlos en los sistemas operativos más usados como Windows y MacOS [5].



(a) Arquitectura computacional basada en kernel.

(b) Arquitectura basada en *namespaces* y *cgroups*.

Figura 1.1: Comparación de arquitecturas computacionales, basado en [1], [3], [4].

1.4.2 Virtualización

Virtualizar es un proceso o tecnología que permite segmentar los recursos de software y hardware de una arquitectura física, para el despliegue de múltiples recursos dedicados en entornos virtuales para procesos o aplicaciones. Por lo tanto, estos entornos alojados sobre una arquitectura física son una disposición lógica de hardware y software. Las tecnologías de virtualización principales son las VMs que permiten ejecutar sistemas operativos completos independientes del sistema operativo base. En tanto que las tecnologías de contenerización permiten virtualizar únicamente aplicaciones sobre sistema operativo anfitrión. A continuación, se detalla cada una de estas tecnologías de virtualización [6].

1.4.2.1 Máquinas Virtuales

Las máquinas virtuales permiten desplegar aplicaciones o servicios en entornos virtuales con su propio sistema operativo y sistema de archivos, independiente de la arquitectura base. Para lograr esta forma de virtualización, se implementa una capa de software adicional sobre la arquitectura computacional convencional, específicamente sobre el sistema operativo subyacente, llamada hipervisor[1]. Además, al sistema físico sobre el cual se ejecuta el hipervisor se le denomina host, mientras que a las VMs creadas sobre esta capa de software se las considera como las guests del host [5].

Este enfoque basado en hipervisor hace que el rendimiento general decaiga, especialmente para entornos de alta demanda de recursos. Dado que esta capa está presente en todas las tecnologías de virtualización tradicionales, el rendimiento no varía de tecnología en tecnología, por lo que la capa del hipervisor afecta negativamente en cualquier caso de su implementación, como se menciona en [4]. Algunos ejemplos de tecnologías de virtualización basadas en hipervisor son VMware (VMware), VirtualBox (Oracle), Xen (Citrix) o Hyper-V (Microsoft).

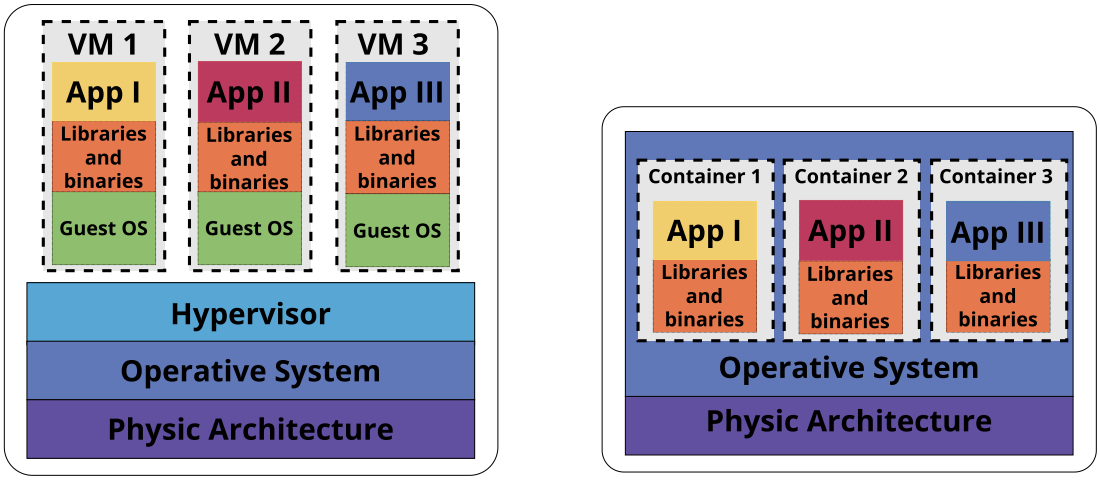
1.4.2.2 Contenedores

Un contenedor es una tecnología de virtualización a nivel del kernel del sistema operativo, que permite ejecutar aplicaciones y servicios en entornos aislados y portátiles sobre un mismo sistema físico. Los contenedores utilizan los *namespaces* y *cgroups* para aislar y gestionar los recursos del sistema [8]. En la práctica, los procesos realizados por los contenedores son más eficiente que las soluciones basadas en hipervisor, ya que eliminan dicha capa de virtualización y se ejecutan directamente sobre el mismo kernel de host, como se puede observar en la Figura 1.2, donde se presenta esta comparación entre la arquitectura de virtualización basada en hipervisor versus la basada en contenedores.

Aparte de los *namespaces* y *cgroups*, los contenedores hacen uso de otros elementos para el despliegue de las aplicaciones, como las imágenes de contenedores, las cuales son los componentes básicos para la virtualización.

---

[1] **Hipervisor:** Es el software encargado de virtualizar y administrar los recursos físicos del sistema, como CPU, memoria RAM, almacenamiento, etc. y distribuirlos a las necesidades de las VMs [7].



(a) Arquitectura basada en Hipervisor.

(b) Arquitectura basada en contenedores.

Figura 1.2: Comparación de arquitecturas de virtualización, basado en [5].

1.4.2.3 Imagen

Una imagen, en el mundo de los contenedores, es el producto final del envolvimiento de una aplicación o servicio con sus respetivas dependencias, binarios y/o metadatos asociados. Una imagen no es ejecutable, ya que una vez empaquetada la aplicación no es accesible sino a través de su instanciación[2] [5]. Este proceso crea un contenedor y ejecuta la aplicación aislado del sistema operativo subyacente, como de los demás contenedores, por lo que se le puede considerar como un proceso de sandbox[3] [11]. En la Figura 1.3 se muestra el proceso de contenerización, donde la aplicación y sus dependencias son empaquetadas (1) en una imagen, para luego ser instanciada (2) y crear un contenedor.

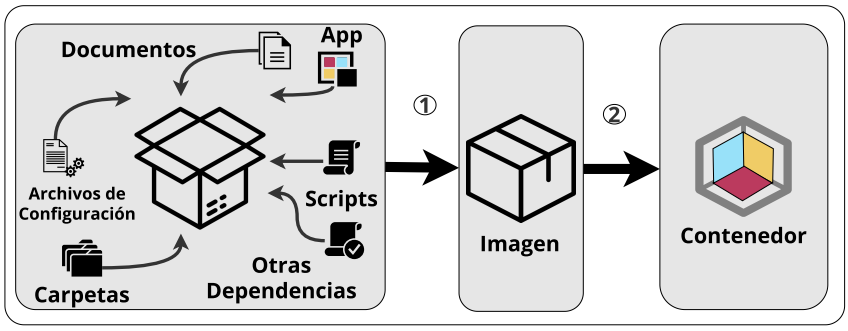


Figura 1.3: Proceso de contenerización: Empaquetado e instanciación, basado en [12].

[2] **Instanciar:** Es la creación de un objeto a partir de una clase. En programación, un objeto es una entidad abstracta que comprende funciones, métodos y características, a diferencia de una clase que es una descripción de las características de un objeto. Por lo que, al instanciar un objeto este hereda las características definidas en la clase y puede tener atributos propios [9].

[3] **Sandbox:** Es una tecnología que hace uso de un entorno aislado del resto del sistema operativo, ejecutando las aplicaciones de manera controlada sobre el entorno asignado, por ejemplo: disco duro, memoria, interfaces, sistema de archivos, red, etc., y no tendrá acceso sobre otras características de entorno que no se le haya asignado [10].

1.4.2.4 Características de Contenedores

Algunas de las características asociadas a los contenedores que se enlistan a continuación:

- **Aislamiento:** Los contenedores presentan un alto grado de aislamiento entre procesos ejecutados en el mismo host, gracias a los *namespaces* y los *cgroups*. Estas características permiten segmentar y gestionar los recursos de hardware y software de manera independiente, garantizando un alto nivel de aislamiento entre contenedores y del sistema base [2].

- **Autónomos:** Los *namespace* dan una vista única por proceso o grupo de procesos asociados a cada contenedor, lo que les permite ejecutarse y gestionar sus propios recursos de manera autónoma e independiente del sistema operativo base [4].

- **Ligeros:** Los contenedores aprovechan una porción de los recursos que usa un sistema convencional. Por lo tanto, los recursos empleados son menores y su despliegue es más sencillo y rápido [4].

- **Portabilidad:** La portabilidad es una característica clave de los contenedores, ya que permite empaquetar las aplicaciones o servicios con todas sus dependencias en una sola imagen. Esto garantiza que el mismo código se pueda ejecutar en diferentes entornos, incluso en sistemas operativos o hardware distintos al entorno de origen [2]. De esta forma, se facilita el despliegue de las aplicaciones.

- **Escalabilidad:** Los contenedores permiten un escalamiento horizontal al desplegar múltiples contenedores livianos para tareas en un corto plazo, aprovechando los recursos del sistema [4].

- **Disponibilidad:** Las aplicaciones en contenedores presentan un alto grado de disponibilidad al desplegarse en múltiples contenedores de una misma imagen, ya sea en el mismo host o en distintos. Esto es útil en el contexto de los microservicios, donde los servicios desplegados pueden distribuirse en diferentes lugares o secciones para cumplir con una única función u ofrecer un servicio final compartido [1].

1.4.2.5 Comparación entre Arquitecturas de Virtualización

Como se describió anteriormente, tanto los contenedores como las VMs presentan su propia arquitectura de virtualización, lo que hace que cada tecnología presente características distintas en su funcionamiento, despliegue, y rendimiento general. A continuación, en la Tabla 1.1 se muestran las principales diferencias entre las tecnologías basadas en hipervisor y las basadas en contenedores.

Tabla 1.1: Máquinas virtuales (hipervisor) vs contenedores, basado en [5], [13], [14].

Parámetros	Máquinas virtuales	Contenedores
Funcionamiento	Se implementan independientemente en una capa sobre el sistema operativo subyacente del host, llamada hipervisor. Requieren sus propios binarios, librerías y recursos físicos.	Corren sobre el mismo kernel del sistema operativo. Sin embargo, requieren las mismas dependencias, binarios y librerías que las VMs.
Despliegue	Asignación de recursos lenta y compleja por parte del hipervisor.	Asignación de recursos rápida y sencilla, mediante la instancia de imágenes.
Aislamiento	Mayor aislamiento entre máquinas virtuales y el sistema base, ya que la capa del hipervisor aísla y gestiona los recursos con independencia del sistema base.	Menor aislamiento, ya que los contenedores comparten los mismos recursos del kernel, lo que significa que, si uno falla, el otro también puede verse afectado.
Escalabilidad	Tienen una escalabilidad muy lenta, debido al tamaño y complejidad de cada VM, limitando el número de instancias creadas en un entorno.	Tienen una escalabilidad muy rápida y una mayor densidad, debido al despliegue de instancias de imágenes ligeras y portables.
Migración	Tienen una mayor complejidad en el proceso de creación y transferencia de imágenes a través de la red, debido a que las imágenes son de mayor tamaño.	Tienen un alto grado de migración en la red, ya que las imágenes de contenedores son muy ligeras y portables, permitiendo una mayor facilidad de transferencia en la red.
Seguridad	La seguridad es alta ya que incorporan la capa del hipervisor, encargada del aislamiento y seguridad.	No presentan medidas de seguridad adicionales, debido a que se despliegan sobre el kernel del host.
Rendimiento general	Tienen una alta sobrecarga de rendimiento en los recursos de hardware, comparables a una maquina física.	Tienen un mayor rendimiento general, al tener un menor gasto de recursos por cada uno de los contenedores.
Casos de uso	Se emplean en servicios que requieren funcionalidades completas de sistemas operativos y sobre entornos con alto consumo de recursos de hardware.	Se emplean en escenarios que requieran únicamente funcionalidades de aplicaciones o servicios y sobre entornos con recursos limitados de hardware.

1.4.3 Servicios de Red

Los servicios de red son un conjunto de software que permiten dotar a un cliente de una funcionalidad y/o recurso de una red de computadoras [15]. Las funciones de red pueden ser cualquiera que se proporcionen mediante protocolos[4] en las diferentes capas de un modelo de Internet, como las del modelo TCP/IP[5] o sus equivalentes en el Modelo OSI[6], como se muestra en la Figura 1.4, donde se especifican algunos de los protocolos usados para servicios de red.

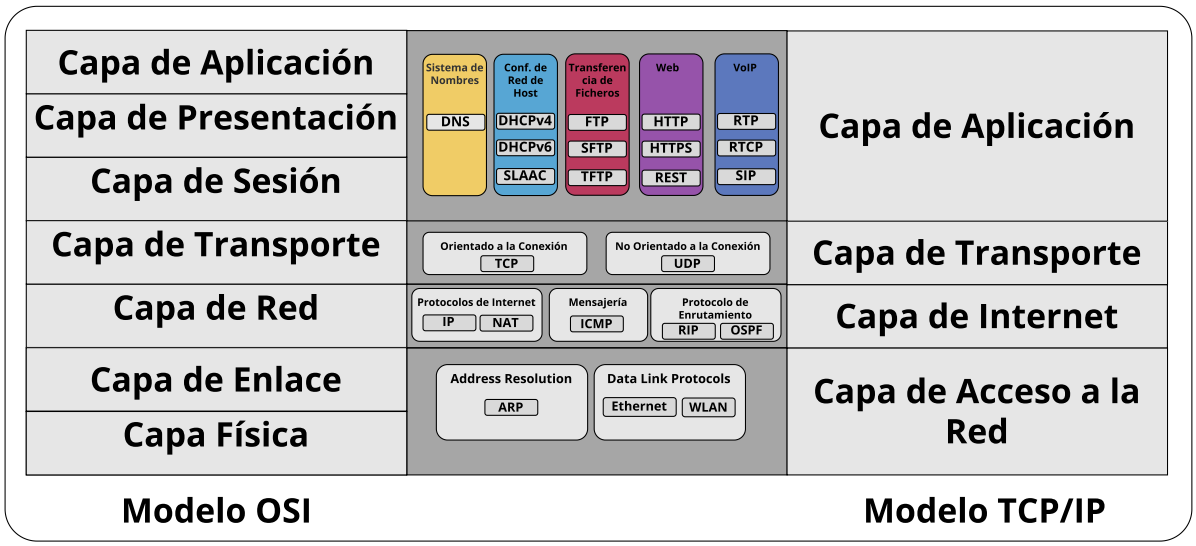


Figura 1.4: Modelos TCP/IP y OSI: Protocolos de Internet, basado en [15].

En este contexto, el presente trabajo se centra en el análisis experimental de algunos servicios de red basados en las capas 2 y 4 del modelo TCP/IP, donde se emplea algunos de los protocolos de red para ofrecer un servicio de red contenerizado. Estos servicios son: DHCP, DNS, FTP, HTTP, VoIP y Routing. A continuación, se realiza una descripción breve de dichos servicios.

[4] **Protocolos:** Son reglas de comunicación utilizadas por hardware o software dentro de un modelo de computación. Cada protocolo tiene una función de red específica. Estas reglas permiten que los dispositivos o programas diferentes puedan comunicarse y trabajar juntos de manera efectiva en la red [15].

[5] **Internet Protocol Suite o TCP/IP:** Es un conjunto de protocolos de computadora que define un estándar para funciones de red en una arquitectura de Internet. El stack de protocolos está dispuesto en 4 capas, capa de acceso a la red, red, transporte, y aplicación [15].

[6] **Open Systems Interconnection (OSI):** Es un modelo de computadora similar a TCP/IP que establece un estándar para los protocolos en una arquitectura de red en Internet. El stack de protocolos se organiza en 7 capas: física, enlace, red, transporte, sesión, presentación y aplicación [15].

1.4.3.1 Protocolo de Configuración Dinámica de Host (DHCP: Dynamic Host Configuration Protocol)

DHCP es un protocolo de la capa de aplicación del modelo TCP/IP (capa 7 del modelo OSI), que proporciona un servicio de direccionamiento de red automático. El servicio de DHCP funciona sobre una arquitectura cliente-servidor que permite asignar direcciones IPv4[7] e IPv6[8] a los clientes en una red, bajo la modalidad de arrendamientos temporales.

También, proporciona información de red adicional como las direcciones de gateway y direcciones de los servidores de DNS. DHCP para IPv4, utiliza los puertos UDP 68 para el cliente DHCPv4 y UDP 67 para el servidor DHCPv4. Por otro lado, para IPv6, DHCP trabaja en los puertos UDP 547 para el cliente DHCPv6 y UDP 548 para el servidor DHCPv6 [15].

1.4.3.2 Servicio de Nombre de Dominio (DNS: Domain Name Service)

DNS es un protocolo de la capa de aplicación del modelo TCP/IP (capa 7 del modelo OSI), que proporciona un servicio para la traducción de *domain names*[9] a direcciones IPv4 o IPv6 y viceversa. Este servicio funciona mediante una arquitectura cliente-servidor, donde los servidores almacenan configuraciones específicas para las traducciones de direcciones y nombres. Además, dependiendo del rol del servidor, pueden procesar las solicitudes para entregar las traducciones a los clientes o reenviarlas a otro servidor para que realice las traducciones. El servidor DNS para IPv4 e IPv6 funciona sobre el puerto UDP 53 [17].

1.4.3.3 Protocolo de transferencia de archivos (FTP: File transfer Protocol)

FTP es un protocolo de la capa de aplicación del modelo TCP/IP (capa 7 del modelo OSI), que proporciona un servicio para la transferencia de archivos en un sistema de directorios

---

[7] **Internet Protocol version 4 (IPv4):** Es un protocolo de la capa de red del modelo TCP/IP en la versión 4, el cual consiste en un arreglo numérico (16 números en 4 octetos separados por un punto decimal xxxx.xxxx.xxxx.xxxx), que permite identificar a un equipo en una red [15].

[8] **Internet Protocol version 6 (IPv6):** Es el protocolo IP en la versión 6, el cual consiste en un arreglo hexadecimal (32 caracteres en 8 octetos separados por dos puntos xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx), que permite identificar a un número mayor de equipos en una red [15].

[9] **Domain Name:** Es una dirección web de Internet definida por una etiqueta alfanumérica, que se asigna a una dirección numerada (dirección IP) para identificar un equipo en la red [16].

[18]. Funciona mediante una arquitectura cliente-servidor, donde el servidor FTP almacena un sistema de archivos con una estructura especifica y lo expone a los clientes FTP, para que tengan acceso a los recursos alojados.

FTP puede operar en dos modos de transferencia: Modo Activo[10] y Modo Pasivo[11]. Para ambos modos, el servicio utiliza el puerto TCP 21 para la transmisión de datos de control. Sin embargo, hay diferencias en los puertos utilizados para la transferencia de datos de usuario. Esto debido a que en el modo activo, la transferencia de datos de usuario se realiza a través del puerto TCP 20 y en el modo pasivo, la transferencia de datos de usuario se realiza a través de un puerto aleatorio superior a 1024 [18].

1.4.3.4 Protocolo de Transferencia de Hipertexto (HTTP: Hypertext Transfer Protocol)

HTTP es un protocolo de la capa de aplicación del modelo TCP/IP (capa 7 del modelo OSI), que proporciona un servicio para la transferencia de páginas de Internet escritas en lenguajes como HTML, JS o similares. HTTP funciona mediante una arquitectura cliente-servidor, donde el servidor HTTP almacena los datos de una página web y responde a las solicitudes de los clientes para obtener el código. HTTP funciona sobre el puerto TCP 80 para el cliente y servidor, y mediante el puerto seguro TCP 443 para HTTPS [15].

1.4.3.5 Routing

El routing es una función de la capa de Internet del modelo TCP/IP (capa 3 del modelo OSI), que permite la comunicación entre equipos de distintas redes mediante el direccionamiento, enrutamiento y encapsulamiento de los datos [15]. Las principales tareas que se realizan en esta función son:

- **Enrutamiento**: Permite el encaminamiento de paquetes de datos a través de la red mediante tablas de enrutamiento que contienen direcciones IP de las redes. Estas

[10] **Modo Activo:** Es el modo principal de FTP, donde el servidor inicia la conexión con el cliente [19].

[11] **Modo Pasivo:** En este modo, el cliente FTP inicia la conexión con el servidor mediante el puerto de control [19].

tablas ayudan a determinar la mejor ruta para enviar los paquetes hacia su destino [15].

- **Encapsulamiento y fragmentación**: Consiste en fragmentar y envolver los datos recibidos de la capa superior en paquetes IP, dividiendo la información y agregando encabezados con direcciones IP de la red de destino. Luego, los paquetes se envían a la capa de enlace. Al recibir un paquete de la capa inferior, se eliminan los encabezados y se lleva a la capa superior [15].

En la Figura 1.5 se muestra una topología convencional en la que se disponen diferentes servicios de red, tales como DHCP, DNS, FTP y HTTP, a través de dispositivos de red como servidores, switchs y routers, que permiten brindar servicios de red al usuario final.

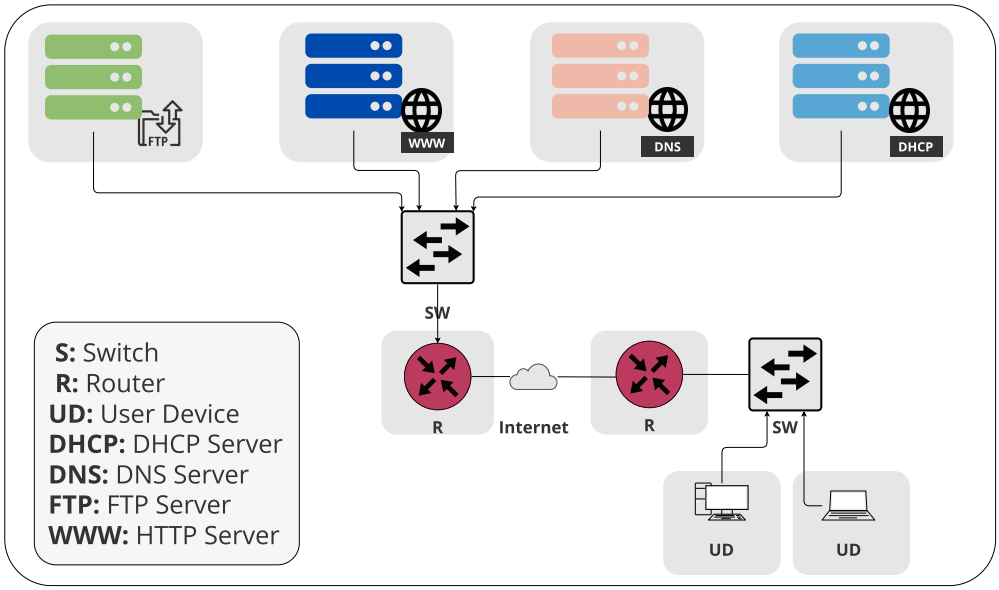


Figura 1.5: Servicios de red: Topología de servicios de red, basado en [15].

1.4.3.6 Voz sobre Protocolo de Internet (VoIP: Voice Over Internet Protocol)

VoIP es una tecnología de red, que permite la transmisión de voz digitalizada en datagramas IP[12]. Esta tecnología funciona sobre una infraestructura basada en la conmutación de paquetes (PS: Package Switching [13]), en lugar de transportar voz analógica sobre una infra-

[12] **Datagrama IP:** Es una unidad de datos empleada por el protocolo IP para enviar información en la red [15].

[13] **Packet Switching:** Es una técnica de transferencia de información en redes de computadores en la que los datos se dividen en paquetes y son enviados de manera independiente a través de la red utilizando el mejor esfuerzo para llegar a su destino [15].

estructura convencional basada en la conmutación de circuitos (CS: Circuit Switching [14]). Un ejemplo de esta última, es la red telefónica pública conmutada (PSTN: Public Switched Telephone Network), que se emplea para las redes telefónicas de hogar. VoIP implementa diferentes protocolos para el transporte y señalización de datos de voz como: protocolo de transporte de tiempo teal (RTP: Real-Time Transport Protocol) que funciona en los puertos UDP 16384 a 32766, protocolo de control de transporte de tiempo real (RTP: Real-Time Transport Protocol), y protocolo de inicio de sesión (SIP: Session Initiation Protocol) que opera en el puerto UDP 5060, conceptos que se ampliarán más adelante [20]. En la Figura 1.6, se muestra la arquitectura de red de VoIP, en la que se muestran cada uno de los elementos que la conforman.

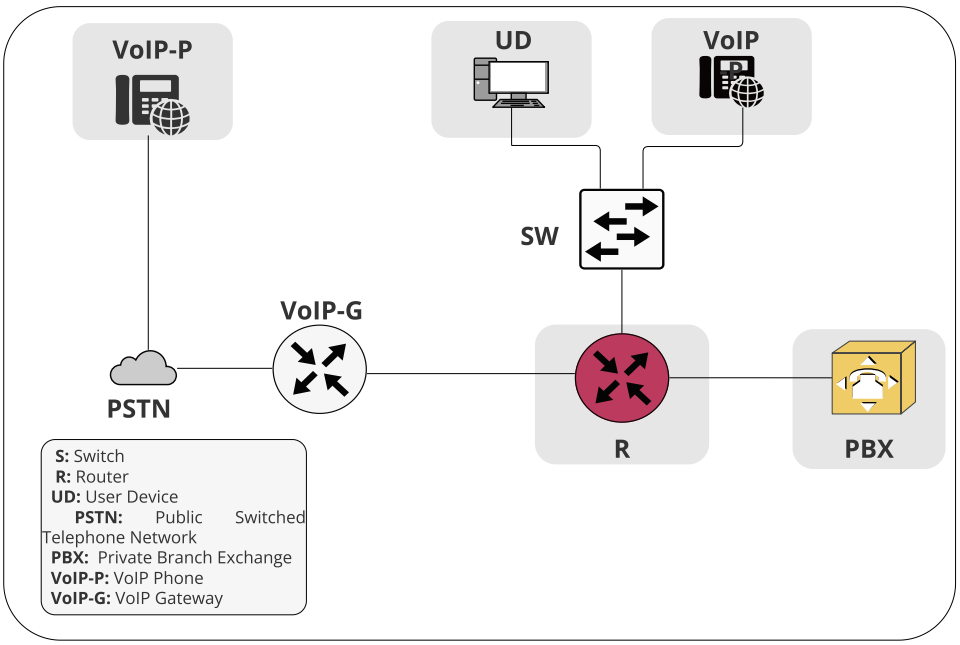


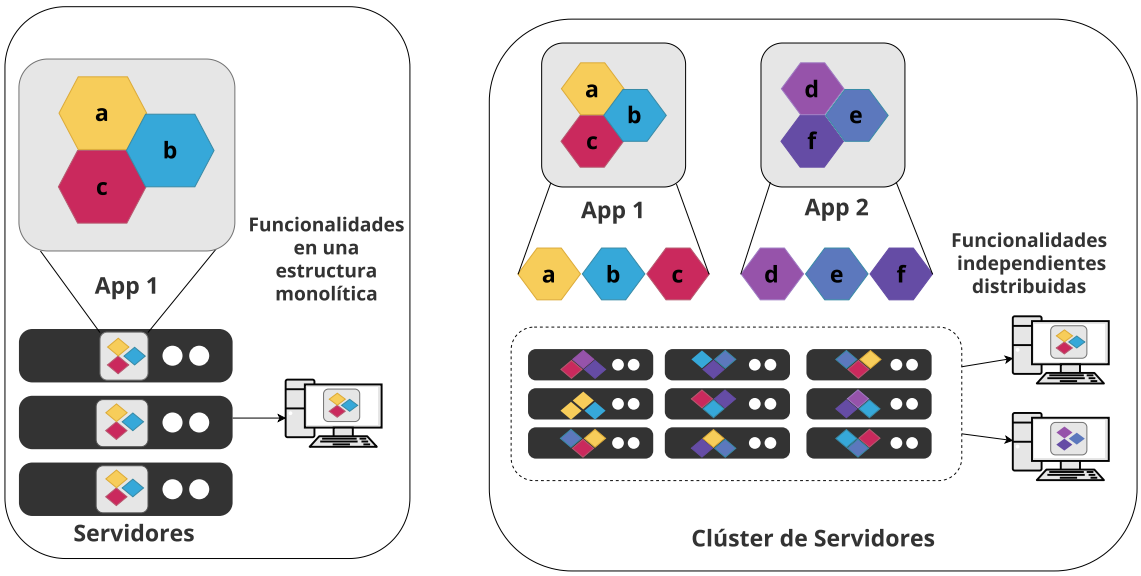
Figura 1.6: Servicio de VoIP: Arquitectura de red VoIP, basado en [20].

1.4.4 Microservicios

La arquitectura basada en microservicios es un modelo de diseño de software que permite la implementación de las funcionalidades de una aplicación o servicio utilizando módulos autónomos e independientes para cada de ellas. Este enfoque es opuesto a la arquitectura tradicional de las aplicaciones basadas en una estructura monolítica, donde las funcionalidades de una aplicación están montadas sobre una única estructura y son codependientes

[14] **Circuit Switching:** Es una técnica de transferencia de información en redes de computadores en la que se establecen canales físicos dedicados para la transmisión de los datos [15].

entre sí. En la Figura 1.7 se ilustra la diferencia entre la arquitectura monolítica (a) y la arquitectura basada en microservicios (b). En el primer modelo, las funcionalidades de una aplicación se encuentran montadas sobre el mismo sistema físico, mientras que, en el segundo modelo, se distribuyen en diferentes entornos físicos o virtuales [1].



(a) Arquitectura Monolítica. **(b)** Arquitectura basada en Microservicios.

Figura 1.7: Comparación de arquitecturas de servicios, basado en [5].

En la actualidad las soluciones basadas en contenedores permiten la implementación y uso dinámicos de microservicios en entornos empaquetados. Esto proporciona mayores beneficios, como mayor escalamiento, independencia y autonomía de los servicios [4]. Es importante destacar que, aunque los conceptos de microservicios y contenedores son complementarios, hablar de uno no implica hablar del otro. Dado que la arquitectura de microservicios puede desplegar servicios sin emplear contenedores y los contenedores pueden desplegar aplicaciones monolíticas sin descentralizar sus funciones [1]. Los requerimientos actuales de diversidad y descentralización de software en aplicaciones y servicios, hacen necesario incorporar modelos basados en microservicios y contenedores. Por lo que estos modelos son de gran ayuda para soportar todas las funciones de una aplicación, sin afectar el rendimiento general de la mismas o de otras funciones [4].

2 METODOLOGÍA

En este capítulo, se describe el proceso y la metodología empleados para la implementación de servicios de red en contenedores Docker.

Al abordar el contexto de Docker, se realiza una breve comparación entre las tecnologías de contenerización disponibles, destacando los beneficios de Docker. Además, se describen las características principales, tecnologías subyacentes, arquitectura y tipos de despliegue de Docker, incluyendo Docker CLI y Docker Compose. Los cuales constituyen los escenarios a desplegar sobre la plataforma Raspberry Pi.

La metodología empleada garantiza una correcta implementación de la solución contenerizada. Este proceso se compone de siete etapas, las cuales se describen en detalle en la Figura 2.1.

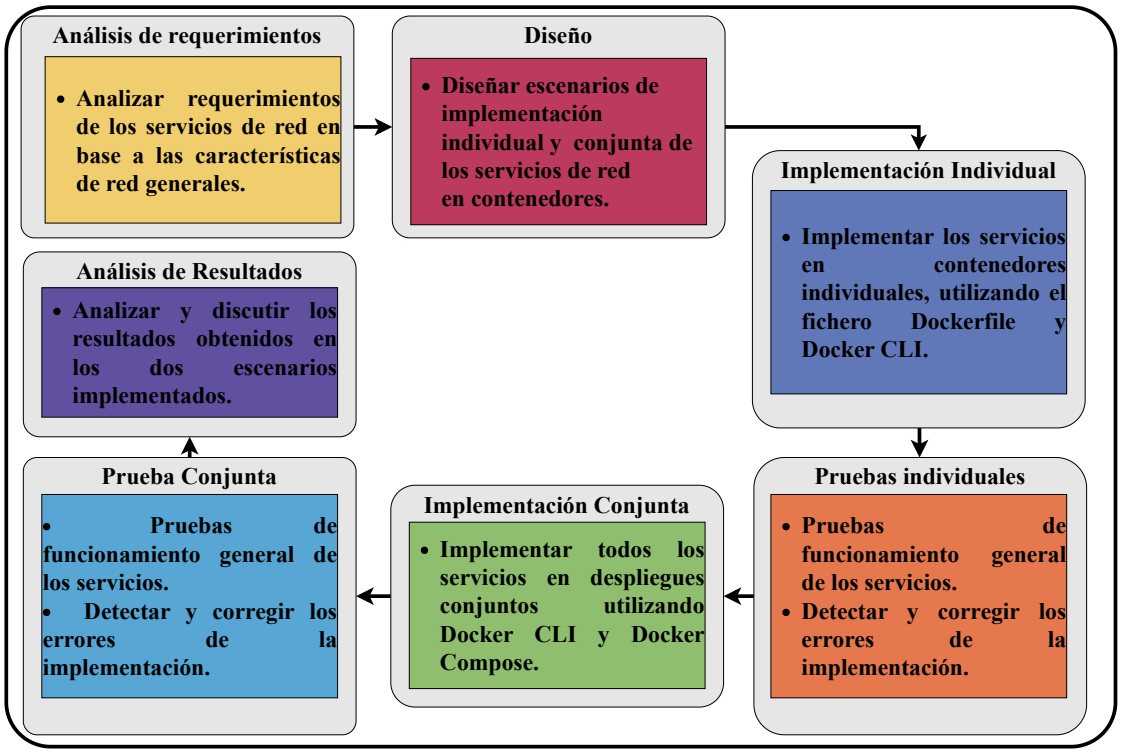


Figura 2.1: Metodología para la implementación de servicios sobre contenedores, basado en [5].

2.1 ANÁLISIS DE REQUERIMIENTOS

Como punto inicial, se realiza un análisis de los requerimientos enfocado en las características de cada uno de los servicios de red a implementar. Es importante tener en cuenta ciertos parámetros generales de los servicios de red a implementar, como: los puertos TCP/UDP utilizados por cada servicio, también se deben considerar otros aspectos relevantes,

como el tipo de conexión y las funcionalidades asociadas a cada servicio. Además, es fundamental tener en cuenta los parámetros de calidad de servicio (QoS: Quality of Service [1]) requeridos como son: disponibilidad, confiabilidad, retardo (delay [2]) y jitter [3]. Por otro lado, para los requisitos del servicio routing, se deben definir parámetros diferentes como la funcionalidad de red asociada, la cual se define en una capa específica del modelo TCP/IP que debe cumplir y/o protocolos asociados a dicha capa. A continuación, en las Tablas 2.1 y 2.2 se resumen los requisitos para cada uno de ellos.

Tabla 2.1: Tabla de requerimientos de servicios de red, basado en [21], [22].

Parámetros	DHCP	DNS	FTP	HTTP	VoIP
Tipo de conexión	Orientado a la conexión.	Orientado y no orientado a la conexión.	Orientado a la conexión.	Orientado a la conexión.	No orientado a la conexión.
Disponibilidad	Alta	Alta	Alta	Alta	Muy alta
Confiabilidad	Alta	Alta	Alta	Alta	Media
Retardo	Alto ($<$400ms)	Alto ($<$400ms)	Alto ($<$400ms)	Medio (150-400ms)	Bajo ($<$150ms)
Jitter admisible	Alto	Alto	Alto	Alto	Bajo
Funciones soportados	Direccionamiento IPv4 y/o IPv6. Asignación de parámetros de red (direcciones de servidores DNS y Gateway).	Almacenamiento y acceso a un registro de *domain names* y zonas DNS. Resolución de *domain names* y direcciones IP.	Almacenamiento y transferencia de archivos. Configuración de credenciales de acceso al sistema.	Capacidad para almacenar y transferir página web.	Configuración de extensiones VoIP. Configuración de parámetros de servicio VoIP, como tipos de vocoders.

2.2 ELEMENTOS DE HARDWARE Y SOFTWARE

Una vez establecido los requerimientos de los servicios, se procede a la descripción de las tecnologías de software y hardware que se emplearán en el diseño a implementar.

[1] **Quality of Service (QoS):** Es la medida de rendimiento de un servicio de extremo a extremo desde el punto de vista del usuario final [21].

[2] **Delay o latency:** Es el tiempo transcurrido entre el envió de un paquete y su recepción en el lado del usuario final. Se mide en milisegundos (ms) [21].

[3] **Jitter:** Es la variación en los retardos entre paquetes [21].

Tabla 2.2: Tabla de requerimientos de servicios de red: Servicios basados en funciones de red, basado en [15].

Parámetros	Router
Capa OSI	3
Capacidad de conexión	3 conexiones para distintas redes.
Protocolos y funciones soportados	❒ Static Routing. ❒ Dynamic Routing: RIP, OSPF y/o BGP. ❒ Tablas de enrutamiento.

2.2.1 Hardware

En el presente trabajo, se emplean principalmente dos placas de desarrollo Raspberry Pi 4 modelo B con 4GB de RAM para ofrecer servicios de red basados en contenedores. A continuación, se resumen las características de la placa de desarrollo utilizada para la implementación.

2.2.1.1 Raspberry Pi

Raspberry Pi es una plataforma de hardware y software libre, distribuida por Raspberry Pi Foundation. Se trata de un ordenador de bajo costo y diseño simple. El sistema operativo predeterminado que se utiliza en Raspberry Pi es Raspberry Pi OS, el cual está basado en Debian. No obstante, es posible instalar y utilizar otros sistemas operativos, incluyendo diferentes distribuciones de GNU/Linux y Windows [23]. Dado que los componentes de una Raspberry Pi varían de acuerdo con la versión, a continuación se describen brevemente los componentes principales de esta plataforma, tal como se ilustra en la Figura 2.2.

- ❒ **General-purpose input/output (GPIO):** Pines GPIO para la interacción de la placa Raspberry Pi con elementos de hardware externos, como sensores, resistencias, LEDs, etc. Los cuales se programan mediante el lenguaje interprete y multiplataforma de Python. Las funcionalidades de los pines incluyen entre otras: 2 pines de alimentación de 5V con un máximo de corriente de 2.5A; 2 pines de 3.3V; 8 pines de tierra (GND) y 26 pines digitales de entrada y salida. Las distribuciones de los pines GPIO para las distintas distribuciones de Raspberry Pi se las puede encontrar en [24].

- ❒ **System On Chip (SoC):** SoC es el elemento principal de una Raspberry Pi, ya que integra en un solo chipset la CPU basado en la arquitectura ARM[4] y la unidad de

[4] **ARM (Advanced RISC Machine):** Es un conjunto de instrucciones de 32 y 64 bits, para el manejo de las

procesamiento gráfico (GPU: Graphics Processing Unit [5]) basado en Broadcom VideoCore graphics core para el procesamiento de video.

- **Ramdon Access Memory (RAM):** Memoria de tipo SDRAM, para un almacenamiento volátil de la información de la Raspberry Pi.
- **Puerto Ethernet**: Puertos RJ45 para la conexión a Internet mediante ethernet que puede alcanzar velocidades Base-T 10/100/1000.
- **Micro SD:** Almacenamiento externo Micro SD.

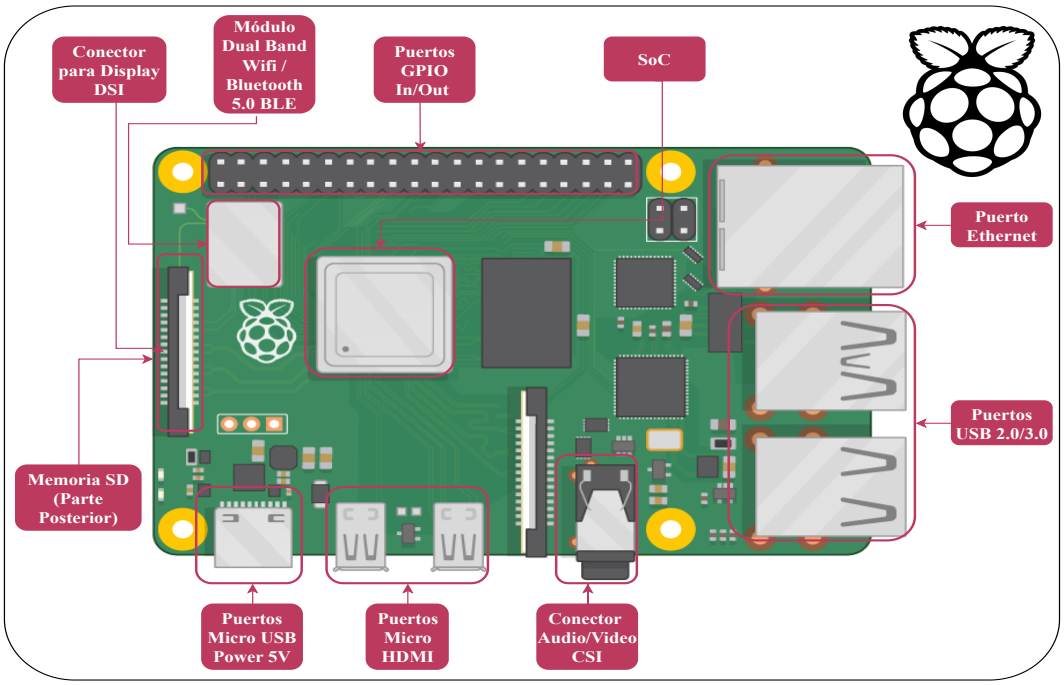


Figura 2.2: Placa de desarrollo Raspberry Pi, basado en [26].

A continuación, en la Tabla 2.3 se realiza un breve resumen de las principales características de las placas Raspberry Pi 4 Model B.

CPUs, basada en la arquitectura RISC (Reduced Instruction Set Computer). La cual presenta características como: aplicaciones de baja potencia, uso de instrucciones menos o más complejas, dependiendo si se basa en arquitectura de 32 bits o de 64 bits [25].

[5] **Graphics Processing Unit (GPU):** Es un circuito integrado diseñado para el procesamiento y aceleramiento gráfico en sistemas computacionales. Esto lo realiza mediante el procesamiento matemático [25].

Tabla 2.3: Cuadro descriptivo de Raspberry Pi 4 Model B, basado en [26].

Raspberry Pi	4 Model B
Fecha de Lanzamiento	06/19.
Dimensiones (mm)	85x56mm.
SoC	Broadcom BCM2711.
CPU	Cortex-A72 (ARM v8) de 64-bit @1.8GHz.
GPU	VideoCore VI.
SDRAM	2GB, 4GB, 8GB LPDDR4-3200.
Pines GPIO	40.
Puertos USB	2 USBv2.0, 2 USBv3.0, 1 USB-C.
Video y Audio	Jack 3.5mm AV, 2 Micro HDMI.
Almacenamiento Externo	Micro SD.
Conectividad Red	Ethernet Base-T 1000, WiFi IEEE 802.11ac Dual-Band (2.4GHz/5GHz), Bluetooth 5.0, BLE.
Alimentación	Micro USB 2.5A 12.5W/5V, GPIO para PoE-HAT.

2.3 TECNOLOGÍAS DE CONTENERIZACIÓN

En la actualidad, existe una amplia variedad de tecnologías de contenerización. Las más empleadas son: LXC, Rkt y Docker [6]. A continuación, se describe cada una de ellas.

- **LXC**: Es una tecnología de virtualización basada en contenedores ligeros que funciona sobre Linux y sus distribuciones. Además, la tecnología LXC emplea una API [6] común y flexible para el despliegue de contenedores [28].
- **CoreOs Rkt (rocket)**: Es una tecnología basada en contenedores que se ejecuta sobre CoreOs, una distribución de Linux diseñada para trabajar con pilas de sistemas de software [29].
- **Docker:** Es una tecnología basada en contenedores que se ejecuta sobre Linux, Windows y MacOS. Aunque la arquitectura Docker se basa en LXC, integra sus propias características, como su eficiencia, bajo peso y portabilidad, lo que han hecho líder en el desarrollo de tecnologías de contenerización en el mercado [11].

En la Tabla 5.1 del ANEXO III se realiza una breve comparación entre algunas de las tecno-

[6] **Application Programming Interface (API):** Es un conjunto de definiciones y protocolos para crear e integrar aplicaciones de software [27].

logías de contenerización mencionadas, tomando en cuenta características como criterios de virtualización, seguridad, arquitectura, complejidad, casos de uso, entre otros. De acuerdo con ello, se puede establecer que Docker ofrece múltiples ventajas, tales como diversidad de compatibilidad con sistemas operativos, compatibilidad de imágenes y facilidad de uso; en comparación con las demás tecnologías de contenerización. Por lo tanto, para la implementación de los servicios de red en el trabajo actual, se enfocará exclusivamente en la tecnología Docker, la cual se describe más a detalle a continuación.

2.3.1 Docker

Es una tecnología open source de contenerización para la creación, ejecución y administración de contenedores ligeros, portátiles y autosuficientes. Estos contenedores pueden ser desplegados tanto localmente como en plataformas de cloud, como Microsoft Azure, Servicios de Web de Amazon (AWS: Amazon Web Services) o Google Cloud [30].

Para el despliegue de contenedores eficientes, ligeros y portables, Docker monta toda su arquitectura sobre un sistema subyacente [31]. Por ello para entender mejor el funcionamiento y despliegue de un contenedor Docker, es importante comprender dichas tecnologías y su arquitectura. A continuación, se describen cada una de ellas.

2.3.1.1 Tecnologías Subyacentes

Los contenedores Docker se ejecuta sobre Docker Engine, nombre especifico de la tecnología de contenerización de Docker, el cual está escrito en lenguaje Golang. Docker, al igual que en muchas tecnológicas de contenedores, utiliza los *namespaces* y *cgroups* para aislar y administrar los recursos de los contenedores. Algunos de los *namespace* usados por Docker para aislar las diferentes funcionalidades que proporciona el host al momento de la contenerización, son: Identification Process Namespace (IDP) [7], Network Namespace (NET) [8] , Mount Namespace (MNT) [9], Interprocess Communication Namespace (IPC)[10]

[7] **Identification Process Namespace (IDP)**: *Namespace* asigna identificadores únicos a los procesos de un contenedor [1].

[8] **Network Namespace (NET)**: *Namespace* que permite aislar los recursos red de un contenedor, como interfaces de red, tablas de enrutamiento, direcciones IP, etc, con respecto a otros contenedores y al sistema en general [1].

[9] **Mount Namespace (MNT)**: *Namespace* que proporciona un sistema de archivos independiente y aislado para un proceso o conjunto de procesos dentro de un contenedor [1].

[10] **Interprocess Communication Namespace (IPC)**: *Namespace* encargado de proporcionar aislamiento de los recursos de comunicación interprocesos de un contenedor [20].

y UNIX Time-Sharing Namespace (UTS) [11]. La Figura 2.3 muestra como los *namespaces* y *cgroups* trabajan en conjunto para desplegar contenedores Docker.

Por otro lado, Docker incorpora un sistema de archivo, denominado sistema de archivos de unificación avanzado (AUFS: Advanced Union File System) [12], para la construcción de imágenes por capas [31]. Esto ayuda a la optimización de recursos de almacenamiento al momento de crear una imagen Docker.

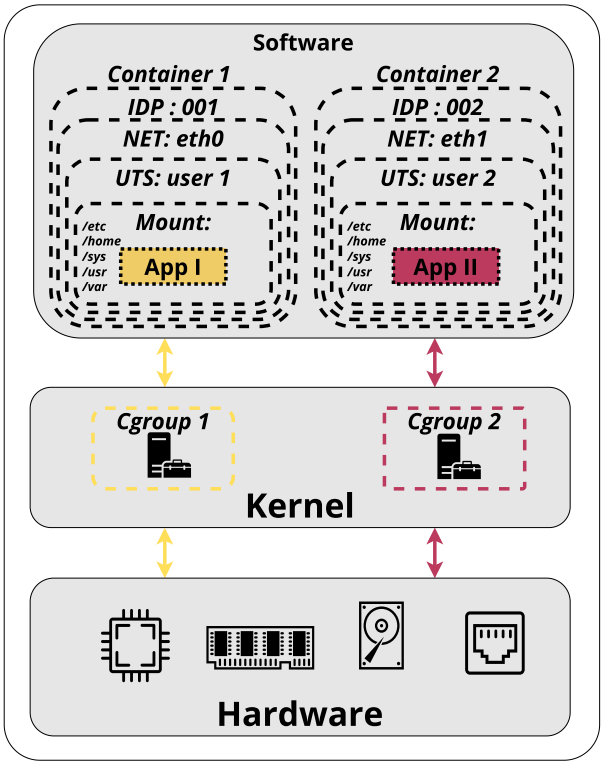


Figura 2.3: Sistema subyacente: *Namespace* y *cgroups* dentro del entorno Docker, basado en [33].

2.3.1.2 Arquitectura Docker

Docker emplea una arquitectura basada en Cliente/Servidor para el despliegue de contenedores en un entorno local (Docker host) o en cloud a través de registros públicos o privados. Tal como se muestra en la Figura 2.4 la arquitectura Docker está constituida por varios elementos que son: Docker CLI (Command Line Interface) o Docker Client, Docker Daemon o

[11] **UNIX Time-Sharing Namespace (UTS)**: *Namespace* que permite el aislamiento de los identificadores del kernel, nombres de host y dominio [20].

[12] **Sistema de archivos de unificación avanzada (AUFS: Advanced Union File System):** AuFS es un sistema de archivos que se utiliza para superponer uno o varios directorios en el host de Linux y formar un solo sistema de archivos único. En este sentido, Docker, emplea AuFS para proporciona capas de imágenes, de solo lectura, apiladas una encima de otra para crear un sistema de archivos raíz de un contenedor. El controlador de almacenamiento de Docker apila estas capas y permite obtener una vista unificada de ellas [32].

Docker Server, Docker Registry y los Docker Objects.

- **Docker Daemon (dockerd):** Es el servidor de Docker, encargado de construir, ejecutar y distribuir contenedores Docker. Docker Daemon escucha las solicitudes del Docker Client mediante una comunicación bidireccional a la API de Docker, llamada API RESFul [13], para administrar los Docker Objects como images, containers, networks y volumes de Docker [11].

- **Docker Client (docker):** Es la interfaz entre el usuario y Docker Daemon. Funciona mediante la aceptación de líneas de comandos del usuario en la terminal, para la ejecución de una acción que Docker Daemon tiene que realizar. Otro cliente Docker es Docker Compose, del que se hablara más adelante [11].

- **Repositorio (repo):** Contiene un conjunto de imágenes de Docker relacionadas entre sí, marcadas con una Tag [14] [1].

- **Registry:** Es un servicio en cloud que contiene y provee acceso a los repositorios de contendores de múltiples grupos de trabajos [5]. Docker cuenta con su registro público, llamado Docker Hub, donde se almacenan las imágenes de contenedores, que pueden ser extraídas (pull) o publicadas (push). En el caso de los registros privados podemos encontrar a Azure Container Registry de Microsoft, Google Container Registry de Google o AWS Container Registry de Amazon [1].

- **DockerFile:** DockerFile es un fichero de texto sin extensión para la construcción de imágenes Docker. En este fichero se declara las configuraciones (imagen base, instrucciones de instalación, etc.) mediante líneas de instrucciones para describir el entorno de trabajo necesario, para el despliegue de un contenedor Docker [1]. Cada comando en DockerFile modifica el sistema de archivos y a su vez crea una capa mediante el sistema AUFS, aumentando el tamaño de la imagen.

[13] **API RESTFUL:** Es una interfaz entre dos sistemas de computadoras emplean para la transferencia de información de manera segura, confiable y eficiente a través de internet [34].

[14] **Tag:** Es un identificador único para las imágenes de Docker, donde se establece un identificador y la versión de la imagen correspondiente. Por defecto, la etiquete colocada en la construcción de la imagen es latest, que indica la última versión de la imagen [1].

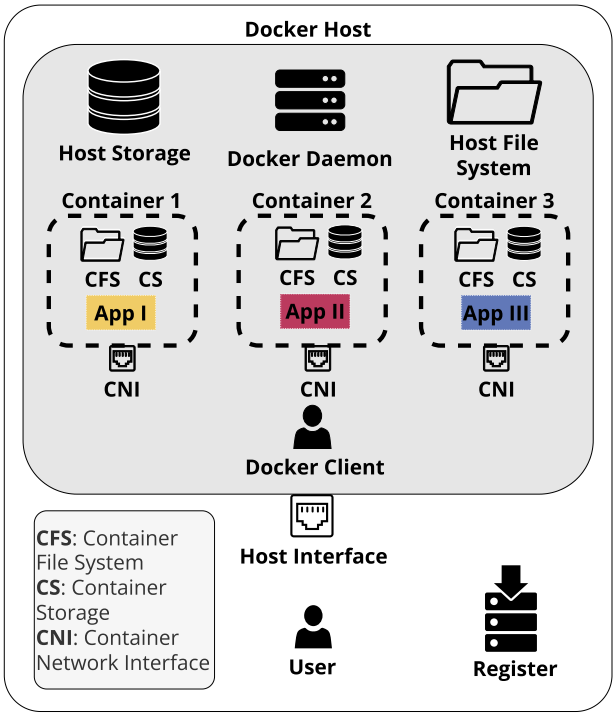


Figura 2.4: Arquitectura Docker: Elementos de la arquitectura de Docker, basado en [11].

2.3.1.3 Docker Objects

Los objetos Docker son los elementos que representan una funcionalidad dentro del entorno de Docker. Estas funcionalidades pueden ser de almacenamiento, red o a imágenes de contenedores y contenedores en sí mismos. Cada uno de estos elementos pueden tener diferentes tipos, según su utilidad y características.

- **Imagen:** En el entorno Docker, las imágenes se definen como una "plantilla" que contiene el código, dependencias y un "snapshot" del sistema de archivos base de una aplicación [31]. Como se mencionó anteriormente, las imágenes se construyen utilizando un sistema de capas basado en AUFS. Una vez que se completa el proceso de construcción de una imagen, esta se vuelve inmutable y no puede ser alterada [5].

- **Container:** Es la instancia de una imagen Docker, que se ejecutan, detienen o eliminan mediante el cliente Docker. A diferencia de las imágenes, los contenedores son modificables ya que se despliegan en la capa de escritura del sistema de archivos. Sin embargo, esta capa no es persistente porque solo existe en el ciclo de vida del contenedor. Por lo tanto, cuando finaliza el ciclo de un contenedor, la capa desaparece y se eliminan las modificaciones realizadas en ella [5]. Es en este contexto donde Docker nos ofrece mecanismos de almacenamiento permanentes llamados Volumes.

❒ **Volumes:** Son mecanismo de almacenamiento creados mediante el Docker Client a partir de directorios o archivos almacenados en el Docker host. Una vez credos, los directorios se montan en el contenedor para tener acceso a un sistema de archivos fuera del entorno aislado. En la Figura 2.5 se ilustra la estructura de almacenamiento de los principales objetos Docker dentro de un host.

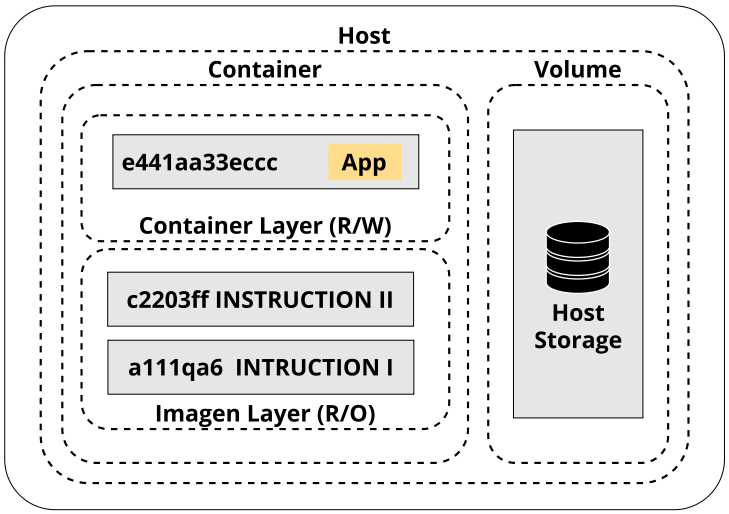


Figura 2.5: Objetos Docker: Imágenes, contenedores y volúmenes Docker, basado en [35].

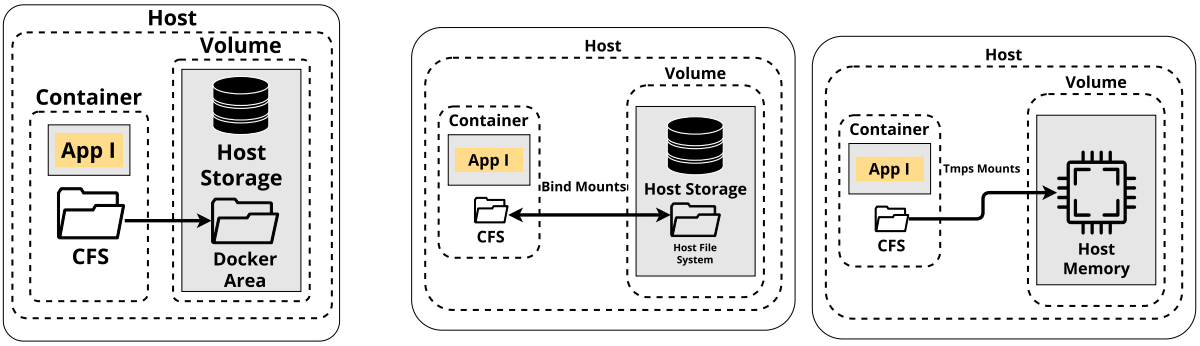
Existen dos tipos principales de volúmenes en Docker: Anonymous Volumes [15] y Named Volumes [16], los cuales se diferencian en la manera en cómo se almacenan y se montan en un contenedor. Además, existen otros mecanismos de almacenamiento, como los Bind Mounts [17], que son similares a los volumes pero con un tipo de almacenamiento diferente y los Tmpfs Mounts [18]. En la Figura 2.6 se muestra los diferentes mecanismos de almacenamiento de contenedores que se puede implementar mediante Docker.

[15] **Anonymous Volumes:** Son volúmenes que se implementan de manera anónima, es decir, que no se especifica el nombre del volumen asociado. Estos volumes se almacenan en un directorio específico del host al momento de crearlos ("/var/lib/docker/volumes" en el caso de Unix). Además, los Anonymous volumes solo persisten dentro del ciclo de vida del contenedor, es decir de manera temporal [35].

[16] **Named Volumes:** Son volúmenes en los que se establece explícitamente una etiqueta al volumen asociado y, al igual que los Anonymous volumes, son almacenan en la misma ruta específica del directorio del host pero con un directorio asociado al nombre del volumen. También, los datos almacenados en este tipo de volumes son persistentes fuera del ciclo de vida del contenedor [35].

[17] **Bind Mounts:** Son mecanismos de almacenamientos similares a los Volumes, pero que a diferencia de estos, no son administrados por Docker. Permiten el montaje de datos en una carpeta específica del host sobre un contenedor y los datos almacenados son persistentes fuera del ciclo de vida del contenedor [36].

[18] **Tmpfs Mounts:** Son mecanismos de almacenamientos temporales que se almacenan en la memoria del host, sin que persistan en el sistema de archivos del contenedor o del host. Esta es una forma de almacenamiento fuera de la capa de escritura del contenedor, pero que existe dentro del ciclo de vida del contenedor [37].



(a) Volumes. **(b)** Bind Mounts. **(c)** Tmpfs Mounts.

Figura 2.6: Mecanismos de almacenamiento en Docker: Volumes, Bind Mounts, Tmpfs Mounts, basado en [35]-[37].

En la Tabla 5.2 del ANEXO IV se resumen las principales características de los distintos mecanismos de almacenamiento en Docker, como tipo de acceso, persistencia, entre otras.

- **Networks:** Son entidades que permiten crear redes Docker y proporcionar funcionalidades de red básica a través de los network drivers[19], que se denominan con el mismo nombre de la red que manejan. Docker ofrece tres redes predeterminadas: None[20], Bridge[21] y host[22].

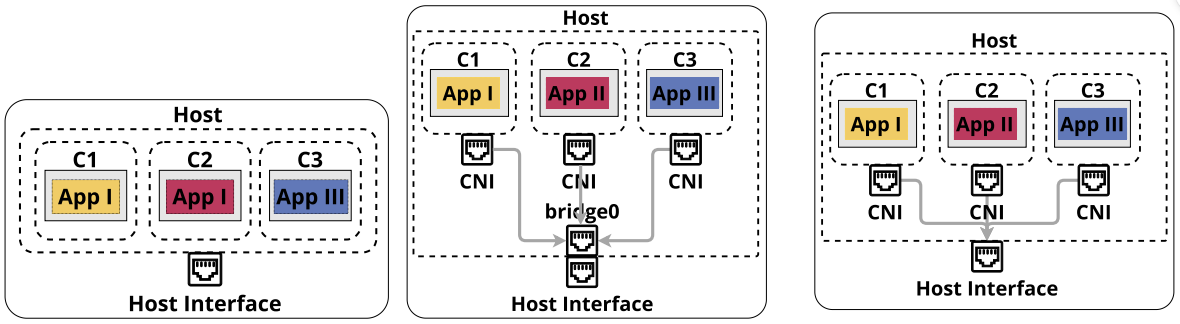
En la Figura 2.7 se muestra los diferentes tipos de redes que se pueden crear en un contenedor Docker.

[19] **Network dirvers:** Son interfaces que se pueden conectar a un contenedor para proporcionar una funcionalidad de red determinada [38].

[20] **None:** No es una red de Docker propiamente dicha, ya que no tiene ninguna interfaz de red fuera del contenedor, sino que solo presentan una conexión entre el contenedor y la interfaz de Loopback, por lo que se emplea para realizar pruebas offline [30].

[21] **Bridge:** Es la red por defecto de Docker que utiliza la funcionalidad bridge de Linux para permitir la comunicación entre contenedores, pero no con el exterior. Para lograr esto, Docker crea conexiones virtuales entre los contenedores y la interfaz virtual de red, llamada “docker0”. Después de lo cual, se crea una red interna y se asignan automáticamente direcciones IP a cada uno de los contenedores. Sin embargo, mediante el uso de iptables y Network Address Translation (NAT) de Docker, se puede establecer un mapeo de puertos desde la red de contenedores hacia el exterior [39].

[22] **Host:** Es una red que permite que el contenedor y el host compartan un mismo *namespace* de red. Es decir, que el contenedor comparte todas las interfaces de red del host sin ningún nivel de abstracción entre ellos. Debido a esto, este tipo de red ofrece un mayor rendimiento que otras redes, ya que la red del contenedor es la red del host [30].



(a) None Network. **(b)** Bridge Network. **(c)** Host Network.

Figura 2.7: Docker networks: None, Bridge y Host Networks, basado en [38].

Además de las redes mencionadas anteriormente, existen otros tipos de redes que proporcionan funcionalidades de redes adicionales, como overlay[23], ipvlan[24] y macvlan[25].

En la Tabla 5.3 del ANEXO V, se muestra una comparación entre las principales redes que se pueden implementar en Docker, enfocándose en sus características más importantes.

2.3.1.4 Otras Definiciones del Entorno Docker

- **Variables de entorno:** Son variables definidas en todo el contexto de un contenedor Docker. Las cuales son accesibles dentro de la configuración y construcción de una imagen; o durante la ejecución de un contenedor. Esto permite una mayor flexibilidad en el despliegue de contenedores.
- **Argumentos:** Son variables definidas dentro del contexto de una imagen de contendor. Esto permite dar una mayor flexibilidad al momento de construir una imagen.
- **Orquestador:** Es un sistema que permite la administración y gestión de tecnologías basadas en contenedores. Este sistema permite la administrar y monitorear múltiples contendores como una sola entidad, con la finalidad de proporcionar un servicio de alta disponibilidad, escalabilidad y gestión [43].

[23] **Overlay:** Es un tipo de red que permite la comunicación de contenedores distribuidos en diferentes hosts, conectados a diferentes Docker Daemons. Para lograr esto, se emplean túneles de red que permiten que los contenedores parezcan estar en el mismo host [30]. Las redes overlay se encuentran disponibles únicamente para Linux y en modo Docker Swarm, el cual se puede revisar en [40].

[24] **Ipvlan:** Es un controlador de red que se basa en el principio de VLANs para permitir un control sobre las direcciones IPv4 e IPv6 para la conectividad a la red física, sin necesidad de emplear un bridge entre la interfaz del contenedor y la NIC del host. Esto se logra mediante la asociación de una interfaz del contenedor a una interfaz o subinterfaz del host [41].

[25] **Macvlan:** Es un controlador de red similar a una Ipvlan, donde emplea la asociación de una interfaz del contenedor a una interfaz o subinterfaz del host; pero sin direccionamiento a nivel de red, dado que asignac direcciones MAC a las interfaces virtuales de los contenedores. Esto permite que un contenedor funcione virtualmente como un dispositivo físico en la red [42].

❒ **Cluster:** Es un conjunto de contenedores administrados por un orquestador y es el elemento principal de gestión de un orquestador [43].

2.3.1.5 Despliegue de Contenedores con Docker

Docker es una tecnología diseñada para desplegar contenedores en entornos locales de desarrollo y en entornos de producción fuera del local. El ciclo de vida de un contenedor puede extenderse más allá de un entorno de prueba local, como se muestra en la Figura 2.8. Donde, el ciclo comienza con la creación de una imagen, ya sea descargándola directamente desde el registro público de Docker Hub o utilizándola como base en un archivo Dockerfile (1). Luego, se instancia esta imagen para crear un contenedor que ejecuta la aplicación contenerizada hasta su detención o finalización (2). Finalmente, la imagen se carga en el registro Docker Hub (3) para que pueda ser descargada en un host remoto (4) y posteriormente ser instanciada (5).

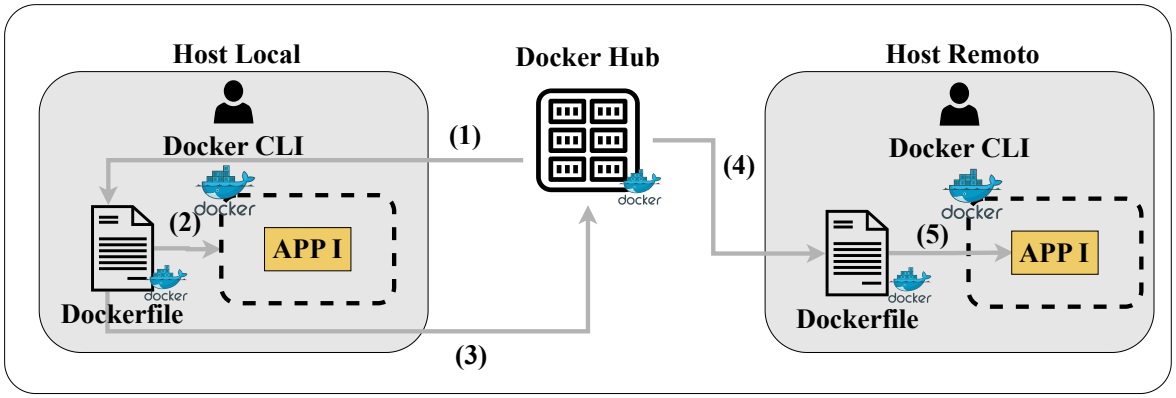


Figura 2.8: Ciclo de vida de un contenedor Docker, basado en [5], [31].

2.3.1.6 Tipos de Despliegues

Docker ofrece diversas opciones nativas para el despliegue de contenedores. Entre ellas, se encuentran Docker Desktop, que es una interfaz gráfica para crear contenedores mediante una aplicación de escritorio; Docker CLI, que permite crear contenedores mediante líneas de comando; Docker Compose, para el despliegue múltiple contenedor y Docker Swarm, que es el orquestador de contenedores Docker. Para el presente trabajo se emplea las opciones basadas en Docker CLI y Docker Compose, para el despliegue de contenedores. No se emplean los modelos de despliegue mediante Docker Desktop, ni ningún sistema de orquestación de contenedores. Esto se debe a que estas herramientas, especialmente las de orquestación, son sistemas que introducen sus propios conceptos de despliegue.

❒ **Docker Desktop:** Es una herramienta que permite construir, ejecutar y probar conte-

nedores Docker mediante una interfaz gráfica interactiva. Esta herramienta está disponible para sistemas operativos como Linux, Windows y MacOs [5].

- **Docker CLI:** Es la forma tradicional de desplegar contenedores mediante líneas de comando. Funciona como una interfaz entre el usuario y Docker Deamon para ejecutar una acción en el entorno Docker. Aunque es propia de Linux, se puede implementar en Windows y MacOs mediante interpretadores de comandos de Linux. La estructura general de los comandos mediante Docker CLI se muestra en el Comando 2.3.1.6:

```
docker [command] [options] values
```

Donde:

- **docker:** Es el cliente docker.
- **command:** Es el comando que docker daemon ejecutará.
- **options:** Opciones de comandos docker.
- **values:** Valores ejecutados por el comando.

A continuación, en la Tabla 2.4, se presenta los comandos básicos que se van a emplear para implementar los servicios de red mediante Docker CLI.

Tabla 2.4: Comandos generales de Docker, basado en [44].

Comando	Descripción
docker version	Muestra la versión del servidor Docker.
docker login	Inicia sesión en la cuenta de Docker Hub.
docker logout	Cerra sesión en la cuenta de Docker Hub.
docker ps	Lista los contenedores en ejecución.
docker restart	Reinicia un contendedor.
docker run	Crea y Ejecuta un contenedor.
docker exec	Ejecuta comandos adicionales en un contenedor en ejecución.
docker stop	Detiene uno o más contenedores.
docker rm	Elimina uno o más contenedores en desuso.
docker logs	Muestra los registros de los contenedores.

Además, se pueden implementar instrucciones para objetos docker específicos, como containers, images, volumes y networks, tal como se muestra en la Tabla 2.5. La estructura de los comandos para los objetos docker se muestra en el Comando 2.3.1.6:

```
docker [docker object] [comnand] [options]
```

Donde:

- ❒ **docker object:** Objeto docker que puede ser: container, image, volume y network.
- ❒ **command:** Es el comando que docker daemon ejecutara.
- ❒ **options:** Opciones de comandos docker.

Tabla 2.5: Comandos generales de objetos Docker, basado en [44].

Comando	Descripción
docker [docker object] create	Crea un objeto docker.
docker [docker object] inspect	Analiza información de uno o varios objetos docker.
docker [docker object] ls	Lista los objetos docker.
docker [docker object] prune	Elimina todos los objetos docker.
docker [docker object] rm	Elimina todos los objetos docker en desuso.

Algunas de las opciones disponibles para comandos como run, create, start y para los objetos docker asociados se muestran en la Tabla 2.6.

Tabla 2.6: Opciones para comandos Docker, basado en [44].

Opción	Descripción
-e	Define una o múltiples variable de entorno.
–name	Asigna un nombre a un contenedor.
-t	Asigna un pseudo TTY al contenedor.
-i	Ejecuta a un contenedor en modo interactivo.
-d	Ejecuta un contenedor en segundo plano (Detach mode).
–restart	Reinicia un contenedor automáticamente.
–rm	Remueve automáticamente un contenedor después de haberse detenido.
–privileged	Asigna permisos de acceso privilegiado a un contenedor.
–net –network	Conecta un contenedor a una red Docker.
-v	Asigna un objeto volumen o bind mount a un contenedor.

❒ **Docker Dockerfile:** En este tipo de implementación se construye una imagen de contenedores a partir de un fichero Dockerfile. Como se definió anteriormente, en este tipo de ficheros se declara instrucciones específicas, que Docker debe ejecutar en el momento de la construcción. El formato para una instrucción en el archivo Dockerfile se puede observar en el Comando 2.3.1.6 y en la Tabla 2.7 las instrucciones básicas en un fichero Dockerfile.

```
INSTRUCTION Argument
```

Donde:

❒ **INSTRUCTION**: Es la orden que se ejecuta en la construcción de la imagen. Cada instrucción se ejecuta de manera secuencial y cada una crea una capa AUFS.

❒ **Argument**: Es el argumento de entrada (identificador, valor, asociación keyword-value comando, etc.) que la instrucción va a ejecutar.

Tabla 2.7: Instrucciones de archivo Dockerfile, basado en [45].

Instrucción	Descripción
ARG	Especifica una variable dentro del contexto del archivo Dockerfile y en tiempo de compilación (build).
FROM	Especifica la imagen base sobre la que se construye una imagen.
RUN	Especifica el comando que se ejecutará sobre la capa de una imagen.
LABEL	Agrega metadatos a una imagen en forma de etiquetas.
EXPOSE	Declara los puertos de red del contenedor que escucha en su ejecución.
WORKDIR	Especifica directorio raíz donde se establecerá la capa de la imagen.
VOLUME	Especifica un volumen para crear un punto de montaje asociada a un contenedor.
ENV	Especifica variables de entorno dentro del contexto del archivo Dockerfile, como en la imagen y en la ejecución del contenedor.
ADD	Permite copiar nuevos archivos o directorios del host y los agrega en la ruta del sistema de archivos de la imagen.
CMD	Define un comando de ejecución en el inicio del despliegue de un contenedor. El comando puede ser remplazado al especificar uno nuevo al ejecutar el contenedor. La sintaxis puede ser de dos formas: **Exec form,** que emplea una sintaxis JSON para definir un comando y sus argumentos; y **Shell form,** que emplea una sintaxis de línea de comandos similar a la de una consola de un sistema operativo.
ENTRYPOINT	Define un comando, que al igual que CMD, ejecuta al inicio de un despliegue de un contenedor. Puede ejecutarse juntamente con el parámetro CMD, al definir un comando ejecutable en un ENTRYPOINT y posterior a ello, definiendo argumentos por defecto en un CMD, que pueden ser modificados.

Para construir una imagen a partir un archivo Dockerfile, se emplea el comando build. La estructura se específica en el Comando 2.3.1.6.

```
docker build [options] {Dockerfile_Path | repository[:tag]}
```

Donde:

- ❒ **options:** Opciones del comando build.
- ❒ **Dockerfile_Path:** Ruta absoluta o relativa del archivo Dockerfile.
- ❒ **repository[:tag]:** Repositorio donde se encuentra el archivo Dockerfile.

Algunas de las opciones que se puede emplear al momento de la construcción de una Imagen Docker se muestran en la Tabla 2.8.

Tabla 2.8: Comandos Build para construcción de imágenes, basado en [45].

Opción	Descripción
docker build –file, -f	Establece el nombre del fichero Dockerfile, para construir la imagen.
docker build –tag, -t	Establece una etiqueta de una imagen.

- ❒ **Docker Compose:** Docker Compose es una herramienta de Docker para la definición y ejecución de servicios múlti-contenedor. Esta herramienta se basa en archivos con formato YAML [26] para definir las configuraciones de los servicios a desplegar, como: opciones de ejecución, detención y reconstrucción de servicios; imágenes, volúmenes y redes asociadas [47]. En la Figura 2.9 se muestra el despliegue de contenedores basados en Docker Compose, al configurar un archivo YAML. Donde, las imágenes de contenedores son creadas o descargadas del registro Docker Hub (1) y son incorporadas en el archivo YAML, para luego ser ejecutadas todas las instancias simultáneamente (2).

Cabe mencionar que los componentes de una aplicación se definen como Servicios, concepto abstraído de la ejecución de una aplicación contenerizada de una misma imagen y ejecutada más de una vez para ofrecer un servicio. La comunicación y almacenamiento se los realizan mediante los Docker Objects, como son las Networks y Volumes respectivamente. Además, incorpora conceptos dedicados de este modelo de implementación [48].

[26] **Yet Another Markup Language (YAML:** Es un lenguaje de serialización de alto nivel, es decir que es fácilmente declarado y leído por seres humanos. Este lenguaje es usualmente empleado para escribir archivos de configuración [46].

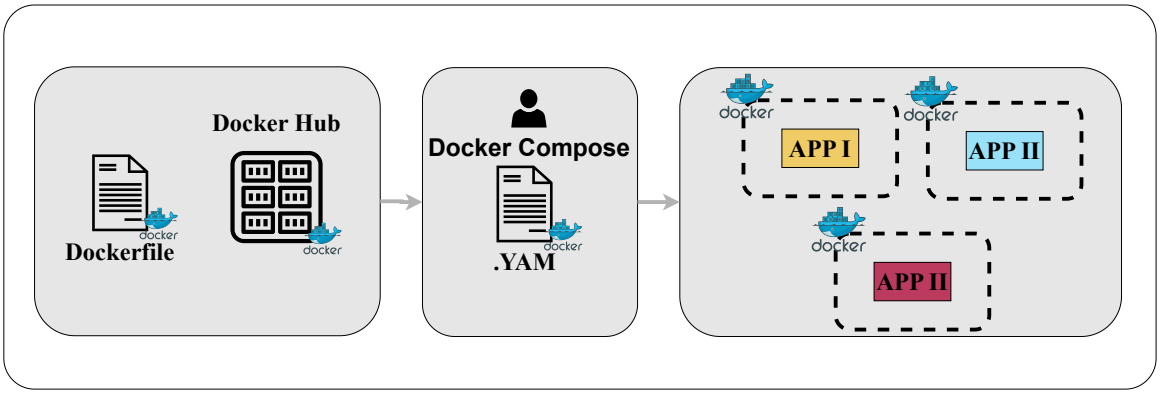


Figura 2.9: Proceso de Despliegue de un Contenedor basado en Docker Compose, basado en [5].

En la Tabla 2.9 se presentan la descripción de las instrucciones que definen los principales elementos de Docker Compose y en la Tabla 2.10, se presentan las instrucciones para definir el elemento de servicio en un fichero YAML.

Tabla 2.9: Instrucciones para los elementos de Docker Compose en el archivo YAML, basado en [48].

Instrucción	Descripción
version	Campo de versión del esquema multi-contenedor.
services	Define los servicios que los contenedores desplegarán.
networks	Define networks como elementos de red para la comunicación de servicios.
volumes	Define los volumes como elementos de almacenamiento persistente.

Tabla 2.10: Instrucciones para definir un Service en el archivo YAML para Docker Compose, basado en [48].

Instrucción	Descripción
imagen	Imagen base para creación de contenedores.
scale	Define un número de contenedores para desplegar un servicio.
labels	Agrega metadatos a un contenedor en forma de etiquetas.
container_name	Prestablece un nombre al contenedor. No aplica para servicios con escalamiento.
enviroment	Define variables de entorno para todos los servicios.
proviliged	Establece privilegios elevados para ejecutar un servicio.
tty	Configura un contenedor con una terminal TTY.
restart	Defina una política de reinicio que la plataforma deberá aplicar en la terminación de un contenedor.
build	Construye una imagen Docker mediante un archivo dockerfile
ports	Especificar los puertos que expone un contenedor.
networks	Define la Docker Network asociado a los contenedores de un servicio.
volumes	Define Docker Volumes (Named, Anonymous volumes) que se asocia a los contenedores para un servicio.

Docker Compose incorpora el Docker Client docker-compose para el despliegue de múltiples contenedores para proporcionar un servicio de red. La estructura general de los co-

mandos mediante Docker CLI se muestra en el Comando 2.3.1.6 y los principales comandos para docker Compose se especifican en la Tabla 2.11:

```
docker compose [command] [options]
```

Tabla 2.11: Comandos Generales de Docker Compose, basado en [49].

Comando	Descripción
docker compose up	Construye y ejecuta servicios multi-contenedor.
docker compose down	Elimina servicios multi-contenedor.
docker compose ps	Muestra los procesos de Docker Compose.
docker compose exec	Ejecuta un comando en un contenedor en ejecución.
docker compose stop	Detiene la ejecución de un contenedor.

2.4 DISEÑO

2.4.1 Escenarios

- **Implementación individual:** En este escenario, se implementa los servicios de red descritos en el Capítulo 1 utilizando contenedores Docker. Como resultado, se construyen los archivos Dockerfile para el despliegue de contenedores que ejecutan estos servicios. Posteriormente, se llevan a cabo pruebas de funcionamiento, para verificar los servicios contenerizados.

  Es importante destacar, que no se realizaran pruebas de rendimiento individuales por cada uno de los servicios resultantes. Esto se debe a que muchos de estos servicios, como FTP y HTTP, dependen de otros, como DHCP y DNS, debido a que requieren direccionamiento y resolución DNS para funcionar correctamente. Por lo tanto, se propone el despliegue conjunto utilizando Docker CLI, con el fin de analizar los parámetros de rendimiento y obtener datos más precisos y realistas.

- **Implementación Conjunta:** Una vez implementada la solución conjunta mediante Docker CLI, se procede a implementar una solución basada en el uso de Docker Compose, donde se utilizan dos placas Raspberry Pi. De las cuales, una placa despliega los servicios anteriores mediante el archivo YAML y en la segunda, se emplea exclusivamente los servicios de DHCP y Routing. Esto se hace con el propósito de verificar la capacidad de enrutamiento y probar la portabilidad de los contenedores.

A continuación, se describe el hardware, software y herramientas a emplear para implementar estos dos escenarios.

2.4.2 Hardware

De acuerdo con lo mencionado en la sección anterior, se utilizan placas de desarrollo Raspberry Pi tanto para implementaciones individuales como para implementaciones conjuntas.

También, para cumplir con los requisitos de puertos del servicio de routing especificados en la Tabla 2.2, se utilizan adaptadores USB-Ethernet de la marca Realtek modelo RTL8152 con capacidad de hasta 100baseT/FullDuplex, para ampliar la cantidad de puertos físicos.

Para llevar a cabo las pruebas de los servicios, se emplean 3 dispositivos de cómputo, dos para probar el funcionamiento de los servicios y el tercero para monitorear el rendimiento del host. Las descripciones de los equipos, se detalla en la Tabla 2.12.

Tabla 2.12: Comandos Generales de Docker Compose, basado en [50]-[52].

Marca - Modelo	**Procesador**	**Memoria Ram**	**Sistema Operativo**
LENOVO IdeaPad S410p	Intel(R) Core(TM) i5-4200 de 64bits CPU@1.6GHz	8 GB	Windows 10 Pro
Toshiba Satellite S55-B5157	Intel(R) Core(TM) i7 i7-5500U de 64bits CPU@2.4GHz	6GB	Windows 10 Home
HP 15-ef1xxx	AMD Ryzen 5 4500U de 64bits CPU@2.38GHz	8GB	Windows 11 Home
Raspberry Pi 4B	Broadcom BCM2711 Cortex-A72 (ARM v8) de 64bit @1.8GHz	4GB	Debian 11 (Bullseye)

2.4.3 Servicios de Red

Los servicios de red que se van a implementar en contenedores Docker se basan en servidores de Linux sobre una arquitectura cliente-servidor. En este sentido, incorporan un daemon que ejecuta y administra los servicios en base a parámetros de configuración predefinidos. A continuación, se describen cada uno del software que se emplea para desplegar los servicios, además del tipo de implementación sobre contenedores.

2.4.3.1 DHCP: Internet Systems Consortium DHCP (ISC DHCP)

ISC DHCP es uno de los servidores DHCP más empleado para el direccionamiento y configuraciones de red. Este servidor utiliza el demonio dhcpd para la administración y configuración del servicio [53]. Para la implementación, se emplea una configuración dinámica de direccionamiento IPv4. Además, proporciona información de configuración de red adicional como: direcciones de Servidores DNS y Gateway.

2.4.3.2 DNS: Berkeley Internet Name Domain (Bind9)

Bind de ISC es uno de los servidores DNS más populares de opensource para las distribuciones de Linux. Este servidor permite una alta flexibilidad en las configuraciones para la asociación y traducción de *domain names* con direcciones IPv4 e IPv6 [54]. El servidor DNS que se implementa es de tipo maestro, es decir que aloja localmente los registros primarios de las zonas DNS[27] y da respuesta a las solicitudes de resolución de nombres.

2.4.3.3 FTP: Very Secure File Transference Protocol (VSFTP)

Vsftp es un servidor FTP bajo licencia de GPL, que permite la transferencia de archivos o directorios de manera sencilla, rápida y segura. Implementa diversas funciones, como acceso a través de usuarios Anonymous [28], locales [29], o usuarios virtuales [30]. Además, cuenta con dos modos de transferencia FTP Modo Activo o Modo Pasivo [57]. El servicio FTP que se implementa sobre contenedores, emplea el modo activo para la transferencia de archivos personalizados de usuarios locales configurados. El servidor está diseñado para encerrar a los usuarios en su propio sistema de archivos, es decir que los usuarios se encuentran en un entorno chroot [31].

2.4.3.4 HTTP: NGINX

Nginx es un servidor HTTP, reverse proxy, balanceador de carga y proxy de correo de open source. NGINX es considerado como un servidor de alto rendimiento, alta disponibilidad, escalabilidad y bajo consumo de recursos [58]. El servicio HTTP basado en Nginx que se implementa, emplea el servidor web basado en virtual blocks [32], para dotar al servicio de

---

[27] **Zonas DNS:** Son la representación de los dominios o subdominios de resolución DNS y contienen registros DNS para ese dominio. Son administrados por un servidor de primer nivel. Existen dos tipos de zonas que son: Fordware Zones, que contienen registros DNS que asignan *domain names* a direcciones IP y las *Revers Zones*, que contienen registros DNS que asignan direcciones IP a *domain names* [55].

[28] **Anonymous User:** Es un usuario ftp administrado internamente por el servidor FTP, y generalmente tiene acceso limitado a los recursos del sistema de archivos [56].

[29] **Local User:** Es un usuario no ftp que cuenta con credenciales del sistema host, para la autenticación en el servidor FTP [56].

[30] **Virtual User:** Es otro tipo de usuario ftp administrado por el servidor FTP y con información de acceso registrada en una base de datos [56].

[31] **Chroot:** Es un entorno virtual, que permite confinar a un usuario FTP dentro de un sistema de archivos dedicado fuera del sistema operativo anfitrión. Esto permite que el usuario únicamente pueda interactuar con el sistema de archivos establecido y no fuera de este [56].

[32] **Virtual Blocks:** Son instancias virtuales de servidores que permiten recibir y responder solicitudes de manera independiente dentro del servidor Nginx. Este concepto es similar a los Virtual Hosts de Apache [58].

varias páginas web.

2.4.3.5 VoIP: Asterisk

Asterisk de Sangoma es un framework de opensourece bajo licencia GPL, para el desarrollo de aplicaciones de comunicación multiprotocolo y en tiempo real, como aplicaciones de video y voz de alta calidad [59]. Asterisk puede implementar elementos de una arquitectura VoIP, como IP PBX [33], VoIP Gateway [34], Voicemail server, Calll center, entre otros [62].

El servicio VoIP que se implementará será de tipo PBX para la configuración y administración de extensiones VoIP, donde se configurarán 4 extensiones de estaciones telefónicas IP, de las cuales se emplearán 2 para las pruebas de funcionamiento.

2.4.3.6 Router: FRRouting

FRRouting es un software de código abierto con licencia GPLv2, basado en el kernel de Linux, que proporciona servicios de enrutamiento IP. Incluye un conjunto de protocolos de enrutamiento estándar, como RIP, OSPF, BGP, IS-IS, entre otros [63]. Para la implementación actual, se utiliza el protocolo OSPF para el enrutamiento entre equipos Raspberry Pi. Dado que OSPF es uno de los protocolos más ampliamente utilizados en la actualidad.

2.4.4 Software asociado

Para permitir que los servicios sean accesibles para los clientes mediante conexión inalámbrica, se llevará a cabo la implementación de una función de Access Point en la placa Raspberry Pi, utilizando el software Hostap[35]. Además, para la instalación de dependencias, el acceso y la configuración de la placa, se utilizará el servicio de acceso remoto SSH[36]. Esto permitirá acceder a la configuración del equipo, tanto de forma inalámbrica como mediante cualquier conexión por cable Ethernet.

---

[33] **Private Branch Exchange (PBX):** Es la central de conmutación telefónica, donde se gestiona el tráfico interno y externo entre las estaciones telefónicas [60].

[34] **VoIP Gateway (VoIP-G):** Es un elemento de borde que permite la conexión de una red telefónica interna con otros sistemas de telefonía legacy (PSTN) y/u otras redes VoIP [61].

[35] **Access Point: Host Access Point (Hostap):** Hostap es un controlador de Linux para la administración de redes inalámbricas WLAN basadas en el estándar IEEE 802.11. Esto permite a un dispositivo compatible con tecnologías 802.11, actuar como un punto de acceso [64]. Algunas de las funcionalidades que ofrece este servicio son Gestión básica de IEEE 802.11a/b/g/n/ac/ax, servicio de autenticación RADIUS, soporte de encriptación WEP, WPA/WPA2 y TKIP/CCMP.

[36] **Secure Shell (SSH):** SSH es un protocolo para la comunicación remota entre dos sistemas, que funciona bajo una arquitectura cliente-servidor. Este tipo de comunicación emplea encriptación para el inicio de sesión y transferencia de archivos [65].

2.4.5 Herramientas de Medición

Para llevar a cabo pruebas de funcionamiento y rendimiento en un sistema contenerizado, se utilizan diversas herramientas de software específicas para medir parámetros como el uso de memoria RAM, la utilización de CPU, la carga promedio de CPU[37], el tráfico de entrada y salida, y la temperatura de la CPU. Estos parámetros se emplean como indicadores físicos, como de uso de recursos para analizar el performance de la implementación. En la Tabla 2.13, se describen estas herramientas, junto con una breve explicación de cada una.

Tabla 2.13: Herramientas de software, basado en [67]-[70].

Herramientas	Descripción	Parámetro de Medición
Wireshark	Es un software de código abierto utilizado para analizar paquetes de red, también conocido como sniffer. Permite capturar paquetes de datos en conexiones de red, lo que facilita la realización de auditorías de ciberseguridad y el diagnóstico de problemas de red.	Análisis de tráfico: DHCP, DNS, FTP, HTTP, SIP, RTP
docker stats	Es un comando que ofrece Docker para mostrar estadísticas de uso de recursos de los contenedores ejecutados actualmente.	Estadísticas de uso de contenedores Docker.
htop	Es una herramienta de software bajo licencia GPL para la visualización de procesos y uso del sistema.	Estadísticas de uso de procesos del sistema.
rpi-monitor	Es una herramienta de software para la medición de parámetros del sistema GNU/Linux y los periféricos conectados. Almacena estadísticas en la base de datos Round Robin local e integra un servidor web que permite mostrar el estado actual y las estadísticas.	Análisis del rendimiento general del sistema.

2.4.6 Container, imagen, volumes y networks

La elección de la imagen base es crucial al construir una imagen de contenedor. Dado que, durante la instalación de dependencias y otros componentes, se añaden capas adicionales que aumentan el tamaño de la imagen base. Por lo tanto, es importante utilizar una imagen base ligera para ahorrar recursos de almacenamiento. En este contexto, las imágenes basadas en Alpine Linux [38] son una opción recomendada, ya que ofrecen un ahorro considerable de recursos en las imágenes resultantes y en las instancias de contenedor, como se muestra en [73]. Por otro lado, como las imágenes de los servicios como Nginx, Asterisk

[37] **CPU Load Average:** Es una métrica esencial para monitorear recursos y representa la carga en el sistema durante diversos lapsos de tiempo. Es una medida adimensional y su interpretación varía según la cantidad de núcleos en el sistema. Por ejemplo, en un sistema con un solo núcleo, una carga promedio de 1.00 indica que la CPU está completamente ocupada, alcanzando el 100 % de uso [66].

[38] **Alpine Linux:** Es una distribución independiente de Linux para uso no comercial de aplicaciones de propósito general, que ofrece un entorno de Linux completo, seguro, simple y de uso eficiente de los recursos [71]. Docker Hub cuenta con una imagen oficial de Alpine Linux, cuyo tamaño de imagen es de tan solo 5MB [72].

y FRRouting, ya que cuentan con imágenes dedicadas en la arquitectura ARM32v7, no se realiza la construcción completa de estos servicios. En su lugar, se construye los archivos basados en estas imágenes, con el propósito de aprovechar la funcionalidad práctica de estas imágenes.

Los volumes a emplear son del tipo Named Volumes y Bind Mounts. Los named volumes se emplean con el propósito de mantener la persistencia de los registros de los servicios contenerizados, así como para compartirlos con otros contenedores. Por otro lado, los Bind Mounts se emplean para configurar los archivos de los contenedores desde el host.

En cuanto a las docker network, se emplea únicamente la red tipo “host”. Esta elección se basa en las consideraciones de rendimiento presentadas en [74], [75], donde han demostrado que las redes tipo “Bridge“ presentan un rendimiento inferior en comparación con otros tipos de redes. Por consiguiente, la utilización de redes tipo “host“, se orienta a optimizar el rendimiento de la implementación. A continuación, en la Tabla 2.14 se resume las características de implementación para los servicios de red contenerizados mediante Docker, en donde se detalla el servicio de red a implementar, la aplicación asociada al servicio, aplicaciones que interfieran en el servicio, imagen base, network y volumes Docker.

Tabla 2.14: Detalle de implementación de servicios sobre contenedores Docker.

Servicios	Aplicación	Incompatible	Imagen Base	Network	Volume
DHCP	ISC DHCP	dhcpcd	alpine:3.11	Host	Named volume
DNS	ISC Bind	dnsmasq	alpine:3.11	Host	Named volume
FTP	Vsftpd	vsftpd	alpine:3.11	Host	Named volume
HTTP	Nginx	httpd	nginx:1.22.1-alpine	Host	Named volume Bind Mount
VoIP	Asterisk	N/A	christoofar/asterisk	Host	Named volume Bind Mount
Routing	FRRouting	N/A	frrouting/frr:v7.5.1	Host	Named volume

2.4.7 Interfaces de Red y Direccionamiento

Para las implementaciones, se utiliza direcciones IPv4 para dos categorías: tipo C para entornos locales para pruebas mediante Docker CLI y tipo A para entornos remotos para Docker Compose.

En cuanto a las interfaces, la primera placa Raspberry Pi, llamada “RPI-I”, emplea dos tipos

de interfaces: físicas y virtuales. Las interfaces virtuales son creadas a partir de la interfaz física “Wlan0”, las cuales se asignan a los servicios contenerizados. Estas interfaces son de la interfaz “wlan0:1” a “wlan0:5”. Además, se tiene una última interfaz virtual “wlan0:6” que se emplea para el monitoreo mediante Rpi-Monitor. En cuanto a las interfaces físicas, se añaden tres módulos USB-Ethernet, para tener un total de cuatro interfaces Ethernet: “eth0”, “eth1”, “eth2” y “eth3”. Donde la interfaz “eth0” se reserva para configuraciones mediante la red local, mientras que Eth1 a Eth3 se utilizan para brindar los servicios contenerizados. El detalle de direccionamiento se muestra en la Tabla 2.15.

Tabla 2.15: Direccionamiento IP de servicio para la RPI II.

Servicios de Red	Interfaz	Dirección IPv4/Mascara
SSH | Hostap | DHFP I	Wlan0	192.168.0.1/24
SSH | DNS	Wlan0:1	192.168.0.2/24
SSH | FTP	Wlan0:3	192.168.0.4/24
SSH | HTTP	Wlan0:4	192.168.0.5/24
SSH | VoIP	Wlan0:5	192.168.0.6/24
SSH | Rpi-Monitor	Wlan0:6	192.168.0.7/24
SSH	Eth0	(Cliente DHCP)
SSH | Routing | DHCP II	Eth1	192.168.1.1/24
SSH | Routing | DHCP III	Eth2	192.168.2.1/24
SSH | Routing	Eth3	10.0.1.1/30

Para el caso de la segunda placa Raspberry Pi, llamada “RPI-II”, se emplean únicamente interfaces Ethernet “eth0” a la “eth3”. Empleando la interfaz “eth0” para configuración, como en la placa anterior. El detalle de direccionamiento se muestra en la Tabla 2.16.

Tabla 2.16: Direccionamiento IP de servicio para la RPI II.

Servicios de Red	Interfaz	Dirección IPv4/Mascara
SSH	Eth0	(Cliente DHCP)
SSH | Routing | DHCP I	Eth1	192.168.1.1/24
SSH | Routing | DHCP II	Eth2	192.168.2.1/24
SSH | Routing	Eth3	10.0.1.2/30

De acuerdo con el diseño anterior, en la Figura 2.10, se muestra la distribución lógica de la implementación sobre la placa Raspberry Pi.

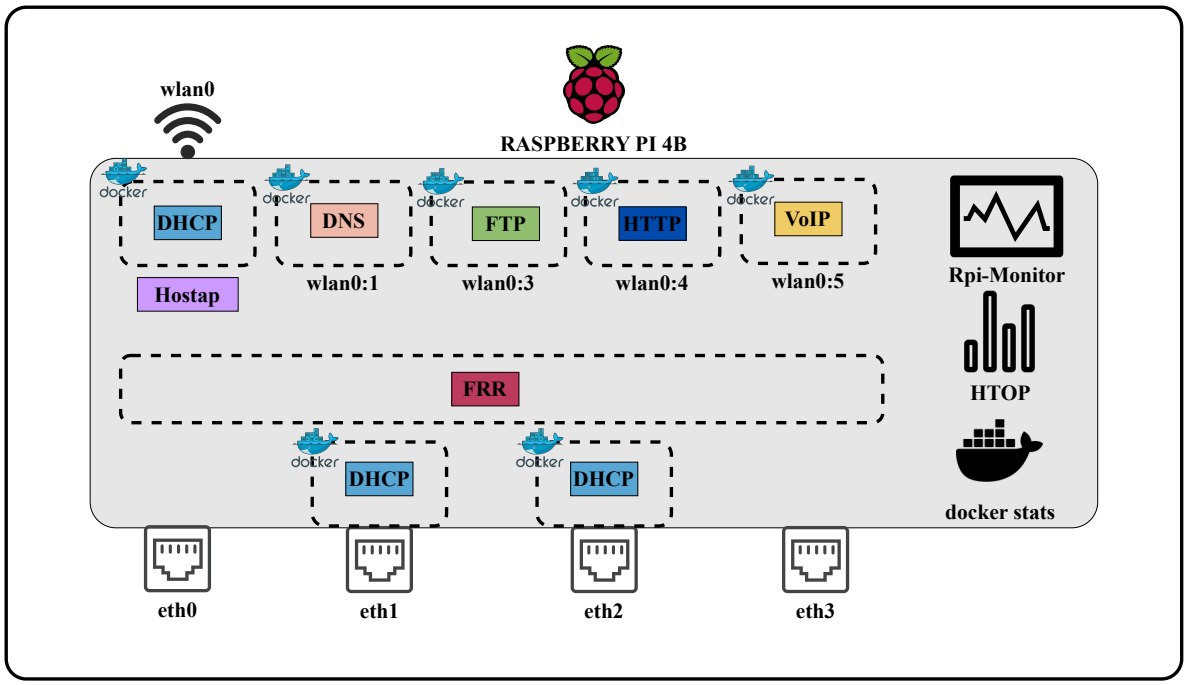


Figura 2.10: Diseño: Distribución lógica de los contenedores Docker sobre la placa Raspberry pi.

2.4.8 Topología de Red

Tomando en cuenta las consideraciones previas, se presenta las topologías para cada uno de los escenarios propuestos.

Para la implementación individual y pruebas de los servicios contenerizados, se utiliza la Topología de la Figura 2.11.

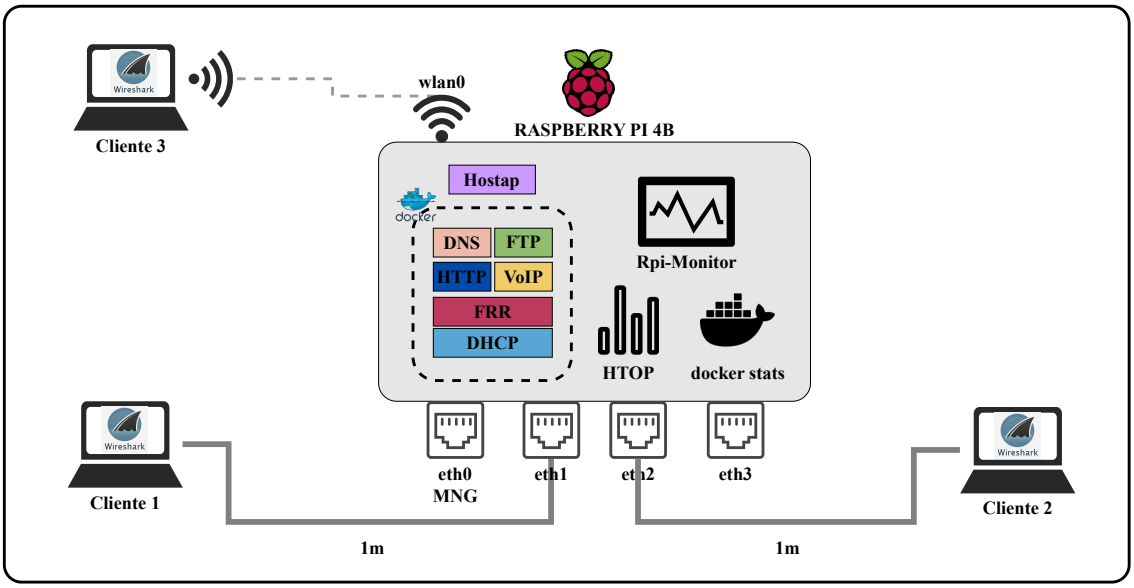


Figura 2.11: Diseño: Topología de despliegue de servicios basada en contenedores mediante Docker CLI.

Para la implementación conjunta mediante Docker Compose se emplea la topología de la

Figura 2.12, donde se implementa parte de la topología anterior.

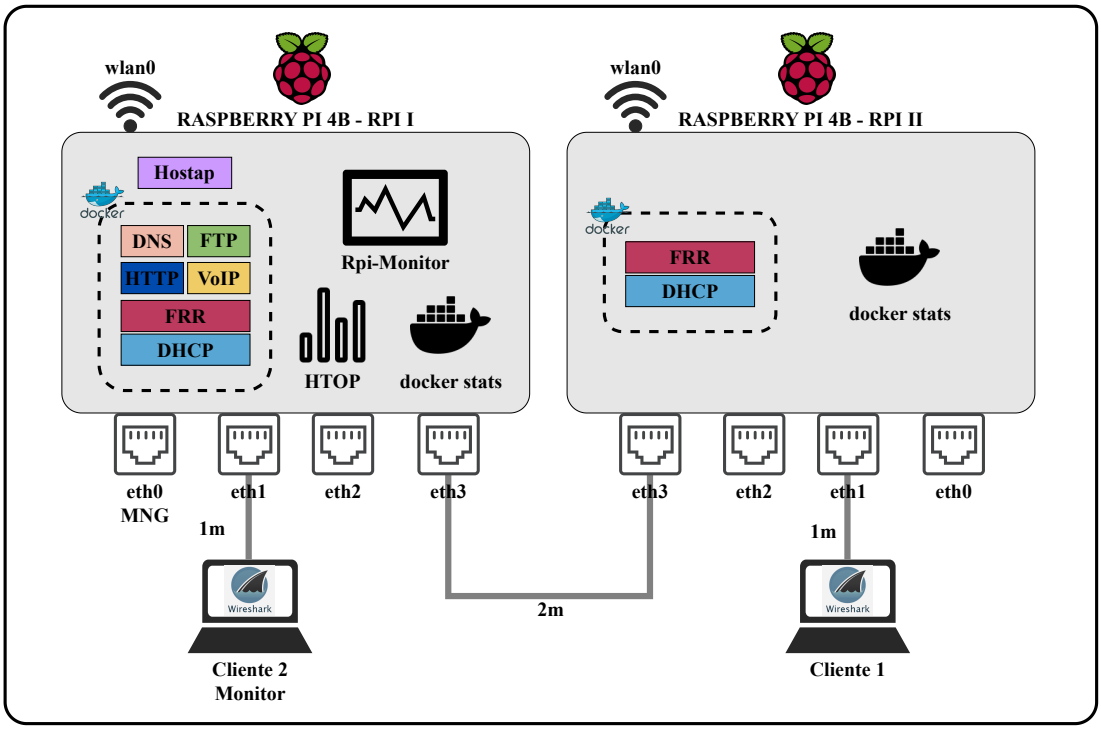


Figura 2.12: Diseño: Topología de despliegue de servicios basada en contenedores mediante Docker Compose.

2.5 IMPLEMENTACIÓN

Los pasos de la implementación se muestran en la Figura 2.13. Los cuales se detallan en el Anexo I, específicamente en la sección de “Pasos de Implementación”.

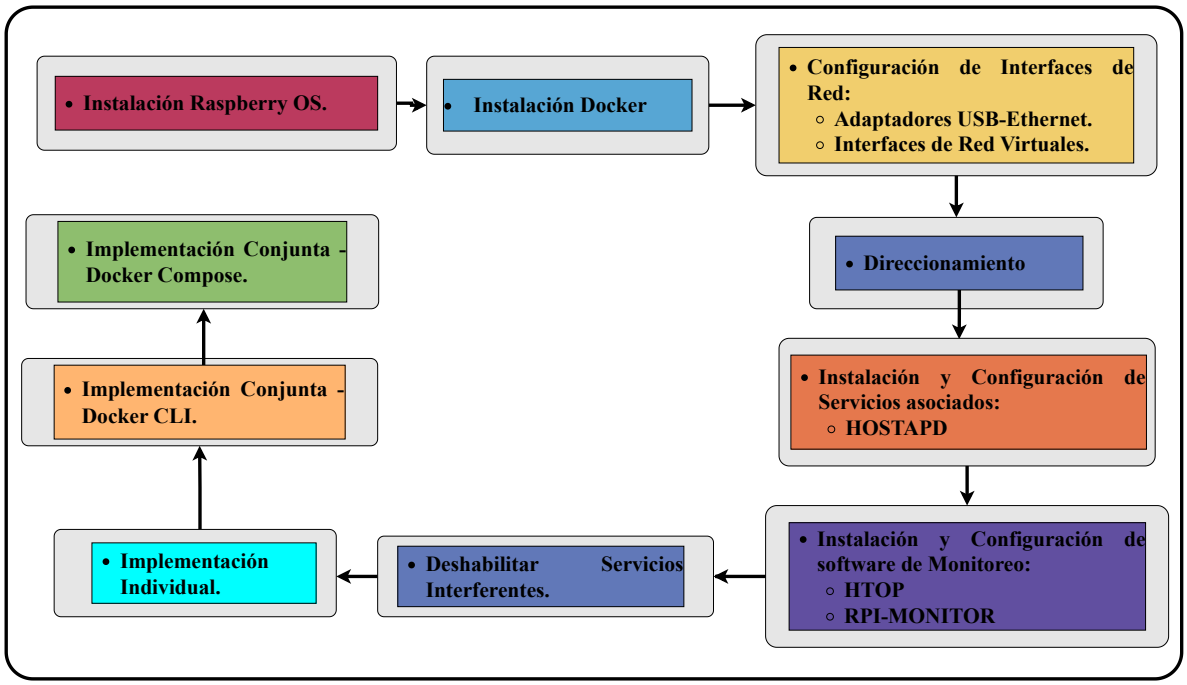


Figura 2.13: Implementación: Diagrama de Flujo de Implementación.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Una vez que se han implementado las soluciones individuales usando Docker CLI y se ha desplegado la solución conjunta a través de Docker Compose, se proceden a analizar los resultados de las pruebas de funcionamiento de los prototipos diseñados.

3.1 RESULTADOS

Como resultado de la implementación, en el primer escenario, Figura 5.3, se generan los archivos finales (“Dockerfiles”), junto con los archivos de dependencias correspondientes para los servicios de DHCP, DNS, FTP y HTTP.

Para el segundo escenario, Figura 5.8, se obtiene los archivos YAML de Docker Compose para ambas placas Raspberry Pi involucradas en este proceso. Para la placa RPI-I, que despliega los servicios contenerizados principales destinados al acceso remoto del cliente 1, se encuentra el archivo (“docker-compose.yaml”). En el caso de la segunda placa, RPI-II, que aloja los servicios de DHCP y enrutamiento para facilitar el acceso y enrutamiento hacia la placa RPI-I del cliente 1, se utiliza el archivo (“docker-compose-routing.yaml”).

Todos estos archivos, incluyendo los (“Dockerfiles”), (“docker-compose.yaml”) y (“docker-compose-routing.yaml”), se encuentran disponibles en el ANEXO I, dentro de la sección dedicada a Docker, para su respectiva implementación a través de Docker CLI y Docker Compose. Además, las imágenes correspondientes a los servicios implementados se encuentran alojadas en el repositorio de Docker Hub en el ANEXO II.

Por otro lado, mediante la herramienta Wireshark se obtienen las capturas completas del tráfico de los 3 clientes de la Figura 2.11, para el caso de Docker CLI y de los 2 clientes de la Figura 2.12, para Docker Composes. Estas capturas se almacenan en un archivo (“.pcapng”) del cual se extraen diagramas de flujo y gráficas de entrada y salida (Gráficas E/S) del tráfico de los servicios. Estos resultados, junto a los datos, ficheros relacionados con los servicios implementados y resultados de funcionamiento de los clientes involucrados, se encuentran en el ANEXO I en el apartado de “Resultados”.

3.1.1 Implementación Mediante Docker CLI

A continuación, se presentan los resultados obtenidos por el Cliente 1, en cuanto a funcionamiento de los servicios implementados a través de Docker CLI. Estos resultados para los servicios DHCP, DNS, FTP y HTTP son análogos para los Clientes 2 y 3.

En relación con los datos generados, todos los servicios comparten registros de las solici-

tudes de los clientes para los servicio implementados, excepto el servicio DHCP. Esto se debe a que el servicio DHCP implementa un servicio contenerizado para cada cliente, lo que resulta en registros separados para cada uno de ellos.

3.1.1.1 Servicio de Routing

En esta sección, no se realizaron pruebas de enrutamiento. En su lugar, se asignaron direcciones IPv4 estáticas a las interfaces Ethernet “eth1” a “eth2”, con el propósito de asociar estas interfaces con los contenedores DHCP. De acuerdo con el direccionamiento establecido en la Tabla 2.15.

3.1.1.2 Servicio de Direccionamiento Mediante DHCP

Como resultado de la implementación del servicio DHCP contenerizado, se presentan en la Tabla 3.1, los datos recopilados de los 2 clientes DHCP a través de conexiones Ethernet y el 1 cliente DHCP a través de WLAN. Esta información puede ser cotejada con los archivos de arrendamiento DHCP obtenidos (“dhcpd.leases”) y (“dhcpd.leases∼”), en los volumes correspondientes a cada uno de los contenedores DHCP.

Tabla 3.1: Servicio DHCP: Resultados de servicio dhcp para los 4 clientes.

Información de Red	**Cliente DHCP 1 Ethernet - eth1**	**Cliente DHCP 2 Ethernet - eth2**	**Cliente DHCP 3 WLAN - Wlan0**
Dirección de subred	192.168.1.0	192.168.2.0	192.168.0.0
Dirección IPv4	192.168.1.10	192.168.2.10	192.168.0.10
Mascara de Subred	255.255.255.0	255.255.255.0	255.255.255.0
Dirección de Gateway	192.168.1.1	192.168.2.1	192.168.0.1
Direcciones de Servidores DNS	192.168.0.2 1.1.1.1 8.8.8.8	192.168.0.2 1.1.1.1 8.8.8.8	192.168.0.2 1.1.1.1 8.8.8.8
Dirección MAC	C4:54:44:AC:34:9E	3C-97-0E-CC-19-4F	64:6C:80:0C:A5:E3

El diagrama de flujo representado en la Figura 3.1 ilustra el proceso de arrendamiento DHCP para el cliente 1. Este proceso puede ser confirmado mediante los registros de salida del servicio DHCP en Docker, comúnmente denominados (“logs”). En este proceso, se adquiere una dirección IP mediante el intercambio de mensajes DHCP y se generan re-

gistros de arrendamiento. Una vez que se ha obtenido la información de red, el proceso continúa con la renovación del arrendamiento. Esto implica que el cliente envía solicitudes para extender su arrendamiento, seguidas de confirmaciones a intervalos específicos, en este caso, cada 60 segundos, como se muestra en la gráfica de entrada y salida de tráfico DHCP en la Figura 5.4 del Anexo VI. Este patrón se deriva de la configuración del tiempo mínimo de arrendamiento establecido en el archivo (“dhcpd.conf”).

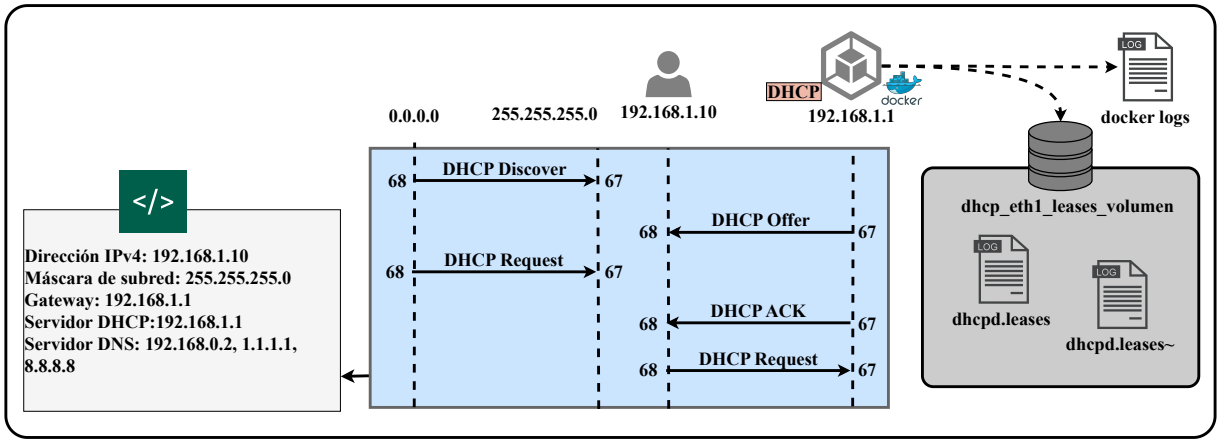


Figura 3.1: Servicio DHCP: Diagrama de flujo del proceso de arrendamiento DHCP para el Cliente 1.

3.1.1.3 Pruebas de Conexión

En la Tabla 5.4 del Anexo VI, se exponen los resultados de las pruebas de conexión a los servidores contenerizados. Los valores incluyen retardos mínimos, máximos y promedio registrados por los clientes, destacando retardos tan bajos entre 1 ms y $<$1 ms, para los clientes conectados por Ethernet y retardos entre 2ms hasta 1921ms para el cliente conectado al Access Point. En el caso del cliente conectado a través de una red inalámbrica, se observan retardos críticos. Esto es especialmente relevante considerando los retardos máximos aceptables para el servicio VoIP, que deben llegar a ser menores a 150 ms.

3.1.1.4 Servicio de Resolución de Nombres Mediante DNS

El resultado del proceso de resolución DNS, como se muestra en el diagrama de flujo de la Figura 3.2, ofrece una visión general del flujo de tráfico DNS necesario para realizar las resoluciones directas e inversas de las solicitudes DNS enviadas por el cliente 1. Para verificar este proceso, podemos consultar los archivos de las zonas DNS directa (“TIC2023.com.db”) e inversa (“0.168.192.rev”), así como el registro (“named.log”) del servidor DNS correspondiente. Sin embargo, a diferencia del servicio DHCP, el registro DNS generado presenta un volumen mucho mayor comparado con los archivos de registros DHCP, llegando a un vo-

lumen de 7.89 MB. Esto se da como resultado de las numerosas solicitudes echas por los clientes. Dato que se refleja en las gráficas E/S del tráfico DNS de la Figura 5.5 del ANEXO VI, volumen tanto de las solicitudes DNS general enviadas y recibidas, como las solicitudes generadas durante las pruebas.

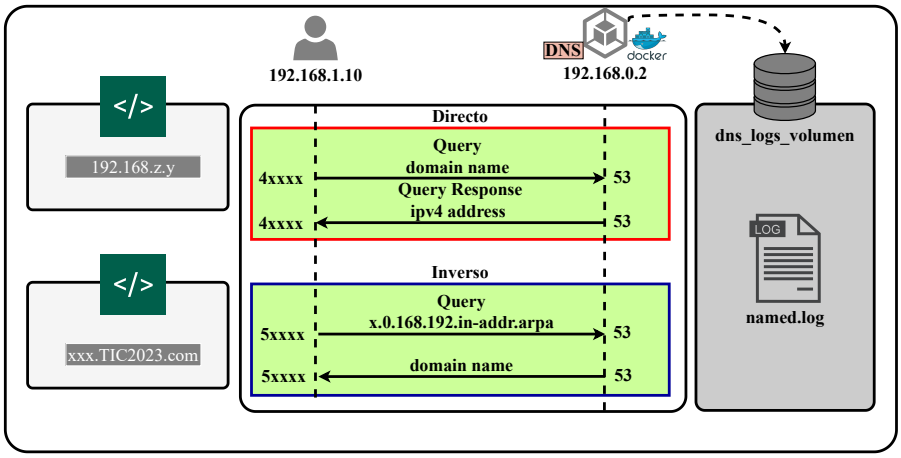


Figura 3.2: Servicio DNS: Diagrama de flujo del proceso de resolución DNS para el Cliente 1.

3.1.1.5 Servicio de Transferencia de Ficheros Mediante FTP

Como resultado de la implementación del servicio FTP contenerizado, se generan dos archivos (“Admin.txt”). Estos archivos son producto de las transferencias FTP que se ilustran en la Figura 3.3. Todo este proceso se registra en el archivo (“vsftpd.log”), el cual documenta tanto el inicio como la finalización de las sesiones de los clientes, así como en el archivo (“xferlog”), donde quedan registradas las transferencias realizadas por los mismos. Además, en el gráfico de E/S de la Figura 5.6 del ANEXO VI, se puede observar el tráfico FTP correspondiente a las dos conexiones y transferencias FTP realizadas por el cliente.

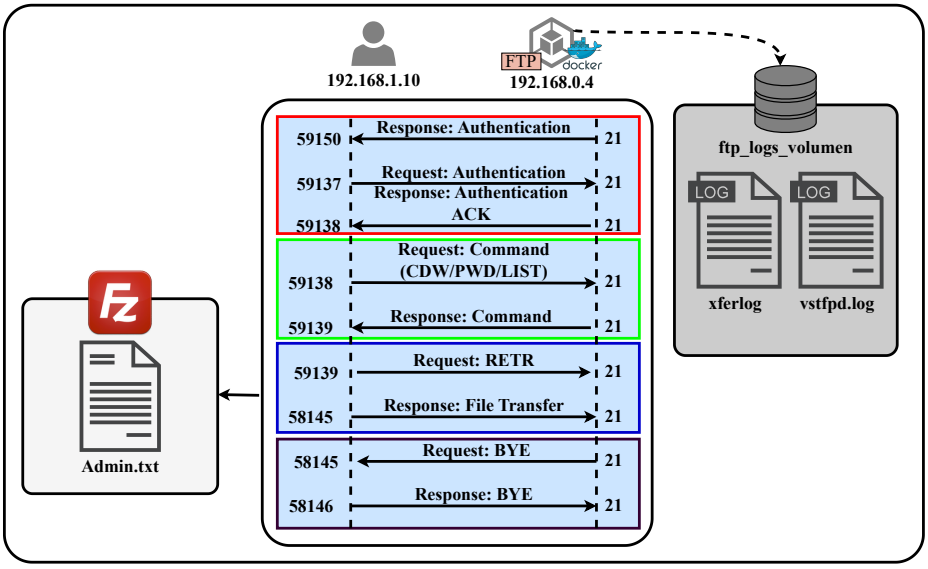


Figura 3.3: Servicio FTP: Diagrama de flujo del proceso de la trasferencia ftp para el cliente 1.

3.1.1.6 Servicio de Página Web Mediante HTTP

Para el servicio HTTP, se obtienen las tres páginas HTML para cada uno de los dominios: “www.tic2023.com”, “web1.tic2023.com” y “web2.tic2023.com”, configuradas en el servidor HTTP contenido. Además, los registros de acceso y errores HTTP generados por el servicio se almacenan en un volumen dedicado de Docker. Esto se puede observar en la Figura 3.4, que muestra el flujo de transferencia HTTP. y en la Figura 5.7 del ANEXO VI, donde se aprecia el gráfico de E/S del tráfico HTTP, el cual presenta tres picos correspondientes al tráfico HTTP generado.

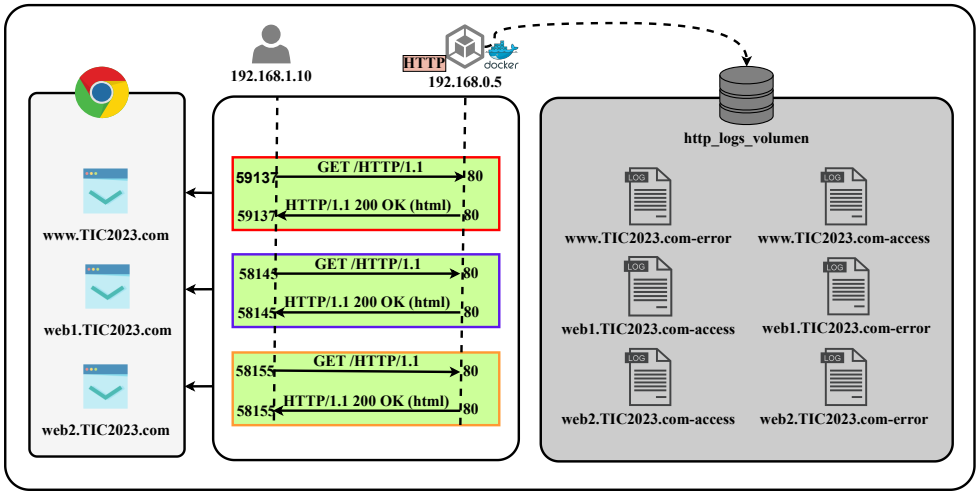


Figura 3.4: Servicio HTTP: Diagrama de flujo de la trasferencia http para el cliente 1.

3.1.1.7 Servicio de Telefonía IP Mediante VoIP

En la Figura 3.5 se muestra el flujo de las llamadas realizadas entre los dos clientes VoIP. El cual termina con un registro en el volumen Docker para el servicio VoIP, llamado (“messages”). Este mismo proceso se repite 3 veces, tanto para las llamadas realizadas desde la extensión 2001 a la extensión 2002, como viceversa.

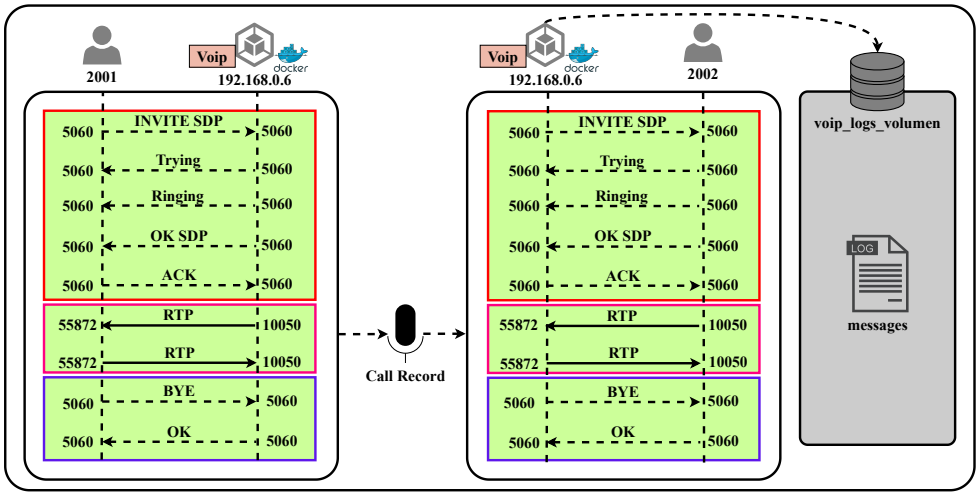


Figura 3.5: Servicio HTTP: Diagrama de flujo de la trasferencia http para el cliente 1.

El desglose pormenorizado de las llamadas VoIP efectuadas por los clientes VoIP 1 y 2, se muestra en la Figura 3.6, donde se presenta información, como la dirección IP de origen de la llamada, las extensiones SIP, la duración de la llamada, etc. Lo cual se describe en la Tabla 5.5 del ANEXO VI.

Start Time	Stop Time	Interlocutor inicial	Desde	A	Protocolo	Duración	Paquetes	Estado	Comentarios
2023-08-10 22:18:25	2023-08-10 22:19:00	192.168.1.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:34	19	COMPLETED	INVITE 401 401 200 200 200 200
2023-08-10 22:19:55	2023-08-10 22:20:32	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.1.10;transport=udp>	SIP	00:00:37	7	COMPLETED	INVITE 200
2023-08-10 22:21:21	2023-08-10 22:22:00	192.168.1.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:38	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-10 22:22:46	2023-08-10 22:23:12	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.1.10;transport=udp>	SIP	00:00:26	7	COMPLETED	INVITE 200
2023-08-10 22:23:38	2023-08-10 22:24:00	192.168.1.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:21	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-10 22:24:10	2023-08-10 22:24:30	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.1.10;transport=udp>	SIP	00:00:19	7	COMPLETED	INVITE 200

(a) Detalle SIP de las llamadas VoIP para el Cliente 1

Start Time	Stop Time	Interlocutor inicial	Desde	A	Protocolo	Duración	Paquetes	Estado	Comentarios
2023-08-10 22:18:29	2023-08-10 22:19:04	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:34	7	COMPLETED	INVITE 200
2023-08-10 22:19:58	2023-08-10 22:20:35	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:37	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-10 22:21:25	2023-08-10 22:22:03	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:38	7	COMPLETED	INVITE 200
2023-08-10 22:22:50	2023-08-10 22:23:16	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:26	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-10 22:23:42	2023-08-10 22:24:04	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:21	7	COMPLETED	INVITE 200
2023-08-10 22:24:14	2023-08-10 22:24:33	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:19	17	COMPLETED	INVITE 401 401 200 200 200

(b) Detalle SIP de las llamadas VoIP para el Cliente 2

Figura 3.6: Servicio VoIP: Detalle SIP de las llamadas VoIP.

Además, en la Figura 3.7, se muestra los datos del QoS de las llamadas, que incluyen las pérdidas de paquete, retardos y Jitter. El análisis de los resultados para una de las llamadas, se describen en la Tabla 5.6 del ANEXO VI.

Source Address	Source Port	Destination Address	Destination Port	SSRC	Start Time	Duración	Payload	Paquetes	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Estado
192.168.1.10	53859	192.168.0.6	10036	0x23f9513f	2023-08-10 22:24:16.022	14.12	g711U	707	0 (0.0%)	17.390000	19.999035	22.431000	0.001125	0.637133	1.168517	
192.168.1.10	51610	192.168.0.6	10002	0xa76bdcfb	2023-08-10 22:23:46.255	14.04	g711U	703	0 (0.0%)	17.228000	19.998917	22.808000	0.043188	0.722307	1.292470	
192.168.1.10	59599	192.168.0.6	10080	0x8971fc35	2023-08-10 22:22:56.361	16.58	g711U	830	0 (0.0%)	16.908000	19.999660	22.969000	0.018078	0.773235	1.255660	
192.168.1.10	58833	192.168.0.6	10072	0xe6bf33f4	2023-08-10 22:21:39.982	20.08	g711U	1005	0 (0.0%)	0.233000	19.998353	34.036000	0.071250	0.460680	2.601294	
192.168.1.10	58037	192.168.0.6	10020	0xfa26889b	2023-08-10 22:20:10.273	21.88	g711U	1095	0 (0.0%)	17.256000	19.999170	22.850000	0.006250	0.879417	1.526595	
192.168.1.10	65188	192.168.0.6	10004	0x831f11fb	2023-08-10 22:18:37.930	22.33	g711U	1118	0 (0.0%)	10.883000	19.991230	22.631000	0.285452	0.634148	1.040007	

(a) Servicio VoIP: Detalle RTP de las llamadas para el Cliente 1.

Source Address	Source Port	Destination Address	Destination Port	SSRC	Start Time	Duración	Payload	Paquetes	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Estado
192.168.2.10	57651	192.168.0.6	10020	0xe0cff07	2023-08-10 22:24:19.843	14.02	g711U	702	0 (0.0%)	18.459000	20.000160	21.839000	0.031562	0.397028	0.602930	
192.168.2.10	49894	192.168.0.6	10082	0x6ce06e9b	2023-08-10 22:23:49.881	14.18	g711U	710	0 (0.0%)	18.348000	19.998697	21.709000	0.011312	0.494975	0.701191	
192.168.2.10	56151	192.168.0.6	10002	0xfb6d791b	2023-08-10 22:23:00.321	16.34	g711U	818	0 (0.0%)	18.417000	19.999360	21.807000	0.002125	0.375521	0.615321	
192.168.2.10	53448	192.168.0.6	10084	0x521b1ac0	2023-08-10 22:21:43.529	20.30	g711U	1016	0 (0.0%)	8.808000	19.999654	31.071000	0.021648	1.651855	5.102633	
192.168.2.10	59753	192.168.0.6	10070	0x9fdb2e13	2023-08-10 22:20:14.287	21.56	g711U	1079	0 (0.0%)	8.744000	19.999810	30.661000	0.014938	4.952824	8.151049	
192.168.2.10	61472	192.168.0.6	10012	0xce79384b	2023-08-10 22:18:40.648	23.34	g711U	1168	0 (0.0%)	18.095000	19.999412	21.918000	0.011375	0.497813	0.818692	

(b) Servicio VoIP: Detalle RTP de las llamadas para el Cliente 2.

Figura 3.7: Servicio VoIP: Detalle RTP de las llamadas.

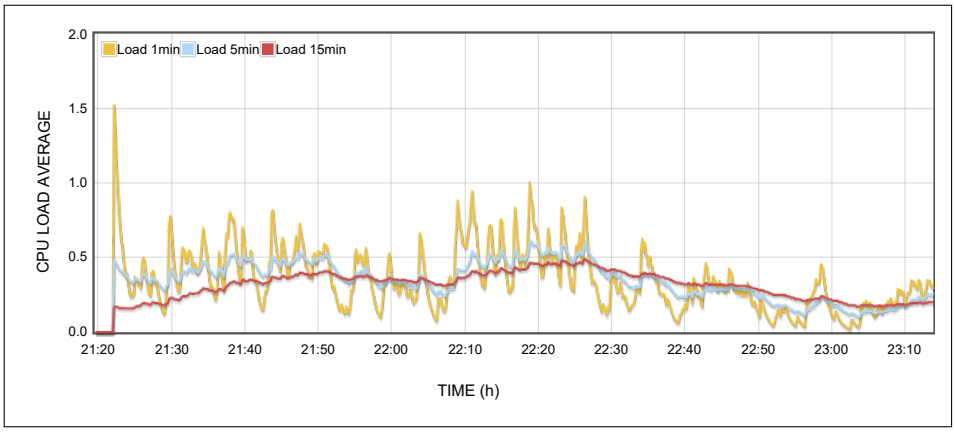
3.1.1.8 Uso de Recursos

Los resultados de rendimiento obtenidos a través de las herramientas “docker stats” y “htop” se detallan en la Tabla 5.7. Destaca el servicio DNS con un uso máximo de CPU del 15.02 %, seguido por el servicio FTP con un uso de hasta el 11.89 % y el servicio de VoIP con un 5.18 %. Los demás servicios contenerizados muestran valores relativamente bajos, todos por debajo del 1 %.

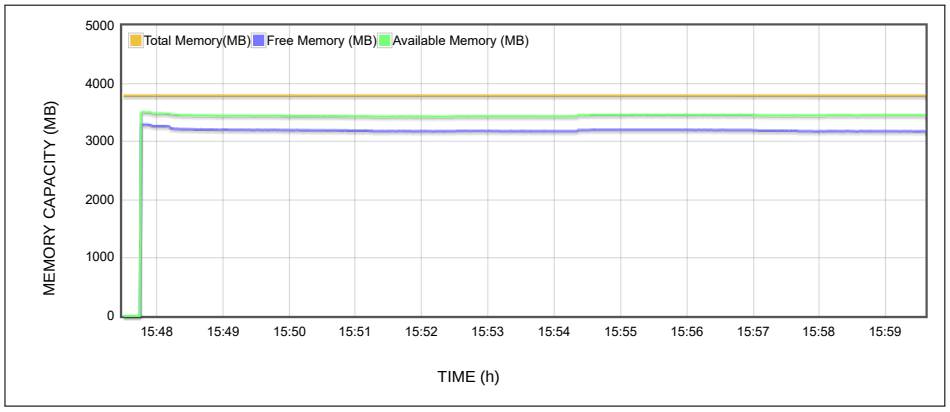
En cuanto al uso de memoria, el servicio DNS también presenta el valor más alto, con un

1.1 %, seguido del servicio de VoIP con un 0.8 %. Es importante destacar que estos valores de memoria son constantes para todos los servicios, tanto durante la implementación con Docker Compose como con Docker CLI.

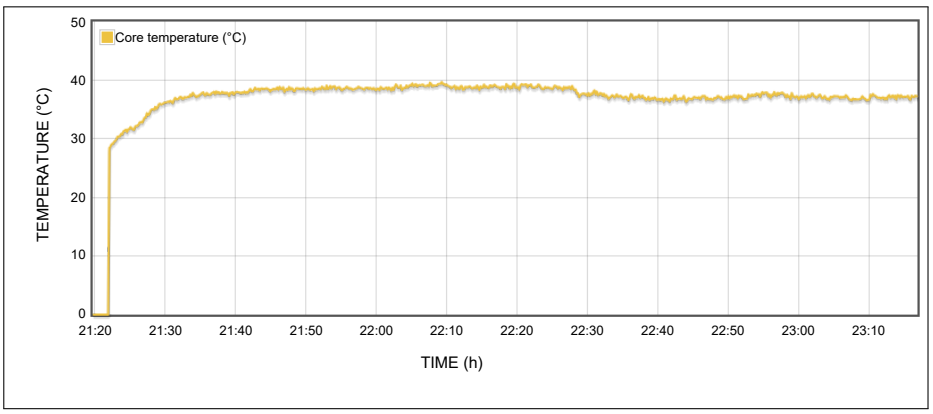
Mediante la Figura 3.8, se presentan las estadísticas de utilización de la CPU Figura 3.8a, memoria Figura 3.8b y la temperatura Figura 3.8c, recopiladas mediante la herramienta “Rpi-Monitor”. Estos datos abarcan el intervalo de tiempo desde las 21:20 hasta las 22:30.



(a) Uso de Recursos: Uso de CPU



(b) Uso de Recursos: Uso de Memoria.



(c) Uso de Recursos: Temperatura.

Figura 3.8: Uso de Recursos Mediante RPI-MONITOR para Docker CLI.

Como se puede observar en la Figura 3.8a, el indicador de carga computacional “load average” comienza con un valor inicial de 1.521, uso de CPU del 38 %, considerando que el 100 % se representa por el valor 4.0, dado que es un CPU Quad-Core. Posteriormente, esta carga disminuye hasta 0.256 (Uso de 6.4 %). A partir de este punto, se inician las pruebas de los servicios de red, divididas en secciones como se detalla en la Tabla 3.2.

Tabla 3.2: Pruebas de Rendimiento: Resultados de CPU Load Average Rpi-Monitor para Docker CLI

Servicio	**Período**	**Descripción**
DHCP	(21:00-21.30)	En esta primera sección de pruebas relacionadas con el servicio DHCP, se conectaron los 3 clientes de prueba. La carga alcanza un máximo de 0.5 (12.5 %) y finaliza con un pico de 0.777 (19.43 %). Este último pico podría deberse al inicio del envío de solicitudes DNS por parte de los clientes Ethernet, que se conectaron recientemente al servidor DNS.
Conexión	(21:30-21:35)	Al realizar conexión de los clientes a los servidores, se observan variaciones en la carga para cada uno de los clientes. Para el cliente 1, los valores de carga alcanzan hasta 0.569 (14.23 %). Para el cliente 2, se llega a valores de 0.696 (17.4 %) y para el cliente 3 la carga es de 0.572 (14.3 %).
Resolución DNS	(21:35-21:40)	Durante el proceso de resolución DNS directa, los valores alcanzados son de 0.536 para el cliente 1, 0.77 (19.25 %) para el cliente 2 y 0.54 para el cliente 3. Por otro lado, durante la resolución inversa llega a valores de 0.804 para el cliente 1, 0.7 (17.5 %) para el cliente 2 y 0.36 (9 %) para el cliente 3.
HTTP	(21:43-21.46)	En esta sección, la carga alcanza valores de 0.817 (20.43 %) para el cliente 1, 0.633 (15.83 %) para el cliente 2 y 0.673 (16.83 %) para el cliente 3. Además, se nota que, de acuerdo con el análisis de las gráficas E/S de tráfico DNS, el tráfico aumenta considerablemente durante el período de las pruebas HTTP y se mantiene elevado después de las pruebas.
FTP	(21:47-21:54)	Durante las pruebas de las 2 conexiones FTP, se registran valores máximos de carga de 0.725 (18.13 %) para el cliente 1, 0.59 (14.75 %) para el cliente 2 y 0.554 (13.85 %) para el cliente 3.
VoIP	(22:05-22:24)	Se observa que durante la primera llamada de la extensión 2001 a la extensión 2002, la carga de CPU alcanza su punto máximo, llegando a 1.003 (25.08 %). Para el resto de las llamadas, la carga se mantiene en un promedio de 0.7 (17.5 %).

La Figura 3.8b muestra el uso de memoria del sistema. Donde se observa que el consumo de recursos no cambia mientras se ejecutan las pruebas de los servicios. Esta observación se basa en la cantidad de memoria libre disponible, que comienza en 3229.67 MB durante el arranque del host y se mantiene en niveles cercanos a 3100.3 MB. Esto refleja un uso continuo de memoria de alrededor de 693.7 MB, considerando un total de 3794.33 MB.

En la Figura 3.8c se puede observar la temperatura del CPU pasa de 28.39°C, en el momento del arranque, y que aumenta hasta mantenerse a una temperatura promedio de 38.77°C, para luego de las pruebas de VoIP descender a 37.69°C.

En el ANEXO I, en el apartado de Resultados para Docker CLI, se pueden encontrar las capturas de los resultados para cada uno de los servicios dns, dhcp, ftp, http y voip, tanto para el cliente 1, cliente 2 y cliente 3; además de los resultados de monitoreo de los servicios capturadas mediante HTOP, Docker Stats y RPI-Monitor.

3.1.2 Implementación Conjunta Mediante Docker Compose

En los resultados del funcionamiento de servicios mediante Docker Compose, se destacan las mediciones realizadas por el Cliente 1. Esto ocurre después de recibir una dirección IP mediante DHCP en la red local de RPI II y establecer una conexión con RPI I utilizando OSPF para acceder a los servicios contenerizados. Los resultados de esta implementación, se enfocaron específicamente en los servicios de DHCP, VoIP y Routing, dado que los resultados de DNS, FTP y HTTP fueron similares a los obtenidos anteriormente.

3.1.2.1 Servicio de Direccionamiento Mediante DHCP

El direccionamiento IPv4 obtenido por ambos clientes DHCP se muestra en la Tabla 3.3.

Tabla 3.3: Servicio DHCP: Resultado de proceso de arrendamiento para el Cliente 1 y 2.

Información de Red	Cliente DHCP 1	Cliente DHCP 2
Dirección de subred	192.168.5.0	192.168.2.0
Dirección IPv4	192.168.5.10	192.168.2.10
Mascara de Subred	255.255.255.0	255.255.255.0
Dirección de Gateway	192.168.5.1	192.168.2.1
Direcciones de Servidores DNS	192.168.0.2 1.1.1.1 8.8.8.8	192.168.0.2 1.1.1.1 8.8.8.8
Dirección MAC	C4-54-44-AC-34-9E	3C-97-0E-CC-19-4F

Como se puede apreciar, ambos hosts pertenecen a subredes diferentes. En detalle, el cliente 1 está situado en la subred 192.168.5.0/24 del host RPI II. Por otro lado, el cliente 2 está ubicado dentro de la subred 192.168.2.0/24 del host RPI I .

3.1.3 Servicio de Routing

En la Figura 5.9 se presentan las tablas de enrutamiento de los contenedores de las placas RPI I y RPI II, evidenciando el descubrimiento OSPF de redes remotas. También se obtienen como resultado, los archivos de configuración de Frrouting almacenados en un volumen Docker, los cuales son: (“staticd”) que contiene datos de direccionamiento estático, (“zebra.conf”) que contiene la configuración del router Frr y (“daemons”) y (“ospfd”). sobre los protocolos de enrutamiento.

En el contexto de las implementaciones evaluadas, tanto en Docker CLI como en Docker Compose, se ha observado que los archivos almacenados en los volúmenes de Docker, desempeñan un papel fundamental en la configuración del servicio de enrutamiento. Esto debido a que los volumenes se los utiliza para migrar las configuraciones entre contenedores. Estrategia que ha demostrado ser sumamente beneficiosa, ya que simplifica de manera considerable el despliegue del servicio. Elimina la necesidad de llevar a cabo configuraciones completas cada vez que se inicia un nuevo contenedor, lo que ahorra tiempo y reduce la posibilidad de errores de configuración.

3.1.3.1 Pruebas de Conexión

En las Tablas 5.8 y 5.9 del ANEXO VII, se presentan los resultados de las pruebas de conexión entre los clientes, lo que permite verificar la calidad de las conexiones. Estos resultados indican que el retardo aumenta gradualmente a medida que se avanza hacia el extremo de la conexión, alcanzando un retardo promedio de aproximadamente 2 ms entre los clientes. Adicionalmente, en la Figura 5.10 del ANEXO VII, se muestra la ruta de extremo a extremo, lo que facilita la verificación de la trayectoria dentro de la topología diseñada.

Para confirmar la conectividad entre el cliente remoto y los servicios contenidos, se presentan los retardos para el cliente 1 conectado mediante Ethernet en la Tabla 5.10 del ANEXO VII, los cuales oscilan alrededor de 1 ms.

3.1.3.2 Servicio de Telefonía IP Mediante VoIP

En el servicio de VoIP, el proceso de llamadas se ilustra en la Figura 3.9, donde se muestran las llamadas realizadas. La mayoría de estas llamadas concluyeron exitosamente, como se indica con el estado “COMPLETED”. Sin embargo, también se puede observar que varias llamadas fueron rechazadas, identificadas por el estado “REJECTED”. Estos rechazos se debieron a procedimientos de llamada inadecuados llevados a cabo desde la aplicación.

Start Time	Stop Time	Interlocutor inicial	Desde	A	Protocolo	Duración	Paquetes	Estado	Comentarios
2023-08-16 12:07:02	2023-08-16 12:07:02	192.168.5.10	<sip:TOSHIBA@192.168.5.10>	sip:2002@192.168.0.6	SIP	00:00:00	5	REJECTED	INVITE 401 401
2023-08-16 12:07:53	2023-08-16 12:07:53	192.168.5.10	<sip:TOSHIBA@192.168.5.10>	sip:2002@192.168.0.6	SIP	00:00:00	5	REJECTED	INVITE 401 401
2023-08-16 12:11:17	2023-08-16 12:11:57	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.5.10;transport=udp>	SIP	00:00:40	7	COMPLETED	INVITE 200
2023-08-16 12:12:31	2023-08-16 12:12:52	192.168.5.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:21	19	COMPLETED	INVITE 401 401 200 200 200
2023-08-16 12:13:03	2023-08-16 12:13:30	192.168.5.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:27	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-16 12:14:02	2023-08-16 12:14:29	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.5.10;transport=udp>	SIP	00:00:27	7	COMPLETED	INVITE 200
2023-08-16 12:15:01	2023-08-16 12:15:02	192.168.5.10	<sip:TOSHIBA@192.168.5.10>	sip:2002@192.168.0.6	SIP	00:00:00	5	REJECTED	INVITE 401 401
2023-08-16 12:15:22	2023-08-16 12:15:53	192.168.5.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:31	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-16 12:16:18	2023-08-16 12:16:46	192.168.0.6	<sip:2002@192.168.0.6>	<sip:2001@192.168.5.10;transport=udp>	SIP	00:00:28	7	COMPLETED	INVITE 200
2023-08-16 12:18:18	2023-08-16 12:18:44	192.168.5.10	<sip:2001@voip.tic2023.com>	sip:2002@voip.tic2023.com	SIP	00:00:26	17	COMPLETED	INVITE 401 401 200 200 200

(a) Detalle SIP de las llamadas VoIP para el Cliente 1

Start Time	Stop Time	Interlocutor inicial	Desde	A	Protocolo	Duración	Paquetes	Estado	Comentarios
2023-08-16 12:09:00	2023-08-16 12:09:00	192.168.2.10	<sip:Andres@192.168.2.10>	sip:2001@192.168.0.6	SIP	00:00:00	5	REJECTED	INVITE 401 401
2023-08-16 12:09:59	2023-08-16 12:09:59	192.168.2.10	<sip:Andres@192.168.2.10>	sip:2001@192.168.0.6	SIP	00:00:00	5	REJECTED	INVITE 401 401
2023-08-16 12:11:20	2023-08-16 12:12:00	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:40	21	COMPLETED	INVITE 401 401 200 200 200 200 200
2023-08-16 12:12:34	2023-08-16 12:12:54	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:20	7	CANCELLED	INVITE 200 487
2023-08-16 12:13:06	2023-08-16 12:13:33	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:27	7	COMPLETED	INVITE 200
2023-08-16 12:14:04	2023-08-16 12:14:32	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:27	19	COMPLETED	INVITE 401 401 200 200 200 200
2023-08-16 12:15:25	2023-08-16 12:15:56	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:31	7	COMPLETED	INVITE 200
2023-08-16 12:16:20	2023-08-16 12:16:49	192.168.2.10	<sip:2002@voip.tic2023.com>	sip:2001@voip.tic2023.com	SIP	00:00:28	17	COMPLETED	INVITE 401 401 200 200 200
2023-08-16 12:18:21	2023-08-16 12:18:47	192.168.0.6	<sip:2001@192.168.0.6>	<sip:2002@192.168.2.10;transport=udp>	SIP	00:00:26	7	COMPLETED	INVITE 200

(b) Detalle SIP de las llamadas VoIP para el Cliente 2

Figura 3.9: Servicio VoIP: Detalle SIP de las llamadas VoIP.

También, mediante el análisis del tráfico RTP se obtiene información del QoS del servicio de VoIP, detallada en la Figura 3.10, donde se observa que los resultados de las llamadas de VoIP son similares a los implementados previamente. Estos son:

- ❒ **Vocoder de payloads:** Vocoder g711U.
- ❒ **Pérdida de paquetes:** perdida del 0 % para todas las llamadas.
- ❒ **Retardos:** Se registran retardos mínimos mayores para el cliente en comparación con el cliente 2. Los promedios de estos valores son 18.39 ms para el cliente 1 y 7.63 ms para el cliente 2; Retardo promedio de 19.99 ms para ambos clientes y retardos máximos de 21.74 ms para el cliente 1 y 32.87 ms para el cliente 2.
- ❒ **Jitter**: En el caso del Jitter, los valores varían significativamente en el cliente 2, donde pueden llegar a 7, mientras que el cliente 1 presenta Jitters de hasta 1.46.

Source Address	Source Port	Destination Address	Destination Port	SSRC	Start Time	Duración	Payload	Paquetes	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Estado
192.168.5.10	50451	192.168.0.6	10076	0x1b6efc54	2023-08-16 12:18:25.110	19.84	g711U	993	0 (0.0%)	18.385000	19.999764	21.690000	0.002938	0.385861	0.581178	
192.168.5.10	60524	192.168.0.6	10048	0x40b37d24	2023-08-16 12:16:25.884	20.86	g711U	1044	0 (0.0%)	17.743000	19.999598	22.879000	0.000562	0.499972	0.931110	
192.168.5.10	52930	192.168.0.6	10050	0x9a9f8eac	2023-08-16 12:15:32.634	20.72	g711U	1037	0 (0.0%)	17.342000	19.999571	22.819000	0.040146	0.805904	1.467908	
192.168.5.10	56019	192.168.0.6	10010	0xffb25126	2023-08-16 12:14:09.019	20.34	g711U	1018	0 (0.0%)	18.604000	20.000493	21.655000	0.002625	0.415269	0.597000	
192.168.5.10	62923	192.168.0.6	10066	0xb3d2565e	2023-08-16 12:13:09.791	21.16	g711U	1059	0 (0.0%)	18.578000	19.998974	21.740000	0.002000	0.364309	0.554573	
192.168.5.10	59881	192.168.0.6	10048	0x8cbcc209	2023-08-16 12:12:52.297	0.26	g711U	14	0 (0.0%)	19.032000	19.940923	20.249000	0.001875	0.040590	0.115222	
192.168.5.10	50420	192.168.0.6	10080	0xbdac85a1	2023-08-16 12:11:37.273	20.46	g711U	1024	0 (0.0%)	13.803000	19.999960	26.110000	0.029500	0.891532	1.422795	

(a) Servicio VoIP: Detalle RTP de las llamadas para el Cliente 1

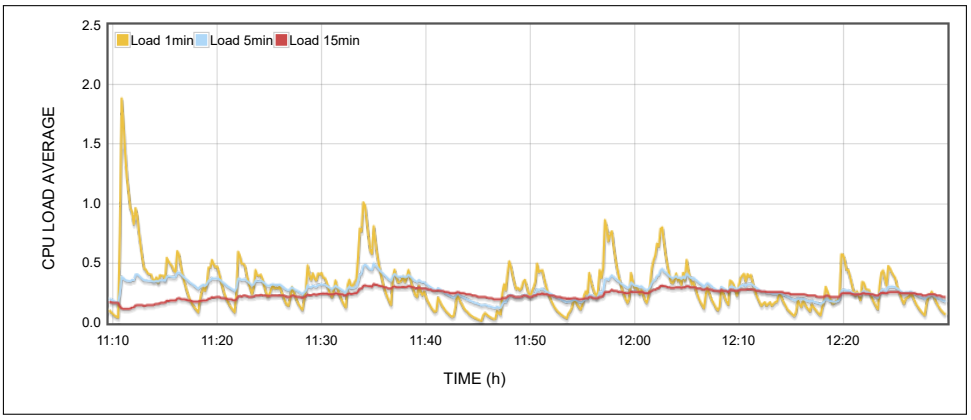
Source Address	Source Port	Destination Address	Destination Port	SSRC	Start Time	Duración	Payload	Paquetes	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Estado
192.168.2.10	60568	192.168.0.6	10012	0x505bdcb0	2023-08-16 12:18:27.916	20.00	g711U	1001	0 (0.0%)	6.581000	19.999271	33.191000	0.698750	4.465511	7.288791	
192.168.2.10	57875	192.168.0.6	10054	0x37974df9	2023-08-16 12:16:29.002	20.64	g711U	1033	0 (0.0%)	8.617000	19.997191	29.940000	0.164649	1.112223	3.607699	
192.168.2.10	62222	192.168.0.6	10004	0x4db8d0e0	2023-08-16 12:15:35.454	20.86	g711U	1044	0 (0.0%)	7.820000	20.002165	33.089000	0.088750	1.071696	2.909141	
192.168.2.10	58833	192.168.0.6	10042	0x77ee67f	2023-08-16 12:14:12.193	20.04	g711U	1003	0 (0.0%)	6.958000	20.002512	32.637000	0.026938	4.760490	7.429664	
192.168.2.10	53212	192.168.0.6	10052	0x1c2d110f	2023-08-16 12:13:12.568	21.30	g711U	1066	0 (0.0%)	16.325000	20.004633	24.366000	0.123213	0.706344	1.245788	
192.168.2.10	58674	192.168.0.6	10062	0xd04f3acd	2023-08-16 12:11:40.761	19.85	g711U	994	0 (0.0%)	7.431000	19.992930	33.104000	0.209277	1.957921	4.742771	

(b) Servicio VoIP: Detalle RTP de las llamadas para el Cliente 2

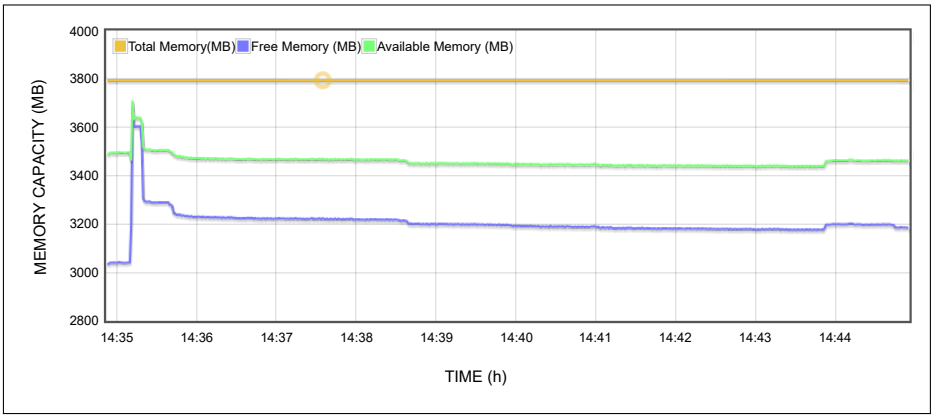
Figura 3.10: Servicio VoIP: Detalle SIP de las llamadas VoIP.

3.1.3.3 Uso de Recursos

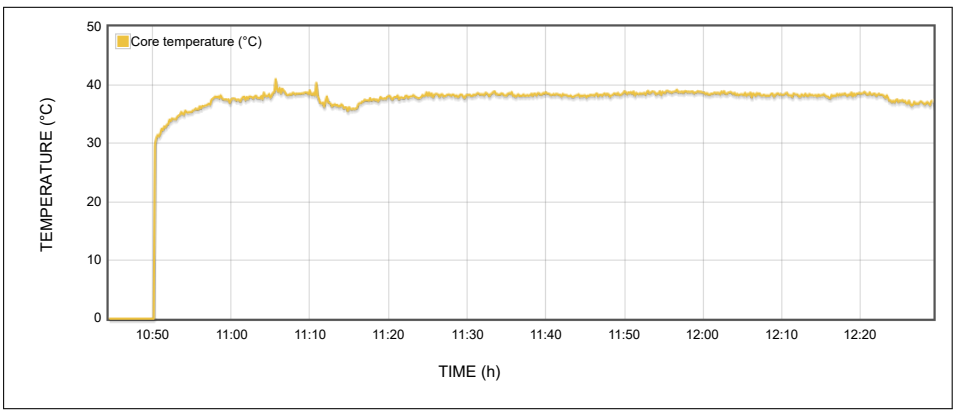
Para Docker Compose se obtiene los datos de la herramienta “Rpi-Monitor”, donde se refleja, mediante la Figura 3.11, las estadísticas de utilización de la CPU Figura 3.11a, memoria Figura 3.11b y la temperatura Figura 3.11c. Estos datos abarcan el intervalo de tiempo desde las 11:45am hasta las 12:20pm. Un punto importante a tener en cuenta es que estas estadísticas de uso se aplican a un único cliente, por lo que los valores para la implementación mediante Docker Compose son inferiores en comparación con Docker CLI .



(a) Uso de Recursos: Uso de CPU.



(b) Uso de Recursos: Uso de Memoria.



(c) Uso de Recursos: Temperatura.

Figura 3.11: Uso de Recursos Mediante RPI-MONITOR para Docker Compose

Para el despliegue mediante Docker Compose, el indicador de carga computacional “load average” comienza con un valor inicial de 1.883(47.08 %) en el arranque del host. Posteriormente, esta carga disminuye hasta 0.37(9.25 %). A partir de este punto, se inician las pruebas de los servicios de red, divididas en secciones como se detalla en la Tabla 3.4.

Tabla 3.4: Pruebas de Rendimiento: Resultados de CPU Load Average Rpi-Monitor para Docker Compose

Servicio	**Período**	**Descripción**
Routing	(11:30-11:40)	Durante la conexión entre los equipos RPI, se puede observar un valor de carga máximo de aproximadamente 1.0 (25 %). Este valor corresponde al proceso inicial de enrutamiento OSFP entre los equipos.
Conexión Extremo a extremo	(11:45-11:50)	Al comenzar las pruebas de conexión de extremo a extremo entre los clientes. Durante este período el valor de carga máximo alcanzado es de 0.518 (12.95 %).
Conexión a Servidores	(11:52-11:53)	La carga máxima alcanzada durante las pruebas de conexiones hacia los servidores contenerizados es de 0.5 (12.5 %).
DNS	(11:53-11:54)	En esta sección, la carga alcanza valores de 0.44 (11 %).
HTTP	(11:59-12:00)	La carga alcanzada durante el acceso a las páginas HTTP se obtiene un valor máximo de 0.863 (21.575 %). Además, de acuerdo con el análisis de las gráficas E/S de tráfico DNS, el tráfico aumenta considerablemente durante el período de las pruebas HTTP y se mantiene elevado después de esta prueba.
FTP	(12:00-12:04)	Durante las pruebas de las 2 conexiones FTP, se registran valores máximos de carga de 0.801 (20.025 %) para el cliente 1.
VoIP	(12:04-12:22)	Durante las pruebas VoIP, se observa que en la llamada de la extensión 2001 a la extensión 2002, la carga de CPU alcanza su punto máximo, llegando a 0.413 (0.33 %). Para el resto de las llamadas, la carga se mantiene baja en un promedio de 0.180 (4.5 %).

La Figura 3.11b muestra el uso de memoria del sistema, que, al igual que en la implementación anterior, permanece constante a medida que se ejecutan las pruebas. Donde la cantidad de memoria libre disponible comienza en 3280 MB durante el inicio del host y se mantiene en niveles cercanos a 3176.5 MB. Esto refleja un uso constante de memoria de aproximadamente 617.8 MB, en relación con un total de 3794.32 MB de memoria disponible.

En la Figura 3.11c se puede observar que la temperatura del CPU pasa de 35.95°C, en el momento del arranque, y aumenta hasta mantenerse a una temperatura promedio de 38.6°C, con un máximo de 39.2°C durante las pruebas de HTTP. Después de las pruebas de VoIP desciende a 36.7°C.

Los resultados de las pruebas conjuntas mediante Docker Compose, se encuentran en el ANEXO I en el aparatado para Docker Compose. Tanto para la placa RPI-I como RPI-II

3.2 CONCLUSIONES

- La implementación de servicios de red a través de contenedores permite un despliegue efectivo en sistemas con recursos limitados de CPU, memoria y almacenamiento, como las placas Raspberry Pi. Tal como se ha demostrado en servicios como HTTP y VoIP. En estas instancias, se obtuvieron un producto final, una página web o una llamada de voz sin una degradación notable en la calidad del servicio y con un uso óptimo de los recursos. Por ejemplo para la implementación mediante Docker CLI, para HTTP se cargaron tres páginas web sin errores, con un uso de CPU de tan solo 0.47 % y un consumo de memoria de 0.1 %. Y en cuanto al servicio de VoIP, las llamadas se llevaron a cabo sin distorsiones, retrasos ni pérdida de audio, manteniendo un uso de CPU del 5.18 % y un consumo de memoria de 0.8 %.

- La implementación de servicios de red a través de contenedores ha evidenciado un uso mínimo de recursos en la mayoría de los casos. Servicios como DHCP, HTTP y Routing muestran un consumo nulo de CPU y memoria. En contraste, servicios como DNS, FTP, VoIP presentan un alto consumo. Esto puede explicarse en parte por factores como el volumen de tráfico de solicitudes, que en el caso de DNS, fue considerablemente mayor en comparación con otros servicios. También puede ser por la transferencia de información, como es el caso de FTP. Por otro lado, su posición en la arquitectura de red debe ser considerada. Esto debido a que su ubicación puede implicar una participación implícita en otros servicios, lo que a su vez puede aumentar el consumo de recursos. Esto es evidente en el caso del servicio DNS cuando se utiliza de manera implícita para respaldar servicios web al momento de realizar resoluciones de domain names para acceder a una página web.

- La arquitectura de los servicios contenerizados también ejerce una influencia significativa en su rendimiento. Esto se debe a que el software subyacente presenta una variedad de arquitecturas para ofrecer el servicio. Algunos servicios adoptan una arquitectura simple de cliente-servidor, basada en daemons y archivos de configuración, como es el caso de DHCP, DNS y FTP. Mientras que otros, como HTTP, VoIP y Routing, presentan arquitecturas más complejas con módulos dedicados para ofrecer el servicio. Por ejemplo, el servicio HTTP, que se basa en NGINX y permite un despliegue eficiente incluso sin contenerización, destaca la importancia de la descentralización, incluso cuando se trata de una aplicación única. Esto se debe a que su estructura

se compone de módulos con una gestión interna que favorece la eficiencia [76]. En contraste, servicios como VoIP y Routing poseen arquitecturas propias que pueden ser más desafiantes en términos de gestión y rendimiento.

- La contenerización de servicios brinda un alto nivel de escalabilidad, permitiendo el despliegue de múltiples contenedores para ofrecer servicios versátiles, flexibles y eficientes. Un ejemplo de esto es el servicio DHCP, donde se despliegan varios contenedores basados en la misma imagen. Esto proporciona un servicio de direccionamiento altamente flexible con múltiples configuraciones disponibles para su despliegue. Este logro se debe en parte al uso de variables de entorno, que permiten modificar un servicio contenerizado sin tener que modificar directamente el servicio en sí. De esta manera, se crea un servicio escalable, dinámico y adaptable que puede ser ofrecido a los usuarios de manera eficiente, como se determinó con el uso de CPU y un consumo de memoria.

- Docker es una herramienta sumamente versátil que aporta beneficios significativos al despliegue de servicios de red. Su ecosistema diverso abarca múltiples complementos esenciales, como el almacenamiento a través de volúmenes y bind mounts. En este contexto, se ha observado que estos mecanismos permiten interactuar con el sistema de archivos de un contenedor, lo que facilita la modificación y configuración de un servicio sin necesidad de acceder directamente al entorno contenerizado o eliminar por completo el entorno aislado para realizar cambios. Además, se ha constatado que la reconstrucción de contenedores mediante el uso de volúmenes es un mecanismo valioso para migrar servicios. Un ejemplo de ello es el empleo de contenedores para almacenar registros de configuración, como en el caso de los servicios de enrutamiento con archivos “daemons” y “zebra.config”, o en el caso del servicio DHCP con registros de arrendamiento en “dhcpd.lease”. Estos registros permiten replicar las mismas configuraciones en otros contenedores, asegurando así la continuidad del servicio. Sin embargo, es importante señalar que los registros “logs” almacenados a través de volúmenes pueden ser propensos a la corrupción. Esto se debe a que, con el tiempo, estos archivos pueden comenzar a registrar cadenas de caracteres incomprensibles, lo que a su vez puede afectar el funcionamiento del servicio.

- Otra herramienta altamente versátil para el despliegue de servicios son los controladores de red a través de los objetos networks. Estas redes posibilitan la configuración y modificación de la gestión y comunicación de servicios contenerizados, lo que otorga

o reduce características clave, como el aislamiento de red, control de tráfico, acceso y escalabilidad. Esta flexibilidad se hace evidente al implementar servicios con redes de tipo “bridge”. En esta configuración, los servicios se mapeaban a puertos específicos para permitir el acceso externo a través del contenedor. Como se pudo observar, el uso de puertos diferentes al valor predeterminado para realizar la traducción de direcciones de red (NAT) entre el contenedor y la red externa brindaba un mayor aislamiento y seguridad del servicio. Además, permitía la incorporación de múltiples contenedores que ofrecían el mismo servicio, pero mapeados a puertos distintos para recibir el servicio. Esto es especialmente útil en el caso de servicios como HTTP, donde se logra una alta escalabilidad empleando un puerto distinto por contenedor. Sin embargo, en servicios como FTP en modo pasivo y VoIP, el mapeo de puertos implica mantener un NAT constante en un rango de puertos (FTP: para transferencia, puerto aleatorio >1048, y VoIP: transmisión de audio mediante RTP, puerto aleatorio entre 10000 y 20000). En estos casos, considerar un NAT que abarca todo el rango de puertos, para obtener la conexión del servicio por un puerto no especifico, resultaba poco práctico, ya que afecta al rendimiento, como se menciona en [74], [75]. Por esta razón, el uso de redes de tipo “host”, que eliminan el nivel de aislamiento entre los contenedores y comparten la red con el host, permitió obtener servicios con un bajo uso de recursos, pero con una capacidad de escalabilidad limitada.

- En lo que respecta al servicio contenerizado de VoIP, cuando se utiliza un medio de transmisión mediante cable Ethernet, se observa un rendimiento y calidad de servicio (QoS) considerablemente alto. Esto se confirma a través de los parámetros RTP capturados con Wireshark, donde el retardo promedio de las llamadas es de 20 ms, en comparación con el retardo máximo permitido de 150 ms, y valores máximos de Jitter de 7 ms entre ambas implementaciones, quedando por debajo del máximo permitido de 50 ms. En resumen, la contenerización de servicios de VoIP para llamadas de extremo a extremo a través de conexiones cableadas no afecta la calidad de servicio (QoS) de este servicio.

3.3 RECOMENDACIONES

- Para una implementación exitosa de contenedores, es esencial considerar inicialmente la arquitectura y el sistema operativo en los que se ejecutarán. Esto es crucial porque las tecnologías de contenerización están estrechamente ligadas al kernel del host. El cual determina la imagen del contenedor a utilizar. Si la imagen no es compatible con la arquitectura o el sistema operativo base, la implementación falla. Por lo tanto, se recomienda verificar la compatibilidad de las imágenes antes de implementarlas, ya sea consultando las descripciones en Docker Hub o utilizando herramientas de construcción multiplataforma. Estas herramientas permiten crear imágenes compatibles con diversas arquitecturas de CPU, lo que facilita la portabilidad y garantiza que las imágenes de contenedores sean ejecutables en diferentes plataformas [77].

- Es necesario considerar las características específicas que requiere un servicio antes de seleccionar el controlador de red. Dado que este tiene un impacto directo en su rendimiento. Si se prioriza el rendimiento, incluso a costa de sacrificar algo de versatilidad, se recomienda implementar redes de tipo “host”. Por otro lado, si se necesita que el servicio contenerizado disponga de múltiples características, como escalabilidad o un nivel superior de abstracción de red, a pesar de ser menos eficiente, se debe considerar el uso de redes de tipo “bridge”. Esta opción brinda más flexibilidad y versatilidad, pero llega a ser menos eficiente en comparación con las redes de tipo “host”.

- La elección de los volúmenes de Docker debe alinearse con las necesidades específicas y funcionalidades requeridas por el servicio. Esto implica, considerar si se necesita mantener la persistencia de ciertos datos contenidos en el contenedor o si es necesario interactuar con estos datos fuera del sistema de archivos del contenedor. En el primer caso, es aconsejable emplear “named volumes”. Los volúmenes, en función de sus características, permiten retener configuraciones y registros de un servicio a lo largo del tiempo, asegurando que los datos se mantengan intactos incluso si el contenedor se detiene o se recrea. Por otro lado, si se requiere interactuar directamente con el sistema de archivos de un contenedor, la elección adecuada son los “Bind Mounts”. Los cuales permiten configuraciones dinámicas de servicios contenerizados, al permitir el acceso a los archivos del contenedor desde el sistema anfitrión y viceversa.

- En relación con los errores que pueden ocurrir en los volúmenes, como la corrupción

de registros con cadenas hexadecimal, se recomienda utilizar mecanismos de gestión de servicios para depurar estos errores. Esto se debe a que, al no tratarse de un error en un contenedor en particular, el servicio continúa funcionando. Por lo tanto, es difícil determinar si un servicio se ha detenido debido a estos errores.

- El servicio DNS contenerizado demostró que la implementación en modo maestro puede llevar a que el servicio comience a recibir solicitudes de todos los clientes y, al no tener acceso a Internet, no puede resolver estas solicitudes, lo que resulta en un aumento del tráfico no resuelto y un alto uso de la CPU. Por lo tanto, se recomienda que las implementaciones de este tipo de servicio sean de tipo Recursivas o Cache, donde no se realiza el procesamiento directo de la resolución, sino que se redirige el tráfico hacia servidores DNS primarios en Internet. Estos servidores actúan como elementos de borde para una respuesta rápida a las solicitudes. Además, se debe considerar la posibilidad de utilizar un grupo de contenedores DNS para distribuir la carga de las solicitudes. Esto se puede lograr mediante el uso de orquestadores para su gestión.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] C. J. Aliaga y A. I. L. Quintero, «Arquitecturas de microservicios para aplicaciones desplegadas en contenedores,» 2018.

[2] A. Khan, «Key Characteristics of a Container Orchestration Platform to Enable a Modern Application,» 2017.

[3] RedHat, *¿Qué es el kernel de Linux?* Feb. de 2019. dirección: https://www.redhat.com/es/topics/linux/what-is-the-linux-kernel.

[4] Z. Kozhirbayev y R. O. Sinnott, «A performance comparison of container-based technologies for the Cloud,» *Future Generation Computer Systems*, vol. 68, págs. 175-182, mar. de 2017, ISSN: 0167739X. DOI: 10.1016/j.future.2016.08.025.

[5] C. D. la Torre, B. Wagner y M. Rousos, *NET-Microservices: Architecture for Containerized .NET Applications*. 2022.

[6] H. Manninen, S. P. Jan, B. Advisor y M. S. V. Jääskeläinen, «Evaluation of Container Technologies for an Embedded Linux Device.» dirección: www.aalto.fi.

[7] RedHat, *¿Qué es un hipervisor? Hypervisor*, mar. de 2023. dirección: https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor.

[8] J. E. Sarabia, «Análisis, Implementación y Despliegue de un Entorno de Computación Distribuida en Contenedores,» jul. de 2020.

[9] U. P. de Valencia, *Fundamentos de programación OOP*. dirección: http://www.upv.es/amiga/43.htm.

[10] geekland, *¿Qué es y para que sirve un sandbox en la informática?* Dirección: https://geekland.eu/que-es-y-para-que-sirve-un-sandbox/.

[11] docker.io, *Docker overview | Docker Documentation*. dirección: https://docs.docker.com/get-started/overview/.

[12] K. A. P. Chicaiza, «IMPLEMENTACIÓN DE SERVIDORES DNS Y WEB BASADOS EN CONTEDORES ALOJADOS EN LA NUBE,» ene. de 2022.

[13] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri e Y. Al-Hammadi, «Performance comparison between container-based and VM-based services,» Institute of Electrical y Electronics Engineers Inc., abr. de 2017, págs. 185-190, ISBN: 9781509036721. DOI: 10.1109/ICIN.2017.7899408.

[14] E. N. Preeth, J. P. Mulerickal, B. Paul e Y. Sastri, «Evaluation of Docker containers based on hardware utilization,» Institute of Electrical y Electronics Engineers Inc., mar. de 2016, págs. 697-700, ISBN: 9781467373494. DOI: 10.1109/ICCC.2015.7432984.

[15] C. N. Academy, *CCNA: Switching, Routing, and Wireless Essentials*, 2021. dirección: https://www.netacad.com/courses/networking/ccna-switching-routing-and-wireless-essentials.

[16] ClaudFlare, *¿Qué es un nombre de dominio? | Nombre de dominio vs. URL | Cloudflare*. dirección: https://www.cloudflare.com/es-es/learning/dns/glossary/what-is-a-domain-name/.

[17] MIT, *Berkeley Internet Name Domain (BIND)*. dirección: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-bind.html#S1-BIND-INTRODUCTION.

[18] MIT, *FTP*. dirección: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ftp.html.

[19] Sapalomera, *Principios básicos de routing y switching*. dirección: https://www.sapalomera.cat/moodlecf/RS/2/course/module10/#10.1.1.2.

[20] H. S. Hoang, A. K. Tran, T. P. Doan et al., «Design and implementation of a VoIP PBX integrated Vietnamese virtual assistant: a case study,» *https://doi.org/10.1080/24751839.2023.2183631*, págs. 1-26, mar. de 2023, ISSN: 2475-1839. DOI: 10.1080/24751839.2023.2183631. dirección: https://www.tandfonline.com/doi/abs/10.1080/24751839.2023.2183631.

[21] M. J. O. Mejía, C. A. A. Ortiz, W. E. V. Ramos y L. E. P. Moscoso, *Gestión del tráfico de red en la calidad de servicio "QoS" WAN en Tambopata-Perú 2021*, ene. de 2022. dirección: https://www.redalyc.org/journal/280/28070565020/html/.

[22] inana, *Service Name and Transport Protocol Port Number Registry*. dirección: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[23] R. P. Foundation, *Raspberry Pi Documentation - Raspberry Pi hardware*. dirección: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[24] pinout.xyz, *SPI at Raspberry Pi GPIO Pinout*. dirección: https://pinout.xyz/pinout/spi.

[25] RedHat, *Procesadores ARM*. dirección: https://www.redhat.com/es/topics/linux/what-is-arm-processor.

[26] R. P. Foundation, *Raspberry Pi 4 Model B specifications – Raspberry Pi*. dirección: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[27] RedHat, *What is an API?* Dirección: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.

[28] linuxcontainer, *Linux Containers - LXC - Introduction*. dirección: https://linuxcontainers.org/lxc/introduction/.

[29] GitHub, *GitHub - rkt/rkt: [Project ended] rkt is a pod-native container engine for Linux. It is composable, secure, and built on standards.* dirección: https://github.com/rkt/rkt.

[30] J. L. M. Tapia, «Tecnología de Contenedores Docker,» oct. de 2017.

[31] K. Kumar y M. Kurhekar, «Economically Efficient Virtualization over Cloud Using Docker Containers; Economically Efficient Virtualization over Cloud Using Docker Containers,» 2016, ISBN: 9781509045730. DOI: 10.1109/CCEM.2016.24.

[32] docker.io, *Use the AUFS storage driver | Docker Documentation*. dirección: https://docs.docker.com/storage/storagedriver/aufs-driver/.

[33] bikramat, *Docker Namespace Vs Cgroup. Namespace and Cgroup | by Bikram | Medium*. dirección: https://bikramat.medium.com/namespace-vs-cgroup-60c832c6b8c8.

[34] amazon, *¿Qué es API RESTful? | Guía sobre API RESTful para principiantes | AWS*. dirección: https://aws.amazon.com/es/what-is/restful-api/.

[35] docker.io, *Volumes | Docker Documentation*. dirección: https://docs.docker.com/storage/volumes/.

[36] docker.io, *Bind mounts | Docker Documentation*. dirección: https://docs.docker.com/storage/bind-mounts/.

[37] docker.io, *tmpfs mounts | Docker Documentation*. dirección: https://docs.docker.com/storage/tmpfs/.

[38] medium, *Docker Network Drivers Overview | Networking in Docker 3 | by Farhim Ferdous | TechMormo | Medium*. dirección: https://medium.com/techmormo/docker-network-drivers-overview-networking-in-docker-3-78308e0839c9.

[39] docker.io, *Use bridge networks | Docker Documentation*. dirección: https://docs.docker.com/network/bridge/.

[40] docker.io, *Swarm mode overview | Docker Documentation*. dirección: https://docs.docker.com/engine/swarm/.

[41] docker.io, *Use IPvlan networks | Docker Documentation*. dirección: https://docs.docker.com/network/ipvlan/.

[42] docker.io, *Use macvlan networks | Docker Documentation*. dirección: https://docs.docker.com/network/macvlan/.

[43] M. Moravcik y M. Kontsek, «Overview of Docker container orchestration tools,» Institute of Electrical y Electronics Engineers Inc., nov. de 2020, págs. 475-480, ISBN: 9780738123660. DOI: 10.1109/ICETA51985.2020.9379236.

[44] docker.io, *Use the Docker command line | Docker Documentation*. dirección: https://docs.docker.com/engine/reference/commandline/cli/.

[45] docker.io, *Dockerfile reference | Docker Documentation*. dirección: https://docs.docker.com/engine/reference/builder/.

[46] RedHat, *What is YAML?* Dirección: https://www.redhat.com/en/topics/automation/what-is-yaml.

[47] docker.io, *Compose application model | Docker Documentation*. dirección: https://docs.docker.com/compose/compose-file/02-model/.

[48] docker.io, *Services top-level element | Docker Documentation*. dirección: https://docs.docker.com/compose/compose-file/05-services/.

[49] docker.io, *docker compose | Docker Documentation*. dirección: https://docs.docker.com/engine/reference/commandline/compose/.

[50] Amazon, *Amazon.com: Samsung Galaxy A12 (32GB, 3GB) 6.5"HD+, cámara cuádruple, batería de 5000mAh, voltaje global 4G (ATT desbloqueado para T-Mobile, Verizon, Metro) A125U (negro) (renovado) : Celulares y Accesorios*. dirección: https://www.amazon.com/-/es/Samsung-cu%C3%A1druple-desbloqueado-T-Mobile-renovado/dp/B0991J62ZY.

[51] Lenovo, *Portátil Lenovo S410p | Notebook multitouch de 14", accesible, delgada y liviana | Lenovo Ecuador*. dirección: https://www.lenovo.com/ec/es/laptops/lenovo/lenovo-serie-s/S410p/p/88IP80S0296?orgRef=https%253A%252F%252Fwww.google.com%252F.

[52] Lenovo, *Laptop Lenovo ThinkPad T430, 14.1", Core i5, 4GB, 500GB, Win 7 Pro 64 Bit - 2342A22*. dirección: https://intercompras.com/p/laptop-lenovo-thinkpad-t430-core-i5-4gb-500gb-win-pro-bit-61123.

[53] linux.die, *dhcpd(8): Dynamic Host config Protocol Server - Linux man page*. dirección: https://linux.die.net/man/8/dhcpd.

[54] Blyx, *El sistema de nombres de dominio: Bind 9.2.1*. dirección: https://blyx.com/public/docs/bind.html#estructuradedirectorios.

[55] CloudFlare, *Tipos de servidores DNS | ¿Cuáles son? | Cloudflare*. dirección: https://www.cloudflare.com/es-es/learning/dns/dns-server-types/.

[56] ArchWiki, *Very Secure FTP Daemon - ArchWiki*. dirección: https://wiki.archlinux.org/title/Very_Secure_FTP_Daemon.

[57] linux.die, *vsftpd(8): Very Secure FTP Daemon - Linux man page*. dirección: https://linux.die.net/man/8/vsftpd.

[58] linux.die, *nginx(8) - Linux man page*. dirección: https://linux.die.net/man/8/nginx.

[59] S. Technologies, *Get Started Asterisk*. dirección: https://www.asterisk.org/get-started/.

[60] S. Technologies, *What Is An IP PBX? Asterisk*. dirección: https://www.asterisk.org/get-started/applications/what-is-an-ip-pbx/.

[61] S. Technologies, *Gateway Asterisk*. dirección: https://www.asterisk.org/get-started/applications/gateway/.

[62] S. Technologies, *Asterisk Applications Asterisk*. dirección: https://www.asterisk.org/get-started/applications/.

[63] L. Fundation, «FRR User Manual,» 2023.

[64] G. Wiki, *Hostapd - Gentoo Wiki*. dirección: https://wiki.gentoo.org/wiki/Hostapd.

[65] RedHat, *Chapter 12. OpenSSH Red Hat Enterprise Linux 7 | Red Hat Customer Portal*. dirección: https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-openssh.

[66] digitalocean, *What is Load Average in Linux? | DigitalOcean*, ago. de 2022. dirección: https://www.digitalocean.com/community/tutorials/load-average-in-linux.

[67] U. C. de Madrid, *UCM-Proyecto de Innovación Software libre para ciencias e ingenierías*. dirección: https://www.ucm.es/pimcd2014-free-software/wireshark.

[68] linux.die, *htop(1): interactive process viewer - Linux man page*. dirección: https://linux.die.net/man/1/htop.

[69] docker.io, *docker stats | Docker Documentation*. dirección: https://docs.docker.com/engine/reference/commandline/stats/.

[70] X. Berger, *RPi-Monitor documentation ! — RPi-Monitor v2.13-r0*, mar. de 2018. dirección: https://xavierberger.github.io/RPi-Monitor-docs/.

[71] alpinelinux, *about | Alpine Linux*. dirección: https://www.alpinelinux.org/about/.

[72] docker.io, *alpine - Official Image | Docker Hub*. dirección: https://hub.docker.com/_/alpine.

[73] J. Islam, E. Harjula, T. Kumar, P. Karhula y M. Ylianttila, «Docker Enabled Virtualized Nanoservices for LocalIoT Edge Networks,» 2019, ISBN: 9781728108643.

[74] S. Kun, Z. Yong, C. Wei y R. Jia, «An Analysis and Empirical Study of Container Networks,» 2018, ISBN: 9781538641286.

[75] L. L. Mentz, W. J. Loch y G. P. Koslovski, «Comparative experimental analysis of Docker container networking drivers,» Institute of Electrical y Electronics Engineers Inc., nov. de 2020, ISBN: 9781728194868. DOI: 10.1109/CloudNet51028.2020.9335811.

[76] A. Pattiwar, *Architecture of NGINX ——— Built for Scale and Performance-Inside NGINX*, jul. de 2021. dirección: https://www.linkedin.com/pulse/architecture-nginx-built-scale-performance-inside-akshat-pattiwar.

[77] docker.io, *Multi-platform images | Docker Documentation*. dirección: https://docs.docker.com/build/building/multi-platform/.

[78] redzone, *Docker es más seguro que LXC y que CoreOS Rkt para la virtualización de software*. dirección: https://www.redeszone.net/2016/08/27/docker-mas-seguro-lxc-coreos-rkt-la-virtualizacion-software/.

5 ANEXOS

Los anexos que forman parte del proyecto son:

I ANEXO I: GITHUB - DOCUMENTACIÓN, ARCHIVOS DE CONFIGURACIÓN Y RESULTADOS DE IMPLEMENTACIÓN DE SERVICIOS CONTENERIZADOS.

La documentación, archivos de configuración configuraciones y pruebas realizadas en la ejecución de este proyecto se encuentran cargados en el siguiente repositorio:

`https://github.com/AndresYE/Network_Service_on_Containers`

Figura 5.1: Página de GitHub - Servicios basados en contenedores.

II ANEXO II: DOCKER HUB - IMÁGENES DE CONTENEDORES DOCKER IMPLEMENTADOS.

Las imágenes de los contenedores Docker de los servicios implementados se encuentran en el siguiente repositorio:

`https://hub.docker.com/u/andresye`

Figura 5.2: Página de Docker Hub - Servicios basados en contenedores.

III ANEXO III: COMPARACIÓN ENTRE ALGUNAS DE LAS TECNOLOGÍAS DE CONTENERIZACIÓN.

Tabla 5.1: Comparación entre tecnologías basadas en contenedores, basado en [4], [29], [78].

Características	Docker	Flockport (LXC)	CoreOS Rkt
Desarrollador	Docker Inc. (dotCloud)	Daniel Lezcano (Canonical Ltd)	CoreOS
Criterio de virtualización	Virtualización de aplicaciones.	Virtualización de Sistema Operativo.	Virtualización de aplicaciones.
Sistemas Operativos	Linux, MacOS, Windows.	Linux.	Linux y MacOS o Windows mediante VMs (Vagrant).
Seguridad	Seguridad Limitada, creación de contenedores únicamente por usuarios root.	Manejo de listas de privilegios (creación de contenedores por usuarios root y no root).	Creación de contenedores sin privilegios (no root). Opción de ejecución de MV aisladas por hardware.
Arquitectura	Cliente-Servidor (Demonio Docker).	Cliente (sin Demonios) Integrado en sistemas init (systemd y upstart).	Cliente (sin Demonios) Integrado en sistemas init (systemd y upstart).
Imágenes	Imágenes OCI (Docker).	Imágenes LXC y OCI	Imágenes ACI y OCI.
Complejidad de Uso	Uso mediante una interfaz descriptiva y/o uso de líneas de comandos.	Uso mediante líneas de comandos.	Uso mediante líneas de comandos.
Casos de uso	Virtualización de aplicaciones.	Virtualización completa de un sistema operativo.	Despliegue multicontenedor. Virtualización de aplicaciones.

IV ANEXO IV: COMPARACIÓN ENTRE MECANISMOS DE ALMACENAMIENTO EN DOCKER.

Tabla 5.2: Mecanismos de almacenamiento en Docker: Anonymous, Named Volumes. Bind y Tmpfs Mounts, basado en [17][18] [19] [20].

Mecanismos de Almacenamiento	**Anonymous Volumes**	**Named Volumes**	**Bind Mounts**	**Tmpfs Mounts**
Sistema Operativo	Linux, Windows o MacOS.	Linux, Windows o MacOS.	Linux, Windows o MacOS.	Unix/Linux.
Accesibilidad de Datos	Lectura y Escritura	Lectura y Escritura	Lectura y Escritura	Lectura y Escritura
Temporalidad	Temporal	Persistente sin contenedor	Persistente sin contenedor	Temporal
Administrador	Docker	Docker	Host	Host
Almacenamiento	Directorio del Host especificado por Docker.	Directorio del Host administrado por Docker.	Directorio del Host especificado por el usuario.	Memoria del host.
Ruta de Carpeta del Host	Desconocido	Desconocido	Conocido	Desconocido
Eliminación	Automáticamente al eliminar contenedor.	Manualmente al eliminar volúmenes.	Manualmente al eliminar carpeta de host.	Automáticamente al eliminar contenedor.
Sobreescritura	No	No	Si	Si
Escritura	Procesos de Docker.	Procesos de Docker.	Procesos de Docker y procesos no Docker (Host).	Procesos de Docker.
Casos de uso	Datos temporales de un contenedor. Evitar sobreescritura de una carpeta o archivo por un Bind Mounts.	Compartir datos entre contenedores a través de un volumen. Migración de datos de un contenedor a otro. Copias de seguridad y restauración de datos.	Edición de datos fuera de contenedores. Compartir archivos desde el host a los contenedores.	Datos temporales de un contenedor como datos confidenciales.

V ANEXO V: COMPARACIÓN ENTRE REDES DOCKER.

Tabla 5.3: Comparación entre redes de Docker, basado en [28] [23]–[25], [29]–[31].

Docker Networks	None	Host	Bridge	Overlay
Requerimientos	Linux, Windows, MacOS.	Linux	Linux, Windows, MacOS.	Linux en modo Docker Swarm.
Direccionamiento	IPv4 e IPv6	IPv4 e IPv6	IPv4 e IPv6	IPv4 e IPv6
Funcionalidades de Red	-	Funciones de red del host.	NAT y direccionamiento IP a contenedores.	Enrutamiento
Interfaz	No existe conexión entre contenedores y las interfaces del host.	Conexión directa entre interfaces del contenedor y las interfaces del host.	Conexión indirecta entre las interfaces del contenedor y las interfaces del host.	Conexión entre demonios Docker de distintos hosts.
Red	Red individual de Contenedor y conexión con Loopback.	Red de host	Red intrahost de contenedores.	Red interhost de contenedores.
Aislamiento	Contenedores y contenedor- red externar.	No existe asilamiento entre la red del contenedor y el host.	Contenedores de distinta redes bridge.	No asilamiento entre demonios Docker.
Casos de Uso	Entornos de prueba Offline.	Manejo de una gran cantidad de puertos. Aislamiento total de redes Docker y no Docker.	Comunicación entre contenedores y el exterior.	Conexión entre contenedores en diferentes hosts.

VI ANEXO VI: RESULTADOS DE PRUEBAS PARA DOCKER CLI.

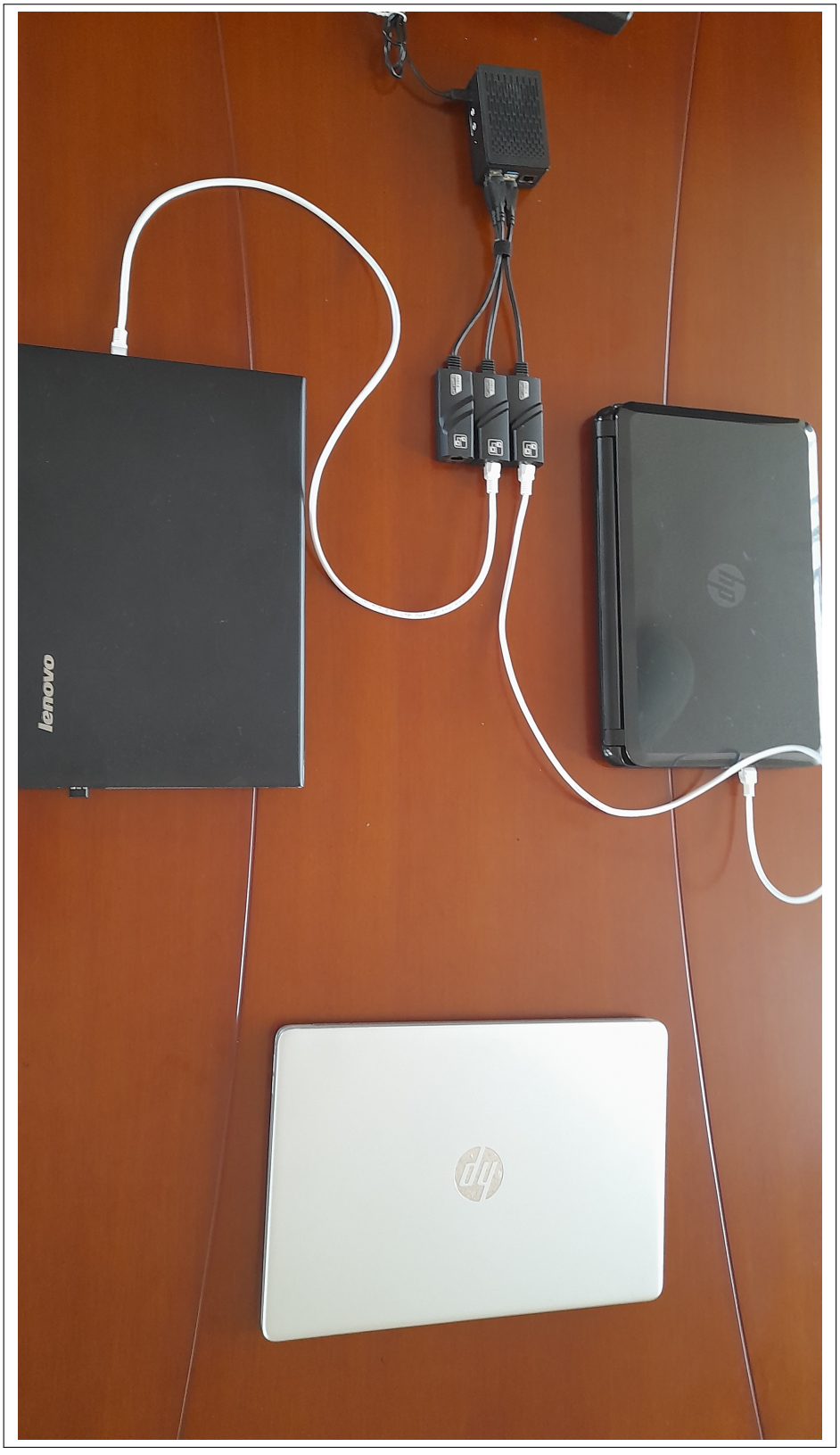


Figura 5.3: Docker CLI: Implementación Física.

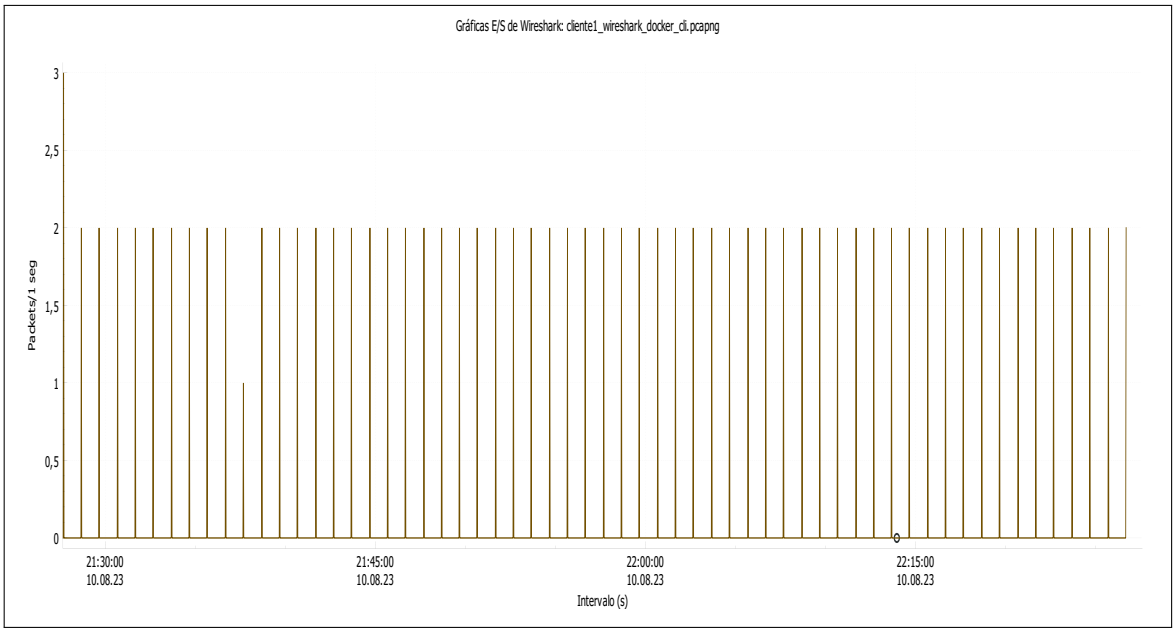
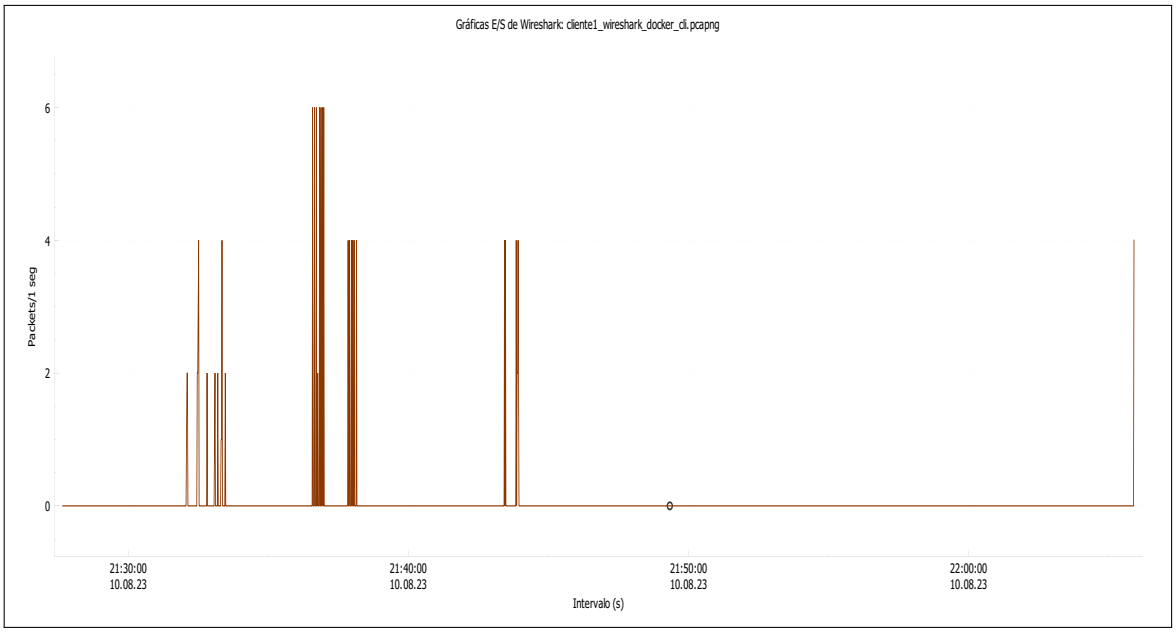


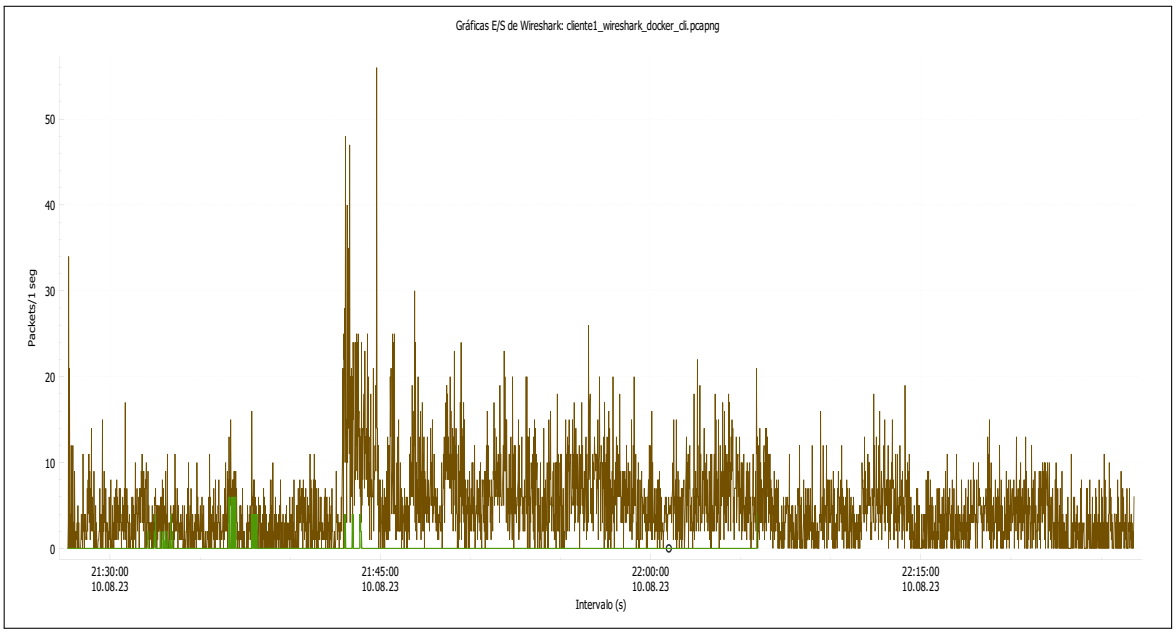
Figura 5.4: Servicio DHCP: Gráfico de E/S del tráfico DHCP.

Tabla 5.4: Pruebas de Conexión: Resultados de conexiones mediante Ping a servidores.

X	**Retardos de Ping para Cliente 1**			**Retardos de Ping para Cliente 2**			**Retardos de Ping para Cliente 3**		
Servidor	**Min.**	**Max.**	**Mean.**	**Min.**	**Max.**	**Mean.**	**Min.**	**Max.**	**Mean.**
DNS	0ms	1ms	0ms	0ms	1ms	0ms	2ms	6ms	4ms
DHCP	0ms	0ms	0ms	0ms	1ms	0ms	3ms	337ms	97ms
FTP	0ms	0ms	0ms	0ms	1ms	0ms	2ms	5ms	3ms
HTTP	0ms	1ms	0ms	0ms	1ms	0ms	5ms	502ms	129ms
web1	0ms	1ms	0ms	0ms	1ms	0ms	8ms	317ms	172ms
web2	0ms	0ms	0ms	0ms	1ms	0ms	2ms	1921ms	827ms
voip	0ms	1ms	0ms	0ms	1ms	0ms	2ms	1028ms	618ms
monitor	0ms	0ms	0ms	0ms	1ms	0ms	3ms	6ms	4ms



(a) Gráfico de E/S del tráfico DNS generado.



(b) Gráfico de E/S del tráfico DNS total.

Figura 5.5: Servicio DNS: (a) Gráfico de E/S del tráfico DNS generado y (b) Gráfico de E/S del tráfico DNS total.

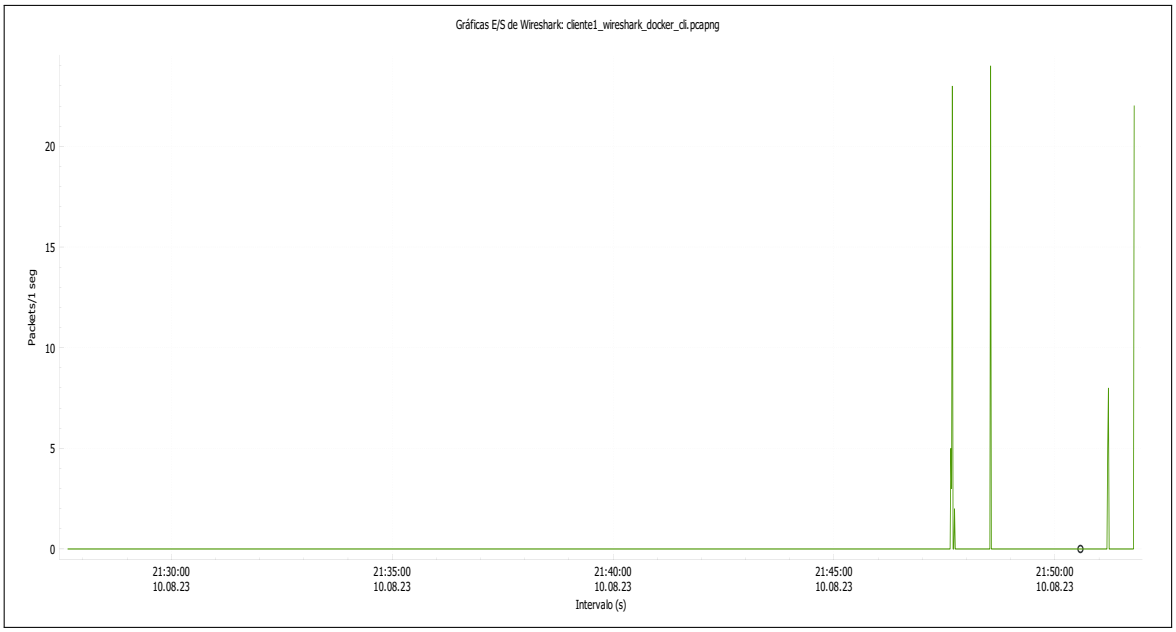


Figura 5.6: Servicio FTP: Gráfico de E/S del tráfico FTP.

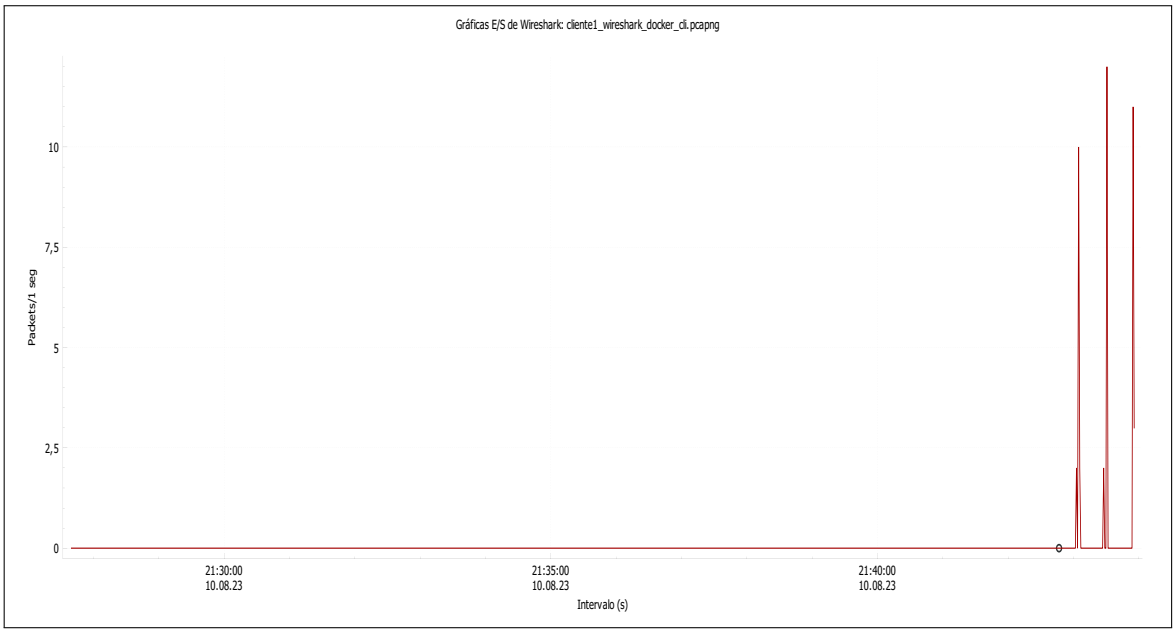


Figura 5.7: Servicio HTTP: Gráfico de E/S del tráfico HTTP.

Tabla 5.5: Servicio VoIP: Descripción de resultados tráfico SIP para el clientes voip.

Detalle	1° llamada	Descripción
Dirección IP de Origen	192.168.1.10	Dirección del cliente SIP de la red local, donde se realiza la petición de conexión al servidor VoIP.
Extensión SIP de Origen	sip: 2001@voip.tic2023.com	Extensión SIP solicitante de la conexión al servidor VoIP.
Extensión SIP de Destino	sip: 2002@voip.tic2023.com	Extensión SIP solicitada para la conexión con el servidor VoIP.
Estado	COMPLETED	Indica que la conexión se ha completado con éxito.
Comentario	INVITE 401 200	INVITE: Se envía una solicitud al servidor para establecer una sesión. 4xx: indica que una solicitud no se ha podido procesar. 2xx: indica que la solicitud se ha procesado correctamente.

Tabla 5.6: Servicio VoIP: Descripción de resultados tráfico RTP para el clientes voip.

Detalle	1° llamada	Descripción
Dirección IP de Origen	192.168.1.10	Dirección de la llamada entrante es la del Gateway desde el servidor VoIP.
Puerto de origen	65188	Puerto SIP del servidor VoIP.
Dirección IP de Destino	192.168.0.6	Dirección del cliente VoIP.
Puerto de destino	10004	Puerto aleatorio del cliente, para la recepción de las llamadas.
Duración	22.33 seg	Duración total de llamada, tanto en la conexión como en el transcurso de esta.
Tipo de Payload	G711U	Codificador de audio (Vocoder) con la que se transmiten los paquetes de voz. Este es el mismo que se configuro en los archivos de asterisk.conf.
Paquetes perdidos (%)	0 %	El porcentaje de paquetes perdidos en la llamada es bajo.
Delta mínimo (ms)	10.88	El retardo de los paquetes transmitidos es aceptable, considerando que el retardo máximo para audio debe ser de 150ms.
Delta promedio (ms)	19.99	El retardo de los paquetes transmitidos es aceptable, considerando que el retardo máximo para audio debe ser de 150ms.
Delta máximo (ms)	22.63	El retardo de los paquetes transmitidos es aceptable, considerando que el retardo máximo para audio debe ser de 150ms.
Jitter mínimo	0.29	El Jitter mínimo es casi nulo. Por lo que no afecta al servicio.
Jitter promedio	0.63	El Jitter promedio es casi nulo. Por lo que no afecta al servicio.
Jitter máximo	1.04	El Jitter máximo es bajo. Por lo que no afecta al servicio.

Tabla 5.7: Uso de Recursos: Uso de CPU mediante Docker stats y Htop.

Servidor	Docker Stats - Uso de CPU máximo (%)	HTOP - Uso de CPU máximo (%)	HTOP - Uso de Memoria (%)
DHCP Wlan0	0.11	0.0	0.2
DHCP Eth1	0.11	0.0	0.2
DHCP Eth2	0.11	0.0	0.2
DNS	15.02	7.3	1.1
FTP	11.89	8.5	0.0
HTTP (www, web 1, web2)	0.47	0.7	0.1
VoIP	5.18	5.3	0.8
Routing	0.12	0.10	0.1

VII ANEXO VII: RESULTADOS DE PRUEBAS PARA DOCKER COMPOSE.

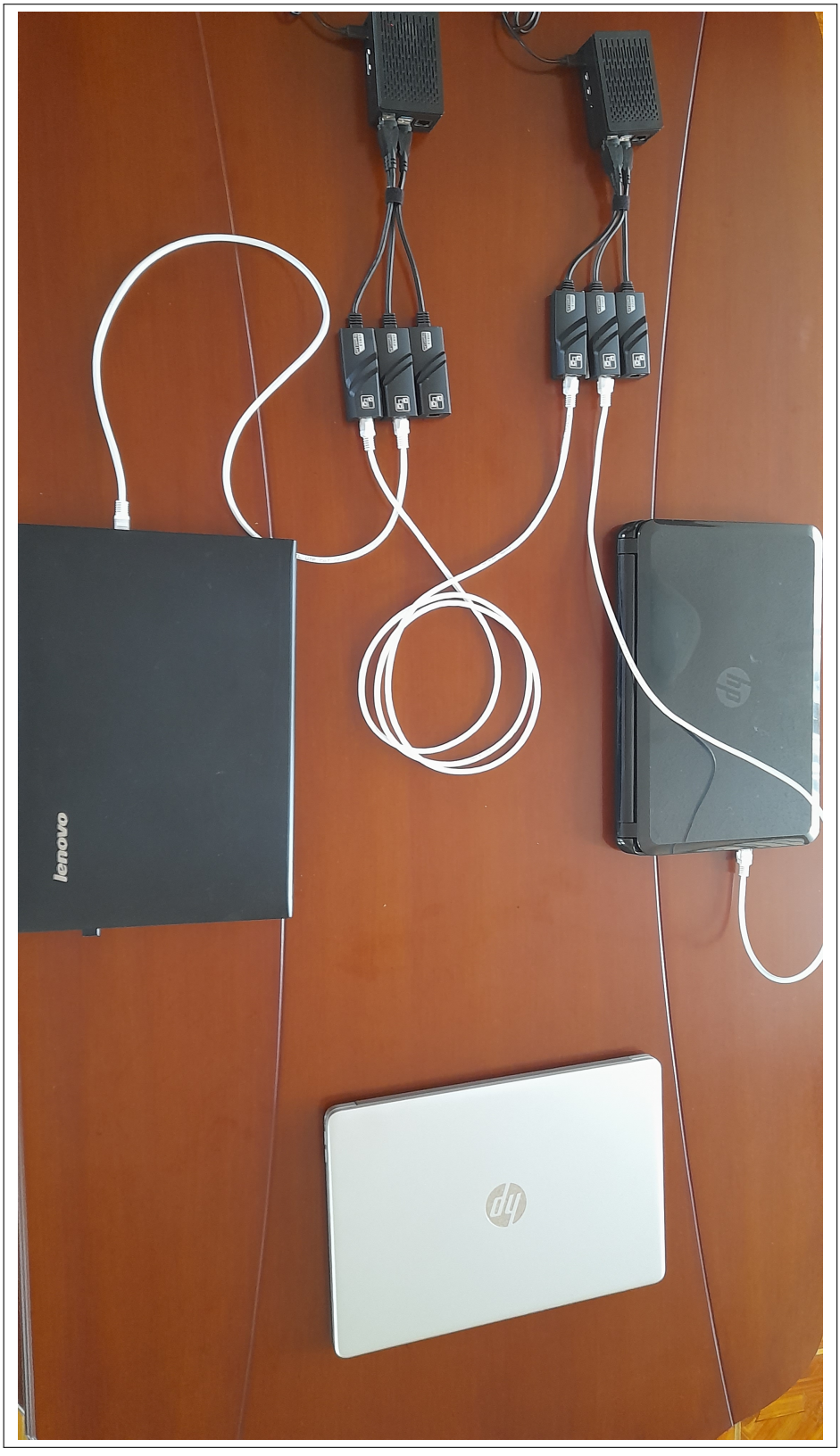


Figura 5.8: Docker Compose: Implementación Física.

```
/ # vtysh
% Can't open configuration file /etc/frr/vtysh.conf due to 'No such file or directory'.

Hello, this is FRRouting (version 8.4_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

pi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O   10.0.1.0/30 [110/1000] is directly connected, eth3, weight 1, 00:03:23
C>* 10.0.1.0/30 is directly connected, eth3, 00:03:37
O>* 192.168.0.0/24 [110/1010] via 10.0.1.1, eth3, weight 1, 00:02:37
O>* 192.168.2.0/24 [110/2000] via 10.0.1.1, eth3, weight 1, 00:02:37
O   192.168.5.0/24 [110/1000] is directly connected, eth2, weight 1, 00:00:49
C>* 192.168.5.0/24 is directly connected, eth2, 00:00:49
pi# show ip route ospf
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O   10.0.1.0/30 [110/1000] is directly connected, eth3, weight 1, 00:03:26
O>* 192.168.0.0/24 [110/1010] via 10.0.1.1, eth3, weight 1, 00:02:40
O>* 192.168.2.0/24 [110/2000] via 10.0.1.1, eth3, weight 1, 00:02:40
O   192.168.5.0/24 [110/1000] is directly connected, eth2, weight 1, 00:00:52
pi#
```

(a) Tablas de enrutamiento del host RPI 1.

```
/ # vtysh
% Can't open configuration file /etc/frr/vtysh.conf due to 'No such file or directory'.

Hello, this is FRRouting (version 8.4_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

pi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O   10.0.1.0/30 [110/10] is directly connected, eth3, weight 1, 00:06:10
C>* 10.0.1.0/30 is directly connected, eth3, 00:06:10
O   192.168.0.0/24 [110/10] is directly connected, wlan0, weight 1, 00:56:23
C>* 192.168.0.0/24 is directly connected, wlan0, 00:56:23
O   192.168.2.0/24 [110/1000] is directly connected, eth2, weight 1, 00:07:02
C>* 192.168.2.0/24 is directly connected, eth2, 00:07:02
O>* 192.168.5.0/24 [110/1010] via 10.0.1.2, eth3, weight 1, 00:03:02
pi# show ip route ospf
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O   10.0.1.0/30 [110/10] is directly connected, eth3, weight 1, 00:06:11
O   192.168.0.0/24 [110/10] is directly connected, wlan0, weight 1, 00:56:24
O   192.168.2.0/24 [110/1000] is directly connected, eth2, weight 1, 00:07:03
O>* 192.168.5.0/24 [110/1010] via 10.0.1.2, eth3, weight 1, 00:03:03
pi#
```

(b) Tablas de enrutamiento del host RPI 2.

Figura 5.9: Servicio de Routing: Tablas de enrutamiento.

Tabla 5.8: Verificación de Conexión: Ping del Cliente 1 al Cliente 2.

Dirección IPv4 de verificación	Retardo Mínimo	Retardo Máximo	Retardo Promedio
Gateway Cliente 1: 192.168.5.1	0ms	0ms	0ms
Gateway Host RPI2: 10.0.1.2	0ms	1ms	0ms
Gateway Host RPI1: 10.0.1.1	1ms	1ms	1ms
Dirección IPv4 de Ciente 2: 192.168.2.10	2ms	4ms	2ms

Tabla 5.9: Verificación de Conexión: Ping del Cliente 2 al Cliente 1.

Dirección IPv4 de verificación	Retardo Mínimo	Retardo Máximo	Retardo Promedio
Gateway Cliente 2: 192.168.2.1	0ms	1ms	0ms
Gateway Host RPI2: 10.0.1.1	0ms	1ms	0ms
Gateway Host RPI1: 10.0.1.2	1ms	1ms	1ms
Dirección IPv4 de Ciente 1: 192.168.5.10	2ms	2ms	2ms

```
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\TOSHIBA>tracert 192.168.2.10

Traza a la dirección DESKTOP-LUBQVII [192.168.2.10]
sobre un máximo de 30 saltos:

  1    <1 ms    <1 ms    <1 ms  192.168.5.1
  2     1 ms     1 ms     1 ms  10.0.1.1
  3     1 ms     1 ms     1 ms  DESKTOP-LUBQVII [192.168.2.10]

Traza completa.

C:\Users\TOSHIBA>
```

(a) Servicio de Routing: Verificación de Ruta de Conexión desde el Cliente 1 hacia el Cliente 2.

```
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Andres>tracert 192.168.5.10

Traza a la dirección DESKTOP-EE99I73 [192.168.5.10]
sobre un máximo de 30 saltos:

  1     *        *        *     Tiempo de espera agotado para esta solicitud.
  2     1 ms    <1 ms     1 ms  10.0.1.2
  3     2 ms     2 ms     2 ms  DESKTOP-EE99I73 [192.168.5.10]

Traza completa.

C:\Users\Andres>
```

(b) Servicio de Routing: Verificación de Ruta de Conexión desde el Cliente 2 hacia el Cliente 1.

Figura 5.10: Verificación de Ruta de Conexión entre clientes.

Tabla 5.10: Verificación de Conexión: Conexión 1 a Servidores Contenerizados.

Servidor	Retardo Mínimo	Retardo Máximo	Retardo Promedio
DNS	1ms	1ms	1ms
DHCP	1ms	1ms	1ms
FTP	1ms	1ms	1ms
HTTP	1ms	1ms	1ms
web1	1ms	1ms	1ms
web2	1ms	1ms	1ms
voip	1ms	1ms	1ms
monitor	1ms	1ms	1ms