

# Curso de Docker

Diego Saavedra

May 17, 2024

# Table of contents

<b>1</b>	<b>Curso de Docker</b>	<b>4</b>
1.1	¿Qué es este Curso?	4
1.2	¿A quién está dirigido?	4
1.3	¿Cómo contribuir?	4
<b>I</b>	<b>Unidad 1: Introducción a Docker</b>	<b>5</b>
<b>2</b>	<b>Docker</b>	<b>6</b>
2.1	Ejemplos:	7
2.2	Comandos básicos de Docker:	8
2.3	Atajos y Comandos Adicionales:	9
2.4	Práctica:	10
2.5	¿Qué Aprendimos?	10
<b>II</b>	<b>Unidad 2: Dockerfile y Docker Compose</b>	<b>11</b>
<b>3</b>	<b>Dockerfile y Docker Compose</b>	<b>12</b>
3.1	Introducción	12
3.1.1	Dockerfile	12
3.1.2	Docker Compose	12
3.2	Ejemplos:	12
3.2.1	server.js	13
3.2.2	Dockerfile	13
3.2.3	docker-compose.yml	14
3.3	Práctica:	15
3.4	¿Qué Aprendimos?	15
<b>III</b>	<b>Unidad 3: Servidores en Docker</b>	<b>16</b>
<b>4</b>	<b>Servidores Docker</b>	<b>17</b>
4.1	MySQL	17
4.2	PostgreSQL	18
4.3	MongoDB	18
4.4	Apache	19
4.5	Nginx	19
4.6	Node.js	19
4.7	React	19
4.8	Angular	19
4.9	Spring Boot	19

4.10 Django . . . . .	19
<b>IV Ejercicios</b>	<b>20</b>
<b>5 Actividad Práctica</b>	<b>21</b>
5.1 Objetivo . . . . .	21
5.2 Instrucciones . . . . .	21
5.3 Entregables: . . . . .	21
<b>6 Actividad Práctica</b>	<b>23</b>
6.1 Objetivo . . . . .	23
6.2 Instrucciones . . . . .	23
6.3 Entregables: . . . . .	23
6.4 Rubrica de Evaluación: . . . . .	24
<b>7 Actividad Práctica</b>	<b>25</b>
7.1 Objetivo . . . . .	25
7.2 Instrucciones . . . . .	25
7.3 Entregables: . . . . .	25
7.4 Rubrica de Evaluación: . . . . .	26
<b>8 Actividad Práctica</b>	<b>27</b>
8.1 Objetivo . . . . .	27
8.2 Instrucciones . . . . .	27
8.3 Entregables: . . . . .	27
8.4 Rubrica de Evaluación: . . . . .	28

# 1 Curso de Docker

¡Bienvenidos al Curso de Docker!

Este curso te guiará a través de un viaje desde los fundamentos hasta el dominio de Docker, la plataforma de contenedores líder en la industria.

## 1.1 ¿Qué es este Curso?

Este curso exhaustivo te llevará desde los conceptos básicos de Docker hasta la implementación práctica de aplicaciones y servicios en contenedores. A través de una combinación de teoría y ejercicios prácticos, explorarás cada aspecto diseñado para fortalecer tus habilidades en el uso de Docker. Desde la instalación inicial hasta la orquestación de contenedores con Docker Compose, este curso te proporcionará las herramientas y conocimientos necesarios para desarrollar, desplegar y escalar aplicaciones con eficiencia.

## 1.2 ¿A quién está dirigido?

Este curso está diseñado tanto para aquellos que están dando sus primeros pasos en Docker como para aquellos que desean profundizar en sus conocimientos. Ya seas un estudiante, un profesional en busca de nuevas habilidades o alguien apasionado por la tecnología de contenedores, este curso te brindará la base necesaria para trabajar de manera efectiva con Docker en cualquier entorno.

## 1.3 ¿Cómo contribuir?

Valoramos tu participación en este curso. Si encuentras errores, deseas sugerir mejoras o agregar contenido adicional, ¡nos encantaría recibir tus contribuciones! Puedes contribuir a través de nuestra plataforma en línea, donde puedes compartir tus comentarios y sugerencias. Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de usuarios de Docker.

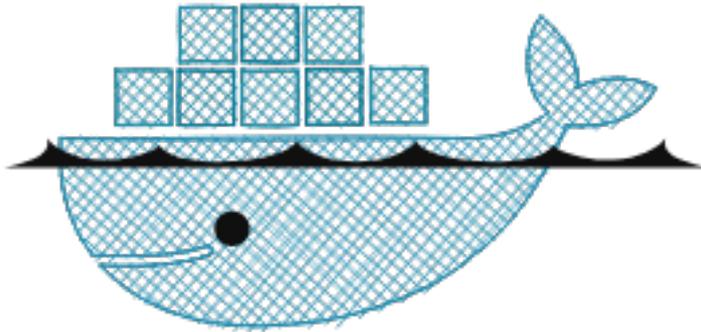
Este curso ha sido creado con el objetivo de proporcionar acceso gratuito y universal al conocimiento de Docker. Estará disponible en línea para que cualquiera, sin importar su ubicación o circunstancias, pueda acceder y aprender a su propio ritmo.

¡Esperamos que disfrutes este emocionante viaje de aprendizaje y descubrimiento en el mundo de Docker y los contenedores!

## **Part I**

# **Unidad 1: Introducción a Docker**

## 2 Docker



Docker es una plataforma que permite desarrollar, enviar y ejecutar aplicaciones en contenedores. Un **contenedor** es una instancia ejecutable de una **imagen**, que es una especie de **plantilla** que contiene todo lo necesario para ejecutar una aplicación.

Haciendo una analogía con los contenedores de transporte, una **imagen** sería el **contenedor** en sí, y el **contenedor** sería la **carga** que se transporta.



Una imagen Docker es una plantilla inmutable que contiene un conjunto de instrucciones para crear un contenedor Docker. Las imágenes son portátiles y pueden ser compartidas, almacenadas y actualizadas.

### 💡 Tip

Las imágenes Docker son inmutables, lo que significa que no se pueden modificar una vez creadas. Si se realizan cambios en una imagen, se debe crear una nueva versión de la imagen.



## Container

Un contenedor Docker es una instancia ejecutable de una imagen Docker. Se ejecuta de manera aislada y contiene todo lo necesario para ejecutar la aplicación, incluyendo el código, las dependencias, el entorno de ejecución, las bibliotecas y los archivos de configuración.

### 💡 Tip

Un contenedor aísla la aplicación de su entorno, lo que garantiza que la aplicación se ejecute de manera consistente en diferentes entornos.

## 2.1 Ejemplos:

Descargar una imagen:

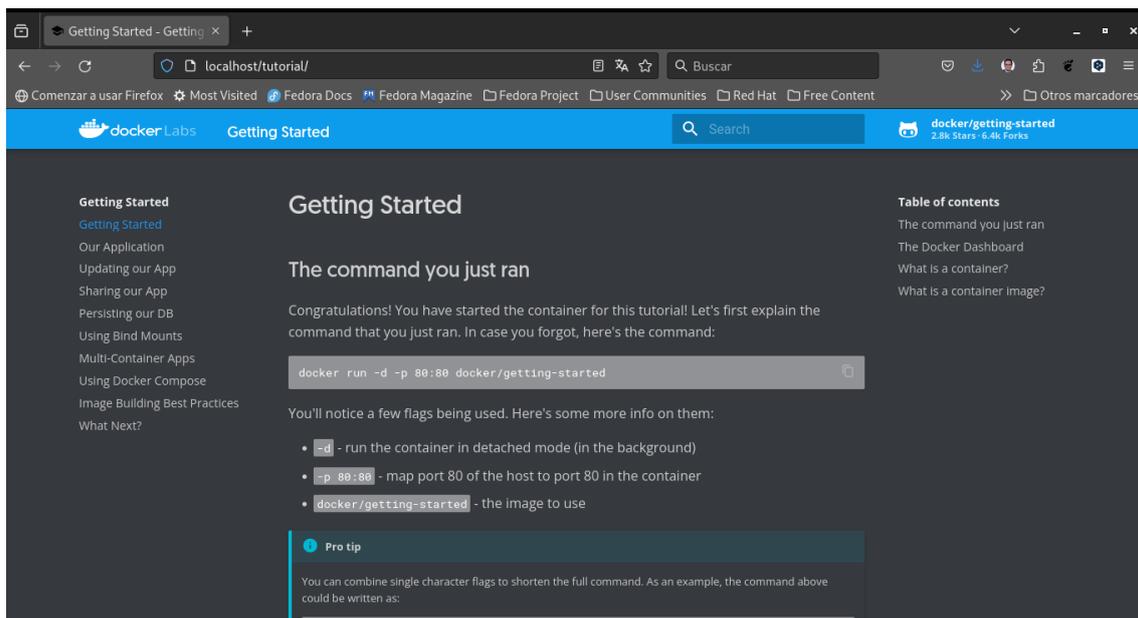
```
docker pull docker/getting-started
```

```
▼ TERMINAL
• statick main U:11 ? :7 ~/workspaces/curso_docker/book docker pull docker/getting-started
Using default tag: latest
latest: Pulling from docker/getting-started
Digest: sha256:d79336f4812b6547a53e735480dde67f8f8f7071b414fbd9297609ffb989abc1
Status: Image is up to date for docker/getting-started:latest
docker.io/docker/getting-started:latest
```

Este comando descarga la imagen **getting-started** desde el registro público de Docker.

Correr un contenedor en el puerto 80:

```
docker run -d -p 80:80 docker/getting-started
```



Este comando ejecuta un contenedor desenlazado en segundo plano (-d) y mapea el puerto 80 de la máquina host al puerto 80 del contenedor (-p 80:80).

Descargar una imagen desde un registro.

#### Tip

El comando -p se utiliza para mapear los puertos de la máquina host al contenedor, muchas personas consideran que significa “puerto”. Sin embargo en realidad significa “publicar” o “publicar puerto”.

## 2.2 Comandos básicos de Docker:

Descargar una imagen desde un registro.

```
docker pull <IMAGE_NAME:TAG>
```

Listar las imágenes descargadas.

```
docker images
```

Listar contenedores en ejecución.

```
docker ps
```

Listar todos los contenedores, incluyendo los detenidos.

```
docker ps -a
```

Ejecutar un contenedor a partir de una imagen.

```
docker run -d -p <HOST_PORT>:<CONTAINER_PORT> <IMAGE_NAME:TAG>
```

Detener un contenedor en ejecución.

```
docker stop <CONTAINER_ID>
```

Iniciar un contenedor detenido.

```
docker start <CONTAINER_ID>
```

Eliminar un contenedor.

```
docker rm <CONTAINER_ID>
```

Eliminar una imagen.

```
docker rmi <IMAGE_NAME:TAG>
```

## 2.3 Atajos y Comandos Adicionales:

Ejecutar comandos dentro de un contenedor en ejecución.

```
docker exec -it <CONTAINER_ID> /bin/bash
```

Obtener detalles sobre un contenedor o imagen.

```
docker inspect <CONTAINER_ID or IMAGE_NAME:TAG>
```

Ver los logs de un contenedor.

```
docker logs <CONTAINER_ID>
```

Utilizar Docker Compose para gestionar aplicaciones multi-contenedor.

```
docker-compose up -d
```

## 2.4 Práctica:

- Descarga la imagen de Nginx desde el registro público.
- Crea y ejecuta un contenedor de Nginx en el puerto 8080.
- Detén y elimina el contenedor creado
- Utiliza los comandos para detener y eliminar un contenedor.

Resolución de la Actividad Práctica

1. Abre tu terminal o línea de comandos.
2. Descarga la imagen de Nginx desde el registro público de Docker:

```
docker pull nginx
```

3. Crea y ejecuta un contenedor de Nginx en el puerto 8080:

```
docker run -d -p 8080:80 nginx
```

Elige un puerto en tu máquina local (por ejemplo, 8080) para mapearlo al puerto 80 del contenedor.

4. Verifica que el contenedor esté en ejecución:

```
docker ps
```

5. Si el contenedor está en ejecución, deténlo utilizando el siguiente comando:

```
docker stop <CONTAINER_ID>
```

Reemplaza con el ID real del contenedor que obtuviste en el paso anterior.

6. Elimina el contenedor detenido:

```
docker rm <CONTAINER_ID>
```

Reemplaza con el ID real del contenedor.

### Tip

Combina los comandos `docker ps`, `docker stop`, y `docker rm` para gestionar contenedores eficientemente.

¡Practica estos pasos para familiarizarte con el ciclo de vida de los contenedores Docker!

## 2.5 ¿Qué Aprendimos?

- Aprendimos a descargar imágenes, correr contenedores y gestionarlos básicamente.
- Entendimos la importancia de las banderas en los comandos Docker.
- Practicamos la creación y gestión de contenedores Docker.

## **Part II**

# **Unidad 2: Dockerfile y Docker Compose**

## 3 Dockerfile y Docker Compose

### 3.1 Introducción

Dockerfile y Docker Compose son herramientas esenciales para la construcción y gestión de aplicaciones Docker. Un Dockerfile es un archivo de texto que define cómo se construirá una imagen Docker, mientras que Docker Compose es una herramienta para definir y gestionar aplicaciones Docker con múltiples contenedores. En esta lección, aprenderemos cómo usar Dockerfile y Docker Compose para personalizar imágenes Docker y orquestar servicios en un entorno multi-contenedor.

A continuación veremos algunos conceptos básicos sobre Dockerfile y Docker Compose.

#### 3.1.1 Dockerfile

Un Dockerfile es un archivo de texto que contiene una serie de instrucciones para construir una imagen Docker. Estas instrucciones incluyen la configuración del sistema operativo base, la instalación de paquetes y dependencias, la configuración de variables de entorno y la definición de comandos para ejecutar la aplicación.

#### 3.1.2 Docker Compose

Docker Compose es una herramienta para definir y gestionar aplicaciones Docker con múltiples contenedores. Permite definir servicios, redes y volúmenes en un archivo YAML y orquestar la ejecución de los contenedores en un entorno de desarrollo o producción.

### 3.2 Ejemplos:

En este ejemplo vamos a dockerizar una aplicación nodejs con un servidor sencillo en express.

Empezamos por el código de nuestra aplicación:

Para ello creamos un nuevo proyecto nodejs con el siguiente comando:

```
npm init -y
```

Instalamos el paquete express con el siguiente comando:

```
npm install express
```

Creamos los siguientes archivos:

- server.js
- package.json
- Dockerfile
- docker-compose.yml

### 3.2.1 server.js

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

### 3.2.2 Dockerfile

```
# Use the official Node.js 14 image
FROM node:14

# Set the working directory in the container
WORKDIR /app

# Copy the dependencies file to the working directory
COPY package.json .

# Install dependencies
RUN npm install

# Copy the app code to the working directory
COPY . .

# Expose the port the app runs on
EXPOSE 3000

# Serve the app
CMD ["node", "server.js"]
```

### 3.2.3 docker-compose.yml

```
services:
  myapp:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./app
```

En este ejemplo, el Dockerfile define una imagen Docker para una aplicación Node.js. El archivo docker-compose.yml define un servicio llamado myapp que utiliza el Dockerfile.nodejs para construir la imagen y expone el puerto 3000 para acceder a la aplicación.

#### Tip

El puerto del lado izquierdo de los 2 puntos en el archivo docker-compose.yml es el puerto en el host, mientras que el puerto del lado derecho es el puerto en el contenedor.

Para probar nuestro ejemplo, ejecutamos el siguiente comando:

```
docker-compose up -d
```

Esto construirá la imagen Docker y ejecutará el contenedor en segundo plano. Podemos acceder a la aplicación en <http://localhost:3000>.

Para verificar que el contenedor está en ejecución, ejecutamos el siguiente comando:

```
docker ps
```

Podemos utilizar una aplicación como Thunder Client o Postman para enviar una solicitud HTTP a la aplicación y ver la respuesta.

Para detener y eliminar el contenedor, ejecutamos el siguiente comando:

```
docker-compose down
```

#### Tip

Recuerda: La imagen que se crea a partir del Dockerfile se almacena en el caché local de Docker. Si realizas cambios en el Dockerfile y deseas reconstruir la imagen, puedes usar el comando

```
docker-compose up --build
```

### 3.3 Práctica:

- Crea un Dockerfile para una aplicación Python simple.
- Configura un archivo docker-compose.yml para ejecutar la aplicación.

Resolución de la Actividad Práctica

Ejemplo de aplicación Python simple:

```
# app.py
print("Hello, World!")
```

Ejemplo de Dockerfile:

```
FROM python:3.12
WORKDIR /app
COPY . .
CMD ["python", "app.py"]
```

Ejemplo de docker-compose.yml:

```
services:
  myapp:
    build:
      context: .
      dockerfile: Dockerfile.python
    image: my-python-app
```

### 3.4 ¿Qué Aprendimos?

- Aprendimos a crear un Dockerfile para personalizar una imagen Docker.
- Entendimos cómo usar Docker Compose para orquestar servicios en un entorno multi-contenedor.
- Practicamos la configuración básica de un Dockerfile y un archivo docker-compose.yml.

## **Part III**

# **Unidad 3: Servidores en Docker**

## 4 Servidores Docker

En esta unidad vamos a aprender acerca de los distintos servidores de bases de datos que podemos utilizar en Docker.

Vamos a utilizar el concepto de Dockerfile y Docker Compose para esta actividad

### 4.1 MySQL

Empezamos con la creación de un archivo llamado `Dockerfile` con el siguiente contenido:

```
# Use the official MySQL image

FROM mysql:8.0.26

# Set the root password

ENV MYSQL_ROOT_PASSWORD=mysecretpassword

# Expose port 3306

EXPOSE 3306

# Start MySQL

CMD ["mysqld"]
```

Para construir la imagen de MySQL, ejecutamos el siguiente comando:

```
docker build -t my-mysql .
```

Para ejecutar el contenedor de MySQL, ejecutamos el siguiente comando:

```
docker run --name some-mysql -d my-mysql
```

Posiblemente en algunos casos necesitemos utilizar un archivo `docker-compose.yml` para la creación del contenedor, el contenido del archivo sería el siguiente:

```
version: '3.3'

services:
  mysql:
    image: mysql:8.0.26
    environment:
      MYSQL_ROOT_PASSWORD: mysecretpassword
    ports:
      - "3306:3306"
```

Para ejecutar el contenedor de MySQL, ejecutamos el siguiente comando:

```
docker-compose up -d
```

Para configurar en un cliente de MySQL, necesitamos conocer los siguientes datos:

- Host: localhost
- Port: 3306
- User: root
- Password: mysecretpassword

## 4.2 PostgreSQL

## 4.3 MongoDB

Otros servidores típicos que vamos a utilizar en el desarrollo de software son:

- Apache
- Nginx
- Tomcat

**4.4 Apache**

**4.5 Nginx**

**4.6 Node.js**

**4.7 React**

**4.8 Angular**

**4.9 Spring Boot**

**4.10 Django**

**Part IV**  
**Ejercicios**

# 5 Actividad Práctica

## 5.1 Objetivo

El objetivo de esta actividad es familiarizarse con los comandos básicos de Docker y aprender a gestionar contenedores de manera eficiente.

## 5.2 Instrucciones

1. Inicia un contenedor de la imagen “nginx” en segundo plano utilizando el comando **docker run**. Asegúrate de mapear el puerto 8080 del host al puerto 80 del contenedor.
2. Detén y elimina el contenedor que acabas de crear utilizando los comandos Docker apropiados.
3. Crea un nuevo contenedor con la imagen “alpine” y ejecuta un terminal interactivo dentro de él.
4. Desde el contenedor alpine, instala el paquete curl utilizando el gestor de paquetes apk.
5. Crea una imagen llamada “alpine-curl” a partir de este contenedor modificado.

Recuerda que puedes consultar la documentación de Docker si necesitas ayuda con los comandos.

## 5.3 Entregables:

- Imagen Docker “alpine-curl” disponible localmente.

Resolución de la Actividad Práctica

- Iniciar un contenedor Nginx:

```
docker run -d -p 8080:80 --name my-nginx nginx
```

Detener y eliminar el contenedor Nginx:

```
docker stop my-nginx  
docker rm my-nginx
```

Crear un contenedor Alpine interactivo:

```
docker run -it --name my-alpine alpine /bin/sh
```

Instalar el paquete curl desde el contenedor Alpine:

```
apk add --no-cache curl
```

Crear una nueva imagen “alpine-curl”:

```
docker commit my-alpine alpine-curl
```

# 6 Actividad Práctica

## 6.1 Objetivo

El objetivo de esta actividad es practicar el uso de volúmenes en Docker para persistir datos entre contenedores.

## 6.2 Instrucciones

1. Crea un nuevo volumen llamado “mydata” utilizando el comando `docker volume create`.
2. Inicia un contenedor de la imagen “nginx” y vincula el volumen “mydata” al directorio “/usr/share/nginx/html” dentro del contenedor utilizando el comando `docker run`.
3. Crea un archivo HTML dentro del volumen “mydata” con el mensaje “Hola, este es un archivo HTML persistente”. Puedes hacer esto iniciando un contenedor temporal que tenga acceso al volumen y utilizando un editor de texto para crear el archivo.
4. Inicia otro contenedor de la imagen “nginx” y vincula el mismo volumen “mydata” al directorio “/usr/share/nginx/html” dentro de este segundo contenedor. Verifica que puedes ver el archivo HTML que creaste en el paso 3.

Recuerda que puedes consultar la documentación de Docker si necesitas ayuda con los comandos.

## 6.3 Entregables:

- Documento explicando los comandos utilizados.
- Capturas de pantalla que demuestren la persistencia de datos entre contenedores.

Resolución de la Actividad Práctica

Crear un nuevo volumen:

```
docker volume create mydata
```

Iniciar el primer contenedor Nginx con el volumen:

```
docker run -d -p 8080:80 --name nginx-1 -v mydata:/usr/share/nginx/html nginx
```

Crear un archivo HTML dentro del volumen:

```
docker exec -it nginx-1 sh -c "echo 'Hola, este es un archivo HTML persistente' > /usr/sh
```

Iniciar el segundo contenedor Nginx con el mismo volumen:

```
docker run -d -p 8081:80 --name nginx-2 -v mydata:/usr/share/nginx/html nginx
```

Verificar la persistencia del archivo HTML:

- Acceder a <http://localhost:8080> en el navegador.
- Acceder a <http://localhost:8081> en el navegador.

## 6.4 Rubrica de Evaluación:

- Correcta creación y vinculación de volúmenes: 6 puntos
- Creación y persistencia de archivos en el volumen: 8 puntos
- Verificación exitosa de la persistencia entre contenedores: 6 puntos

# 7 Actividad Práctica

## 7.1 Objetivo

El objetivo de esta actividad es practicar la creación de imágenes personalizadas utilizando Dockerfile y la orquestación de servicios con Docker Compose.

## 7.2 Instrucciones

1. Crea un Dockerfile para una aplicación Python simple que imprima “Hola, Docker” al ejecutarse. Asegúrate de que tu aplicación esté configurada para escuchar en el puerto 5000.
2. Construye la imagen a partir del Dockerfile utilizando el comando `docker build`.
3. Crea un archivo `docker-compose.yml`. Dentro de este archivo, define un servicio que utilice la imagen que acabas de crear. Asegúrate de mapear el puerto 5000 del host al puerto 5000 del contenedor.
4. Inicia el servicio con Docker Compose utilizando el comando `docker-compose up`.
5. Accede a la aplicación en <http://localhost:5000> y verifica que imprime “Hola, Docker”.

Recuerda que puedes consultar la documentación de Docker y Docker Compose si necesitas ayuda con los comandos.

## 7.3 Entregables:

- Dockerfile para la aplicación Python.
- Archivo Docker Compose.
- Documento explicando los comandos utilizados.
- Capturas de pantalla que demuestren el acceso a la aplicación.

Resolución de la Actividad Práctica

Dockerfile para la aplicación Python:

```
FROM python:3.9
CMD ["python", "-c", "print('Hola, Docker)']
```

Construir la imagen:

```
docker build -t my-python-app .
```

Archivo Docker Compose (docker-compose.yml):

```
version: '3'
services:
  myapp:
    image: my-python-app
    ports:
      - "5000:5000"
```

Iniciar el servicio con Docker Compose:

```
docker-compose up -d
```

Verificar el acceso a la aplicación:

Acceder a <http://localhost:5000> en el navegador.

## 7.4 Rubrica de Evaluación:

- Correcta creación del Dockerfile: 6 puntos
- Imagen construida correctamente: 4 puntos
- Configuración adecuada en Docker Compose: 6 puntos
- Acceso exitoso a la aplicación: 4 puntos

# 8 Actividad Práctica

## 8.1 Objetivo

El objetivo de esta actividad es aplicar buenas prácticas y medidas de seguridad al trabajar con Docker.

## 8.2 Instrucciones

1. Utiliza la herramienta Snyk para escanear la imagen “alpine:latest” en busca de vulnerabilidades.
2. Implementa una política de etiquetado para las imágenes Docker que siga las mejores prácticas. Por ejemplo, podrías incluir información sobre la versión del software y la fecha de creación en las etiquetas.
3. Utiliza la herramienta “docker-compose lint” para verificar la validez de tu archivo Docker Compose. Asegúrate de corregir cualquier advertencia o error que encuentres.
4. Implementa una red de contenedores utilizando el comando `docker network create`. Asegúrate de que solo los contenedores que necesitan comunicarse entre sí tengan acceso a esta red.
5. Crea un archivo “.dockerignore” para excluir archivos y directorios innecesarios en la construcción de imágenes Docker. Esto puede ayudar a reducir el tamaño de tus imágenes y a evitar la inclusión accidental de información sensible.

Recuerda que puedes consultar la documentación de Docker y Snyk si necesitas ayuda con los comandos.

## 8.3 Entregables:

- Capturas de pantalla del escaneo de vulnerabilidades.
- Política de etiquetado para imágenes Docker.
- Resultado de la verificación del archivo Docker Compose.
- Documento explicando la implementación de la red de contenedores.
- Archivo “.dockerignore”.

Resolución de la Actividad Práctica

Escaneo de vulnerabilidades:

```
snyk container test alpine:latest
```

Política de etiquetado (ejemplo):

Se etiquetarán las imágenes con el formato “versión-año-mes-día” (ejemplo: 1.0-20230115).

Verificación del archivo Docker Compose:

```
docker-compose config
```

Implementación de una red de contenedores:

```
docker network create my-network
```

Archivo “.dockerignore” (ejemplo):

```
node_modules  
.git  
.env
```

## 8.4 Rubrica de Evaluación:

- Escaneo de vulnerabilidades realizado con éxito: 4 puntos
- Correcta implementación de la política de etiquetado: 4 puntos
- Validación exitosa del archivo Docker Compose: 4 puntos
- Implementación adecuada de la red de contenedores: 4 puntos
- Correcta configuración del archivo “.dockerignore”: 4 puntos