Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Herramienta para el despliegue de laboratorios virtuales mediante Docker

Autor: Álvaro de Castro García
Tutor: Vicente Jesús Mayor Gallego

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Herramienta para el despliegue de laboratorios virtuales mediante Docker

Autor:
Álvaro de Castro García

Tutor:
Vicente Jesús Mayor Gallego
Profesor Ayudante Doctor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Herramienta para el despliegue de laboratorios virtuales mediante Docker

Autor: Álvaro de Castro García

Tutor: Vicente Jesús Mayor Gallego

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Se acerca el final de una de las etapas más enriquecedoras de mi vida, en la que he descubierto mi pasión por el mundo de las telecomunicaciones. Nada de esto habría sido posible sin la ayuda y el apoyo de toda la gente que ha estado a mi lado a lo largo de estos años y, cuyo interés por mi formación ha culminado en el presente documento.

Debo agradecer a mi familia por haberme ofrecido la oportunidad de formarme en las materias que me apasionan y que, a pesar de todas las noches de insomnio y semanas sin levantar la cabeza del pupitre, me han ayudado a ver el mundo desde otros ojos y a estar más cerca de dedicar mi futuro profesional a los ámbitos que realmente me interesan.

En segundo lugar, me gustaría agradecer a mi tutor, Vicente, por haberme apoyado tanto en la realización de este trabajo, en el que no sólo se ven volcadas mis horas de trabajo y de estudio de la materia, sino también las incontables tutorías y correos electrónicos que hemos intercambiado a lo largo de todos estos meses, sin los cuales nada de esto habría sido posible.

Por último, me gustaría agradecer a todos mis compañeros de titulación y a los demás profesores que me han ayudado a obtener los conocimientos y habilidades necesarios para llegar hasta este punto.

A todos ellos, y a toda la demás gente que me han ayudado a lo largo de estos años, gracias.

Álvaro de Castro García

Sevilla, 2023

Resumen

En el ecosistema digital actual, las redes de computadores son cada vez más complejas y su funcionamiento es cada vez más difícil de comprender sin ver a las mismas en ejecución. Para solucionar este problema, existen múltiples de simuladores, emuladores y otras herramientas que permiten comprobar el funcionamiento de una red sin necesidad de montarla físicamente.

El presente documento tiene como objetivo presentar una alternativa a las herramientas actuales de despliegue de laboratorios virtuales, dockerlab, la cual reúne funcionalidades de algunas de las más populares y consta de una sintaxis reducida para su funcionamiento, ofreciéndole así al usuario la posibilidad de trabajar con redes de computadores virtuales con mucha más facilidad que mediante medios convencionales.

En el presente documento, además, se puede encontrar un análisis de la situación actual con respecto a las tecnologías con finalidades similares, que servirá para verificar la utilidad que presenta el desarrollo de dockerlab frente al uso de herramientas convencionales.

Abstract

In the current digital ecosystem, computer networks are becoming increasingly complex and challenging to study solely from an external perspective without testing their functionality. To address this issue, there are multiple simulators, emulators, and other tools that allow you to verify the operation of a network without the need for physical setup.

This document aims to present an alternative to current virtual lab deployment tools, called "dockerlab," which combines functionalities from some of the most popular tools and features a simplified syntax for operation. This provides users with the ability to work with virtual computer networks much more easily than through conventional means.

Furthermore, this document also includes an analysis of the current situation regarding technologies with similar purposes. This analysis will help assess the utility of developing dockerlab compared to using conventional tools.

Índice

Agradecimientos **IX**

Resumen **XI**

Abstract **XIII**

Índice **XV**

Índice de Tablas **XVII**

Índice de Figuras **XIX**

Índice de Códigos **XXI**

1 Introducción **1**
1.1 Objetivos *1*
1.2 Limitaciones *2*

2 Tecnologías y Trabajos Relacionados **3**
2.1 Máquinas virtuales *3*
2.1.1 ¿Qué es una máquina virtual? 3
2.1.2 Tecnologías Existentes 4
2.1.3 Orquestación y Despliegue de Máquinas Virtuales 4
2.1.4 Ventajas e Inconvenientes 5
2.2 Contenedores *5*
2.2.1 Herramientas de despliegue y orquestación 6
2.2.2 Herramientas adicionales 8
2.2.3 Ventajas e inconvenientes 10
2.3 Simuladores de red *11*
2.3.1 ns: Simuladores de red en tiempo discreto 11
2.3.2 GNS3: Entorno virtual de simulación de redes en tiempo real 12
2.3.3 Ventajas e Inconvenientes 13
2.4 Conclusiones *14*

3 Diseño e Implementación **15**
3.1 Tecnologías empleadas *15*
3.1.1 Docker: Despliegue de aplicaciones en contenedores 15
3.1.2 Docker Compose: Definición de aplicaciones multicontenedor 16
3.1.3 Tshark: Analizador de protocolos de red 17
3.2 Diseño y Arquitectura *17*
3.2.1 Visión general del sistema 18
3.2.2 Elementos de entrada 19
3.2.3 Instalación de Dependencias 21
3.2.4 Comportamiento de la herramienta 22
3.3 Ejemplo de ejecución/uso *25*

4 Casos de Uso **29**
4.1 Red MQTT *29*
4.2 Generación de dataset *33*

5 Conclusiones **39**

6 Referencias **41**

7 Anexo: Código del script Dockerlab **43**

ÍNDICE DE TABLAS

Tabla 1: Comparación entre VirtualBox y VMware Workstation 4
Tabla 2: Comparación entre las distintas herramientas de despliegue de Docker 8
Tabla 3: Diferencias entre Network simulator y GNS3 13

ÍNDICE DE FIGURAS

Ilustración 1: Arquitectura de una máquina virtual 3

Ilustración 2: Funcionamiento de docker swarm. Se aprecia como un mismo servicio tiene varias réplicas de una tarea, las cuales son acarreadas por uno o varios contenedores. 7

Ilustración 3: Funcionamiento de kubernetes. Se aprecian los nodos que forman un clúster de kubernetes, así como los servicios que estos ofrecen y los pods que componen cada servicio. 8

Ilustración 4: Ejemplo de uso de Compose Generator 10

Ilustración 5: Proyecto de ejemplo en GNS3 13

Ilustración 6: Funcionamiento básico de dockerlab 15

Ilustración 7: Diferencias entre contenedores y Máquinas virtuales. 16

Ilustración 8: Funcionamiento resumido de dockerlab 18

Ilustración 9: Clasificación de los archivos de la aplicación 19

Ilustración 10: Extracto de config.yml 20

Ilustración 11: Archivo docker-compose.yml generado tras la ejecución de dockerlab 21

Ilustración 12: Subprocesos generados al construir el archivo docker-compose.yml 23

Ilustración 13: Procesos generados durante la ejecución del software 24

Ilustración 14: Procesos generados para la gestión de entrada de usuario 24

Ilustración 15: Resultado de ejecutar la bandera “-h” o “--help" 25

Ilustración 16: Ejemplo de pasos previos a la ejecución del software 25

Ilustración 17: Ejemplo de ejecución de docker-compose mediante dockerlab 26

Ilustración 18: Ejemplo de salida al detener la ejecución de los contenedores 26

Ilustración 19: Indicación de que se ha detenido la ejecución de los contenedores correctamente 26

Ilustración 20: Ventana de "utilización" antes de recibir ningún dato 27

Ilustración 21: Ventana de "utilización" una vez hay datos que mostrar 27

Ilustración 22: Fragmento de output.pcap 27

Ilustración 23: Escenario 1. Red Industrial MQTT 29

Ilustración 24: Árbol de ficheros empleados en el caso 1 30

Ilustración 25: Comparación entre el archivo config.yml y docker-compose.yml generado en el escenario 1 31

Ilustración 26: Captura del tráfico generado en el escenario 1 32

Ilustración 27: Utilización de recursos de cada uno de los contenedores del escenario 1 32

Ilustración 28: Salida estándar de la ejecución del escenario 1 33

Ilustración 29: Escenario 2. Generación de dataset 33

Ilustración 30: Árbol de ficheros empleado en el escenario 2 34

Ilustración 31: Utilización de recursos de cada uno de los contenedores del escenario 2 35

Ilustración 32: Utilización de recursos del IDS del escenario 2 35

Ilustración 33: Captura del tráfico de red durante el ataque de ICMP flooding. 36

Ilustración 34: Log generado por el IDS tras analizar el archivo de captura de tráfico de la red 36

Ilustración 35: Comparación entre el archivo de configuración y el docker-compose.yml generados en el escenario 2. 37

ÍNDICE DE CÓDIGOS

Código 1: entrada de composerize para comprobar su funcionamiento 9
Código 2: Archivo docker-compose.yml generado por composerize 9
Código 3: Comandos a ejecutar para actualizar la lista de repositorios y los paquetes instalados 22
Código 4: Comandos para instalar el software del que depende dockerlab 22
Código 5: Comando para instalar las dependencias de Python 22
Código 6: Dockerfile del broker MQTT del caso de uso 1 30
Código 7: Contenido de mosquitto.conf del caso de uso 1 30
Código 8: Script a ejecutar por los clientes MQTT del caso de uso 1 30
Código 9: Dockerfile de la imagen de Kali Linux personalizada empleado en el caso de uso 2 34
Código 10: Script ejecutado por el contenedor "Hacker" del caso de uso 2 34
Código 11: Script ejecutado por el contenedor "Víctima" del caso de uso 2 34
Código 12: Script ejecutado por el contenedor "IDS" en el caso de uso 2 35

1 INTRODUCCIÓN

La posibilidad de conocer el comportamiento de una red antes de su despliegue es una de las bases de la ingeniería de redes. A lo largo de los años, han surgido múltiples herramientas que permiten el despliegue o simulación de redes de computadores en entornos controlados, sin embargo, el funcionamiento de las herramientas más potentes es a veces tedioso, y su configuración puede tomar mucho más tiempo del que realmente sería necesario cuando se pretende recrear escenarios sencillos.

Este trabajo propone el diseño e implementación de nueva herramienta, llamada **dockerlab**, que facilita el despliegue de laboratorios virtuales. Por ejemplo, la herramienta podría ser utilizada para la reproducción de escenarios en redes de área local; facilitando las pruebas de nuevo software (e.g., IDS), la generación de datasets (logs o capturas de tráfico), y la evaluación del rendimiento.

A continuación, se describen los principales objetivos y limitaciones de la herramienta descrita en este documento.

1.1 Objetivos

A pesar de que existen herramientas que podrían llegar a realizar las funciones de dockerlab, su uso implicaría un gran conocimiento por parte del usuario, ya que, al ser multipropósito, éstas ofrecen muchas opciones no relevantes en los casos de uso anteriormente mencionados o, simplemente, su configuración es compleja, monótona o incluso repetitiva.

Dockerlab es una herramienta por línea de comandos que simplifica el despliegue de laboratorios virtuales, apoyándose en algunas de las tecnologías ya existentes. Gracias a sus diversas opciones de funcionamiento, puede ser empleado para realizar cualquiera de las tareas expuestas anteriormente individualmente o cualquier combinación de ellas, lo cual es logrado haciendo uso de una serie de banderas y opciones que se verán con mayor detenimiento en capítulos posteriores.

Los principales objetivos que se han buscado satisfacer en el diseño de dockerlab son los siguientes:

- **Sintaxis textual sencilla**: Un software como dockerlab requiere de una simplificación de la sintaxis con respecto a las herramientas que utiliza para proporcionar el resultado esperado para ser más amigable al usuario.
- **Abstracción:** Ligado al objetivo anterior, se desea que el usuario no requiera de especiales conocimientos sobre las tecnologías subyacentes.
- **Reproducibilidad**: Se busca que dockerlab genere un archivo que permita ejecutar el laboratorio virtual en cualquier lugar, sin necesidad de que sea en la misma máquina que lo ha generado.
- **Uso interactivo**: El usuario debe poder interactuar con el software para decidir cuándo ejecutar el laboratorio virtual, cuándo pararlo y cuándo dar por finalizada la sesión.
- **Monitorización de recursos**: Dockerlab debe proporcionar al usuario la opción de obtener información acerca del consumo de recursos de cada uno de los **nodos** que está ejecutando en todo momento.
- **Captura automática del tráfico de red**: Se debe ofrecer al usuario también la posibilidad de monitorizar el tráfico de red generado por los contenedores y almacenar el mismo en un archivo para su posterior estudio.
- **Bajo consumo de recursos**: Para ofrecer una ventaja con respecto al resto de herramientas en el mercado, se debe ofrecer un menor consumo de recursos con respecto a otras herramientas de simulación, emulación o virtualización.

1.2 Limitaciones

En lo que respecta a las limitaciones de dockerlab derivadas de las decisiones tomadas para su diseño, caben destacar:

- **Funcionamiento sólo sobre ethernet**: Por la naturaleza de las herramientas de las que dockerlab hace uso, no se pueden diseñar sistemas que utilicen tarjetas de red que no sean ethernet, ya que las máquinas harán uso de puentes ethernet virtuales para comunicarse entre sí.
- **No permite ejecución multinodo**: Debido a que se ha decidido no utilizar la tecnología "Docker swarm" (se verá en detalle en apartados posteriores), la complejidad del escenario quedará limitada por los recursos hardware de la máquina en la que se ejecute el software. En un futuro, dockerlab podría ser ampliado para incorporar las funcionalidades de "docker swarm".
- **Monitorización de recursos limitada**: Dockerlab pretende realizar una monitorización de recursos básicas en la que se muestren algunos de los parámetros más relevantes de cada uno de los contenedores, sin embargo, no consta de herramientas de análisis avanzadas.
- **Diseñada para Linux**: Debido al funcionamiento del motor Docker, algunas funcionalidades, como la captura del tráfico desde el equipo anfitrión, podría no ser compatible con otros sistemas operativos (e.g., MacOS).

2 TECNOLOGÍAS Y TRABAJOS RELACIONADOS

Este capítulo tiene como objetivo la revisión de herramientas disponibles en el mercado – preferentemente herramientas *open source* o gratuitas – que ofrezcan funcionalidades similares o relacionadas con la simulación, despliegue y monitorización de laboratorios virtuales, con el fin de tomar las mejores decisiones en el diseño de la misma.

Es importante destacar que, aunque estas constituyan algunas de las herramientas más populares, existe una amplia gama de software diseñado con el mismo propósito y que, por tanto, el propósito de este capítulo no es más que ilustrar brevemente las alternativas disponibles a la herramienta expuesta en el presente documento, y no se trata de una revisión exhaustiva.

Para ello, se propone la revisión de tecnologías en tres secciones: la primera presenta el uso de máquinas virtuales como alternativa, la segunda introduce el uso de contenedores, y la tercera trata el uso de simuladores de red. Finalmente, se ofrece una conclusión del análisis realizado.

2.1 Máquinas virtuales

El despliegue de laboratorios virtuales puede verse como la ejecución una red de equipos completa en el interior de una máquina anfitriona. Para lograr esto, la primera tecnología que puede venir a la mente son las **máquinas virtuales**, software que es capaz de cargar en su interior otro sistema operativo, sin embargo, esta no es una solución óptima para la aplicación deseada. En el presente apartado se estudiará en detenimiento qué son estas "máquinas virtuales", sus utilidades y los motivos por el que se ha decantado por utilizar una solución distinta.

2.1.1 ¿Qué es una máquina virtual?

Tras la breve introducción expuesta anteriormente, se puede entender a una máquina virtual como un software que proporciona la misma funcionalidad que ordenadores físicos, esto es, ejecutan aplicaciones y un sistema operativo. Sin embargo, a pesar de ejecutarse sobre un ordenador físico, el hecho de que se comporten como otro hace que cada máquina virtual que se ejecute en un mismo dispositivo se comporte como un sistema informático totalmente independiente.

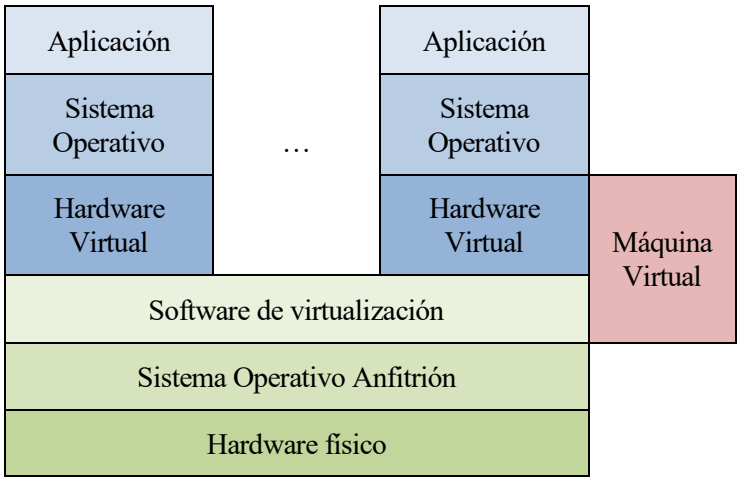

Ilustración 1: Arquitectura de una máquina virtual

Es importante destacar que existen dos tipos de máquinas virtuales según su funcionalidad: máquinas virtuales **de sistema** y máquinas virtuales **de proceso**, aunque, normalmente, cuando se hace referencia al concepto de "máquina virtual", se está refiriendo a las de sistema.

Una máquina virtual de sistema se encarga de simular un equipo completo, es decir, se trata de un software que puede hacerse pasar por otro equipo diferenciado, de modo que ejecuta otro sistema operativo en su interior, reservando recursos de la máquina anfitriona (tales como memoria RAM o espacio de almacenamiento en el disco duro) para su propio uso, lo cual supone un gasto de recursos muy superior al estrictamente necesario si se desea destinar la máquina virtual para una tarea específica.

Una máquina virtual de proceso tiene un funcionamiento distinto. A diferencia de las de sistema, las máquinas virtuales de proceso no emulan un PC por completo, si no que ejecutan un proceso concreto – como una aplicación – en su entorno de ejecución, lo cual es especialmente útil ya que permite desarrollar aplicaciones para varias plataformas fácilmente, debido a que libera al programador de la necesidad de desarrollar software específicamente para cada sistema. Este es el tipo de tecnología de virtualización empleada por cualquier aplicación Java o basada en el framework .NET. Este tipo de máquinas virtuales por tanto no resultan de utilidad para el propósito que se persigue en este documento, y por tanto quedan descartadas.

2.1.2 Tecnologías Existentes

Existe un gran repertorio de software disponible para la virtualización de sistemas operativos. A continuación, se estudiarán algunas de las opciones más populares en el mercado.

2.1.2.1 VirtualBox

Se trata de una de las herramientas más empleadas por el público general para la creación y ejecución de máquinas virtuales. Es un software gratuito multiplataforma que puede ser obtenido a través de su página oficial.

VirtualBox ofrece multitud de funciones y parámetros personalizables, tales como la posibilidad de realizar determinadas tareas como compartir archivos, unidades o periféricos entre las diversas máquinas virtuales y el equipo anfitrión. Su popularidad ofrece una ventaja adicional: existe una gran cantidad de documentación disponible en internet que guía al usuario para llevar a cabo cualquier tipo de configuración que desee.

2.1.2.2 VMware Workstation

Es uno de los softwares más veteranos disponibles para el uso de máquinas virtuales. VMware Workstation lleva siendo considerado por casi veinte años el software de referencia a la hora de la creación de máquinas virtuales, aunque abarca también multitud de necesidades adicionales en lo que respecta a la virtualización.

Algunas de las funciones adicionales que presenta son la posibilidad de configurar y administrar redes virtuales o virtualizar varios sistemas operativos al mismo tiempo, lo cual la convierten en una herramienta compleja para la virtualización tanto de sistemas como de redes de equipos.

VirtualBox	VMware Workstation
× Ofrece multitud de funciones y parámetros, pero menos avanzadas que su competencia. ✓ Se trata de un software más amigable para nuevos usuarios, con multitud de ayuda en línea.	✓ Es más completo, consta de un mayor repertorio de herramientas adicionales para usuarios avanzados. × Es más complejo, por lo que menos amigable para usuarios nuevos.

Tabla 1: Comparación entre VirtualBox y VMware Workstation

2.1.3 Orquestación y Despliegue de Máquinas Virtuales

En lo que respecta al despliegue y orquestación de un conjunto de máquinas virtuales, tal y como se ha visto en apartados anteriores, no se trata de una tarea trivial debido a la complejidad tanto del software de dichas máquinas virtuales como la configuración necesaria para la creación de un sistema que intercomunique a las mismas.

Existen, sin embargo, algunas herramientas que facilitan la creación de este tipo de sistemas, tales como **Hyper-V**, **VMware** o, como alternativa de software no propietario, **Proxmox VE**, la cual es una solución de virtualización de software libre que posee una sencilla y completa interfaz que permite realizar cualquier tarea de administración de sistemas virtuales habitual.

La virtualización de máquinas, a pesar de esto, sigue siendo un proceso muy costoso en cuanto a recursos y tiempo, y, a pesar de las herramientas expuestas en este apartado, está principalmente enfocada al despliegue de software en fase de producción, por lo que consideramos que no resultan una alternativa válida para la consecución de los objetivos descritos en la introducción.

2.1.4 Ventajas e Inconvenientes

El uso de máquinas virtuales para recrear laboratorios virtuales ofrece al usuario una serie de ventajas que son especialmente para algunos proyectos determinados. A continuación, se exponen algunas de estas:

- **Versatilidad de Sistema Operativo**: La gran mayoría de software de virtualización disponible en el mercado permite ejecutar casi cualquier sistema operativo, permitiéndole al usuario disponer, en el mismo equipo o servidor, de máquinas con el sistema operativo que requiera para una tarea determinada.
- **Garantizan la independencia de cada entorno**: Gracias a que las máquinas virtuales encapsulan todo lo necesario para su funcionamiento, se puede usar su entorno independiente para realizar todo tipo de pruebas sin necesidad de preocuparse de la estabilidad del equipo anfitrión.
- **Seguridad**: Debido a la independencia de entornos en las máquinas virtuales, aunque una de estas se vea comprometida por un software malicioso, esto no supondrá un riesgo directo para la máquina anfitriona.
- **Replicabilidad**: Las máquinas virtuales pueden ser clonadas, y reemplazadas con facilidad en caso de fallo en la máquina anfitriona.

Esto no significa que el uso de esta tecnología no tenga inconvenientes, en concreto, caben destacar los siguientes:

- **Disminución del rendimiento**: Debido a que una máquina virtual ejecuta un sistema operativo dentro de otro, el rendimiento de la máquina virtual siempre será inferior al del propio equipo físico que la ejecuta, por lo que hay casos en los que es mejor emplear hardware físico exclusivo que garantice mayor rendimiento.
- **Complejidad de uso**: debido a que la estructura interna de una máquina virtual es compleja (especialmente en sistemas que constan de interconexiones con distintas redes y con gran variedad de hardware), la configuración de una máquina virtual en ciertos entornos no es una tarea trivial, requiriendo de conocimientos avanzados y de una gran cantidad de tiempo y esfuerzo para su puesta en marcha.
- **Uso de recursos**: Debido a su elevado coste computacional, hay escenarios en los que es una mejor opción emplear contenedores, los cuales consumen muchos menos recursos.

2.2 Contenedores

En el contexto de la virtualización de sistemas, un contenedor es una unidad de software que encapsula una aplicación junto con todas sus dependencias y configuraciones necesarias para hacer que esta funcione de manera independiente en un entorno aislado. Se tratan de una forma de virtualización a nivel de sistema operativo que permite a las aplicaciones ejecutarse de manera consistente en diferentes entornos sin tener que preocuparse por la arquitectura del hardware subyacente.

Los contenedores emplean características del sistema operativo, tales como los *namespaces* (espacios de nombres) y los *cgroups* (grupos de control) para aislar aplicaciones entre sí y del sistema anfitrión, lo que se traduce en que cada contenedor tiene su propio entorno de ejecución independiente y garantiza que las aplicaciones no interfieran unas con otras.

A la hora de tratar con redes de contenedores, se hará uso de *virtual bridges* (puentes virtuales), los cuales son un componente clave de estas, especialmente cuando se trata de comunicar contenedores entre sí o con equipos exteriores a la red. Dichos puentes virtuales se emplean para conectar los contenedores a una red virtual y permitir así la comunicación entre ellos y con otros dispositivos de la red. Dado que permiten aislar y conectar a los contenedores en redes virtuales, son un aspecto clave para la creación de aplicaciones distribuidas y la gestión de la conectividad de los contenedores.

2.2.1 Herramientas de despliegue y orquestación

En el siguiente capítulo, se tratará en detenimiento qué es y cómo funciona Docker, pero por el momento, basta con comprender que el motor de Docker permite ejecutar contenedores (similares a máquinas virtuales ligeras) en una o varias máquinas reales, así como gestionar las interacciones que estos tienen entre sí.

Del mismo modo, en dicho capítulo también se estudiará el método de despliegue que se ha decidido emplear para el presente proyecto: Docker Compose.

Es por eso por lo que, en este apartado se verán algunas de las alternativas más comunes a Docker Compose, sus propiedades, funcionalidades y situaciones en las que emplearlas.

2.2.1.1 Docker Swarm

Una Docker swarm (o, por su traducción al español, un enjambre Docker) es una herramienta de orquestación que permite a múltiples contenedores Docker unirse a un clúster o grupo. Las actividades del clúster serán controladas por un manejador (o "manager") del enjambre, y las máquinas que se hayan unido al clúster serán referenciadas como "nodos".

Haciendo un símil entre Docker swarm y una orquesta musical, "en un sistema de orquestación de contenedores, los músicos son las tareas, y el director es un servicio líder. Las tareas […] ejecutan algún trabajo computacional, por ejemplo, corren un servidor web. El director, swarm, es responsable de su aprovisionamiento, su disponibilidad, su conexión, su escalado." [1]

Llamaremos clúster de Docker swarm a un conjunto de nodos, que son máquinas físicas o virtuales que ejecutan el software de Docker. Del mismo modo, en Docker swarm, las aplicaciones se definen mediante servicios, donde cada servicio representará a una aplicación distribuida cuya ejecución tiene lugar en múltiples contenedores y tendrá una cantidad definida de réplicas, siendo estas las instancias del mismo contenedor en diferentes nodos con el objetivo de poder realizar distribución de carga.

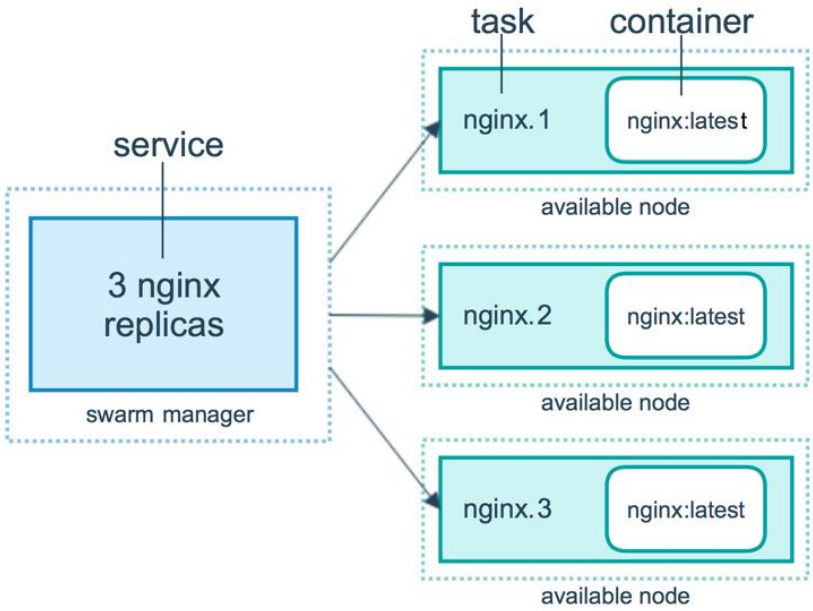

Ilustración 2: Funcionamiento de docker swarm. Se aprecia como un mismo servicio tiene varias réplicas de una tarea, las cuales son acarreadas por uno o varios contenedores.

Es gracia a la existencia de estas réplicas que Docker swarm se caracteriza por su escalabilidad horizontal (permite aumentar o disminuir la cantidad de réplicas de un servicio en función de la demanda), su posibilidad de realizar balanceo de carga, su sistema de descubrimiento de servicios (permite a los contenedores encontrar y comunicarse entre sí de manera transparente), su resiliencia y recuperación ante fallos y la facilidad que ofrece para realizar la implementación de los servicios y actualizaciones mediante comandos específicos.

Sin embargo, el uso directo de docker swarm no se adapta a las necesidades descritas en este documento, ya que está pensada para desplegar microservicios en producción, aunque podría ser tenida en cuenta en un futuro para proporcionar capacidades multinodo a dockerlab.

2.2.1.2 Kubernetes

En palabras de Marko Lukša, Ingeniero de Software en Red Hat, “Kubernetes abstrae la infraestructura de hardware y expone a su centro de datos al completo como un único recurso computacional enorme. Permite desplegar y correr sus componentes software sin necesidad de conocer los verdaderos servidores que hay debajo. Cuando se despliega una aplicación multicomponente a través de Kubernetes, se selecciona un servidor para componente, se despliega y se habilita para que éste encuentre y se comunique fácilmente con todos los demás componentes de su aplicación.” [2]

Al igual que en Docker swarm, tendremos elementos tales como los clústeres o nodos maestros y de trabajo, aunque, a diferencia de Docker swarm, Kubernetes opera en términos de “objetos”, que son descripciones de cómo debe ejecutarse una aplicación. Dichos objetos contienen “Pods” – las unidades más pequeñas y que contienen uno o varios contenedores –, “Servicios” que permiten la comunicación entre contenedores, “Sets de réplicas” que garantiza un número deseado de réplicas de un pod, y “Despliegues” para manejar actualizaciones y versiones de las aplicaciones.

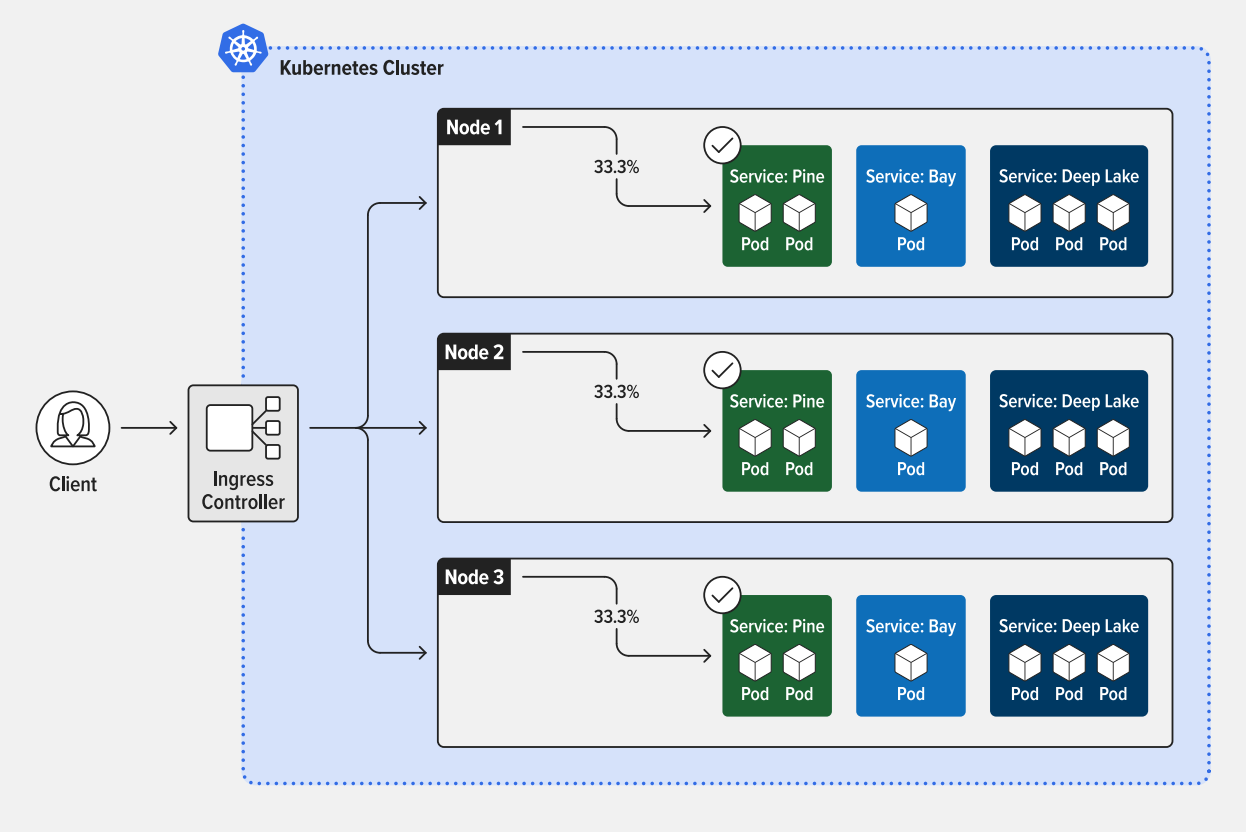

Ilustración 3: Funcionamiento de kubernetes. Se aprecian los nodos que forman un clúster de kubernetes, así como los servicios que estos ofrecen y los pods que componen cada servicio.

	¿Qué es?	¿Cuándo usarlo?
Docker	Tecnología para desplegar aplicaciones en contenedores individuales.	Si se desea desplegar un único contenedor accesible en la red.
Docker Compose	Permite configurar y ejecutar múltiples contenedores Docker en una misma máquina anfitriona.	Si se desean desplegar múltiples contenedores en un único cliente con un único archivo de configuración.
Docker swarm	Herramienta de orquestación que permite ejecutar e interconectar contenedores en múltiples anfitriones.	Si se desea desplegar un clúster de nodos Docker (múltiples anfitriones para una aplicación escalable).
Kubernetes	Similar a Docker swarm, pero con un mayor atractivo dada su facilidad de automatización y habilidad de soportar una mayor demanda.	Para manejar un gran despliegue de contenedores escalables automatizados.

Tabla 2: Comparación entre las distintas herramientas de despliegue de Docker

2.2.2 Herramientas adicionales

Las herramientas expuestas anteriormente son alternativas a las que finalmente se han decidido emplear para el desarrollo del proyecto. Sin embargo, en el presente apartado se estudiarán algunos proyectos relacionados que aportan funcionalidades similares a dockerlab para así mostrar los aspectos en los que innova y las novedades que ofrece con respecto a la competencia.

2.2.2.1 Composerize: Parseador de órdenes a Docker Compose

El primer proyecto relacionado que se va a mencionar es **composerize** (https://www.composerize.com/). Se trata de una sencilla aplicación web que analiza las opciones proporcionadas por línea de comando a Docker y genera un archivo docker-compose en formato YAML para así facilitar el proceso de ejecución en veces posteriores.

Composerize logra, por tanto, hacer de una orden para el intérprete de Docker un archivo de directivas para ejecutar utilizando Docker Compose.

A continuación, se muestra un ejemplo del funcionamiento de composerize.

```
docker run -p 80:80 -v /var/run/docker.sock:/tmp/docker.sock:ro --restart \
   always --log-opt max-size=1g nginx
```

Código 1: entrada de composerize para comprobar su funcionamiento

Si introducimos el anterior comando anterior en la web de composerize, obtendremos la siguiente salida:

```
version: '3.3'
services:
    nginx:
        ports:
            - '80:80'
        volumes:
            - '/var/run/docker.sock:/tmp/docker.sock:ro'
        restart: always
        logging:
            options:
                max-size: 1g
        image: nginx
```

Código 2: Archivo docker-compose.yml generado por composerize

Con pegar la salida anterior en un archivo docker-compose.yml, se podría ejecutar Docker Compose y obtener un resultado similar al de la orden de Docker que se mostró al principio.

Sin embargo, composerize no ofrece ninguna función adicional, limitándose únicamente a transcribir comando de docker a ficheros docker-compose, por lo que no resulta una herramienta que cumpla las funcionalidades que se esperan de un software como dockerlab.

2.2.2.2 Compose Generator: Generador de configuración para docker-compose

Compose Generator (https://www.compose-generator.com/) es una herramienta de línea de comandos que facilita el despliegue de proyectos en Docker lo más rápido posible. Su interfaz de línea de comandos (CLI) actúa como un ayudante para manejar las tareas más comunes en Docker, tales como gestionar las configuraciones de Docker Compose con una sencilla interfaz.

El principal foco de Compose Generator es aumentar la productividad y eficiencia de casos de uso comunes, tales como configurar Docker y Docker Compose y desplegar un *stack* software. Hace uso de plantillas de servicios predefinidos que pueden ser utilizadas por el usuario para obtener exactamente la funcionalidad deseada. Gracias a que Compose Generator es extensible, se pueden añadir nuevas plantillas de servicios predefinidos.

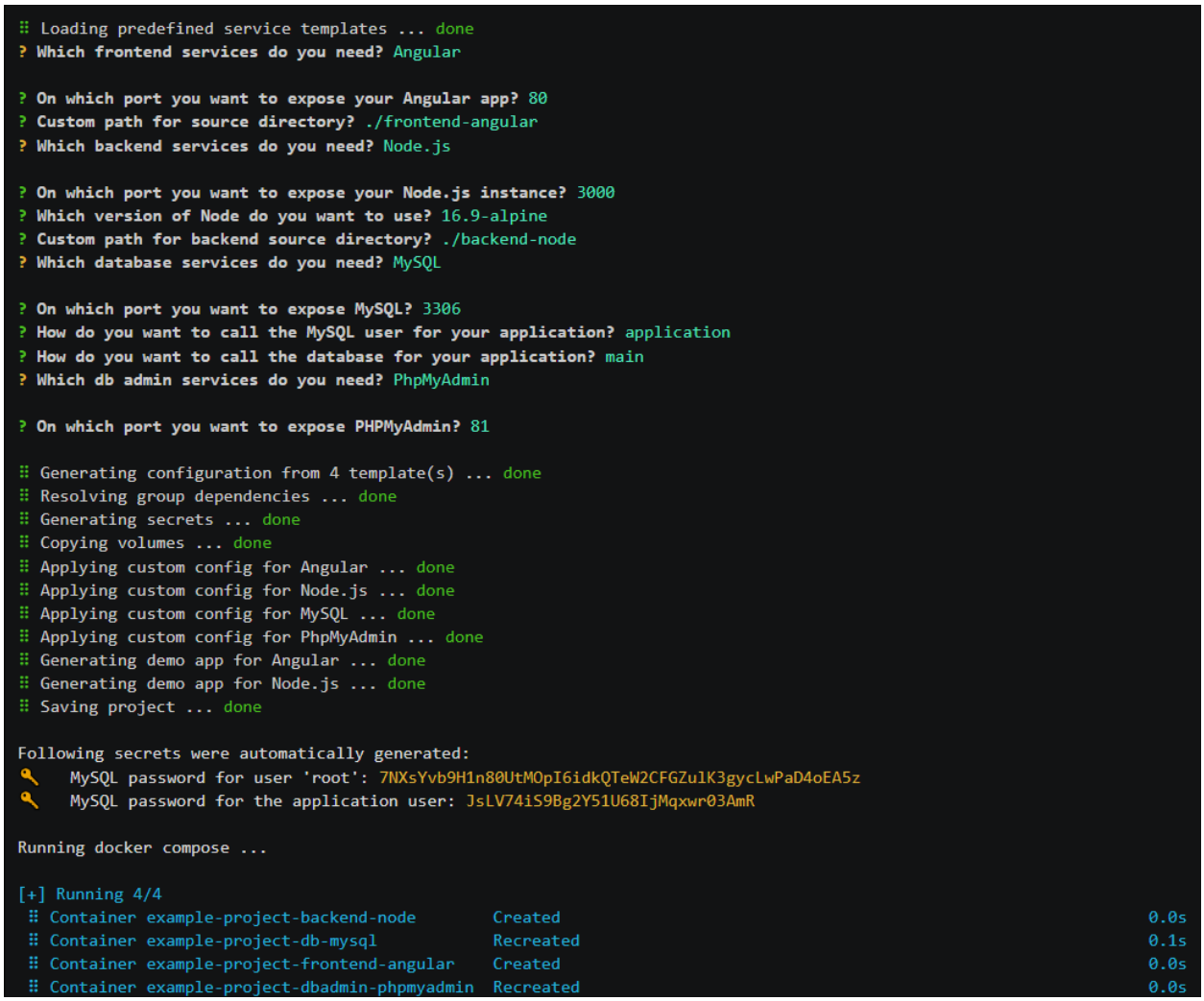

Ilustración 4: Ejemplo de uso de Compose Generator

En la imagen anterior se muestra un ejemplo de uso de Compose Generator, en el que se pone en funcionamiento una configuración de Docker Compose para un *stack* con Angular, Node.js, MySQL y PhpMyAdmin en unos pocos seguros.

Se trata de un proyecto parecido a dockerlab, sin embargo, no facilita la configuración avanzada de cada uno de los contenedores ni proporciona un servicio de monitorización en detalle del sistema.

2.2.3 Ventajas e inconvenientes

Al utilizar contenedores en un proyecto, exponemos al mismo a una serie de ventajas y desafíos que deben ser considerados según las necesidades y objetivos específicos del mismo. En lo que respecta a las ventajas, a continuación, se muestran algunas de las principales:

- **Aislamiento**: Al igual que con las máquinas virtuales, los contenedores proporcionan un alto nivel de aislamiento entre las aplicaciones y sus dependencias. Gracias a que cada contenedor es independiente y encapsula su entorno de ejecución, se evitan conflictos y problemas de compatibilidad.
- **Portabilidad**: Mediante contenedores, una aplicación contenida se ejecutará de la misma manera en diferentes entornos, lo cual facilita la migración y el despliegue multiplataforma.
- **Eficiencia de recursos**: Una de las principales ventajas que ofrecen con respecto al uso de máquinas virtuales. Los contenedores comparten el kernel del sistema operativo anfitrión, lo que los hace más ligeros y eficientes en términos de recursos, permitiendo así que se ejecuten múltiples contenedores en un solo servidor físico.

- **Rápido despliegue**: A diferencia de las máquinas virtuales, los contenedores se pueden crear y desplegar rápidamente, acelerando así el proceso de desarrollo y entrega de aplicaciones.
- **Orquestación**: Existen herramientas como Docker Compose, Docker Swarm o Kubernetes (se estudiarán en apartados posteriores) que simplifican la administración, escalabilidad y automatización de contenedores en entornos complejos.

Sin embargo, los contenedores no son herramientas perfectas y constan también de algunos inconvenientes a la hora de su utilización, algunos de los cuales serán expuestos a continuación:

- **Complejidad Inicial**: Los contenedores son una herramienta muy poderosa pero también compleja, por lo que su adopción requiere de un aprendizaje inicial y de una correcta configuración de redes, volúmenes y mecanismos de orquestación.
- **Seguridad**: A diferencia de con las máquinas virtuales, si no se configuran y gestionan adecuadamente, los contenedores pueden introducir riesgos de seguridad, especialmente si se utilizan imágenes no confiables.
- **Gestión de redes**: La configuración de redes y puentes virtuales puede resultar compleja, y la conexión entre los contenedores y el mundo exterior puede requerir conocimientos adicionales.
- **Compatibilidad de Sistemas operativos**: Algunas aplicaciones pueden requerir de sistemas operativos específicos que no sean compatibles con contenedores, limitando su uso en ciertos casos.
- **Tamaño de imágenes**: En caso de no gestionarse adecuadamente, las imágenes de contenedores pueden volverse grandes, lo cual puede afectar al tiempo de descarga y al espacio en disco.

2.3 Simuladores de red

Llamamos simulador de red al software que permite probar el funcionamiento de un sistema de comunicación entre distintos dispositivos sin necesidad de montar dicho sistema en la vida real. “La tecnología de redes de computadores está sujeta a cambios constantes e innovaciones. […] Debido a su complejidad y necesidad de compatibilidad con versiones anteriores, surgen muchos desafíos al desarrollar, implementar, probar y comprender estas tecnologías. Aquí es donde entra en juego la simulación de redes.” [3]

Debido a que el software que se expone en el presente documento tiene como finalidad ser una herramienta para el despliegue de redes virtuales, es importante comprender las funcionalidades ofrecidas por otras herramientas para el estudio de redes.

2.3.1 ns: Simuladores de red en tiempo discreto

Network Simulator 2 y 3 (ns-2 y ns-3) son simuladores de redes basados en eventos discretos empleados principalmente en entornos educativos y de investigación, siendo especialmente utilizados en el entorno de la investigación de redes móviles ad-hoc.

“Network Simulator […] se ha convertido en un estándar de facto debido a su amplia utilización. Su estructura permite obtener una visión global de las redes que facilita la relación de conceptos de distintas áreas como podría ser la propagación de señales en medios inalámbricos con el desarrollo de nuevos mecanismos de comunicación. A pesar de estas ventajas, presenta como principal inconveniente la dificultad asociada al desarrollo de nuevos protocolos para este entorno.” [4]

Ns comenzó como una variante del simulador de redes “REAL” en 1989 y su evolución a lo largo de las últimas décadas lo ha convertido en una de las herramientas de simulación más empleadas en entornos académicos y de investigación. Sus últimas versiones (ns-2 y ns-3) son empleadas por universidades e investigadores independientes de todo el mundo por su versatilidad y la cantidad de opciones de configuración que ofrecen.

2.3.1.1 ns-2

"Network Simulator Versión 2 […] es, simplemente, una herramienta de simulación basada en eventos que ha demostrado ser útil para estudiar la naturaleza dinámica de las redes de comunicación. La simulación de funciones y protocolos de redes cableadas e inalámbricas (por ejemplo, algoritmos de enrutamiento, TCP, UDP) se puede realizar utilizando NS2. En general, NS2 proporciona a los usuarios una forma de especificar dichos protocolos de red y simular sus comportamientos correspondientes." [5]

ns-2 ha sido desarrollado en C++ y proporciona una interfaz de simulación a través de una versión Orientada a Objetos de Tcl (Tool Command Language, lenguaje de herramientas de comando para el desarrollo rápido de prototipos) llamada OTcl.

Para utilizarlo, el usuario deberá definir una topología de red por medio de scripts OTcl para que, posteriormente, ns-2 realice las simulaciones pertinentes sobre dicha topología teniendo en cuenta los parámetros definidos.

La última versión del software fue publicada en 2011 y ha sido reemplazado mayoritariamente por su sucesor, ns-3.

2.3.1.2 ns-3

"Durante muchos años, […] ns–2 fue el estándar de facto para la investigación académica en protocolos de redes y métodos de comunicación. Innumerables artículos de investigación fueron escritos informando resultados obtenidos utilizando ns–2, y cientos de nuevos modelos fueron escritos y contribuidos a la base de código de ns–2. A pesar de esta popularidad, […], los autores y otros investigadores emprendieron un proyecto en 2005 para diseñar un nuevo simulador de redes que reemplazara a ns–2 en la investigación en redes." [6]

ns-3 fue desarrollado desde cero, utilizando el lenguaje C++ y basándose en el paquete yans (Yet Another Network Simulator). Haciendo uso de ns-3 se pueden obtener modelos de simulación de alto desempeño, permitiendo así ser utilizado como emulador.

ns-3 soporta simulaciones de multitud de tipos de redes, tanto IP, LTE, Wi-fi o WiMAX, así como varios protocolos de enrutamiento. Es gratuito, de código abierto, consta de una licencia GNU GPLv2 y es mantenido por una amplia comunidad.

Su última versión (ns-3.39) publicada el 5 de julio de 2023 proporciona actualizaciones del modelo Wi-Fi para los estándares de Wi-Fi 6 y 7, soporte de escaneo de huérfanos IEE 802.15.4 LR-WPAN y extensiones de modelos de pérdidas de propagación para construir pérdidas de penetración, así como otras mejoras a sus características y arreglos de errores.

2.3.2 GNS3: Entorno virtual de simulación de redes en tiempo real

En palabras de Jeremy Grossmann, cofundador de GNS3, "Para entender, diseñar y manejar las redes complejas actuales, los profesionales de las redes deben no solo dominar la teoría, si no también la práctica y validar conceptos en estos entornos constantemente cambiantes. Es aquí donde entra en juego GNS3: provee al usuario una inmensa flexibilidad para construir sus propios laboratorios de redes, permitiéndoles experimentar con nuevos elementos de red, capturar paquetes para diseccionar protocolos y verificar configuraciones para su posterior despliegue en equipos reales." [7]

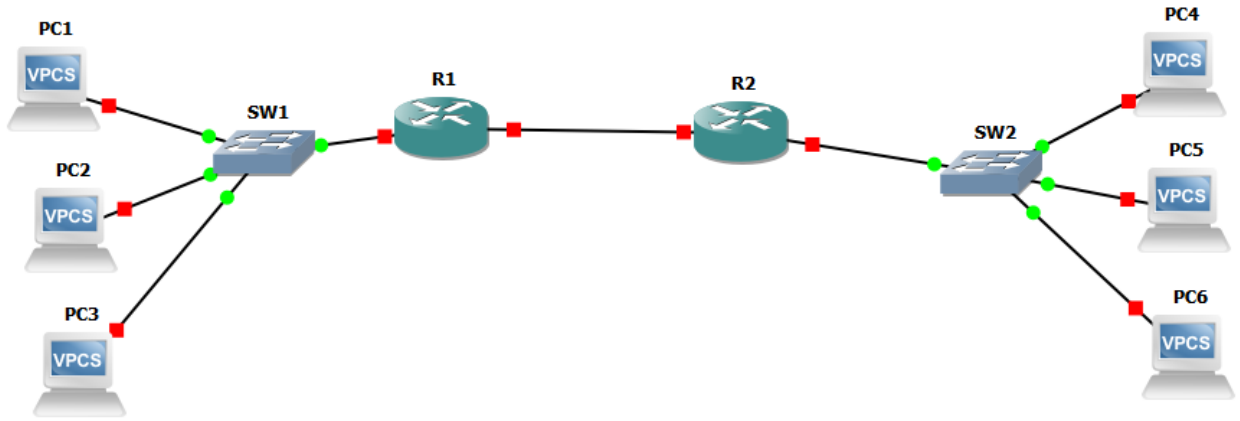

Ilustración 5: Proyecto de ejemplo en GNS3

GNS3 es un simulador gráfico de red que permite al usuario diseñar y configurar redes complejas y ejecutar simulaciones con las mismas, permitiendo interconectar dispositivos reales y virtuales. Dicha hazaña es posible gracias a las aplicaciones asociadas a este software, tales como Dynamips (emulador de *IOS*), Dynagen (*front-end* de Dynamips), Qemu, VirtualBox (permiten emplear máquinas virtuales como un *firewall*), VPCS (emulador de PC con funcionalidades básicas de *networking*) o IOU (compilaciones del IOS de Cisco específicas para correr en sistemas UNIX).

GNS3 es empleado por miles de ingenieros de redes alrededor del mundo para emular, configurar, hacer tests y corregir errores de redes físicas y virtuales. GNS3 permite al usuario ejecutar una pequeña topología compuesta por unos pocos equipos en su ordenador portátil o una topología compuesta por un gran número de equipos en múltiples servidores o, incluso, en la nube. Se trata de una herramienta gratuita y de código abierto disponible para cualquier usuario que desee instalarla.

En la siguiente tabla podemos apreciar las principales diferencias entre el software de simulación de redes mencionado hasta este punto para comprender los casos de uso que abarca cada uno:

	ns-2, ns-3	***GNS3***
Tipo de simulador	Discreto, sin entorno gráfico	Continuo, con entorno gráfico
Objetivo	Monitorización y diseño de redes	Diseño y configuración de redes
Funcionalidad	Basado en scripts	Ejecuta el firmware real
Lenguaje	C++ y Python	Python

Tabla 3: Diferencias entre Network simulator y GNS3

2.3.3 Ventajas e Inconvenientes

Al igual que con respecto a las demás tecnologías, los simuladores de red tienen una serie de ventajas que los convierten en herramientas esenciales para la investigación, desarrollo y capacitación en el campo de las redes de computadores. Algunas de estas son:

- **Experimentación segura**: Gracias que los simuladores de red no ejecutan realmente el software sobre equipos reales, se pueden probar configuraciones y escenarios de red sin necesidad de afectar a una red real, lo cual ayuda a identificar errores graves o problemas de seguridad.
- **Coste reducido**: Debido a que no se requieren de dispositivos reales, la experimentación con simuladores es mucho más económica a la hora de crear y mantener una infraestructura de red que las pruebas en equipos reales.

- **Control total**: Ya que se trata de simulaciones, los parámetros de las pruebas realizadas pueden ser totalmente controlados, facilitando la creación de escenarios específicos.
- **Reproducibilidad**: Los experimentos en simuladores son altamente reproducibles, facilitando así el contraste de resultados y la realización de pruebas rigurosas.

El uso de simuladores presenta, por el contrario, una serie de inconvenientes derivados de su naturaleza y que también hay que tener en cuenta antes de decantarse por su uso. A continuación, se muestran algunos de los principales:

- **Simplificaciones de la realidad**: A pesar de que los simuladores pretenden replicar entornos reales, estas simulaciones no son perfectas y no pueden tener en cuenta todos los parámetros existentes en la vida real, lo cual puede desembocar en resultados que no se correspondan totalmente con lo que pueda encontrarse en la práctica.
- **Recursos necesarios**: Existen múltiples simuladores, tales como ns-2 o ns-3, que en algunos casos pueden resultar intensivos en recursos para las máquinas que ejecuten las simulaciones, por lo que, en algunos casos, se puede requerir de hardware lo suficientemente potente para poder realizar simulaciones complejas de forma eficiente.
- **Curva de aprendizaje**: No es trivial manejar simuladores como los que se han expuesto en el presente apartado, ya que su funcionamiento interno es complejo y se requiere de grandes cantidades de tiempo y esfuerzo para dominar su manejo.
- **Limitaciones de modelos matemáticos**: Los simuladores utilizan modelos matemáticos y del comportamiento de los diversos protocolos que pueden no estar completamente adaptados a la realidad, por lo que las simulaciones pueden no resultar completamente precisas en algunos casos.

2.4 Conclusiones

Tras haber estudiado las diversas tecnologías disponibles en el mercado, su funcionamiento y los casos de uso para las cuales están diseñadas se ha optado por emplear tecnología de contenedores, en concreto, se ha decidido utilizar Docker y docker-compose como herramientas subyacentes para mayor compatibilidad, y se ha decidido hacer un *parser* orientado al despliegue de escenarios según los requisitos expuestos en el capítulo 1, el cual consta de una sintaxis de configuración sencilla y que ofrece funcionalidades adicionales, tales como la gestión de validación y el *parseo* del fichero de configuración, asistiendo posteriormente de forma interactiva al despliegue y monitorización.

En el capítulo siguiente, se estudiará en mayor detenimiento las tecnologías a emplear en el desarrollo del software.

3 DISEÑO E IMPLEMENTACIÓN

En el presente capítulo, se expone el funcionamiento interno de cada uno de los elementos que componen la aplicación, así como las herramientas empleadas para lograr el comportamiento final que se espera de la misma. Se explicará en detalle todo lo correspondiente al funcionamiento de dichos elementos, así como la forma de interactuar con el sistema desde el punto de vista del usuario para lograr un correcto funcionamiento de este.

Dockerlab es una herramienta que permite el despliegue de laboratorios virtuales haciendo uso del motor Docker. Su funcionamiento queda representado en la Ilustración 6, tomando como entrada un fichero de configuración en formato YAML con una sintaxis reducida que describe el laboratorio virtual a desplegar. A continuación, Dockerlab interpretará la configuración aportada y asistirá al usuario tanto en el despliegue como en la monitorización del escenario, además de proporcionar un archivo **docker-compose.yml** que permitirá la reproducibilidad del escenario en cualquier máquina.

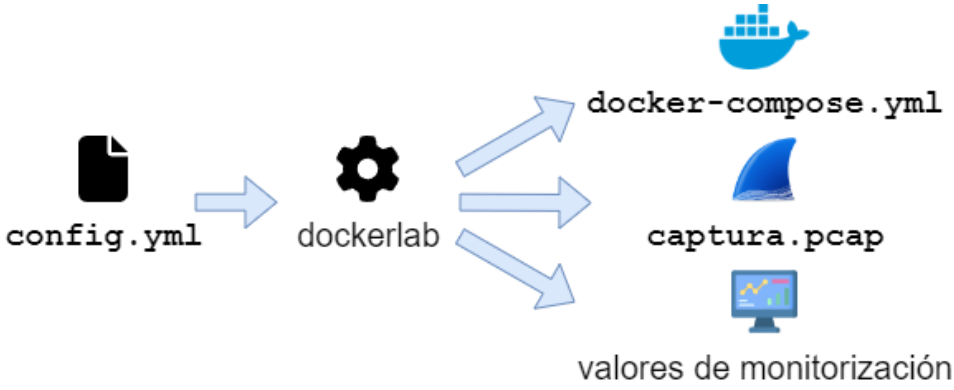

Ilustración 6: Funcionamiento básico de dockerlab

Se adjunta, además, el código del script principal de la aplicación, "*dockerlab.py*", a modo de anexo al final del presente documento.

3.1 Tecnologías empleadas

Para el diseño y realización de la aplicación, se ha trabajado con diferentes tecnologías, las cuales serán expuestas en los siguientes subapartados para así tener un correcto entendimiento de la capacidad de estas y del motivo por el cual se han decidido emplear.

3.1.1 Docker: Despliegue de aplicaciones en contenedores

Docker es una plataforma de virtualización ligera y flexible que permite crear, distribuir y ejecutar aplicaciones de manera eficiente y reproducible. Mediante Docker, se pueden encapsular aplicaciones y todas sus dependencias en **contenedores**.

En un contenedor Docker, se crea un entorno ligero y aislado que opera sobre el sistema operativo anfitrión, utilizando facilidades proporcionadas por el kernel Linux como los espacios de nombres (*namespaces*) y los grupos de control (*cgroups*). Esto permite que cada contenedor tenga su propio sistema de archivos, su propia configuración de red y sus propios recursos, al tiempo que comparte el kernel del sistema operativo anfitrión. La principal diferencia entre un contenedor Docker y una máquina virtual, por tanto, es que el primero se basa en las funcionalidades del kernel y utiliza el aislamiento de recursos y espacios de nombres separados para aislar la vista de una aplicación del sistema operativo anfitrión, mientras que una máquina virtual ejecuta un sistema operativo completo sobre un hipervisor que puede operar en distintos niveles.

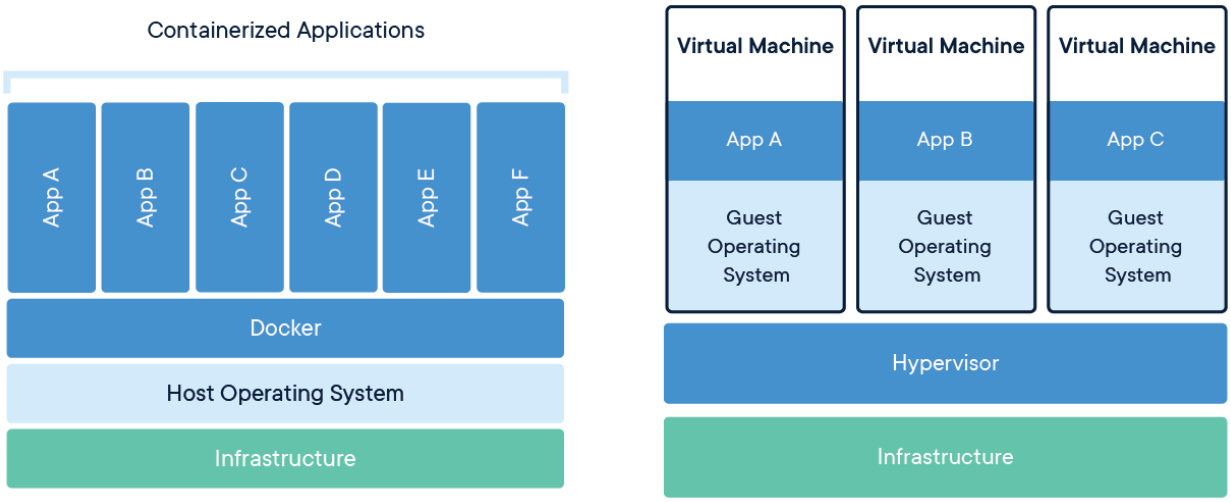

Ilustración 7: Diferencias entre contenedores y Máquinas virtuales.

El funcionamiento de Docker se basa en el uso de imágenes, a partir de las cuáles se crean los contenedores. Una imagen de Docker es un paquete autónomo y autocontenido que incluye todo lo necesario para ejecutar una aplicación, desde el código fuente hasta las bibliotecas y las dependencias. Estas imágenes se crean a partir de un archivo llamado *Dockerfile*, que describe paso a paso los comandos necesarios para construir la imagen.

Tras construir una imagen Docker, ésta se puede ejecutar en uno o varios contenedores a la vez. El contenedor ofrecerá un entorno aislado y consistente para la ejecución de aplicaciones, garantizando que tanto las dependencias como las configuraciones necesarias estén presentes y sean coherentes en cada ejecución. Cabe destacar que las imágenes Docker pueden llegar a ser altamente portables, ya que no son completamente independientes de la infraestructura subyacente, pero se pueden diseñar Dockerfiles multiplataforma, los cuales pueden ser ejecutadas en cualquier entorno que tenga Docker instalado, ya sea en una máquina local, en un servidor de desarrollo o en la nube. Esto permite la creación de entornos de desarrollos consistentes y facilita la implementación de aplicaciones en diferentes entornos sin que las diferencias de configuración supongan ningún problema.

Docker también permite la creación de redes y volúmenes para facilitar tanto la comunicación entre contenedores como la persistencia de datos, lo cual es especialmente útil en entornos de aplicaciones distribuidas, donde diferentes componentes de una aplicación necesitan comunicarse entre sí.

En lo que respecta a la ejecución, Docker proporciona un motor que se ocupa de la gestión de los contenedores y las imágenes, el cual se comunica con el *kernel* del sistema operativo para crear y gestionar los contenedores de manera eficiente. A su vez, Docker proporciona una interfaz de línea de comandos (CLI) y una API que permite a sus usuarios interactuar con los contenedores e imágenes, lo cual facilita la automatización e integración con otras herramientas y sistemas.

3.1.2 Docker Compose: Definición de aplicaciones multicontenedor

Para la definición y el manejo de aplicaciones compuestas por varios contenedores Docker interconectados, existen varias alternativas de herramientas disponibles. Se utilizará Docker Compose, cuyo objetivo principal es simplificar la administración de aplicaciones complejas que consten de múltiples servicios (tales como bases de datos, servidores web o microservicios), al permitir que todos estos componentes funcionen juntos de manera coordinada entre ellos.

La principal funcionalidad ofrecida por Docker Compose está basada en la descripción de la configuración de la aplicación en un archivo YAML llamado “docker-compose.yml”, en el cual se definen los servicios, propiedades, dependencias y configuraciones de la aplicación. Cada servicio se configurará a partir de una imagen de Docker que, o bien puede existir anteriormente, o se puede crear mediante un Dockerfile personalizado.

Los siguientes pasos tienen lugar a lo largo del proceso de uso de Docker Compose:

1. Creación y definición de los archivos de configuración: Las directrices de construcción y ejecución de los contenedores que conforman la aplicación serán encapsuladas en el archivo "docker-compose.yml". Esta definición incluirá todos los servicios, imágenes, volúmenes, redes y configuraciones necesarias para la aplicación.
2. Ejecución de servicios: Haciendo uso del comando "docker-compose up", el archivo de configuración será interpretado por la herramienta, la cual desplegará los contenedores según las indicaciones provistas en el archivo previamente mencionado. Los servicios serán arrancados en orden secuencial, mientras que las redes serán configuradas automáticamente para que los distintos contenedores puedan comunicarse entre sí.
3. Supervisión y gestión: Con los servicios en ejecución, se pueden emplear múltiples comandos para estudiar el estado de estos. Empleando "docker-compose ps", se puede verificar el estado de cada uno de los contenedores, al igual que empleando "docker-compose logs" se pueden consultar los registros de cada servicio. Por último, empleando "docker-compose down" se detendrán y eliminarán los contenedores que se hayan creado anteriormente (en caso de que se desee almacenar información de estos de forma persistente, se deberán utilizar volúmenes).
4. Actualización de servicios: En lo referente a actualizaciones y ajustes de configuración, en caso de desear modificar la imágenes o configuraciones empleadas, se deberá modificar el archivo "docker-compose.yml" y, posteriormente, volver a ejecutar la orden "docker-compose up".

3.1.3 Tshark: Analizador de protocolos de red

En lo que respecta a herramientas de análisis de protocolos, algunas de las más populares son **Wireshark** o **Ethereal**, ambas utilizadas mediante entorno gráfico y que permiten analizar capturas de tráfico de red, sin embargo, debido a la naturaleza del presente proyecto, se ha decidido hacer uso de **Tshark,** una herramienta para el análisis de protocolos que se utiliza, a diferencia de las mencionadas anteriormente, por línea de comandos. Se trata de la versión CLI (command line interface) de Wireshark, proporcionando funcionalidades muy similares a esta.

Debido a que pertenece al conjunto de herramientas Wireshark, Tshark se presenta como una alternativa versátil y eficiente para capturar, decodificar y analizar paquetes de red en tiempo real, almacenar la información recibida en ficheros *pcap* o *pcapng* o analizar el contenido de dichos ficheros, siendo una herramienta esencial para profesionales de seguridad de red, administradores de sistema o analistas de tráfico.

Tshark permite la captura de paquetes de red en vivo desde una o varias interfaces de red, tales como adaptadores Ethernet o interfaces inalámbricas (Wifi, Bluetooth, etcétera). Dichas capturas podrán abarcar diversos tipos de tráfico, permitiendo realizar un profundo análisis sobre el comportamiento de la red.

Gracias a su capacidad para decodificar los paquetes recibidos, Tshark ofrece los detalles y contenidos de cada paquete traducidos a un formato legible y estructurado, permitiendo aplicar filtros para seleccionar y visualizar los paquetes que se deseen en función de ciertos criterios específicos. Dichos filtros pueden estar basados en direcciones de origen o destino, puertos, protocolos empleados u otros atributos, agilizando así el proceso de análisis y permitiendo ignorar los paquetes que no se consideren como relevantes para un escenario particular.

Por último, Tshark permite plasmar los resultados obtenidos de la captura y el análisis en diversos formatos, tales como texto plano, CSV, JSON o XML, lo cual facilita su integración con otras herramientas tales como la que se ha recogido en el presente documento.

3.2 Diseño y Arquitectura

En lo que respecta al software desarrollado, en este apartado se estudiará la utilización, así como la arquitectura interna que permiten obtener el resultado deseado. Se estudiarán, además, las diversas opciones disponibles al usuario mediante las cuales se puede obtener un comportamiento concreto.

3.2.1 Visión general del sistema

La contribución principal de este trabajo ha sido llamada 'dockerlab', y tiene como objetivo facilitar el despliegue, ejecución y monitorización de laboratorios virtuales simplificando en gran medida la complejidad de los ficheros docker-compose. Para ello, puede llamarse por línea de comandos indicando un fichero de configuración (en adelante, config.yml), y unas banderas de ejecución, iniciando un proceso interactivo. La Ilustración 8 representa el diagrama de flujo que sigue la aplicación.

En el siguiente diagrama se expone de forma resumida el funcionamiento de la aplicación dockerlab.

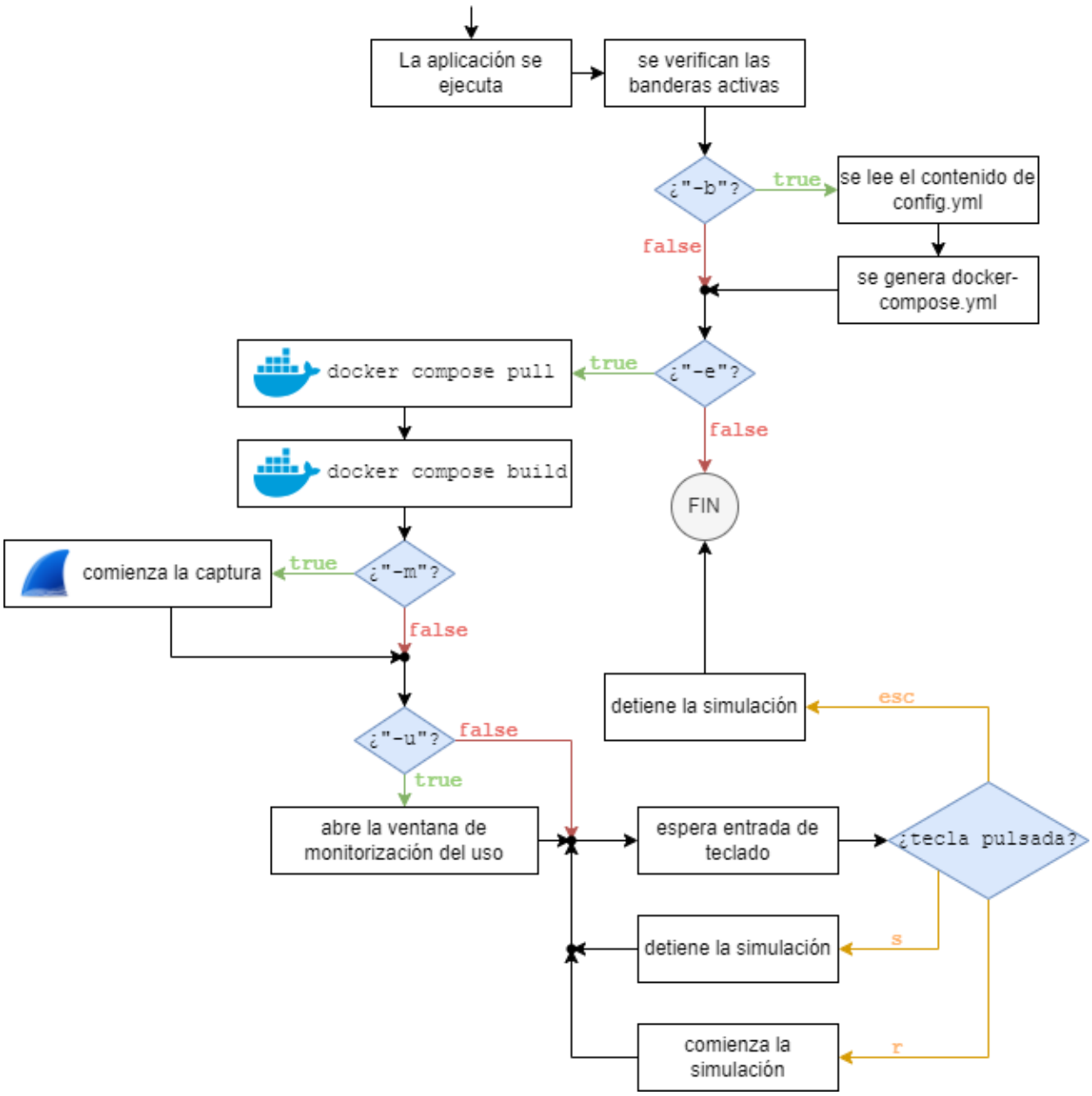

Ilustración 8: Funcionamiento resumido de dockerlab

Dockerlab puede ser ejecutado con cuatro banderas distintas (**b**uild, **e**xecute, **m**onitor y **u**sage) las cuales definirán el comportamiento del programa, ya sea si se desea generar un archivo docker-compose.yml a partir del archivo de configuración proporcionado (config.yml), si se desea ejecutar Docker Compose a partir de dicho archivo, o si se desea algún tipo de monitorización (de red o de recursos) a lo largo de dicha ejecución.

3.2.2 Elementos de entrada

Para utilizar la aplicación, primero se debe comprender el funcionamiento del sistema de ficheros que la componen. A continuación de muestra una imagen con el contenido de la carpeta en la que está situado el script "dockerlab.py".

Cada uno de los archivos que componen la aplicación tiene un propósito, algunos no han de ser modificados debido a que definen el modo de funcionamiento del software, mientras que otros sirven de archivos de configuración del escenario de ejemplo. Estos últimos deberán ser modificados o sustituidos por los archivos de configuración del escenario que deseemos ejecutar.

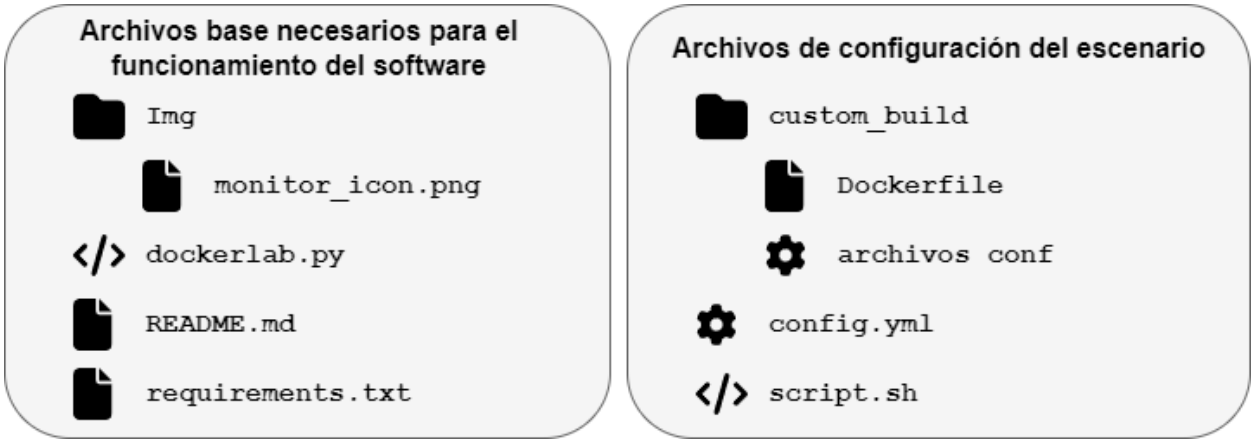

Ilustración 9: Clasificación de los archivos de la aplicación

En lo que respecta a los archivos base necesarios para el funcionamiento del software, tienen el siguiente propósito:

- Carpeta "**Img**": contiene el icono (monitor_icon.png) de las ventanas que se mostrará en caso de desear ejecutar la aplicación en modo "monitor".
- **dockerlab.py**: script troncal de la aplicación. Más adelante se estudiará cómo manejarlo, ya que es el archivo que ha de ser interpretado por Python para permitir la ejecución del software.
- **README.md**: archivo "léeme" escrito en *Markdown*. Contiene toda la información necesaria para comprender el funcionamiento de la aplicación, así como su manejo.
- **requirements.txt**: archivo que contiene una lista con las librerías de Python de las que depende el software. Más adelante se estudiará

Por otro lado, los archivos de configuración del escenario de ejemplo son los siguientes:

- **config.yml**: archivo de configuración en formato YAML, el cual contendrá las especificaciones del laboratorio virtual a desplegar. Se estudiará su contenido más adelante, sin embargo, cabe destacar que la existencia de este archivo es obligatoria para el correcto funcionamiento del sistema.
- Carpeta "**custom_build**": en caso de querer basar un nodo en un contenedor definido por el usuario, se incluirá una carpeta en el directorio actual que contenga los archivos necesarios para su despliegue y se indicará en esta directiva su nombre.
    - **Dockerfile**: para construir una imagen personalizada para el contenedor, se deberá incluir un archivo "Dockerfile" en el cual se especifique la configuración de dicha imagen.
    - **Archivos conf**: archivo(s) de configuración necesarios para la imagen "custom_build".
- **script.sh**: se trata de un script de ejemplo. Contiene las órdenes que ejecutarán los diversos contenedores una vez comience a funcionar la aplicación.

Es importante destacar que dockerlab mapea el directorio en el que se encuentra y lo monta en la carpeta */workspace* dentro de cada uno de cada uno de los contenedores que genera, por lo que se puede acceder a cualquier script o archivo que esté situado en dicha carpeta desde cualquiera de los contenedores que se levanten, y este directorio puede ser utilizado para volcar cualquier fichero de salida de la ejecución, garantizando su persistencia.

Se va a mostrar ahora un escenario de ejemplo, el cual consta de un contenedor servidor y otro que hace de cliente. Ambos contenedores ejecutarán un script distinto cada uno (*script1.sh* y *script0.sh* respectivamente). El primer contenedor tendrá una imagen personalizada llamada *custom_build*, mientras que el segundo utilizará una imagen de *alpine* genérica.

En lo que respecta al contenido de **config.yml**, a continuación, se muestra un extracto de este:

```yaml
prueba_ping:
  network: 10.10.0.0/24
  nodes:
    servidor:
      build: custom_build
      ip: 10.10.0.2
      script: script1.sh
    cliente:
      image: alpine:latest
      script: script0.sh
      ip: 10.10.0.3
      needs:
        - servidor
```

Ilustración 10: Extracto de config.yml

El archivo YAML debe seguir la estructura expuesta a continuación:

- **<lab_name>**: nombre del laboratorio (en el ejemplo, “lab”).
    - **network**: dirección IPv4 que indica la subred del laboratorio.
    - **nodes**: lista de nodos que componen el laboratorio.
        - **<node_name>**: nombre del nodo en cuestión.
            - **image**: imagen en la que se va a basar el nodo. Es excluyente con la funcionalidad “build”.
            - **build**: en caso de querer basar un nodo en un contenedor definido por el usuario, se incluirá una carpeta en el directorio actual que contenga los archivos necesarios para su despliegue y se indicará en esta directiva su nombre. Es excluyente con la funcionalidad “image”.
            - **script**: en caso de querer que se ejecute un shell-script en el contenedor a desplegar cuando este se inicie, se debe indicar aquí su nombre tal y como esté almacenado en el directorio actual.
            - **network**: si se desea que se le asigne una dirección IP en un subrango de la red del laboratorio, se indicará en esta directiva. Es excluyente con la funcionalidad “ip”.
            - **ip**: si se desea que al nodo actual se le despliegue con una dirección IP concreta dentro del rango de la red del laboratorio, se indicará en esta directiva. Es excluyente con las funcionalidades “network” y “replicas”.

- **replicas**: en caso de desear desplegar varios contenedores con configuraciones similares, se indicará en esta directiva el número de instancias a desplegar. Es incompatible con la funcionalidad "ip" sólo en caso de que su valor sea superior a "1". Cada réplica tendrá una variable de entorno "**$REPLICA_ID**" con un identificador que ayude a diferenciarla de las demás (un valor entre 0 y el número máximo de réplicas - sin incluir este último).
- **needs**: lista de dependencias para el despliegue del contenedor. Sirve para generar un orden de despliegue personalizado.

Es especialmente interesante comparar el archivo de configuración expuesto en la figura anterior con el archivo docker-compose.yml generado como resultado de la ejecución de dockerlab. En este caso concreto, a pesar de ser una configuración relativamente trivial, el docker-compose.yml resultante casi duplica el número de líneas necesarias para la configuración del escenario. Esta simplificación en lo que respecta a la generación del escenario es especialmente visible en archivos de configuración más extensos, con múltiples réplicas e imágenes de contenedores variadas.

```
networks:
  prueba_ping_network:
    external: true
services:
  cliente:
    depends_on:
    - servidor
    entrypoint: /bin/sh /workspace/script0.sh
    image: alpine:latest
    networks:
      prueba_ping_network:
        ipv4_address: 10.10.0.3
    volumes:
    - ./:/workspace
  servidor:
    build: custom_build
    entrypoint: /bin/sh /workspace/script1.sh
    networks:
      prueba_ping_network:
        ipv4_address: 10.10.0.2
    volumes:
    - ./:/workspace
version: '3.3'
```

Ilustración 11: Archivo docker-compose.yml generado tras la ejecución de dockerlab

3.2.3 Instalación de Dependencias

Para la ejecución del script "dockerlab.py" se deben cumplir los siguientes requisitos:

- Docker 23.0.3 o posteriores
- Docker-compose 1.29.2 o posteriores
- Python 3.10.6 o posteriores
- Librerías contenidas en el archivo "requirements.txt".

Para la instalación de dependencias se muestra a continuación el proceso a seguir en un equipo con una distribución de Linux basada en Debian (se han de realizar las siguientes acciones con permisos de superusuario):

1. Se actualiza la lista de repositorios y los paquetes actualmente instalados:

```
apt update
apt upgrade
```

Código 3: Comandos a ejecutar para actualizar la lista de repositorios y los paquetes instalados

2. Haciendo uso del gestor de paquetes, se instalan las últimas versiones del software necesario:

```
apt install docker
apt install docker-compose
apt install python3
apt install python3-tk
```

Código 4: Comandos para instalar el software del que depende dockerlab

3. Para instalar las librerías necesarias, situarse en la carpeta contenedora del archivo "requirements.txt" y ejecutar la orden:

```
python3 -m pip install -r requirements.txt
```

Código 5: Comando para instalar las dependencias de Python

Cabe destacar que se ha probado el correcto funcionamiento de la aplicación con las versiones indicadas de las dependencias, por lo que, aunque puede que funcione correctamente con algunas versiones anteriores, sólo se puede garantizar que no se van a encontrar problemas inesperados con las versiones previamente indicadas.

También es importante destacar el hecho de que, en caso de querer utilizar la herramienta mediante ssh, se deberá redirigir adecuadamente el servidor X. En caso contrario, dockerlab no funcionará.

3.2.4 Comportamiento de la herramienta

En el presente apartado se estudiará el comportamiento interno del software, diferenciando entre algunos de los subprocesos más relevantes que se generarán según el modo de ejecución seleccionado.

Para empezar, en el momento de creación del archivo "docker-compose.yml", actuarán dos subprocesos adicionales al proceso principal del programa: uno de ellos será el encargado de eliminar otras redes docker definidas que interfieran con la red que se desea crear, mientras que el otro será el encargado de generar dicha red. En la Ilustración 12 se pueden apreciar los procesos generados para esta tarea, así como los mensajes que intercambian entre sí.

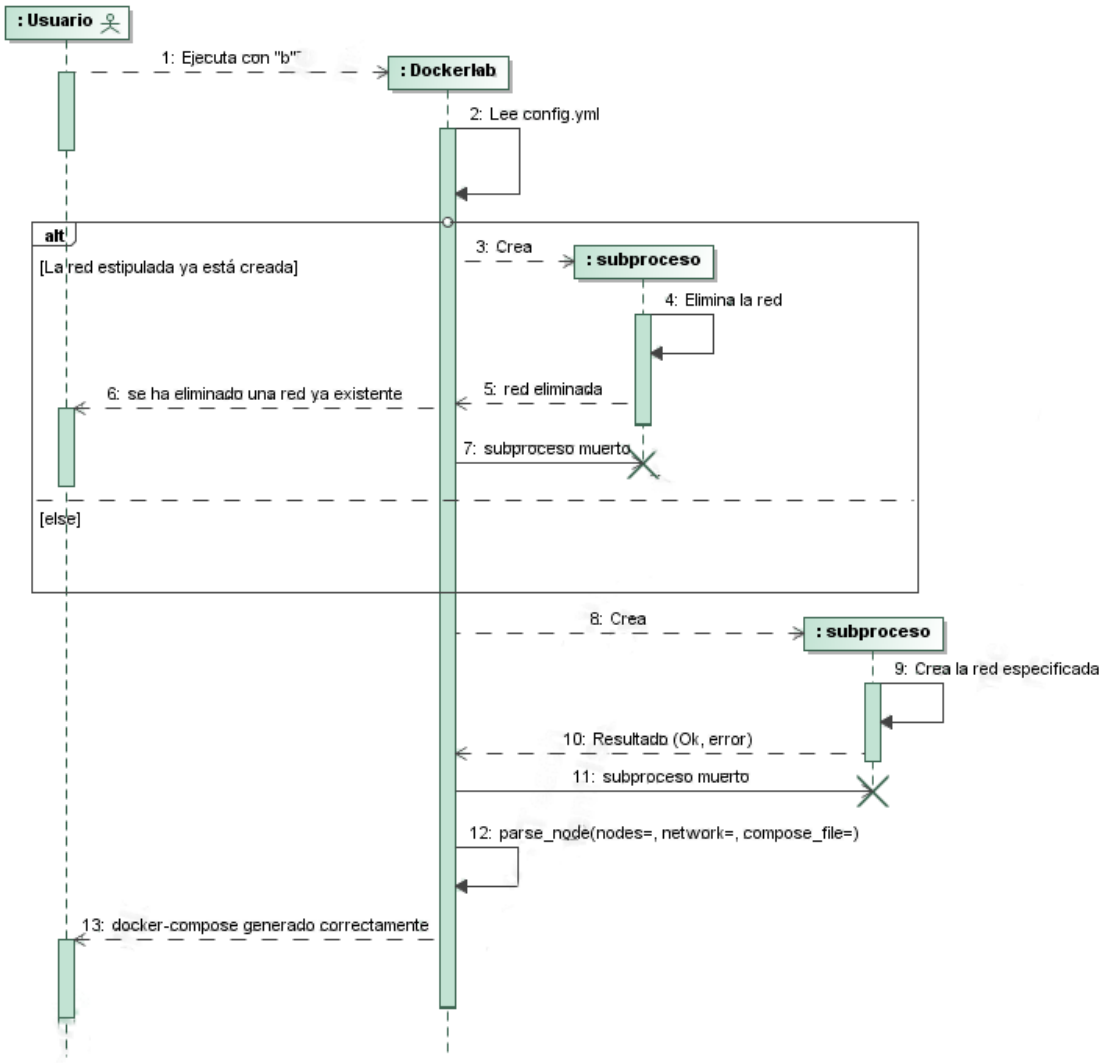

Ilustración 12: Subprocesos generados al construir el archivo docker-compose.yml

A continuación, en caso de desear ejecutar la simulación con el archivo docker-compose.yml generado, se diferenciará entre el modo de ejecución "básica", la ejecución en modo "monitor" y la ejecución en modo "utilización". Los tres modos de ejecución son compatibles entre sí y en el diagrama expuesto a continuación se pueden observar los hilos y subprocesos involucrados en el proceso.

En la Ilustración 13 se puede apreciar el uso de dos subprocesos distintos para realizar las órdenes de *docker-compose pull* y *docker-compose build* necesarias para el correcto funcionamiento de la aplicación. Del mismo modo, en caso de estar en modo monitor, se creará un hilo que ejecutará el subproceso *tshark* para la captura del tráfico de red, sin embargo, en caso de estar en modo utilización, se generará un hilo que se ocupará de la gestión de la interfaz gráfica de usuario (GUI) que aparecerá en pantalla, y que actualizará los valores de la monitorización de recursos utilizados cada vez que el usuario pulse el botón de "actualizar".

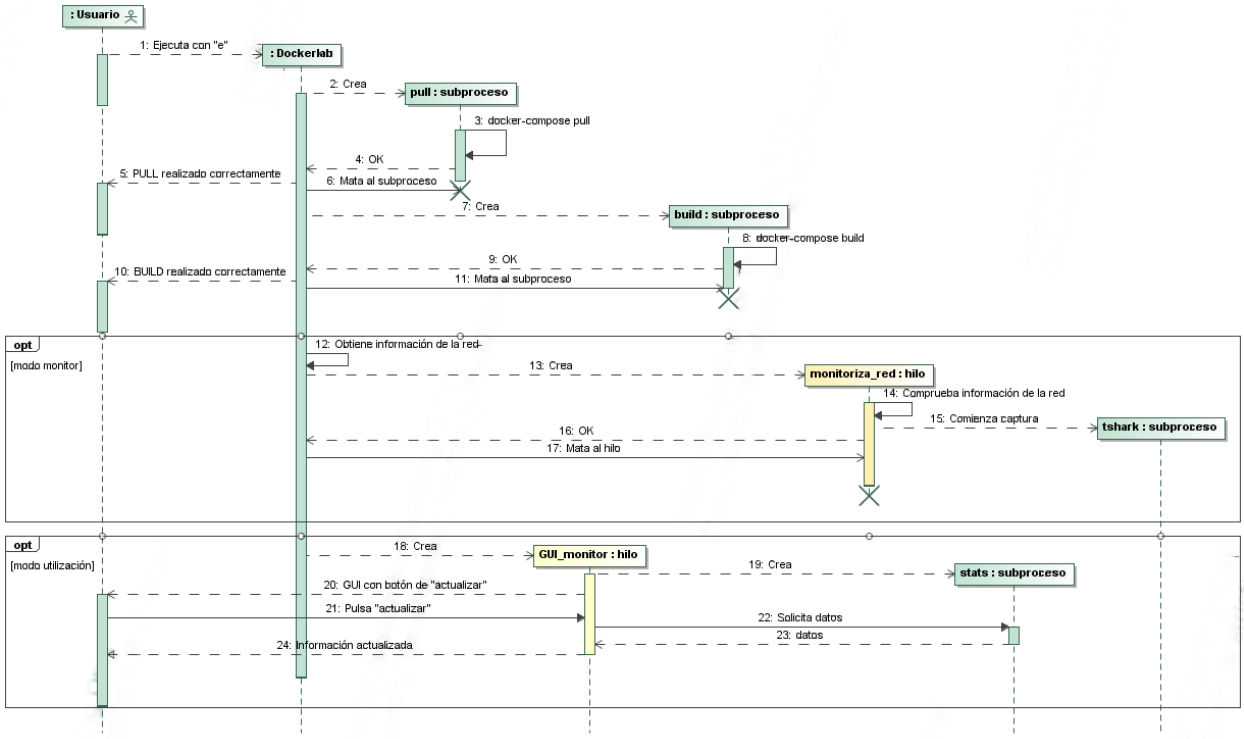

Ilustración 13: Procesos generados durante la ejecución del software

Por último, hay que tener en cuenta que mientras la aplicación está en ejecución, el usuario podrá controlarla haciendo uso de algunas teclas a las que se les otorga una tarea específica: "r" para ejecutar el sistema, "s" para pararlo y "esc" para finalizar la aplicación. La Ilustración 14 muestra cómo se gestionan estos eventos desde el punto de vista de los subprocesos, parando el sistema en caso de que se pulse "s" o "esc" e iniciándolo en caso de pulsar "r".

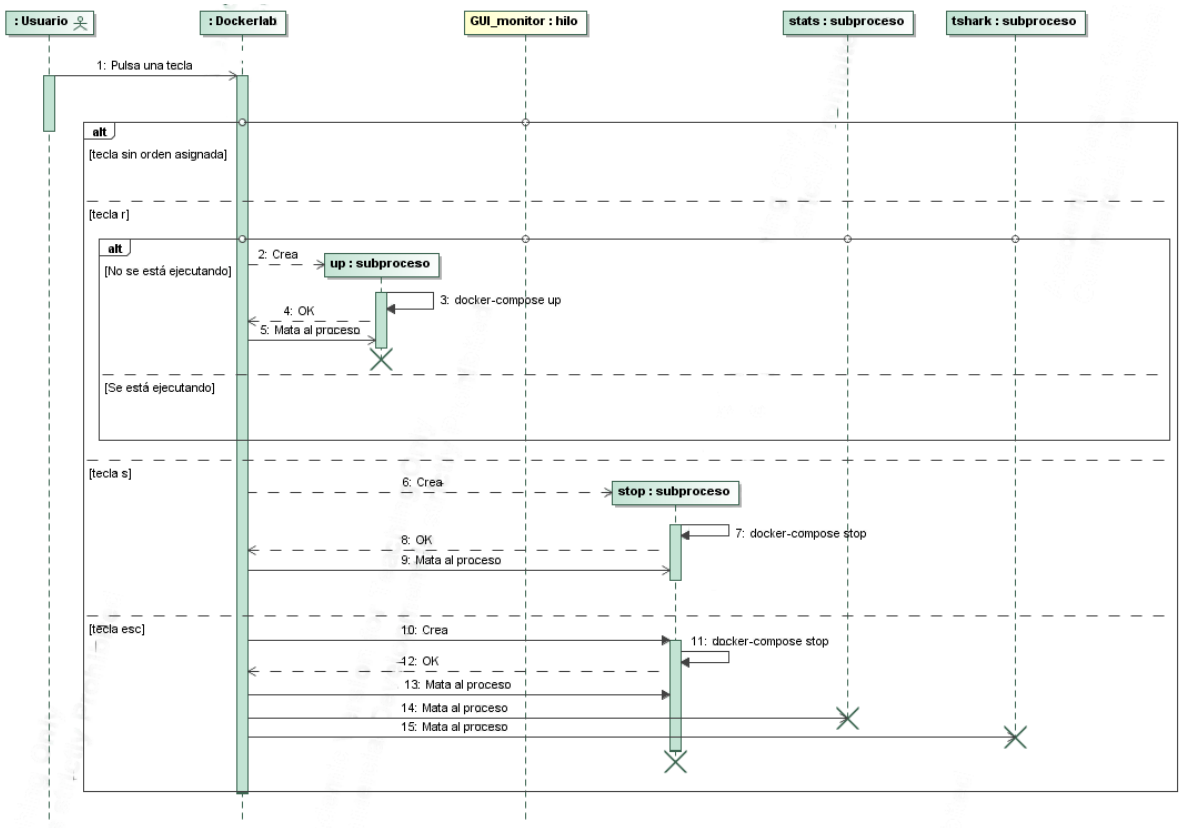

Ilustración 14: Procesos generados para la gestión de entrada de usuario

3.3 Ejemplo de ejecución/uso

En el siguiente apartado se estudiarán las diferentes opciones de ejecución disponibles para dockerlab. El caso de ejemplo expuesto consistirá en un intercambio de mensajes PING entre dos equipos conectados a una misma red.

Para conocer las diversas funcionalidades que ofrece el software, basta con ejecutar el archivo "dockerlab.py" con la bandera "-h" o "--help". Una vez lo hagamos, se obtendrá la siguiente información por pantalla:

```
usage: dockerlab.py [-h] [-b] [-e] [-m] [-u]

Herramienta para el despliegue de laboratorios virtuales mediante Docker. Por defecto, en caso de no proporcionar parámetros, se ejecutará con las banderas -be.

options:
  -h, --help     show this help message and exit
  -b, --build    Indica que se desea generar el archivo docker-compose.yml. Si sólo se selecciona esta opción, no se crearán los contenedores pertinentes.
  -e, --execute  Indica que se desean crear y levantar los contenedores definidos en docker-compose.yml.
  -m, --monitor  Monitoriza el tráfico de paquetes en la red simulada. Debe usarse junto con -e.
  -u, --usage    Monitoriza el uso de recursos dentro de los contenedores de la simulación. Debe usarse junto con -e.
```

Ilustración 15: Resultado de ejecutar la bandera "-h" o "--help"

Como podemos comprobar, existen cuatro modos de funcionamiento diferenciados:

- **-b** o **--build**: Indica que se desea generar el archivo "docker-compose.yml". Si sólo se selecciona esta opción, no se crearán los contenedores pertinentes.
- **-e** o **--execute**: Indica que se desean crear y levantar los contenedores definidos en "docker-compose.yml".
- **-m** o **--monitor**: Monitoriza el tráfico de paquetes en la red simulada. Debe usarse junto con `-e`.
- **-u** o **--usage**: Monitoriza el uso de recursos dentro de los contenedores de la simulación. Debe usarse junto con `-e`.

Por defecto, en caso de no proporcionar parámetros, se ejecutará con las banderas "**-be**".

Una vez el archivo "docker-compose.yml" haya sido creado, se proporcionará la opción de correr la simulación pulsando la tecla "**r**" y de pararla pulsando la tecla "**s**". Para salir de la aplicación, se debe pulsar la tecla "**esc**".

A continuación, se muestra un ejemplo de ejecución del software hasta el punto de solicitar una entrada por parte del usuario.

```
        Red [10.10.0.0/24] añadida
        Nodo [servidor] añadido
        Nodo [cliente] añadido
Se ha leído config.yml correctamente
Red implementada correctamente.
'docker-compose.yml' generado correctamente.
Haciendo pull a los contenedores...
Pulling cliente ... done
Código de retorno: 0
¡Pull realizado correctamente!
Construyendo contenedores...
cliente uses an image, skipping
Building servidor
Step 1/1 : FROM alpine:latest
 ---> 7e01a0d0a1dc

Successfully built 7e01a0d0a1dc
Successfully tagged tfg-escenario0_servidor:latest
Código de retorno: 0
¡Contenedores construidos satisfactoriamente!
10.10.0.0/24 -> 10.10.0.1
Comienza la monitorización!
Pulse 'r' para ejecutar docker-compose en segundo plano y 's' para parar su ejecución. Para salir de la aplicación, pulse la tecla 'esc'.
```

Ilustración 16: Ejemplo de pasos previos a la ejecución del software

En caso de pulsar "r", docker-compose se ejecutará en segundo plano con el escenario que haya sido programado. La salida estándar de todos los contenedores que se estén ejecutando pasará a ser la salida estándar del programa, tal y como se puede apreciar en la siguiente captura de pantalla.

```
Capturing on 'br-dockerlab'
Valores de monitorización refrescados
Valores de monitorización refrescados
Valores de monitorización refrescados
Corriendo el subproceso 'docker-compose'...
Creando subproceso docker-compose...
Subproceso creado correctamente!
Recreating tfg-escenario0_servidor_1 ... done
Recreating tfg-escenario0_cliente_1  ...
Recreating tfg-escenario0_cliente_1  ... done
Attaching to tfg-escenario0_servidor_1, tfg-escenario0_cliente_1
servidor_1  | PING 10.10.0.3 (10.10.0.3): 56 data bytes
cliente_1   | PING 10.10.0.2 (10.10.0.2): 56 data bytes
cliente_1   | 64 bytes from 10.10.0.2: seq=0 ttl=64 time=0.110 ms
servidor_1  | 64 bytes from 10.10.0.3: seq=0 ttl=64 time=344.525 ms
10 servidor_1  | 64 bytes from 10.10.0.3: seq=1 ttl=64 time=0.072 ms
16 cliente_1   | 64 bytes from 10.10.0.2: seq=1 ttl=64 time=0.052 ms
19 servidor_1  | 64 bytes from 10.10.0.3: seq=2 ttl=64 time=0.145 ms
cliente_1   | 64 bytes from 10.10.0.2: seq=2 ttl=64 time=0.088 ms
30 Valores de monitorización refrescados
40 servidor_1  | 64 bytes from 10.10.0.3: seq=3 ttl=64 time=0.171 ms
cliente_1   | 64 bytes from 10.10.0.2: seq=3 ttl=64 time=0.162 ms
46 Valores de monitorización refrescados
servidor_1  | 64 bytes from 10.10.0.3: seq=4 ttl=64 time=0.144 ms
50 cliente_1   | 64 bytes from 10.10.0.2: seq=4 ttl=64 time=0.159 ms
55 Valores de monitorización refrescados
servidor_1  | 64 bytes from 10.10.0.3: seq=5 ttl=64 time=0.183 ms
58 cliente_1   | 64 bytes from 10.10.0.2: seq=5 ttl=64 time=0.151 ms
Valores de monitorización refrescados
```

Ilustración 17: Ejemplo de ejecución de docker-compose mediante dockerlab

Una vez comienza la ejecución de los contenedores, esta podrá ser detenida al pulsar la tecla "s", mandando así una señal simultánea a cada uno de dichos contenedores para terminar su ejecución.

```
Parando el subproceso...
Valores de monitorización refrescados
Stopping tfg-escenario0_cliente_1  ...
Stopping tfg-escenario0_servidor_1 ...
98 servidor_1  | 64 bytes from 10.10.0.3: seq=9 ttl=64 time=0.178 ms
104 cliente_1   | 64 bytes from 10.10.0.2: seq=9 ttl=64 time=0.165 ms
Valores de monitorización refrescados
106 servidor_1  | 64 bytes from 10.10.0.3: seq=10 ttl=64 time=0.166 ms
113 cliente_1   | 64 bytes from 10.10.0.2: seq=10 ttl=64 time=0.167 ms
Valores de monitorización refrescados
```

Ilustración 18: Ejemplo de salida al detener la ejecución de los contenedores

```
Stopping tfg-escenario0_cliente_1  ... done
160 tfg-escenario0_cliente_1 exited with code 137
164 Valores de monitorización refrescados
165 Valores de monitorización refrescados
171 Valores de monitorización refrescados
174 Valores de monitorización refrescados
180 Valores de monitorización refrescados
187 Valores de monitorización refrescados
195 Valores de monitorización refrescados
197 Valores de monitorización refrescados
204 Valores de monitorización refrescados
Stopping tfg-escenario0_servidor_1 ... done
tfg-escenario0_servidor_1 exited with code 137
Código de retorno: 0
Código de retorno: 0
Valores de monitorización refrescados
```

Ilustración 19: Indicación de que se ha detenido la ejecución de los contenedores correctamente

En caso de haber ejecutado dockerlab con la bandera "**-u**", cuando se pulse "r" por primera vez, saltará una ventana como la que se muestra a continuación, la cual podrá ser refrescada al pulsar el botón de "Refrescar" situado en la esquina inferior izquierda de la misma.

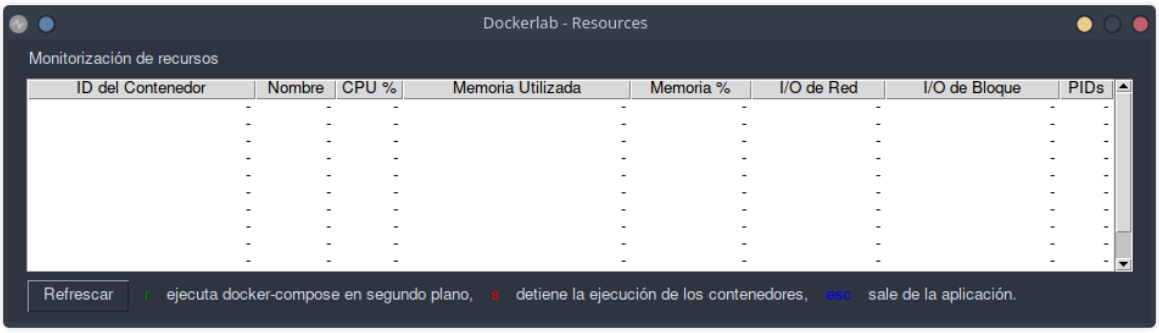

Ilustración 20: Ventana de "utilización" antes de recibir ningún dato

El programa refrescará automáticamente los datos a mostrar en la ventana anterior periódicamente, indicando al usuario que hay nuevos datos disponibles mediante el mensaje "Valores de monitorización refrescados".

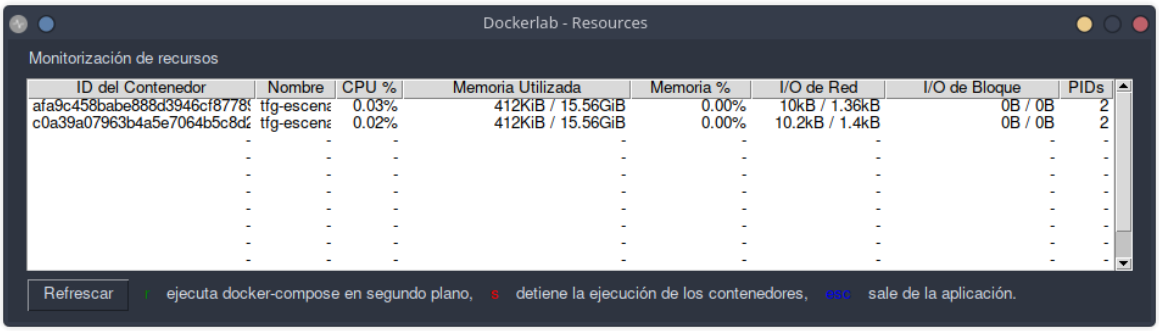

Ilustración 21: Ventana de "utilización" una vez hay datos que mostrar

Por último, cabe destacar que el uso de la bandera "**-m**" generará un archivo "output.pcap" en el que se recogerá todo el tráfico de red generado entre los contenedores. Para poder observar su contenido, se podrán utilizar herramientas como Wireshark, las cuales mostrarán la información contenida en cada uno de los paquetes intercambiados.

No.	Time	Source	Destination	Protocol	Lengtl	Info
1	0.000000000	02:42:8a:e7:d2:ce	Broadcast	ARP	42	ARP Announcement for 10.10.0.1
4	0.082458945	02:42:0a:0a:00:02	Broadcast	ARP	42	Who has 10.10.0.3? Tell 10.10.0.2
5	0.561755600	02:42:0a:0a:00:03	Broadcast	ARP	42	Who has 10.10.0.2? Tell 10.10.0.3
6	0.561792570	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) request id=0x0008, seq=0/0, ttl=64 (reply in 9)
7	0.561795210	02:42:0a:0a:00:02	02:42:0a:0a:00:03	ARP	42	10.10.0.2 is at 02:42:0a:0a:00:02
8	0.561835776	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) request id=0x0007, seq=0/0, ttl=64 (reply in 10)
9	0.561836088	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) reply id=0x0008, seq=0/0, ttl=64 (request in 6)
10	0.561860494	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) reply id=0x0007, seq=0/0, ttl=64 (request in 8)
13	1.082654713	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) request id=0x0008, seq=1/256, ttl=64 (reply in 14)
14	1.082722660	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) reply id=0x0008, seq=1/256, ttl=64 (request in 13)
17	1.561942415	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) request id=0x0007, seq=1/256, ttl=64 (reply in 18)
18	1.562014173	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) reply id=0x0007, seq=1/256, ttl=64 (request in 17)
25	2.000153683	02:42:8a:e7:d2:ce	Broadcast	ARP	42	ARP Announcement for 10.10.0.1
27	2.082844397	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) request id=0x0008, seq=2/512, ttl=64 (reply in 28)
28	2.082913477	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) reply id=0x0008, seq=2/512, ttl=64 (request in 27)
35	2.562109440	10.10.0.3	10.10.0.2	ICMP	98	Echo (ping) request id=0x0007, seq=2/512, ttl=64 (reply in 36)
36	2.562180858	10.10.0.2	10.10.0.3	ICMP	98	Echo (ping) reply id=0x0007, seq=2/512, ttl=64 (request in 35)

Ilustración 22: Fragmento de output.pcap

4 CASOS DE USO

Dokerlab es una herramienta que puede ser adaptada para diversos escenarios. En el siguiente apartado se estudiarán distintos casos de uso, mostrándose el resultado de la ejecución de la herramienta en distintos escenarios y analizando la información obtenida de la ejecución de cada uno de ellos.

4.1 Red MQTT

El siguiente caso de estudio tratará una red MQTT, en la que cinco clientes envían mensajes en un tema determinado a un contenedor que cumplirá la función de bróker. Se empleará una imagen personalizada de Eclipse Mosquitto para el bróker y la imagen general para los clientes.

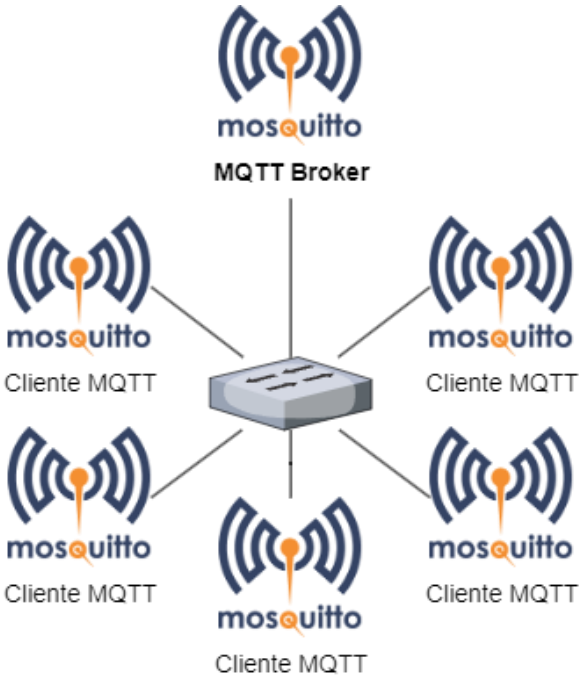

Ilustración 23: Escenario 1. Red Industrial MQTT

El presente escenario tiene su inspiración en replicar las necesidades del artículo "*Smart home anomaly-based IDS: Architecture proposal and case study*" [8], en el que se partía de una serie de ficheros csv que indicaban el contenido y el *timestamp* de los paquetes que tenían lugar en una *smarthome*, sin embargo, la complejidad no estaba ahí, si no que radicaba en la recreación del tráfico MQTT.

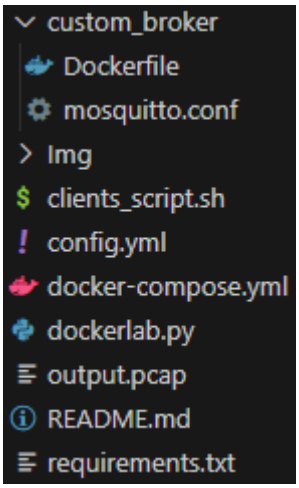

Ilustración 24: Árbol de ficheros empleados en el caso 1

La imagen personalizada que se ha empleado para el bróker viene definida en el archivo Dockerfile, dentro de la carpeta "custom_broker". Importa la imagen oficial de Eclipse Mosquitto y le añade el archivo de configuración necesario, tal y como se ve a continuación:

```
FROM eclipse-mosquitto:latest

ADD mosquitto.conf /mosquitto/config/mosquitto.conf
```

Código 6: Dockerfile del broker MQTT del caso de uso 1

El archivo de configuración empleado sirve para permitir conexiones anónimas (esto es, no se requiere de autenticación de los clientes) y pone a escuchar al bróker en el puerto 1883. Su contenido es el siguiente:

```
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log

listener 1883 0.0.0.0
allow_anonymous true
```

Código 7: Contenido de mosquitto.conf del caso de uso 1

En lo que respecta al script que van a ejecutar los clientes para enviar los mensajes periódicamente, tiene el contenido expuesto a continuación:

```
#!/bin/bash
echo REPLICA_ID = $REPLICA_ID

for i in `seq 1 60`
do
   mosquitto_pub -h mqtt_broker -t /test/message -m "Mensaje número ${i}" && \
      echo "Cliente $REPLICA_ID envió el mensaje número $i" || \
      echo "Cliente $REPLICA_ID no pudo envíar el mensaje número $i"

   sleep $((REPLICA_ID + 1))
done
```

Código 8: Script a ejecutar por los clientes MQTT del caso de uso 1

En este ejemplo en concreto, además, se puede apreciar cómo el uso de dockerlab ha ayudado a reducir considerablemente la cantidad configuración requerida por parte del usuario, transformando un archivo de configuración de la aplicación de 12 líneas en un docker-compose.yml de más de 70.

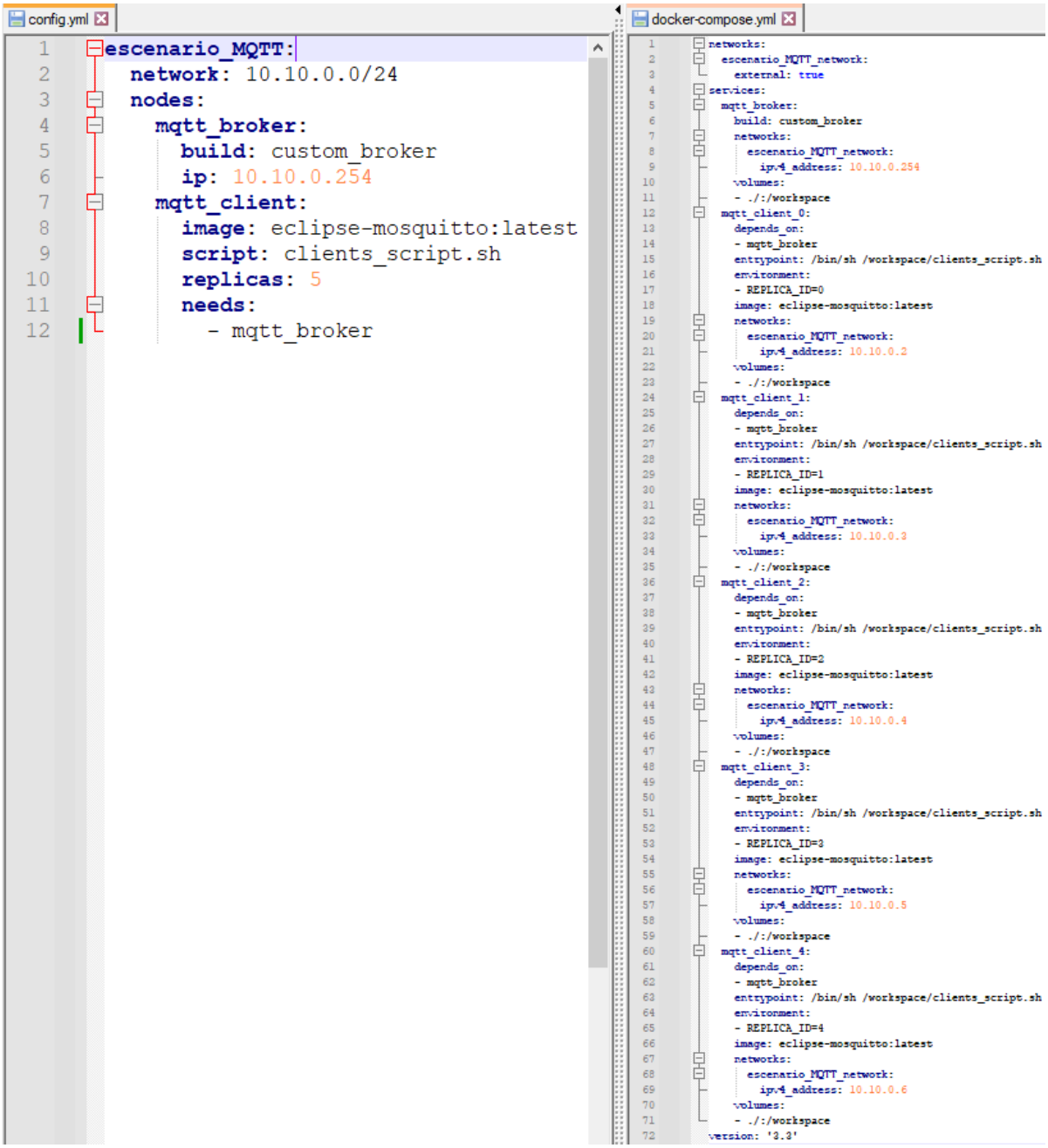

Ilustración 25: Comparación entre el archivo config.yml y docker-compose.yml generado en el escenario 1

En lo que respecta al tráfico generado, se pueden apreciar fácilmente los mensajes MQTT enviados en ambos sentidos gracias a la captura de tráfico realizada por dockerlab.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.481982053	10.10.0.5	10.10.0.254	TCP	74	41726 → 1883 [SYN] Seq=0 Win=642
8	0.482023182	10.10.0.254	10.10.0.5	TCP	74	1883 → 41726 [SYN, ACK] Seq=0 Ac
9	0.482043434	10.10.0.5	10.10.0.254	TCP	66	41726 → 1883 [ACK] Seq=1 Ack=1 W
10	0.482102457	10.10.0.5	10.10.0.254	MQTT	80	Connect Command
11	0.482115753	10.10.0.254	10.10.0.5	TCP	66	1883 → 41726 [ACK] Seq=1 Ack=15
12	0.482286838	10.10.0.254	10.10.0.5	MQTT	70	Connect Ack
13	0.482309046	10.10.0.5	10.10.0.254	TCP	66	41726 → 1883 [ACK] Seq=15 Ack=5
14	0.485159051	10.10.0.5	10.10.0.254	MQTT	100	Publish Message [/test/message]
15	0.485217213	10.10.0.5	10.10.0.254	MQTT	68	Disconnect Req
16	0.485282052	10.10.0.254	10.10.0.5	TCP	66	1883 → 41726 [ACK] Seq=5 Ack=52
17	0.485359695	10.10.0.254	10.10.0.5	TCP	66	1883 → 41726 [FIN, ACK] Seq=5 Ac
18	0.485377985	10.10.0.5	10.10.0.254	TCP	66	41726 → 1883 [ACK] Seq=52 Ack=6
23	0.782971952	10.10.0.2	10.10.0.254	TCP	74	34022 → 1883 [SYN] Seq=0 Win=642
24	0.783010840	10.10.0.254	10.10.0.2	TCP	74	1883 → 34022 [SYN, ACK] Seq=0 Ac
25	0.783030781	10.10.0.2	10.10.0.254	TCP	66	34022 → 1883 [ACK] Seq=1 Ack=1 W
26	0.783078949	10.10.0.2	10.10.0.254	MQTT	80	Connect Command
27	0.783093340	10.10.0.254	10.10.0.2	TCP	66	1883 → 34022 [ACK] Seq=1 Ack=15
28	0.783183850	10.10.0.254	10.10.0.2	MQTT	70	Connect Ack
29	0.783207036	10.10.0.2	10.10.0.254	TCP	66	34022 → 1883 [ACK] Seq=15 Ack=5
30	0.783248986	10.10.0.2	10.10.0.254	MQTT	100	Publish Message [/test/message]
31	0.783288058	10.10.0.2	10.10.0.254	MQTT	68	Disconnect Req

Ilustración 26: Captura del tráfico generado en el escenario 1

A continuación, se muestran capturas tanto de la utilización de recursos de los distintos contenedores durante su ejecución como de su salida estándar visto desde la consola de dockerlab.

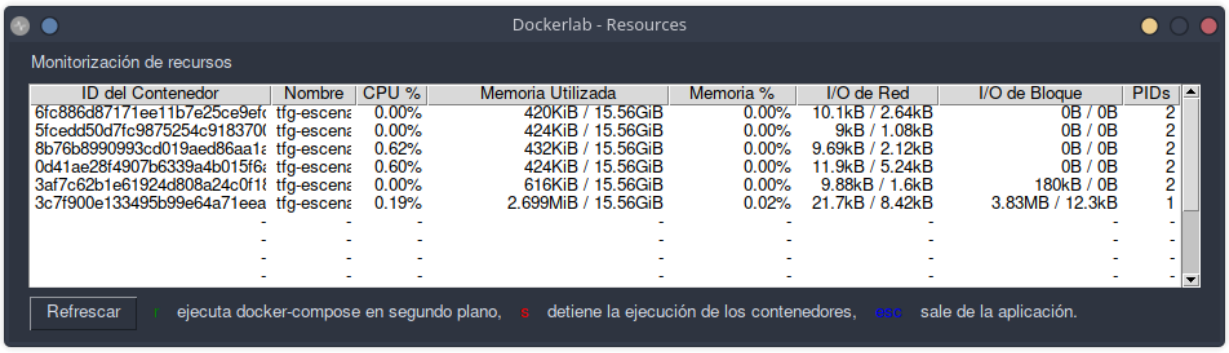

Ilustración 27: Utilización de recursos de cada uno de los contenedores del escenario 1

```
mqtt_client_1_1  | REPLICA_ID = 1
mqtt_client_1_1  | Cliente 1 envió el mensaje número 1
mqtt_client_0_1  | REPLICA_ID = 0
mqtt_client_2_1  | REPLICA_ID = 2
mqtt_client_2_1  | Cliente 2 envió el mensaje número 1
mqtt_client_0_1  | Cliente 0 envió el mensaje número 1
mqtt_client_3_1  | REPLICA_ID = 3
mqtt_client_3_1  | Cliente 3 envió el mensaje número 1
mqtt_client_4_1  | REPLICA_ID = 4
mqtt_client_4_1  | Cliente 4 envió el mensaje número 1
77 mqtt_client_0_1  | Cliente 0 envió el mensaje número 2
101 mqtt_client_0_1  | Cliente 0 envió el mensaje número 3
mqtt_client_1_1  | Cliente 1 envió el mensaje número 2
Valores de monitorización refrescados
133 mqtt_client_0_1  | Cliente 0 envió el mensaje número 4
Valores de monitorización refrescados
Valores de monitorización refrescados
mqtt_client_2_1  | Cliente 2 envió el mensaje número 2
149 mqtt_client_3_1  | Cliente 3 envió el mensaje número 2
177 mqtt_client_0_1  | Cliente 0 envió el mensaje número 5
mqtt_client_1_1  | Cliente 1 envió el mensaje número 3
203 mqtt_client_0_1  | Cliente 0 envió el mensaje número 6
Valores de monitorización refrescados
mqtt_client_4_1  | Cliente 4 envió el mensaje número 2
216 mqtt_client_0_1  | Cliente 0 envió el mensaje número 7
230 mqtt_client_1_1  | Cliente 1 envió el mensaje número 4
```

Ilustración 28: Salida estándar de la ejecución del escenario 1

4.2 Generación de dataset

En el siguiente apartado se tratará el caso de una generación de dataset simple, en el que un equipo con Kali Linux realiza un ataque de ICMP flooding sobre una máquina mientras que un sistema de detección de intrusos (IDS) realiza un *log* de dicho ataque.

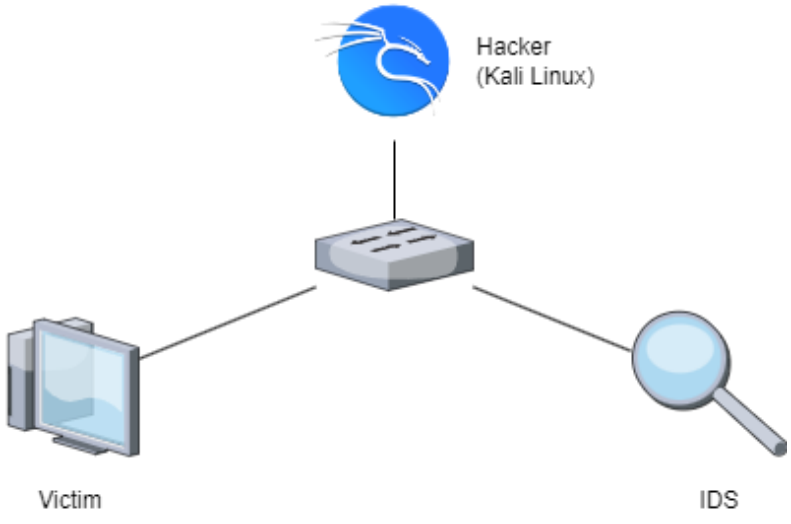

Ilustración 29: Escenario 2. Generación de dataset

En el presente ejemplo, se ha generado una imagen personalizada de Kali Linux que cumpla el perfil de Hacker (atacante) haciendo uso de la herramienta *hping3*, la cual permitirá realizar el ataque deseado sobre la máquina objetivo. El IDS detectará el ataque y generará un *log* con información de este una vez haya terminado.

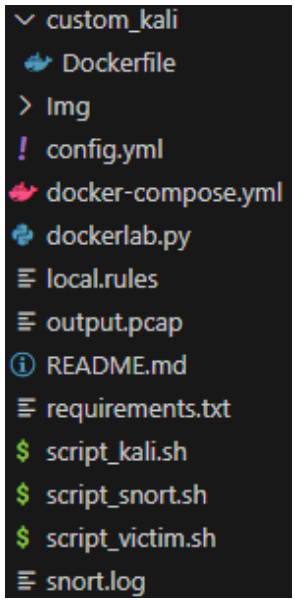

Ilustración 30: Árbol de ficheros empleado en el escenario 2

Para generar la imagen personalizada de Kali Linux, se ha empleado un ***Dockerfile*** con el contenido expuesto a continuación, el cual adapta la imagen oficial de Kali Linux instalándole la herramienta a utilizar en este ejemplo.

```
FROM kalilinux/kali-rolling
RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install hping3
```

Código 9: Dockerfile de la imagen de Kali Linux personalizada empleado en el caso de uso 2

Del mismo modo, para hacer uso de dicha herramienta, el script "***script_kali.sh***" empleará la siguiente orden:

```
#!/bin/sh
timeout 6 hping3 --rand-source victim -1 --faster
```

Código 10: Script ejecutado por el contenedor "Hacker" del caso de uso 2

El script que define el comportamiento de la víctima, "***script_victim.sh***", hará que esta esté en espera mientras recibe el ataque

```
#!/bin/bash
sleep infinite
```

Código 11: Script ejecutado por el contenedor "Víctima" del caso de uso 2

Por último, el script que ejecuta el IDS, "***script_snort.sh***", generará el log con información a cerca del ataque recibido. Su contenido es el siguiente:

```
#!/bin/bash

sleep 10

cd /workspace
echo "Running snort"
/home/snorty/snort3/bin/snort -c \
 /home/snorty/snort3/etc/snort/snort.lua -q --talos \
 -r output.pcap > snort.log
echo "Log generated at $PWD/snort.log"
```

Código 12: Script ejecutado por el contenedor "IDS" en el caso de uso 2

Se muestra, a continuación, una captura en la que se puede apreciar el consumo de recursos de cada uno de los dispositivos individuales. Cabe destacar que el contenedor "Hacker" hacer el mayor uso de recursos debido a que utiliza toda su capacidad para inundar la red de mensajes ICMP e impedir la comunicación entre ambos equipos.

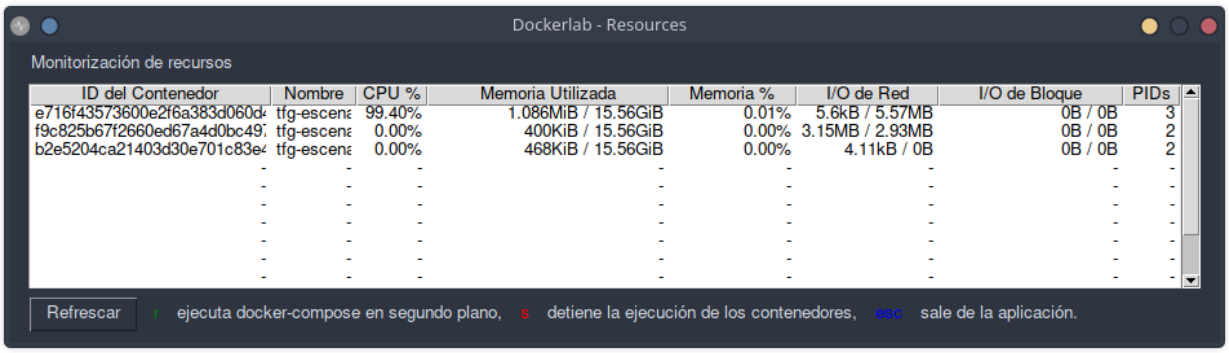

Ilustración 31: Utilización de recursos de cada uno de los contenedores del escenario 2

Una vez realizado el ataque, el contenedor "Hacker" detendrá su ejecución automáticamente y el mayor consumo de recursos pasará a ser por parte del IDS.

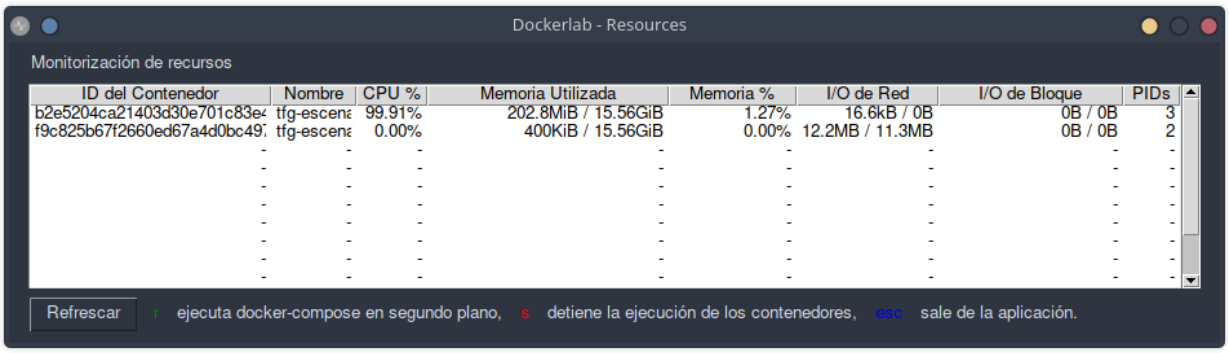

Ilustración 32: Utilización de recursos del IDS del escenario 2

La siguiente captura muestra el tráfico de red generado en la red. Es importante destacar que la víctima tiene la dirección IP 192.168.0.4. Del mismo modo, resulta interesante fijarse en el tiempo de emisión de cada uno de los paquetes mostrados, así como el hecho de que cada uno de ellos proviene de una dirección IP distinta y aleatoria.

No.	Time	Source	Destination	Protocol	Length	Info
11	0.394273783	143.240.247.222	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=0/0, ttl=64 (reply in 14)
14	0.394302756	192.168.0.4	143.240.247.222	ICMP	42	Echo (ping) reply id=0x0800, seq=0/0, ttl=64 (request in 11)
15	0.394381881	217.239.187.135	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=256/1, ttl=64 (reply in 16)
16	0.394392744	192.168.0.4	217.239.187.135	ICMP	42	Echo (ping) reply id=0x0800, seq=256/1, ttl=64 (request in 15)
17	0.394410407	194.109.160.229	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=512/2, ttl=64 (reply in 18)
18	0.394417943	192.168.0.4	194.109.160.229	ICMP	42	Echo (ping) reply id=0x0800, seq=512/2, ttl=64 (request in 17)
19	0.394431421	225.132.33.86	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=768/3, ttl=64 (no response found!)
20	0.394443828	196.183.254.0	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=1024/4, ttl=64 (reply in 21)
21	0.394450035	192.168.0.4	196.183.254.0	ICMP	42	Echo (ping) reply id=0x0800, seq=1024/4, ttl=64 (request in 20)
22	0.394462492	12.109.125.126	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=1280/5, ttl=64 (reply in 23)
23	0.394468236	192.168.0.4	12.109.125.126	ICMP	42	Echo (ping) reply id=0x0800, seq=1280/5, ttl=64 (request in 22)
24	0.394480040	219.109.113.56	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=1536/6, ttl=64 (reply in 25)
25	0.394487223	192.168.0.4	219.109.113.56	ICMP	42	Echo (ping) reply id=0x0800, seq=1536/6, ttl=64 (request in 24)
26	0.394500840	18.230.67.165	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=1792/7, ttl=64 (reply in 27)
27	0.394507686	192.168.0.4	18.230.67.165	ICMP	42	Echo (ping) reply id=0x0800, seq=1792/7, ttl=64 (request in 26)
28	0.394519787	113.253.56.129	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=2048/8, ttl=64 (reply in 29)
29	0.394526687	192.168.0.4	113.253.56.129	ICMP	42	Echo (ping) reply id=0x0800, seq=2048/8, ttl=64 (request in 28)
30	0.394539926	102.84.195.43	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=2304/9, ttl=64 (reply in 31)
31	0.394545667	192.168.0.4	102.84.195.43	ICMP	42	Echo (ping) reply id=0x0800, seq=2304/9, ttl=64 (request in 30)
32	0.394557239	240.80.50.151	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=2560/10, ttl=64 (reply in 33)
33	0.394563795	192.168.0.4	240.80.50.151	ICMP	42	Echo (ping) reply id=0x0800, seq=2560/10, ttl=64 (request in 32)
34	0.394575582	104.150.80.169	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=2816/11, ttl=64 (reply in 35)
35	0.394581829	192.168.0.4	104.150.80.169	ICMP	42	Echo (ping) reply id=0x0800, seq=2816/11, ttl=64 (request in 34)
36	0.394597194	200.183.118.230	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=3072/12, ttl=64 (reply in 37)
37	0.394604417	192.168.0.4	200.183.118.230	ICMP	42	Echo (ping) reply id=0x0800, seq=3072/12, ttl=64 (request in 36)
38	0.394616907	215.236.163.107	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=3328/13, ttl=64 (reply in 39)
39	0.394622548	192.168.0.4	215.236.163.107	ICMP	42	Echo (ping) reply id=0x0800, seq=3328/13, ttl=64 (request in 38)
40	0.394633986	241.142.236.84	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=3584/14, ttl=64 (reply in 41)
41	0.394641195	192.168.0.4	241.142.236.84	ICMP	42	Echo (ping) reply id=0x0800, seq=3584/14, ttl=64 (request in 40)
42	0.394653189	109.98.33.219	192.168.0.4	ICMP	42	Echo (ping) request id=0x0800, seq=3840/15, ttl=64 (reply in 43)
43	0.394660027	192.168.0.4	109.98.33.219	ICMP	42	Echo (ping) reply id=0x0800, seq=3840/15, ttl=64 (request in 42)

Ilustración 33: Captura del tráfico de red durante el ataque de ICMP flooding.

También resulta interesante fijarse en el archivo *log* generado por el IDS. A continuación, se muestra un fragmento de este en el que se observa el análisis realizado a los paquetes cursados por la red.

```
##### output.pcap #####
    [1:384:8] PROTOCOL-ICMP PING (alerts: 289445)
    [1:408:8] PROTOCOL-ICMP Echo Reply (alerts: 269002)
#####
-------------------------------------------------
rule profile (all, sorted by total_time)
#       gid   sid rev    checks matches alerts time (us) avg/check avg/match avg/non-match timeouts suspends
=       ===   === ===    ====== ======= ====== ========= ========= ========= ============= ======== ========
1         1 29456   2    558454       0      0    294445         0         0             0        0        0
2         1   384   8    558454  289445 289445    230500         0         0             0        0        0
3         1   408   8    558454  269002 269002    224796         0         0             0        0        0
4         1   417   8    558454       0      0    173167         0         0             0        0        0
5         1   441  10    558454       0      0    162016         0         0             0        0        0
6         1   386   8    558454       0      0    159834         0         0             0        0        0
7         1   423   8    558454       0      0    155716         0         0             0        0        0
8         1   413   8    558454       0      0    155590         0         0             0        0        0
9         1   449   9    558454       0      0    154814         0         0             0        0        0
10        1   439   9    558454       0      0    154521         0         0             0        0        0
11        1   411   8    558454       0      0    153511         0         0             0        0        0
12        1   415   8    558454       0      0    152954         0         0             0        0        0
13        1   445   8    558454       0      0    152552         0         0             0        0        0
14        1   429   9    558454       0      0    152218         0         0             0        0        0
15        1   443   9    558454       0      0    152019         0         0             0        0        0
16        1   458  12    558454       0      0    151879         0         0             0        0        0
17        1   462  12    558454       0      0    151773         0         0             0        0        0
18        1   460  12    558454       0      0    151447         0         0             0        0        0
19        1   453   8    558454       0      0    151033         0         0             0        0        0
20        1   419   8    558454       0      0    150982         0         0             0        0        0
21        1   401   9    558454       0      0    150877         0         0             0        0        0
22        1   451   8    558454       0      0    150654         0         0             0        0        0
```

Ilustración 34: Log generado por el IDS tras analizar el archivo de captura de tráfico de la red

En este caso en concreto, dockerlab encuentra su utilidad en la facilidad de monitorización de recursos y tráfico de red que proporciona, así como en la comodidad para el despliegue de la red. Se muestra a continuación una comparación entre el fichero de configuración empleado y el archivo docker-compose.yml generado.

config.yml

```yaml
lab:
  network: 192.168.0.0/24
  nodes:
    ids:
      image: ciscotalos/snort3
      script: script_snort.sh
      needs:
        - hacker
    hacker:
      build: custom_kali
      script: script_kali.sh
      needs:
        - victim
    victim:
      image: alpine:latest
      script: script_victim.sh
```

docker-compose.yml

```yaml
networks:
  lab_network:
    external: true
services:
  hacker:
    build: custom_kali
    depends_on:
    - victim
    entrypoint: /bin/sh /workspace/script_kali.sh
    networks:
      lab_network:
        ipv4_address: 192.168.0.3
    volumes:
    - ./:/workspace
  ids:
    depends_on:
    - hacker
    entrypoint: /bin/sh /workspace/script_snort.sh
    image: ciscotalos/snort3
    networks:
      lab_network:
        ipv4_address: 192.168.0.2
    volumes:
    - ./:/workspace
  victim:
    entrypoint: /bin/sh /workspace/script_victim.sh
    image: alpine:latest
    networks:
      lab_network:
        ipv4_address: 192.168.0.4
    volumes:
    - ./:/workspace
version: '3.3'
```

Ilustración 35: Comparación entre el archivo de configuración y el docker-compose.yml generados en el escenario 2.

5 CONCLUSIONES

El presente documento muestra el trabajo realizado para el diseño y desarrollo del software **dockerlab**, el cual permite al usuario levantar laboratorios virtuales haciendo uso de Docker Compose pero con una sintaxis más sencilla y herramientas de monitorización tanto de uso de recursos como del tráfico de red generado por los contenedores que se estén ejecutando. Se trata de una herramienta especialmente útil en el ámbito de la investigación y ayuda a facilitar el trabajo de puesta en marcha y *debugging* de sistemas complejos utilizando Docker.

Dockerlab ha demostrado ser una herramienta versátil, pudiendo ser utilizada para generar distintos tipos de redes cableadas, tal y como se expuso en el capítulo de "pruebas". Sus principales puntos fuertes son su sintaxis sencilla, su uso interactivo, las capacidades de monitorización de recursos y de tráfico de red y la reproducibilidad de los escenarios que genera. Sin embargo, las principales limitaciones de la herramienta son el hecho de no permitir ejecución en modo "*swarm*", las limitaciones existentes en la monitorización de recursos y las dificultades extra que pueden encontrarse si se desea emplear la herramienta mediante ssh o en otros sistemas operativos como MacOS.

En lo que respecta al espacio de mejora existente para el software, algunas características adicionales que podrían añadirse a la herramienta para hacerla más completa serían la posibilidad de simular topologías más complejas (con, por ejemplo, nodos que realicen tareas de enrutamiento), perfeccionar el sistema de *logs* para hacer el *debugging* más sencillo y probar más casos de uso, tales como generar datasets de tráfico HTTP o realizar pruebas de nuevos IDS.

En definitiva, se trata de un proyecto complejo con espacio para mejora para funciones adicionales, pero que ha resultado en una herramienta que resulta de gran ayuda a la hora de diseñar redes de contenedores y realizar pruebas sobre los mismos.

6 REFERENCIAS

[1] F. Soppelsa y K. Chanwit, Native Docker Clustering with Swarm, Birmingham, R.U.: Packt Publishing Ltd, 2016.

[2] M. Lukša, Kubernetes in Action, Simon and Schuster, 2017.

[3] S. Rampfl, «Network Simulation and its Limitations,» *Proceeding zum seminar future internet (FI), Innovative Internet Technologien und Mobilkommunikation (IITM) und autonomous communication networks (ACN),* vol. 57, p. 1, 2013.

[4] R. M. Hernández Quintero, J. A. Gil García, C. M. Torcatt García y F. Hernández, «Network Simulator - NS2,» La Victoria, 2009.

[5] T. Issariyakul y E. Hossain, Introduction to Network Simulator 2 (NS2), Boston, MA: Springer, 2008.

[6] G. F. Riley y T. R. Henderson, The ns-3 Network Simulator, Berlin: Springer-Verlag Berlin Heidelberg, 2010.

[7] J. C. Neumann, «Foreword,» de *The book of GNS3: Build virtual network labs using cisco, juniper and more*, San Francisco, CA, No Starch Press, Inc, 2015, pp. xiii-xiv.

[8] A. Lara, V. Mayor, R. Estepa, A. Estepa y J. Díaz-verdejo, «Smart home anomaly-based IDS: Architecture proposal and case study,» *Internet of Things; Engineering Cyber Physical Human Systems,* vol. 22, 2023.

7 ANEXO: CÓDIGO DEL SCRIPT DOCKERLAB

```
"""
El presente repositorio muestra una herramienta para automatizar el
despliegue de laboratorios virtuales mediante Docker.
Se trata de un Trabajo de Fin de Grado (TFG) desarrollado en el curso 2022/23
para el Grado en Ingeniería de las Tecnologías de
Telecomunicación de la Universidad de Sevilla.
"""

from yaml import load, dump

try:
    from yaml import CLoader as Loader, CDumper as Dumper
except ImportError:
    from yaml import Loader, Dumper
from colorama import Fore, Back
from typing import TypedDict, Optional
from ipaddress import ip_address, ip_network, IPv4Address, IPv4Network
import copy
import argparse
import subprocess
import shlex
from pynput import keyboard
from threading import Thread
import re
import PySimpleGUI as sg
import time
import json
import psutil


##############################
#  DEFINICIÓN DE TYPED DICTS  #
##############################
class Arguments(TypedDict):
    build: bool
    execute: bool
    monitor: bool
    usage: bool


class Compose(TypedDict):
    version: str
    # name:str
    services: dict
    networks: dict


class Service(TypedDict):
    build: Optional[str]
    image: Optional[str]
    entrypoint: Optional[str]
    depends_on: Optional[list]
    deploy: Optional[dict]
    environment: Optional[list]
    volumes: list
    networks: list


class Docker_Network_List(TypedDict):
    network_id_short: str
    name: str
    driver: str
    scope: str
```

```
class Docker_Network(TypedDict):
    Name: str
    Id: str
    Created: str
    Scope: str
    Driver: str
    EnableIPv6: bool
    IPAM: dict
    Internal: bool
    Attachable: bool
    Ingress: bool
    ConfigFrom: dict
    ConfigOnly: bool
    Containers: dict
    Options: dict
    Labels: dict


################################
#  DEFINICIÓN DE VAR GLOBALES #
################################
tshark_process: subprocess.Popen = None
continue_monitor: bool = True
compose_name: str = ""


################################
#  DEFINICIÓN DE EXCEPCIONES  #
################################
class ReaderException(Exception):
    def __init__(self, *args: object) -> None:
        super().__init__(*args)


class ParseNodeException(Exception):
    def __init__(self, *args: object) -> None:
        super().__init__(*args)


class IpAddrException(Exception):
    def __init__(self, *args: object) -> None:
        super().__init__(*args)
```

```
##############################
#   DEFINICIÓN DE FUNCIONES   #
##############################


def reader(config: dict, compose: Compose, *args, **conf) -> tuple:
    """
    Función "reader" que tomará como parámetro un diccionario que contenga
    la definición del laboratorio de "config.yml" y el diccionario mediante
    el cual se generará el docker-compose. Comprobará que sólo hay
    una clave definida en dicho diccionario y dividirá su contenido en dos
    diccionarios, "network" y "nodes". Guardará el nombre de la clave en el
    diccionario "compose".
    """
    network: IPv4Network = ip_network("10.0.0.0/8")
    nodes: dict = {}

    # Comprobamos que sólo hay una clave en "config"
    if len(config) != 1:
        raise (
            ReaderException(
                f"'config' contiene {len(config)} elementos. Sólo se admite 1."
            )
        )
    else:
        if conf["debug"]:
            print(
                f"{Fore.BLUE}Contenido del laboratorio: ”+
                  f"{config[list(config)[0]]}{Fore.RESET}"
            )
        # Comprobamos el contenido de "config", de momento,
        # sólo se admiten las cláusulas "network" y "nodes"
        lab = config[list(config)[0]]
        if len(lab) > 2:
            raise (
                ReaderException(
                    f"Se han definido más parámetros de los admitidos: {list(lab)}"
                )
            )
        else:
            global compose_name
            compose_name = list(config)[0]
            try:
                network = ip_network(lab["network"])
                print(f"{Fore.GREEN}\tRed [{network}] añadida{Fore.RESET}")
            except KeyError:
                if "debug" in conf and conf["debug"]:
                    print(
                        f"{Fore.BLUE}'network' no está definido. "+
                        f"Proporcionando el valor por defecto.{Fore.RESET}"
                    )
            try:
                nodes = lab["nodes"]
                for node in nodes:
                    print(f"{Fore.GREEN}\tNodo [{node}] añadido{Fore.RESET}")
            except KeyError:
                if "debug" in conf and conf["debug"]:
                    print(
                        f"{Fore.BLUE}'nodes' no está definido. "+
                        f"Proporcionando el valor por defecto.{Fore.RESET}"
                    )
    return (network, nodes)
```

```
def list_networks(is_debugging=False, *args, **conf) -> list[Docker_Network_List]:
    """
    Función que devuelve una lista con un resumen de cada una de
    las redes actualmente definidas por docker.
    """
    network_list: list[Docker_Network_List] = []
    header: bool = True
    with subprocess.Popen(
        shlex.split("docker network list"),
        stdout=subprocess.PIPE,
        universal_newlines=True,
    ) as p:
        for line in p.stdout:
            if not header:
                network: Docker_Network_List = {}
                split_line: list[str] = re.split(r"\s{2,}", line)
                network["network_id_short"] = split_line[0]
                network["name"] = split_line[1]
                network["driver"] = split_line[2]
                network["scope"] = split_line[3]
                network_list.append(network)
            header = False
        if is_debugging:
            print(
                f"{Fore.BLUE}Código de retorno (Network listing): "+
                f"{p.wait()}{Fore.RESET}"
            )
    return network_list

def get_network_data(
    docker_network: Docker_Network_List, is_debugging: bool = False, *args, **conf
    ) -> Docker_Network:
    """
    Función que, proporcionada la información resumida de una red, devuelve
    la información completa de la misma.
    """
    result: Docker_Network = None
    if is_debugging:
        print(
            f"{Fore.BLUE}Ejecutando 'docker network inspect"+
            f" {docker_network['name']}'...{Fore.RESET}"
        )
    with subprocess.Popen(
        shlex.split(f"docker network inspect {docker_network['name']}"),
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        universal_newlines=True,
    ) as p:
        stdout, stderr = p.communicate()
    # Verifica si hubo errores en la salida estándar
    if p.returncode == 0:
        # Intenta parsear la salida como JSON
        try:
            result = json.loads(stdout)
        except json.JSONDecodeError as e:
            print(f"{Fore.RED}Error al parsear la salida JSON:{e}{Fore.RESET}")
    else:
        print(f"{Fore.RED}Error al ejecutar el comando Docker: "+
              f"{stderr}{Fore.RESET}")
    if is_debugging:
        print(
            f"{Fore.BLUE}Código de retorno (Network specification): "+
            f"{p.wait()}{Fore.RESET}"
        )
    return result[0]
```

```
def create_network(network: IPv4Network, name: str, *args, **conf) -> bool:
    """
    Función que realiza los comandos necesarios para la creación de una red
    de docker en función de la red y el nombre proporcionados
    """
    network_created: bool = True

    with subprocess.Popen(
        shlex.split(
            "docker network create --driver=bridge "
            + "--opt com.docker.network.bridge.name=br-dockerlab "
            + "--opt com.docker.network.bridge.enable_icc=true "
            + "--opt com.docker.network.bridge.enable_ip_masquerade=true "
            + "--opt com.docker.network.bridge.host_binding_ipv4=0.0.0.0 "
            + f"--subnet {network} {name}"
        ),
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        universal_newlines=True,
    ) as p:
        for err in p.stderr:
            network_created = False
            if "network with name" not in err and "already exists" not in err:
                print(f"{Fore.RED}{err}{Fore.RESET}", end="")

        if "debug" in conf and conf["debug"]:
            print(f"Código de retorno (Network creation): {p.wait()}")
    return network_created


def generate_network(network: IPv4Network, compose: Compose, *args, **conf) -> None:
    """
    Función "generate_network", que genera la red que utilizaremos en
    nuestro compose.
    """
    is_debugging = False
    name = f"{compose_name}_network"
    compose["networks"] = {f"{name}": {"external": True}}  # "name":f"{name}",
    if "debug" in conf and conf["debug"]:
        print(f"{Fore.BLUE}compose['networks']={compose['networks']}{Fore.RESET}")
    if not create_network(network, name):
        # Red creada anteriormente o red con la misma IP
        if "debug" in conf and conf["debug"]:
            print(f"{Fore.BLUE}network, name = {network}, {name}{Fore.RESET}")
            is_debugging = True
        for i in list_networks(is_debugging):
            if "debug" in conf and conf["debug"]:
                print(f"{Fore.BLUE}i (list_networks)={i}{Fore.RESET}")
            docker_network: Docker_Network = get_network_data(i, is_debugging)
            if "debug" in conf and conf["debug"]:
                print(f"{Fore.BLUE}docker_network={docker_network}{Fore.RESET}")
            if (
                docker_network["Name"] not in {"none", "host", "bridge"}
                and len(docker_network["IPAM"]["Config"]) > 0
                and "Subnet" in docker_network["IPAM"]["Config"][0]
                and (
                   ip_network(docker_network["IPAM"]
                   ["Config"][0]["Subnet"]).subnet_of(
                        network
                    )
                    or ip_network(
                        docker_network["IPAM"]["Config"][0]["Subnet"]
                    ).supernet_of(network)
                )
            ) or docker_network["Name"] == name:
                with subprocess.Popen(
                    shlex.split(f"docker network remove {docker_network['Name']}"),
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
```

```
                    universal_newlines=True,
                ) as p:
                    for err in p.stderr:
                        print(f"{Fore.RED}{err}{Fore.RESET}", end="")
                if "debug" in conf and conf["debug"]:
                    print(f"Código de retorno (Network removal): {p.wait()}")

            create_network(network, name)

        if "debug" in conf and conf["debug"]:
            print(f"{Fore.BLUE}\tnetworks: {compose['networks']}{Fore.RESET}")


  def new_ip_addr(
      network: IPv4Network, ip_list: set[IPv4Address], n: int
  ) -> list[IPv4Address]:
      """
      Función "new_ip_addr". Toma como parámetros una red IPv4, una lista de
direcciones IP y un
      número de direcciones a obtener. Lanza una excepción en caso de que no hayan
suficientes
      direcciones disponibles en la subred.
      """

      result: list[IPv4Address] = []
      if n >= 1:
          for addr in network:
              if (
                  not (addr in ip_list)  # Si la IP no está en la lista
                  and addr.packed[3:] != b"\x00"  # Si la IP no acaba en '.0'
                  and addr.packed[3:] != b"\x01"  # Si la IP no acaba en '.1' (Host)
                  and addr.packed[3:] != b"\xff"
              ):  # Si la IP no acaba en '.255'
                  result.append(addr)
                  n -= 1
              if n < 1:
                  break
          if n > 1:
              raise (
                  IpAddrException(
                      "no hay suficientes direcciones IP disponibles en la "+
                      f"subred {network}"
                  )
              )
      return result
```

```
 def parse_node(
     nodes: dict, compose: Compose, network: IPv4Network, *args, **conf
     ) -> None:
     """
     Función "parse_node" que, para cada nodo,
     parseará su contenido en el diccionario "compose"
     """
     compose["services"] = {}
     ip_list = set()
     for node in nodes:
         case1, case2 = False, False
         replicas: int = 1

         if "build" in nodes[node]:
             case1 = True
         if "image" in nodes[node]:
             case2 = True

         if case1 and case2:
             raise ParseNodeException(
                 f"El nodo {node} contiene cláusulas 'build': "+
                 f"{nodes[node]['build']} e 'image':{nodes[node]['image']}"
             )
         elif not (case1 or case2):
             raise ParseNodeException("El nodo no contiene cláusulas "+
                                      "'build' ni 'image'")
         else:
             service: Service = {}
             service_replica: list[Service] = []

             if case1:  # Caso 1: contiene "build"
                 service["build"] = nodes[node]["build"]
             elif case2:  # Caso 2: contiene "image"
                 service["image"] = nodes[node]["image"]

             service["volumes"] = ["./:/workspace"]
             if "needs" in nodes[node]:
                 service["depends_on"] = nodes[node]["needs"]
             if "script" in nodes[node]:
                 service["entrypoint"] =
                    f'/bin/sh /workspace/{nodes[node]["script"]}'


##############################################################################
             # Existen varias opciones posibles:
             #   1. El nodo está replicado y tiene una IP asignada -> ✗ ERROR
             #   2. El nodo está replicado y NO tiene una IP asignada:
             #       2.1. El nodo tiene una subred definida:
             #           2.1.1. La subred pertenece al rango del laboratorio ->
             #             ✓ Se le asignan a los nodos IPs en dicho rango
             #           2.1.2. La subred NO pertenece al rango del laboratorio ->
             #             ✗ ERROR
             #       2.2. El nodo no tiene una subred definida ->
             #           ✓ Se le asignan a los nodos IPs en el rango del laboratorio
             #   3. El nodo NO está replicado y tiene una IP asignada:
             #       3.1. La IP pertenece al rango del laboratorio:
             #           3.1.1. La IP está repetida -> ✗ ERROR
             #           3.1.2. La IP NO está repetida -> ✓ Se le asigna dicha IP
             #       3.2. La IP NO pertenece al rango del laboratorio ->✗ ERROR
             #   4. El nodo NO está replicado y NO tiene una IP asignada ->
             #       ✓ Se le asigna una IP en el rango del laboratorio
             # A la hora de asignar direcciones IP, SIEMPRE se verificará que
             # queden suficientes direcciones disponibles en el rango en cuestión,
             # en caso contrario, se lanzará una excepción.

##############################################################################
```

```
            if "replicas" in nodes[node]:
                replicas = nodes[node][
                    "replicas"
                ]  # Si tiene más de una réplica, se crean varios nodos similares
                for i in range(replicas):
                    service_replica.append(copy.deepcopy(service))
            else:
                replicas = 1
            if "debug" in conf and conf["debug"]:
                print(
                    f"{Fore.BLUE}\nEl nodo {node} tiene "+
                    f"{replicas} réplica(s){Fore.RESET}"
                )
            if replicas > 1:
                if "ip" in nodes[node]:  # [1.]
                    raise ParseNodeException(
                        f"No se puede replicar un nodo al que se le ha asignado"+
                        f" IP: {node}"
                    )
                else:  # [2.]
                    if "network" in nodes[node]:  # [2.1.]
                        node_network: IPv4Network =
                           ip_network(nodes[node]["network"])
                        if node_network.subnet_of(network):  # [2.1.1.]
                            ips: list[IPv4Address] = new_ip_addr(
                                node_network, ip_list, replicas
                            )
                            for i in range(replicas):
                                ip_list.add(ips[i])
                                service_replica[i]["networks"] = {
                                    list(compose["networks"])[0]: {
                                        "ipv4_address": f"{ips[i]}"
                                    }
                                }
                                service_replica[i]["environment"] =
                                   [f"REPLICA_ID={i}"]

                        else:  # [2.1.2.]
                            raise ParseNodeException(
                                f"La subred {node_network} no pertenece a la "+
                                f"red del laboratorio ({network})"
                            )

                    else:  # [2.2.]
                        ips: list[IPv4Address] = new_ip_addr(network,
                                                             ip_list, replicas)
                        for i in range(replicas):
                            ip_list.add(ips[i])
                            service_replica[i]["networks"] = {
                                list(compose["networks"])[0]: {
                                    "ipv4_address": f"{ips[i]}"
                                }
                            }
                            service_replica[i]["environment"] = [f"REPLICA_ID={i}"]

            elif "ip" in nodes[node]:  # [3.]
                ip: IPv4Address = ip_address(nodes[node]["ip"])
                if ip in ip_list:  # [3.1.1.]
                    raise ParseNodeException(f"La ip {ip} está repetida")
                elif not (ip in network):  # [3.2.]
                    raise ParseNodeException(
                        f"La ip {ip} no está contenida en el rango {network}"
                    )
                else:  # [3.1.2.]
                    ip_list.add(ip)
                    service["networks"] = {
                        list(compose["networks"])[0]: {"ipv4_address": f"{ip}"}
                    }
            else:  # [4.]
```

```
                ip: IPv4Address = new_ip_addr(network, ip_list, 1)[0]
                ip_list.add(ip)
                service["networks"] = {
                    list(compose["networks"])[0]: {"ipv4_address": f"{ip}"}
                }

            if replicas == 1:
                compose["services"][node] = service
                if "debug" in conf and conf["debug"]:
                    print(f"{Fore.BLUE}\tservice "+
                          f"'{node}':\n\t\t{service}{Fore.RESET}")
            else:
                for i in range(replicas):
                    compose["services"][f"{node}_{i}"] = service_replica[i]
                    if "debug" in conf and conf["debug"]:
                        print(
                            f"{Fore.BLUE}\tservice "+
                            f"'{node}_{i}':\n\t\t{service_replica[i]}{Fore.RESET}"
                        )


def read_output(proc) -> None:
    """
    Función 'read_output', que imprime por pantalla la salida estándar
    de un subproceso y su código de retorno al finalizar.
    """
    for line in proc.stdout:
        print(line, end="")
    # espera a que el subproceso termine y obtiene su código de retorno
    return_code = proc.wait()
    # imprime el código de retorno del subproceso
    print(f"Código de retorno (read output): {return_code}")


def stop_compose() -> None:
    with subprocess.Popen(
        shlex.split("docker-compose stop"),
        stdout=subprocess.PIPE,
        universal_newlines=True,
    ) as p:
        t = Thread(target=read_output, args=(p,))
        t.start()
```

```
def read_monitor_output(proc: subprocess.Popen, content: list[list]) -> None:
    """
    Función read_monitor_output. Actualiza el contenido de una lista bidimensional
    'content' con el output del proceso de monitorización 'proc' en un
    instante determinado.
    """
    header: bool = True
    result: list[list] = []
    for line in proc.stdout:
        current_stats: list = ["-", "-", "-", "-", "-", "-", "-", "-"]
        if header:
            header = False
        else:
            split_line: list[str] = re.split(r"\s{2,}", line)
            if len(split_line) == 8:
                for i in range(len(split_line)):
                    current_stats[i] = split_line[i].strip()

            else:
                print("La línea contiene", len(split_line), "parámetros:")
                for i in split_line:
                    print("\t", i)
                print()
            result.append(current_stats)

    for i in range(len(content)):
        if i < len(result):
            content[i] = result[i]
        else:
            content[i] = ["-", "-", "-", "-", "-", "-", "-", "-"]

    return_code = proc.wait()
    # imprime el código de retorno del subproceso en caso de error
    if return_code != 0:
        print(
            f"Proceso 'read_monitor_output' finalizado con código: "+
            f"{Fore.RED}{return_code}{Fore.RESET}"
        )
    else:
        print(f"{Fore.BLUE}Valores de monitorización refrescados{Fore.RESET}")


def itera_monitor_output(content: list[list]) -> None:
    """
    Función itera_monitor_output, que actualiza el contenido de una lista
    bidimensional 'content' con las estadísticas de los contenedores actualmente
    en ejecución en el equipo cada segundo.
    """
    while continue_monitor:
        process: subprocess.Popen = subprocess.Popen(
            shlex.split(
                'docker stats --no-trunc --no-stream --format "table
                {{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t
                {{.MemPerc}}\t{{.NetIO}}\t{{.BlockIO}}\t{{.PIDs}}"'
            ),
            stdout=subprocess.PIPE,
            universal_newlines=True,
        )
        Thread(
            target=read_monitor_output,
            args=(
                process,
                content,
            ),
        ).start()
        time.sleep(1)
```

```
def interfaz_monitor(max_containers: int) -> None:
    """
    Función interfaz_monitor que mostrará la interfaz gráfica de monitorización del
    uso de recursos.
      Toma como único parámetro el número máximo de contenedores que monitorizar.
    """
    global continue_monitor
    sg.theme("Default 1")
    content = []
    for i in range(max_containers):
        content.append(["-", "-", "-", "-", "-", "-", "-", "-"])

    Thread(target=itera_monitor_output, args=(content,), daemon=True).start()

    layout = [
        [
            sg.Text("Monitorización de recursos"),
            sg.Text(text_color="red", key="-WARNING-"),
        ],
        [
            sg.Table(
                content,
                [
                    "ID del Contenedor",
                    "Nombre",
                    "CPU %",
                    "Memoria Utilizada",
                    "Memoria %",
                    "I/O de Red",
                    "I/O de Bloque",
                    "PIDs",
                ],
                key="-TABLE-",
            )
        ],
        [
            sg.Button("Refrescar"),
            sg.Text("r", text_color="green"),
            sg.Text("ejecuta docker-compose en segundo plano,"),
            sg.Text("s", text_color="red"),
            sg.Text("detiene la ejecución de los contenedores,"),
            sg.Text("esc", text_color="blue"),
            sg.Text("sale de la aplicación."),
        ],
    ]
    window = sg.Window("Dockerlab - Resources", layout,
                       icon="./Img/monitor_icon.png")

    while continue_monitor:
        event, values = window.read()

        if event == sg.WIN_CLOSED or event == "Exit" or event == None:
            continue_monitor = False
            break
        window["-TABLE-"].update(content)
        if content[-1:][0] != ["-", "-", "-", "-", "-", "-", "-", "-"]:
            window["-WARNING-"].update(
                "ADVERTENCIA: algunos contenedores pueden no estar siendo mostrados"
            )
        else:
            window["-WARNING-"].update("")
    window.close()
```

```
def monitoriza_red(network: IPv4Network) -> None:
    """
    Función que monitoriza el táfico de una red que se le pasa por argumentos.
    La interfaz del equipo conectada a dicha red debe tener una dirección
    IPv4 asociada que acabe en .1.
    El resultado de la monitorización se almacenará en un archivo `output.pcap`.
    """
    global tshark_process
    print(network, "->", network[1])
    if_name: str = None
    process: subprocess.Popen = None

    addrs = psutil.net_if_addrs()
    for i, j in addrs.items():
        try:
            if ip_address(j[0].address) == network[1]:
                if_name = i
        except ValueError as e:  # No hay dirección IP en esta interfaz
            pass
            # if_name=i

    if if_name is not None:
        print(f"{Fore.GREEN}Comienza la monitorización!{Fore.RESET}")
        tshark_process = subprocess.Popen(
            shlex.split(f"tshark -i {if_name} -w output.pcap"),
            stdout=subprocess.PIPE,
            universal_newlines=True,
        )


################################
#      SCRIPT PRINCIPAL        #
################################
def dockerlab(
    debug: bool = False,
    flags: Arguments = {
        "build": True,
        "execute": True,
        "monitor": False,
        "execute": False,
    },
) -> None:
    # Caso por defecto
    if (
        not (flags["build"])
        and not (flags["execute"])
        and not (flags["monitor"])
        and not (flags["execute"])
    ):
        flags["build"] = True
        flags["execute"] = True

    if flags["build"]:
        file = open("./config.yml", "r")
        compose_file = open("./docker-compose.yml", "w")
        config: dict = load(file, Loader=Loader)
        compose: Compose = {}
        compose["version"] = "3.3"
        try:
            (network, nodes) = reader(config, compose, debug=debug)
            print("Se ha leído config.yml correctamente")
            if debug:
                print(
                    f"{Fore.BLUE}\tNetwork:\t{network}"+
                    f"\n\tNodes:\t{nodes}{Fore.RESET}"
                )
            generate_network(network, compose, debug=debug)
            print("Red implementada correctamente.")
            parse_node(nodes, compose, network, debug=debug)
```

```
            dump(compose, compose_file)
            print(
                f"{Fore.GREEN}'docker-compose.yml' generado"+
                f" correctamente.{Fore.RESET}"
            )
        except KeyError as e:
            print(
                f"{Fore.RED}KeyError: No existe el parámetro {e} en "+
                f"config.yml{Fore.RESET}"
            )
        except ReaderException as e:
            print(f"{Fore.RED}Ha habido un problema en la lectura: "+
                  f"\n\t{e}{Fore.RESET}")
        except ParseNodeException as e:
            print(
                f"{Fore.RED}Ha habido un problema en el parseo de elementos: "+
                f"\n\t{e}{Fore.RESET}"
            )
        except Exception as e:
            print(f"{Fore.RED}Ha ocurrido un error desconocido: "+
                  f"\n\t{e}{Fore.RESET}")
        finally:
            file.close()
            compose_file.close()

    if flags["execute"]:
        # Hacemos pull y build
        print(f"{Fore.GREEN}Haciendo pull a los contenedores...{Fore.RESET}")
        read_output(
            subprocess.Popen(
                shlex.split("docker-compose pull"),
                stdout=subprocess.PIPE,
                universal_newlines=True,
            )
        )
        print(f"{Fore.GREEN}¡Pull realizado correctamente!{Fore.RESET}")
        print(f"{Fore.GREEN}Construyendo contenedores...{Fore.RESET}")
        read_output(
            subprocess.Popen(
                shlex.split("docker-compose build"),
                stdout=subprocess.PIPE,
                universal_newlines=True,
            )
        )
        print(f"{Fore.GREEN}¡Contenedores construidos "+
              f"satisfactoriamente!{Fore.RESET}")

        if flags["monitor"]:
            network_name: str = ""
            found: bool = False

            with open("./docker-compose.yml", "r") as compose_file:
                network_name = list(
                    load(compose_file, Loader=Loader)["networks"].keys()
                )[0]

            for i in list_networks():
                docker_network: Docker_Network = get_network_data(i)

                if (
                    docker_network["Name"] == network_name
                    and len(docker_network["IPAM"]["Config"]) > 0
                    and "Subnet" in docker_network["IPAM"]["Config"][0]
                ):
                    found = True
                    Thread(
                        target=monitoriza_red,
                        args=(
                            IPv4Network(
```

```
                                docker_network["IPAM"]["Config"][0]["Subnet"]
                            ),
                        ),
                        daemon=True,
                    ).start()
            if not found:
                print(f"{Fore.RED}La red {network} no ha sido "+
                      f"encontrada.{Fore.RESET}")

        if flags["usage"]:
            Thread(
                target=interfaz_monitor, args=(len(compose["services"]) + 10,)
            ).start()

        running: bool = False
        with keyboard.Events() as events:
            print(
                f"Pulse '{Fore.BLACK}{Back.WHITE}r{Fore.RESET}{Back.RESET}' "+
                f"para ejecutar docker-compose en segundo plano"
                + f" y '{Fore.BLACK}{Back.WHITE}s{Fore.RESET}{Back.RESET}' "+
                f"para parar su ejecución."
                + f" Para salir de la aplicación, pulse la "+
                f"tecla '{Fore.BLACK}{Back.WHITE}esc{Fore.RESET}{Back.RESET}'."
            )
            for event in events:
                global continue_monitor
                if f"{event.key}" == "'r'" and not running:
                    print("Corriendo el subproceso 'docker-compose'...")
                    running = True

                    print("Creando subproceso docker-compose...")
                    process_compose: subprocess.Popen = subprocess.Popen(
                        shlex.split(
                            "docker-compose up --remove-orphans --force-recreate"
                        ),
                        stdout=subprocess.PIPE,
                        universal_newlines=True,
                    )
                    print(f"{Fore.GREEN}Subproceso creado"+
                          f" correctamente!{Fore.RESET}")
                    t = Thread(target=read_output, args=(process_compose,))
                    t.start()

                elif f"{event.key}" == "'s'" and running:
                    running = False
                    print(f"{Fore.GREEN}Parando el subproceso...{Fore.RESET}")
                    stop_compose()

                elif event.key == keyboard.Key.esc:
                    if running:
                        print(f"{Fore.GREEN}Parando el subproceso...{Fore.RESET}")
                        stop_compose()
                    if flags["monitor"] and tshark_process is not None:
                        tshark_process.kill()
                    if flags["usage"] and continue_monitor:
                        continue_monitor = False
                    break

    elif flags["monitor"] or flags["usage"]:
        print(
            f"{Fore.RED}Las opciones 'monitor' (-m) y 'usage' (-u) "+
            f"deben ir acompañadas por la opción 'execute' (-e).{Fore.RESET}"
        )
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Herramienta para el despliegue de laboratorios virtuales "+
                    "mediante Docker. Por defecto, en caso de no proporcionar"+
                    " parámetros, se ejecutará con las banderas -be."
    )
    for i in [
        [
            "-b",
            "--build",
            "Indica que se desea generar el archivo docker-compose.yml. Si sólo "+
            "se selecciona esta opción, no se crearán los "+
            "contenedores pertinentes.",
        ],
        [
            "-e",
            "--execute",
            "Indica que se desean crear y levantar los contenedores definidos"+
            " en docker-compose.yml.",
        ],
        [
            "-m",
            "--monitor",
            "Monitoriza el tráfico de paquetes en la red simulada. "+
            "Debe usarse junto con -e.",
        ],
        [
            "-u",
            "--usage",
            "Monitoriza el uso de recursos dentro de los contenedores "+
            "de la simulación. Debe usarse junto con -e.",
        ],
    ]:
        parser.add_argument(i[0], i[1], action="store_true", help=i[2])
    flags: Arguments = vars(parser.parse_args())
    dockerlab(debug=False, flags=flags)
```