

ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

**PROTOTIPO DE NETWORKING MEDIANTE DOCKER EN UNA
RASPBERRY**

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR
EN REDES Y TELECOMUNICACIONES**

ERICK DAMIAN TEPAN SILVA

DIRECTOR: ING. JAVIER ALEJANDRO ARMAS NAVARRETE

DMQ, septiembre 2022

CERTIFICACIONES

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



Erick Damian Tepan Silva

erick.tepan@epn.edu.ec

erickt.2016damerck@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Erick Damián Tepán Silva, bajo mi supervisión.

Javier Alejandro Armas Navarrete
DIRECTOR

javier.armas@epn.edu.ec

DECLARACIÓN DE AUTORÍA

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

Erick Tepán

DEDICATORIA

El presente trabajo está siendo dedicado para todas aquellas personas que confiaron en mí y me apoyaron. Especialmente para mis padres, los cuales estuvieron en los momentos más difíciles y alegres durante el transcurso de mi carrera. También todo el sacrificio realizado en el trabajo, en las asignaturas pasadas como la felicidad de aprobarlas son dedicadas hacia mí ya que estas han sido aquellas que durante todo este transcurso de tiempo se han llevado gran parte del mismo, con las cuales me han servido para ingresar al ambiente laboral y profesional.

Durante toda mi formación académica han existido tanto personas como ocasiones las cuales se han opuesto en mi superación personal como profesional, más sin embargo estas no han sido motivos para rendirme, este trabajo y esfuerzo son dedicadas para todas aquellas personas que no confiaron en mis habilidades como en las oportunidades que me está presentando la vida y que hoy en día a servido para lograr todo lo propuesto y todo lo que un día aspiré.

Se la dedico de igual manera a todos mis amigos, hermanos, compañeros y excompañeros que con su ayuda, ocurrencias, preocupaciones y la toma de decisiones que muchas veces fueron erróneas hoy en día nos encontramos en las mismas circunstancias y que siempre que estábamos reunidos nos decíamos que confiábamos en todos y cada uno de nosotros, hoy en día, lo dedicamos a nuestros familiares y al esfuerzo de todos nosotros.

AGRADECIMIENTO

En primer lugar, el agradecimiento más grande que tengo es a Dios, a mis padres en segundo lugar ya que ellos fueron los que incentivaron en mí las ganas de superarme y con los cuales aprendí que con esfuerzo, dedicación y concentración se puede llegar a conseguir grandes cosas, a cumplir las metas y objetivos que en anterioridad solo eran sueños, hoy en día agradezco la paciencia que me han tenido como la forma en la cual me han entendido ya sea en el cambio de humor como en las acciones que he tomado.

Mi familia en general se lleva gran parte de mi agradecimiento ya que con su preocupación como con su constante indagación de cómo me va en la universidad como la de saber si alguna vez necesito ayuda, tanto directa como indirectamente han sido gran parte de mi motivación. Existen amigos y amigas los cuales han demostrado con sencillez y honestidad lo importante que pueden llegar a ser, demostrando la felicidad y los cuales compartían mi felicidad y por ello les estoy hoy en día agradecido.

Finalmente, y no menos importante doy gracias a las personas que se han cruzado en mi camino: amigos, conocidos, profesores e ingenieros los cuales compartían sus conocimientos como sus experiencias los cuales me han ayudado en más de una ocasión a abrir los ojos para entender que la acción o acciones que eh estado realizando son incorrectas y las eh corregido a tiempo.

No paro de decírmelo y de auto agradecerme ya que he mostrado un carácter positivo y alegre frente a ocasiones que no todos hubieran afrontado, me agradezco de una manera que me motivo pensando en ello y mi forma de demostrarlo es siendo un buen profesional.

ÍNDICE DE CONTENIDOS

CERTIFICACIONES.....	¡Error! Marcador no definido.
DECLARACIÓN DE AUTORÍA	¡Error! Marcador no definido.
DEDICATORIA	¡Error! Marcador no definido.
AGRADECIMIENTO.....	¡Error! Marcador no definido.
ÍNDICE DE CONTENIDO.....	¡Error! Marcador no definido.
RESUMEN.....	¡Error! Marcador no definido.
ABSTRACT.....	¡Error! Marcador no definido.
1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO.....	i
1.1 Objetivo general	ii
1.2 Objetivos específicos	ii
1.3 Alcance.....	ii
1.4 Marco Teórico.....	ii
Networking.....	iii
2 METODOLOGÍA.....	vii
3 RESULTADOS	vii
3.1 Analizar Docker y sus características de Networking.	viii
3.2 Diseñar la sección de Networking en Docker mediante el manejo de un servidor Samba.....	xii
4 CONCLUSIONES.....	xxviii
5 RECOMENDACIONES	xxx
6 REFERENCIAS BIBLIOGRÁFICAS	¡Error! Marcador no definido.
7 Trabajos citados	¡Error! Marcador no definido.
8 ANEXOS.....	¡Error! Marcador no definido.
ANEXO I :	xxxiii

RESUMEN

El proyecto consiste en hacer una red de recursos compartidos entre dispositivos dentro de una red, los cuales funcionan en contenedores que serán levantados en *Docker* y este funcionará dentro de un dispositivo electrónico denominado *Raspberry*.

En el primer capítulo, se tiene una introducción de lo que contiene el documento, en donde, se plantea el objetivo general y los objetivos específicos enfocados para la implementación del proyecto. Se presentan los conceptos fundamentales de los elementos a utilizar para el desarrollo del proyecto, como las herramientas de *networking*, las características y elementos de *Docker*. También, se muestran especificaciones de cómo usar y habilitar tanto el *Raspberry* como el servidor implementado.

En la segunda sección se tiene toda la descripción del proyecto, en donde estará detallado cuál fue el procedimiento de análisis para escoger las características adecuadas y así cumplir con los objetivos solicitados, abarcando cada uno y detallando lo más importante y relevante en el desarrollo.

La tercera sección contiene los resultados obtenidos. Se describe detalladamente los parámetros escogidos y necesarios para establecer la comunicación entre el servidor y los clientes, es decir, el *driver*, el archivo, los apartados y las comunicaciones internas, las cuales, realizadas en conjunto, hacen una red de dispositivos (*networking*) que se pueden comunicar entre sí. También, se presentan los resultados y las pruebas de funcionamiento del servidor y los documentos dentro de este.

En el cuarto capítulo se encuentran las conclusiones y recomendaciones que se obtuvieron durante el transcurso del desarrollo del proyecto, que ayudan para un futuro desarrollo del mismo.

Finalmente se presentan los anexos, como el video de funcionamiento del prototipo.

PALABRAS CLAVE: *networking*, archivos, *raspberry*, *Docker*, *Samba*.

ABSTRACT

The project consists of making a network of shared resources between devices within a network, which work in containers that will be raised in Docker and this will work within an electronic device called Raspberry.

In the first section, there is an introduction of what the document contains, where the general objective and the specific objectives focused on the implementation of the project are presented. The fundamental concepts of the elements to be used for the development of the project are presented, such as the networking tools, the characteristics and elements of Docker. Also, specifications of how to use and enable both the Raspberry and the implemented server are shown.

The second section contains the description of the project, where it will be detailed the procedure to choose the features and thus meet the requested objectives, covering each one and detailing the most important and relevant in the development.

The third section contains the results obtained. It describes in detail the parameters chosen and necessary to establish the communication between the server and the clients, that is to say, the driver, the file, the sections and the internal communications, which, carried out together, make a network of devices (networking) that can communicate with each other. Also, the results and tests of the server and the documents within it are presented.

In the fourth chapter are the conclusions and recommendations that were obtained during the course of the development of the project, which help for a future development of the same one.

KEYWORDS: *networking, files, Raspberry, Docker, Samba.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El *networking* en redes es una relación entre dispositivos activos (ordenadores, teléfonos inteligentes, otros dispositivos y usuarios), los cuales se buscan vincular con el propósito de compartir datos. Esta funcionalidad hace que hoy en día, en varias empresas se utilice un servicio con el cual podamos acceder a medios de manera remota siempre que se encuentren en la red, ya sea por medios cableados o inalámbricos. Además, el administrador del servidor puede permitir o denegar que un usuario acceda a la información compartida.

La idea del proyecto se da por su funcionalidad en el área empresarial, teniendo en cuenta que dentro de una empresa varios usuarios pueden utilizar un mismo archivo o documento y, en el caso de que una empresa sea grande, la manera más sencilla es accediendo a un servidor que cuente con los archivos necesarios y que dicho servicio sea rápido, fácil de implementar y seguro, tanto para la empresa como para los usuarios. Una opción que cumple con las características antes mencionadas es el servidor *Samba*.

Con este proyecto se busca crear una herramienta de *networking* por medio de un servidor *Samba*, el cual estará funcionando dentro de un contenedor *Docker* instalado en un *Raspberry*, permitiendo a los dispositivos acceder al servidor mediante su dirección IP. Esto, a su vez, logrará que cualquier dispositivo que se encuentre en la red del servidor *Samba* pueda acceder a él y a sus documentos compartidos.

Actualmente, la mayoría de actividades que realiza una persona involucran el uso de dispositivos inteligentes. En el ambiente laboral, el personal de una empresa necesita de un tipo de almacenamiento que permita encontrar, subir y descargar archivos de suma importancia. Es indispensable tener una característica de *networking*, la cual nos ayuda a la intercomunicación tanto de usuarios como de dispositivos.

1.1 Objetivo general

Desarrollo de un prototipo de *Networking* mediante *Docker* en una *Raspberry*.

1.2 Objetivos específicos

1. Analizar *Docker* y sus características de *Networking*.
2. Diseñar la sección de *Networking* en *Docker* mediante el manejo de un servidor *Samba*.
3. Implementar los contenedores en una *Raspberry*.
4. Verificar el funcionamiento de contenedores con la función de *Networking*.

1.3 Alcance

Por medio del presente proyecto se busca implementar *Docker networking* sobre una plataforma libre como es *Raspberry*. También se estudiará sobre los diferentes tipos de implementación de *Docker networking* que se puede realizar, se realizará el código de los diferentes tipos de implementaciones posibles de *Docker networking*. El contenedor se realizará con el servicio de *Samba*.

1.4 Marco Teórico

Raspberry Pi.

Es un mini PC de tamaño accesible a la mano y con un bajo precio en el mercado, se puede realizar actividades idénticas a la una PC normal, pero en pequeñas capacidades, es decir, tiene interfaces de entrada y de salida. Funciona por medio de un SO denominado Debian instalada en una tarjeta SD, es muy utilizada en el ámbito de la informática y la programación y siendo compatible para proveedores externos como *Docker*. [1].

Se puede utilizar como un ordenador portable y que permite realizar las funciones básicas del mismo como: ingresar a internet, navegar, uso de mensajería ya sea instantánea como correo electrónico y otras aplicaciones de un computador, pero en pequeñas características.

Networking

Networking en redes se basa en una red de dispositivos con el fin de compartir recursos, datos o información a personas específicas que se encuentren dentro de la red.

Es decir, una red de contactos que buscan transmitir u obtener una información específica, un punto de comunicación en donde los dispositivos pueden intercambiar información de manera segura y confiable.

Docker

Es un software de código abierto que es enfatizado el área de la automatización y despliegue de servicios o aplicaciones dentro de contenedores que virtualizan el sistema operativo de un servidor. Con *Docker* toda la implementación para que el servicio se desplegue correctamente están incluidas o almacenadas en la imagen las cuales pueden ser utilizadas en diferentes instancias en cualquier sistema, lo que hace que el contenedor con la aplicación se aisle [2].

En los contenedores no se proporcionará un sistema operativo completo, por lo que pueden ser iniciados como detenidos de manera muy rápida, lo que hace que no consuman tantos recursos del dispositivo y, por lo mismo, en la actualidad son muy utilizados en la industria por su sencillez y velocidad de poder modificar, iniciar o parar un contenedor con diferentes tipos de servicios [4].

La principal ventaja es que *Docker* ya tiene todos los requerimientos a excepción del sistema operativo, lo facilita la recopilación de software, es decir, no se debe crear un nuevo contenedor de alguna aplicación, simplemente utilizar alguna que ya fue creada por algún usuario antiguo, evitando utilizar una imagen que contenga alguna aplicación errónea o de prueba.

También, existe un registro o una base de datos denominado *Docker Hub* en donde se almacenan las distribuciones o imágenes de los contenedores con los servicios ya creados, los cuales pueden ser utilizados por otras personas siempre y cuando el autor las adjunte con las especificaciones realizadas y la etiqueta. Existe varias imágenes, por lo que encontrar las características o especificaciones de alguna aplicación es muy importante para ser descargada y utilizada por otros usuarios que tengan una cuenta en *Docker Hub* [4].

Funcionamiento de una red Docker.

Docker File: Es un archivo que contiene las instrucciones que se ejecutarán con los procesos del contenedor (creará la imagen *Docker*) [5].

Docker Image: Son todas las instrucciones utilizadas para crear *Docker Containers*, código que se ejecutarán internamente.

Docker volumes: Son los mecanismos que se utilizarán para guardar o conservar los datos que se están registrando dentro de los contenedores, siendo esta una de las mejores opciones para que los datos persistan [4].

Docker Container: Son todas aquellas dependencias que se ejecutan de una aplicación. Es la imagen creada con anterioridad pero que se encuentra en ejecución.

Comandos para ejecutar un contenedor en *Docker*.

Docker ps: Comando con el cual podemos observar los contenedores existentes, el tiempo que ha sido creado, el id que tiene el contenedor y el nombre con el que está siendo creado [6].

Docker stop: Con el apartado de *stop* detenemos el contenedor

Docker rm: Con *rm* se puede eliminar el contenedor, siempre y cuando este haya sido detenido con anterioridad. Para realizar estas actividades es necesario especificar el contenedor con su el ID perteneciente que los verificamos con *Docker ps* [6].

Docker network: Comando que se utiliza con una serie de caracteres para observar, inspeccionar y crear redes con diferentes tipos de *drivers*.

Tipos de controladores de red

Driver: Es el utilizado por defecto, siendo la mejor solución.

Host: Elimina el aislamiento entre los contenedores y el anfitrión.

Overlay: Permite realizar la conexión de diferentes contenedores en diferentes nodos.

Macvlan: Asigna las MAC a un contenedor, IP lo que permite utilizarlo como una PC más.

None: inhabilita las redes [5].

Docker-compose

Es una herramienta en la cual permite definir mediante una serie de instrucciones las aplicaciones o los servicios (contenedores), el almacenamiento (volúmenes), las variables (contraseñas, usuarios) y redes (con los diferentes *drivers*) [3].

Docker-compose creará las redes con los *drivers* si estas no existen. También regenerará los contenedores que han sido modificados dentro del archivo *.yml* en donde será creado el *docker-compose*.

- Un archivo *.yml* es un código estándar (lenguaje de marcado) de una serie de datos que es muy amigable y compatible con los lenguajes de programación existentes.

Esta herramienta viene instalada por defecto al momento de instalar *Docker*. El comando es necesario para levantar los archivos y ponerlos en marcha.

docker-compose up -d tiene estas especificaciones para ejecutar, levantar los contenedores y continuar ejecutando otros comandos. Esto permite salir del contenedor sin que este llegue a detenerse o fallar.

docker-compose -f docker-compose.yml permite correr un fichero específico de *docker-compose*.

Servidor Samba

Es un espacio de aplicaciones, el cual usa un protocolo muy importante y antiguo de red *SMB (Server Message Block)*, el cual está basado en el modelo cliente-servidor y puede compartir información (archivos, directorios, impresoras) entre diferentes nodos. *Samba* soluciona y permite el uso de *SMB* en *Linux* y *Unix*, lo que permite una comunicación multiplataforma [3]. Entre sus características están:

- Permite compartir varios archivos.
- Permite compartir dispositivos previamente instalados (impresoras).
- Permite la navegación en la red.
- Autentifica a usuarios que ingresan a un dominio, ya sea *Linux*, *Windows* o *Unix*.
- Permite que el servidor *Linux* actúe como un controlador de un dominio.

Samba contiene programas que brindan servicios como: **nmbd** y **smbd**, denominados demonios, que ayudarán a la compartición de los respectivos recursos [4].

Nmbd es un demonio, un servidor que brinda funciones de WINS (servicio de nombres para *Windows*) y proporciona direcciones IP cuando este presenta alguna petición [4].

Smbd es un demonio que controla toda la información o recursos compartidos entre el servidor y los respectivos usuarios, va a ser el que se encarga de la autenticación, bloqueo y compartición de los mismos mediante SMB [3].

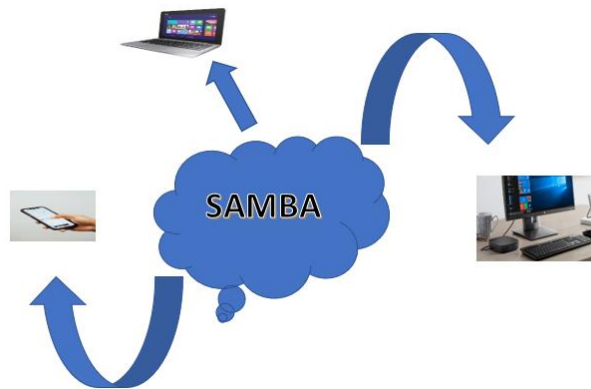


Figura 1.1 Servidor *Samba*

Este servicio permite compartir recursos por medio de la red, los cuales pueden ser administrados por usuarios que tengan los respectivos permisos [8].

Networking en Docker

Es una función que fue creada para que los contenedores se comuniquen entre sí, o para dar alguna especificación a algún contenedor. Permite crear redes que no han sido creadas con anterioridad y modificarlas de manera sencilla. Existen maneras para realizar el *networking* en donde los controladores cumplen funciones importantes, ya que estos darán especificaciones relevantes para los servicios que se quieren implementar [5].

Networking y *Samba*, conjuntamente, permiten comunicar dispositivos de la misma red y acceder a un centro de recursos y carpetas compartidas.

2 METODOLOGÍA

La realización del presente proyecto se realizó en cuatro etapas enfocadas en los objetivos. Cada una de las etapas fueron concretadas mediante un análisis previo, es decir, se especifica los parámetros utilizados y los comparamos con los objetivos.

Inicialmente, se realizó una investigación sobre los manejos, funcionalidades y características de los contenedores en *Docker*, en este caso, se analizó la forma más sencilla para el levantamiento de un contenedor con un servidor *Samba* y la manera en cómo dicho contenedor se va a comunicar con los diferentes dispositivos de la red.

Una vez entendida la parte del levantamiento y funcionalidad del contenedor, se diseñó el apartado de *networking* en *Docker*, analizando los drivers existentes, las características que tiene cada uno y se escogió el adecuado al momento de ser implementado internamente en el servidor *Samba*, ya que tiene las características necesarias para realizar una red de recursos compartidos como archivos e impresoras. También, se agregó un usuario y contraseña para el ingreso al servidor, además de los respectivos permisos en las carpetas existentes.

Luego, se procedió a implementar los contenedores en el *Raspberry* con una serie de procesos: primero, se creó un archivo con la imagen del servidor y los comandos internos, luego, se guardó el archivo y finalmente, se levantó el contenedor con la ayuda de una función específica.

A continuación, se ingresó al contenedor, a su red conectada y se observó por medio de comandos que las características existentes sean las indicadas previamente. Finalmente, se verificó el funcionamiento del prototipo y la conectividad de los dispositivos de la red con el servidor mediante un *ping*, y también, se accedió al servidor para verificar el acceso a los archivos y carpetas del mismo.

3 RESULTADOS

El prototipo de *networking* mediante *Docker* en una *Raspberry* va a interconectar diferentes dispositivos que están dentro de la red mediante un servidor *Samba*. El funcionamiento del prototipo está centrado en el levantamiento de contenedores con la herramienta *Docker* y al implementarlo, internamente constará de datos los cuales podrán ser vistos por los dispositivos de la red.

Las características con las que cuenta el servidor, trabajan de la siguiente forma: primero se crea y se levanta el contenedor. A continuación, se ingresa hacia el servidor mediante su dirección IP, su usuario y contraseña, se observa las carpetas compartidas con las que cuenta el servidor y se procede a acceder a las mismas para observar los archivos existentes que se tienen acceso.

Para la creación de este prototipo, se desarrolló varias etapas principales: primero se instaló el sistema operativo del *Raspberry*, luego se instaló *Dockery* se analizaron sus respectivas características con las que va a trabajar el prototipo. Una vez que se analizó y se escogió la herramienta con la que se levantó el contenedor se creó el apartado de *networking* con su *driver* específico. Finalmente se agregó al servidor a nuestra red interna, se lo levantó y se lo colocó en un lugar seguro que no se pueda manipular fácilmente.

3.1 Analizar *Docker* y sus características de *Networking*.

Una de la funcionalidad de *Docker* es que permite realizar actividades de virtualización similares a las máquinas virtuales, en donde cada servicio, así como sus dependencias para que esta funcione se almacenan en la imagen con su tag, lo que permite que el contenedor con el servicio empleado se pueda mover y ejecutarse desde cualquier parte de manera sencilla [9].

Para el levantamiento del contenedor en *Docker*, se escogió a **docker-compose**, herramienta que viene por defecto instalada en el momento que se instala *Docker*, esta permite realizar una serie de instrucciones para que los contenedores se levanten y realicen actividades específicas dentro de los mismos, correr comandos internamente e incluso llegar a crear datos y redes sin la necesidad de la intervención humana. Esto facilitó en la implementación del contenedor debido a que no es necesario seguir una serie de comandos para levantar correctamente el contenedor, sino, simplemente ingresar al archivo *docker-compose* guardarlo y levantarlo. Cave recalcar que es más sencillo encontrar errores o fallas en el archivo interno de *docker-compose* ya que es un código de programación y también es más sencillo ponerlo en marcha con un simple comando. [3].

Como *docker-compose* utiliza la ayuda de archivos *.yml*, dentro de este archivo se realizaron las configuraciones requeridas para el servicio, sin embargo, la indentación y la forma en cómo se establece los servicios deben estar perfectamente equilibradas con su formato y las versiones, ya que dependiendo de la versión este tendrá un tipo de indentación distinto a otro [5].

Al utilizar docker-compose las modificaciones que se realicen dentro del archivo pueden ser asignadas a un contenedor específico y los cambios afectaran solo a él. Al levantarlo nuevamente este se encargará de regenerarlo con las modificaciones hechas de manera automática.

Compose puede ser utilizado conjunto a *dockerfile* en donde los comandos o procesos para una imagen se van a procesar uno detrás de otro hasta ejecutarla y tener un contenedor en *Docker*.

Los volúmenes de *Docker* es lo que se utiliza para que los datos que se utilicen en el contenedor se conserven en el sistema de archivos del host, es decir que realiza de una carpeta compartida en una máquina virtual por lo que los contenedores pueden compartir un volumen, esta función como otras fueron añadidas en el archivo *docker-compose.yml*.

Networking en Docker

Esta es una característica principal y fundamental de *Docker* ya que permite que los contenedores y todos los servicios implementados puedan llegar a tener comunicación entre sí y con otros dispositivos.

Los controladores existentes tienen funcionalidades específicas y cada uno de estos van a funcionar de diferente forma, lo que hace que funcione correctamente en algunos servicios implementados y en otros no.

Para poder implementarlos o correrlos los podemos realizar mediante un **Docker run** especificando que controlador se desea usar en un servicio o por medio de **docker-compose** en donde en el apartado de *network* se especifica el *driver* a utilizar.

A continuación, se presentan los diferentes *drivers* existentes en *Docker* con sus características analizadas.

Bridge: Usado comúnmente por defecto lo que es realizar un puente, generalmente cuando las aplicaciones se están ejecutando de manera independiente llegando a comunicarse entre sí [8].

Host: Realiza una transferencia de puertos del ordenador hacia el contenedor es decir que usará la misma dirección IP de la máquina, usados para contenedores independientes eliminando la conexión entre el contenedor y el *host* de *Docker*.

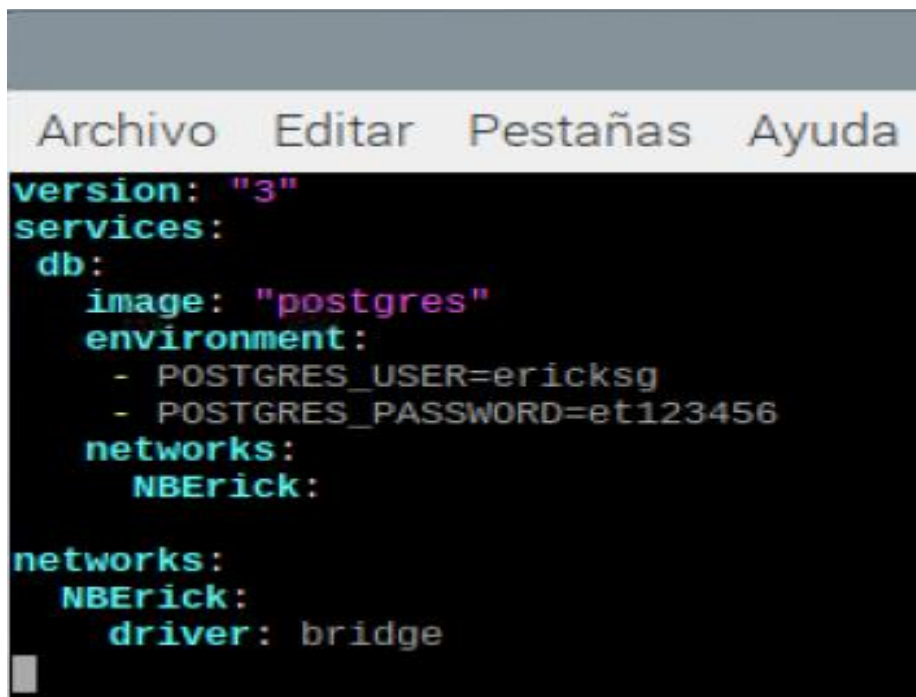
Overlay: Es la que crea las redes virtuales en los contenedores es decir que se puede tener varios contenedores en diferentes nodos, pero manteniendo la misma red entre ellos, lo que nos permite eliminar el enrutamiento a nivel de sistema operativo [9].

Macvlan: Va a asignar una dirección física a un contenedor es decir tendrá MAC, IP lo que hará que el contenedor se comporte como otra máquina adicional dentro de la red, es utilizado cuando se va a utilizar aplicaciones heredadas (desactualizadas) que van a ser conectadas a la red física en vez de ser enrutadas por medio del *host* de *Docker* [8].

None: Va a desactivar el *networking* del contenedor es decir deshabilitar todas las redes usado conjunto a un controlador de red.

Para realizar la funcionalidad de *networking* se pueden utilizar, *none*, *host*, *macvlan*, *overlay* y *bridge*. Sin embargo, para este caso, en el presente proyecto se busca que el servidor tenga una dirección IP estática para que se pueda acceder directamente y que el DHCP de la red no le dé una que pueda variar en cada instante que se active el contenedor, también debe tener una dirección MAC para que pueda funcionar como una máquina o un servidor físico [9]. Con estas especificaciones se escoge el *driver macvlan* con el cual podemos hacer que los dispositivos se conecten al servidor como si fuera una interfaz física con su respectiva puerta de enlace y a la red a la cual pertenece [8].

EJEMPLOS DE LOS DRIVERS CON APLICACIONES O SERVICIOS DISTINTOS.



```
Archivo  Editar  Pestañas  Ayuda
version: "3"
services:
  db:
    image: "postgres"
    environment:
      - POSTGRES_USER=ericksg
      - POSTGRES_PASSWORD=et123456
    networks:
      NBERick:
networks:
  NBERick:
    driver: bridge
```

Figura 3.1 *Driver bridge*

```
ericksg@raspberrypi:~/bridgeErick $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
ab48e0983529       bridge              bridge              local
669e4cc6c5b1       bridgeerick_NBErick bridge              local
7a3aa33471ec       bridgeerick_default bridge              local
```

Figura 3.2 Red con *driver bridge*

```
ericksg@raspberrypi:~ $ docker run -it --name appE2 --network none alpine sh
```

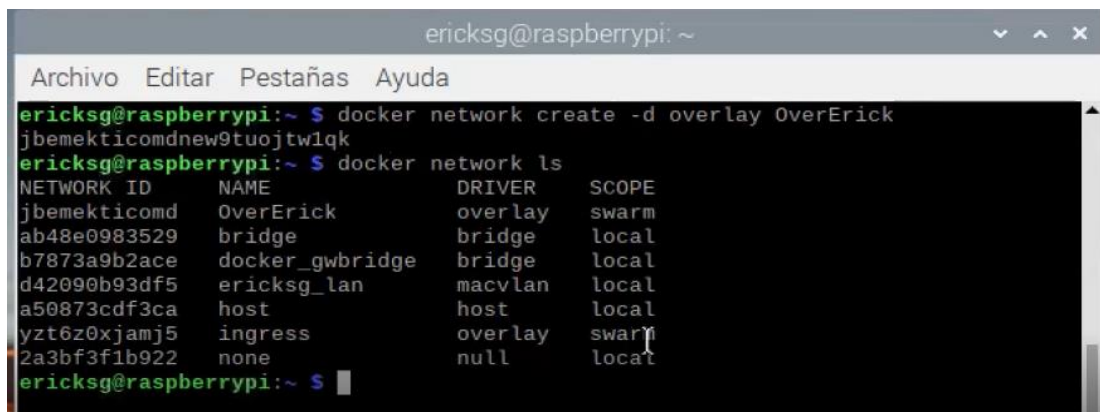
Figura 3.3 *Driver none*

```
ericksg@raspberrypi: ~
Archivo  Editar  Pestañas  Ayuda
ericksg@raspberrypi:~ $ docker run -it --name appE2 --network none alpine sh
/ # ping google.com
ping: bad address 'google.com'
/ #
```

Figura 3.4 Funcionamiento de *none*

```
version: "3.2"
services:
  proxy:
    image: jwilder/nginx-proxy
    container_name: ErickH
    labels:
      com.github.jrcs.letsencrypt_nginx_proxy_companion.nginx_proxy: "true"
    ports:
      - "80:80"
      - "443:443"
    networks:
      ErickH:
networks:
  ErickH:
    driver: host
```

Figura 3.5 Contenedor con *Driver host*



```
ericksg@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
ericksg@raspberrypi:~ $ docker network create -d overlay OverErick  
jbeemkticomdnew9tuojtw1qk  
ericksg@raspberrypi:~ $ docker network ls  
NETWORK ID          NAME                DRIVER              SCOPE  
jbeemkticomd        OverErick           overlay             swarm  
ab48e0983529        bridge             bridge             local  
b7873a9b2ace        docker_gwbridge    bridge             local  
d42090b93df5        ericksg_lan        macvlan            local  
a50873cdf3ca        host               host               local  
yzt6z0xjamj5        ingress            overlay            swarm  
2a3bf3f1b922        none              null               local  
ericksg@raspberrypi:~ $
```

Figura 3.6 Red con driver host

Al analizar se comprendió que la funcionalidad de un servidor *Samba* va de la mano con nuestro *driver*, estos fueron utilizados con funcionalidad de conexión y distribución de archivos, por lo mismo el utilizar otro tipo de *driver* que no sea *macvlan* hace que el servicio no funcione correctamente en la red e incluso no poder ser utilizada [5].

3.2 Diseñar la sección de *Networking* en *Docker* mediante el manejo de un servidor *Samba*.

Para la sección de *networking* se creó las líneas en el archivo de *docker-compose* y se definieron los parámetros en el arvhivo *.yaml*.

En las líneas con características de *networking*, para el servidor están siendo especificadas tanto el nombre, la red a la cual pertenece, la dirección IP estática, la IP del *gateway*, así como el *driver* utilizado (existen características que se les asigna o se crea por defecto). Además, se creó las carpetas que serán visualizadas en el servidor y el volumen en donde se almacenan los diferentes documentos.

Debido a la funcionalidad de *Docker* y las características de *Samba*, se procedió a realizar dicha sección, ayudándonos de la función de *docker hub* que es un repositorio en la nube en donde se pueden encontrar imágenes las cuales se pueden aprovechar, como *Samba*. En dicha nube se escogió el servidor el cual contiene características ya necesarias instaladas previamente por algún otro autor, esta al ser llamada y agregada en el archivo en una de sus versiones se procedió a dar comandos de ayuda como que se reinicie siempre que nuestro *Raspberry* se desconecte y se vuelva a conectar, otros comandos como el nombre y la contraseña de acceso hacia el servidor y finalmente

comandos que creen carpetas directamente para que se presenten en el servidor con los permisos que van a tener siempre y cuando se ingresen con el usuario y contraseña.

```
version: "2"

services:

  samba:
    image: dperson/samba:rpi
    restart: always
    command: '-u "ericksg;et123456" -s "media;/media;yes;no" -s "downloads;/downloads;yes;no" -s "archivos;/archivos;yes;no" -s "erick;/erick;yes;no" -s "seguridad;/seguridad;yes;no" -s "redes;/redes;yes;no" -s "archivos_prueba;/archivos_prueba;yes;no" -s "funcionamiento;/funcionamiento;yes;no" -s "docker_user;/docker_user;yes;no"'
    stdin_open: true
    tty: true
```

Figura 3.7 Imagen y comandos implementados

Exportación de los puertos

En esta parte de la sección se redireccionan los puertos, es decir que se reservan dichos puertos en el servicio de *Docker* y el servidor *Samba*.

El puerto 139 es utilizado para compartir archivos e impresoras y se expondrá para que los paquetes lleguen en el mismo orden en cómo se envían y que garantizó la comunicación ya que es TCP.

Finalmente, el puerto 445 es smb el cual es utilizado para ofrecer el acceso a las carpetas compartidas y otras comunicaciones entre nodos que se expone para cumplir con la misma función del puerto 139 en cuanto al orden.

```
ports:
  - 139:139
  - 445:445
```

Figura 3.8 Puertos expuestos

Creación del almacenamiento los volúmenes.

En dicha sección se creó el lugar en donde se almacenará los datos del contenedor y por ende los datos que se tienen dentro de las carpetas.

Con el implemento de dicha sección se facilitó la transferencia y observación de los datos existentes en el contenedor. Dichos almacenamientos se encuentran ubicados en el directorio *home* y por ende dentro del usuario maestro del servidor *Samba*. Los archivos se encuentran en las carpetas que cuentan con un almacenamiento similar a su nombre

```
volumes:
- /home/ericksg/media:/media
- /home/ericksg/downloads:/downloads
- /home/ericksg/archivos:/archivos
- /home/ericksg/erick:/erick
- /home/ericksg/seguridad:/seguridad
- /home/ericksg/redes:/redes
- /home/ericksg/archivos_prueba:/archivos_prueba
- /home/ericksg/funcionamiento:/funcionamiento
- /home/ericksg/docker_user:/docker_user
```

Figura 3.9 Almacenamientos creados

Creación de nuestra red y su función de *Networking*.

En dicha sección se dividió en 2 partes: la primera es asignar el nombre de la red a la cual pertenece el servidor y especificar una dirección IP estática. La dirección asignada fue para que todos los dispositivos que estén dentro de la red puedan ingresar al servidor de manera sencilla, es decir, cada vez que un dispositivo conozca la dirección del servidor y tenga comunicación con él puede ingresar en este y observar su contenido por medio de la dirección IP 192.168.0.6.

La segunda parte es la red principal, la red que fue creada para el *networking* original, es decir, la red en la que fue implementando el servicio que tiene como dirección IP 192.168.0.0/24 y su respectiva puerta de enlace *gateway* 192.168.0.1, con la que se comunicará con los demás dispositivos.

Todo el apartado de *networking* fue configurado para ser usado con *macvlan*, y la manera en que se realiza la conexión se refleja en el apartado de *ethernet*, es decir, que

el *Raspberry* está siendo conectado por ethernet (eth0) y fue por ese medio de comunicación cableado por donde se transmitió información al *gateway*.

```
networks:
  lan:
    ipv4_address: 192.168.0.6

networks:
  lan:
    driver: macvlan
    driver_opts:
      parent: eth0
    ipam:
      config:
        - subnet: "192.168.0.0/24"
          gateway: "192.168.0.1"
```

Figura 3.10 Dirección y red del Servidor

Estructura del prototipo.

En esta sección se añade la versión que tiene el archivo interno *docker-compose* (es importante ya que esto afecta de manera importante en la estructura como se está formando el archivo *.yaml*).

Finalmente se añadió las estructuras anteriores como la imagen, los comandos, volúmenes, puertos y *networking* anteriormente desarrolladas, se verificó que todo esté correctamente y se lo guardo.

Comando para guardar: **:wq**

3.3 Implementar los contenedores en una Raspberry.

Para implementar los contenedores dentro de una *Raspberry* primero es necesario realizar una investigación previa para la instalación del sistema operativo de la misma (*Raspberry pi OS/Raspbian*) [10].

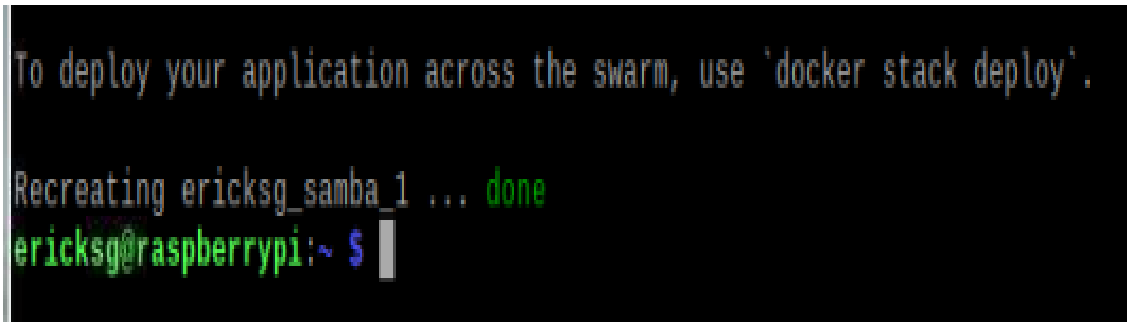
- Se descargó el SO en la micro SD con ayuda de una computadora, una vez finalizada se extrae y se insertó la micro SD.
- Se instaló el SO y seleccionamos el idioma que será utilizado.

- Se procedió a la instalación de *Docker* por lo que fue necesario instalar previamente unas características como los paquetes de los repositorios.
- Se actualizó por medio de comandos e instalamos *Docker*.
- Se agregó una serie de comandos con las *keys* de repositorios e instalamos los *kernels-headers*.
- Se agregó el usuario con el que iniciaremos *Docker*.

Finalmente, culminado con las especificaciones del servidor *Samba* en el archivo *docker-compose*, se salió del archivo y se levantó al contenedor. Se verificó que el contenedor esté funcionando correctamente en *Docker*, por ende, en el *Raspberry* y realizamos las inspecciones adecuadas del contenedor [10].

Para la implementación de los contenedores se llama a nuestro archivo *.yaml* de *docker-compose* y lo corremos, con el comando:

Docker-compose up -d



```
To deploy your application across the swarm, use 'docker stack deploy'.
Recreating ericksg_samba_1 ... done
ericksg@raspberrypi:~ $
```

Figura 3.11 Levantamiento del contenedor

El parámetro de *-d* en el comando hace que el contenedor siga corriendo incluso ya saliendo de él sin que este presente alguna falla.

Si el archivo tiene todo correcto y las características antes mencionadas están con la indentación y parámetros adecuados, correrá el contenedor y a su lado presentará la palabra *done* y este podrá ser verificado con el comando:

Docker ps

```
ericksg@raspberrypi:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
NAMES
033a1168861e  dperson/samba:rpi  "samba.sh -u ericksh..."  5 minutes ago  Up 5 minutes
ericksg_samba_1
```

Figura 3.12 Contenedor corriendo

En la línea de código emergente se puede observar la ID, la imagen que está corriendo en el contenedor, comandos que son utilizados, el tiempo que fue creado, el estado actual del contenedor ya sea, *up* o *stop*, y el nombre con el que está siendo definido el contenedor.

Como el apartado de *status* se encuentra en *up*, nos indicó que el contenedor se levantó correctamente por lo que ahora podemos ingresar y verificar el funcionamiento.

3.4 Verificar el funcionamiento del contenedor con la función de *Networking*.

Una vez Instalados, el sistema operativo (*Raspbian*), *Docker*, *Samba* y agregada las funciones de *networking*, se realiza las pruebas de funcionamiento que garanticen que el prototipo funciona correctamente y las pruebas contribuyeron a corregir errores que se presentaron al momento de verificar su funcionalidad.

Para esta sección, para realizar el control y verificación del funcionamiento de nuestro prototipo verificaremos los pasos anteriores:

Primero, la Instalación del sistema operativo *Raspbian* en la micro SD y por ende en del dispositivo *Raspberry* [10].

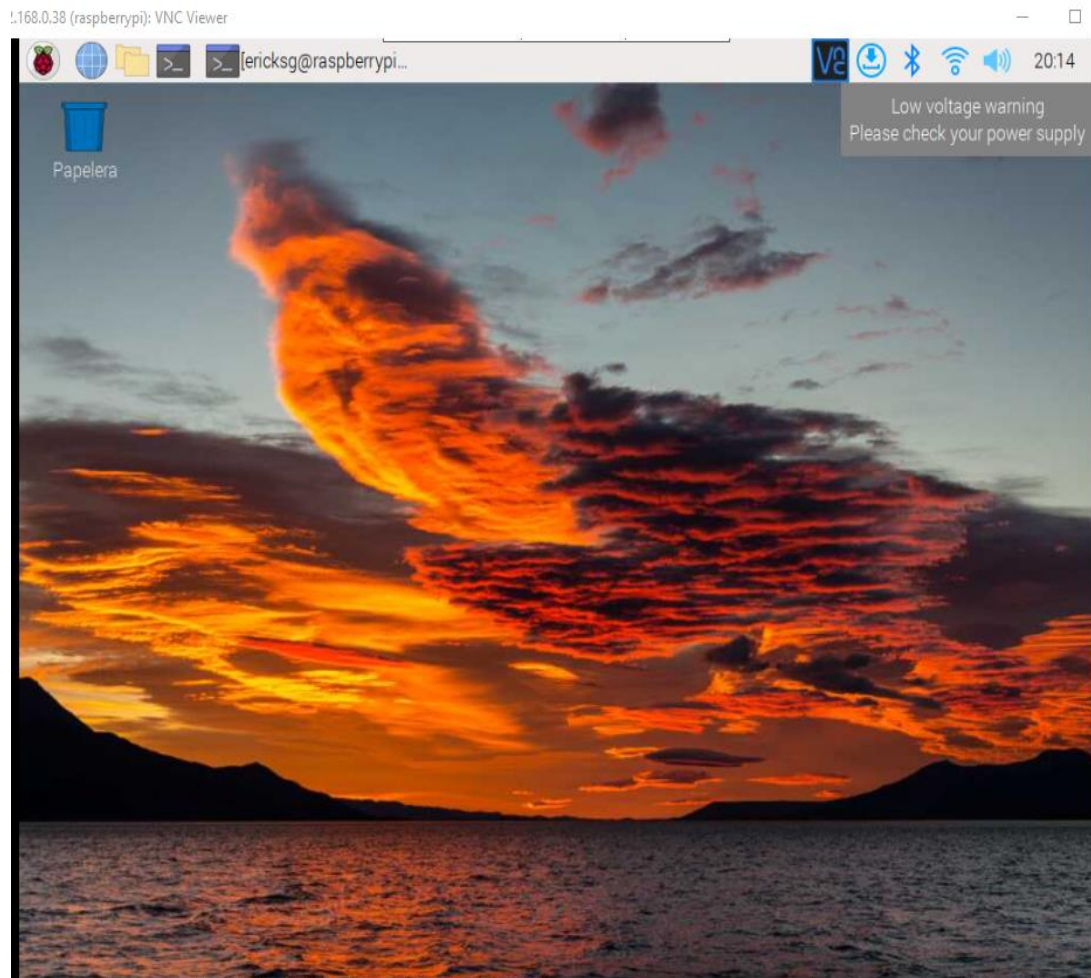


Figura 3.13 Sistema operativo *Raspbian* en *Raspberry*

Como el SO del *Raspberry* fue instalado correctamente se procede a verificar la instalación de *Docker* y su versión, lo verificamos mediante el comando. [6]

docker version.

```
ericksg@raspberrypi:~ $ docker version
Client: Docker Engine - Community
 Version:           20.10.16
 API version:       1.41
 Go version:        go1.17.10
 Git commit:        aa7e414
 Built:             Thu May 12 09:17:16 2022
 OS/Arch:           linux/arm
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:           20.10.16
  API version:       1.41 (minimum version 1.12)
  Go version:        go1.17.10
  Git commit:        f756502
  Built:             Thu May 12 09:15:21 2022
  OS/Arch:           linux/arm
  Experimental:      false
 containerd:
  Version:           1.6.4
  GitCommit:         212e8b6fa2f44b9c21b2798135fc6fb7c53
 runc:
  Version:           1.1.1
  GitCommit:         v1.1.1-0-g52de29d
 docker-init:
  Version:           0.19.0
  GitCommit:         de40ad0
ericksg@raspberrypi:~ $ █
```

Figura 3.14 Versión de *Docker*

Una vez que se visualizó la versión de *Docker*, se procede a llamar al archivo *docker-compose* y a levantarlo como en la **Figura 3.11** y observamos los contenedores existentes, como en la **Figura 3.12**.

Ahora una vez que fue instalado *Docker*, y que el servidor *Samba* haya sido levantado por medio de *docker-compose*, se ingresa de manera iterativa al contenedor.

Comando: **docker exec -it contenedor bin/bash** [8]

Una vez ingresado al contenedor se procedió a verificar la versión del servidor con el comando

smbstatus. [6]

```
ericksg@raspberrypi:~$ docker exec -it d1890167abff bin/bash
bash-5.1# smbstatus

Samba version 4.13.7
PID      Username   Group      Machine                                     Protocol Version Encryption
-----
171      nobody    nobody     192.168.0.8 (ipv4:192.168.0.8:9771)       SMB3_11      -

Service  pid      Machine   Connected at                               Encryption  Signing
-----
archivos 171      192.168.0.8 Thu Jun 30 22:05:28 2022 -03      -          -

Locked files:
Pid      User(ID)  DenyMode  Access   R/W     Oplock      SharePath  Name  Time
-----
171      100      DENY_NONE 0x100081 RONLY   NONE        /archivos  .    Thu Jun 30 22:0
171      100      DENY_NONE 0x100081 RONLY   NONE        /archivos  .    Thu Jun 30 22:0
171      100      DENY_NONE 0x100080 RONLY   NONE        /archivos  .    Thu Jun 30 22:0

bash-5.1#
```

Figura 3.15 Versión del servidor *Samba*

Una vez que fue verificado que el contenedor está funcionando con el servicio adecuado, que está activo, se realizó la comprobación de la red, es decir que nuestro servidor conste con: el nombre de la red que le fue asignado y el *driver* el cual se está utilizando, mediante.

Comando: **docker network ls**.

```
ericksg@raspberrypi:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
jbemekticomd       OverErick           overlay             swarm
dc920f4085b5       bridge             bridge             local
b7873a9b2ace       docker_gwbridge     bridge             local
d42090b93df5       ericksg_lan        macvlan            local
a50872edf2ea       host                host               local
```

Figura 3.16 Red del Servidor

Se observa que la red con *driver macvlan* posee una ID a la cual ingresaremos para inspeccionarla si es necesaria, se puede observar el nombre, el *driver* utilizado y se verifica que sean los correctos para que el servidor esté dentro de la red.

Al observar que todo lo anterior funciona correctamente, ingresamos a nuestra red y se la procedió a inspeccionarla y así observar el contenedor con el servicio que se encuentra dentro de ella, como algunos detalles extras, ya sea la dirección del servidor, la red a la que pertenece, el nombre de la red y nuevamente el *driver* utilizado.

```
cksg@raspberrypi:~ $ docker network inspect d42090b93df5

[
  {
    "Name": "ericksg_lan",
    "Id": "d42090b93df5ef67142c2492aae0a92964eefc088d4078dfef51c5b4e8901c32",
    "Created": "2022-06-08T13:05:16.185088468-05:00",
    "Scope": "local",
    "Driver": "macvlan",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "192.168.0.0/24",
          "Gateway": "192.168.0.1"
        }
      ]
    }
  }
],
```

Figura 3.17 Nombre de la red, *driver*, IP de la red y *gateway*

```

"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
  "Network": ""
},
"ConfigOnly": false,
"Containers": {
  "d1890167abffe0b981bd6fe9a91340f3a1685abd07a196c49a0dd8812ad12bbf": {
    "Name": "ericksg_samba_1",
    "EndpointID": "5c95c68487e2670c6057da7b959f348d41a168d7f986aa07c7c5",
    "MacAddress": "02:42:c0:a8:00:06",
    "IPv4Address": "192.168.0.6/24",
    "IPv6Address": ""
  }
},
"Options": {
  "parent": "eth0"
},
"Labels": {}

```

Figura 3.17 Servicio y dirección IP del servidor

Como se verificó que en la red creada está el servidor *Samba* con todas sus características, ahora se procede a salir de la inspección de la red y se ingresa al contenedor nuevamente y para verificar la dirección IP de nuestro contenedor (servidor *Samba*) con el comando

ifconfig.

```

root@55e7f647cd66:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.6 netmask 255.255.255.0 broadcast 192.168.0.255
    ether 02:42:c0:a8:00:06 txqueuelen 0 (Ethernet)
    RX packets 11936 bytes 17686210 (16.8 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 6175 bytes 409356 (399.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 1178 (1.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 1178 (1.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figura 3.18 Dirección IP del servidor

Una vez que se ingresó al contenedor y corroborando que la dirección IP sea la misma que se ha definido con anterioridad se realiza las pruebas de funcionamiento del servidor.

Como se encuentra dentro del contenedor de forma iterativa se procede a realizar un *ping* a los diferentes dispositivos que estén en la misma de la red que constan con diferentes direcciones IP dadas por el DHCP de la red, así como también un *ping* a nuestra puerta de enlace (*gateway* 192.168.0.1)

```
root@55e7f647cd66:/# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.83 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=0.587 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=0.634 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=64 time=0.651 ms
64 bytes from 192.168.0.1: icmp_seq=5 ttl=64 time=0.590 ms
64 bytes from 192.168.0.1: icmp_seq=6 ttl=64 time=0.608 ms
```

Figura 3.19 Conectividad a la puerta de enlace (*gateway*)

Luego se verificó la conexión del servidor y de un dispositivo dentro de la red haciendo *ping* hacia el servidor.

```
C:\Users\erick>ping 192.168.0.6

Haciendo ping a 192.168.0.6 con 32 bytes de datos:
Respuesta desde 192.168.0.6: bytes=32 tiempo=128ms TTL=64
Respuesta desde 192.168.0.6: bytes=32 tiempo=4ms TTL=64
Respuesta desde 192.168.0.6: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.0.6: bytes=32 tiempo=6ms TTL=64
```

Figura 3.20 *Ping* hacia el servidor

Luego que se tuvo un *ping* correctamente, con lo cual concluimos que se tuvo conexión y comunicación completa entre los diferentes dispositivos y el servidor. Ahora se verificó nuevamente que al salir del contenedor este no se detenga.

Observamos que el contenedor *Samba* este corriendo y correctamente funcionando mediante el comando **docker ps**.

```
ericksg@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
TS           NAMES
26a71b787635  dperson/samba:rpi  "samba.sh -u ericksg..."  15 minutes ago
ericksg_samba_1
```

Figura 3.21 Verificación del contenedor

Las pruebas de funcionamiento para el acceso a las carpetas y archivos se realizaron mediante dos opciones y fueron:

La primera opción es por medio de ejecutar, el cual se accedió por medio del teclado con **windows + r** y se escribe la dirección del servidor *Samba* con **** en el principio, se presiona en aceptar y se abrió el servidor con las carpetas y archivos compartidos.

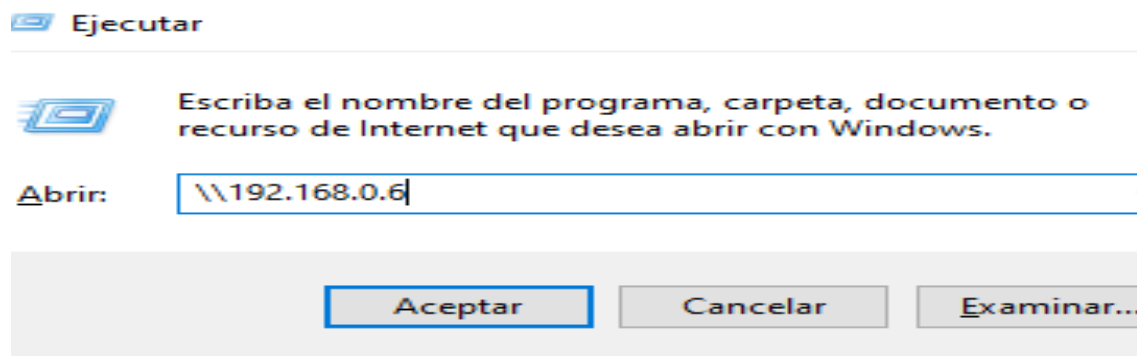


Figura 3.22 Acceder al servidor *Samba*

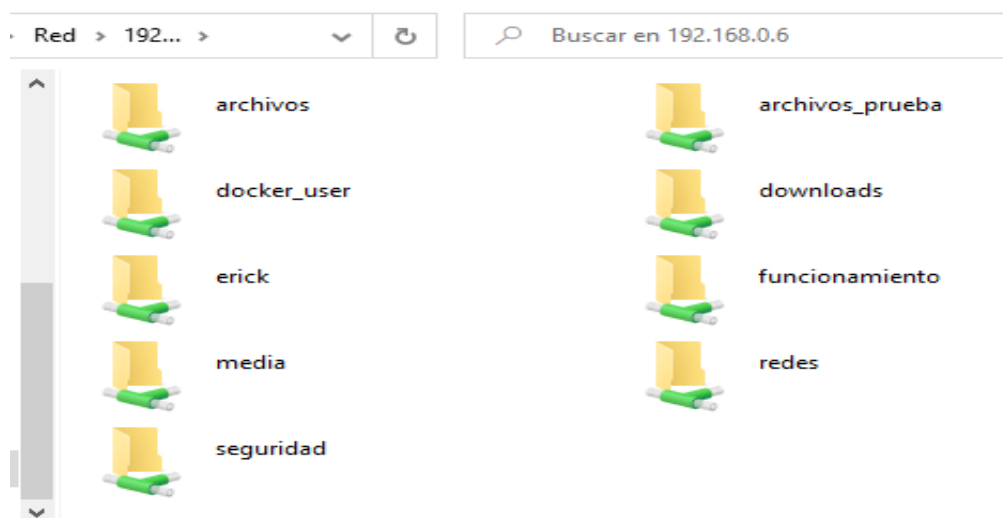


Figura 3.23 Carpetas del servidor

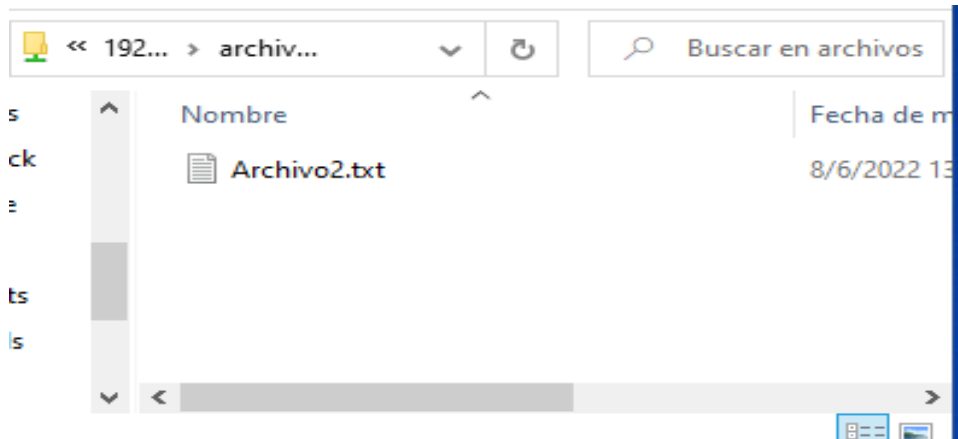


Figura 3.24 Archivos del servidor

La segunda manera que se utilizó para ingresar al servidor es mediante un explorador de archivos el cual se lo abrió y en la parte superior, en el apartado de acceso rápido se borra el contenido y volvemos a escribir la dirección IP del servidor con los \\ al principio.

El funcionamiento es igual que la anterior forma, las carpetas y los archivos antes visualizados por el modo de ejecutar, se presentarán y podrán ser accesibles para los dispositivos que ingresen al servidor.

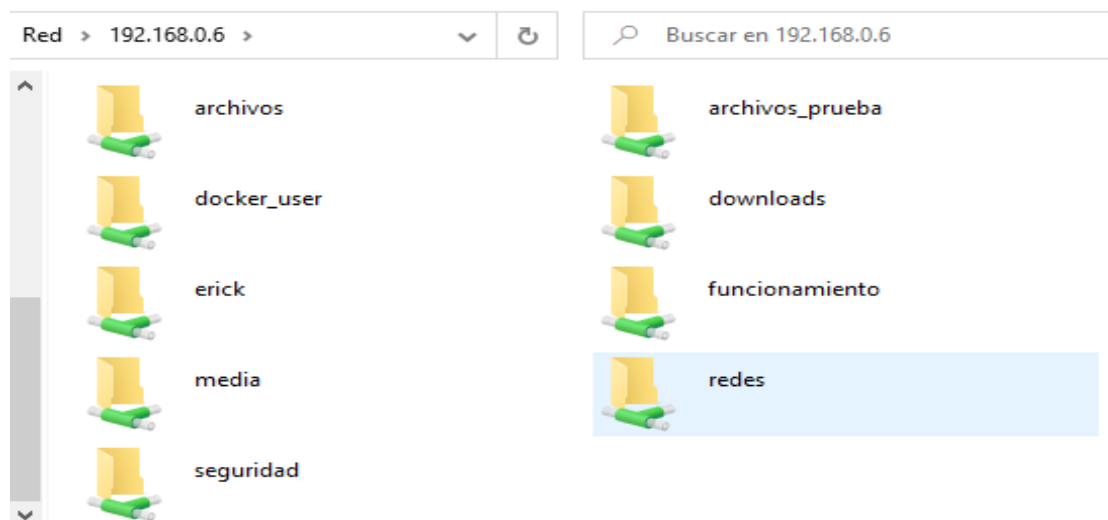


Figura 3.25 Acceder al servidor mediante un explorador de archivos

Costo de la implementación del prototipo.

Una vez finalizado la creación del prototipo de *networking* es necesario inspeccionar los parámetros que influenciaron al momento de dar un costo del mismo. Como el prototipo

funciona internamente en el Raspberry, se escoge la placa necesaria, y el valor más accesible comercialmente.

En varias de las ocasiones el *Raspberry* puede venir separado o con algunos dispositivos que le pueden ayudar para las visualizaciones o instalaciones, si este fuera el caso sería necesario inspeccionar mucho más a fondo dichos dispositivos como: pantalla con entrada de puerto HDMI, cable HDMI, mouse, teclado y una tarjeta SD en donde se instalará el sistema operativo *Raspbian*.

Otra de las formas en cómo se puede utilizar el *Raspberry* es mediante alguna aplicación que nos facilite utilizar VNC que como sus siglas en internet lo dicen llega a ser una computación virtual en red, con el cual se puede observar en la pantalla del ordenador que instalemos este software a nuestro *Raspberry* siempre y cuando se ingrese con la dirección IP del *Raspberry*. Por dicha razón también es necesario tener algún tipo de software que te permita observar las direcciones IP que están siendo utilizadas en la red, así como el de observar el dispositivo que usa dicha dirección, y con el nombre de usuario y contraseña del mismo se puede acceder al *Raspberry*.

Siendo este el caso, el costo de implementación del prototipo solo llega a ser el valor comercial del *Raspberry* con su respectiva tarjeta SD.

Ahora se procederá a dar un valor a nuestro prototipo el cual estará basado en: *Raspberry pi 3B* y tarjeta SD ya que el presente prototipo fue realizado por medio de un *software VNC*.

El valor presentado a continuación, ha sido empleado por medio de comparaciones de diferentes páginas web como mercado libre, Amazon y diferentes locales comerciales como electrónicas, robóticas o personas que pueden realizar ventas del dispositivo. Las comparaciones fueron realizadas y se escogió el valor más accesible, es decir aquel sea más económico y que disponga con todas las características necesarias para su funcionamiento [2].

A continuación, se presenta las características del *Raspberry pi 3B* implementado en el proyecto, como su costo a nivel general en el mercado [2].

Tabla 2.1 *Raspberry pi 3B* y su valor

<i>Raspberry pi 3 B</i>	Procesador <i>Bradcom BCM2837B</i> Cortex-A53 de 1.4 GHz	\$269.99
-------------------------	--	----------

	1GB de SDRAM Bluetooth 4.2 BLE y Wifi Dual Band /b/g/n/ac Conexión por cable Gigabit Ethernet Admite POE Puertos para cámara y 4 puertos USB 2.0 Alimentación de 5V	
--	--	--

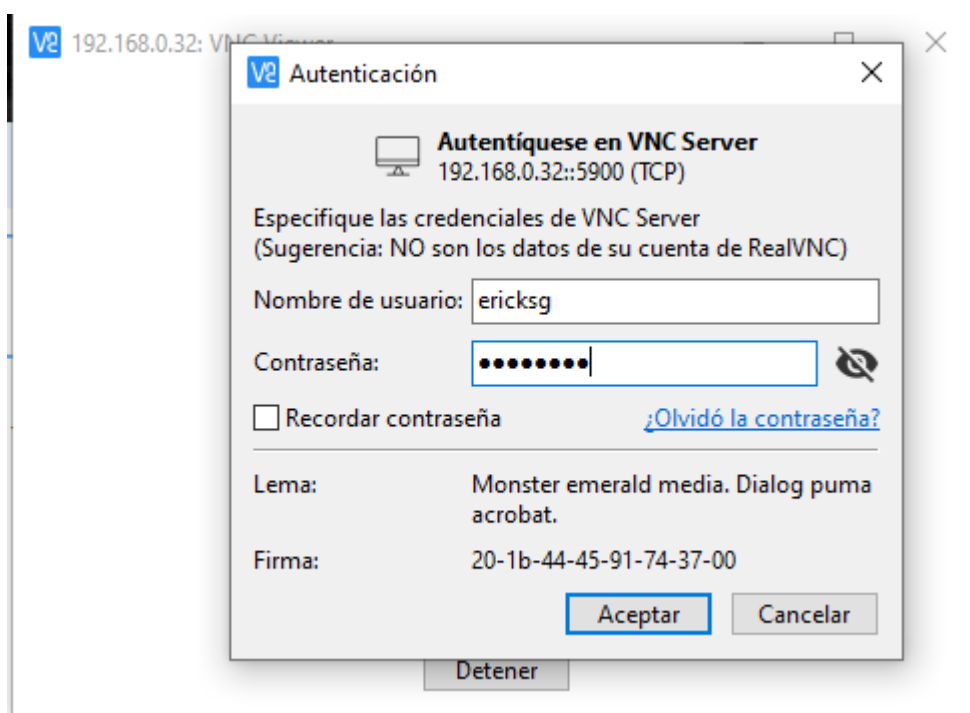


Figura 3.26 Acceder al *Raspberry* por medio de VNC

Lugar en donde se encuentra localizado el dispositivo.

Una vez terminada la configuración del prototipo y de verificar que esté totalmente funcional, se procede a ubicarlo en un área adecuada. De preferencia cerca del *router*, para que este esté conectado directamente por cable *ethernet* y no por *wifi*, un lugar en donde el prototipo no pueda ser movido con facilidad o que alguna tercera persona desconecte accidentalmente ya sea el cable, la tarjeta SD o el cargador, lo que accionaría que nuestro prototipo deje de funcionar o funcione con algunas fallas.

El lugar puede ser el que se desee, pero recomendable que sea cerca del *router*, en un sitio central del hogar o de la empresa en donde funcionará el servidor *Samba*. El servidor no necesariamente debe de estar en un sitio central se lo localizó en dicho lugar

por acomodamiento y proximidad al *router*, evitando así que los dispositivos que funcionen en conjunto con nuestro prototipo estén de manera desordenada y ocasionen algún accidente intencional o no intencional tanto al prototipo como para otros dispositivos.

La alimentación del prototipo fue mediante corriente continua. Como el prototipo implementado se encuentra dentro del *Raspberry* el cual está siendo conectado por medio de un cargador de 5V, necesariamente para su funcionamiento debe estar en conectado todo el tiempo, por lo que se recomienda usar dispositivos adecuados y que cumplan con este requerimiento.



Figura 3.27 Ubicación del Raspberry



Figura 3.28 Docker y Samba en el Raspberry

4 CONCLUSIONES

- Una vez ingresada nuestra opción de *networking* al servidor *Samba*, la forma más rápida y sencilla para verificar el funcionamiento es mediante el *ping*,

aunque muchas veces las modificaciones que se realicen dentro del servidor no se actualizan de manera inmediata por lo que es necesario que se detenga a nuestro contenedor y que lo eliminemos, así las modificaciones que se realicen puedan ser completadas y levantadas cada vez que coloquemos el comando para activar el archivo *Docker-compose*, ya que en el presente proyecto existía el problema de actualización de modificaciones lo que ocasionaba que se podía realizar un ping pero no se podía ingresar al servidor ya que las configuraciones fueron realizadas en el anterior contenedor y no en el actual.

- Al momento de realizar nuestro archivo *Docker-compose* en el apartado del servicio a levantar específicamente en la imagen, es necesario que especifiquemos de donde va a ser utilizada, ya que, si por alguna razón optamos por la opción de *latest* que es la última versión de la imagen, alguna de las opciones no puede llegar a funcionar de la manera correcta, es decir que puede funcionar en la imagen anterior que utilizaste y no en la última. Por dicha razón el especificar la versión y el lugar en donde se está obteniendo la imagen es fundamental al momento de levantar el contenedor ya sea por el archivo *Docker-compose* o por *Docker run*, puede existir muchas más funciones en otras versiones, pero el funcionamiento depende de la imagen que se llega a utilizar.
- Con *Docker* se logra obtener una mayor velocidad ya que una de la principal características es que no arranca un sistema operativo, lo que hace que no utilice recursos que se puede llegar a necesitar en un futuro, esto en términos de velocidad para levantar o detener algún servicio lo hace extremadamente sencillo ya que no es necesario verificar un error crucial sino que simplemente se puede volver a utilizar una versión anterior del servicio que se quiere utilizar o simplemente ocupar una copia y la original dejarla sin actualizaciones hasta que se logre implementarla con su función en su totalidad, incluso llegando a ser muy útil en caso de llegar algún virus malicioso ya que como actúan por separados, solo se ve afectado el contenedor en donde se genera el virus y no se llega a propagar a los demás.
- La creación de los usuarios es muy importante al momento de implementar el servicio de *Samba* ya que estos son los que tienen acceso o permisos para la modificación del mismo, pero más sencillo aún es implementar el servidor con un solo usuario con los permisos adquiridos, esto permitiendo que el servidor tenga el acceso, es decir cualquier dispositivo que esté en la red puede ingresar al servidor y acceder a las carpetas compartidas, esto puede ser perjudicial al momento de tener ataques internos, pero en caso de que la seguridad para

algunas carpetas sea crucial lo más práctico es crear usuarios y así permitir acceso a diferentes carpetas y no a todas.

5 RECOMENDACIONES

- La recomendación más importante es que al momento de utilizar *Docker* para implementar los contenedores, *docker-compose* es la mejor solución, debido a que el *Raspberry* utiliza como su principal almacenamiento una tarjeta SD y estas no están diseñadas para que contengan una gran cantidad de información, siendo así, que, si llegara el caso de que la tarjeta SD deje de funcionar, lo más fácil o lo más recomendable es recrear nuestro archivo *.yml*, copiar el código y ejecutarlo, así nos evitamos de volver a llamar los contenedores, crear las redes e incluso tener que llamar una imagen la cual puede llegar a ser modificada.
- Es recomendable saber utilizar los drivers, como saber del funcionamiento que tienen cada uno de ellos debido a que, en el presente proyecto al momento de verificar los funcionamientos de los *drivers*, uno de los más problemáticos es *overlay* debido a que se tiene que activar funciones de *swarm*, y este lo que ocasiona es que al momento de levantar otro tipo de redes con otros drivers se presente un error o una advertencia de que el modo *swarm* este activo es decir que puede funcionar como un cliente, en el presente proyecto la advertencia salía pero no existía un problema debido a que se estaba trabajando en modo maestro, pero en otras condiciones es recomendable salir del modo *swarm* y activar los servicios que se necesite siempre y cuando no sea el de compartir la red en otros nodos siendo este el caso el modo *swarm* estaría correctamente siendo activado.
- Para las pruebas de funcionamiento es recomendable verificar que en las imágenes que se estén procediendo a instalar o levantar, tengan todas las herramientas o características instaladas ya que un ejemplo claro es que cuando quieren realizar las pruebas de funcionamiento algunas características como *ifconfig* no se encuentran previamente instaladas, por lo que puede aparecer como comando no encontrado o algún otro error, lo que puede ocasionar que se llegue a realizar otras acciones que no son necesarias. Para esto es simplemente es proceder a instalar las características necesarias, aunque la mayoría de las imágenes ya tienen instaladas estas características.

- Personalmente recomiendo enviar una actualización a todos los servicios que se están levantando, simplemente para actualizar alguna característica e incluso para que instale alguna que debió de venir instalada por defecto y no estuvo.

6 REFERENCIAS BIBLIOGRAFICAS

- [1] *Raspberrypi*, «¿Que es *Raspberry Pi?*,» 21 08 2021. [En línea]. Available: <https://raspberrypi.cl/que-es-raspberry/>. [Último acceso: 21 06 2022].
- [2] L. Llamas, «Modelos y características de raspberry pi,» 17 11 2017. [En línea]. Available: <https://www.luisllamas.es/modelos-de-raspberry-pi/>. [Último acceso: 23 06 2022].
- [3] IONOS Digitalguide, «Servidor Samba: la solución perfecta para redes multiplataforma,» 31 01 2020. [En línea]. Available: <https://www.ionos.es/digitalguide/servidores/configuracion/servidor-samba-una-solucion-multiplataforma/>. [Último acceso: 15 07 2022].
- [4] R. A. Lázaron y M. M. Miriel Mesa, «Implementación de un Servidor Samba con autenticación LDAP como alternativa Libre a los servidores de Dominio Windows,» 11 09 2017. [En línea]. Available: <https://www.monografias.com/trabajos-pdf4/implementacion-servidor-samba-autenticacion-ldap/implementacion-servidor-samba-autenticacion-ldap.pdf>. [Último acceso: 17 07 2022].
- [5] Dockertips, «Aprendiendo a utilizar Docker Compose,» 07 06 2020. [En línea]. Available: <https://dockertips.com/utilizando-docker-compose>. [Último acceso: 27 06 2022].
- [6] JCSNS, «Una introducción a docker y análisis de su rendimiento.,» 17 03 2017. [En línea]. Available: https://www.researchgate.net/profile/Harrison-Bhatti/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance/links/61facc0c007fb504472fd6c7/An-Introduction-to-Docker-and-Analysis-of-its-Performance.pdf. [Último acceso: 15 07 2022].

- [7] Atareao, «Networking en docker,» 25 11 2019. [En línea]. Available: <https://atareao.es/tutorial/docker/redes-en-docker/>. [Último acceso: 13 07 2022].
- [8] Pelado Nerd, «INSTALANDO DOCKER en una RASPBERRY PI,» 13 12 2018. [En línea]. Available: <https://www.youtube.com/watch?v=pliGG1M87W8>. [Último acceso: 23 06 2022].
- [9] Pelado nerd, «NETWORKING EN D|OCKER!,» 24 12 2018. [En línea]. Available: <https://www.youtube.com/watch?v=BNHNMoSJz4g>. [Último acceso: 24 06 2022].
- [10] Carlo y C. Arjona Quijano, «Docker creando un entorno seguro,» 10 09 2020. [En línea]. Available: <https://ridda2.utp.ac.pa/bitstream/handle/123456789/11488/Docker%202020-UCC.pdf?sequence=1>. [Último acceso: 23 06 2022].

ANEXOS

ANEXO I: Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 13 de septiembre de 2022

De mi consideración:

Yo, **JAVIER ALEJANDRO ARMAS NAVARRETE**, en calidad de Director del Trabajo de Integración Curricular titulado **PROTORIPO DE NETWORKING MEDIANTE DOCKER EN UNA RASPBERRY**, elaborado por el estudiante **ERICK DAMIAN TEPAN SILVA** de la carrera **TECNOLOGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES**, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del %.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.



Atentamente,

JAVIER ALEJANDRO ARMAS NAVARRETE

Docente

Escuela de Formación de Tecnólogos

ANEXO 2: Estructura del archivo *docker-compose.yml*

```
Archivo Editar Pestañas Ayuda
version: "2"

services:

  samba:
    image: dperson/samba:rpi
    restart: always
    command: '-u "ericksg:et123456" -s "media;/media;yes;no" -s "downloads;/downloads;yes;no" -s "archivos;/archivos;yes;no"
-s "erick;/erick;yes;no" -s "seguridad;/seguridad;yes;no" -s "redes;/redes;yes;no" -s "archivos_prueba;/archivos_prueba;yes;n
o" -s "funcionamiento;/funcionamiento;yes;no" -s "docker_user;/docker_user;yes;no"'
    stdin_open: true
    tty: true
    ports:
      - 139:130
      - 445:445
    volumes:
      - /home/ericksg/media:/media
      - /home/ericksg/downloads:/downloads
      - /home/ericksg/archivos:/archivos
      - /home/ericksg/archivos_prueba:/archivos_prueba
      - /home/ericksg/docker_user:/docker_user
      - /home/ericksg/erick:/erick
      - /home/ericksg/funcionamiento:/funcionamiento
      - /home/ericksg/redes:/redes
      - /home/ericksg/seguridad:/seguridad
    networks:
      lan:
        ipv4_address: 192.168.0.6

networks:
  lan:
    driver: macvlan
    driver_opts:
      parent: eth0
    ipam:
      config:
        - subnet: "192.168.0.0/24"
          gateway: "192.168.0.1"
-- INSERTAR --
```

38, 33

Todo