Docker

Implantación de Aplicaciones Web

Curso 2025/2026

Índice general

1 Docker

Docker es una plataforma de código abierto diseñada para facilitar la creación, implementación y ejecución de aplicaciones en contenedores.

Un contenedor se puede definir como un entorno ligero, aislado y portable, que contiene todo lo necesario (código fuente, dependencias, etc.) para ejecutar una aplicación

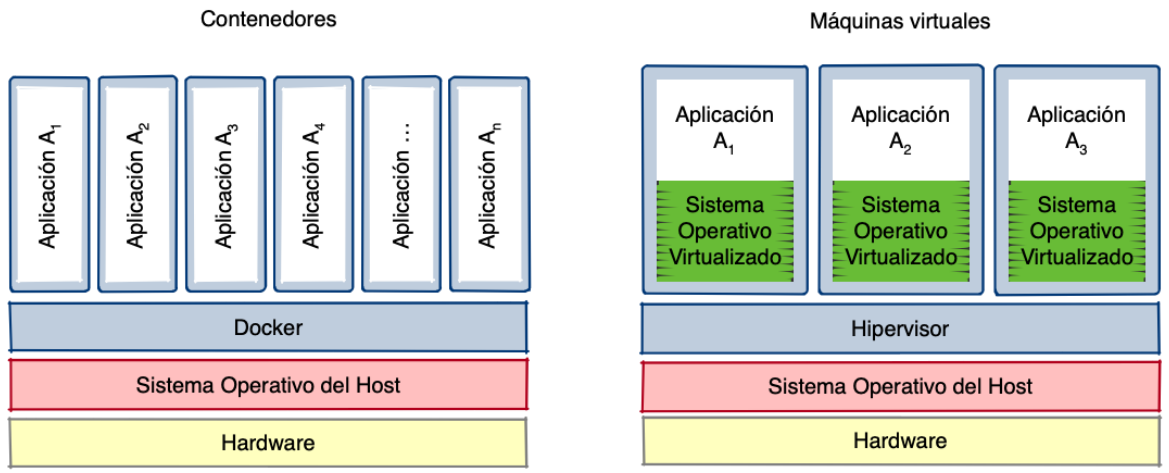
Un contenedor suele tener un único proceso en ejecución, aunque es posible tener varios.

Una de las ventajas que aporta el uso de contenedores es que garantiza que una aplicación se ejecute de la misma manera en cualquier entorno.

Referencias:

- ¿Qué es Docker?.

1.1. Contenedores vs Máquinas Virtuales



1.2. STACK de Contenerización

La siguiente tabla muestra qué lugar ocupa Docker en el stack de contenerización.

	Ejemplos
Plataforma	OpenShift, Docker Enterprise Edition, Rancher, DC/OS
Orquestador	Kubernetes, Docker Swarm, Mesos
Motor de contenerización	Docker, Podman, rkt, LXD, cri-o
Sistema Operativo	Windows, Linux, macOS

1.3. Tecnologías de contenerización

Docker no es la única tecnología de contenerización que existe. A continuación se enumeran algunas de las más conocidas.

- Docker
- Podman
- rkt
- LXD
- cri-o

Referencias:

- ¿Qué es un contenedor de Linux?.

1.4. Orquestador

Entre los orquestadores más conocidos se encuentran:

- Kubernetes
- Docker Swarm
- Mesos

Referencias:

- ¿Qué es Kubernetes?.

1.5. Plataforma

También existen plataformas de contenedores que integran un orquestador y un motor de contenerización. Estas herramientas ofrecen un conjunto de herramientas y servicios para facilitar el despliegue de aplicaciones en contenedores.

- OpenShift
- Docker Enterprise Edition
- Rancher
- DC/OS

1.6. Conceptos básicos

1.6.1. Dockerfile

Es un **archivo de configuración** escrito en texto plano, que contiene todas las instrucciones necesarias para crear una imagen Docker.

Referencias:

- Dockerfile Overview.
- Dockerfile reference.

1.6.2. Imagen

Una imagen es como una plantilla que utilizamos para crear nuestros contenedores.

Podemos decir que las imágenes de Docker son **una instantánea de un contenedor** y que los contenedores se crean a partir de una imagen.

Referencia:

- What is an image?

1.6.3. Contenedor

Un contenedor es una **instancia en ejecución de una imagen** que puede contener uno o más procesos ejecutándose. Para crear un contenedor solo hay que iniciar una imagen con el comando `docker run`.

Referencia:

- What is a container?.

1.6.4. Docker Registry

Un Docker Registry o registro de contenedores Docker, es un servicio encargado de almacenar repositorios de imágenes Docker. Un Docker Registry puede ser público o privado.

Docker Hub es el registro oficial donde están alojados los repositorios de las imágenes Docker que podemos utilizar en nuestros contenedores. En Docker Hub pueden existir imágenes públicas y privadas, así como imágenes oficiales y otras que han sido creadas por desarrolladores independientes.

Para realizar la búsqueda de imágenes podemos hacerlo **desde la web oficial**:

https://hub.docker.com

También podemos hacerlo **desde consola** con el comando `docker search`. Por ejemplo, para buscar todas las imágenes que contengan la palabra *ubuntu* usamos el comando:

```
1 docker search ubuntu
```

Referencias:

- What is a registry?.

1.6.5. Volúmenes

Los volúmenes son el mecanismo que utiliza Docker para hacer persistentes los datos en un contenedor Docker.

Referencias:

- Volumes.

1.6.6. Redes

Docker nos permite crear diferentes tipos de redes que permiten a los contenedores comunicarse entre sí y con el exterior del host.

Referencias:

- Networking overview.

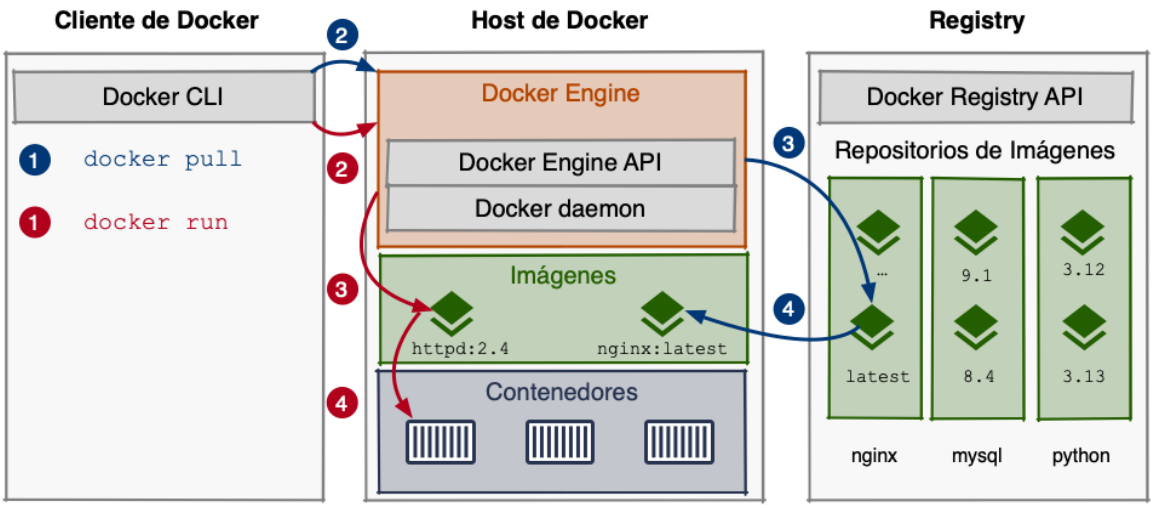
1.6.7. Docker Compose

Docker Compose es una herramienta que nos permite definir y ejecutar aplicaciones Docker con múltiples contenedores. Con Docker Compose podemos definir en un archivo YAML todos los servicios necesarios para una aplicación y gestionarlos todos a la vez con un solo comando.

Referencias:

- Docker Compose overview.

1.7. Fundamentos de la arquitectura de Docker



1.8. Instalación de Docker

Para realizar la instalación de Docker se recomienda seguir la documentación oficial.

Si has instalado Docker sobre Linux, tendrás que realizar alguna configuración adicional. Se recomienda seguir la documentación oficial sobre los pasos que hay seguir tras una instalación de Docker en Linux.

1.9. Imágenes en Docker

En esta sección vamos a ver los comandos básicos para trabajar con imágenes Docker.

1.9.1. `docker search`

Este comando nos permite buscar imágenes en Docker Hub.

Ejemplo:

Por ejemplo, para buscar todas las imágenes que contengan la palabra *ubuntu* usamos el comando:

```
1 docker search ubuntu
```

```
1 NAME                                                 DESCRIPTION
                                       STARS               OFFICIAL
      AUTOMATED
2 ubuntu                                               Ubuntu is a Debian-based
      Linux operating …sys   8763                [OK]
```

```
3 dorowu/ubuntu-desktop-lxde-vnc                          Ubuntu with openssh-
      server and NoVNC            242                                  [OK]
4 ...
```

Ejemplo:

Para buscar todas las imágenes que contengan la palabra *wordpress* ejecutaríamos el siguiente comando.

```
1 docker search wordpress
```

```
1 NAME                DESCRIPTION                          STARS   OFFICIAL
      AUTOMATED
2 wordpress           The WordPress rich content mana...   1983    [OK]
3 bitnami/wordpress   Bitnami Docker Image for WordPress   51                  [OK
      ]
4 ...
```

1.9.2. `docker pull`

Este comando nos permite descargar una imagen de Docker Hub.

Ejemplo:

Por ejemplo, para descargarnos la imagen `ubntu` ejecutaríamos lo siguiente.

```
1 docker pull ubuntu
```

Ejemplo:

Para descargarnos la imagen `wordpress` haríamos lo siguiente.

```
1 docker pull wordpress
```

1.9.3. `docker images`

Muestra un listado con todas las imágenes locales disponibles.

Ejemplo:

Para ver el listado de de las imágenes que tenemos descargadas en nuestro equipo, ejecutaríamos el siguiente comando.

```
1 docker images
```

```
1 REPOSITORY          TAG                 IMAGE ID            CREATED
                SIZE
2 wordpress           latest              fcf3e41b8864        2 weeks ago
            408MB
3 ubuntu              latest              2d696327ab2e        2 months ago
           122MB
```

Ejemplo:

El modificador `-q` nos permite mostrar solamente el identificador de la imagen en el listado de salida. Esta opción nos será de utilidad cuando quiera eliminar todas las máquinas de forma masiva.

```
docker images -q
```

```
fcf3e41b8864
2d696327ab2e
```

1.9.4. `docker rmi`

Este comando nos permite eliminar una o varias imágenes.

Por ejemplo, para eliminar la imagen *wordpress* usamos:

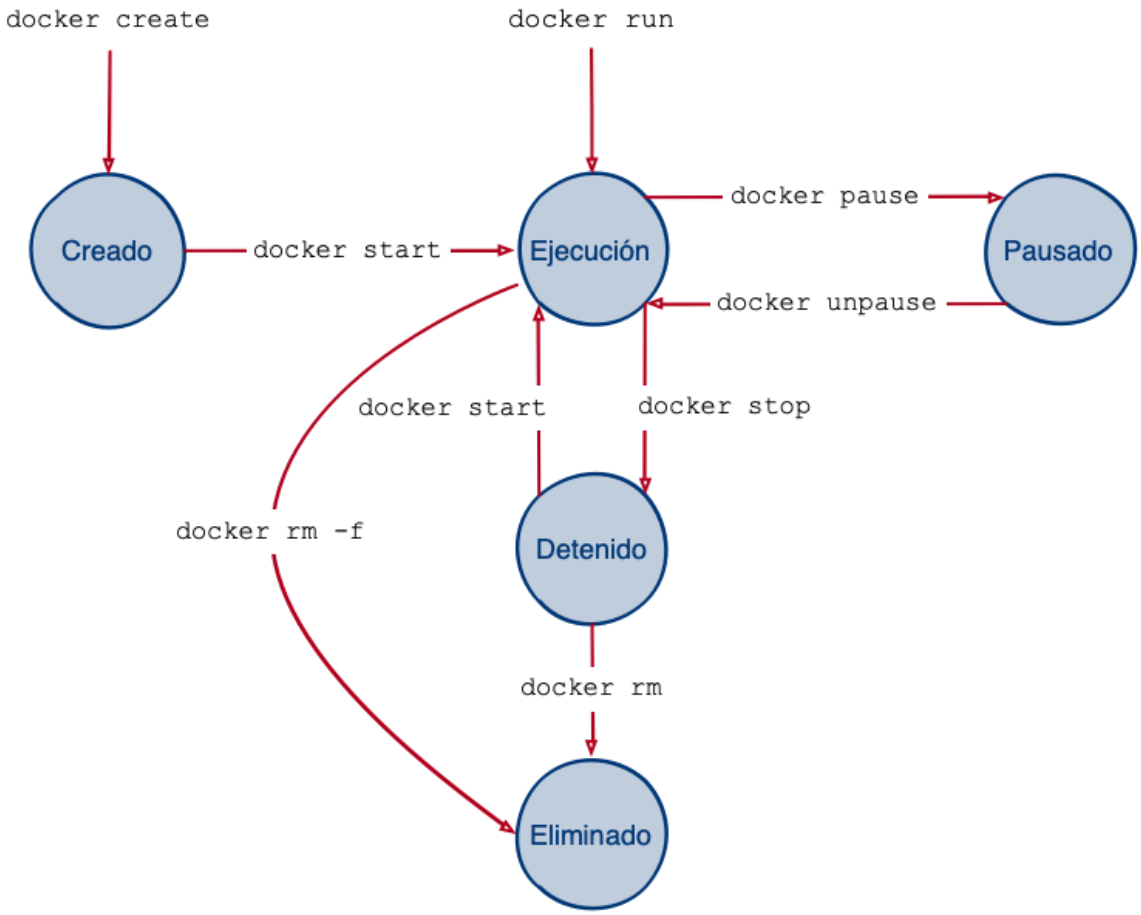
```
docker rmi wordpress
```

1.9.5. `docker rmi $(docker images -q)`

Este comando nos permite eliminar todas las imágenes que tenemos en local.

```
docker rmi $(docker images -q)
```

1.10. Ciclo de vida de un contenedor



1.11. Creación de contenedores en Docker

Para crear contenedores en Docker se utiliza el comando `docker run`.

Existen dos formas de crear un contenedor en Docker:

- `docker run -it`: Crea contenedores en **modo interactivo** que se ejecutan en primer planno y que nos permiten interactuar con ellos a través de la entrada estándar STDIN.
- `docker run -d`: Crea contenedores en **modo detached** con que se ejecutan en segundo plano.

1.12. Creación de contenedores en modo interactivo

En esta sección vamos a ver algunos ejemplos de cómo crear contenedores en modo interactivo.

1.12.1. Creación de un contenedor con Alpine Linux

Alpine Linux es una distribución Linux muy ligera. La imagen de Alpine Linux para Docker ocupa menos de 5 MB.

```
1 REPOSITORY          TAG                 IMAGE ID            CREATED
                SIZE
2 alpine              latest              196d12cf6ab1        2 months ago
            4.41MB
```

El gestor de paquetes de Alpine Linux es `apk`. En la documentación oficial podemos encontrar más detalles sobre cómo usarlo.

Ejemplo:

```
1 docker run -it --name alpine-container --rm alpine
```

- `docker run` es el comando que nos permite crear un contenedor a partir de una imagen Docker.
- El parámetro `-i` nos permite mantener interaccionar con el contenedor a través de la entrada estándar STDIN.
- El parámetro `-t` nos asigna un terminal dentro del contenedor.
- Los dos parámetros `-it` nos permiten usar un contenedor como si fuese una máquina virtual tradicional.
- El parámetro `--name` nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre Docker nos asignará un nombre automáticamente.
- El parámetro `--rm` hace que cuando salgamos del contenedor, éste se elimine y no ocupe espacio en nuestro disco.
- `alpine` es el nombre de la imagen. Si no se indica lo contrario buscará las imágenes en el repositorio oficial Docker Hub.

Una vez ejecutado el comando anterior nos aparecerá un terminal del contenedor que acabamos de crear.

```
1 / #
```

Si quisiéramos instalar `nano` en el contenedor tendríamos que hacer lo siguiente.

1) Actualizar el índice de paquetes disponibles

```
1 apk update
```

2) Añadir el nuevo paquete al sistema.

```
1 apk add nano
```

Para salir del contenedor escribimos el comando `exit`.

```
1 exit
```

Como hemos iniciado el contenedor con el parámetro `--rm`, al salir del contenedor, éste se elimina y no ocupa espacio en nuestro disco. Podemos comprobarlo con siguiente comando.

```
docker ps -a
```

1.12.2. Creación de un contenedor con Ubuntu

```
docker run -it --name ubuntu --rm ubuntu
```

1.13. Creación de contenedores en modo *detached*

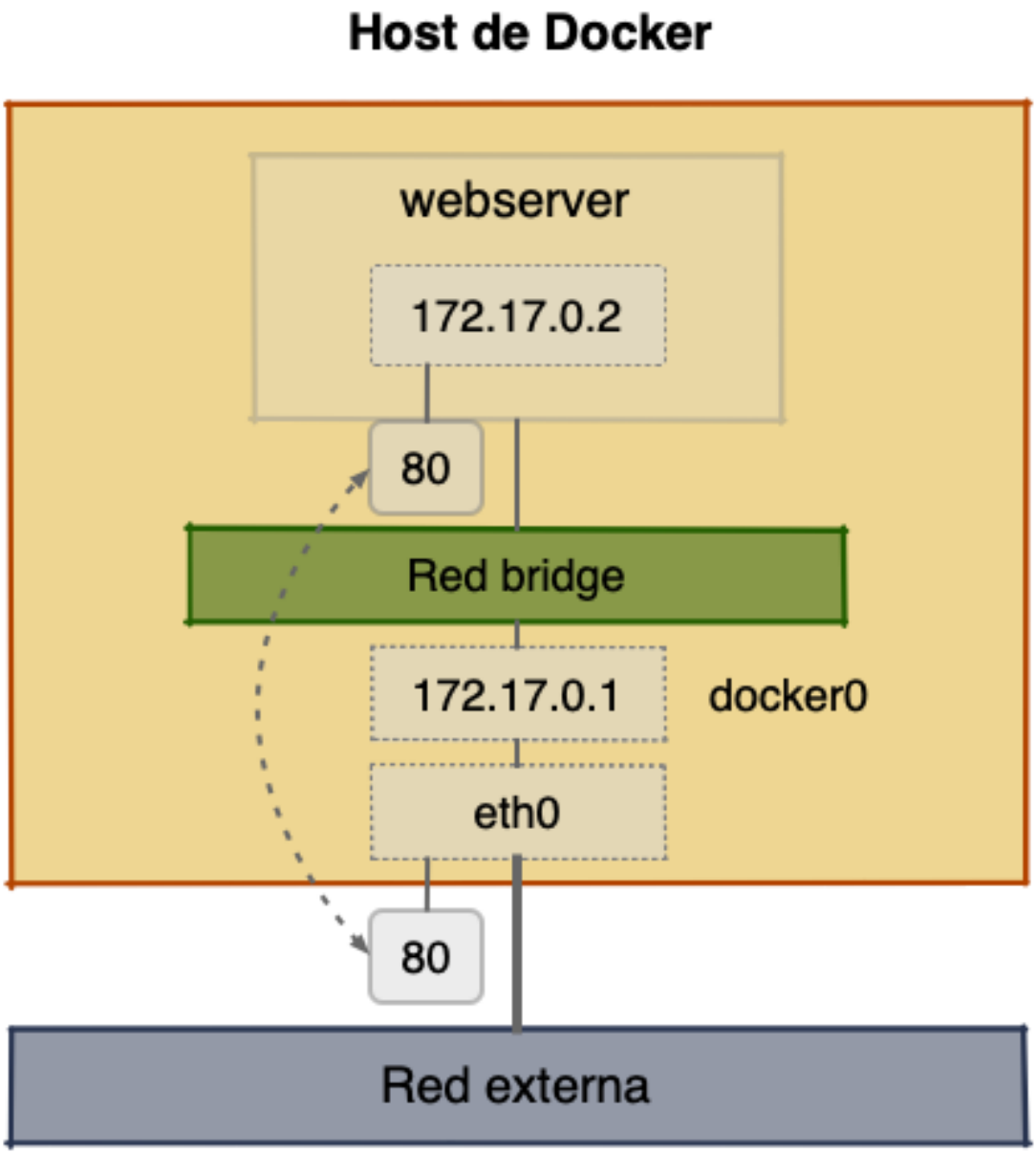
En esta sección vamos a ver algunos ejemplos de cómo crear contenedores en modo *detached*.

1.13.1. Creación de un contenedor con Nginx en modo *detached*

Ejemplo:

```
docker run -d --name webserver --rm -p 80:80 nginx
```

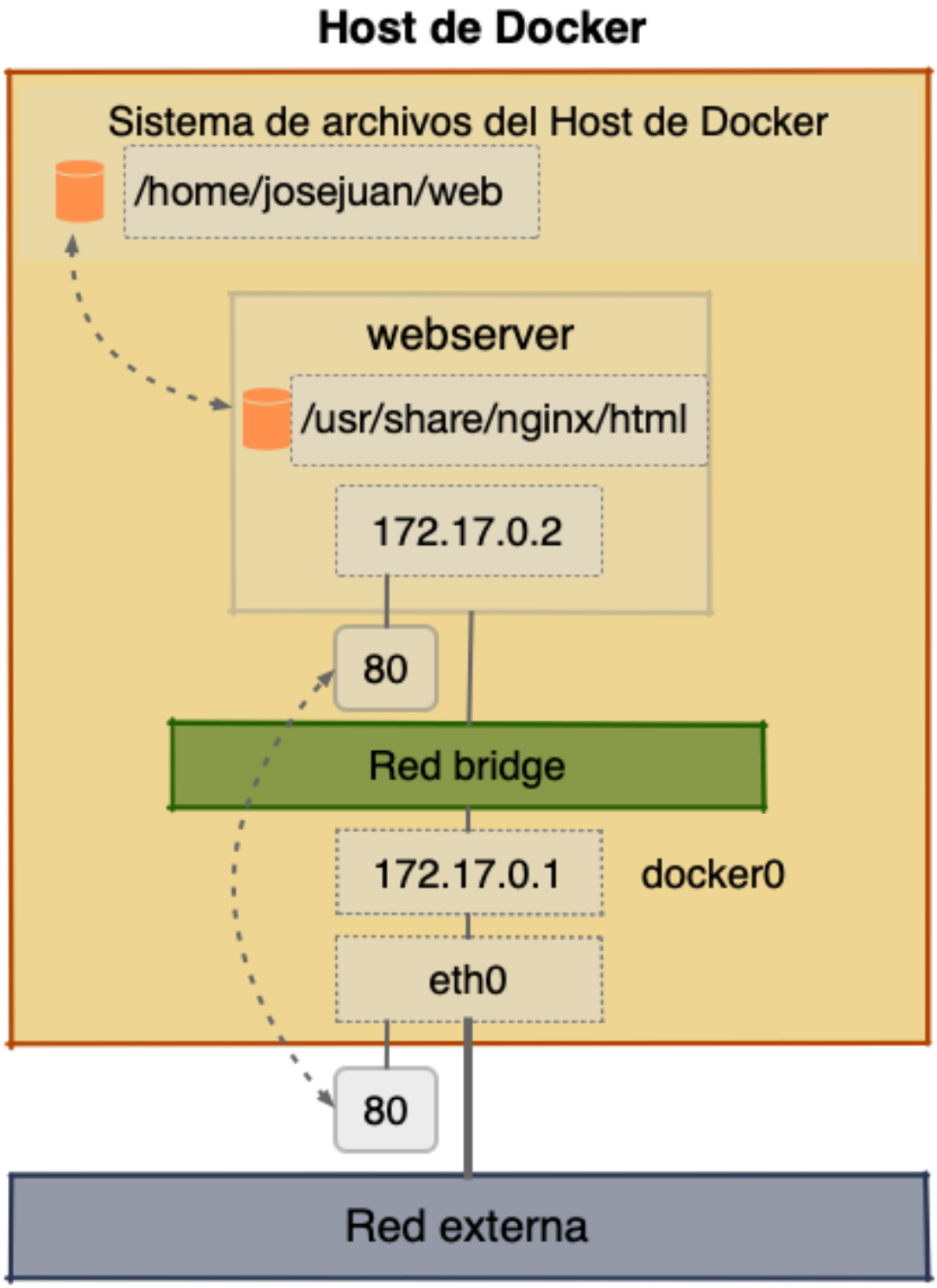
- Con el parámetro `-d` indicamos que queremos ejecutar el contenedor en background.
- Con el parámetro `--name` le asignamos un nombre al contenedor.
- Con el parámetro `--rm` indicamos que queremos que el contenedor se elimine cuando se detenga.
- Con el parámetro `-p` indicamos que queremos mapear el puerto 80 de nuestro equipo con el puerto 80 del contenedor.



Ejemplo:

```
1  docker run -d --name webserver --rm -p 80:80 -v /home/josejuan/web:/usr/share/
       nginx/html nginx
```

Con el parámetro `-v` podemos crear un volumen para mapear un directorio de nuestro equipo con el directorio que utiliza Nginx para servir las páginas webs.


Host de Docker
Sistema de archivos del Host de Docker
/home/josejuan/web
webserver
/usr/share/nginx/html
172.17.0.2
80
Red bridge
172.17.0.1
docker0
eth0
80
Red externa

También podemos hacer uso de `$(pwd)` para indicar que queremos crear un volumen en nuestro directorio actual.

```
docker run -d --name webserver --rm -p 80:80 -v $(pwd):/usr/share/nginx/html
    nginx
```

1.14. Ejecución de comandos en un nuevo contenedor

1.14.1. `docker run`

El comando `docker run` nos permite ejecutar un comando en un contenedor.

Por ejemplo, para ejecutar el comando `cat /etc/os-release` en el contenedor *ubuntu* haríamos lo siguiente.

```
docker run ubuntu cat /etc/os-release
```

Y como salida tendríamos el siguiente resultado.

```
NAME="Ubuntu"
VERSION="18.04.1 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.1 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-
    policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

El contenedor finaliza su ejecución una vez que ha finalizado la ejecución del comando.

1.15. Ejecución de comandos en un contenedor que está en ejecución

1.15.1. `docker exec`

Nos permite ejecutar comandos concretos en un contenedor o abrir un terminal como si fuera una máquina virtual.

Ejemplo:

Permite ejecutar un comando en un contenedor que se está ejecutando.

```
docker exec -it webserver ls -la
```

Ejemplo:

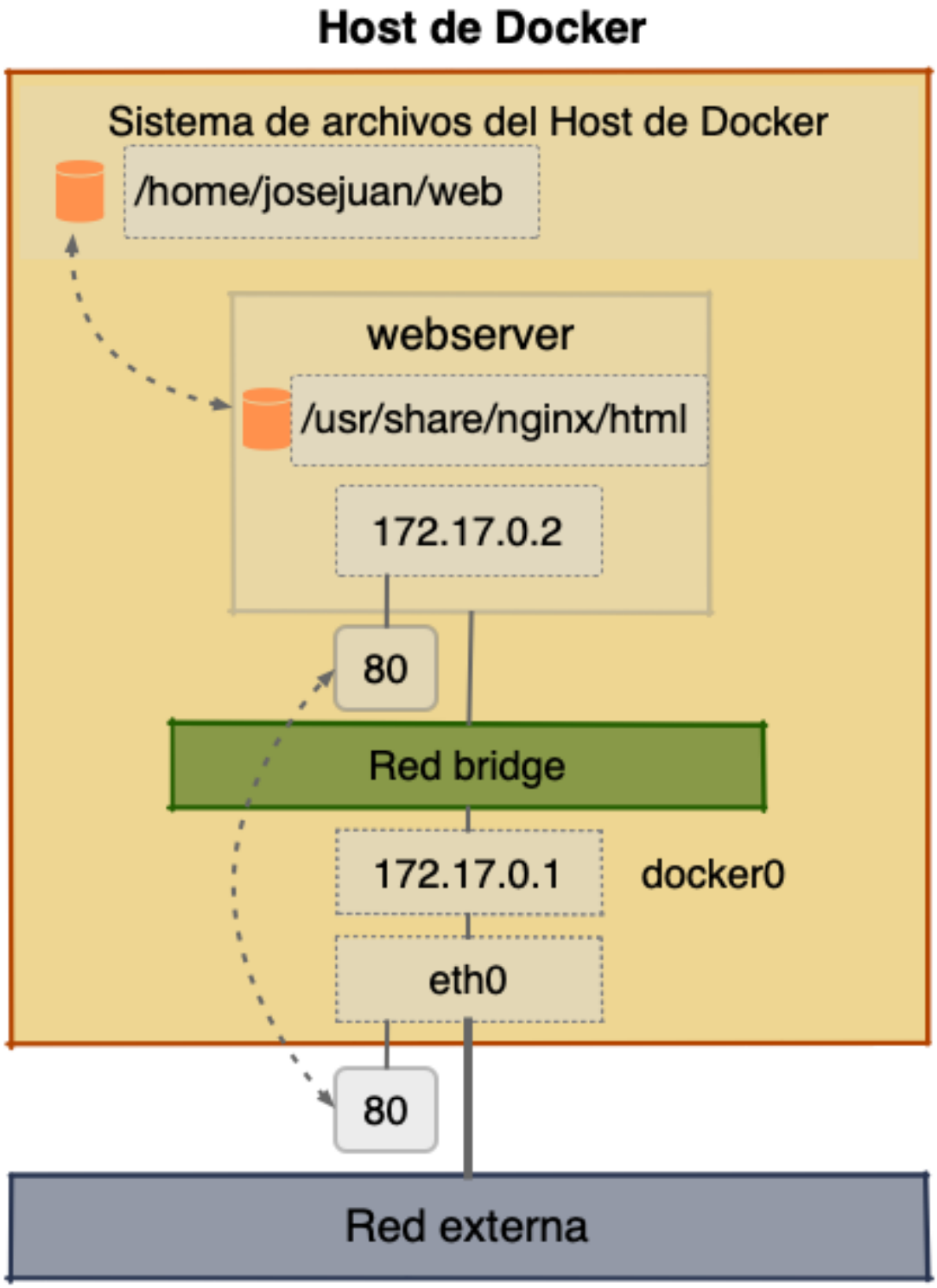
Podemos lanzar como comando `/bin/sh` para abrir una consola e interactuar con el contenedor como si fuera una «máquina virtual».

```
1 docker exec -it webserver /bin/sh
```

1.16. Almacenamiento de datos

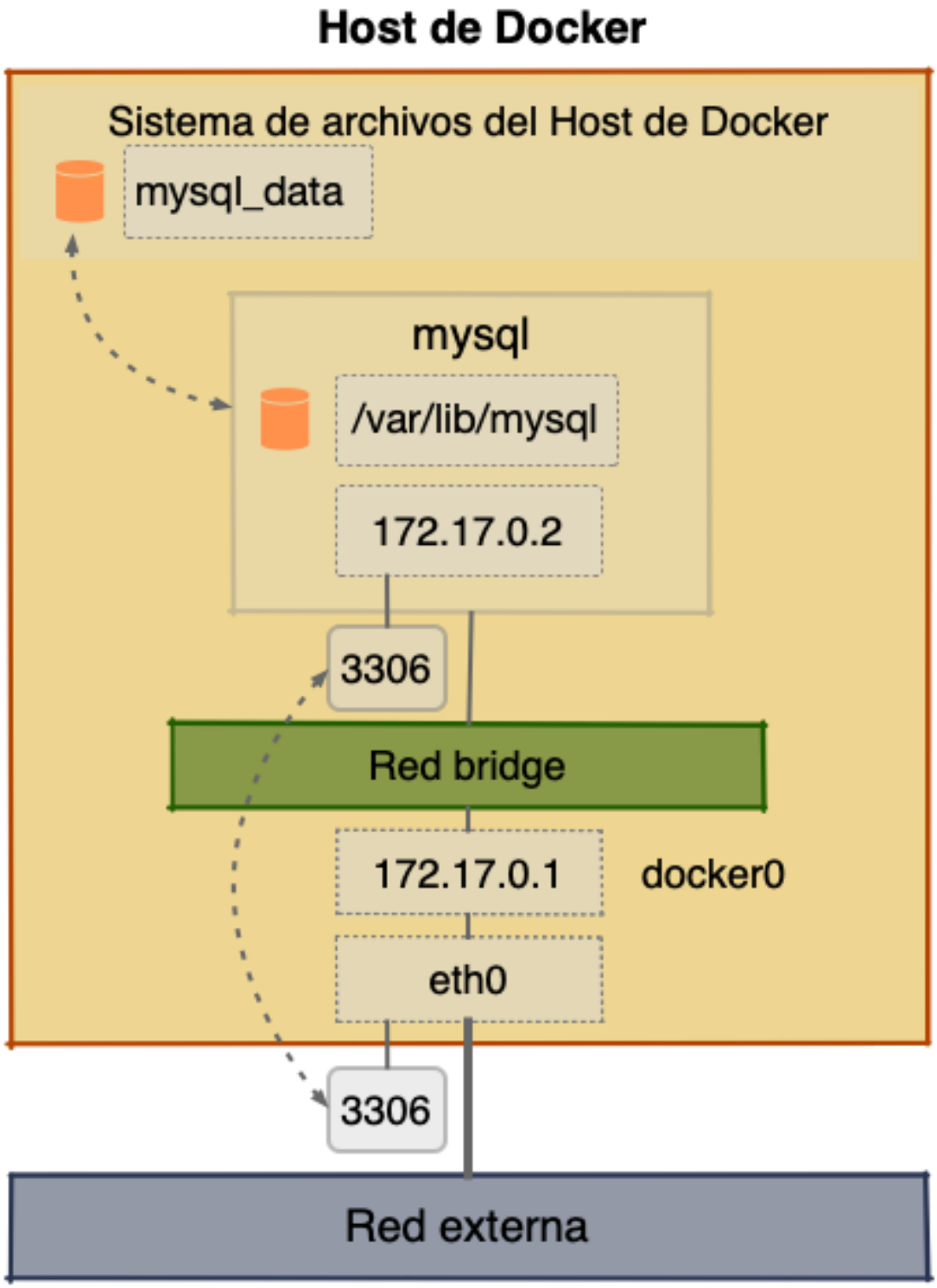
Esta sección está en progreso…

Ejemplo de un *bind mount* gestionado por el usuario:


Host de Docker
Sistema de archivos del Host de Docker
/home/josejuan/web
webserver
/usr/share/nginx/html
172.17.0.2
80
Red bridge
172.17.0.1
docker0
eth0
80
Red externa

```
1 docker run -d --name webserver --rm -p 80:80 -v /home/josejuan/web:/usr/share/
      nginx/html nginx
```

Ejemplo de un *volume* gestionado por Docker:


Host de Docker
Sistema de archivos del Host de Docker
mysql_data
mysql
/var/lib/mysql
172.17.0.2
3306
Red bridge
172.17.0.1
docker0
eth0
3306
Red externa

```
docker run -d --name mysql --rm -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v
    mysql_data:/var/lib/mysql mysql
```

1.17. Administración de contenedores

1.17.1. `docker ps`

Este comando muestra todos los contenedores **que hay en ejecución**.

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED
                STATUS              PORTS               NAMES
```

1.17.2. `docker ps -a`

Muestra todos los contenedores, los que están ejecución y los que están detenidos.

```
docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
                STATUS                      PORTS               NAMES
cfc8008e704b        ubuntu              "/bin/echo 'Hello ..."   7 seconds ago
          Exited (0) 5 seconds ago                              boring_almeida
```

1.17.3. `docker stop`

Permite detener un contenedor que está en ejecución.

En este ejemplo estaría deteniendo un contenedor con el id `abc1102e802c`.

```
docker stop abc1102e802c
```

También puedo detener todos los contenedores que hay en ejecución con el siguiente comando.

```
docker stop $(docker ps -a -q)
```

1.17.4. `docker start`

Permite iniciar un contenedor que está detenido.

```
docker start <ID | NAME>
```

1.17.5. `docker rm`

Para eliminar un contenedor **que no está en ejecución** referenciado por el nombre `wordpress` usamos:

```
docker rm wordpress
```

También podemos eliminarlo indicando su id. Por ejemplo:

```
docker rm 99ed74b743ec
```

Para eliminar todos los contenedores que no están ejecución.

```
docker rm $(docker ps -aq)
```

1.17.6. `docker rm -f`

Para eliminar un contenedor que está en ejecución tenemos que usar el modificador `-f`.

```
docker rm -f wordpress
```

Para eliminar todos los contenedores, aunque estén en ejecución:

```
docker rm -f $(docker ps -aq)
```

1.17.7. `docker logs`

Muestra información de log de un contenedor.

```
docker logs <ID | NAME>
```

Para mostrar la información del log de un contenedor en tiempo real.

```
docker logs -f <ID | NAME>
```

1.17.8. `docker inspect`

Muestra información de bajo nivel de una imagen o un contenedor.

```
docker inspect <ID | NAME>
```

2 Dockerfile

Es un **archivo de configuración** escrito en texto plano, que contiene todas las instrucciones necesarias para crear una imagen Docker.

2.1. Instrucciones

Las instrucciones más habituales que podemos usar en un archivo `Dockerfile` son:

- `FROM`: Indica la imagen base que vamos a utilizar. Primero buscará la imagen en local y si no la encuentra la descargará de un registry.
- `LABEL`: Permite añadir metatados a una imagen, como el nombrel del autor, la versión, o una descripción. Podemos añadir tantas etiquetas como queramos.
- `RUN`: Permite definir los comandos que queremos que se ejecuten sobre la imagen base. En una instrución `RUN` se pueden ejecutar varios comandos de forma encadenada. Cada instrucción `RUN` que aparezca en el archivo `Dockerfile` añadirá una nueva capa sobre la imagen base.
- `ADD`: Añade un archivo o un directorio al contenedor. Puede descargar un archivo de una URL y puede descomprimir automáticamente archivos `.tar` y `.tar.gz`.
- `COPY`: Copia archivos o directorios desde la máquina donde estamos creando la imagen al contenedor.
- `ENV`: Nos permite configurar variables de entorno dentro del contenedor. Los valores por defecto de estas variables se pueden sobrescribir con la opción `-e` o `-env` al iniciar el contenedor. Ejemplo: `docker run -e CLAVE=valor`.
- `EXPOSE`: Indica que el contenedor utiliza los puertos especificados durante su ejecución. Esta instrucción no publica los puertos, es sólo de carácter informativo. Sólo se usará si se inicia el contenedor con la opción `-P`.
- `WORKDIR`: Indica el directorio de trabajo donde se ejecutarán los comandos que se indiquen con las directivas `RUN`, `CMD`, `ENTRYPOINT`, `COPY`y `ADD`.
- `VOLUME`: Esta instrución permite crear un volumen donde el contenedor puede almacenar datos de forma persistente en el host donde se está ejecutando.
- `CMD`: Nos permite indicar cuál será la instrucción que se ejecute por defecto al iniciar el contenedor. Solo puede existir una instrucción `CMD` en un `Dockerfile`, si tenemos más de una sólo se ejecutará la última. El comando que se defina con `CMD` se puede sobrescribir al iniciar el contenedor.

  Existen tres tipos de sintaxis para `CMD`:

  - `CMD ["executable", "param1", "param2"]`. El comando se ejecuta como un proceso sin utilizar el shell (*exec form*).
  - `CMD ["param1", "param2"]`. Se indican los parámetros que se pasarán al `ENTRYPOINT`.
  - `CMD command param1 param2`. El comando se ejecuta en un shell (*shell form*).

- `ENTRYPOINT`: Nos permite indicar cuál será el comando que se ejecute por defecto al iniciar el contenedor. Solo puede existir una instrucción `ENTRYPOINT` en un `Dockerfile`, si tenemos más de una sólo se ejecutará la última.

  Si el archivo `Dockerfile` también contiene la instrucción `CMD`, entonces el valor de `CMD` se usará como parámetro para `ENTRYPOINT`.

  El comando que se defina en el `ENTRYPOINT` se puede sobrescribir al iniciar el contenedor indicándolo de forma explícita con la opción `--entrypoint`.

  Existen dos tipos de sintaxis para `ENTRYPOINT`:

  - `ENTRYPOINT ["executable", "param1", "param2"]`. El comando se ejecuta como un proceso sin utilizar el shell (*exec form*).
  - `ENTRYPOINT command param1 param2`. El comando se ejecuta en un shell (*shell form*).

Referencias:

- Dockerfile Overview.
- Dockerfile reference.

2.2. Ejemplos

Los ejemplos que se muestran a continuación están disponibles en este repositorio de GitHub:

- https://github.com/josejuansanchez/docker-playground/

Ejemplo 1:

Ejemplo de un archivo `Dockerfile` que muestra un mensaje por pantalla utilizando el comando `echo`.

```
1  FROM ubuntu:24.04
2
3  CMD echo "Hello World!"
```

En la instrucción `CMD` podríamos haber utilizado cualquiera de las tres formas de sintaxis disponibles:

- `CMD echo "Hello World!"`
- `CMD ["echo", "Hello World!"]`
- `CMD ["/bin/bash", "-c", "echo Hello World!"]`

Ejemplo 2:

Ejemplo de una imagen Docker que ejecuta un script de **Python**.

Contenido del archivo `hello.py`.

```python
print("Hola mundo!")
```

Contenido del archivo `Dockerfile`.

```
FROM python:3.13-alpine

WORKDIR /app

COPY hello.py /app

CMD ["python", "hello.py"]
```

Ejemplo 3:

Ejemplo de una imagen que ejecuta un servidor **Express** en **Node.js**.

Contenido del archivo `server.js`.

```javascript
const express = require("express");
const app = express();

const PORT = 3000;

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

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
}).on("error", (err) => {
    console.error("Failed to start server:", err);
});
```

Contenido del archivo `Dockerfile`.

```
# Definimos cuál será la imagen base
FROM node:18-alpine

# Configuramos el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copiamos los archivos package*.json que contienen las dependencias
COPY package*.json ./

# Instalamos las dependencias
RUN npm install

# Copiamos el código de la aplicación del host al contenedor
COPY . .

# Indicamos el puerto que usará la aplicación por defecto
EXPOSE 3000

# Iniciamos la aplicación
CMD ["node", "server.js"]
```

Ejemplo 4:

En este ejemplo vamos a clonar un repositorio de GitHub que contiene un sitio web estático y lo vamos a servir con un servidor **Nginx**.

```
FROM ubuntu:24.04

RUN apt update && \
  apt install nginx -y && \
  apt install git -y && \
  rm -rf /var/lib/apt/lists/*

RUN git clone https://github.com/josejuansanchez/lab-cep-awesome-docker/ /app \
  && cp -R /app/site/* /var/www/html/

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

2.3. Creación de una imagen Docker a partir del archivo Dockerfile

Para crear una imagen de Docker a partir del archivo `Dockerfile` deberá situarse dentro del directorio donde se encuentra el archivo `Dockerfile` y ejecutar el siguiente comando.

```
docker build -t mi-imagen .
```

Con el parámetro `-t` indicamos el nombre y el tag que le vamos a asignar. En este ejemplo, `mi-imagen` es el nombre que le vamos a asignar a la imagen. Si no le indicamos ningún tag, por defecto se le asignará el tag `latest`.

El comando anterior es equivalente a:

```
docker build -t mi-imagen:latest .
```

Para comprobar que la imagen se ha creado correctamente podemos ejecutar el comando:

```
docker images
```

Para publicar la imagen en Docker Hub es necesario que en el nombre de la imagen aparezca **nuestro nombre de usuario de Docker Hub**. Por ejemplo, si mi nombre de usuario es `josejuansanchez` la imagen debería llamarse `josejuansanchez/mi-imagen`.

También es una buena práctica asignarle una etiqueta a la imagen. Por ejemplo, en este caso vamos a asignarle las etiquetas `1.0` y `latest`.

```
docker tag mi-imagen josejuansanchez/mi-imagen:1.0
```

```
docker tag mi-imagen josejuansanchez/mi-imagen:latest
```

Comprobamos que la imagen tiene el nombre y las etiquetas correctas:

```
docker images
```

2.4. Publicar la imagen en Docker Hub

Una vez que le hemos asignado un nombre correcto a la imagen y le hemos añadido las etiquetas, podemos publicarla en Docker Hub.

En primer lugar, tendremos que iniciar la sesión en Docker Hub con el comando:

```
docker login
```

Para iniciar la sesión nos preguntará el nombre de usuario y la contraseña de Docker Hub. En lugar de introducir nuestra contraseña podemos crear un token en Docker Hub y utilizarlo para iniciar la sesión.

Una vez iniciada la sesión, podemos publicar la imagen con el comando `docker push`. Tenemos que publicar la imagen con las dos etiquetas que hemos creado.

```
docker push josejuansanchez/mi-imagen:1.0
```

```
docker push josejuansanchez/mi-imagen:latest
```

3 Docker Compose

Docker Compose es una herramienta que nos permite definir y ejecutar aplicaciones Docker con múltiples contenedores. Con Docker Compose podemos definir en un archivo `YAML` todos los servicios necesarios para una aplicación y gestionarlos todos a la vez con un solo comando.

Referencias:

- Docker Compose overview.

3.1. Comandos básicos

- `docker compose up`. Crea e inicia los contenedores.
- `docker compose up -d`. Crea e inicia los contenedores en modo *detach*.
- `docker compose down`. Detiene los contenedores que están en ejecución.
- `docker compose down -v`. Detiene los contenedores que están en ejecución y elimina los volúmenes.
- `docker compose ps`. Muestra los contenedores que están ejecución.
- `docker compose ps -a`. Muestra todos los contenedores incluyendo los que están detenidos.
- `docker compose logs`. Muestra las últimas líneas de los archivos de *logs* de los contenedores.
- `docker compose logs -f`. Muestra los *logs* de los contenedores en tiempo real.
- `docker compose exec`. Permite ejecutar un comando dentro de un contenedor.
- `docker compose start`. Inicia los contenedores que están detenidos.
- `docker compose stop`. Detiene los contenedores que están en ejecución.
- `docker compose build`. Reconstruye los contenedores.

3.2. Ejemplos

En el siguiente repositorio de GitHub puede encontrar una gran variedad de ejemplos sobre cómo usar Docker Compose.

- https://github.com/josejuansanchez/docker-compose-playground/

4 Referencias

- The Docker Book. James Turnbull.
- Get started with Docker.
- Tutorial de Docker basado en el libro «Docker Cookbook» de O'Reilly.
- Blog de Carlos Milán. Carlos Milán.
- Meet Docker.
- Docker for beginners.
- Play with Docker Classroom.
- Cursos de Docker en Katacoda.
- Documentación oficial de Docker.
- Docker. Una nueva forma de ejecutar y desarrollar aplicaciones. Manolo Torres.
- Awesome Docker. A curated list of Docker resources and projects.
- Valuable Docker Links.
- Tutorial labs and Library references. Docker.

5 Licencia