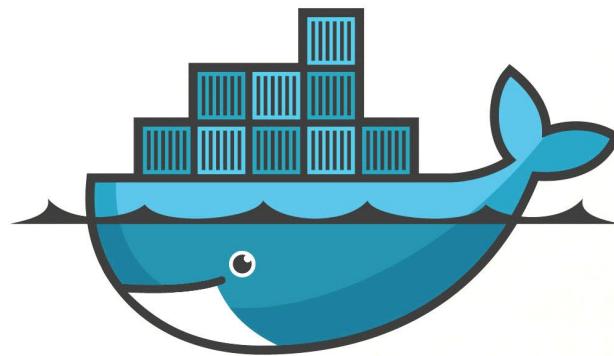


Introducción

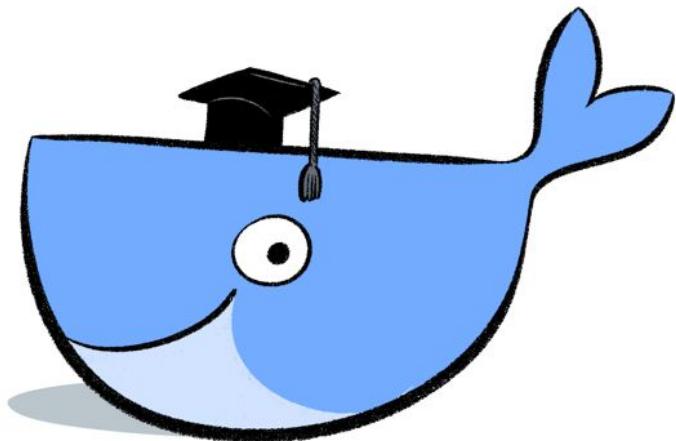


docker

Índice

- ¿Qué es Docker?
- ¿Qué es un contenedor?
- En resumen, ¿Qué es Docker?
- Y entonces, ¿Qué es un contenedor?
- ¿Para qué? Casos de uso.
 - Empaquetamiento de aplicaciones para su distribución.
 - Replicación de entornos de desarrollo. *It works on my machine!!*
 - Replicación de entorno de producción en desarrollo.
 - Despliegue de aplicaciones.
 - Gestión de aplicaciones como un todo. (Redundancia, tolerancia a fallos, etc.)

¿Qué es Docker?



¿Qué es Docker?

*“Docker es un proyecto de código abierto que **automatiza el despliegue de aplicaciones dentro de contenedores** de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.”*

-- Wikipedia (es) --

¿Qué es Docker?

*“Docker es un proyecto de código abierto para **automatizar la implementación de aplicaciones como contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente**.*

Docker es también una empresa que promueve e impulsa esta tecnología, en colaboración con proveedores de la nube, Linux y Windows, incluido Microsoft.”

-- Microsoft --

¿Qué es Docker?

*“Docker Engine is an open source containerization technology **for building and containerizing your applications**. Docker Engine acts as a client-server application with:*

- A server with a long-running daemon process **dockerd**.
- **APIs** which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (**CLI**) client **docker**.“

¿Qué es un contenedor?



¿Qué es un contenedor?

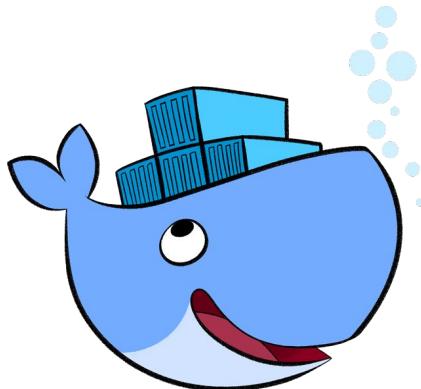
“La **virtualización a nivel de sistema operativo**, también llamada virtualización basada en **contenedores**, (...) es un método de virtualización en el que, sobre el núcleo del sistema operativo se ejecuta una capa de virtualización que permite que existan **múltiples instancias aisladas** de espacios de usuario, en lugar de solo uno...”

-- Wikipedia --

¿Qué es un contenedor?

*“... Tales instancias, las cuales son llamadas **contenedores**, (...), pueden verse (...) como un servidor real (...). Además de mecanismos de **aislamiento**, el kernel a menudo proporciona mecanismos de **administración de recursos** para limitar el impacto de las actividades de un contenedor sobre otros contenedores.”*

Resumen



En resumen:

¿Qué es Docker?

- Es un software de virtualización ligera que permite crear y gestionar contenedores.
- Asigna y reparte el hardware del sistema entre los contenedores.

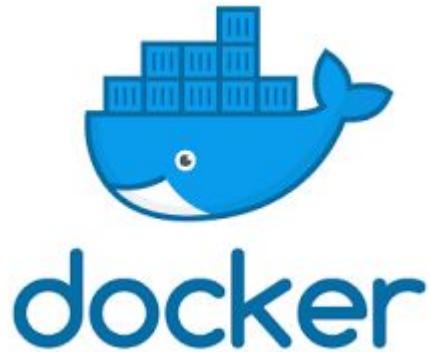
Docker no es una virtualización completa, sino que los contenedores usan el sistema subyacente. Por ejemplo, como Docker funciona sobre el SO Linux, en los contenedores solo puede instalarse Linux.

En resumen:

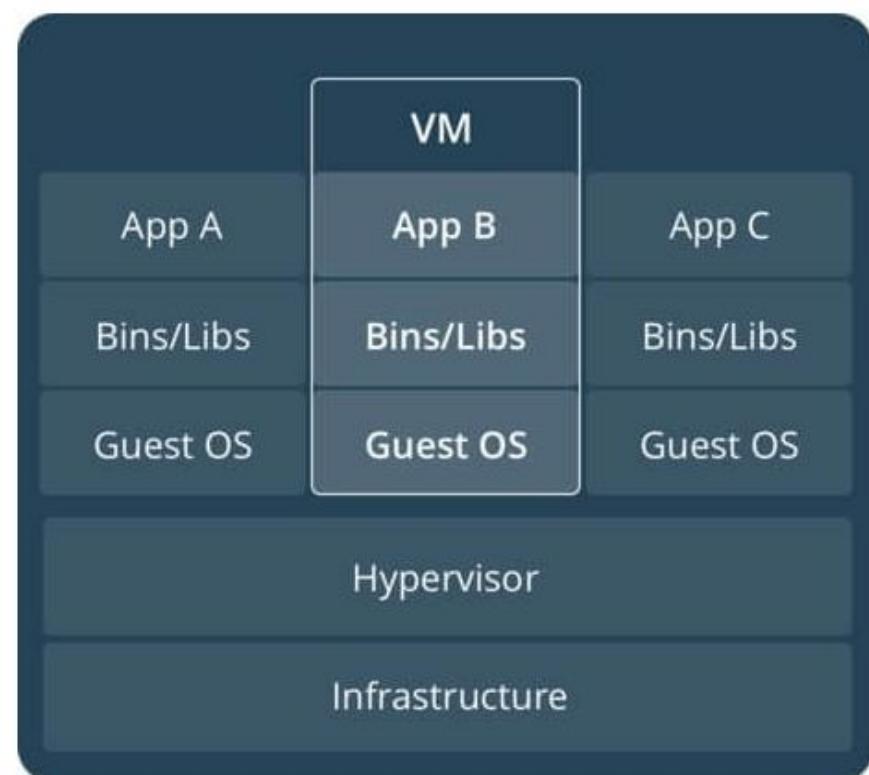
¿Qué es un contenedor?

- Es una máquina virtual ligera.
- Están aislados del resto del sistema en el que se ejecutan.

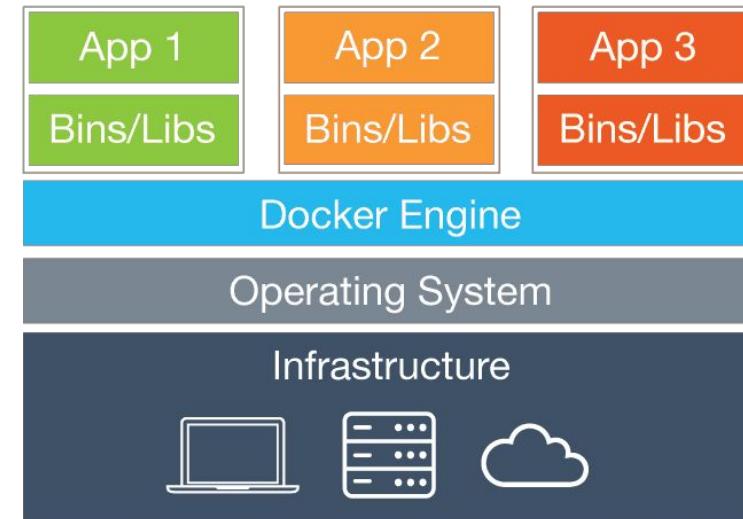
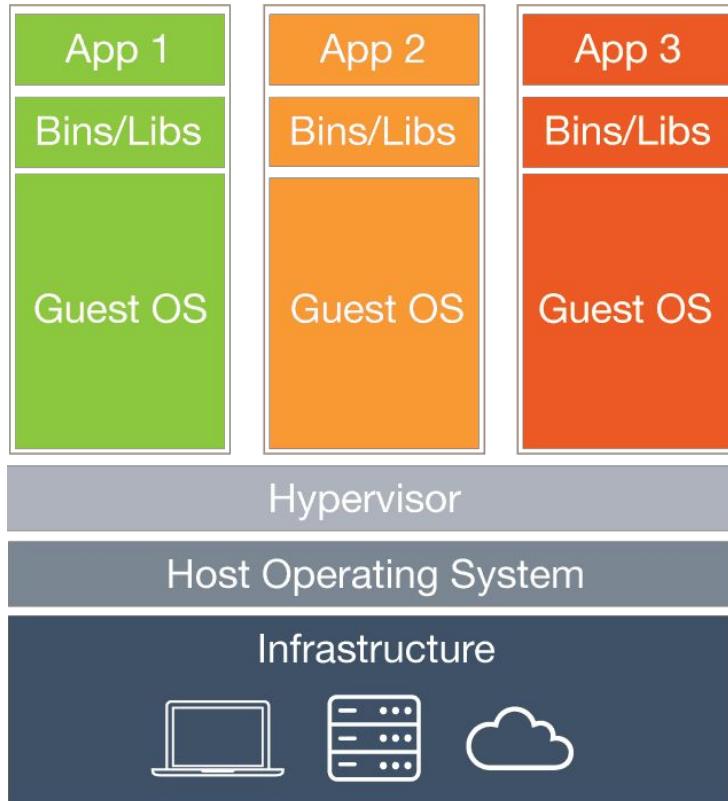
Docker vs Hypervisor



Docker vs Hypervisor



Docker vs Hypervisor



Ventajas de los contenedores frente a las máquinas virtuales

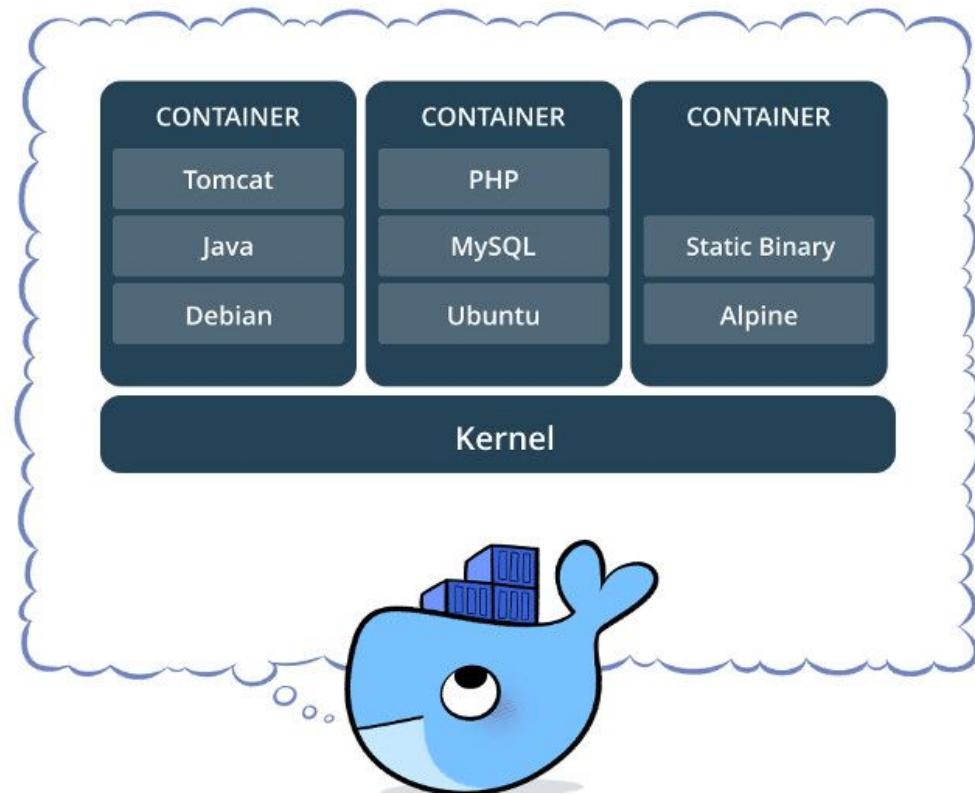
- Ocupan menos espacio.
- Su ejecución es más rápida.
- Son un estándar ampliamente soportado por la industria.

¿Para qué sirve Docker?

Casos de uso

¿Para qué sirve Docker?

Empaquetar aplicaciones



Casos de uso

Vamos a ver 5 ejemplos de casos de uso típicos de Docker:

1. Empaquetamiento de aplicaciones para su distribución.
2. Replicación de entornos de desarrollo.
3. Replicación de entorno de producción en desarrollo.
4. Despliegue de aplicaciones.
5. Gestión de aplicaciones como un todo. (Redundancia, tolerancia a fallos, escalado horizontal, etc.)

Caso 1

Empaquetamiento de aplicaciones
para su distribución



1. Empaquetamiento de aplicaciones para su distribución

Las aplicaciones actuales son **complejas**.

Supongamos que un desarrollador hace **una aplicación web** que quiere distribuir para que sus clientes las puedan utilizar.

La aplicación consta de las siguientes partes/**servicios**:

- | | |
|----------------------------|---------------------------|
| ❑ Servidor web | → Nginx |
| ❑ Backend | → Python + Django |
| ❑ Base de datos | → MySQL |
| ❑ Caché | → Redis |
| ❑ Cola de tareas asíncrona | → Celery |
| ❑ Broker de mensajes | → RabbitMQ |
| ❑ Frontend | → HTML + CSS + Javascript |

1. Empaquetamiento de aplicaciones para su distribución

Si un **cliente** quiere utilizar la aplicación, tendrá que:

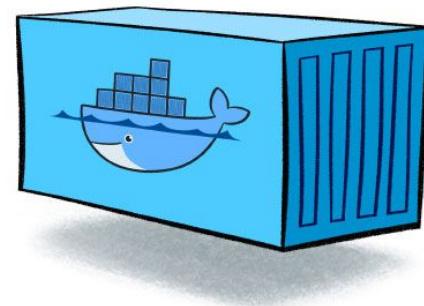
- **Instalar** cada una de las partes por separado, **respetando las versiones** que indica el desarrollador.
- **Configurar** cada una de las partes para que trabajen juntas.

Distribuir esta aplicación es, por tanto, **complicado y requiere unos conocimientos** que el cliente no tiene porqué tener.

1. Empaquetamiento de aplicaciones para su distribución

Utilizando **docker**, el **desarrollador** puede **empaquetar todos los servicios en un contenedor**, de modo que **el cliente solo necesita**:

- Tener **docker instalado**, lo que es muy sencillo, o usar un proveedor en la nube que lo lleve instalado por defecto (todos lo llevan).
- Usar un solo comando para **ejecutar el contenedor**.



Caso 2

Replicación de entornos de desarrollo



2. Replicación de entornos de desarrollo

En la actualidad una aplicación rara vez es desarrollada por 1 único desarrollador, sino por un **grupo**.

Tomando la aplicación web anterior como ejemplo, **cada desarrollador necesitaría instalarse todos los servicios en su máquina local** para poder programar.

2. Replicación de entornos de desarrollo

A la complejidad de la instalación, se une aquí la **dificultad de mantener los servicios y el código fuente actualizados** a las mismas versiones, dando lugar a un problema llamado:

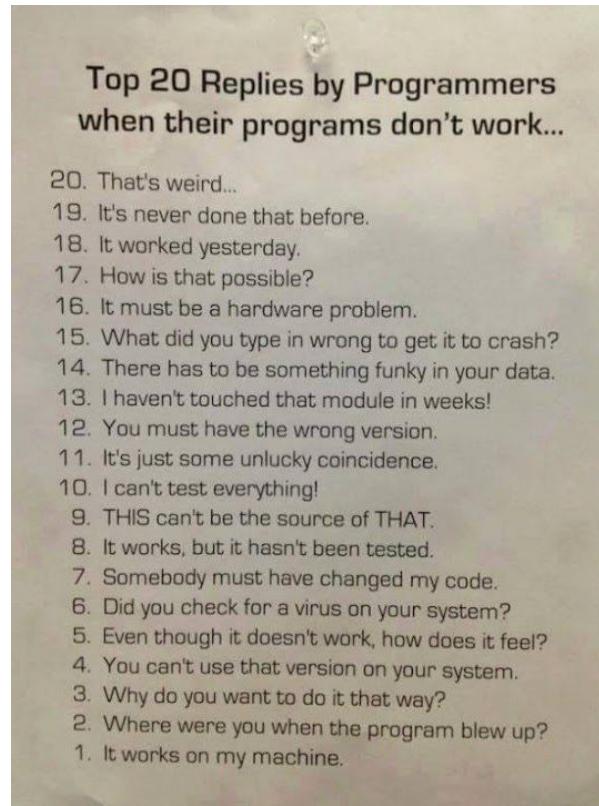


“It works on my machine”

IT WORKS
on my machine

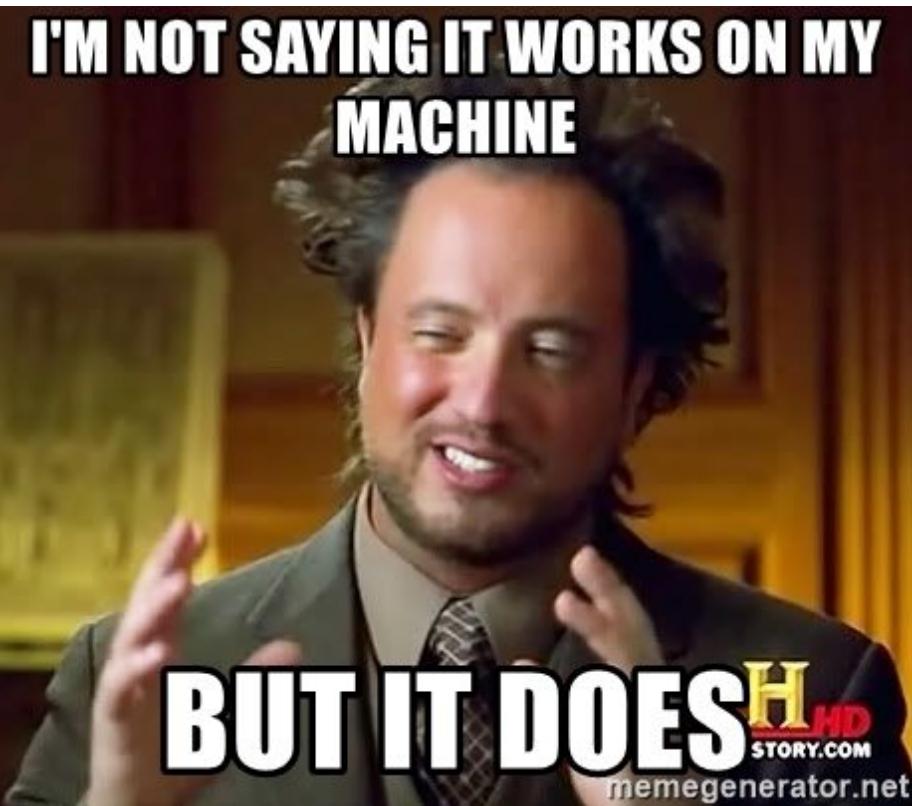
que consiste en que algo que funciona en el ordenador de un desarrollador, no le funciona a otro cuando lo instala y configura, siendo muy complicado detectar donde está el fallo.

2. Replicación de entornos de desarrollo



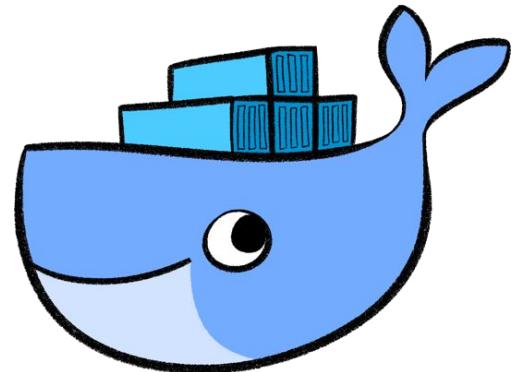
Problem? What Problem?

2. Replicación de entornos de desarrollo



2. Replicación de entornos de desarrollo

Si la aplicación está total o parcialmente **empaquetada** en contenedores, la instalación y configuración se hace **solo 1 vez**, al crear el contenedor (o después al modificarlo), de forma que si al ejecutar el contenedor funciona, lo va a hacer de igual forma **en todos los equipos**.



Caso 3

Replicación de entorno de producción
en desarrollo



3. Replicación de entorno de producción en desarrollo

Otro problema con el que se encuentran comúnmente los desarrolladores es que el entorno de **producción**; es decir, donde se ejecuta la aplicación **no es igual que el entorno de desarrollo**, que es donde se programa.

Esto normalmente se debe a la dificultad o a la imposibilidad de instalar o configurar los servicios que la aplicación usa en producción, en un ordenador personal.

3. Replicación de entorno de producción en desarrollo

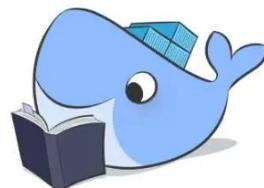
Como consecuencia, algo que funcionaba en desarrollo, puede que no funcione en producción.

Lo que se busca siempre es **que el entorno de desarrollo se asemeje lo máximo posible al de producción**, pero éste último no lo configuran programadores, sino administradores de sistemas o **devops**.

3. Replicación de entorno de producción en desarrollo

Docker proporciona un medio de simplificar el problema: se pueden **crear contenedores** con una configuración similar a la de producción **y dárselos a los programadores** para que los utilicen.

Si los programadores saben, podrían hacerlos ellos, pero en el caso peor; es decir, en el que no supieran, los propios **devops y administradores que instalan y configuran el entorno** de producción podrían crear los contenedores para que los desarrolladores los usen como una **caja negra**.



Caso 4

Despliegue de aplicaciones



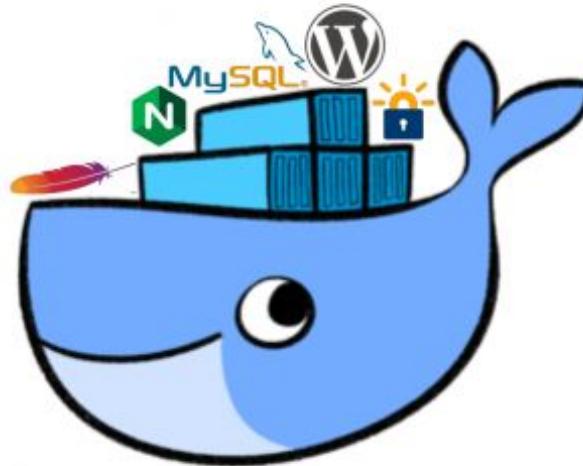
4. Despliegue de aplicaciones

De forma análoga a los casos anteriores, cuando la aplicación se quiere desplegar, debe hacerse la **instalación y configuración** de todos los servicios **en cada uno de los equipos** en los que se quiera hacer el despliegue.

En aplicaciones web, por ejemplo, es muy común que una aplicación se despliegue en varios servidores por temas de redundancia y para poder dar respuesta a más usuarios.

4. Despliegue de aplicaciones

El proceso de **despliegue es laborioso**, y si se crea **un contenedor, solo debe hacerse 1 vez**, ya que el contenedor se puede copiar en cada máquina y ejecutar como si fuera una única aplicación.



Caso 5

Gestión de aplicaciones como un todo
(Redundancia, tolerancia a fallos, escalado horizontal, etc.)



5. Gestión de aplicaciones como un todo

Finalmente, y sobretodo en sistemas **en la nube**, el tener las aplicaciones en contenedores permite instaurar políticas de **redundancia, tolerancia a fallos escalado horizontal o ejecución bajo demanda**; ya que los contenedores pueden replicarse, encenderse y apagarse fácilmente y de forma **automática**.

5. Gestión de aplicaciones como un todo

Por ejemplo, se puede tener un contenedor con la aplicación web anterior funcionando, y **si hay más usuarios** de los que puede soportar, “**levantar automáticamente otro contenedor** para que den servicio los dos. Y posteriormente, cuando bajara la demanda, el segundo contenedor se podría **apagar**, también automáticamente.

Otro ejemplo sería que un contenedor **fallara y dejara de funcionar**, y entonces automáticamente se levantara otro contenedor.

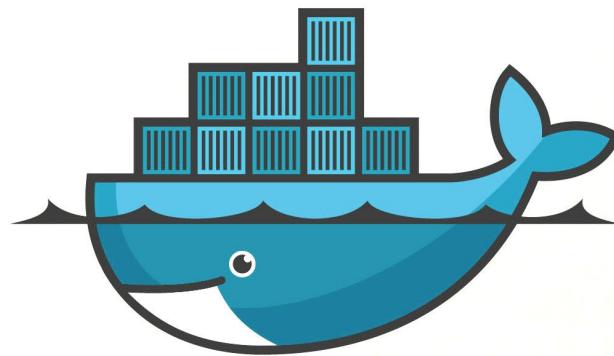
5. Gestión de aplicaciones como un todo

Si el despliegue de los contenedores se puede automatizar, eso significa que existe un ecosistema que los contiene, y unos parámetros de configuración para esta automatización, o alguien que programe esa automatización.

Si esta no es una tarea puramente de programación, pero tampoco de configuración. ¿Quién se encarga de realizarla? Los DevOps. Normalmente se dice que los devops programan la infraestructura de las aplicaciones.



Instalación

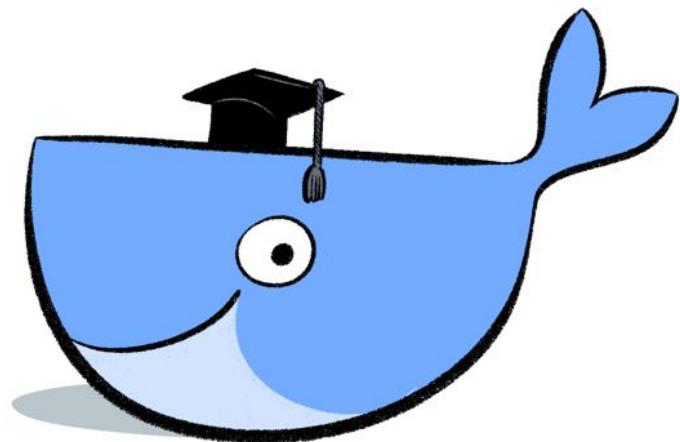


docker

Contenido

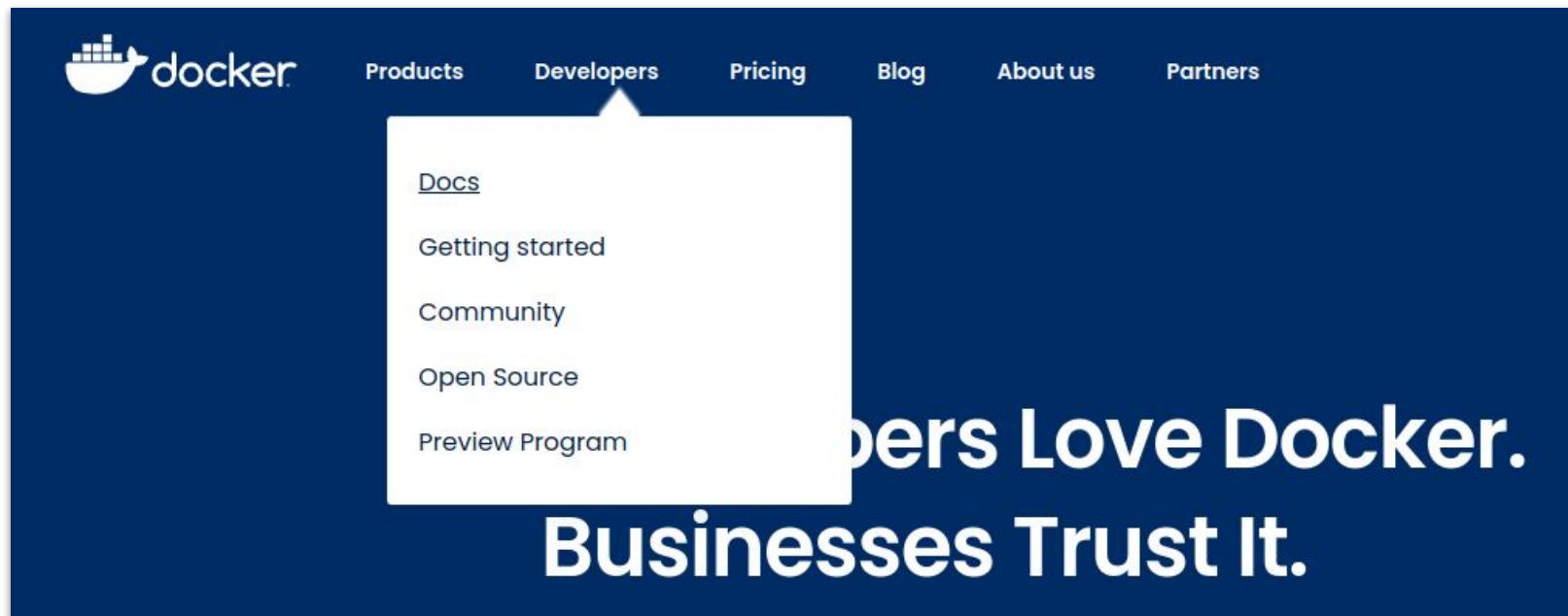
1. Documentación de la instalación
2. Proceso de instalación
3. Comprobación de la instalación
4. Operaciones de mantenimiento

Documentación de la instalación



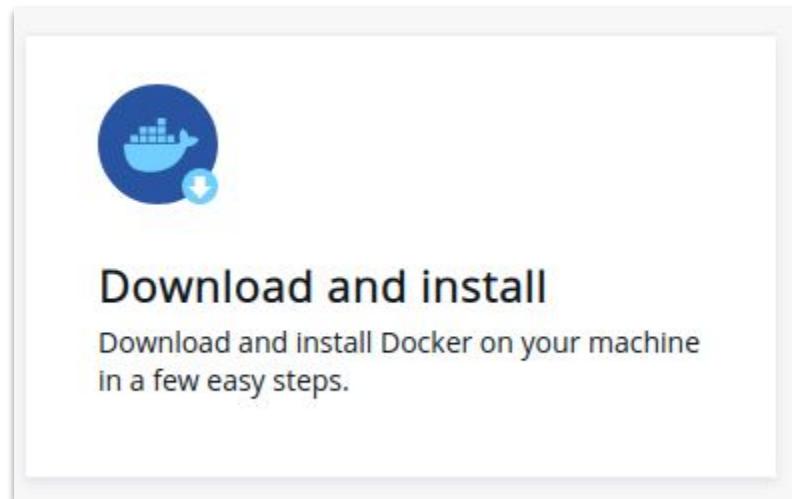
1. Documentación de la instalación

<https://www.docker.com/> → Developers / Docs



1. Documentación de la instalación

<https://docs.docker.com/> → Download and install



1. Documentación de la instalación

Seleccionamos **Docker for Linux**



Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.



Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.



Docker for Linux

Install Docker on a computer which already has a Linux distribution installed.

1. Documentación de la instalación

Server

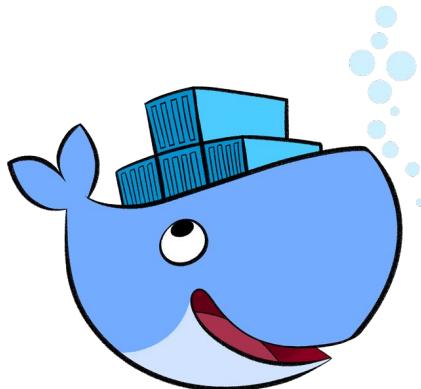
Docker provides [.deb](#) and [.rpm](#) packages for the following platforms:

Platform	
CentOS	✓
Debian	✓
Fedora	✓
Raspbian	
RHEL	
SLES	
Ubuntu	✓

Seleccionamos **Ubuntu**



Proceso de instalación



Proceso de instalación

1. Prerrequisitos
 - a. Comprobar compatibilidad con el SO
 - b. Desinstalar versiones anteriores de los paquetes
2. Instalación
 - a. Añadir el repositorio
 - b. Instalar docker

Comprobar compatibilidad con el SO

Requisitos

Prerequisites

OS requirements

To Install Docker Engine, you need the 64-bit version of

- Ubuntu Impish 21.10
- Ubuntu Hirsute 21.04
- **Ubuntu Focal 20.04 (LTS)**
- Ubuntu Bionic 18.04 (LTS)

Tenemos



Kubuntu 20.04

<https://www.kubuntu.org>

```
pepe@pepe-VBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.4 LTS
Release:        20.04
Codename:       focal
```

Desinstalar versiones anteriores

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

```
pepe@pepe-VBox:~$ sudo apt remove docker docker-engine docker.io containerd runc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
E: No se ha podido localizar el paquete docker-engine
pepe@pepe-VBox:~$ sudo apt remove docker docker.io containerd runc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
El paquete «docker» no está instalado, no se eliminará
El paquete «runc» no está instalado, no se eliminará
El paquete «docker.io» no está instalado, no se eliminará
El paquete «containerd» no está instalado, no se eliminará
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
```

Añadir el repositorio

1. Actualizar índice de paquetes e instalar prerequisitos

```
$ sudo apt-get update  
  
$ sudo apt-get install \  
  ca-certificates \  
  curl \  
  gnupg \  
  lsb-release
```

```
pepe@pepe-VBox:~$ sudo apt update  
Obj:1 http://es.archive.ubuntu.com/ubuntu focal InRelease  
Obj:2 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease  
Obj:3 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease  
Obj:4 http://security.ubuntu.com/ubuntu focal-security InRelease  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Todos los paquetes están actualizados.
```

```
pepe@pepe-VBox:~$ sudo apt install ca-certificates curl gnupg lsb-release  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
lsb-release ya está en su versión más reciente (11.1.0ubuntu2).  
fijado lsb-release como instalado manualmente.  
ca-certificates ya está en su versión más reciente (20210119~20.04.2).  
fijado ca-certificates como instalado manualmente.  
gnupg ya está en su versión más reciente (2.2.19-3ubuntu2.1).  
fijado gnupg como instalado manualmente.  
Se instalarán los siguientes paquetes NUEVOS:  
  curl
```

Añadir el repositorio

2. Añadir clave GPG del repositorio de Docker

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-
```

```
pepe@pepe-VBox:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
pepe@pepe-VBox:~$ █
```

Añadir el repositorio

3. Añadir el repositorio de Docker

```
$ echo \  
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
pepe@pepe-VBox:~$ echo \  
> "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \  
> $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
pepe@pepe-VBox:~$ cat /etc/apt/sources.list.d/docker.list  
deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu focal stable
```

Instalar Docker

```
$ sudo apt-get update  
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

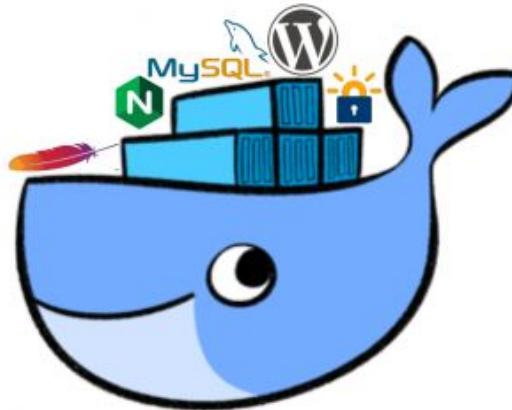
```
pepe@pepe-VBox:~$ sudo apt update  
Obj:1 http://es.archive.ubuntu.com/ubuntu focal InRelease  
Obj:2 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease  
Obj:3 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease  
Des:4 https://download.docker.com/linux/ubuntu focal InRelease [57,7 kB]  
Obj:5 http://security.ubuntu.com/ubuntu focal-security InRelease  
Des:6 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [14,8 kB]  
Descargados 72,5 kB en 1s (119 kB/s)  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Todos los paquetes _están actualizados.
```

Instalar Docker

```
$ sudo apt-get update  
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

```
pepe@pepe-VBox:~$ sudo apt install docker-ce docker-ce-cli containerd.io  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  docker-ce-rootless-extras docker-scan-plugin git git-man liberror-perl patch pigz  
  slirp4netns  
Paquetes sugeridos:  
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit  
  git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn  
  diffutils-doc  
Se instalarán los siguientes paquetes NUEVOS:  
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-scan-plugin  
  git git-man liberror-perl patch pigz slirp4netns  
0 actualizados, 11 nuevos se instalarán, 0 para eliminar y 0 no actualizados.  
Se necesita descargar 104 MB de archivos.  
Se utilizarán 448 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n]  
Des:1 http://es.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1 [57,4 kB]
```

Comprobación de la instalación



Comprobación de la instalación

1. Visualizar versiones instaladas
2. Visualizar información de docker
3. Ejecutar contenedor de ejemplo

Visualizar versiones instaladas

```
sudo docker version
```

```
pepe@pepe-VBox:~$ sudo docker version
Client: Docker Engine - Community
  Version:           20.10.13
  API version:      1.41
  Go version:       go1.16.15
  Git commit:       a224086
  Built:            Thu Mar 10 14:07:51 2022
  OS/Arch:          linux/amd64
  Context:          default
  Experimental:    true

Server: Docker Engine - Community
  Engine:
    Version:          20.10.13
    API version:     1.41 (minimum version 1.12)
    Go version:      go1.16.15
    Git commit:      906f57f
    Built:           Thu Mar 10 14:05:44 2022
    OS/Arch:         linux/amd64
    Experimental:   false
  containerd:
    Version:          1.5.10
    GitCommit:        2a1d4dbdb2a1030dc5b01e96fb110a9d9f150ecc
  runc:
    Version:          1.0.3
    GitCommit:        v1.0.3-0-gf46b6ba
  docker-init:
    Version:          0.19.0
    GitCommit:        de40ad0
```

Visualizar información de Docker

```
sudo docker info
```

```
pepe@pepe-VBox:~$ docker info
Client:
  Context:    default
  Debug Mode: false
  Plugins:
    app: Docker App (Docker Inc., v0.9.1-beta3)
    buildx: Docker Buildx (Docker Inc., v0.8.0-docker)
    scan: Docker Scan (Docker Inc., v0.17.0)

Server:
  Containers: 3
    Running: 0
    Paused: 0
    Stopped: 3
  Images: 2
  Server Version: 20.10.13
  Storage Driver: overlay2
    Backing Filesystem: extfs
    Supports d_type: true
    Native Overlay Diff: true
    userxattr: false
    Logging Driver: json-file
    Cgroup Driver: cgroupfs
    Cgroup Version: 1
  Plugins:
    Volume: local
    Network: bridge host ipvlan macvlan null overlay
    Log: awslogs fluentd gcplogs gelf journalctl json-file local logentries splunk syslog
  Swarm: inactive
  Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
  Default Runtime: runc
```

Ejecutar contenedor de ejemplo

```
sudo docker run hello-world
```

```
pepe@pepe-VBox:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:4c5f3db4f8a54eb1e017c385f683a2de6e06f75be442dc32698c9bbe6c861edd
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

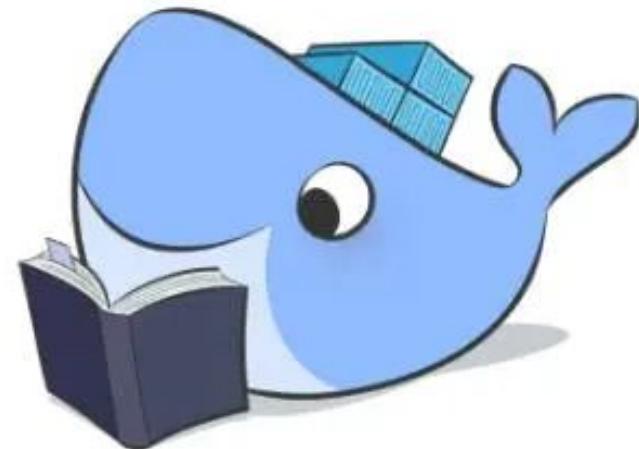
Share images, automate workflows, and more with a free Docker ID:

```
https://hub.docker.com/
```

For more examples and ideas, visit:

```
https://docs.docker.com/get-started/
```

Operaciones de mantenimiento



Operaciones de mantenimiento

1. Actualizar Docker
2. Eliminar Docker
3. Ejecutar Docker como un usuario normal (sin sudo)

Actualizar Docker

Al instalarse con un repositorio, Docker se actualizará **con el resto del sistema**.

```
sudo apt update  
sudo apt upgrade
```

Eliminar Docker

Para eliminar **Docker**, hay que borrar los paquetes correspondientes:

```
sudo apt-get purge docker-ce docker-ce-cli containerd.io
```

Para eliminar los **contenedores, la imágenes y los volúmenes**, hay que borrar los directorios que las contienen:

```
sudo rm -rf /var/lib/docker  
sudo rm -rf /var/lib/containerd
```

Ejecutar Docker como un usuario normal (sin sudo)

Si un usuario normal quiere usar docker, se debe añadir al grupo **docker**:

```
usermod -aG docker username
```

Se deberá volver a iniciar sesión para que el cambio de grupo surta efecto.

Los contenedores que ejecute ese usuario, heredarán sus permisos en vez de los del root.

Ejercicio 1



Ejercicio 1

Instala Docker en una máquina virtual y haz un manual con capturas de pantalla y explicaciones sobre el proceso.

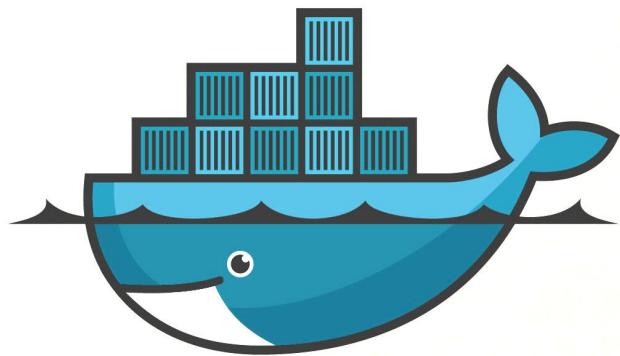
El prompt de la máquina tiene que identificarte, como por ejemplo:

```
pepe@pepe-VBox:~$
```

La instalación debe contemplar 4 fases:

1. Prerrequisitos.
2. Instalación.
3. Comprobación.
4. Añadir usuario al grupo docker.

Arquitectura

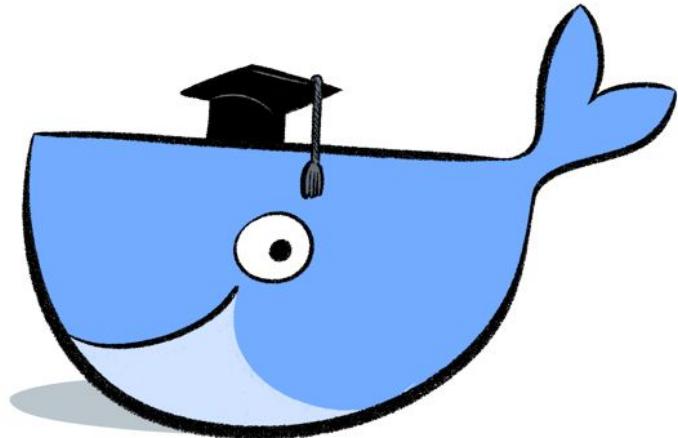


docker

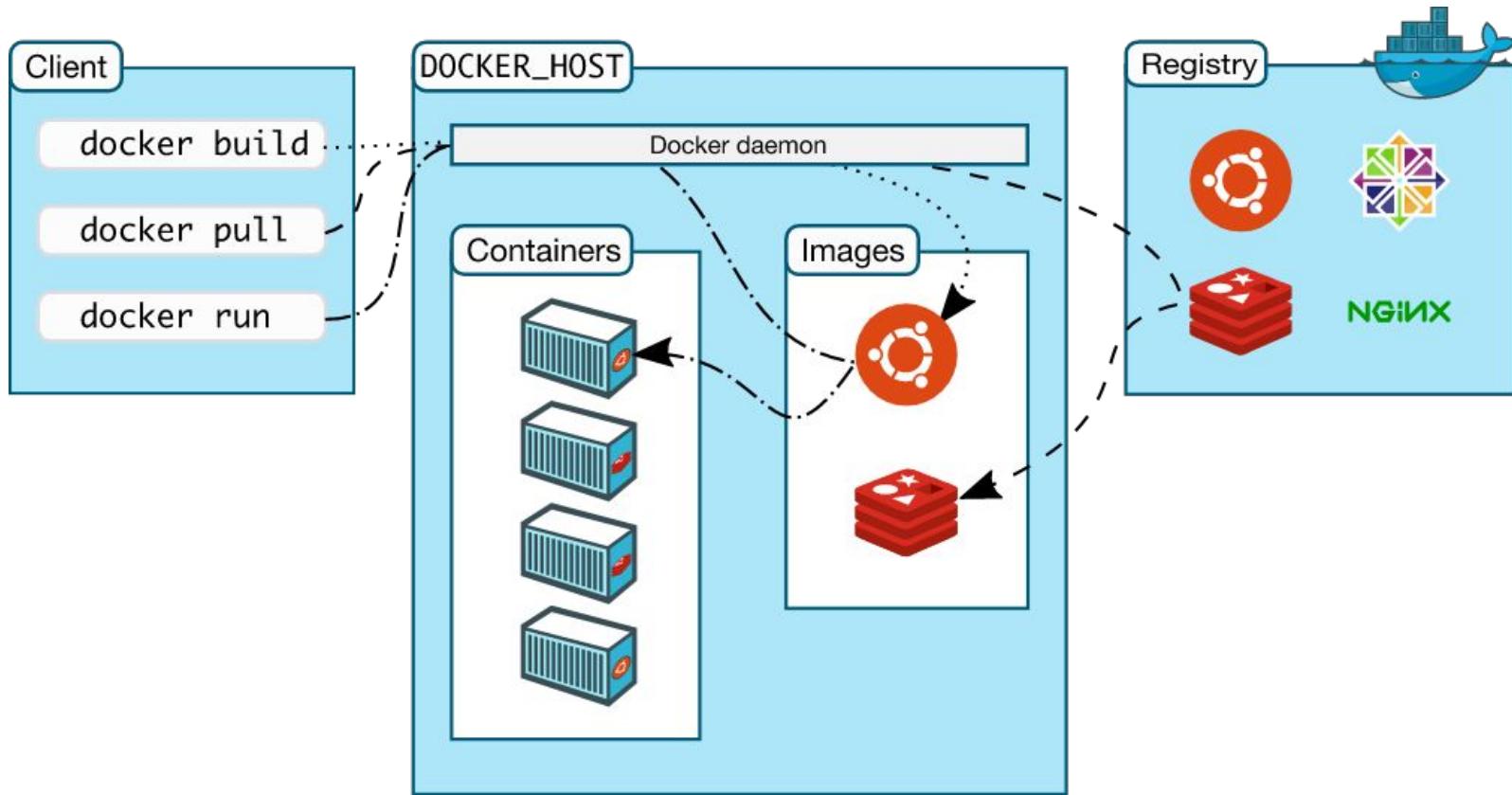
Contenido

1. Arquitectura básica
2. Componentes

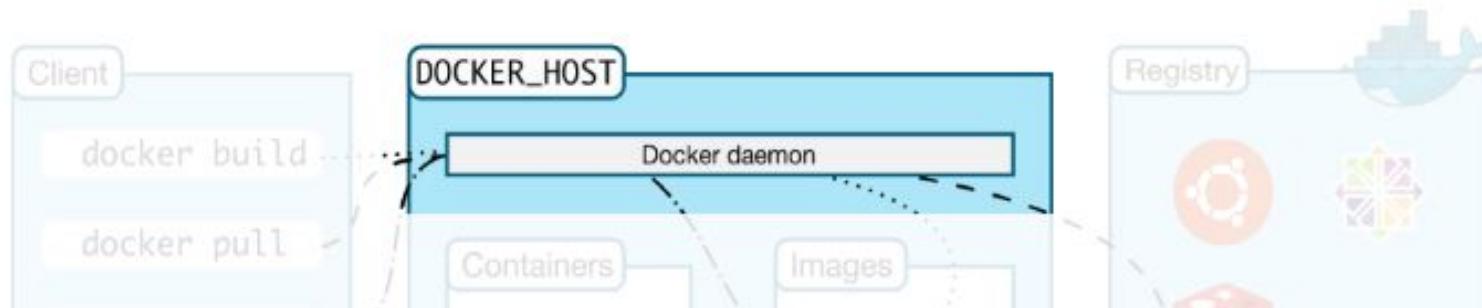
Arquitectura básica



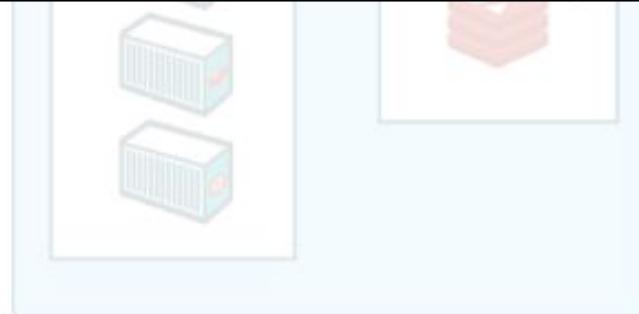
Arquitectura básica



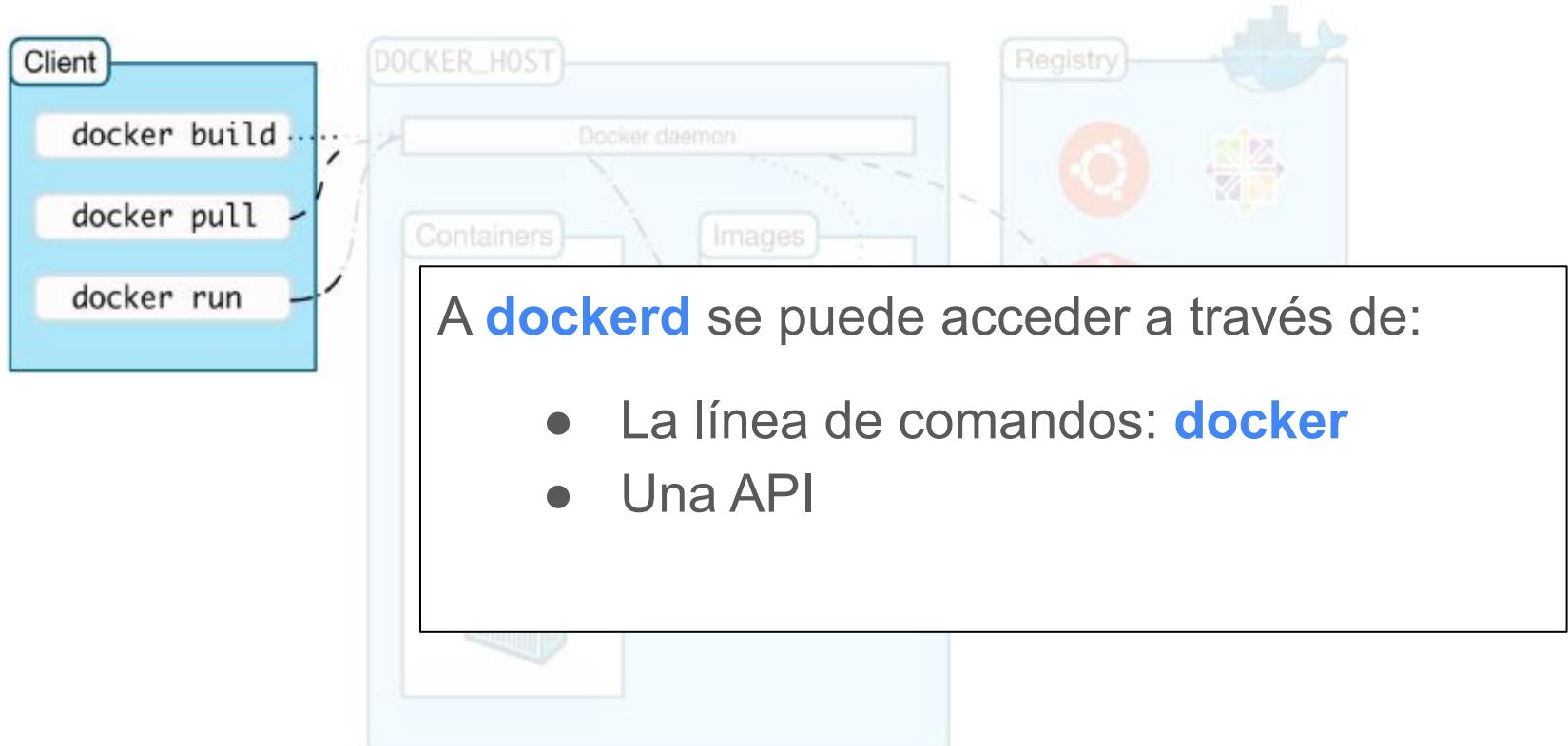
Arquitectura básica



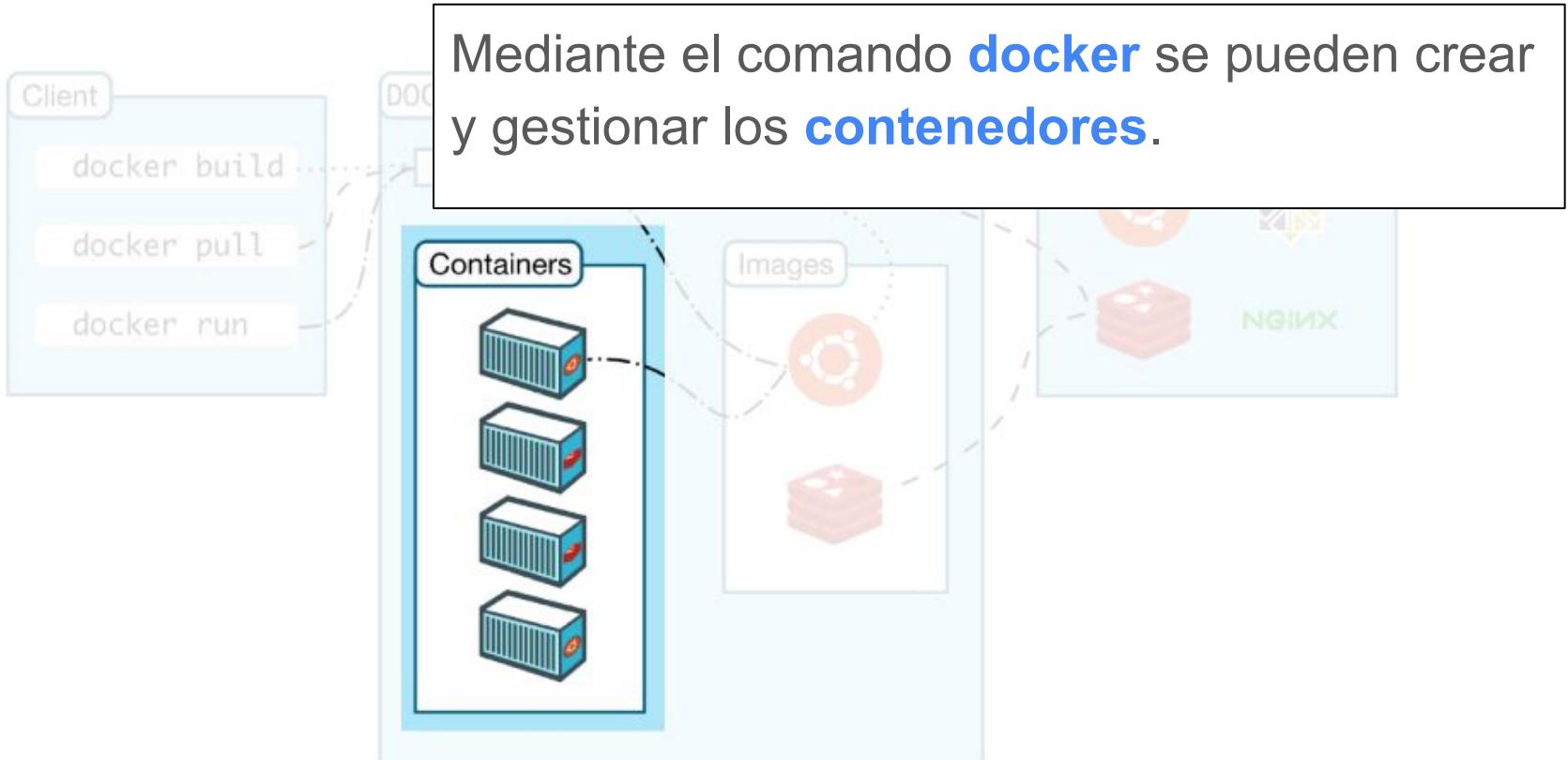
Docker funciona con un programa que se ejecuta en segundo plano (demonio) llamado **dockerd**.



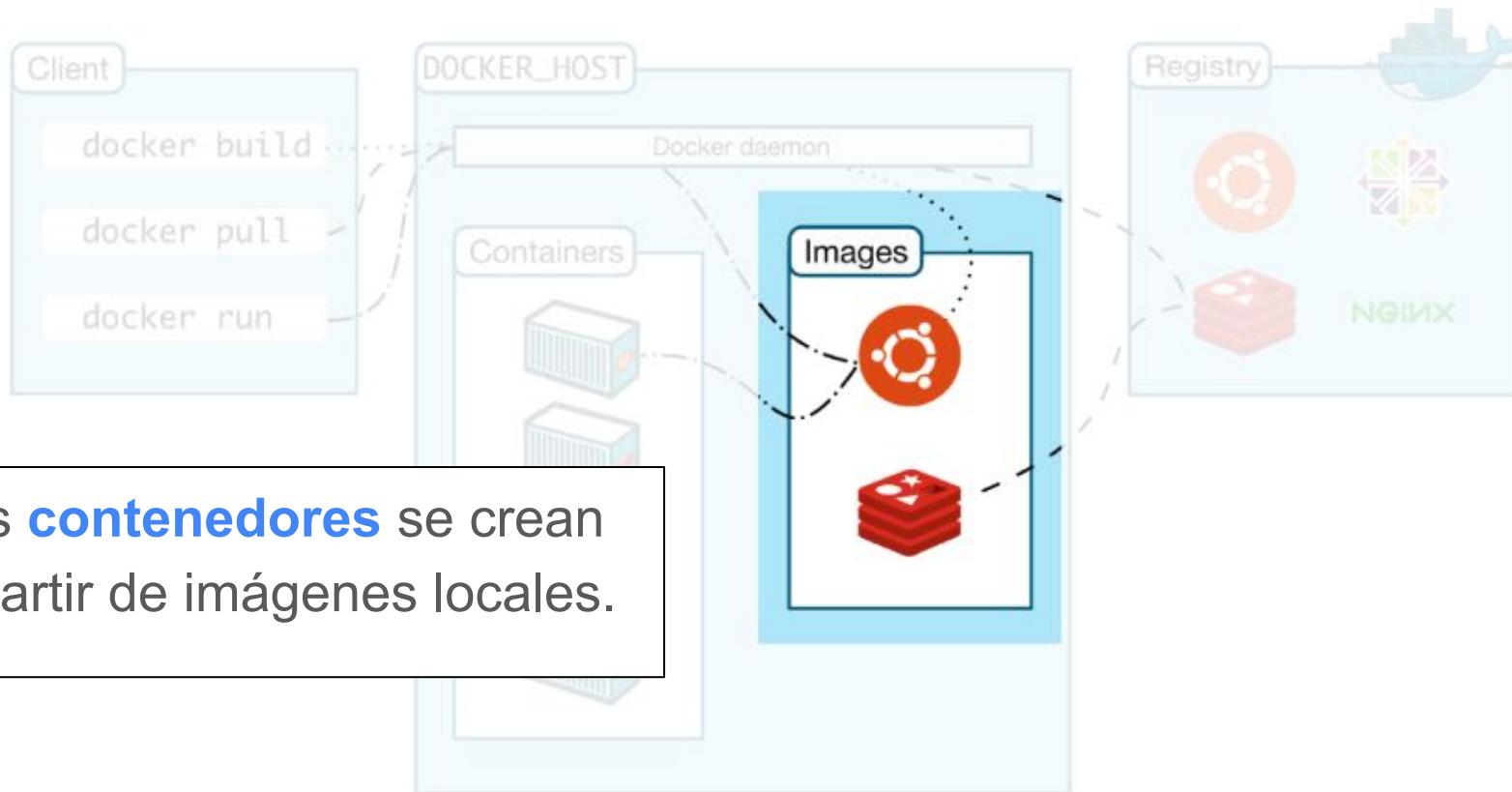
Arquitectura básica



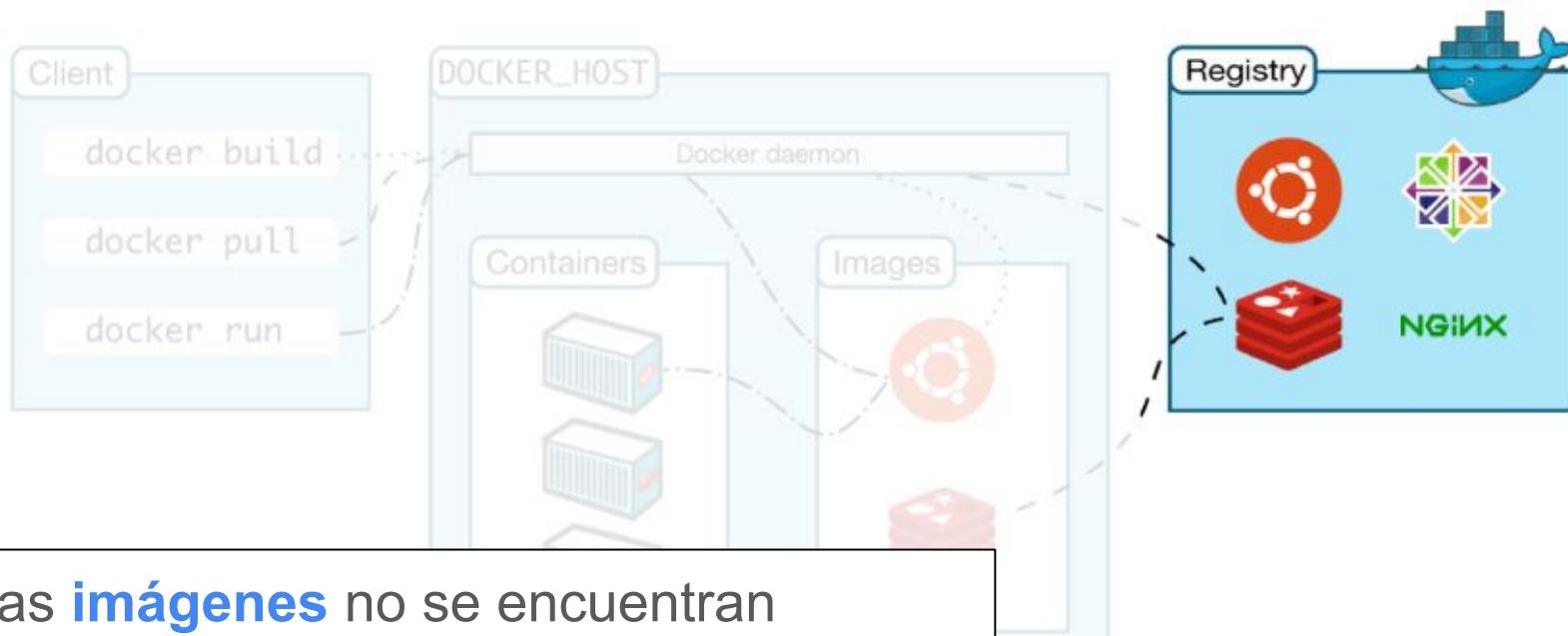
Arquitectura básica



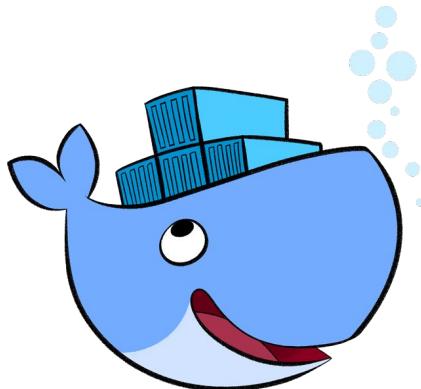
Arquitectura básica



Arquitectura básica



Componentes



Componentes

dokerd

redes

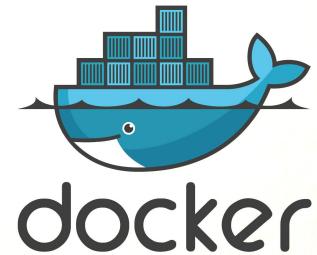
registro

API

doker

contenedores

volúmenes



imágenes

Componentes - *dockerd*

dockerd es el programa que provee la funcionalidad.

Funciona en segundo plano (**daemon**, demonio)

Es accesible a través de:

- La línea de comandos: **docker**
- Una **API** (por programa)

Componentes - *docker*

docker es el programa en línea de comandos que permite acceder a *dockerd*.

Se utiliza para hacer operaciones desde el **terminal** de forma **interactiva**.

Las operaciones principales que permite son para **gestionar**:

- Imágenes.
- Contenedores.
- Redes.
- Volúmenes.

Componentes - API

dockerd ofrece una **API** para que se pueda acceder a las funcionalidades de Docker desde otros programas externos.

Se utiliza para cosas como:

- Acceder a las funcionalidades de Docker presentando una UI.
- Automatizar tareas.

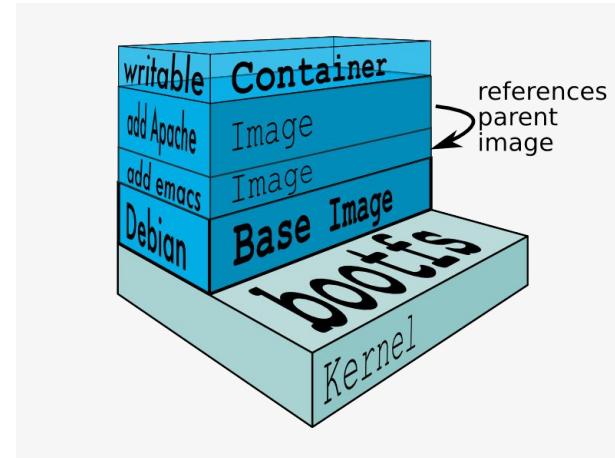
Componentes - *Imágenes*

Las imágenes son **plantillas** para crear contenedores.

A partir de una imagen si pueden crear **múltiples** contenedores.

Pueden tener **parámetros** a concretar al crear el contenedor.

Se crean a partir de otras, añadiendo modificaciones en forma de **capas**.



Componentes - *Imágenes*

Cuando creamos un contenedor, es necesaria la imagen base.

Si la imagen no se encuentra localmente, se intenta descargar del registro.

```
pepe@pepe-VBox:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:4c5f3db4f8a54eb1e017c385f683a2de6e06f75be442dc32698c9bbe6c861edd
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Componentes - *Registro*

El **registro** es un repositorio de imágenes.

Cuando Docker no encuentra una imagen localmente, intenta **descargarla** del registro que tenga seleccionado.

El registro más utilizado y el que viene por defecto es el **Docker Hub**:

<https://hub.docker.com/>



Componentes - Registro

Dockerhub Search for great content Explore Pricing Sign In Sign Up

Docker Containers Plugins

Filters 1 - 25 of 8.961.203 available images. Suggested

Images

- Verified Publisher
- Official Images Official Images Published By Docker

Categories

- Analytics
- Application Frameworks
- Application Infrastructure
- Application Services
- Base Images
- Databases
- DevOps Tools

 ubuntu  Updated 10 days ago

Ubuntu is a Debian-based Linux operating system based on free software.

Container Linux ARM ARM 64 386 PowerPC 64 LE IBM Z riscv64
x86-64 Base Images Operating Systems

 alpine  Updated 5 days ago

A minimal Docker image based on Alpine Linux with a complete package index and on...

Container Linux ARM riscv64 ARM 64 386 PowerPC 64 LE IBM Z
x86-64 Featured Images Base Images Operating Systems

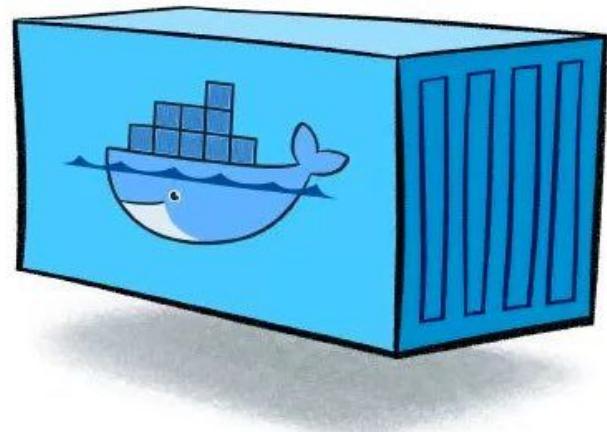
Componentes - Contenedores

Un **contenedor** es una instancia de una imagen.

Contienen las **aplicaciones** que se ejecutan.

Pueden ser arrancados, parados y ejecutados.

Poseen un **identificador** único de 64 caracteres (con una versión corta de 12)



Componentes - *Redes*

Docker permite crear **redes virtuales**.

Las redes virtuales sirven para:

- Comunicar contenedores **entre ellos**.
- Comunicar contenedores **con el anfitrión**.
- Comunicar contenedores **con el exterior**.

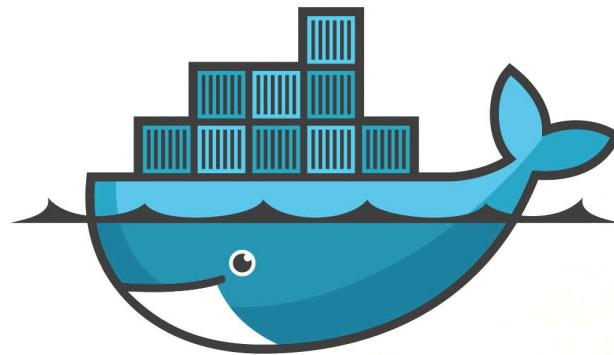
Componentes - Volúmenes

En principio, el sistema de ficheros de un contenedor es **local** y está **aislado del exterior**.

Los volúmenes permiten **comunicar** el sistema de ficheros del contenedor **con el exterior**, por ejemplo, con el anfitrión.

El uso más común es **montar directorios** del anfitrión en los contenedores.

Contenedores



docker

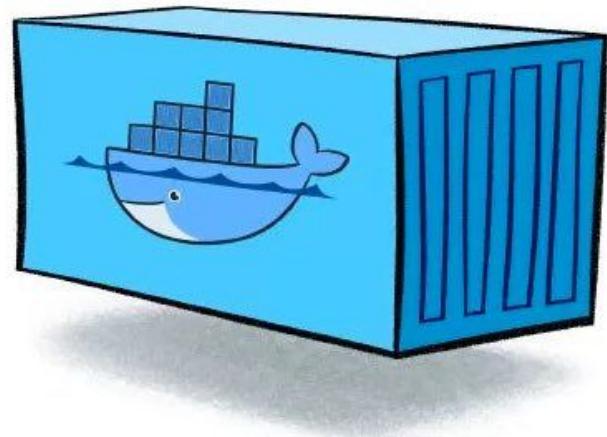
Repaso Contenedores

Un **contenedor** es una instancia de una imagen.

Contienen las **aplicaciones** que se ejecutan.

Pueden ser arrancados, parados y ejecutados.

Poseen un **identificador** único de 64 caracteres (con una versión corta de 12)



Comandos básicos

run	Crear y ejecutar
create	Crear
ps	Listar
start/stop/restart	Iniciar/Parar/Reiniciar
inspect	Obtener información
exec	Ejecutar comando

Comandos básicos

cp	Copiar archivos
attach	Enlazar stdin/stdout
log	Ver logs
rename	Cambiar nombre
rm	Eliminar



Comandos básicos - run

docker run *imagen:version*

Crea un contenedor a partir de la imagen y versión especificadas, y lo **ejecuta**.

```
pepe@pepe-VBox:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:4c5f3db4f8a54eb1e017c385f683a2de6e06f75be442dc32698c9bbe6c861edd
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Comandos básicos - run

Si se ejecuta otra vez, crea otro contenedor.

```
pepe@pepe-VBox:~$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ae860cc19af8 hello-world "/hello" 8 seconds ago Exited (0) 6 seconds ago
a49065fee71f hello-world "/hello" 6 days ago Exited (0) 6 days ago
```

Comandos básicos - run

Tiene muuuuchas **opciones**

```
pepe@pepe-VBox:~$ docker run --help
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
Run a command in a new container

Options:
  --add-host list          Add a custom host-to-IP mapping (host:ip)
  -a, --attach list         Attach to STDIN, STDOUT or STDERR
  --blkio-weight uint16     Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
  --blkio-weight-device list Block IO weight (relative device weight) (default [])
  --cap-add list            Add Linux capabilities
  --cap-drop list           Drop Linux capabilities
  --cgroup-parent string    Optional parent cgroup for the container
  --cgroupons string        Cgroup namespace to use (host|private)
                            'host': Run the container in the Docker host's cgroup namespace
                            'private': Run the container in its own private cgroup namespace
                            ''': Use the cgroup namespace as configured by the
                            default-cgroupons-mode option on the daemon (default)
  --cidfile string          Write the container ID to the file
  --cpu-period int          Limit CPU CFS (Completely Fair Scheduler) period
  --cpu-quota int           Limit CPU CFS (Completely Fair Scheduler) quota
  --cpu-rt-period int       Limit CPU real-time period in microseconds
  --cpu-rt-runtime int      Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int      CPU shares (relative weight)
  --cpus decimal             Number of CPUs
  --cpuset-cpus string      CPUs in which to allow execution (0-3, 0,1)
  --cpuset-mems string      MEMs in which to allow execution (0-3, 0,1)
  -d, --detach               Run container in background and print container ID
  --detach-keys string       Override the key sequence for detaching a container
```

Comandos básicos - run

Lanzar contenedor y ejecutar un **comando**:

```
pepe@pepe-VBox:~$ docker run ubuntu echo 'hola'  
hola
```

Lanzar contenedor y abrir un **terminal** interactivo:

```
pepe@pepe-VBox:~$ docker run -it ubuntu /bin/bash  
root@8e73db266b01:/# exit  
exit  
pepe@pepe-VBox:~$
```

Lanzar contenedor con un **nombre** concreto:

```
pepe@pepe-VBox:~$ docker run --name ubuntu01 ubuntu echo 'hola'  
hola  
pepe@pepe-VBox:~$ docker ps -a  
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS     NAMES  
d9ed63ac696e        ubuntu      "echo hola"   8 seconds ago   Exited (0) 7 seconds ago   0.0.0.0:22->22/tcp  ubuntu01
```

Comandos básicos - run

Lanzar contenedor y **borrarlo** cuando se pare:

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS      NAMES
pepe@pepe-VBox:~$ docker run --rm --name ubuntu01 ubuntu echo 'hola'
hola
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS      NAMES
```

Lanzar contenedor en **segundo plano**:

```
pepe@pepe-VBox:~$ docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is
/docker-entrypoint.sh: Looking for shell scripts
/docker-entrypoint.sh: Launching /docker-entrypo
10-listen-on-ipv6-by-default.sh: info: Getting t
10-listen-on-ipv6-by-default.sh: info: Enabled l
/docker-entrypoint.sh: Launching /docker-entrypo
/docker-entrypoint.sh: Launching /docker-entrypo
```

```
pepe@pepe-VBox:~$ docker run -d nginx
31eac0bb3371d81827dbf06974b839b098240661d5f4f5e8c1c54b928905e6a0
pepe@pepe-VBox:~$
```

Primer plano

Segundo plano

Comandos básicos - run

Lanzar contenedor y **mapear puertos** con el anfitrión:

```
pepe@pepe-VBox:~$ docker run -d -p 3000:80 nginx
0b533713d5dedac3c8563493fcb781d7d560a3da2ad4a7c33f02220edf847f78
pepe@pepe-VBox:~$ docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS          PORTS          NAMES
0b533713d5de   nginx      "/docker-entrypoint..."   58 seconds ago   Up  57 seconds   0.0.0.0:3000->80/tcp, :::3000->80/tcp   quirky_elgamal
```



El mapeo de puertos **no se puede cambiar** una vez creado el contenedor.

Si no se especifica el puerto del anfitrión, lo elige Docker.

Comandos básicos - run

Lanzar contenedor y **mapear automáticamente los puertos expuestos** con el anfitrión:

```
→ docker run -d -P nginx  
7e8b670e8e8a10a8c1773f085e9277403b2395b0a5d6b2d7f5feb862ec59df8a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
7e8b670e8e8a	nginx	"/docker-entrypoint...."	5 seconds ago	Up 4 seconds	0.0.0.0:49154->80/tcp, :::49154->80/tcp

Los puertos expuestos son configurados a la hora de hacer la imagen.

Comandos básicos - run

Lanzar contenedor y definir **variables de entorno**:

```
pepe@pepe-VBox:~$ docker run -it -e VAR='hola' bash
bash-5.1# echo $VAR
hola
bash-5.1# exit
exit
pepe@pepe-VBox:~$
```

Las variables de entorno se utilizan **para configurar** aspectos de contenedor.

Algunas son **opcionales** y otras **obligatorias**.

Hay que **consultar la documentación** de cada imagen.

Comandos básicos - run

Lanzar contenedor y **enlazarlo a otro contendor**:

```
~ ➔ docker run --link mysql:db -d wordpress  
45541b0f4f32c6e7ada08b9d95040471914f5d1ca158b2023c0407510ce9f221
```

La opción **--link** permite acceder a otro contenedor por nombre del contenedor o por alias, si se especifica (nombre:alias).

Lo que se hace internamente es modificar el archivo **/etc/hosts** del contenedor. Es lo que se conoce como **resolución de nombres estática**.

También **comparte** las variables de entorno del contenedor enlazado.

Esta opción se considera **obsoleta**, en favor de usar redes privadas de docker.



Comandos básicos - create

```
docker create imagen:version
```

Crea un contenedor a partir de la imagen y versión especificadas.

A diferencia de run, no lo ejecuta.

Devuelve el identificador del contenedor.

```
pepe@pepe-VBox:~$ docker create hello-world  
286c9463200c48242922e11816580d37570a883592ebea5bb0d0f88280a91966
```

```
pepe@pepe-VBox:~$ docker ps -a  
CONTAINER ID        IMAGE               COMMAND            CREATED             STATUS              PORTS          NAMES  
286c9463200c        hello-world        "/hello"           2 minutes ago     Created  
4cde466fed02        hello-world        "/hello"           6 minutes ago    Exited (0) 5 minutes ago      vibrant_kepler  
                                .....
```

Comandos básicos - create

Admite las mismas **opciones** que run.

```
pepe@pepe-VBox:~$ docker create --help

Usage: docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

Create a new container

Options:
  --add-host list          Add a custom host-to-IP mapping (host:ip)
  -a, --attach list         Attach to STDIN, STDOUT or STDERR
  --blkio-weight uint16     Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
  --blkio-weight-device list Block IO weight (relative device weight) (default [])
  --cap-add list            Add Linux capabilities
  --cap-drop list           Drop Linux capabilities
  --cgroup-parent string    Optional parent cgroup for the container
  --cgroups string          Cgroup namespace to use (host|private)
                            'host': Run the container in the Docker host's cgroup namespace
                            'private': Run the container in its own private cgroup namespace
                            ''': Use the cgroup namespace as configured by the
                            default-cgroups-mode option on the daemon (default)
  --cidfile string          Write the container ID to the file
  --cpu-period int          Limit CPU CFS (Completely Fair Scheduler) period
  --cpu-quota int           Limit CPU CFS (Completely Fair Scheduler) quota
  --cpu-rt-period int       Limit CPU real-time period in microseconds
  --cpu-rt-runtime int      Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int      CPU shares (relative weight)
  --cpus decimal             Number of CPUs
  --cpuset-cpus string      CPUs in which to allow execution (0-3, 0,1)
  --cpuset-mems string      MEMs in which to allow execution (0-3, 0,1)
  --device list              Add a host device to the container
  --device-cgroup-rule list  Add a rule to the cgroup allowed devices list
```



Comandos básicos - ps

docker ps
docker ps -a

Lista los contenedores en ejecución.

El parámetro **-a** lista también los contenedores parados.

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
286c9463200c hello-world "/hello" 2 minutes ago Created
4cde466fed02 hello-world "/hello" 6 minutes ago Exited (0) 5 minutes ago
vibrant_kepler
zealous_shockley
```



Comandos básicos - start/stop/restart

```
docker start id/name  
docker stop id/name  
docker restart id/name
```

Inicia/para/reinicia un contenedor.

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS          PORTS      NAMES
ae860cc19af8   hello-world "/hello"    18 minutes ago   Exited (0) 18 minutes ago
a49065fee71f   hello-world "/hello"    6 days ago    Exited (0) 6 days ago
pepe@pepe-VBox:~$ docker start trusting_rosalind
trusting_rosalind
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS          PORTS      NAMES
ae860cc19af8   hello-world "/hello"    19 minutes ago   Exited (0) 19 minutes ago
a49065fee71f   hello-world "/hello"    6 days ago    Exited (0) 4 seconds ago
```

Comandos básicos - start/stop/restart

La opcion **-a** muestra la salida del contenedor.

```
pepe@pepe-VBox:~$ docker start -a trusting_rosalind
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
```

Comandos básicos - start/stop/restart

La opción **-i** enlaza la entrada estándar.

Se usa para contenedores interactivos que esperan entrada de datos.

El contenedor debe haberse creado con la opción **-it** antes.

```
pepe@pepe-VBox:~$ docker create -it bash
9efc2839480fcea64fc8fdab371236f17658b9a0521384b19709a65775a95df
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND          CREATED        STATUS          PORTS     NAMES
9efc2839480f   bash        "docker-entrypoint.s..."  4 seconds ago  Created
ae860cc19af8   hello-world  "/hello"        38 minutes ago  Exited (0) 38 minutes ago
a49065fee71f   hello-world  "/hello"        6 days ago    Exited (0) 15 minutes ago
pepe@pepe-VBox:~$ docker start -i 9efc2839480f
bash-5.1# ls
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
bash-5.1# exit
exit
pepe@pepe-VBox:~$ █
```



Comandos básicos - inspect

docker inspect *id/name*

Proporciona mucha **información** sobre el contenedor.

Containers						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9efc2839480f	bash	"docker-entrypoint.s..."	7 minutes ago	Exited (0) 7 minutes ago		gifted_kare
ae860cc19af8	hello-world	/hello	46 minutes ago	Exited (0) 46 minutes ago		competent_bose
a49065fee71f	hello-world	/hello	6 days ago	Exited (0) 23 minutes ago		trusting_rosalind

Comandos básicos - inspect

```
pepe@pepe-VBox:~$ docker inspect gifted_kare
[
  {
    "Id": "9efc2839480fceea64fca8fdab371236f17658b9a0521384b19709a65775a95df",
    "Created": "2022-03-28T18:10:30.465728548Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "bash"
    ],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2022-03-28T18:10:44.266357581Z",
      "FinishedAt": "2022-03-28T18:10:48.943354892Z"
    },
    "Image": "sha256:91d5f5779560f72b02d2bf2d7eca9969faabf6a26bbea96d61c4af5c7d6ea76a",
    "ResolvConfPath": "/var/lib/docker/containers/9efc2839480fceea64fca8fdab371236f17658b9a0521384b19709a65775a95df/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/9efc2839480fceea64fca8fdab371236f17658b9a0521384b19709a65775a95df/hostname",
    "HostsPath": "/var/lib/docker/containers/9efc2839480fceea64fca8fdab371236f17658b9a0521384b19709a65775a95df/hosts",
    "LogPath": "/var/lib/docker/containers/9efc2839480fceea64fca8fdab371236f17658b9a0521384b19709a65775a95df/9efc2839480fceea64f17658b9a0521384b19709a65775a95df-json.log",
    "Name": "gifted_kare"
  }
]
```



Comandos básicos - exec

```
docker exec [opciones] id/name COMANDO [argumentos]
```

Ejecuta un **comando** en un contenedor.

El contenedor debe estar **en ejecución**.

Si el contenedor no está en ejecución, se usa docker run para ejecutar comandos.

Comandos básicos - exec

Vamos a usar la imagen de nginx como **ejemplo**, ya que al crear un contenedor se ejecuta hasta que lo paremos.

Nginx es un servidor web.

```
pepe@pepe-VBox:~$ docker run -d -p 80:80 nginx
62c38067607ff0ad07c7e48657dc687dff4594ea417809f627ea143417f8b060
```

La opción -d de run hace que el contenedor se ejecute en segundo plano y no bloquee el terminal.

La opción -p de run mapea puertos entre el contenedor y el host.

```
pepe@pepe-VBox:~$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS          NAMES
62c38067607f        nginx      "/docker-entrypoint..."   35 seconds ago   Up  34 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   pensive_babbage
```

Comandos básicos - exec

Podemos ver que el servidor funciona accediento a: `http://localhost`



Comandos básicos - exec

Ahora podemos usar exec para crear un archivo:

```
pepe@pepe-VBox:~$ docker exec -d pensive_babbage touch /tmp/mi_archivo.txt  
pepe@pepe-VBox:~$ █
```

También podemos abrir un terminal con las opciones **-it**:

```
pepe@pepe-VBox:~$ docker exec -it pensive_babbage bash  
root@62c38067607f:/# ls /tmp/  
mi_archivo.txt  
root@62c38067607f:/# exit  
exit  
pepe@pepe-VBox:~$ █
```

Se ve el archivo que acabamos de crear

Comandos básicos - exec

También es útil poder abrir un terminal con variables de entorno definidas:

```
pepe@pepe-VBox:~$ docker exec -it -e MY_VAR=5 pensive_babbage bash
root@62c38067607f:/# echo $MY_VAR
5
root@62c38067607f:/# exit
exit
pepe@pepe-VBox:~$ █
```



Comandos básicos - cp

```
docker cp origen destino
```

Copia archivos entre un contenedor y el anfitrión.

No se puede copiar archivos directamente entre contenedores

Ejemplo:

1. Creamos un contenedor
2. Creamos un archivo en el anfitrión y lo copiamos al contenedor
3. Accedemos al archivo desde el contenedor

Comandos básicos - cp

1

```
pepe@pepe-VBox:~$ docker run -it ubuntu /bin/bash
root@3c01aee1ad10:/# pwd
/
root@3c01aee1ad10:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

2

```
pepe@pepe-VBox:~$ ls
Descargas  Documentos  Escritorio  Imágenes  Música  Plantillas  Público  Vídeos
pepe@pepe-VBox:~$ echo 'hola' > archivo_01.txt
pepe@pepe-VBox:~$ cat archivo_01.txt
hola
pepe@pepe-VBox:~$ ls
archivo_01.txt  Descargas  Documentos  Escritorio  Imágenes  Música  Plantillas  Público  Vídeos
pepe@pepe-VBox:~$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS     NAMES
3c01aee1ad10        ubuntu      "/bin/bash"   About a minute ago   Up About a minute          funny_wiles
pepe@pepe-VBox:~$ docker cp archivo_01.txt funny_wiles:/
pepe@pepe-VBox:~$
```

3

```
root@3c01aee1ad10:/# ls
archivo_01.txt  bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin
root@3c01aee1ad10:/# cat archivo_01.txt
hola
```



Comandos básicos - attach

```
docker attach id/name
```

Enlaza stdin y stdout con un contenedor en ejecución.

Normalmente los contenedores se ejecutan en segundo plano (opción `-d`), y con `attach` pueden volver a “primer plano”.

Comandos básicos - attach

Ejemplo:

```
pepe@pepe-VBox:~$ docker run -d --name=imprime_fecha ubuntu sh -c "while true; do $(echo date); sleep 1; done"
648decdf8cef93edcbd64bcb0c70cc8133eb4b57a88d82cb39470f50114a5c13
```

```
pepe@pepe-VBox:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          PORTS     NAMES
648decdf8cef   ubuntu     "sh -c 'while true; ...'"   14 seconds ago   Up 13 seconds   0.0.0.0:32785->22/tcp   imprime_fecha
```

```
pepe@pepe-VBox:~$ docker attach imprime_fecha
Tue Mar 29 14:50:37 UTC 2022
Tue Mar 29 14:50:38 UTC 2022
Tue Mar 29 14:50:39 UTC 2022
Tue Mar 29 14:50:40 UTC 2022
Tue Mar 29 14:50:41 UTC 2022
Tue Mar 29 14:50:42 UTC 2022
^Cpepe@pepe-VBox:~$
```



Comandos básicos - logs

```
docker logs id/name
```

Muestra el **stdout/stderr** de un contenedor en ejecución.

Opciones interesantes:

- **-f**: sigue mostrando los logs.
- **-n num**: muestra los últimos “num” mensajes.

Comandos básicos - logs

Ejemplos:

```
pepe@pepe-VBox:~$ docker run -d --name=imprime_fecha ubuntu sh -c "while true; do $(echo date); sleep 5; done"
4011ccf86bc6e173f1d288e2b551bb22fdfa9d868f27e3fb3ae2f45ed5323893
```

```
pepe@pepe-VBox:~$ docker logs -f imprime_fecha
Tue Mar 29 15:10:48 UTC 2022
Tue Mar 29 15:10:53 UTC 2022
Tue Mar 29 15:10:58 UTC 2022
^C
```

```
pepe@pepe-VBox:~$ docker logs -n 2 imprime_fecha
Tue Mar 29 15:11:03 UTC 2022
Tue Mar 29 15:11:08 UTC 2022
```

```
pepe@pepe-VBox:~$ docker logs -f -n 2 imprime_fecha
Tue Mar 29 15:11:13 UTC 2022
Tue Mar 29 15:11:18 UTC 2022
Tue Mar 29 15:11:23 UTC 2022
^C
pepe@pepe-VBox:~$
```



Comandos básicos - rename

docker rename *id/name new_name*

Cambia el nombre de un contenedor

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS               NAMES
4011ccf86bc6        ubuntu      "sh -c 'while true; ...'"   8 minutes ago     Exited (137) 30 seconds ago
3c01aee1ad10        ubuntu      "/bin/bash"                43 minutes ago    Exited (0) 29 minutes ago
pepe@pepe-VBox:~$ docker rename 3c01aee1ad10 funny_meli
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS               NAMES
4011ccf86bc6        ubuntu      "sh -c 'while true; ...'"   8 minutes ago     Exited (137) 54 seconds ago
3c01aee1ad10        ubuntu      "/bin/bash"                43 minutes ago    Exited (0) 29 minutes ago
pepe@pepe-VBox:~$ docker rename imprime_fecha print_date
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS               NAMES
4011ccf86bc6        ubuntu      "sh -c 'while true; ...'"   9 minutes ago     Exited (137) 2 minutes ago
3c01aee1ad10        ubuntu      "/bin/bash"                44 minutes ago    Exited (0) 30 minutes ago
```



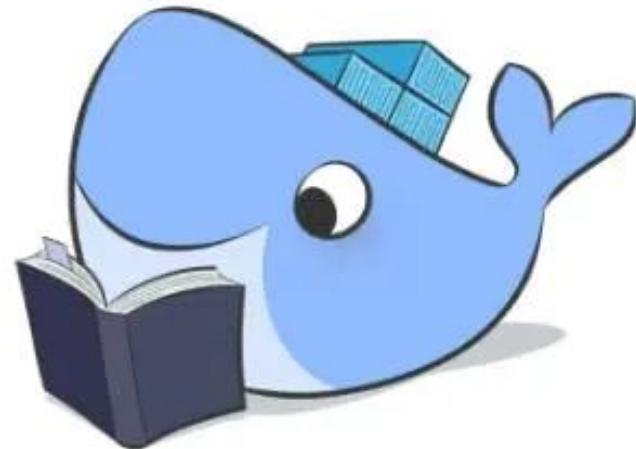
Comandos básicos - rm

docker rm *id/name*

Borra un contenedor que no está en ejecución.

```
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND           CREATED        STATUS          PORTS     NAMES
4011ccf86bc6   ubuntu     "sh -c 'while true; ...'"   9 minutes ago   Exited (137) 2 minutes ago
3c01aee1ad10   ubuntu     "/bin/bash"          44 minutes ago  Exited (0) 30 minutes ago
pepe@pepe-VBox:~$ docker rm print_date
print_date
pepe@pepe-VBox:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND           CREATED        STATUS          PORTS     NAMES
3c01aee1ad10   ubuntu     "/bin/bash"          47 minutes ago  Exited (0) 33 minutes ago
pepe@pepe-VBox:~$
```

Ejercicios



Ejercicio 1



Ejercicio 1a

Crea y ejecuta un contenedor a partir de la imagen de nginx con las siguientes características:

- Nombre: webserver.
- Mapeo de puertos: el 80 del contenedor al 8080 del anfitrión.
- Que se ejecute en segundo plano.

Accede a la web desde Firefox para que se vea.

Ejercicio 1b

Ahora abre un terminal interactivo con el contenedor y cambia la página web que se ve por defecto, que está en “/usr/share/nginx/html” por una que muestre un título y una imagen.

La imagen tendrás que copiarla desde el anfitrión al contenedor y meterla en la carpeta correspondiente para que se vea en la web.

Ejercicio 2



WORDPRESS



Ejercicio 2

Utilizando las imágenes oficiales de Docker de Wordpress y MySQL monta una página web con Wordpress funcional.

Después añade un contenedor con phpmyadmin que permita la administración de la base de datos.



Ejercicio 2



Paso 1: Ejecutar un contenedor con MySQL

Accede a la documentación oficial de la imagen oficial de MySQL en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **database**.

Ten en cuenta que necesitarás utilizar al menos las siguientes variables de entorno:

- MYSQL_ROOT_PASSWORD
- MYSQL_DATABASE

También tendrás que mapear el puerto que usa MySQL con el anfitrión para poder acceder luego desde el contenedor de Wordpress.



WORDPRESS

Ejercicio 2

Paso 2: Ejecutar un contenedor con Wordpress

Accede a la documentación oficial de Wordpress en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **web**.

No hace falta usar ninguna variable de entorno, ya que tienen una interfaz web de configuración; pero sí que tendrás que mapear el puerto HTTP con el anfitrión para poder acceder desde este a la web.



WORDPRESS

Ejercicio 2

Paso 2: Ejecutar un contenedor con Wordpress

Desde el contenedor de Wordpress se podrá acceder a la base de datos de 2 formas:

1. Usando la opción:

`--link <nombre_contenedor_mysql>:db`

2. A través de la dirección siguiente (siempre que se haya mapeado el puerto de MySQL correctamente):

`ip_interfaz_docker0:puerto`



Ejercicio 2

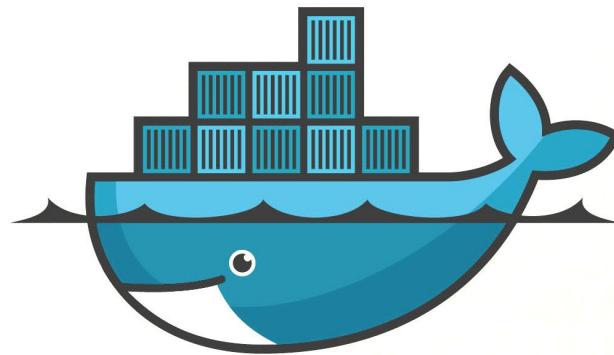
Paso 3: Ejecutar un contenedor con PhpMyAdmin

Accede a la documentación oficial de la imagen oficial de PhpMyAdmin en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **dbadmin**.

Hay que usar la opción:

--link <nombre_contenedor_mysql>:db

Contenedores



docker

Ejercicio 1



Ejercicio 1a

Crea y ejecuta un contenedor a partir de la imagen de nginx con las siguientes características:

- Nombre: webserver.
- Mapeo de puertos: el 80 del contenedor al 8080 del anfitrión.
- Que se ejecute en segundo plano.

Accede a la web desde Firefox para que se vea.

Ejercicio 1a - Solución

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

Ejercicio 1b

Ahora abre un terminal interactivo con el contenedor y cambia la página web que se ve por defecto, que está en “/usr/share/nginx/html” por una que muestre un título y una imagen.

La imagen tendrás que copiarla desde el anfitrión al contenedor y meterla en la carpeta correspondiente para que se vea en la web.

Ejercicio 2



WORDPRESS



Ejercicio 2

Utilizando las imágenes oficiales de Docker de Wordpress y MySQL monta una página web con Wordpress funcional.

Después añade un contenedor con phpmyadmin que permita la administración de la base de datos.



Ejercicio 2



Paso 1: Ejecutar un contenedor con MySQL

Accede a la documentación oficial de la imagen oficial de MySQL en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **database**.

Ten en cuenta que necesitarás utilizar al menos las siguientes variables de entorno:

- MYSQL_ROOT_PASSWORD
- MYSQL_DATABASE

También tendrás que mapear el puerto que usa MySQL con el anfitrión para poder acceder luego desde el contenedor de Wordpress.



Ejercicio 2 - Solución

Paso 1: Ejecutar un contenedor con MySQL

```
docker run \
  --name mysql \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=pass \
  -e MYSQL_DATABASE=wordpress \
  mysql:8
```



WORDPRESS

Ejercicio 2

Paso 2: Ejecutar un contenedor con Wordpress

Accede a la documentación oficial de Wordpress en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **web**.

No hace falta usar ninguna variable de entorno, ya que tienen una interfaz web de configuración; pero sí que tendrás que mapear el puerto HTTP con el anfitrión para poder acceder desde este a la web.



WORDPRESS

Ejercicio 2

Paso 2: Ejecutar un contenedor con Wordpress

Desde el contenedor de Wordpress se podrá acceder a la base de datos de 2 formas:

1. Usando la opción:

`--link <nombre_contenedor_mysql>:db`

2. A través de la dirección siguiente (siempre que se haya mapeado el puerto de MySQL correctamente):

`ip_interfaz_docker0:puerto`



WORDPRESS

Ejercicio 2 - Solución

Paso 2: Ejecutar un contenedor con Wordpress

```
docker run --name wordpress_01 -p 8080:80 -d wordpress
```

```
docker run --name wordpress_01 --link mysql -p 8080:80 -d wordpress
```

```
docker run --name wordpress_01 --link mysql:db -p 8080:80 -d wordpress
```



Ejercicio 2

Paso 3: Ejecutar un contenedor con PhpMyAdmin

Accede a la documentación oficial de la imagen oficial de PhpMyAdmin en [Docker Hub](#) para saber como funciona, y ejecuta un contenedor a partir de ella llamado **dbadmin**.

Hay que usar la opción:

--link <nombre_contenedor_mysql>:db

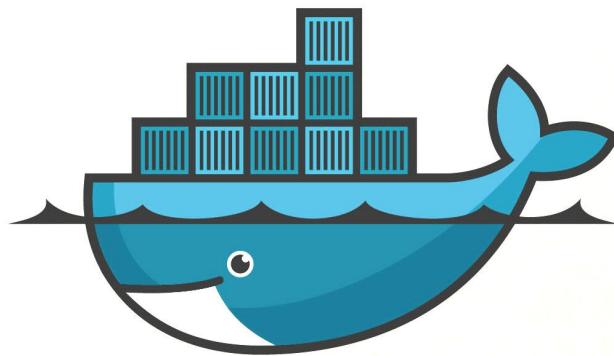


Ejercicio 2 - Solución

Paso 3: Ejecutar un contenedor con PhpMyAdmin

```
docker run --name dbadmin -d --link mysql:db -p 8081:80 phpmyadmin
```

Imágenes



docker

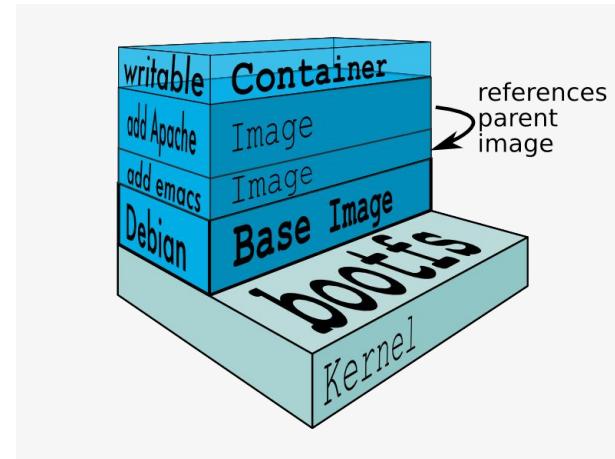
Repaso - *Imágenes*

Las imágenes son **plantillas** para crear contenedores.

A partir de una imagen si pueden crear **múltiples** contenedores.

Pueden tener **parámetros** a concretar al crear el contenedor.

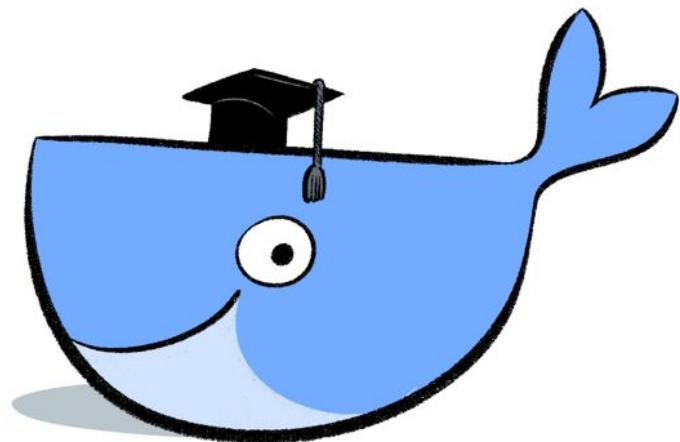
Se crean a partir de otras, añadiendo modificaciones en forma de **capas**.



Contenido

1. Comandos básicos de gestión de imágenes
2. Creación de imágenes
3. Contenedores e imágenes
4. Publicación de imágenes

Comandos básicos de gestión de imágenes



Comandos básicos de gestión de imágenes

1. Listado local
2. Búsqueda en el repositorio
3. Descarga
4. Ver historial / capas
5. Eliminar imagen local
6. Eliminar imágenes y contenedores parados



Listado local

docker images

```
→ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	fpm-alpine	54b80d601342	5 days ago	290MB
wordpress	latest	ef25498ff6a9	5 days ago	605MB
phpmyadmin	fpm-alpine	7625aeffad8a	5 days ago	126MB
phpmyadmin	latest	88a6fdf429bf	5 days ago	508MB
mysql	8	667ee8fb158e	5 days ago	521MB
postgres	14-alpine	114818c12d10	6 days ago	211MB
hello-world	latest	feb5d9fea6a5	6 months ago	13.3kB

Listado local - Filtro básico

```
docker images [REPOSITORIO[:TAG]]
```

```
→ docker images wordpress
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	fpm-alpine	54b80d601342	5 days ago	290MB
wordpress	latest	ef25498ff6a9	5 days ago	605MB

```
→ docker images "*:latest"
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	latest	ef25498ff6a9	5 days ago	605MB
phpmyadmin	latest	88a6fdf429bf	5 days ago	508MB
hello-world	latest	feb5d9fea6a5	6 months ago	13.3kB



Búsqueda en el repositorio

docker search TERM

```
→ docker search wordpress
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
wordpress	The WordPress rich content management system...	4665	[OK]	
bitnami/wordpress	Bitnami Docker Image for WordPress	189	[OK]	
bitnami/wordpress-nginx	Bitnami Docker Image for WordPress with NGINX	57	[OK]	
tutum/wordpress	Out-of-the-box Wordpress docker image	41		
aveltens/wordpress-backup	Easily backup and restore your WordPress blo...	26	[OK]	
arm32v7/wordpress	The WordPress rich content management system...	15		
centurylink/wordpress	Wordpress image with MySQL removed.	14	[OK]	
appsvorg/wordpress-alpine-php	This is a WordPress Docker image which can ...	13		
wordpressdevelop/php	PHP images for the WordPress local developme...	9		
wordpressdevelop/cli	WP-CLI images for the WordPress local develo...	7		
arm64v8/wordpress	The WordPress rich content management system...	7		
dalareo/wordpress-ldap	Wordpress images with LDAP support automatic...	7	[OK]	
ansibleplaybookbundle/wordpress-ha-apb	An APB which deploys Wordpress HA	5	[OK]	
wordpressdevelop/phpunit	PHPUnit images for the WordPress local devel...	5		

Búsqueda en el repositorio

La opción **-f** se puede usar para filtrar según diferentes contextos.

```
→ docker search -f is-official=true wordpress
NAME          DESCRIPTION              STARS      OFFICIAL      AUTOMATED
wordpress     The WordPress rich content management system... 4665      [OK]
```

```
→ docker search -f stars=40 wordpress
NAME          DESCRIPTION              STARS      OFFICIAL      AUTOMATED
wordpress     The WordPress rich content management system... 4665      [OK]
bitnami/wordpress Bitnami Docker Image for WordPress           189       [OK]
bitnami/wordpress-nginx Bitnami Docker Image for WordPress with NGINX 57        [OK]
tutum/wordpress   Out-of-the-box Wordpress docker image         41
```

Búsqueda en el repositorio

La opción **-limit** limita el número de resultados (25 por defecto).

```
→ docker search --limit 4 wordpress
NAME                  DESCRIPTION              STARS      OFFICIAL      AUTOMATED
wordpress             The WordPress rich content management system... 4665      [OK]
bitnami/wordpress     Bitnami Docker Image for WordPress          189       [OK]
bitnami/wordpress-nginx Bitnami Docker Image for WordPress with NGINX 57        [OK]
bitnami/wordpress-intel                                0
```



Descarga

docker pull REPOSITORIO[:TAG]

```
→ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
c229119241af: Already exists
2215908dc0a2: Pull complete
08c3cb2073f1: Pull complete
18f38162c0ce: Pull complete
10e2168f148a: Pull complete
c4ffe9532b5f: Pull complete
Digest: sha256:2275af0f20d71b29391
Status: Downloaded newer image for docker.io/library/nginx:latest

→ docker pull nginx:alpine
alpine: Pulling from library/nginx
40e059520d19: Already exists
f206cf0d6188: Pull complete
065a4ca9176e: Pull complete
67124ec378c3: Pull complete
b17ba2c5bc9f: Pull complete
fed8f5509a6a: Pull complete
Digest: sha256:44e208ac2000daeff77c27a409d1794d6bbdf52067de627c2da13e36c7d59582
Status: Downloaded newer image for nginx:alpine
docker.io/library/nginx:alpine
```

Descarga

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	fpm-alpine	54b80d601342	5 days ago	290MB
wordpress	latest	ef25498ff6a9	5 days ago	605MB
phpmyadmin	fpm-alpine	7625aeffad8a	5 days ago	126MB
phpmyadmin	latest	88a6fdf429bf	5 days ago	508MB
mysql	8	667ee8fb158e	5 days ago	521MB
nginx	alpine	53722defe627	6 days ago	23.4MB
nginx	latest	12766a6745ee	6 days ago	142MB
postgres	14-alpine	114818c12d10	6 days ago	211MB
hello-world	latest	feb5d9fea6a5	6 months ago	13.3kB



Ver historial / capas

docker history IMAGE[:TAG]

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	latest	ef25498ff6a9	5 days ago	605MB
phpmyadmin	latest	88a6fdf429bf	5 days ago	508MB
mysql	8	667ee8fb158e	6 days ago	521MB
nginx	alpine	53722defe627	6 days ago	23.4MB
hello-world	latest	feb5d9fea6a5	6 months ago	13.3kB

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
feb5d9fea6a5	6 months ago	/bin/sh -c #(nop) CMD ["/hello"]	0B	
<missing>	6 months ago	/bin/sh -c #(nop) COPY file:50563a97010fd7ce...	13.3kB	

Ver historial / capas

```
→ docker history nginx:alpine
```

IMAGE	CREATED	CREATED BY		SIZE	COMMENT
53722defe627	6 days ago	/bin/sh -c #(nop) CMD ["nginx" "-g" "daemon..."]		0B	
<missing>	6 days ago	/bin/sh -c #(nop) STOP SIGNAL SIGQUIT		0B	
<missing>	6 days ago	/bin/sh -c #(nop) EXPOSE 80		0B	
<missing>	6 days ago	/bin/sh -c #(nop) ENTRYPOINT ["/docker-entr..."]		0B	
<missing>	6 days ago	/bin/sh -c #(nop) COPY file:09a214a3e07c919a...		4.61kB	
<missing>	6 days ago	/bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7...		1.04kB	
<missing>	6 days ago	/bin/sh -c #(nop) COPY file:0b866ff3fc1ef5b0...		1.96kB	
<missing>	6 days ago	/bin/sh -c #(nop) COPY file:65504f71f5855ca0...		1.2kB	
<missing>	6 days ago	/bin/sh -c set -x && addgroup -g 101 -S ...		17.8MB	
<missing>	6 days ago	/bin/sh -c #(nop) ENV PKG_RELEASE=1		0B	
<missing>	6 days ago	/bin/sh -c #(nop) ENV NJS_VERSION=0.7.2		0B	
<missing>	6 days ago	/bin/sh -c #(nop) ENV NGINX_VERSION=1.21.6		0B	
<missing>	6 days ago	/bin/sh -c #(nop) LABEL maintainer=NGINX Do...		0B	
<missing>	6 days ago	/bin/sh -c #(nop) CMD ["/bin/sh"]		0B	
<missing>	6 days ago	/bin/sh -c #(nop) ADD file:3b5a33c96fd3c10d0...		5.57MB	



Eliminar imagen local

```
docker rmi /IMAGE[:TAG]
```

```
→ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
wordpress       latest        ef25498ff6a9  5 days ago   605MB
phpmyadmin      fpm-alpine   7625aeffad8a  5 days ago   126MB
phpmyadmin      latest        88a6fdf429bf  5 days ago   508MB
mysql           8            667ee8fb158e  6 days ago   521MB
nginx           alpine       53722defe627  6 days ago   23.4MB
hello-world     latest        feb5d9fea6a5  6 months ago  13.3kB
```

```
→ docker rmi phpmyadmin:fpm-alpine
Untagged: phpmyadmin:fpm-alpine
Untagged: phpmyadmin@sha256:0d2a83968332cc00040e6f2a7ebfff6863d6c
Deleted: sha256:7625aeffad8a1744022b079bcb7684001ee899fbc512c398d
Deleted: sha256:a408c8eb7ae339953ea0ae12846e864e32a739183c8ea076d
Deleted: sha256:b2d73959f5006ece83187567487ac92621295377e393c3c9d
```

Eliminar imagen local

Solo se pueden eliminar imágenes que no tengan contenedor asociado, a menos que se use la opción **-f**.

```
→ docker rmi wordpress                                19:51:15
Error response from daemon: conflict: unable to remove repository reference "wordpress" (must fo
rce) - container 45541b0f4f32 is using its referenced image ef25498ff6a9
```

```
→ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS          PORTS     NAMES
45541b0f4f32   wordpress  "docker-entrypoint.s..."  4 days ago   Exited (0) 4 days ago
f6558ab16dbc   wordpress  "docker-entrypoint.s..."  4 days ago   Exited (0) 4 days ago
aa64663f981a   phpmyadmin  "/docker-entrypoint...."  4 days ago   Exited (0) 4 days ago
```

Eliminar imagen local

```
→ docker rmi -f wordpress
Untagged: wordpress:latest
Untagged: wordpress@sha256:df2edd42c943f0925d4634718d1ed1171ea63e043a39201c0b6cbff9d470d571
Deleted: sha256:ef25498ff6a9743827d71366f30d54ed1afcbe1ae9d65ef4325a635dea7edce8
```

Forzar el borrado de una imagen asociada a un contenedor no impide que el contenedor funcione.



Eliminar imágenes y contenedores parados

```
docker system prune -a
```

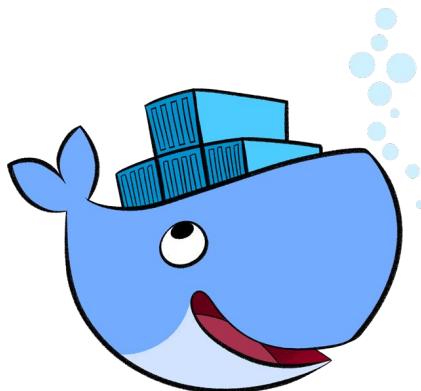
```
→ docker system prune -a
WARNING! This will remove:
 - all stopped containers
 - all networks not used by at least one container
 - all images without at least one container associated to them
 - all build cache
```

```
Are you sure you want to continue? [y/N] y
```

```
Deleted Containers:
```

```
81b9551d89ea17be35f63af6b461527c7c0c415ace5710ed7f1a7406b898223b
45541b0f4f32c6e7ada08b9d95040471914f5d1ca158b2023c0407510ce9f221
f6558ab16dbc1be027e175027ed54690d01ff5581aa7d2502c6f1f8092e62d05
```

Creación de imágenes



Creación de imágenes

1. A partir de un contenedor
2. Exportando / Importando archivos
3. Usando el archivo Dockerfile



A partir de un contenedor

Si queremos hacer persistentes las modificaciones de un contenedor, tenemos que convertirlo en una imagen.

A partir de esa nueva imagen, se podrán crear otros contenedores que tendrán las modificaciones del inicial.

Para convertir un contenedor en una imagen, se utiliza el comando **docker commit**:

```
docker commit [opciones] id/name [REPOSITORY[:TAG]]
```

A partir de un contenedor

```
→ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ad6662e312f2 hello-world "/hello" 7 seconds ago Exited (0) 6 seconds ago
20f8c843487d phpmyadmin "/docker-entrypoint...." 2 minutes ago Exited (0) 23 seconds ago
dead1ecb588d mysql:8 "docker-entrypoint.s..." 2 minutes ago Exited (0) 23 seconds ago
08eea60ef6c6 wordpress "docker-entrypoint.s..." 2 minutes ago Exited (0) 23 seconds ago
```

```
→ docker commit sleepy_montalcini
sha256:4cc5c4d14a8deb3aa7ba6e15e633ff4b9673f023c323df1c84a4f071630917a0
```

```
~
→ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
<none> <none> 4cc5c4d14a8d 5 seconds ago 13.3kB
wordpress latest ef25498ff6a9 5 days ago 605MB
phpmyadmin latest 88a6fdf429bf 5 days ago 508MB
mysql 8 667ee8fb158e 6 days ago 521MB
hello-world latest feb5d9fea6a5 6 months ago 13.3kB
```

A partir de un contenedor

```
→ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
<none>          <none>       4cc5c4d14a8d  5 seconds ago  13.3kB
wordpress        latest        ef25498ff6a9   5 days ago    605MB
phpmyadmin       latest        88a6fdf429bf   5 days ago    508MB
mysql            8             667ee8fb158e   6 days ago    521MB
hello-world      latest        feb5d9fea6a5   6 months ago   13.3kB
```

```
~
→ docker run 4cc5c4d14a8d
```

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
→ docker ps -a
CONTAINER ID  IMAGE          COMMAND           CREATED        STATUS          PORTS     NAMES
3bdabb274a67  4cc5c4d14a8d  "/hello"         11 seconds ago  Exited (0) 9 seconds ago
ad6662e312f2  hello-world   "/hello"         5 minutes ago   Exited (0) 5 minutes ago
```

A partir de un contenedor

Sin usar opciones, la imagen se crea sin repositorio ni etiqueta, lo que es poco útil más allá de una prueba.

Lo normal es asignar:

- Autor
- Comentario
- Repositorio
- Etiqueta

docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	4cc5c4d14a8d	5 seconds ago	13.3kB
wordpress	latest	ef25498ff6a9	5 days ago	605MB

A partir de un contenedor

```
docker commit -a "Autor" -m "Mensaje" id/name REPOSITORY[:TAG]]
```

```
→ docker commit -a "Pepe" -m "Prueba de imagen" sleepy_montalcini pepe/hello-mod:001  
sha256:9ffb433aff99e74bac82d3d26c3986919bec0c62903bd7715a552f5aecdb63d7
```

~

```
→ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
pepe/hello-mod	001	9ffb433aff99	8 seconds ago	13.3kB
<none>	<none>	4cc5c4d14a8d	12 minutes ago	13.3kB
wordpress	latest	ef25498ff6a9	5 days ago	605MB
phpmyadmin	latest	88a6fdf429bf	5 days ago	508MB
mysql	8	667ee8fb158e	6 days ago	521MB
hello-world	latest	feb5d9fea6a5	6 months ago	13.3kB

A partir de un contenedor

```
→ docker run pepe/hello-mod:001
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
→ docker ps -a
CONTAINER ID        IMAGE               COMMAND           CREATED          STATUS            PORTS      NAMES
7b11ccf8ddbb        pepe/hello-mod:001   "/hello"         3 minutes ago   Exited (0) 3 minutes ago   practical_lamarr
3bdabb274a67        4cc5c4d14a8d       "/hello"         14 minutes ago  Exited (0) 14 minutes ago  dreamy_meitner
ad6662e312f2        hello-world        "/hello"         19 minutes ago  Exited (0) 19 minutes ago  sleepy_montalcini
```



Exportando / Importando archivos

```
docker save IMAGE > archivo.tar  
docker save -o archivo.tar IMAGE
```

El comando **save** permite exportar una imagen a un archivo -tar.

```
→ docker save pepe/hello-mod:001 > pepe_hello_mod.001.tar  
~  
→ ls -l pepe_hello_mod.001.tar  
-rw-rw-r-- 1 pepe pepe 23552 abr 4 20:55 pepe_hello_mod.001.tar
```



Exportando / Importando archivos

```
docker load < archivo.tar  
docker load -i archivo.tar
```

El comando **load** permite importar una imagen exportada con el comando **save**.

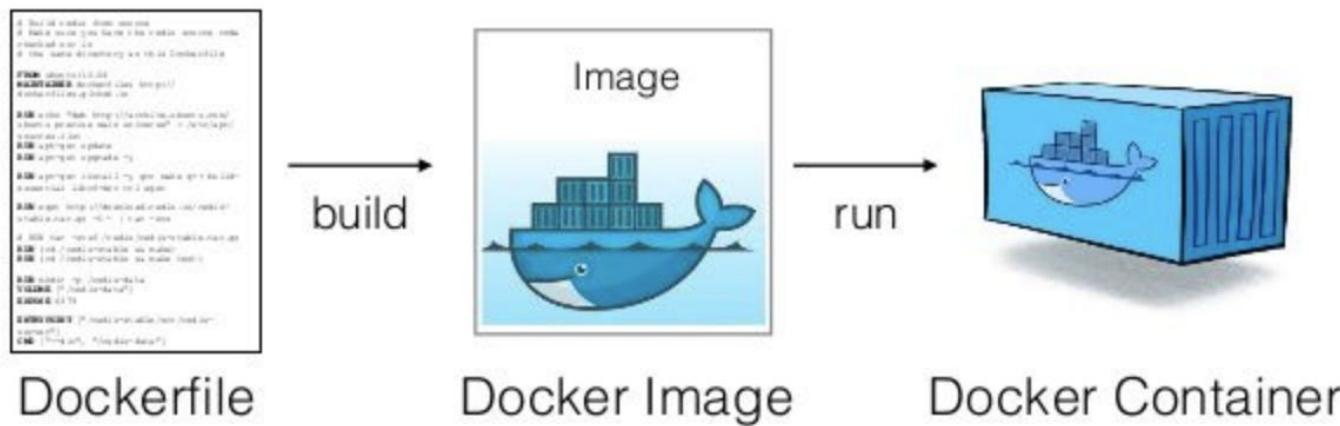
```
→ docker images  
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE  
wordpress        latest        ef25498ff6a9  5 days ago    605MB  
phpmyadmin       latest        88a6fdf429bf  5 days ago    508MB  
mysql            8             667ee8fb158e  6 days ago    521MB  
hello-world      latest        feb5d9fea6a5  6 months ago   13.3kB  
  
~  
→ docker load -i pepe_hello_mod.001.tar  
Loaded image: pepe/hello-mod:001  
  
~  
→ docker images  
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE  
pepe/hello-mod  001          9ffb433aff99  23 minutes ago  13.3kB  
wordpress        latest        ef25498ff6a9  5 days ago    605MB  
phpmyadmin       latest        88a6fdf429bf  5 days ago    508MB  
mysql            8             667ee8fb158e  6 days ago    521MB  
hello-world      latest        feb5d9fea6a5  6 months ago   13.3kB
```



Usando el archivo Dockerfile

La última opción para crear imágenes, y quizá la más utilizada, es el archivo **Dockerfile**.

Este archivo contiene los **comandos** necesarios para crear paso a paso una imagen a partir de otra.



Usando el archivo Dockerfile

```
FROM ubuntu:latest  
  
RUN apt update && apt install -y nano  
  
# Aquí un comentario  
  
CMD /bin/bash
```

Dockerfile

Dockerfile es el nombre por defecto que debe tener el archivo.



Usando el archivo Dockerfile

docker build [opciones] PATH

Para crear la imagen, se usa el comando **docker build**:

```
~/Documentos/docker-examples
* ls
Dockerfile
```

```
~/Documentos/docker-examples
→ docker build .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
4d32b49e2995: Pull complete
Digest: sha256:bea6d19168bbfd6af8d77c2cc3c572114eb5d113e6f422573c93cb605a0e2ffb
Status: Downloaded newer image for ubuntu:latest
---> ff0fea8310f3
→ Step 2/3 : RUN apt update && apt install -y nano
---> Running in 49167e9cf4de
```

Usando el archivo Dockerfile

Sin usar opciones, la imagen se crea sin repositorio ni etiqueta, lo que es poco útil más allá de una prueba.

```
~/Documentos/docker-examples
→ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	4db5e6df3016	4 seconds ago	108MB
pepe/hello-mod	001	9ffb433aff99	20 hours ago	13.3kB

Usando el archivo Dockerfile

La opción **-t** permite especificar nombre del repositorio y etiqueta:

```
~/Documentos/docker-examples
→ docker build -t ubuntu_pepe:v1.0 .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM ubuntu:latest
--> ff0fea8310f3
Step 2/3 : RUN apt update && apt install -y nano
--> Using cache
--> 9f911dda0bbb
Step 3/3 : CMD /bin/bash
--> Using cache
--> 4db5e6df3016
Successfully built 4db5e6df3016
Successfully tagged ubuntu_pepe:v1.0

~/Documentos/docker-examples
→ docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
ubuntu_pepe         v1.0      4db5e6df3016  23 minutes ago  108MB
pepe/hello-mod      001      9ffb433aff99  21 hours ago   13.3kB
wordpress           latest     ef25498ff6a9   6 days ago    605MB
phpmyadmin          latest     88a6fdf429bf  6 days ago    508MB
mysql               8         667ee8fb158e  6 days ago    521MB
ubuntu              latest     ff0fea8310f3  2 weeks ago   72.8MB
hello-world          latest     feb5d9fea6a5   6 months ago  13.3kB
```

Docker detecta que la imagen ya estaba y la sustituye.

Usando el archivo Dockerfile

La opción **-f** permite especificar la ruta al archivo Dockerfile (nombre incluido)

1 Copiamos el Dockerfile con otro nombre

2 Construimos la imagen con otra etiqueta

3 Docker detecta que es la misma imagen, pero al tener otra etiqueta, crea otra imagen.

```
~/Documentos/docker-examples
→ cp Dockerfile Dockerfile.local

~/Documentos/docker-examples
→ docker build -t ubuntu_pepe:v1.1 -f Dockerfile.local .
Sending build context to Docker daemon 3.072kB
Step 1/3 : FROM ubuntu:latest
--> ff0fea8310f3
Step 2/3 : RUN apt update && apt install -y nano
--> Using cache
--> 9f911dda0bbb
Step 3/3 : CMD /bin/bash
--> Using cache
--> 4db5e6df3016
Successfully built 4db5e6df3016
Successfully tagged ubuntu_pepe:v1.1

~/Documentos/docker-examples
→ docker images
REPOSITORY      TAG        IMAGE ID      CREATED       SIZE
ubuntu_pepe     v1.0      4db5e6df3016  37 minutes ago  108MB
ubuntu_pepe     v1.1      4db5e6df3016  37 minutes ago  108MB
pepe/hello-mod  001      9ffb433aff99  21 hours ago   13.3kB
wordpress        latest    ef25498ff6a9   6 days ago    605MB
phpmyadmin       latest    88a6fdf429bf  6 days ago    508MB
mysql            8        667ee8fb158e  6 days ago    521MB
ubuntu           latest    ff0fea8310f3  2 weeks ago   72.8MB
hello-world      latest    feb5d9fea6a5  6 months ago  13.3kB
```

Usando el archivo Dockerfile

El archivo Dockerfile admite muchos comandos. Los más importantes son:

- FROM
- RUN
- CMD
- ENTRYPOINT
- EXPOSE
- ADD/COPY
- USER
- WORKDIR
- ENV

A continuación se presenta un breve resumen de cada comando. Para detalles de su utilización, debe consultarse la [documentación oficial](#).



Usando el archivo Dockerfile

FROM se usa para especificar la imagen base. Debe ser la primera instrucción del Dockerfile.



```
FROM ubuntu:latest

RUN apt update && apt install -y nano

# Aquí un comentario

CMD /bin/bash
```



Usando el archivo Dockerfile

RUN se usa para ejecutar un comando y crear una nueva capa en la imagen.

```
FROM ubuntu:latest  
→ RUN apt update && apt install -y nano  
  
# Aquí un comentario  
  
CMD /bin/bash
```



Usando el archivo Dockerfile

CMD se usa para especificar el comando que se ejecutará al ejecutar el contendor.

Solo puede haber un CMD en el Dockerfile. Si hay varios, solo el último tiene efecto.

```
FROM ubuntu:latest  
  
RUN apt update && apt install -y nano  
  
# Aquí un comentario  
  
CMD /bin/bash
```





Usando el archivo Dockerfile

ENTRYPOINT se usa para especificar el comando que se ejecutará al ejecutar el contenedor.

Solo puede haber un ENTRYPOINT en el Dockerfile. Si hay varios, solo el último tiene efecto.

ENTRYPOINT y **CMD** interactúan para determinar el comando que se ejecutará al ejecutarse el contenedor. El resultado depende de si se han declarado o no, y de qué formato se ha usado.

Usando el archivo Dockerfile

En la siguiente tabla se ve el resultado de diferentes combinaciones:

	No ENTRYPPOINT	ENTRYPPOINT exec_entry p1_entry	ENTRYPPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd



Usando el archivo Dockerfile

EXPOSE se usa para indicar los puertos que escucha un contenedor.

Se utiliza a modo de **documentación**, ya que los puertos expuestos son los susceptibles de ser mapeados (publicados) con la opción **-p de docker run**.

```
FROM ubuntu:latest  
  
RUN apt update && apt install -y apache2  
  
→ EXPOSE 80  
  
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```



Usando el archivo Dockerfile

COPY se usa para copiar archivos entre el host y el contenedor.

Solo permite copiar **archivos locales**.

```
FROM ubuntu:latest

RUN apt update && apt install -y apache2

EXPOSE 80

COPY ["index.html","/var/www/html/"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```





Usando el archivo Dockerfile

ADD se usa para copiar archivos entre el host y el contenedor.

Permite copiar **archivos locales y remotos** a través de la url.

También permite **descomprimir** archivos tar comprimidos.

```
FROM ubuntu:latest

RUN apt update && apt install -y apache2

EXPOSE 80

ADD ["index.html","/var/www/html/"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```





Usando el archivo Dockerfile

USER especifica el usuario y/o grupo con los que se ejecutarán el contenedor y los comandos RUN, CMD y ENTRYPOINT.

Por defecto es el root.

```
FROM ubuntu:latest

RUN apt update && apt install -y apache2

EXPOSE 80

→ USER pepe

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```



Usando el archivo Dockerfile

WORKDIR se usa para cambiar el directorio actual para cualquier comando RUN, CMD, ENTRYPOINT, COPY y ADD posterior.

Si no existe, se crea.

```
FROM ubuntu:latest

RUN apt update && apt install -y apache2
EXPOSE 80

USER pepe
WORKDIR /home/pepe
ADD ["files.zip","archivos"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```





Usando el archivo Dockerfile - *ENV*

ENV se usa para crear variables de entorno.

```
FROM ubuntu:latest

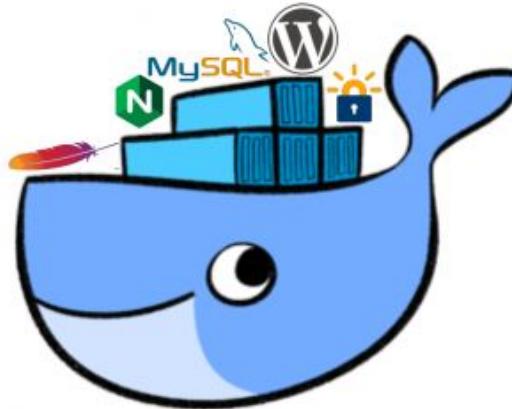
RUN apt update && apt install -y apache2
EXPOSE 80

USER pepe
ENV V1="valor1" V2="valor2"
ENV MY_NAME="John Doe"

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```



Contenedores e imágenes

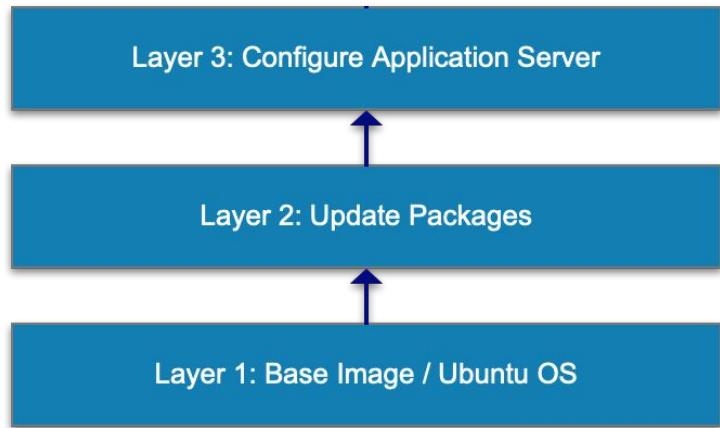


Contenedores e imágenes

Las imágenes se crean **por capas**.

Cada instrucción RUN, COPY y ADD añade **una nueva capa**.

Docker **comparte** las capas iguales entre imágenes para ahorrar espacio.



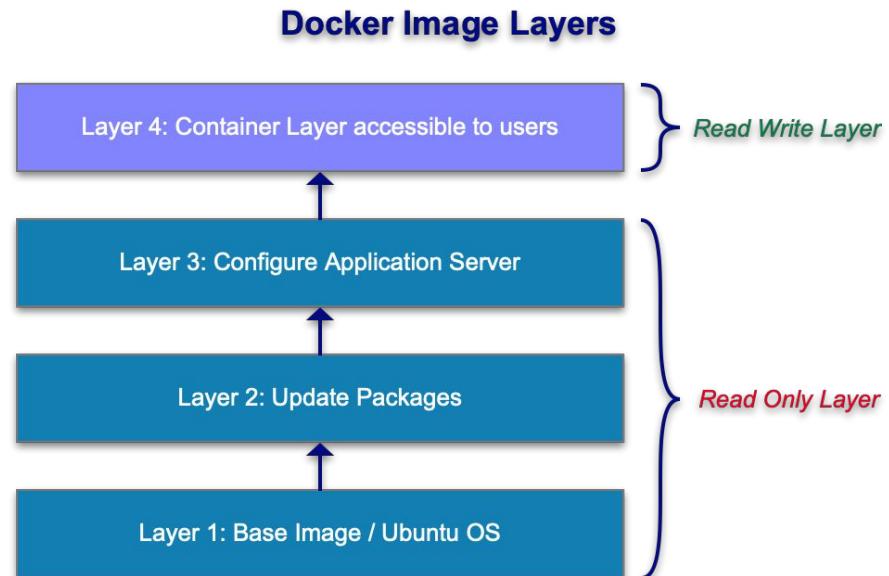
Contenedores e imágenes

<p>ubuntu:14.04 188 mb Layers: 4</p> <p>ADD file:f4d7b4b3402b5c5... 188 mb</p> <p>RUN echo '#!/bin/sh' > /usr... 195 kb</p> <p>RUN sed -i 's/^#s"(deb.*u... 2 kb</p> <p>CMD "/bin/bash" 0 bytes</p>	<p>busybox:latest 2 mb Layers: 3</p> <p>MAINTAINER Jérôme Peta... 0 bytes</p> <p>ADD file:8cf517d90fe79547... 2 mb</p> <p>CMD "/bin/sh" 0 bytes</p>	<p>debian:wheezy 85 mb Layers: 2</p> <p>ADD file:20cd6318f68d34c... 85 mb</p> <p>CMD "/bin/bash" 0 bytes</p>
---	---	---

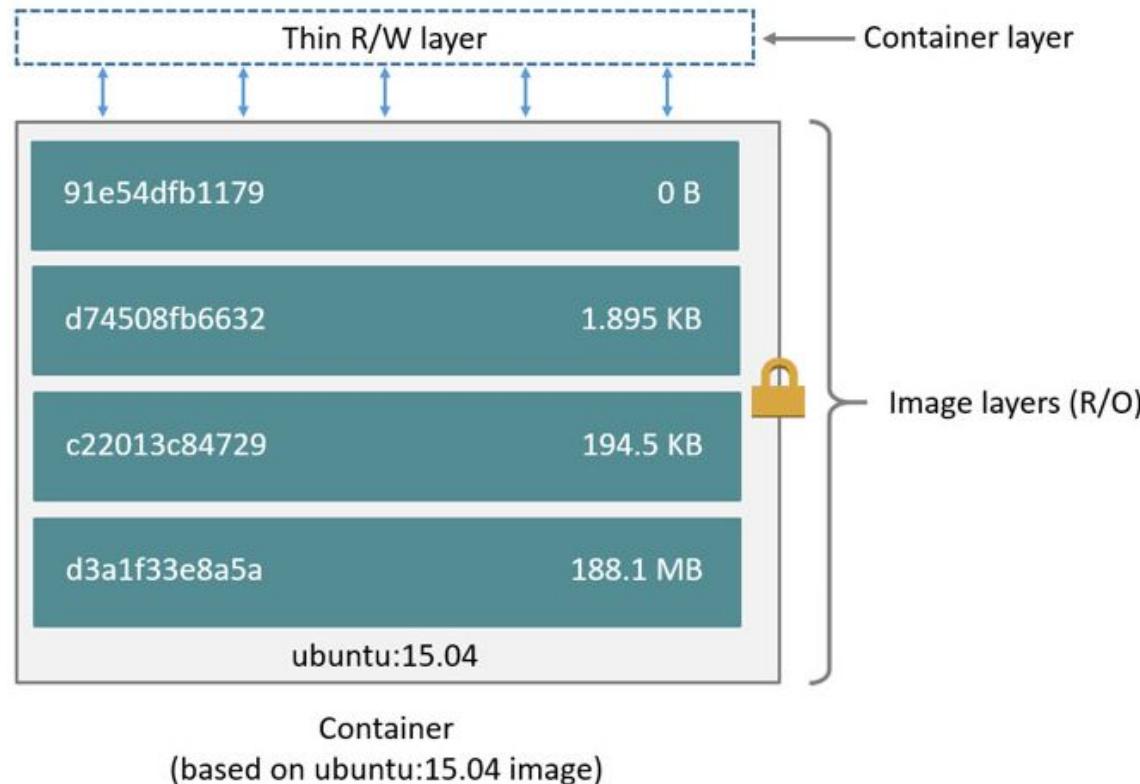
Contenedores e imágenes

Cuando se crea un contenedor, se usan las capas de la imagen como capas de **solo lectura**.

Encima se añade una capa de **lectura y escritura** que contienen las modificaciones hechas por el contenedor.

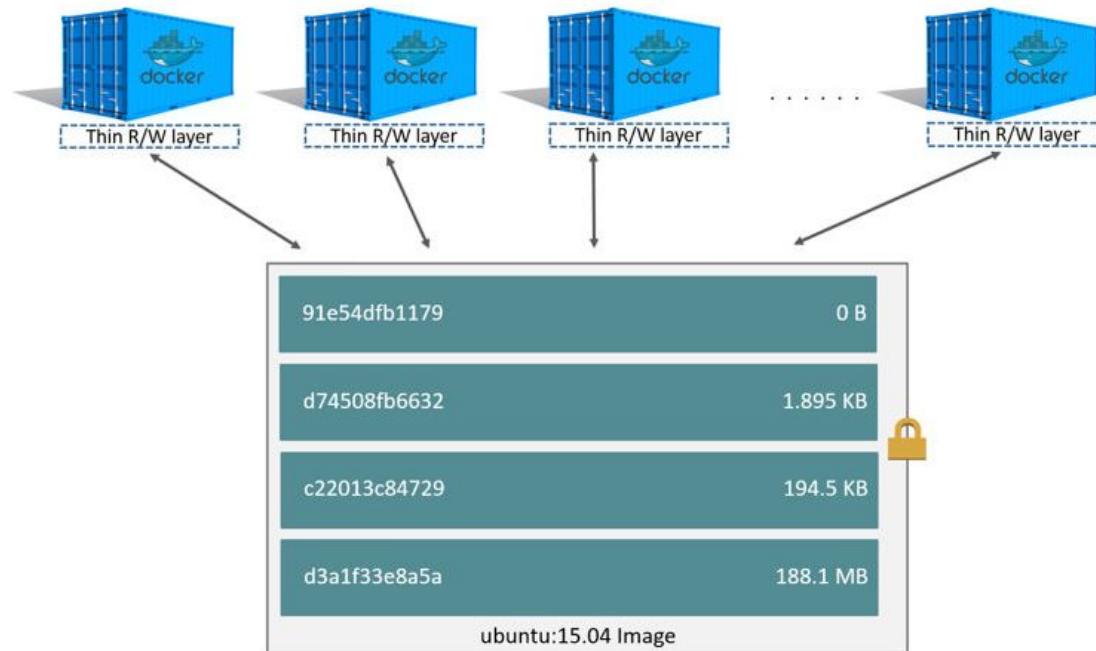


Contenedores e imágenes



Contenedores e imágenes

Varios contenedores creados a partir de una misma imagen, comparten las capas de dicha imagen, ahorrando espacio en disco.



Contenedores e imágenes

Cuando se crea una imagen a partir de un contenedor, usando **docker commit**, la capa de lectura y escritura se convierte en una capa de solo lectura.

docker history nos permite ver las capas:

```
→ docker history hello-world:latest
IMAGE      CREATED      CREATED BY
feb5d9fea6a5  6 months ago  /bin/sh -c #(nop)  CMD ["/hello"]
<missing>  6 months ago  /bin/sh -c #(nop) COPY file:50563a97010fd7ce...  13.3kB
```

```
→ docker history pepe/hello-mod:001
IMAGE      CREATED      CREATED BY
9ffb433aff99  45 hours ago  /hello
<missing>  6 months ago  /bin/sh -c #(nop)  CMD ["/hello"]
<missing>  6 months ago  /bin/sh -c #(nop) COPY file:50563a97010fd7ce...  13.3kB
```

Ejercicio 1

Ejercicio 1

Se tiene una carpeta html con una web estática (index.html) y se pide crear una imagen Docker con el servidor web Apache que muestre la web.

Requisitos:

- Imagen base: debian
- Comandos para instalar Apache:
 - apt-get update
 - apt-get install -y apache2
 - apt-get clean
 - rm -rf /var/lib/apt/lists/*
- Debe exponer el puerto 80
- Comando para ejecutar Apache: /usr/sbin/apache2ctl -D FOREGROUND

index.html

```
<h1>Ejercicio 1</h1>
<p>Hola! Todo OK!</p>
```

Ejercicio 2

Ejercicio 2

Repite el ejercicio 1 pero usando como imagen base la oficial de Apache.

Ejercicio 3

Ejercicio 3

Repite el ejercicio 2 pero usando como imagen base la oficial de Nginx.

Ejercicio 4

Ejercicio 4

Se tiene una carpeta app con una página web dinámica con php (index.php) y se pide crear una imagen Docker con el servidor web Apache y php que muestre la web.

Requisitos:

- Imagen base: debian
- Comandos para instalar Apache:
 - apt-get update
 - apt-get install -y apache2 libapache2-mod-php7.3 php7.3
 - apt-get clean
 - rm -rf /var/lib/apt/lists/*
 - rm /var/www/html/index.html
- Debe exponer el puerto 80
- Comando para ejecutar Apache: /usr/sbin/apache2ctl -D FOREGROUND

index.php

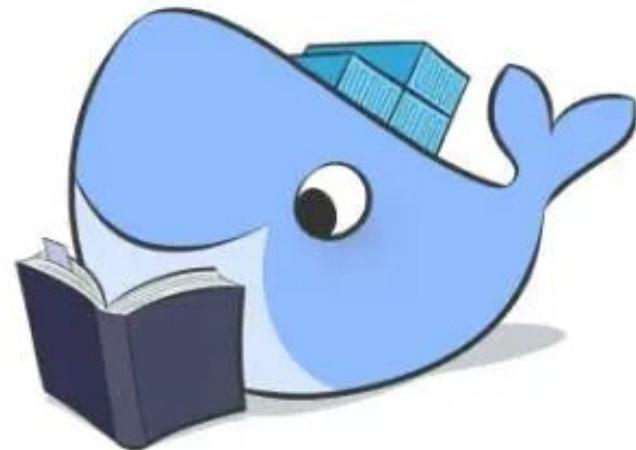
```
<?php echo phpinfo();?>
```

Ejercicio 5

Ejercicio 5

Repite el ejercicio 4 pero usando como imagen base `php:apache`, que es una imagen de oficial de PHP con Apache ya instalado.

Publicación de imágenes



Publicación de imágenes

Se pueden publicar imágenes en el repositorio para ser utilizadas posteriormente.

Para publicar en **Docker Hub**, es necesario tener **una cuenta**.

El procedimiento tiene 2 pasos:

1. Iniciar sesión con **docker login**
2. Subir la imagen con **docker push**



Publicación de imágenes

docker login

```
→ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: pposca
Password:
WARNING! Your password will be stored unencrypted in /home/pepe/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```



Publicación de imágenes

docker push

```
→ docker push pposca/ejer5:v1
The push refers to repository [docker.io/pposca/ejer5]
1bca420e374b: Pushed
78d1dc086f2e: Mounted from library/php
c67db8aa2359: Mounted from library/php
9108a3392c29: Mounted from library/php
fd7fb2d8362b: Mounted from library/php
be00d6205d59: Mounted from library/php
d3ac239ebf10: Mounted from library/php
d1a9206e71d3: Mounted from library/php
70c7e28ebf9b: Mounted from library/php
8da8ace4fad7: Mounted from library/php
a9a0b6ccdc20: Mounted from library/php
1fef65785df5: Mounted from library/php
ea61222fc24f: Mounted from library/php
608f3a074261: Mounted from library/php
v1: digest: sha256:bd57620325d291741a6091383f15cf2bd7abfc81bcfb4e46dca190933dabb468 size: 3242
```

Publicación de imágenes

El nombre de la imagen tienen que ser: **username/imagen:tag**

Repositories Starred Contributed

Displaying 1 of 1 repository



pposca/ejer5
By [pposca](#) • Updated an hour ago

Container

0 Stars

Ejercicio 6

Ejercicio 6

Crea una cuenta el Docker Hub, creo una imagen y súbelo.

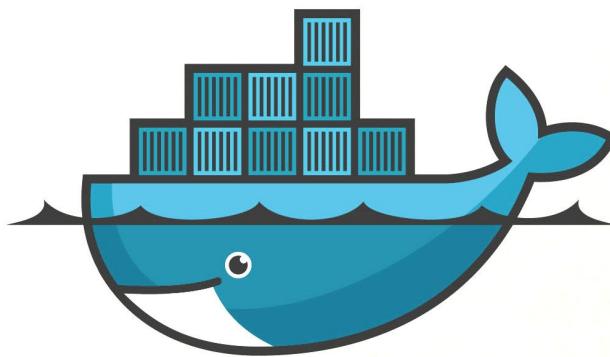
Puede utilizar una imagen de las creadas en los ejercicios anteriores.

Recuerda que la primera parte del nombre de la imagen tiene que coincidir con el nombre de usuario de Docker Hub.

Luego borra la imagen local y usa docker run para que se la descargue del repositorio, cree un contenedor y lo ejecute.

Documenta el proceso con explicaciones y capturas de pantalla.

Gestión de imágenes



docker

Ejercicio 1

Ejercicio 1

Se tiene una carpeta html con una web estática (index.html) y se pide crear una imagen Docker con el servidor web Apache que muestre la web.

Requisitos:

- Imagen base: debian
- Comandos para instalar Apache:
 - apt-get update
 - apt-get install -y apache2
 - apt-get clean
 - rm -rf /var/lib/apt/lists/*
- Debe exponer el puerto 80
- Comando para ejecutar Apache: /usr/sbin/apache2ctl -D FOREGROUND

index.html

```
<h1>Ejercicio 1</h1>
<p>Hola! Todo OK!</p>
```

Ejercicio 1 - Solución

```
→ tree
.
└── Dockerfile
    └── html
        └── index.html
```

```
→ cat html/index.html
<h1>Ejercicio 1</h1>
<p>Hola! Todo OK!</p>
```

```
FROM debian

RUN apt-get update && \
    apt-get install -y apache2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

ADD html /var/www/html/

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Ejercicio 1 - Solución

```
→ docker build -t pepe/ejer1:v1 .
Sending build context to Docker daemon 3.584kB
Step 1/5 : FROM debian
latest: Pulling from library/debian
dbba69284b27: Pull complete
Digest: sha256:87eefc7c15610cca61db5c0fd280911c6a737c0680d807432c0bd80cd0cca39b
Status: Downloaded newer image for debian:latest
---> d69c6cd3a20d
Step 2/5 : RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
---> Running in 8fcc071f6184
```



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
pepe/ejer1	v1	63e5c913d1a6	6 seconds ago	235MB
debian	latest	d69c6cd3a20d	8 days ago	124MB

Ejercicio 1 - Solución

```
→ docker run -d -p 80:80 pepe/ejer1:v1  
35855bacf8a1a83b67c9ef55ab4dbc24c345e65da58537cf8856d173c6bc0aba
```

```
→ docker ps  
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES  
35855bacf8a1   pepe/ejer1:v1   "/usr/sbin/apache2ct..."   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp, :::80->80/tcp   cool_cerf
```



Ejercicio 2

Ejercicio 2

Repite el ejercicio 1 pero usando como imagen base la oficial de Apache.

Ejercicio 2 - Solución

Ya no es necesario instalar Apache ni ejecutarlo, ya que se hacen en la imagen base.

Solo hay que cambiar la ruta de destino de los archivos de la web, que según la documentación de la imagen de Apache es `usr/local/apache2/htdocs/`.

```
→ tree
.
└── Dockerfile
    └── html
        └── index.html
```

```
→ cat html/index.html
<h1>Ejercicio 1</h1>
<p>Hola! Todo OK!</p>
```

```
FROM httpd:2.4

ADD html /usr/local/apache2/htdocs/
```

```
EXPOSE 80
```

Ejercicio 2 - Solución

```
→ docker build -t pepe/ejer1b:v1 .
```

```
Sending build context to Docker daemon 3.584kB  
Step 1/3 : FROM httpd:2.4
```

```
→ docker images  
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE  
pepe/ejer1b    v1      7ff1c88432ba  11 seconds ago  144MB  
pepe/ejer1    v1      63e5c913d1a6  16 minutes ago  235MB  
httpd          2.4      118b6abfbf55  8 days ago    144MB  
debian         latest    d69c6cd3a20d  8 days ago    124MB
```

```
→ docker run -d -p 8080:80 pepe/ejer1b:v1
```

```
f8810abd4ff1986ba1f97c6b130e0f84d72c1050f3997cd80c615100e8b018b1
```

```
→ docker ps  
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES  
f8810abd4ff1    pepe/ejer1b:v1  "httpd-foreground"  About a minute ago  Up About a minute  0.0.0.0:8080->80/tcp, :::8080->80/tcp  thirsty_rubin  
35855bacf8a1    pepe/ejer1:v1  "/usr/sbin/apache2ct..."  15 minutes ago    Up 15 minutes  0.0.0.0:80->80/tcp, :::80->80/tcp  cool_cerf
```

← → C ⌂ ⓘ localhost:8080

Ejercicio 1

Hola! Todo OK!

Ejercicio 3

Ejercicio 3

Repite el ejercicio 2 pero usando como imagen base la oficial de Nginx.

Ejercicio 3 - Solución

```
→ tree
.
└── Dockerfile
    └── html
        └── index.html

→ cat html/index.html
<h1>Ejercicio 1</h1>
<p>Hola! Todo OK!</p>
```

```
FROM nginx

ADD html /usr/share/nginx/html

EXPOSE 80
```

Ejercicio 3 - Solución

```
→ docker build -t pepe/ejer3:v1 .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM nginx
```

```
→ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
pepe/ejer3      v1       e53d7dbae092  About a minute ago  142MB
pepe/ejer1b     v1       7ff1c88432ba  21 hours ago   144MB
pepe/ejer1      v1       63e5c913d1a6  21 hours ago   235MB
nginx           latest    12766a6745ee  8 days ago    142MB
httpd            2.4      118b6abfbf55  9 days ago    144MB
debian           latest    d69c6cd3a20d  9 days ago    124MB
```

```
→ docker run -d -p 8081:80 pepe/ejer3:v1
8889566e11bd89ea85ef4c15c593a22128da7a5bfc5121a99766383edc1d919c
```

← → C ⌂ ⓘ localhost:8081

Ejercicio 1

Hola! Todo OK!

```
→ docker ps
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
8889566e11bd     pepe/ejer3:v1  "/docker-entrypoint..."  11 seconds ago  Up 9 seconds  0.0.0.0:8081->80/tcp, :::8081->80/tcp  nostalgic_johnson
```

Ejercicio 4

Ejercicio 4

Se tiene una carpeta app con una página web dinámica con php (index.php) y se pide crear una imagen Docker con el servidor web Apache y php que muestre la web.

Requisitos:

- Imagen base: debian
- Comandos para instalar Apache:
 - apt-get update
 - apt-get install -y apache2 libapache2-mod-php7.4 php7.4
 - apt-get clean
 - rm -rf /var/lib/apt/lists/*
 - rm /var/www/html/index.html
- Debe exponer el puerto 80
- Comando para ejecutar Apache: /usr/sbin/apache2ctl -D FOREGROUND

index.php

```
<?php echo phpinfo();?>
```

Ejercicio 4 - Solución

```
→ tree
.
└── app
    └── index.php
Dockerfile
```

```
→ \cat app/index.php
<?php echo phpinfo();?>
```

FROM debian

```
RUN apt-get update && \
    apt-get install -y apache2 libapache2-mod-php php && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

```
ADD app /var/www/html/
```

EXPOSE 80

```
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Ejercicio 4 - Solución

```
→ docker build -t pepe/ejer4:v1 .
Sending build context to Docker daemon 3.584kB
Step 1/5 : FROM debian
```

```
→ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
pepe/ejer4      v1      5bfc4cab10a7  4 seconds ago  254MB
pepe/ejer3      v1      e53d7dbae092  40 minutes ago 142MB
```

```
→ docker run -d -p 80:80 pepe/ejer4:v1
bec8023f63fe6b72afac024ffa5cd26b9acad5f9471b3875ae527e81957a93b2
```

```
→ docker ps
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
bec8023f63fe      pepe/ejer4:v1  "/usr/sbin/apache2ct..."  3 minutes ago  Up 3 minutes  0.0.0.0:80->80/tcp, :::80->80/tcp  blissful_turing
```

Ejercicio 4 - Solución

The screenshot shows a web browser window with the URL `localhost/index.php` in the address bar. The page content displays the following information:

PHP Version 7.4.28

php

System	Linux bec8023f63fe 5.11.0-49-generic #55-Ubuntu SMP Wed Jan 12 17:36:34 UTC 2022 x86_64
Build Date	Feb 17 2022 16:17:19
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-ffi.ini, /etc/php/7.4/apache2/conf.d/20-fileinfo.ini, /etc/php/7.4/apache2/conf.d/20-ftp.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-json.ini, /etc/php/7.4/apache2/conf.d/20-phar.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvsem.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini

Ejercicio 5

Ejercicio 5

Repite el ejercicio 4 pero usando como imagen base `php:apache`, que es una imagen de oficial de PHP con Apache ya instalado.

Ejercicio 5 - Solución

```
→ tree
.
└── app
    └── index.php
Dockerfile
```

```
→ \cat app/index.php
<?php echo phpinfo();?>
```

```
FROM php:apache
ADD app /var/www/html/
EXPOSE 80
```

Ejercicio 5 - Solución

```
→ docker build -t pepe/ejer5:v1 .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM php:apache
```

```
→ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
pepe/ejer5      v1          dbbd7d12a183  24 seconds ago  458MB
pepe/ejer4      v1          5bfc4cab10a7   17 minutes ago  254MB
```

```
→ docker run -d -p 80:80 pepe/ejer5:v1
7e2b688b15b523d5dff43e70b04f7ce869c45ccb1be84e0200350aa190160ae
```

```
→ docker ps
CONTAINER ID  IMAGE          COMMAND           CREATED        STATUS          PORTS          NAMES
7e2b688b15b5  pepe/ejer5:v1  "docker-php-entrypoi..."  6 seconds ago  Up 5 seconds  0.0.0.0:80->80/tcp, :::80->80/tcp  interesting_gould
```

Ejercicio 5 - Solución

← → C ⌂ i localhost/index.php

PHP Version 8.1.4 

System	Linux 7e2b688b15b5 5.11.0-49-generic #55-Ubuntu SMP Wed Jan 12 17:36:34 UTC 2022 x86_64
Build Date	Mar 29 2022 01:23:23
Build System	Linux c9ba2fd33143 5.10.0-11-cloud-amd64 #1 SMP Debian 5.10.92-2 (2022-02-28) x86_64 GNU/Linux
Configure Command	<pre>./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlind' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'</pre>
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)

Ejercicio 6

Ejercicio 6

Crea una cuenta el Docker Hub, creo una imagen y súbelo.

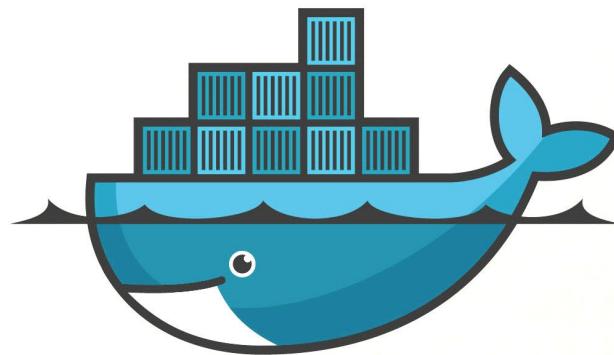
Puede utilizar una imagen de las creadas en los ejercicios anteriores.

Recuerda que la primera parte del nombre de la imagen tiene que coincidir con el nombre de usuario de Docker Hub.

Luego borra la imagen local y usa docker run para que se la descargue del repositorio, cree un contenedor y lo ejecute.

Documenta el proceso con explicaciones y capturas de pantalla.

Almacenamiento

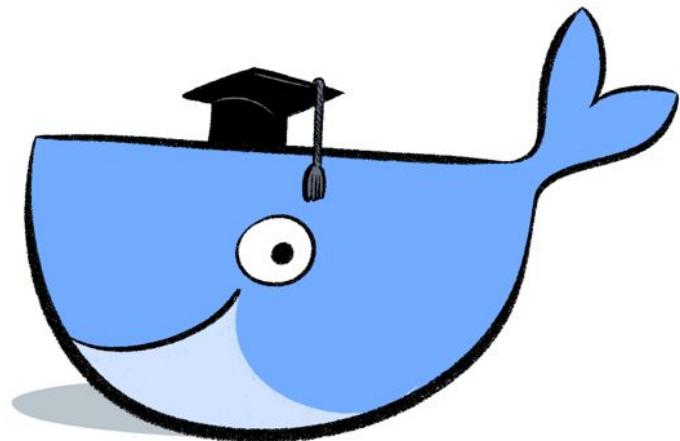


docker

Contenido

1. Persistencia de datos
2. *Bind mount*
3. Volúmenes

Persistencia de datos



Persistencia de datos

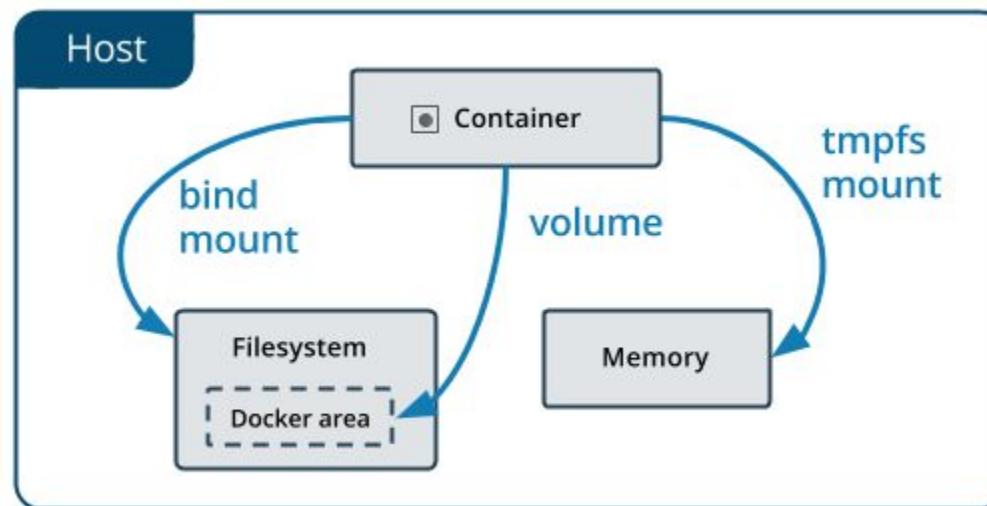
Los contenedores son **efímeros**; es decir, su configuración y sus datos sobreviven a paradas, pero se borran cuando se elimina el contenedor.

Para persistir los datos, Docker proporciona 2 sistemas:

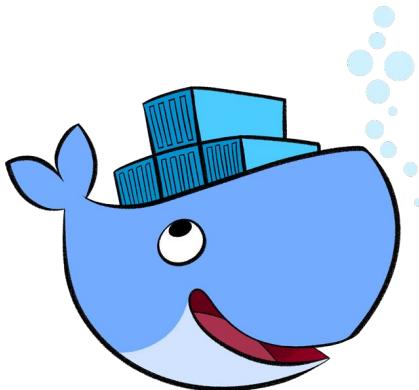
- ***Bind mount***
- **Volúmenes**

Persistencia de datos

También existe el **tmpfs mount**, para montar un sistema de ficheros en memoria, pero no lo vamos a ver.



Bind mount





Bind mount

Consiste en **montar directorios o ficheros** del host en el contenedor, de modo que ambos tienen acceso. Si el destino tenía datos, dejarán de ser accesibles.

Se establece **al crear el contenedor**, con opciones de docker run o create.

Require **rutas absolutas**.

Tienen 2 sintaxis:

- Simple: usando la opción **-v** o **--volume**
- Compleja: usando la opción **--mount**

Bind mount - Sintaxis simple

```
-v ruta_host:ruta_contenedor:opciones
```

Si **ruta_host** es un directorio y no existe, se crea.

opciones es opcional. La única que vamos a ver es **ro**, que significa **read-only**.

Bind mount - Sintaxis simple - Ejemplo

Vamos a lanzar un contenedor Nginx que muestre una web que está en un directorio del host.

```
~/Documentos/docker-examples/06-ejemplo01
→ tree
.
└── html
    └── index.html
```

```
~/Documentos/docker-examples/06-ejemplo01
→ \cat html/index.html
<h1>Almacenamiento</h1>
<p>Ejemplo 1 - bind mount</p>
```

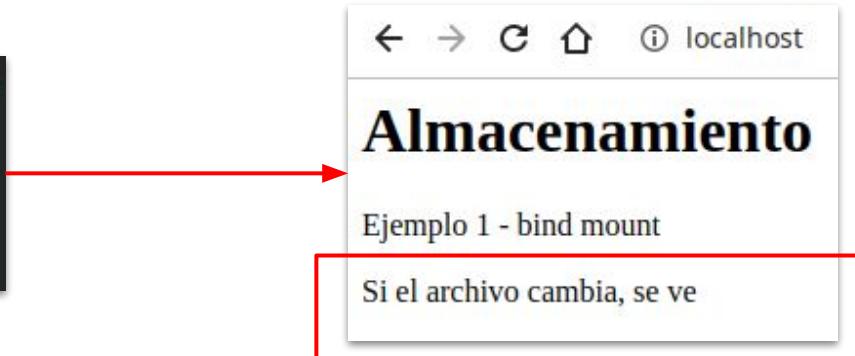
```
~/Documentos/docker-examples/06-ejemplo01
→ docker run -d -p 80:80 -v ~/Documentos/docker-examples/06-ejemplo01/html:/usr/share/nginx/html nginx
8b49c8abc731f22a99f76b6505dffbbcc3ff6c029015af43e6fc4fe086e9d0d6
```

Bind mount - Sintaxis simple - Ejemplo



Si cambiamos el archivo index.html desde el host, el contenedor lo ve.

```
~/Documentos/docker-examples/06-ejemplo01
→ \cat html/index.html
<h1>Almacenamiento</h1>
<p>Ejemplo 1 - bind mount</p>
<p>Si el archivo cambia, se ve</p>
```



Bind mount - Sintaxis simple - Ejemplo

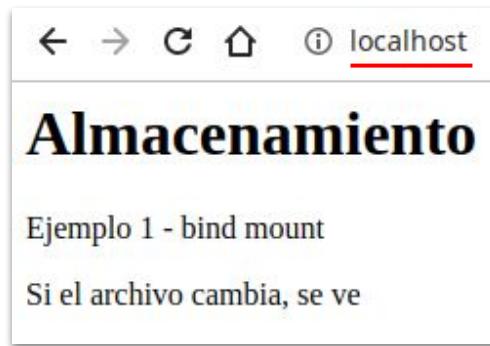
En este caso, como Nginx solo lee el archivo, sería más seguro montarlo con **ro**:

```
~/Documentos/docker-examples/06-ejemplo01
→ docker run -d -p 80:80 -v ~/Documentos/docker-examples/06-ejemplo01/html:/usr/share/nginx/html:ro nginx
bca1e32d2f72f7e88bc5ca36afafc40ed3e4e31c0bc66ea99c6e4cbef0911507
```

También se podría crear otro contenedor con el mismo directorio montado:

```
~/Documentos/docker-examples/06-ejemplo01
→ docker run -d -p 8080:80 -v ~/Documentos/docker-examples/06-ejemplo01/html:/usr/share/nginx/html nginx
420e325750924df2aaa7ac0a0f166a072b72215f7ae0e0b6632311c11469bb13
```

Bind mount - Sintaxis simple - Ejemplo



```
~/Documentos/docker-examples/06-ejemplo01
$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS          NAMES
420e32575092        nginx      "/docker-entrypoint...."   3 minutes ago   Up 3 minutes   0.0.0.0:8080->80/tcp, :::8080->80/tcp   happy_franklin
bca1e32d2f72        nginx      "/docker-entrypoint...."   6 minutes ago   Up 6 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp    busy_zhukovsky
```

Bind mount - Sintaxis compleja

Usa la opción **--mount** de docker run o create.

No la vamos a ver en detalle, sino en comparación con el ejemplo de la simple.

Simple:

```
-v ~/Documentos/docker-examples/06-ejemplo01/html:/usr/share/nginx/html
```

Complejo:

```
--mount type=bind,src=/home/pepe/Documentos/docker-examples/06-ejemplo01/html,dst=/usr/share/nginx/html
```

Bind mount - Sintaxis compleja

Simple solo lectura:

```
-v ~/Documentos/docker-examples/06-ejemplo01/html:/usr/share/nginx/html:ro
```

Complejo solo lectura:

```
--mount type=bind,src=/home/pepe/Documentos/docker-examples/06-ejemplo01/html,dst=/usr/share/nginx/html,readonly
```

Ejercicio 1

Ejercicio 1

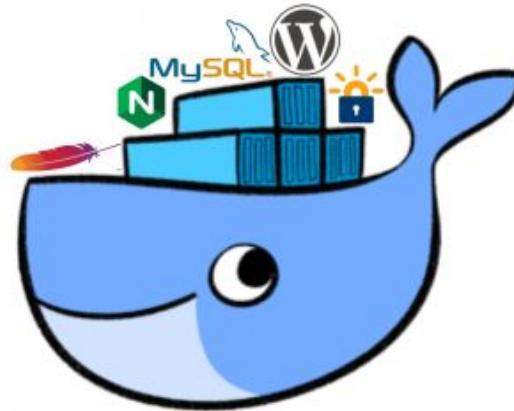
1. Crea un contenedor a partir de la imagen oficial de NextCloud usando *bind mount* para persistir los datos en una carpeta llamado “datos”. Selecciona como base de datos SQLite durante la instalación.
2. Sube algún archivo.
3. Comprueba desde el host que el archivo está dentro de la carpeta “datos”.
4. Luego para el contenedor y bórralo (el contenedor, no el archivo).
5. Finalmente crea otro contenedor usando el mismo directorio “datos” que has usado en el primero y comprueba que el archivo que has subido sigue ahí en la interfaz de NextCloud.

Ejercicio 2

Ejercicio 2

1. Crea un contenedor a partir de la imagen oficial de MySQL usando *bind mount* para persistir las bases de datos en una carpeta llamado “db”.
2. Lanza un contenedor phpmyadmin que se conecte al MySQL y crea una base de datos.
3. Para y elimina los contenedores.
4. Vuelve a hacer los pasos 1 y 2 y comprueba que la base de datos ha persistido.

Volúmenes



Volúmenes

Los volúmenes son similares a los bind mounts, pero los **gestiona Docker**, por lo que **no** es necesario especificar la **ruta**.

En cada sistema se almacenan en un sitio. Por ejemplo, en Linux están en /var/lib/docker/volumes.

Son entidades propias. Se pueden **crear, listar, eliminar e inspeccionar**.

Tienen las mismas 2 sintaxis que los bind mounts, pero usando el nombre del volumen:

- Simple: usando la opción **-v** o **--volume**
- Compleja: usando la opción **--mount**

Volúmenes vs *bind mounts*

Volúmenes

- Se refieren por **nombre**.
- La ruta **no** importa.
- Pensados para la **comunicación entre contenedores**.
- Si el destino en el contenedor tiene datos y el volumen está vacío, **los datos se copian al volumen** al montarse.

Bind mounts

- Se refieren por **ruta**.
- La ruta es **importante**.
- Pensados para la **comunicación entre el host y los contenedores**.
- Si el destino en el contenedor tiene datos, al montar el origen esos datos quedan **inaccesibles**.

Volúmenes - Comandos

docker volume comando

<code>create <i>nombre</i></code>	Crea un volumen
<code>ls</code>	Lista todos los volúmenes
<code>rm <i>nombre</i></code>	Elimina un volumen
<code>prune</code>	Elimina los volúmenes no usados por ningún contenedor
<code>inspect</code>	Proporciona información sobre el volumen



Volúmenes - Comandos

```
docker volume create nombre
```

```
→ docker volume create web  
web
```

```
→ docker volume create database  
database
```



Volúmenes - Comandos

docker volume ls

```
→ docker volume ls
DRIVER      VOLUME NAME
local       database
local       web
```



Volúmenes - Comandos

docker volume inspect name

```
→ docker volume inspect database
[
  {
    "CreatedAt": "2022-04-11T17:54:48+02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/database/_data",
    "Name": "database",
    "Options": {},
    "Scope": "local"
  }
]
```



Volúmenes - Comandos

```
docker volume rm name
```

```
→ docker volume rm database
database
```

```
→ docker volume ls
DRIVER      VOLUME NAME
local       web
```



Volúmenes - Comandos

docker volume prune

```
→ docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
web

Total reclaimed space: 0B
```

```
→ docker volume ls
DRIVER      VOLUME NAME
```

Volúmenes - Sintaxis compleja

```
→ docker run -d -p 80:80 --mount type=volume,src=web2,dst=/usr/share/nginx/html nginx  
0d0a304dc6195886d71d5e77879a945542081d399ec8dc5a45910c0772042f99
```

Si el volumen no existe, se crea automáticamente.

```
→ docker volume ls  
DRIVER      VOLUME NAME  
local        web  
local        web2
```

Si se omite el src, se crea un volumen anónimo.

```
→ docker volume ls  
DRIVER      VOLUME NAME  
local        ffbe44ea68044c29066cbcaed7b881c7ba980d68b865  
local        web  
local        web2
```

Ejercicio 1

Ejercicio 1

Haz el ejercicio 1 de bind mount usando ahora volúmenes.

Ejercicio 2

Ejercicio 2

Haz el ejercicio 2 de bind mount usando ahora volúmenes.

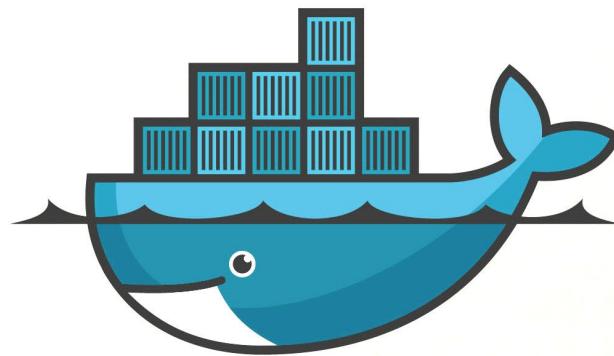
Ejercicio 3

Ejercicio 3

A partir del ejercicio anterior, en que la persistencia de la base de datos mysql se está haciendo en un volumen, se pide crear un contenedor que haga un archivo tar.gz con el contenido del volumen en un directorio del host llamado backup.

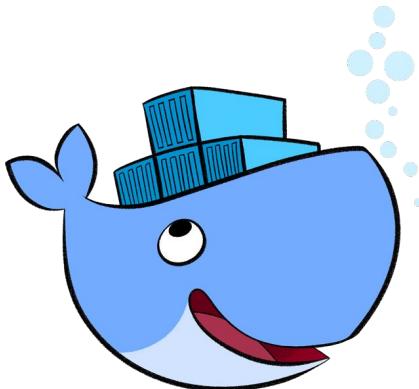
El contenedor tienen que borrarse después de hacer la copia.

Almacenamiento



docker

Bind mount



Ejercicio 1

Ejercicio 1

1. Crea un contenedor a partir de la imagen oficial de NextCloud usando *bind mount* para persistir los datos en una carpeta llamado “datos”. Selecciona como base de datos SQLite durante la instalación.
2. Sube algún archivo.
3. Comprueba desde el host que el archivo está dentro de la carpeta “datos”.
4. Luego para el contenedor y bórralo (el contenedor, no el archivo).
5. Finalmente crea otro contenedor usando el mismo directorio “datos” que has usado en el primero y comprueba que el archivo que has subido sigue ahí en la interfaz de NextCloud.

Ejercicio 1 - Solución

1. Crea un contenedor a partir de la imagen oficial de NextCloud usando *bind mount* para persistir los datos en una carpeta llamado “datos”. Selecciona como base de datos SQLite durante la instalación.

```
~/Documentos/docker-examples/06-ejer01
→ docker run -d -p 80:80 -v $(pwd)/datos:/var/www/html --name nextcloud nextcloud
0af12d95c5ab50abd761725bd58aa5b66e5f5210b767391ebdf503c58a8fdeec
```

Ejercicio 1 - Solución

```
~/Documentos/docker-examples/06-ejer01
→ l
total 12K
drwxrwxr-x 3 pepe      pepe 4,0K abr 11 16:43 .
drwxrwxr-x 9 pepe      pepe 4,0K abr 11 16:28 ..
drwxr-xr-x 4 www-data  root 4,0K abr 11 16:43 datos
```

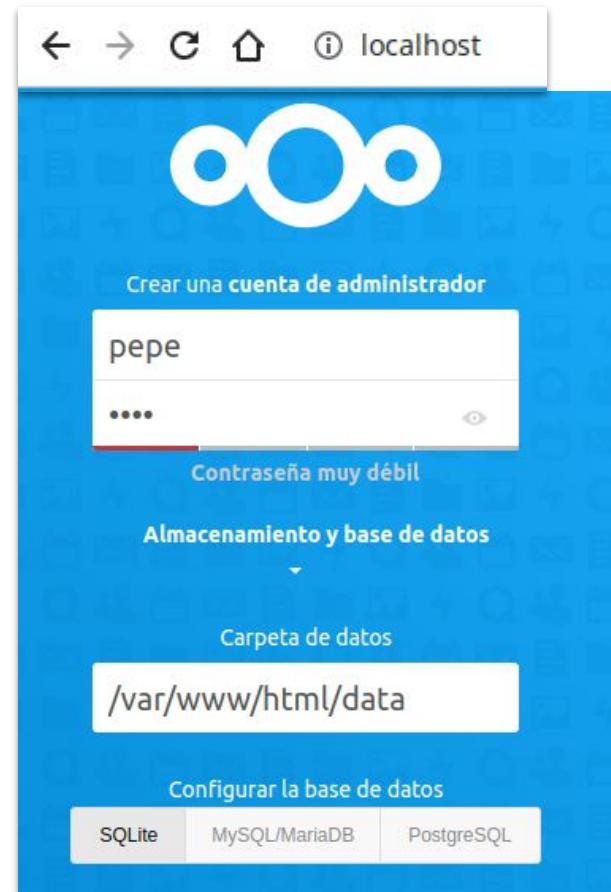
Contiene los datos de NextCloud:
configuraciones, archivos subidos,
etc.

Se ha creado la carpeta “datos”

```
~/Documentos/docker-examples/06-ejer01
→ l datos
total 168K
drwxr-xr-x 14 www-data  root 4,0K abr 11 16:43 .
drwxrwxr-x  3 pepe      pepe 4,0K abr 11 16:43 ..
drwxr-xr-x 44 www-data  root 4,0K abr 11 16:43 3rdparty
drwxr-xr-x 48 www-data  root 4,0K abr 11 16:43 apps
-rw-r--r--  1 www-data  root 19K abr 11 16:43 AUTHORS
drwxr-xr-x  2 www-data  root 4,0K abr 11 16:43 config
-rw-r--r--  1 www-data  root 3,9K abr 11 16:43 console.php
-rw-r--r--  1 www-data  root 34K abr 11 16:43 COPYING
drwxr-xr-x 22 www-data  root 4,0K abr 11 16:43 core
-rw-r--r--  1 www-data  root 6,0K abr 11 16:43 cron.php
drwxr-xr-x  2 www-data  root 4,0K abr 11 16:43 custom_apps
```

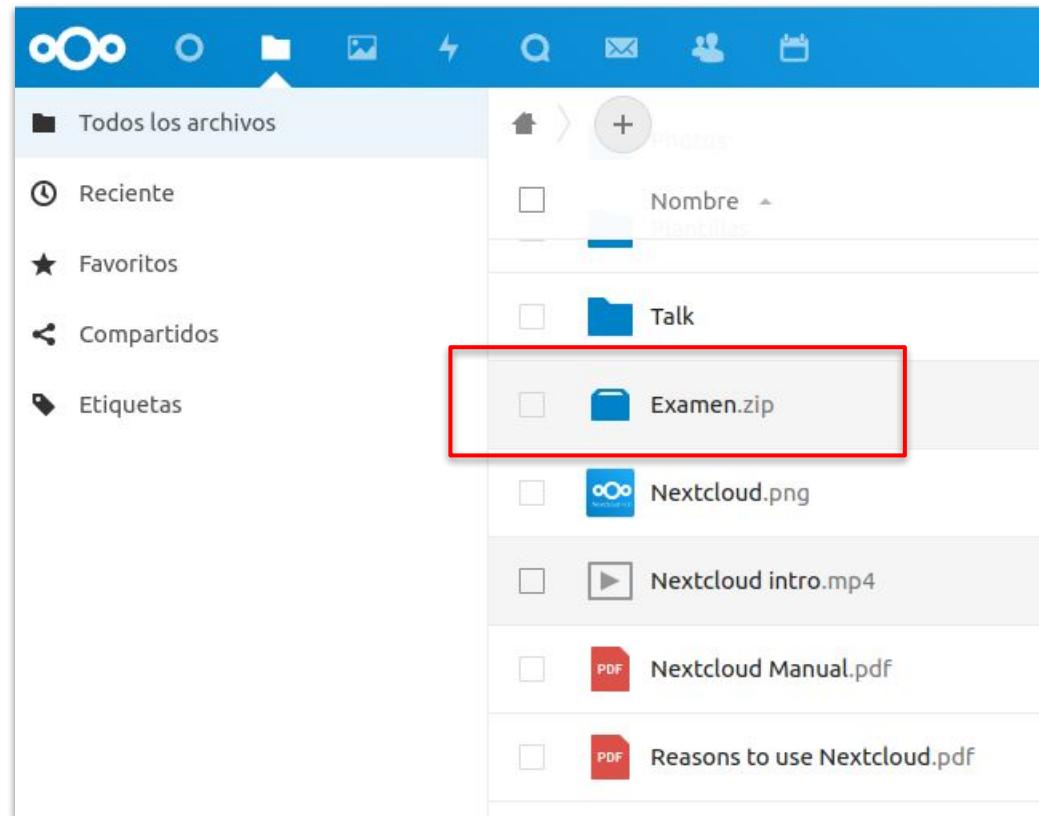
Ejercicio 1 - Solución

1. Vamos a localhost.
2. Asignamos un usuario y contraseña.
3. Seleccionamos SQLite como base de datos.
4. Le damos a instalar.
5. Instalamos las aplicaciones recomendadas.



Ejercicio 1 - Solución

2. Vamos a archivos y subimos uno. En este caso “Examen.zip”



Ejercicio 1 - Solución

3. Comprobamos que podemos acceder desde el host, y está en el directorio “datos”.

```
~/Documentos/docker-examples/06-ejer01
→ sudo ls -l --color datos/data/pepe/files
total 17808
drwxr-xr-x 2 www-data www-data    4096 abr 11 16:53 Documents
→ -rw-r--r-- 1 www-data www-data 512374 feb 24 18:23 Examen.zip
-rw-r--r-- 1 www-data www-data 3963036 abr 11 16:53 'Nextcloud intro.mp4'
-rw-r--r-- 1 www-data www-data 12704899 abr 11 16:53 'Nextcloud Manual.pdf'
-rw-r--r-- 1 www-data www-data   50598 abr 11 16:53 Nextcloud.png
drwxr-xr-x 2 www-data www-data    4096 abr 11 16:53 Photos
drwxr-xr-x 2 www-data www-data    4096 abr 11 16:53 Plantillas
-rw-r--r-- 1 www-data www-data 976625 abr 11 16:53 'Reasons to use Nextcloud.pdf'
drwxr-xr-x 2 www-data www-data    4096 abr 11 16:54 Talk
```

Ejercicio 1 - Solución

4. Paramos y borramos el contenedor.

```
~/Documentos/docker-examples/06-ejer01
→ docker stop nextcloud && docker rm nextcloud
nextcloud
nextcloud
```

5. Lanzamos otro contenedor haciendo bind mount del mismo directorio “datos”.

```
~/Documentos/docker-examples/06-ejer01
→ docker run -d -p 80:80 -v $(pwd)/datos:/var/www/html --name nextcloud2 nextcloud
745d853b598530580a28fadbb133b05762f6b6317d22d8c77646a392b5d0fea8
```

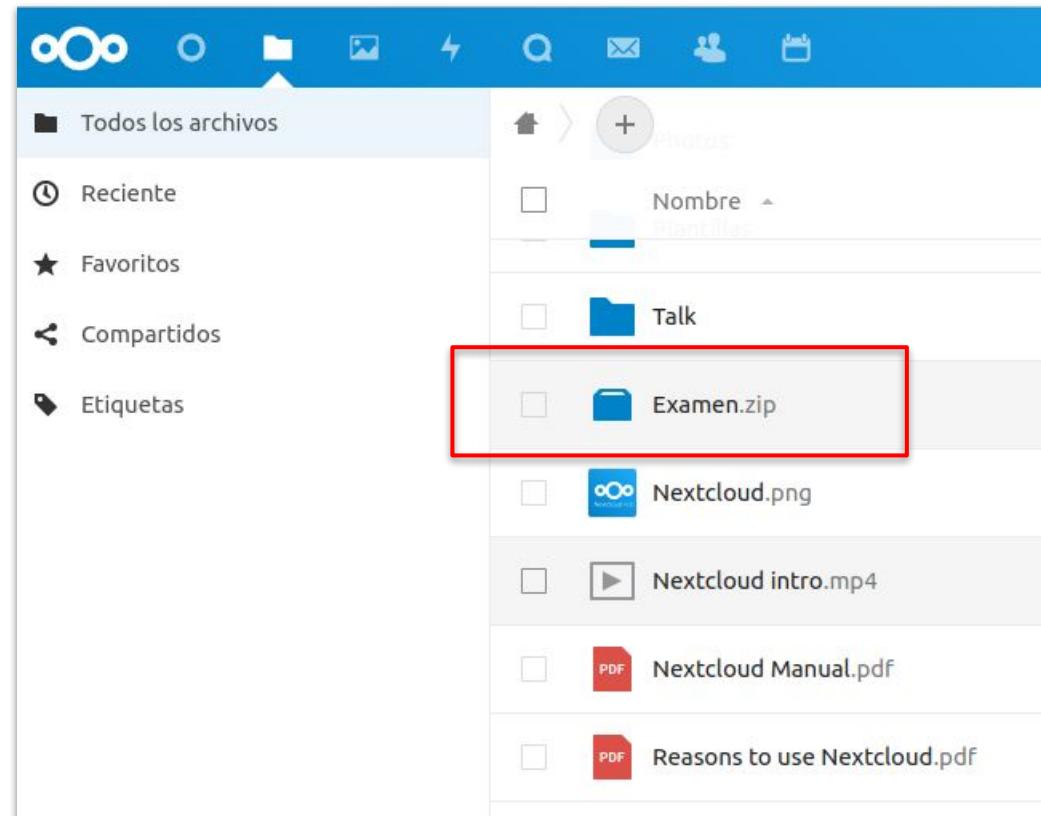
Ejercicio 1 - Solución

Reconoce que hay una instalación y nos pide directamente iniciar sesión.



Ejercicio 1 - Solución

Vamos a archivos
comprobamos que tenemos
nuestro “Examen.zip”



Ejercicio 2

Ejercicio 2

1. Crea un contenedor a partir de la imagen oficial de MySQL usando *bind mount* para persistir las bases de datos en una carpeta llamado “db”.
2. Lanza un contenedor phpmyadmin que se conecte al MySQL y crea una base de datos.
3. Para y elimina los contenedores.
4. Vuelve a hacer los pasos 1 y 2 y comprueba que la base de datos ha persistido.

Ejercicio 2 - Solución

1. Crea un contenedor a partir de la imagen oficial de MySQL usando *bind mount* para persistir las bases de datos en una carpeta llamado “db”.

```
→ docker run -d --name mysql -v $(pwd)/db:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root mysql  
b30f2a92d9f740dbbdf0b11b41785172bdd17cc9d526bf194a7dfc63a36abfce
```

2. Lanza un contenedor phpmyadmin que se conecte al MySQL y crea una base de datos.

```
→ docker run --name dbadmin -d --link mysql:db -p 8080:80 phpmyadmin  
3a9073dfe901dc638ef31a56c5dd4544b34a5d3c472081d187ab88c4a63d26a4
```

Ejercicio 2 - Solución

The screenshot shows the phpMyAdmin login interface. At the top is the logo and the text "Bienvenido a phpMyAdmin". Below it is a "Idioma - Language" dropdown set to "Español - Spanish". A "Iniciar sesión" button is followed by fields for "Usuario:" (root) and "Contraseña:" (****). A "Continuar" button is at the bottom.

Iniciamos sesión y creamos la base de datos
“Examenes”

The screenshot shows the phpMyAdmin dashboard after logging in. It features a navigation bar with icons for home, export, import, and other functions. Below it are tabs for "Reciente" and "Favoritas". The main area displays a tree view of databases. A red arrow points from the "Examenes" node in this tree to the right. The tree includes nodes for "Nueva", "Examenes", "information_schema", "mysql", "performance_schema", and "sys".

Ejercicio 2 - Solución

Podemos verla ya creada también en el sistema de archivos:

```
→ l db
total 194M
drwxr-xr-x 7 systemd-coredump root          4,0K abr 11 17:26 .
drwxrwxr-x 3 pepe      pepe          4,0K abr 11 17:23 ..
-rw-r----- 1 systemd-coredump systemd-coredump 56 abr 11 17:23 auto.cnf
-rw-r----- 1 systemd-coredump systemd-coredump 3,0M abr 11 17:24 binlog.000001
-rw-r----- 1 systemd-coredump systemd-coredump 407 abr 11 17:26 binlog.000002
-rw-r----- 1 systemd-coredump systemd-coredump 32 abr 11 17:24 binlog.index
-rw----- 1 systemd-coredump systemd-coredump 1,7K abr 11 17:24 ca-key.pem
-rw-r--r-- 1 systemd-coredump systemd-coredump 1,1K abr 11 17:24 ca.pem
-rw-r--r-- 1 systemd-coredump systemd-coredump 1,1K abr 11 17:24 client-cert.pem
-rw----- 1 systemd-coredump systemd-coredump 1,7K abr 11 17:24 client-key.pem
drwxr-x--- 2 systemd-coredump systemd-coredump 4,0K abr 11 17:26 Examenes
-rw-r----- 1 systemd-coredump systemd-coredump 192K abr 11 17:26 '#ib_16384_0 dblwr'
-rw-r----- 1 systemd-coredump systemd-coredump 8,2M abr 11 17:23 '#ib_16384_1 dblwr'
```

Ejercicio 2 - Solución

3. Paramos y eliminamos los contenedores.

```
~/Documentos/docker-examples/06-ejer02
→ docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED        STATUS          PORTS          NAMES
3a9073dfe901   phpmyadmin   "/docker-entrypoint..."   6 minutes ago   Up 6 minutes   0.0.0.0:8080->80/tcp, :::8080->80/tcp   dbadmin
b30f2a92d9f7   mysql       "docker-entrypoint.s..."   6 minutes ago   Up 6 minutes   3306/tcp, 33060/tcp   mysql

~/Documentos/docker-examples/06-ejer02
→ docker stop 3a9073dfe901 b30f2a92d9f7 && docker rm 3a9073dfe901 b30f2a92d9f7
3a9073dfe901
b30f2a92d9f7
3a9073dfe901
b30f2a92d9f7

~/Documentos/docker-examples/06-ejer02
→ docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED        STATUS          PORTS          NAMES
```

Ejercicio 2 - Solución

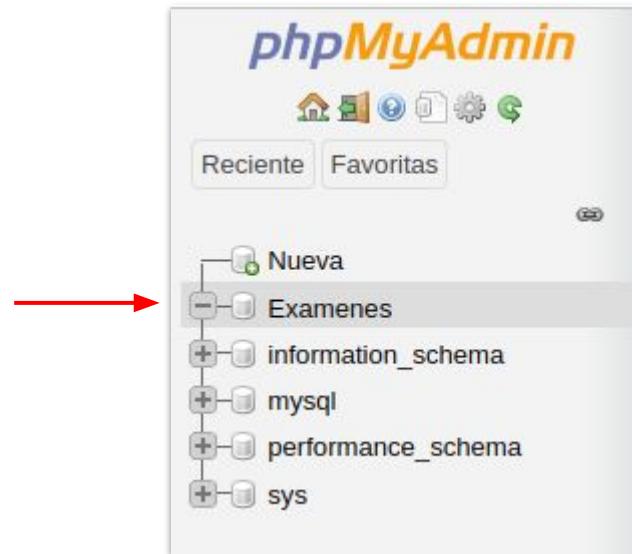
4. Volvemos a crear los contenedores.

```
~/Documentos/docker-examples/06-ejer02
→ docker run -d --name mysql -v $(pwd)/db:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root mysql
12ca70c726fa93a93571ba2d3c4280e8be4fcf3bb9276d0496d57947ec1199a

~/Documentos/docker-examples/06-ejer02
→ docker run --name dbadmin -d --link mysql:db -p 8080:80 phpmyadmin
457be790fdcb01c888a9865bf48a3faf8495f0de2e672fe9a593da66e172b041
```

Ejercicio 2 - Solución

Y comprobamos que la base de datos “Examenes” ha persistido.



Volúmenes



Ejercicio 1

Ejercicio 1

Haz el ejercicio 1 de bind mount usando ahora volúmenes.

Ejercicio 1 - Solución

La única diferencia es el comando para lanzar el contenedor nextcloud:

```
~/Documentos/docker-examples/06-ejer01
→ docker run -d -p 80:80 -v datos:/var/www/html --name nextcloud nextcloud
2ea87f42f551f3853e0f71ef331ef3129c8dd698123b2d610162680838cca2ee
```

```
~/Documentos/docker-examples/06-ejer01
→ docker volume ls
DRIVER      VOLUME NAME
local       datos
```

Ejercicio 2

Ejercicio 2

Haz el ejercicio 2 de bind mount usando ahora volúmenes.

Ejercicio 2 - Solución

La única diferencia es el comando para lanzar el contenedor mysql:

```
→ docker run -d --name mysql -v db:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root mysql  
3e712558fc17fd9a64b4abafc2efd6f6abbfc96945639c8ab4f76df42e3db0ac
```

```
→ docker volume ls  
DRIVER      VOLUME NAME  
local        db
```

Ejercicio 3

Ejercicio 3

A partir del ejercicio anterior, en que la persistencia de la base de datos mysql se está haciendo en un volumen, se pide crear un contenedor que haga un archivo tar.gz con el contenido del volumen en un directorio del host llamado backup.

El contenedor tienen que borrarse después de hacer la copia.

Ejercicio 3 - Solución

```
~/Documentos/docker-examples/06-ejer03
→ docker run --rm -v db:/database -v $(pwd)/backup:/backup bash bash -c "cd database && tar czvf ../backup/db.tar.gz ."
./
./public_key.pem
./ca-key.pem
./ib_buffer_pool
./undo_002
./ib_logfile0
./mysql.ibd
./performance_schema/
./performance_schema/replication_appl_170.sdi
./performance_schema/events_transacti_131.sdi
./performance_schema/events_errors_su_139.sdi
./performance_schema/replication_appl_165.sdi
./performance_schema/users_144.sdi
./performance_schema/table_handles_158.sdi
./performance_schema/session_account__151.sdi
./performance_schema/replication_conn_164.sdi
./performance_schema/file_instances_93.sdi
./performance_schema/setup_objects_105.sdi
./performance_schema/events_stages_su_115.sdi
./performance_schema/keyring_keys_152.sdi
```

Ejercicio 3 - Solución

```
~/Documentos/docker-examples/06-ejer03
→ tree
.
└── backup
    └── db.tar.gz

1 directory, 1 file
```

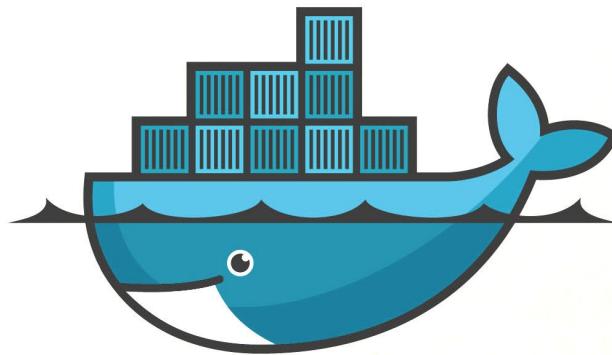
```
~/Documentos/docker-examples/06-ejer03
→ ll backup
total 11M
-rw-r--r-- 1 root root 11M abr 11 20:34 db.tar.gz
```

```
~/Documentos/docker-examples/06-ejer03
→ tar --list -f backup/db.tar.gz
./
./public_key.pem
./ca-key.pem
./ib_buffer_pool
./undo_002
./ib_logfile0
./mysql.ibd
./performance_schema/
./performance_schema/replication_appl_170.sdi
```

El contenedor se ha borrado gracias a la opción **--rm**:

```
~/Documentos/docker-examples/06-ejer03
→ docker ps -a
CONTAINER ID   IMAGE      COMMAND           CREATED        STATUS          PORTS          NAMES
0828bd301e20   phpmyadmin "/docker-entrypoint..."  15 minutes ago  Up 15 minutes  0.0.0.0:8080->80/tcp, :::8080->80/tcp  dbadmin
3e712558fc17   mysql     "docker-entrypoint.s..."  23 minutes ago  Up 23 minutes  3306/tcp, 33060/tcp  mysql
```

Redes



docker

Repaso - Redes

Docker permite crear **redes virtuales**.

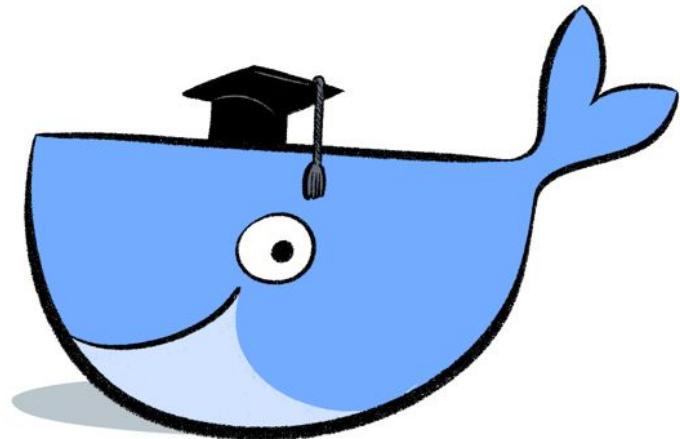
Las redes virtuales sirven para:

- Comunicar contenedores **entre ellos**.
- Comunicar contenedores **con el anfitrión**.
- Comunicar contenedores **con el exterior**.

Contenido

1. Redes por defecto
2. Conectar contenedores a redes
3. Gestión de redes bridge creadas por el usuario

Redes por defecto



Redes por defecto

Docker crea automáticamente unas redes cuando se instala para que los contenedores puedan comunicarse entre sí, con el host y con el exterior.

Estas redes no se pueden eliminar.

En concreto, se crean 3 redes:

NETWORK ID	NAME	DRIVER	SCOPE
8e64d9f74a87	bridge	bridge	local
195de2662bc3	host	host	local
b1eabba11a92	none	null	local

Redes por defecto - Red bridge

Es la red a la que se conectan los contenedores **por defecto**.

Utiliza la interfaz virtual “**docker0**” del anfitrión.

```
→ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 50:65:f3:28:df:84 brd ff:ff:ff:ff:ff:ff
    altname enp0s25
    inet 10.2.2.48/24 brd 10.2.2.255 scope global dynamic noprefixroute eno1
        valid_lft 40403sec preferred_lft 40403sec
    inet6 fe80::728a:9daa:a9:1d34/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:93:55:99:9b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
```

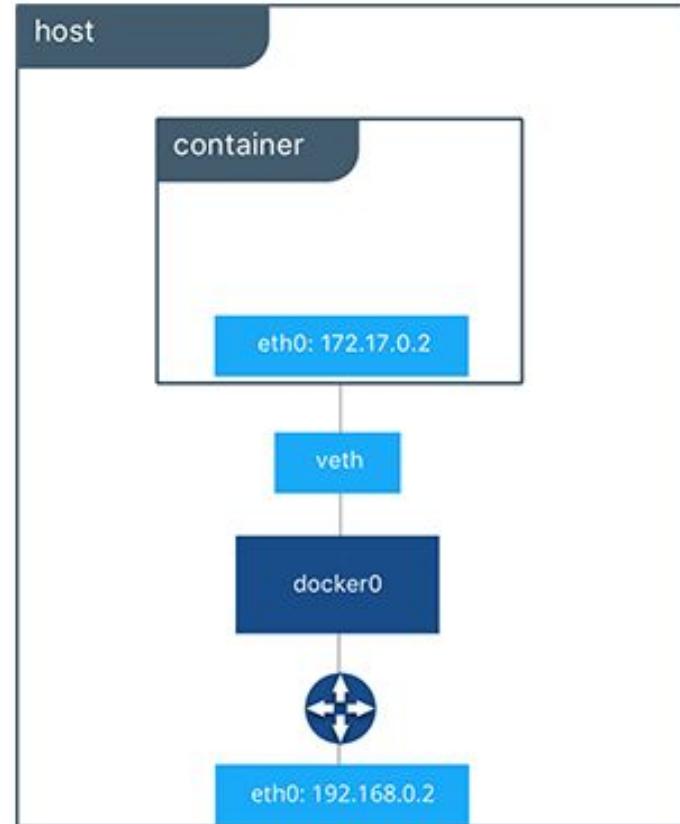
Redes por defecto - Red bridge

El direccionamiento por defecto es:

172.17.0.0/16.

Los contenedores conectados a esta red que quieren exponer puertos al exterior tienen que usar la opción **-p**.

Docker crea las reglas **iptables** necesarios para gestionarla.



Redes por defecto - Red bridge

```
→ docker ps
CONTAINER ID   IMAGE      COMMAND     CREATED      STATUS      PORTS      NAMES
0f6102f1f077   ubuntu     "bash"      8 minutes ago  Up 8 minutes

→ docker inspect -f '{{json .NetworkSettings.Networks}}' wizardly_shirley | json_pp
{
    "bridge" : {
        "Aliases" : null,
        "DriverOpts" : null,
        "EndpointID" : "0034bb7eee7a1f3bb840506e0347c3f8ccd6100545259ee39bddf4de73256cca",
        "Gateway" : "172.17.0.1",
        "GlobalIPv6Address" : "",
        "GlobalIPv6PrefixLen" : 0,
        "IPAMConfig" : null,
        "IPAddress" : "172.17.0.2",
        "IPPrefixLen" : 16,
        "IPv6Gateway" : "",
        "Links" : null,
        "MacAddress" : "02:42:ac:11:00:02",
        "NetworkID" : "1745b4d8250d7fe9cdc5223815441123bd3e64d20a4516ea8502c01eb6637ca4"
    }
}
```

Redes por defecto - Red host

Si un conectanedor se conecta a la red host es como si utilizara directamente la **red del anfitrión**.

El contenedor **no tiene IP** propia, sino que usa la del anfitrión.

Los puertos utilizados son directamente los del anfitrión, no hay que mapearlos.

Es como si los servicios se ejecutaran directamente los del anfitrión.

Redes por defecto - Red host

Por ejemplo lanzamos un nginx conectado a la red host, u no es necesario mapear el puerto 80, sino que es accesible directamente desde el anfitrión.

```
→ docker run -d --network host nginx  
e3dbeb129668dec19ddc91b2928fd5e68f75b4982a7d7d9a063822c7403dc782
```



Redes por defecto - Red host

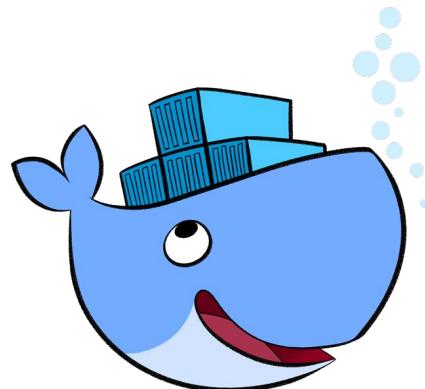
```
→ docker inspect -f '{{json .NetworkSettings.Networks}}' xenodochial_noether | json_pp
{
    "host" : {
        "Aliases" : null,
        "DriverOpts" : null,
        "EndpointID" : "96c0f6bd0f548b137cb010dbfe24e1f492f9d24698e6e3ec7bc8d75c7d14d86c",
        "Gateway" : "",
        "GlobalIPv6Address" : "",
        "GlobalIPv6PrefixLen" : 0,
        "IPAMConfig" : null,
        "IPAddress" : "",
        "IPPrefixLen" : 0,
        "IPv6Gateway" : "",
        "Links" : null,
        "MacAddress" : "",
        "NetworkID" : "195de2662bc330d032c8a93b0f0c20269e2786b065accb2546a5798af1c3f8e7"
    }
}
```

Redes por defecto - Red none

Si un conector se conecta a la red none, no tiene conexión a ninguna red.

La única interfaz de red que tiene es *loopback*.

Coneectar contenedores a redes



Conectar contenedores a redes

1. Al crear el contenedor.
2. Con contenedor ya creado a una red.

Conectar redes al crear un contenedor

docker **run** y docker **create** admiten el parámetro **--network** para especificar la red a la que conectar el contenedor.

Por defecto se conecta a la red “**bridge**”.

Dentro de la misma red definida por el usuario, los contenedores se pueden referenciar por sus **nombres**, ya que Docker tienen un DNS que resuelve la IP.

En el caso de la red “**bridge**”, para que haya resolución de nombres se tiene que usar **--link**.

Conectar redes al crear un contenedor

```
→ docker network create RedPrueba  
d34c748a55fdcd53b62714969ce95530b37cb7b5156854047c2eaa8b81821fc
```

```
→ docker run -d --name webserver --network RedPrueba nginx  
5c89b6ec509f2c90f381227b5b9aeef8a8e40ab8d43758cedd508441557ea68e
```

```
→ docker run -it --network RedPrueba bash  
bash-5.1# ping webserver  
PING webserver (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.185 ms  
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.074 ms  
^C  
--- webserver ping statistics ---  
2 packets transmitted, 2 packets received, 0% packet loss  
round-trip min/avg/max = 0.074/0.129/0.185 ms  
bash-5.1# exit  
exit
```

Conectar redes al crear un contenedor

También se puede asignar un alias al contenedor dentro de la red con la opción **--network-alias**:

```
→ docker run -d --name webserver --network RedPrueba --network-alias nginxserver nginx  
ec2d06c772d08abf8f56e7b0de1745b4685c674a9612c7329bc0d10f11d2df77
```

```
→ docker run -it --network RedPrueba bash  
bash-5.1# ping nginxserver  
PING nginxserver (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.119 ms  
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.078 ms  
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.168 ms  
^C  
--- nginxserver ping statistics ---  
3 packets transmitted, 3 packets received, 0% packet loss  
round-trip min/avg/max = 0.078/0.121/0.168 ms  
bash-5.1# ping webserver  
PING webserver (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.098 ms
```



Conectar redes a un contenedor ya creado.

docker connect/disconnect red contenedor

Algunas opciones interesantes son:

- **--alias**: para especificar un alias del contenedor en la red.
- **--ip**: si se quiere especificar una ip4 fija.
- **--ip6**: si se quiere especificar una ip6 fija.

Conectar redes a un contenedor ya creado.

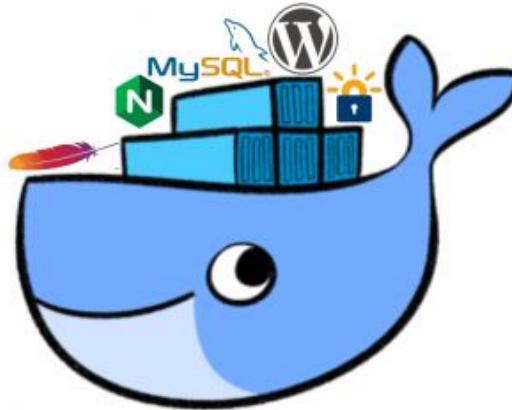
```
→ docker run -d --name webserver nginx  
70c481a16c3fe25cc2759a8de2744e183fb88460b1909f223f1c70eb59d8d1e
```

```
→ docker run --rm -it --network RedPrueba bash  
bash-5.1# ping webserver  
ping: bad address 'webserver'
```

```
→ docker network connect RedPrueba webserver
```

```
→ docker run --rm -it --network RedPrueba bash  
bash-5.1# ping webserver  
PING webserver (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.129 ms
```

Gestión de redes bridge creadas por el usuario



Gestión de redes bridge creadas por el usuario

1. Redes bridge.
2. Red “bridge” por defecto vs redes bridge.
3. Gestión de redes bridge.

Redes bridge

Además de las 3 redes que se crean por defecto, se pueden **crear** más redes de tipo **bridge**.

Su uso es para implementar redes **internas/privadas** entre varios contenedores.

Estas redes bridge creadas **no** son exactamente iguales que la red “brigde” por defecto.

En sistemas **en producción**, se **aconseja** su uso en lugar de la red “bridge” por defecto.

Red “bridge” por defecto vs redes bridge

Red “*bridge*” por defecto

- Se crea automáticamente al instalar Docker.
- No se puede eliminar.
- Se asigna por defecto a los contenedores si no se explicita una red.
- Solo proporciona resolución de nombres si se usa --link, que está depreciado. Además es resolución estática.

Redes *bridge*

- Tienen que crearse de forma explícita.
- Se pueden eliminar.
- No se asignan por defecto a los contenedores. Debe explicitarse.
- Proporcionan resolución DNS.

Red “bridge” por defecto vs redes bridge

Red “*bridge*” por defecto

- No permite conexión en caliente de contenedores. Tienen que apagarse antes.
- Al ser la red por defecto, proporciona menos aislamiento y control.
- Los contenedores comparten ciertas variables de entorno, lo que puede ser problemático.

Redes *bridge*

- Permiten conexión en caliente de contenedores.
- Proporcionan más aislamiento y control.
- Los contenedores no comparten variables de entorno.

Red “bridge” por defecto vs redes bridge

```
→ docker run -d --name webserver -e NGINX_PASS=12345 nginx  
878c4f67c36e4efb2a985a402a952b9766b928aefd8229b426726e990b119cfa
```

Se observa:

- Resolución de nombres estática
- Compartición de variables de entorno

```
→ docker run --rm -it --link webserver bash  
bash-5.1# cat /etc/hosts  
127.0.0.1      localhost  
::1      localhost ip6-localhost ip6-loopback  
fe00::0 ip6-localnet  
ff00::0 ip6-mcastprefix  
ff02::1 ip6-allnodes  
ff02::2 ip6-allrouters  
172.17.0.2      webserver 878c4f67c36e  
172.17.0.3      b595f3d92bc1
```

```
bash-5.1# env  
HOSTNAME=b595f3d92bc1  
WEBSERVER_PORT_80_TCP_ADDR=172.17.0.2  
WEBSERVER_PORT_80_TCP_PORT=80  
WEBSERVER_ENV_NGINX_VERSION=1.21.6  
PWD=/  
WEBSERVER_PORT_80_TCP=tcp://172.17.0.2:80  
_BASH_BASELINE_PATCH=16  
WEBSERVER_PORT=tcp://172.17.0.2:80  
HOME=/root  
_BASH_VERSION=5.1.16  
_BASH_BASELINE=5.1.16  
WEBSERVER_PORT_80_TCP_PROTO=tcp  
_BASH_LATEST_PATCH=16  
TERM=xterm  
SHLVL=1  
WEBSERVER_ENV_NGINX_PASS=12345  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin  
WEBSERVER_ENV_NJS_VERSION=0.7.2  
WEBSERVER_ENV_PKG_RELEASE=1~bullseye  
WEBSERVER_NAME=/upbeat_brown/webserver  
_=~/usr/bin/env
```

Red “bridge” por defecto vs redes bridge

Ambos contenedores **comparten DNS con el host.**

bash

webserver

```
→ docker run --rm -it --link webserver bash  
bash-5.1# cat /etc/resolv.conf  
nameserver 10.2.1.254  
search lliurex
```

```
→ docker exec webserver cat /etc/resolv.conf  
nameserver 10.2.1.254  
search lliurex
```

Red “bridge” por defecto vs redes bridge

host

```
→ resolvectl status
Global
    Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (enp2s0)
    Current Scopes: DNS
        Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.2.1.254
DNS Servers: 10.2.1.254
    DNS Domain: lliurex
```

Red “bridge” por defecto vs redes bridge

```
→ docker run -d --name webserver -e NGINX_PASS=12345 --network RedPrueba nginx  
75e9f71d20d79273e67f1a909566f50e04f8dbe28aed0d04ad7af5c4c11cda28
```

Se observa:

- Resolución de nombres dinámica
- No hay compartición de variables de entorno

```
→ docker run --rm -it --network RedPrueba bash  
bash-5.1# cat /etc/hosts  
127.0.0.1      localhost  
::1      localhost ip6-localhost ip6-loopback  
fe00::0 ip6-localnet  
ff00::0 ip6-mcastprefix  
ff02::1 ip6-allnodes  
ff02::2 ip6-allrouters  
172.18.0.3      948b37a58169  
  
bash-5.1# ping webserver  
PING webserver (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.297 ms
```

```
bash-5.1# env  
HOSTNAME=948b37a58169  
PWD=/  
_BASH_BASELINE_PATCH=16  
HOME=/root  
_BASH_VERSION=5.1.16  
_BASH_BASELINE=5.1.16  
_BASH_LATEST_PATCH=16  
TERM=xterm  
SHLVL=1  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin  
_=usr/bin/env
```

Red “bridge” por defecto vs redes bridge

Ambos contenedores **comparten un DNS** que es **diferente al del host**.

bash

```
bash-5.1# cat /etc/resolv.conf
search lliurex
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
```

webserver

```
→ docker exec webserver cat /etc/resolv.conf
search lliurex
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
```

Este DNS hace forward por defecto al del host, a no ser que se especifique otro con el parámetro **--dns** de docker create o run.

Red “bridge” por defecto vs redes bridge

host

```
→ resolvectl status
Global
    Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (enp2s0)
    Current Scopes: DNS
        Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.2.1.254
DNS Servers: 10.2.1.254
    DNS Domain: lliurex
```

Gestión de redes bridge - Comandos

docker network comando

<code>create <i>nombre</i></code>	Crea una red
<code>ls</code>	Lista todas las redes
<code>rm <i>nombre</i></code>	Elimina una red
<code>prune</code>	Elimina las redes no usados por ningún contenedor
<code>inspect</code>	Proporciona información sobre una red
<code>connect / disconnect</code>	Conecta/Desconecta un contenedor a una red



Gestión de redes bridge - Comandos

docker network create *nombre*

```
→ docker network create RedPrueba  
970c0f93c3dcdb580a9f60c5eba53e0733108e12e2f7c473f9899b085ffc1a36
```

```
→ docker network create RedIntranet  
bc3c6e9170a1b59282830800042f917f246bf2b4e6321e7ac8c58e6ad7a49d09
```



Gestión de redes bridge - Comandos

docker network ls

```
→ docker network ls
NETWORK ID      NAME        DRIVER      SCOPE
bc3c6e9170a1    RedIntranet  bridge      local
970c0f93c3dc    RedPrueba   bridge      local
fbac97951f5c    bridge      bridge      local
195de2662bc3    host        host       local
b1eabba11a92    none        null       local
```



Gestión de redes bridge - Comandos

docker network inspect name

```
→ docker inspect RedIntranet
[
  {
    "Name": "RedIntranet",
    "Id": "bc3c6e9170a1b59282830800042f917f246bf2b4e6321e7ac8c58e6ad7a49d09",
    "Created": "2022-04-13T17:38:09.086288056+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

```
→ docker network inspect RedPrueba
[
  {
    "Name": "RedPrueba",
    "Id": "970c0f93c3dcdb580a9f60c5eba53e0733108e12e2f7c473f9899b085ffc1a36",
    "Created": "2022-04-13T17:37:35.662840335+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```



Gestión de redes bridge - Comandos

```
docker network rm name
```

```
→ docker network rm RedPrueba  
RedPrueba
```

```
→ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
bc3c6e9170a1	RedIntranet	bridge	local
fbac97951f5c	bridge	bridge	local
195de2662bc3	host	host	local
b1eabba11a92	none	null	local



Gestión de redes bridge - Comandos

docker network prune

```
→ docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
RedIntranet
```

```
→ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
fbac97951f5c    bridge    bridge      local
195de2662bc3    host      host       local
b1eabba11a92    none      null       local
```

Ejercicio 1

Ejercicio 1

Cuando hicimos el ejercicio 2 del tema 04 Contenedores, en el que se montaba un Wordpress con MySQL y PhpMyAdmin utilizamos la red “bridge” por defecto y tuvimos que usar la opción --link para que el contenedor de PhpMyAdmin pudiera resolver el nombre del contenedor de la base de datos.

Dado que usar la red “bridge” no está recomendado en producción, y --link está depreciado, se pide hacer el ejercicio usando una red bridge normal para la comunicación entre los contenedores.

Ejercicio 1 - Solución

Creamos la red:

```
→ docker network create RedWordpress  
fc5248d71fe23e35454e104204990411b70c08ad0834ca8579ca38593fe9cd1c
```

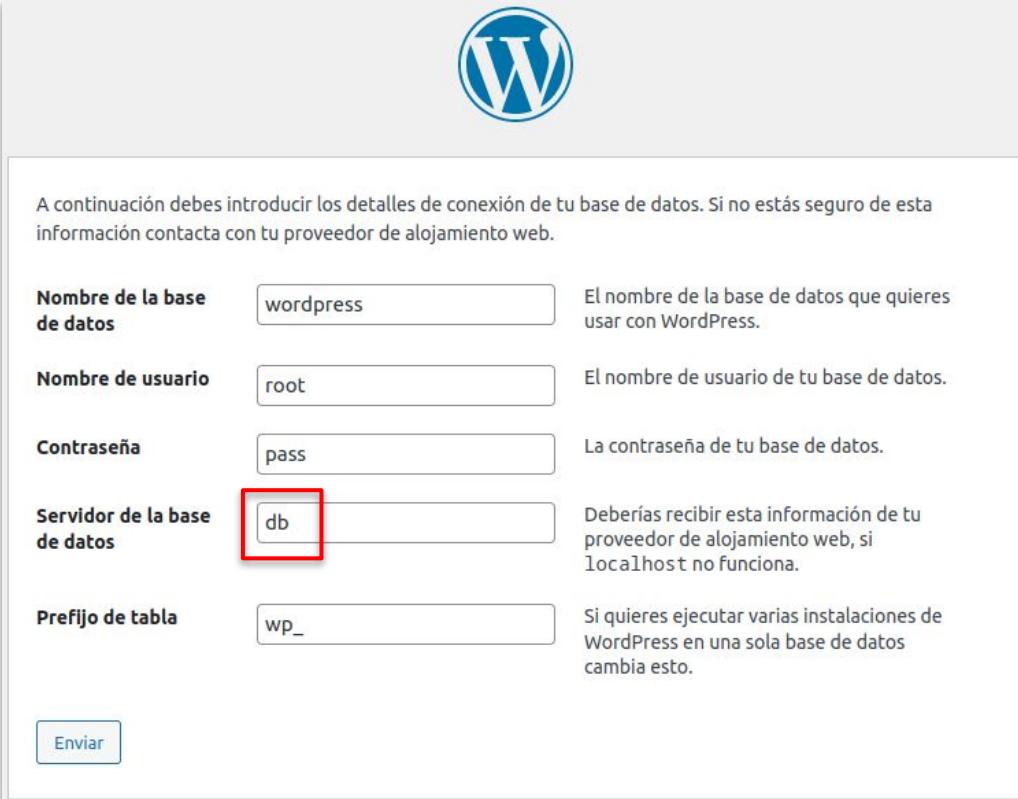
Ejecutamos el contenedor con MySQL con el nombre que espera PhpMyAdmin:

```
→ docker run -d --name db --network RedWordpress -e MYSQL_ROOT_PASSWORD=pass -e MYSQL_DATABASE=wordpress mysql:8  
e4d216eb0bfa93148e7cb074f5b09593ee7b08332d44919b1467f95c2d5afa65
```

Ejecutamos el contenedor con Wordpress:

```
→ docker run -d --name wordpress --network RedWordpress -p 8080:80 wordpress  
f367eab6e511355f667f46aee3da756b8f3cac33446d7ac5cea8837dc799d2fc
```

Ejercicio 1 - Solución



A continuación debes introducir los detalles de conexión de tu base de datos. Si no estás seguro de esta información contacta con tu proveedor de alojamiento web.

Nombre de la base de datos	<input type="text" value="wordpress"/>	El nombre de la base de datos que quieres usar con WordPress.
Nombre de usuario	<input type="text" value="root"/>	El nombre de usuario de tu base de datos.
Contraseña	<input type="text" value="pass"/>	La contraseña de tu base de datos.
Servidor de la base de datos	<input type="text" value="db"/> db	Deberías recibir esta información de tu proveedor de alojamiento web, si localhost no funciona.
Prefijo de tabla	<input type="text" value="wp_"/>	Si quieras ejecutar varias instalaciones de WordPress en una sola base de datos cambia esto.

Ejercicio 1 - Solución

Ejecutamos el contenedor con PhpMyAdmin:

```
→ docker run -d --name dbadmin --network RedWordpress -p 8081:80 phpmyadmin  
a9b2a7b4ebcd16d89db27a1a8f69964254ff49ce27abd821184031dc18903684
```

