

 [Enrere](#) |  [Pàgina principal](#)

# Docker

---

Docker permite automatizar el despliegue de aplicaciones dentro de contenedores.

## Instalar Docker

Debe instalarse Docker en el sistema operativo para comenzar a usarlo. Esto se puede hacer descargando el paquete de instalación apropiado para su sistema operativo desde el sitio web oficial de Docker.

Docker se basa en varios conceptos básicos. Estos incluyen contenedores, imágenes, volúmenes, redes y contenedores de ejecución. Estos conceptos ayudan a entender la forma en que funciona Docker.

## Contenedores

Los **contenedores** no tienen un sistema operativo dentro, el contenedor aísla el espacio de usuario.

Son muy ligeros porque corren como un proceso sobre el SO del host.

Los contenedores escalan en función de la demanda, mientras que las MV tienen que ser aprovisionadas previamente.

## Contenedores Windows y contenedores Linux

Los contenedores Windows corren sobre Windows y los contenedores Linux sobre Linux. Se diferencian en los tipos de aislamiento que tienen ambos.

Los contenedores Linux:

- Se ejecutan sobre el host de modo que son visibles a ellos
- Se ejecutan sobre el kernel como un proceso visible desde el host

En Hyper-V Windows lanza una MV super fina que tiene su propio kernel y por tanto los contenedores no son visibles desde el propio SO.

## Imágenes

una **imagen** de Docker es una instantánea del sistema de archivos y configuración de un contenedor. Estas imágenes se pueden descargar desde el registro de Docker Hub. Un **contenedor** es una instancia en ejecución de una imagen concreta.

Las imágenes se construyen sobre una tecnología de **sistema de ficheros por capas**.

Para crear una imagen, generalmente se crea un archivo de texto llamado docker file formado por diferentes **instrucciones**. Cada línea representa una instrucción, y cada vez que se ejecuta el dockerfile se ejecutan dichas instrucciones de arriba a abajo.

Estos dockerfiles se almacenan como texto y se pueden compartir con facilidad, así como almacenarse en sistemas de control de versiones.

Cada **instrucción** que se ejecuta cambia ligeramente el estado del sistema de archivos respecto a la instrucción anterior.

La diferencia entre el estado del sistema de ficheros antes y después de cada instrucción se guarda en disco como un archivo, que conforma una **capa**.

Cada **imagen** es un conjunto de capas que contienen los diferentes cambios que se van realizando sobre el sistema de archivos empaquetados.

Al final del dockerfile, la última instrucción define el **comando** que se ejecutará cuando arranquemos el contenedor.

Al ejecutar un comando a partir de la imagen creada, se ejecuta el comando especificado y se convierte en el proceso con PID 1 dentro del árbol virtual de procesos.

El contenedor seguirá en marcha mientras el proceso creado siga en ejecución.

## Aprovechamiento de las capas

Cada vez que se ejecuta una instrucción, se crea un contenedor y se etiqueta con un hash creado para obtener un nombre único. De este modo, podemos reutilizar estas capas intermedias y solo tener que construirlas una vez.

Si dos docker files comienzan por el mismo set de instrucciones desde el comienzo, todas las capas que coincidan solo existirán en el sistema de ficheros una vez, para ambos contenedores.

Para cada RUN se crea una capa, por lo que se pueden agrupar varios comandos en un solo RUN y crear una única capa.

## Dockerfiles

Permiten automatizar la construcción de una imagen, a través de un fichero que contiene instrucciones para fabricar una imagen, a través de una serie de pasos.

### Formato de un dockerfile

Formato del Dockerfile, que se creará dentro de la carpeta donde tengamos el proyecto

```
FROM ubuntu:14.04
ENTRYPOINT ["/bin/echo"]
```

### Ejecutar un comando

Si en lugar de utilizar entrypoints queremos pasar parámetros, podemos utilizar CMD

```
CMD ["/bin/echo" , "Hi Docker !"]
```

## Construir una imagen

Construimos la imagen con el siguiente comando. Al haber un Dockerfile en la carpeta la detecta y monta la imagen a partir de las instrucciones del Dockerfile.

```
docker build .
```

## Definir etiqueta para una imagen

Al no definir repositorios ni tags, se asigna a la imagen una ID hexadecimal. Podemos especificar un nombre y una etiqueta al construir la imagen:

```
docker build -t cookbook:hello .
```

## Consultar imágenes

Consultar las imágenes disponibles en nuestra máquina. A partir de una imagen, podemos levantar tantos contenedores como queramos.

```
docker images
```

Nos mostrará:

REPOSITORY SIZE	TAG	IMAGE ID	CREATED
node.js-mongodb 111MB	latest	2ce837b1b0ca	11 minutes ago
danimrtic/apache2 208MB	first	4223faeadc8f	2 months ago
danimrprofe/apache2 208MB	first	4223faeadc8f	2 months ago
danimrprofe/apache2 208MB	latest	4223faeadc8f	2 months ago
ubuntu 87.5MB	latest	20bb25d32758	3 months ago
node 70.3MB	10.13-alpine	93f2dcbcddfe	4 months ago

## Ejecutar un contenedor

A partir de una imagen podemos crear y ejecutar un contenedor, especificando el ID de la imagen:

```
docker run 20bb25d32758
```

## Docker compose

Si quisiera levantar varios contenedores de diferentes imágenes, me vería obligado a definir diferentes Dockerfiles para cada uno y levantar los contenedores uno a uno.

**Docker-compose** es una herramienta que nos permite:

- Definir varios **servicios** en un solo archivo de configuración
- Con un solo comando podemos poner en marcha varios contenedores al mismo tiempo.
- Podemos decir que un servicio solo se levante si otro se ha iniciado previamente

## Definición de servicios

Para ello definimos la configuración de los servicios mediante un archivo en formato YAML, que llamaremos docker-compose.

Ejemplo de docker-compose, en el que podemos ver dos servicios:

- Un servicio **basededatos** que cargará una imagen de **mysql:5.7**
- Un servicio **wordpress** que cargará una imagen **wordpress:latest**

Estas imágenes saldrán de docker hub. Los propios desarrolladores generan las imágenes docker y las publican allí.

## Montaje de volúmenes

Como ya sabemos, los contenedores tienen un ciclo de vida y dentro de ellos los datos no persisten, por lo que tendremos que enlazar una carpeta externa a una interna del contenedor.

Por ejemplo, la carpeta **db\_data** que tenemos en la carpeta contenedora de los dockerfiles y el código fuente se montará dentro del contenedor en la carpeta **/var/lib/mysql**.

De este modo, al morir el contenedor los datos no serán borrados.

## Variables de entorno

Los diferentes servicios pueden tener que compartir variables entre ellos. Por ejemplo, **wordpress** necesitará saber los datos de acceso a la BD donde guardará la información que necesite.

Esto, que generalmente haríamos a mano modificando archivos de configuración, se puede hacer definiendo variables externamente dentro del dockerfile.

## Ejemplo de docker-compose

```
version: '3.3'

services:
  basededatos:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
```

```
restart: always
environment:
  MYSQL_ROOT_PASSWORD: somewordpress
  MYSQL_DATABASE: wordpress
  MYSQL_USER: wordpress
  MYSQL_PASSWORD: wordpress

wordpress:
  depends_on:
    - basededatos
  image: wordpress:latest
  ports:
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: basededatos:3306
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
    WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```

## Ejecutar docker-compose

Para ejecutar docker-compose:

```
docker-compose up
```

El inconveniente de hacerlo así es que la consola se verá bloqueada por la ejecución del proceso hasta que el contenedor finalice.

De todos modos, también podemos ejecutar los contenedores en segundo plano (detached), utilizando:

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

## Visualizar contenedores

Si queremos visualizar los contenedores que se están ejecutando:

```
docker ps
```

## Monitorizar contenedores

- Comprobar recursos que utiliza un contenedor: `docker stats xxx`
- Comprobar los logs de un contenedor: `docker logs`

- Comprobar todos los eventos que han ocurrido a un contenedor: docker events
- Listar procesos de un contenedor: docker top xxx

## Eliminar contenedores

### Eliminar contenedores linux

Actualización: podemos borrar todas las imágenes:

docker system prune -a

Eliminar todos los contenedores existentes (subshell):

```
docker stop $(docker ps -q)
docker rm -v $(docker ps -aq)
```

### Eliminar contenedores Windows

Borrar todos los contenedores en windows. Meter en un bat

```
FOR /f "tokens=*" %i IN ('docker ps -a -q') DO docker rm %i
```

### Eliminar contenedores y imágenes con batch

Un bat que te lo hace todo (maravilloso), borra contenedores e imágenes.

```
@echo off
FOR /f "tokens=*" %%i IN ('docker ps -aq') DO docker rm %%i
FOR /f "tokens=*" %%i IN ('docker images --format "{{.ID}}"') DO docker rmi %%i
```

### Eliminar contenedores en powershell:

```
docker ps -aq | foreach {docker rm $_}
```

## Persistencia

Los contenedores por defecto están completamente aislados del mundo exterior.

Podemos montar un volumen que funcionará como si fuer una carpeta compartida a la que puedes acceder desde otro lugar.

## Dive

Existe una herramienta llamada dive que permite monitorizar una imagen para ver detalles, espacio ocupado, sistema de archivos, etc. e intentar optimizarla.

No he tenido tiempo de probarlo, pero el proyecto está en:

<https://github.com/yosifkit/dive>

## Multi stage builds

Definimos un **dockerfile** y ponemos en primer lugar una línea FROM. A continuación ponemos otra línea FROM. Por ejemplo:

- FROM alpine golang
- FROM alpine

Puedes cargar el código fuente en una imagen y compilarlo. Luego borrar el código fuente. Etiquetamos el estado y utilizarlo como un nuevo statement. Todo lo que hay antes del segundo FROM se elimina.

## Buenas prácticas para construir contenedores

- Una sola aplicación por contenedor. Por ejemplo, PHP y Mysql en dos contenedores.
- Agregar el código fuente de la aplicación lo más tarde posible. Las capas y dependencias se pueden cachear y acelerar las builds posteriores.
- Eliminar todo lo que no sea necesario, o utilizar una imagen scratch o distroless.
- Hacer las imágenes lo más pequeñas posibles. Reduce downtimes, tiempo de arranque y espacio en disco.
- Etiquetar las imágenes
- Utiliza volúmenes para manejar guardar la configuración y los datos fuera de los contenedores

## Ventajas de dockerizar

Algunas ventajas de cambiar a un entorno docker son:

- Ciclos de desarrollo más rápidos
- Aplicaciones más pequeñas
- Ayuda a tener más documentados los pasos de aprovisionamiento del servidor, al tenerlos que definir como instrucciones en un Dockerfile.
- Al hacer los dockerfiles tratas tu infraestructura como código que puede ser committed y compartido, así como ver un histórico de cambios.

## Almacenar en repositorio

Podemos pushear la imagen a un registro del tipo **Docker hub**, para poderla utilizar y construir y escalar contenedores a partir de ella.

## Construir la imagen

Primero creo la imagen, si no la tenía ya creada:

```
docker build -t danimrtic/nodejs-mongodb .
```

Comprobamos que me la ha creado:

```
C:\xxx>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			
danimrtic/nodejs-mongodb	latest	1d3b7e77a948	32 seconds ago
117MB			

Ahora podríamos crear todos los contenedores que queramos a partir de ella, o utilizarla como punto de partida para crear otras imágenes.

## Login en docker hub

Ahora vamos a subir la imagen al repositorio. Para ello nos logueamos:

```
C:\xxx>docker login -u danimrtic
Password:
Login Succeeded
```

Una vez esto, me guarda la configuración en un archivo [JSON](#).

## Subir imagen

Pusheo la imagen:

```
docker push danimrtic/nodejs-mongodb
```

Y ale, a subir se ha dicho:

```
C:\xxx> docker push danimrtic/nodejs-mongodb
The push refers to repository [docker.io/danimrtic/nodejs-mongodb]
c9b7262cf2a9: Pushed
484f9423b541: Pushed
6faf95ed340e: Pushed
a4c2fd79fb93: Pushed
2faeaaebb113: Mounted from library/node
387bc77dd3f2: Mounted from library/node
df64d3292fd6: Mounted from library/node
latest: digest:
sha256:c0540a9bcce05b35ef3b0738aee2681b78246b91503d0ae25091b52644c4df49 size: 1788
```

Al no darle etiqueta, se convierte en la versión [latest](#). Ahora la puedo pullear desde cualquier proveedor cloud y montar los contenedores que quiera.



Borro todas las imágenes para obligar la descarga:

```
docker system prune -a
```

## Descargar imagen

Y a descargar la imagen:

```
docker pull danimrtic/nodejs-mongodb
```

Respuesta:

```
C:\Users\dnick\Desktop\github\node.js-mongodb>docker pull danimrtic/nodejs-mongodb
Using default tag: latest
latest: Pulling from danimrtic/nodejs-mongodb
4fe2ade4980c: Pull complete
c245f6a8ecc5: Pull complete
82bdc9503d50: Pull complete
f4a77d5c469c: Pull complete
628735677efb: Pull complete
69feec983dbb: Pull complete
7a0d47fa11c7: Pull complete
Digest: sha256:c0540a9bcce05b35ef3b0738aee2681b78246b91503d0ae25091b52644c4df49
Status: Downloaded newer image for danimrtic/nodejs-mongodb:latest
```