

---

# **Docker**

Implantación de Aplicaciones Web

José Juan Sánchez Hernández

Curso 2024/2025

# Índice

<b>1 Docker</b>	<b>1</b>
1.1 Contenedores vs Máquinas Virtuales	1
1.2 STACK de Contenerización	1
1.3 Tecnologías de contenerización	2
1.4 Orquestador	2
1.5 Plataforma	2
1.6 Conceptos básicos	3
1.6.1 Dockerfile	3
1.6.2 Imagen	3
1.6.3 Contenedor	3
1.6.4 Docker Registry	3
1.6.5 Volúmenes	4
1.6.6 Redes	4
1.6.7 Docker Compose	4
1.7 Fundamentos de la arquitectura de Docker	5
1.8 Instalación de Docker	5
1.9 Imágenes en Docker	5
1.9.1 docker search	5
1.9.2 docker pull	6
1.9.3 docker images	6
1.9.4 docker rmi	7
1.9.5 docker rmi \$(docker images -q)	7
1.10 Ciclo de vida de un contenedor	8
1.11 Creación de contenedores en Docker	8
1.12 Creación de contenedores en modo interactivo	8
1.12.1 Creación de un contenedor con Alpine Linux	9
1.12.2 Creación de un contenedor con Ubuntu	10
1.12.3 Creación de un contenedor con Nginx	10
1.13 Creación de contenedores en modo <i>detached</i>	10
1.13.1 Creación de un contenedor con Nginx en modo <i>detached</i>	10
1.13.2 Creación de un contenedor con MySQL en modo <i>detached</i>	10
1.13.3 Creación de un contenedor con Adminer en modo <i>detached</i>	11
1.13.4 Creación de un contenedor con PostgreSQL en modo <i>detached</i>	11
1.13.5 Creación de un contenedor con pgadmin4 en modo <i>detached</i>	11
1.14 Ejecución de comandos en un nuevo contenedor	11
1.14.1 docker run	11

---

1.15	Ejecución de comandos en un contenedor que está en ejecución . . . . .	12
1.15.1	docker exec . . . . .	12
1.16	Administración de contenedores . . . . .	12
1.16.1	docker ps . . . . .	12
1.16.2	docker ps -a . . . . .	12
1.16.3	docker stop . . . . .	13
1.16.4	docker start . . . . .	13
1.16.5	docker rm . . . . .	13
1.16.6	docker rm -f . . . . .	13
1.16.7	docker logs . . . . .	14
1.16.8	docker inspect . . . . .	14
<b>2</b>	<b>Dockerfile</b>	<b>15</b>
<b>3</b>	<b>Docker Compose</b>	<b>17</b>
3.1	Comandos básicos . . . . .	17
<b>4</b>	<b>Referencias</b>	<b>18</b>
<b>5</b>	<b>Licencia</b>	<b>19</b>

## **Índice de figuras**

## Índice de cuadros

# 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

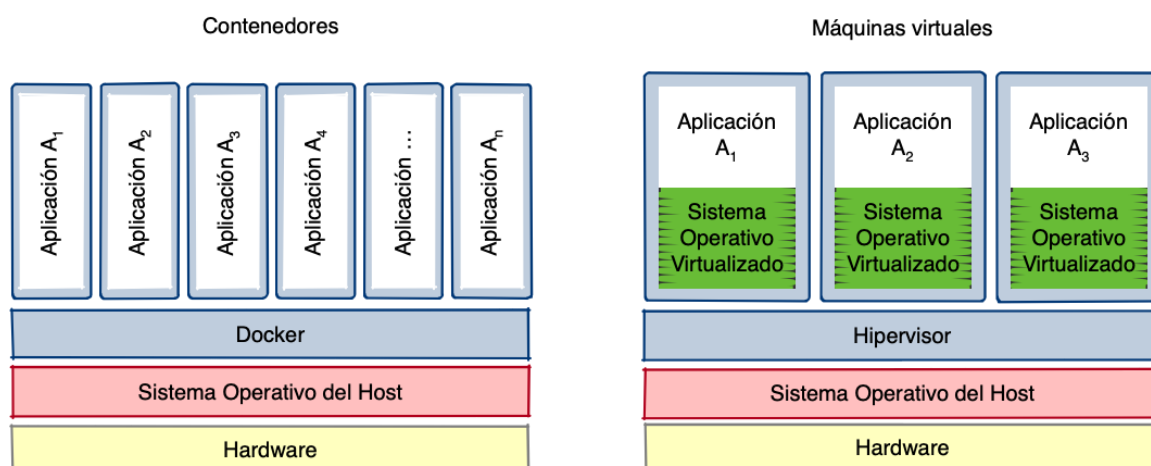
Un contenedor suele tener un único proceso en ejecución, aunque es posible tener varios.

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
<b>Plataforma</b>	OpenShift, Docker Enterprise Edition, Rancher, DC/OS
<b>Orquestador</b>	Kubernetes, Docker Swarm, Mesos
<b>Motor de contenerización</b>	Docker, Podman, rkt, LXD, cri-o
<b>Sistema Operativo</b>	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](#).

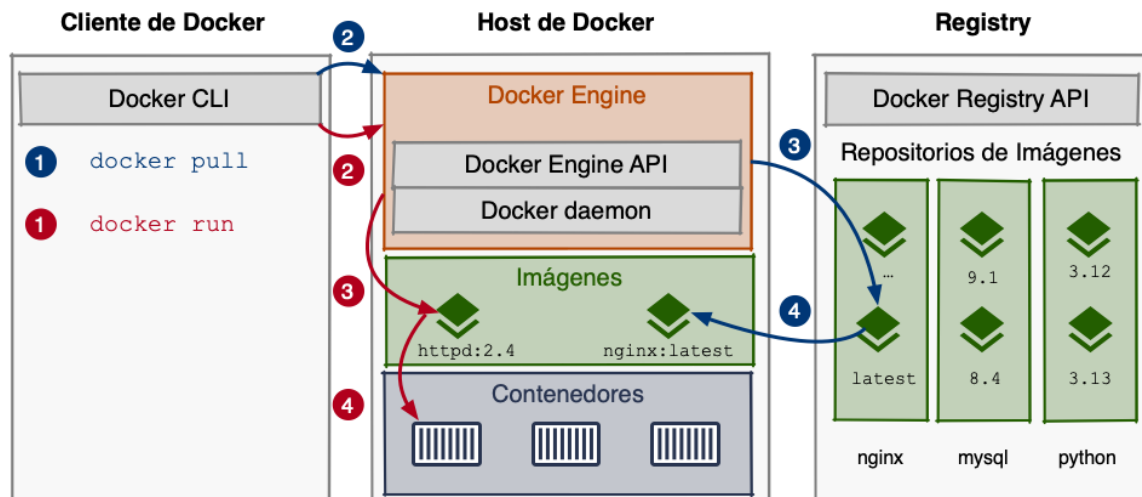
### 1.6.7 Docker Compose

[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	STARS	DESCRIPTION
	AUTOMATED		OFFICIAL
2	ubuntu		Ubuntu is a Debian-based
	Linux operating ...sys	8763	[OK]

3	dorowu/ubuntu-desktop-lxde-vnc	242	Ubuntu with openssh-server and NoVNC	[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 <b>for</b> 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 *ubuntu* 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.

```
1 docker images -q
```

```
1 fcf3e41b8864
2 2d696327ab2e
```

**1.9.4 docker rmi**

Este comando nos permite eliminar una o varias imágenes.

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

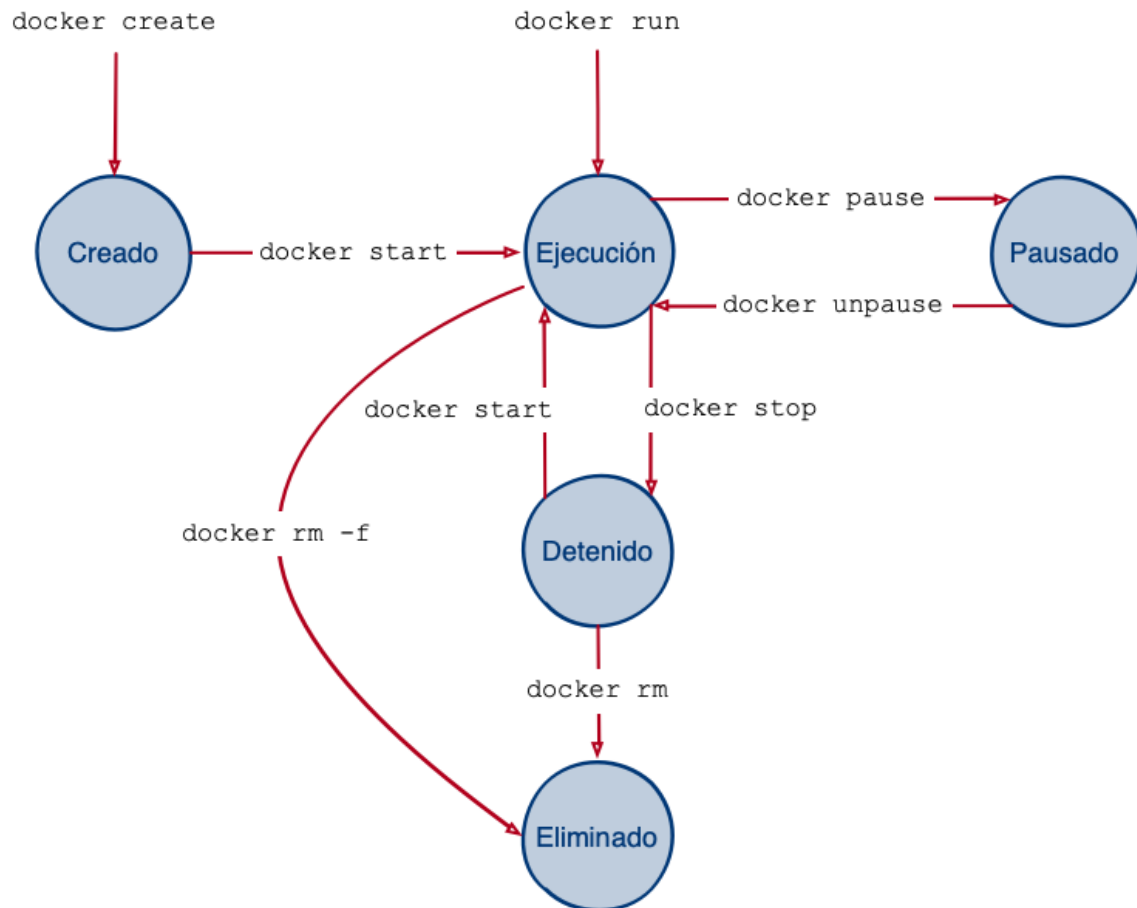
```
1 docker rmi wordpress
```

**1.9.5 docker rmi \$(docker images -q)**

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

```
1 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 plano 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
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 interacción 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.

```
1 docker ps -a
```

### 1.12.2 Creación de un contenedor con Ubuntu

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

### 1.12.3 Creación de un contenedor con Nginx

```
1 docker run -it --name webserver --rm -p 80:80 nginx
```

## 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*

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

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

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

### 1.13.2 Creación de un contenedor con MySQL en modo *detached*

```
1 docker run -d --name mysql --rm -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v /  
  home/josejuan/data:/var/lib/mysql mysql:5.7.22
```

Podemos hacer uso de `$(pwd)` para indicar que queremos crear el volumen en nuestro directorio actual.

```
1 docker run -d --name mysql --rm -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v $(  
  pwd):/var/lib/mysql mysql:5.7.22
```

### 1.13.3 Creación de un contenedor con Adminer en modo *detached*

En primer lugar debe existir un contenedor con MySQL Server.

```
1 docker run --name mysql --rm -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root -v $(pwd)
  :/var/lib/mysql mysql:5.7.22
```

Una vez que la instancia de MySQL está en ejecución podemos crear el contenedor con Adminer.

```
1 docker run --link mysql:db -p 8080:8080 adminer
```

### 1.13.4 Creación de un contenedor con PostgreSQL en modo *detached*

```
1 docker run -d --name postgresql --rm -p 5432:5432 -e POSTGRES_PASSWORD=
  mysecretpassword postgres
```

Nota: El nombre de usuario para conectar con el servidor PostgreSQL es `postgres`.

### 1.13.5 Creación de un contenedor con pgadmin4 en modo *detached*

Este contenedor lanza pgAdmin 4, una aplicación web que nos permite administrar una base de datos PostgreSQL.

```
1 docker run -p 80:80 \
2 -e "PGADMIN_DEFAULT_EMAIL=user@domain.com" \
3 -e "PGADMIN_DEFAULT_PASSWORD=SuperSecret" \
4 -d dpage/pgadmin4
```

## 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.

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

Y como salida tendríamos el siguiente resultado.

```
1 NAME="Ubuntu"
2 VERSION="18.04.1 LTS (Bionic Beaver)"
3 ID=ubuntu
4 ID_LIKE=debian
5 PRETTY_NAME="Ubuntu 18.04.1 LTS"
6 VERSION_ID="18.04"
```



```
7 HOME_URL="https://www.ubuntu.com/"
8 SUPPORT_URL="https://help.ubuntu.com/"
9 BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
10 PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
11 VERSION_CODENAME=bionic
12 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.

```
1 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 Administración de contenedores

### 1.16.1 docker ps

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

```
1 docker ps
```

1	CONTAINER ID	IMAGE	COMMAND	CREATED
		STATUS	PORTS	NAMES

### 1.16.2 docker ps -a

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

```
1 docker ps -a
```

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

### 1.16.3 docker stop

Permite detener un contenedor que está en ejecución.

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

```
1 docker stop abc1102e802c
```

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

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

### 1.16.4 docker start

Permite iniciar un contenedor que está detenido.

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

### 1.16.5 docker rm

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

```
1 docker rm wordpress
```

También podemos eliminarlo indicando su id. Por ejemplo:

```
1 docker rm 99ed74b743ec
```

Para eliminar todos los contenedores que no están ejecución.

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

### 1.16.6 docker rm -f

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

```
1 docker rm -f wordpress
```

Para eliminar todos los contenedores, aunque estén en ejecución:

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

### 1.16.7 docker logs

Muestra información de log de un contenedor.

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

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

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

### 1.16.8 docker inspect

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

```
1 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.

Ejemplo:

```
1 # Pull base image.
2 FROM ubuntu:24.04
3
4 # Install Nginx.
5 RUN \
6     add-apt-repository -y ppa:nginx/stable && \
7     apt-get update && \
8     apt-get install -y nginx && \
9     rm -rf /var/lib/apt/lists/* && \
10    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
11    chown -R www-data:www-data /var/lib/nginx
12
13 # Define mountable directories.
14 VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/log/nginx", "/var/www/html"]
15
16 # Define working directory.
17 WORKDIR /etc/nginx
18
19 # Define default command.
20 CMD ["nginx"]
21
22 # Expose ports.
23 EXPOSE 80
24 EXPOSE 443
```

Los comandos más habituales en un **Dockerfile** son:

- **FROM**: Indica la imagen que vamos a utilizar. Primero buscará la imagen en local y si no la encuentra la descargará de Internet.
- **MAINTAINER**: Datos de la persona que mantiene el contenedor.
- **RUN**: Ejecuta una instrucción en el contenedor y hace un commit de los resultados.
- **ADD**: Añade un archivo o un directorio al contenedor.
- **ENV**: Nos permite configurar variables de entorno en el contenedor. Pueden ser sustituidas pasando la opción `-e` o `-env` al usar el comando `docker run`. Ejemplo: `docker run -e CLAVE=valor`.
- **EXPOSE**: Indica que el contenedor escucha en los puertos especificados durante su ejecución.

- **CMD**: Solo puede existir una instrucción CMD en un **Dockerfile**, si colocamos más de uno, solo el último tendrá efecto. Esta instrucción nos permite indicar que se ejecuten instrucciones por defecto al iniciar un contenedor.
- **ENTRYPOINT**: Nos permite indicar el comando que queremos que se ejecute de forma indefinida en nuestro contenedor. Si al iniciar un contenedor con **docker run** hacemos uso del parámetro **-entrypoint** podemos omitir los comandos especificados en esta instrucción.

**Ejemplos:**

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

## 3 Docker Compose

### 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 en 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.

#### Ejemplos:

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

## 4 Referencias

- [The Docker Book](#). James Turnbull.
- [Get started with Docker](#).
- [¿Cómo instalar y usar Docker en Ubuntu 16.04?](#).
- [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. Cloud-DI Team - Departamento de Informática UAL.
- [Awesome Docker](#). A curated list of Docker resources and projects.
- [Valuable Docker Links](#).
- [Tutorial labs and Library references](#). Docker.

## 5 Licencia

Esta página forma parte del curso Implantación de Aplicaciones Web de José Juan Sánchez Hernández y su contenido se distribuye bajo una licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.