

# Contenedores

Antonio Espín Herranz

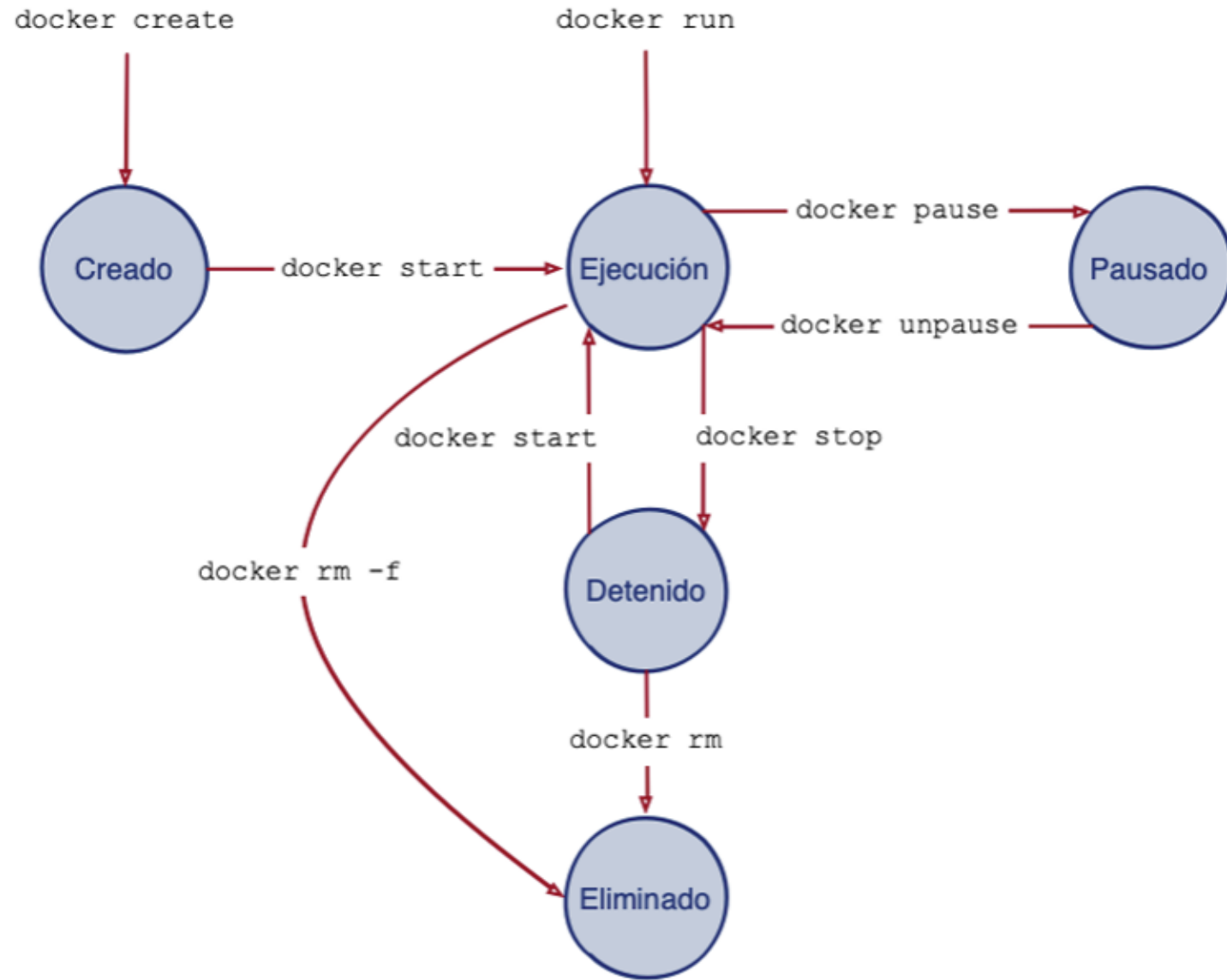
# Contenidos

- Introducción
- Ciclo de vida de un contenedor
- Comando docker
- Crear contenedores
- Trabajar con contenedores
- Registro Docker Hub
- Contenedores en Background / Interactivos
- Pasar variables de entorno a los contenedores
- Mapeo de puertos
- Estadísticas de uso
- VS Code

# Introducción

- **Definición:** Un contenedor es una instancia ligera, portátil y aislada de un proceso o conjunto de procesos que se ejecutan en un sistema operativo compartido.
- Los contenedores permiten empaquetar una aplicación con todas sus dependencias en un solo paquete, lo que facilita su despliegue y ejecución en cualquier entorno que tenga Docker instalado.
- **Características:**
  - **Ligereza:** A diferencia de las máquinas virtuales, los contenedores comparten el mismo núcleo del sistema operativo, lo que los hace mucho más ligeros en términos de uso de recursos.
  - **Portabilidad:** Un contenedor creado en un sistema puede ser ejecutado en otro sistema con Docker sin necesidad de ajustes adicionales.
  - **Aislamiento:** Aunque los contenedores comparten el núcleo del sistema operativo, están aislados unos de otros y del sistema host, lo que significa que los procesos dentro de un contenedor no afectan a otros contenedores o al sistema host.

# Ciclo de Vida



# Comando docker

- Es el cliente en modo línea
- **\$ docker**
- Sin opciones muestra la ayuda
  - Comandos más habituales
  - Comandos sobre la gestión del entorno
  - Otros comandos

# Comandos

- **docker version**
  - Mostrar la versión que tenemos instalada de Docker.
- **docker --help**
  - Mostrar la ayuda del comando.
- **docker comando --help**
  - Para preguntar por la ayuda de un **comando** concreto.
  - **docker version --help**

# Crear contenedores

- **docker create contenedor**
  - Crea el contenedor, pero no lo lanza
- Normalmente vamos a utilizar:
- **docker run contenedor**
  - Para crear y lanzar el contenedor
  - Tendremos posibilidades de crearlo interactivo y en background

# Crear contenedores

- Un contenedor se crea a partir de una imagen.
- La imagen(plantilla) se puede descargar de Docker Hub.
  - Docker Hub repositorio de imágenes
- **El contenedor (versión ejecutable) de la imagen.**
- El contenedor **se crea** a partir de:
- **docker run**
  - Si la imagen no la tiene en local la descarga
  - Se pueden consultar las opciones del comando con:
    - `docker run --help`



# docker run hello-world

- Para probar que todo funciona ok, disponemos de esta imagen para hacer una prueba.
- Primero la busca en local y si no existe la descarga de: library/hello-world → De Docker Hub

```
D:\docker>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:e0b569a5163a5e6be84e210a2587e7d447e08f87a0e90798363fa44a0464a1e8
Status: Downloaded newer image for hello-world:latest
```

La imagen lo único que hace es lanzar un mensaje ... y poco más, ¡es de prueba!

# Algunos comandos

- Desde una consola de Windows.
- Con el comando Docker: ver imágenes y contenedores locales.
  - **docker images**: las imágenes locales.
  - **docker ps**: los contenedores arrancados (locales)
  - **docker ps -a**: los contenedores parados (locales)

```
C:\Users\Anton>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	74cc54e27dc4	4 weeks ago	10.1kB

```
C:\Users\Anton>docker ps
```

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

```
C:\Users\Anton>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
90fbc306c5a1	hello-world	"/hello"	21 hours ago	Exited (0) 21 hours ago		wonderful_austin

# docker ps -a

- La información que muestra:
  - El id del contenedor
  - hello-world: Nombre de la imagen
  - /hello: el comando que se ha ejecutado al crear el contenedor
  - Created: Cuando se ha creado
  - Status: Cuando hemos salido del contenedor. El (0) hemos salido sin errores.
  - Ports: Puertos
  - Names: Nombre del contenedor, se lo asigna docker

# Docker Hub

- **Repositorio centralizado en internet** donde hay miles de imágenes de contenedores que podemos utilizarlas para crear nuestros contenedores. Igual que hemos hecho con hello-world.
  - <https://hub.docker.com/>
- **Necesitaremos** tener una **cuenta** en **Docker**.
- En la opción: **Explore** podemos establecer filtros por S.O, arquitectura, tecnología, etc.
- Algunas están creadas por empresas, otras por desarrolladores, y también podremos subir nuestras propias imágenes.

# Docker Hub

- Uno de los filtros que se pueden establecer es para buscar contenido de confianza, por ejemplo:
  - **Imágenes oficiales**
  - Editor Verificado (formalmente verificadas → son seguras)
  - O por sponsors.
- Mejor **utilizar imágenes** que están **verificadas** o son **oficiales**.
- Podemos tener los mismos problemas que con el Software libre.
- Los filtros son acumulativos.
- Además, en la barra de búsqueda podemos buscar una imagen.

# Docker Hub

- Es importante el nombre de la imagen. Se indica con:
  - Nombre de usuario/nombre de la imagen
  - Permite tener la misma imagen, pero de distintos usuarios.
  - Por ejemplo, una imagen de Tomcat y con distintos usuarios.

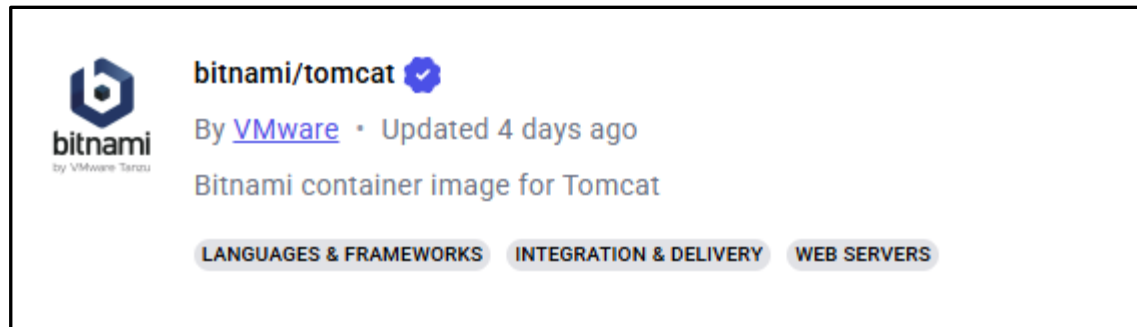


Imagen de Tomcat verificada

- Si nos creamos una cuenta en Docker Hub vamos a poder subir nuestras propias imágenes.

# Docker Hub

- El tipo de cuenta puede ser personal así no tenemos que pagar nada.
- Si es para una empresa tendremos que elegir un tipo de suscripción.
- Dentro de Docker Hub podremos tener nuestros propios repositorios con nuestras imágenes.
- El tema de las suscripciones y precios lo tenemos en la página de Docker: Con la cuenta personal es suficiente para el curso.
  - <https://www.docker.com/pricing/>
- Sin la cuenta en Docker nos permite descargar imágenes, pero con limitaciones, no podemos crear repositorios en Docker Hub y no podemos subir nuestras propias imágenes

## Docker Personal

# \$0

/mo

For individual developers who need the essential tools to build and deploy containers.

### Includes:

- ✓ Docker Desktop
- ✓ Docker Engine + Kubernetes
- ✓ Docker Hub
- ✓ Docker Scout

### Included usage:

- ✓ 1 user
- ✓ 1 Docker Scout-enabled repo\*
- ✓ 100 Docker Hub pulls/hr\*
- ✓ 1 private Docker Hub repo
- ✓ Docker Build Cloud and Testcontainers Cloud free trial

# Docker Hub

- Las imágenes que tenemos dentro de Docker Hub no tienen porque ser todas gratuitas.
- Podemos encontrarnos imágenes que haya que comprar algún tipo de licencia, eso lo vemos al entrar dentro de la imagen nos ofrece una documentación con los detalles de la imagen.
- Dentro de la imagen selecciona nos muestra el comando:
  - **docker pull ubuntu** (o la imagen seleccionada)
  - Cuando hacemos docker run ubuntu (por debajo hace la operación pull)



# Descargar imágenes

- Se puede hacer en dos pasos, primero descargar la imagen:
  - **docker pull ubuntu**

```
C:\Users\Anton>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
5a7813e071bf: Pull complete
Digest: sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Si la imagen no la tenía ya descargada, la descarga

# Descargar imágenes

- Las imágenes que tenemos descargadas se pueden consultar con:
  - **docker images**

```
C:\Users\Anton>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	a04dc4851cbc	4 weeks ago	78.1MB
hello-world	latest	74cc54e27dc4	4 weeks ago	10.1kB

# Crear el contenedor con la imagen

Una vez tenemos descargada la imagen, podemos crear un contenedor lanzando el comando:  
**docker run ubuntu**

```
C:\Users\Anton>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
43cb2afd545f	ubuntu	"/bin/bash"	33 seconds ago	Exited (0) 30 seconds ago		sweet_galileo
90fbc306c5a1	hello-world	"/hello"	23 hours ago	Exited (0) 23 hours ago		wonderful_austin

- Los contenedores basados en S.O. se suelen utilizar como base para otras imágenes, pero no para ejecutarlos de manera individual.

# Crear el contenedor con la imagen

- A partir de la misma imagen podemos crear múltiples contenedores

```
C:\Users\Anton>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3a160987a90d	fedora	"/bin/bash"	7 seconds ago	Exited (0) 5 seconds ago		kind_austin
060589bd8250	fedora	"/bin/bash"	12 seconds ago	Exited (0) 9 seconds ago		brave_mclaren
302a5624cde5	fedora	"/bin/bash"	5 minutes ago	Exited (0) 5 minutes ago		dreamy_diffie
43cb2afd545f	ubuntu	"/bin/bash"	12 minutes ago	Exited (0) 12 minutes ago		sweet_galileo
90fbc306c5a1	hello-world	"/hello"	23 hours ago	Exited (0) 23 hours ago		wonderful_austin

# docker ps

- Con la opción **-l** vemos el último contenedor que se ha creado:

```
C:\Users\Anton>docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3a160987a90d	fedora	"/bin/bash"	15 minutes ago	Exited (0) 15 minutes ago		kind_austin

- **docker ps -n número** (Con el número, indicamos los n últimos contenedores).

```
C:\Users\Anton>docker ps -n 3
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3a160987a90d	fedora	"/bin/bash"	18 minutes ago	Exited (0) 18 minutes ago		kind_austin
060589bd8250	fedora	"/bin/bash"	18 minutes ago	Exited (0) 18 minutes ago		brave_mclaren
302a5624cde5	fedora	"/bin/bash"	23 minutes ago	Exited (0) 23 minutes ago		dreamy_diffie

# docker ps

- **docker ps -a -q** (ojo, si no podemos la opción -a sólo saldrán los que están en ejecución). Para mostrar los **IDs** de los contenedores.

```
C:\Users\Anton>docker ps -q -a
3a160987a90d
060589bd8250
302a5624cde5
43cb2afd545f
90fbc306c5a1
```

- **docker ps -a -s** (se puede añadir -n num para limitar el número de resultados.
  - Para mostrar el tamaño de cada contenedor.
- **docker ps -a -f "name=youthful\_hypatia"**
- <https://docs.docker.com/reference/cli/docker/container/ls/#filter>

# docker ps

- Tenemos algo similar con las imágenes (que veremos en el apartado de imágenes) → **-f -a -q**.
  - **Filtrar, listar, etc.**

# Contenedores en Docker Desktop

## Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage   
*No containers are running.*

Container memory usage   
*No containers are running.*

☐ Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	CPU (%)	Last started	Actions
<input type="checkbox"/>	<input type="radio"/> wonderful_austin	90fbc306c5a1	<a href="#">hello-world</a>	N/A	23 hours ago	<input type="play"/> <input type="vertical"/> <input type="trash"/>
<input type="checkbox"/>	<input type="radio"/> sweet_galileo	43cb2afd545f	<a href="#">ubuntu</a>	N/A	53 minutes ago	<input type="play"/> <input type="vertical"/> <input type="trash"/>
<input type="checkbox"/>	<input type="radio"/> dreamy_diffie	302a5624cde5	<a href="#">fedora</a>	N/A	46 minutes ago	<input type="play"/> <input type="vertical"/> <input type="trash"/>
<input type="checkbox"/>	<input type="radio"/> brave_mclaren	060589bd8250	<a href="#">fedora</a>	N/A	41 minutes ago	<input type="play"/> <input type="vertical"/> <input type="trash"/>
<input type="checkbox"/>	<input type="radio"/> kind_austin	3a160987a90d	<a href="#">fedora</a>	N/A	41 minutes ago	<input type="play"/> <input type="vertical"/> <input type="trash"/>

**Tenemos opciones para arrancar, parar, borrar, etc.**



# Imágenes en Docker Desktop

## Images [Give feedback](#)

View and manage your local and Docker Hub images. [Learn more](#)

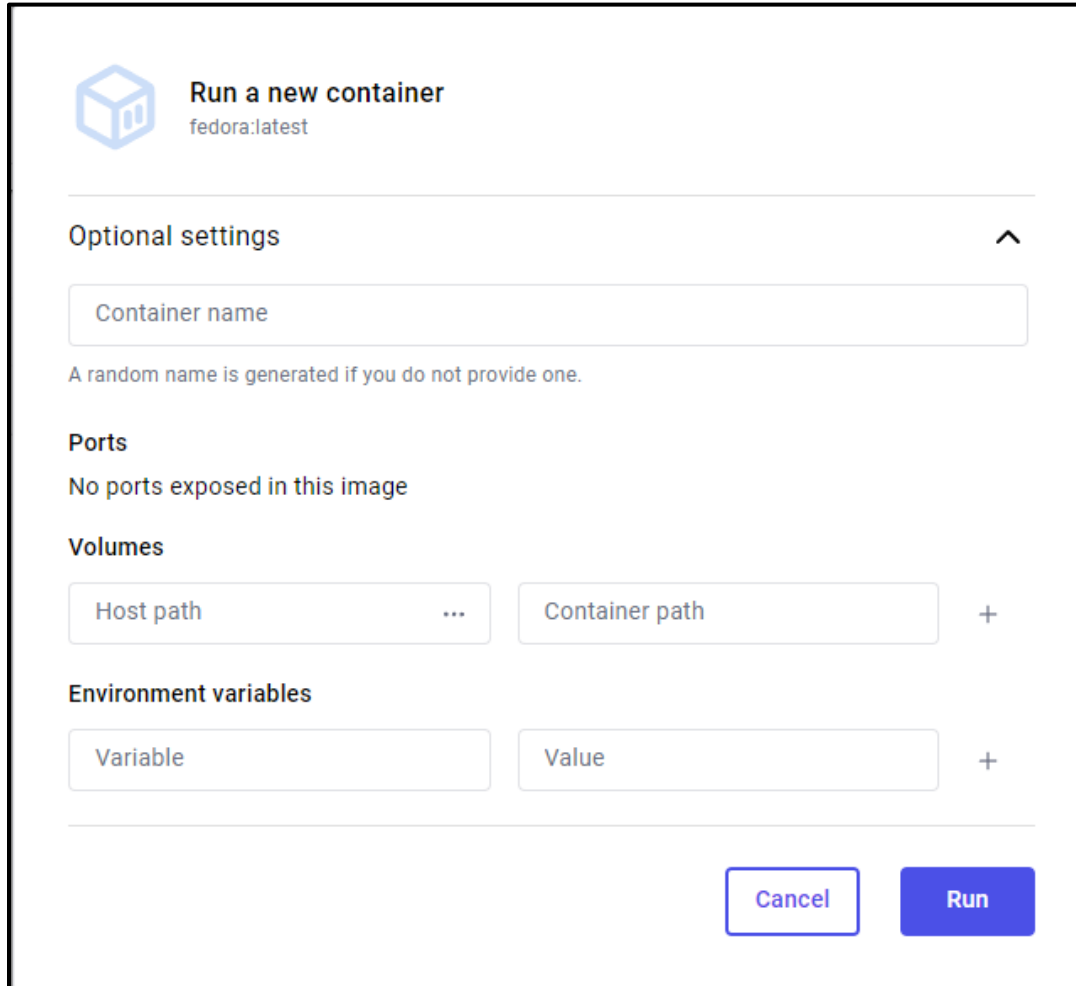
**Local** Hub repositories

236.29 MB / 0 Bytes in use 3 images

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	hello-world	latest	74cc54e27dc4	1 month ago	10.07 KB	
<input type="checkbox"/>	ubuntu	latest	a04dc4851cbc	29 days ago	78.13 MB	
<input type="checkbox"/>	fedora	latest	aa6787b90fe6	4 months ago	158.15 MB	

- Operaciones y vista muy similares a los contenedores.
- Podemos ver la información de la imagen, y crear los contenedores desde aquí.
- ***Entrar en una imagen y crear el nuevo contenedor → con el botón RUN***

# Crear contenedor desde Docker Desktop



The screenshot shows the 'Run a new container' dialog in Docker Desktop. At the top left is the Docker logo and the text 'Run a new container' with 'fedora:latest' below it. Below this is a section titled 'Optional settings' with an upward arrow. It contains a 'Container name' input field and a note: 'A random name is generated if you do not provide one.' Below that is a 'Ports' section stating 'No ports exposed in this image'. Next is a 'Volumes' section with two input fields: 'Host path' (with a dropdown arrow) and 'Container path', followed by a plus sign. Below that is an 'Environment variables' section with two input fields: 'Variable' and 'Value', followed by a plus sign. At the bottom right are two buttons: 'Cancel' and 'Run'.

Run a new container  
fedora:latest

Optional settings ^

Container name

A random name is generated if you do not provide one.

Ports  
No ports exposed in this image

Volumes

Host path ... Container path +

Environment variables

Variable Value +

Cancel Run

- Se pueden indicar varios parámetros:

# Nombres de los contenedores

- Mantiene un diccionario de nombres y genera de manera aleatoria unos nombres y se los asigna aleatoriamente a los contenedores.
- Es mejor asignar nuestros propios nombres, el id es complicado para trabajar.
- Podemos crear un contenedor asignando un nombre:
  - `docker run --name ubuntu1 ubuntu`
  - `docker run --name nombre_contenedor nombre_imagen`

# Contenedores interactivos

- Los contenedores creados si son de un S.O. o no tienen ninguna tarea que hacer directamente después de crearlos se paran.
- Evidentemente tendremos que tener el contenedor funcionando, por ejemplo, si en este tenemos un apache o un mysql.
  - Voy a necesitar que me de un servicio.
- Hay dos formas de mantener un contenedor en funcionamiento
  - Modo interactivo
  - Modo background (lo más habitual)

# Contenedores interactivos

- **docker run --name ubuntu3 -it ubuntu**
- **-it** modo interactivo

```
C:\Users\Anton>docker run --name ubuntu3 -it ubuntu
root@08620028f190:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@08620028f190:/# _
```

- Cuando arrancamos en modo interactivo, en el prompt del S.O. aparece el usuario y id del contenedor. Si lanzamos el comando **docker ps** desde otro terminal podemos ver el contenedor que está arrancado:

```
C:\Users\Anton>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS   NAMES
08620028f190   ubuntu   "/bin/bash"             14 seconds ago Up 11 seconds      ubuntu3
```

Con la opción **docker ps -a** en el status del contenedor, vemos STATUS (**up**)

# Contenedores interactivos

- Los comandos que lanzamos dentro del contenedor ya se ejecutan dentro del contenedor.
- Esta forma de arrancar el contenedor no es muy útil. Es mejor utilizarlo en modo background.
- Si tecleamos el comando **exit** dentro del contenedor.
  - El contenedor se para y volvemos al S.O. de la máquina.
- Ya no lo vemos como arrancado al lanzar el comando: `docker ps`
  - Tenemos que lanzar un comando: **docker ps -a**

# Parar y arrancar contenedores

- Para parar un contenedor arrancado ponemos:
  - `docker stop` (nombre o id del contenedor)
  - **`docker stop ubuntu4`**
    - Se lanza desde otro terminal distinto de donde tenemos el contenedor lanzado.
- Para arrancar un contenedor: `docker start nombre_contenedor`
- **`docker start ubuntu4`**
  - Se pone en marcha y si lanzamos el comando `docker ps` veremos que está arrancado.
  - También para pararlo se puede utilizar el id o simplemente las 4 primeras letras del id.

# Parar y arrancar contenedores

```
C:\Users\Anton>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS      NAMES
ef12b0a5325c   ubuntu   "/bin/bash"             7 minutes ago Up 3 minutes          ubuntu4

C:\Users\Anton>docker stop ef12
ef12

C:\Users\Anton>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS      NAMES
```

- Le paramos utilizando los 4 primeros caracteres del id.
- Para volver al modo interactivo:
  - **docker start -i ubuntu4**
  - Si lo estamos creando si que ponemos **-it**, pero si ya está creado ponemos: **-i**



# Contenedores en background

- Trabajar con un contenedor en modo **background**.
- Tenemos que utilizar la opción **-d** (detach)
- **docker run -d nginx**
  - Si no está la imagen la descargará de Docker hub.
  - Al lanzarlo se quedará arrancado, si consultamos con: **docker ps**, lo veremos arrancado.
  - Con **docker ps -a** veremos el **status** → **up**

# Contenedores en background

- Con la **opción -d no es suficiente** para que un contenedor se quede funcionando en modo background.
- Este ejemplo:
  - **docker run --name fedora2 -d fedora**
  - No deja el contenedor funcionando en modo background
  - Pero al ejecutar **docker ps -a no está ejecutándose (se ha parado).**
  - **Hay que preparar las imágenes para que puedan funcionar en modo background.**
    - Por ejemplo, la imagen de Fedora es un ejemplo de una imagen que no está preparada para trabajar en modo background.

# Contenedores en background

- Al arrancar o crear el contenedor necesitamos dos cosas para poder poner el contenedor en modo background:
  - La opción **-d** (tiene que estar en el comando)
  - Y además **la imagen tiene que estar preparada** para trabajar en modo background.
- Una opción es **dejarlos en background** en la combinación de las opciones de **interactivo** y **background**
  - **docker run --name fedora4 -d -it fedora**

```
C:\Users\Anton>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
09c6dc43d684	fedora	"/bin/bash"	About a minute ago	Up About a minute		fedora3
4c330cccecef	nginx	"/docker-entrypoint..."	19 minutes ago	Up 19 minutes	80/tcp	magical_satoshi

En Docker hub podemos comprobar en la documentación de las imágenes si podremos lanzar el contenedor en modo background

# Borrar contenedores interactivos

- Cuando creamos un contenedor interactivo tenemos la opción de borrarlo cuando se pare.
- Con esto nos evitamos borrarlo de una forma manual.
- Tenemos la opción **--rm**
- **docker run -it --rm ubuntu**
- Este comando nos abre una Shell en el contenedor:
- `root@81d3....:/# exit`
- **De la Shell salimos con exit y al salir se elimina automáticamente.**

# Contenedores en Background

## Arrancar y Parar

- Los contenedores que se encuentran en modo background si se paran y se vuelve a lanzar continúan en modo background.
- docker **stop** id-contenedor
- docker **start** id-contenedor
- Comprobar con **docker ps**

# Etiquetas de las imágenes

- Comprobar las imágenes: **docker images**
- El nombre de las imágenes se indica en la primera columna.
- Y la segunda es la etiqueta.

```
C:\Users\Anton>docker images
REPOSITORY          TAG                 IMAGE ID
nginx                latest             b52e0b094bc0
ubuntu              latest             a04dc4851cbc
httpd                latest             0de612e99135
hello-world         latest             74cc54e27dc4
fedora               latest             aa6787b90fe6
```

# Etiquetas de las imágenes

- Desde Docker hub: <https://hub.docker.com/>
- Podemos entrar en las imágenes (**httpd**, por ejemplo) y comprobar las etiquetas (**tags**) de las que se dispone.
- Las etiquetas de la misma línea son equivalentes (el mismo producto puede ser conocido por nombres distintos).
  - Por ejemplo, las etiquetas de segunda fila están pensadas para un Linux ligero como **alpine** (S.O.)
  - La de la primera fila están basadas en **Debian**.

## Supported tags and respective Dockerfile links

- [2.4.63](#) [2.4](#) [2](#) [latest](#) [2.4.63-bookworm](#) [2.4-bookworm](#) [2-bookworm](#) [bookworm](#) [↗](#)
- [2.4.63-alpine](#) [2.4-alpine](#) [2-alpine](#) [alpine](#) [2.4.63-alpine3.21](#) [2.4-alpine3.21](#) [2-alpine3.21](#) [alpine3.21](#) [↗](#)

# Etiquetas de las imágenes

- Si no indicamos la etiqueta, por defecto, Docker busca la etiqueta **latest**.
  - Esta versión suele ser la más estable, aunque puede haber otras versiones más modernas pero que no sean tan estables.
- En **Docker hub**, dentro de una imagen, tenemos una pestaña “**TAGs**” donde están todas las etiquetas de ese producto.
- Para crear un contenedor a partir de una imagen que no sea la marcada como latest en el comando de Docker tenemos que indicar el nombre de la etiqueta: la versión de la imagen.
  - **docker run --name apache2 -d httpd:alpine**



# Etiquetas de las imágenes

- En el nombre de la imagen vemos: httpd:alpine

```
C:\Users\Anton>docker ps
CONTAINER ID   IMAGE          COMMAND
1b6d56c7d2ce   httpd:alpine   "httpd-foreground"
```

- Si no le indicamos el nombre de la imagen, directamente busca la “latest”.

```
C:\Users\Anton>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    b52e0b094bc0   3 weeks ago   192MB
ubuntu        latest    a04dc4851cbc   5 weeks ago   78.1MB
httpd         alpine    fcd9ece3a2cc   5 weeks ago   63.8MB
httpd         latest    0de612e99135   5 weeks ago   148MB
hello-world   latest    74cc54e27dc4   6 weeks ago   10.1kB
fedora        latest    aa6787b90fe6   4 months ago   158MB
```

# Ejemplo

- Descargar otra imagen de Ubuntu que no sea latest:
  - **docker run --name ubuntu10 -d -it ubuntu:bionic**
    - En modo **interactivo**
  - Si ya teníamos descargada la imagen de Ubuntu (latest) va a descargar esta otra imagen de Ubuntu (bionic).
- Consultando con el comando: **docker images**

# Borrar contenedores

- Primero podemos ver todos los contenedores que tenemos:
  - **docker ps -a**
- Para borrar el contenedor: **docker rm id**
  - Podemos utilizar las primeras letras del id. El comando responde con el mismo id tecleado.
  - También podemos borrar el contenedor indicando el nombre del contenedor, en este caso poner el nombre completo:
    - **docker rm king\_Austin**

# Borrar contenedores

- Hay veces que creamos un contenedor, y cuando termine de ejecutar el comando que le hemos indicado lo queremos borrar.
  - `docker rm contenedor`
  - Este comando lo borra si está parado
- En caso contrario habría que forzar con la opción **-f**
  - `docker rm -f contenedor`
  - `docker kill contenedor` (también sería válido)

# Borrar imágenes

- `docker rmi id_imagen o nombre_imagen`
  - Se indica la “i”
  - Solo podemos borrar imágenes que no estén asociadas a contenedores
  - Si hemos borrado previamente el contenedor, nos dará ningún problema
- También se puede borrar la imagen, aunque tenga contenedores asociados:
- Con la opción `-f`
  - **`docker rmi -h id_imagen o nombre_imagen`**
  - **Los contenedores se mantienen**

# Lanzar comandos al contenedor: exec

- **docker exec**

- Ejecutar comandos contra los contenedores.
- Los contenedores estarán en modo interactivo y background
- El comando obviamente se tiene que lanzar contra un contenedor que está arrancado.

- `docker exec nombre_contenedor comando`

- **docker exec ubuntu10 date**

- Se ejecuta el comando date dentro del contenedor ubuntu10.

```
C:\Users\Anton>docker exec apache2 date  
Wed Mar  5 11:57:35 UTC 2025
```

# Lanzar comandos al contenedor: exec

- **docker exec ubuntu5 uname -a**
  - Obtener información del S.O. del contenedor
- **docker exec ubuntu5 ls -l**
  - Ejecutar ls -l en el contenedor
- También podemos **entrar dentro** del contenedor para lanzar comandos:
  - **docker exec -it ubuntu5 bash**
  - Bash se indica si el contenedor está basado en Linux

# docker attach

- Nos permite asociarnos a un contenedor
- De tal forma que lanzamos comando dentro del contenedor, parecido al exec de los comandos:
  - `docker exec -it contenedor bash`
- Al hacer el attach entro directamente en el comando (es como si estuviéramos dentro de la máquina del contenedor).

```
C:\Users\Anton>docker attach ubuntu10
root@86a63987fe7f:/# pwd
/
root@86a63987fe7f:/# date
Wed Mar  5 18:41:34 UTC 2025
root@86a63987fe7f:/#
```



# docker attach

- Para salir de la bash del contenedor: **exit**
- Podemos lanzar el contenedor y a la vez el comando top
  - **docker run --name ubuntu1 -d -it ubuntu top**
  - O probar a **lanzar el comando top dentro del contenedor.**

root@86a63987fe7f: /

```
Tasks:   3 total,   1 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.8 us,   0.1 sy,   0.0 ni, 97.9 id,   0.0 wa,   0.0 hi,   0.2 si,   0.0 st
MiB Mem : 15935.2 total, 13982.0 free,   1543.4 used,   695.1 buff/cache
MiB Swap:  4096.0 total,  4096.0 free,    0.0 used. 14391.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	4588	3976	3376	S	0.0	0.0	0:00.01	bash
9	root	20	0	2800	1152	1060	S	0.0	0.0	0:00.00	sh
18	root	20	0	8876	5064	2944	R	0.0	0.0	0:00.04	top

# docker logs

- El comando **docker logs** lo podemos utilizar para **consultar errores** que se hayan podido dar en el arranque del contenedor.
- También se puede utilizar para ver que está sacando por la terminal un contenedor que está en modo background.
- Podemos ejecutar un comando como:
  - **docker run -d Ubuntu sh -c “while true; do date; done”**
  - Con la opción **-c** indicamos el comando que tiene que ejecutar cuando arranca el contenedor.
  - En este caso en un bucle infinito imprimiendo la fecha/hora del sistema.

# docker logs

- **docker logs contenedor**

- En contenedor como siempre se indica el nombre o el id del contenedor, también podemos indicar los 4 primeros caracteres del id.
- Vemos lo que está sacando por pantalla y muestra todo.

- **docker logs contenedor --tail 10**

- Los 10 últimos comandos

- **docker logs contenedor -f**

- Se queda fijo y va mostrando sin parar.

# docker kill

- Sirve para matar el contenedor, no lo borra, pero si está arrancado lo parará.
- Más drástico que: **docker stop contenedor**
- Se envía la señal **SIGKILL** al proceso principal dentro del contenedor.
- Tenemos un parámetro `--signal` que le podemos pasar al comando. Indicando una señal de Linux.
  - **docker kill --signal=<<señal>> contenedor**
  - <https://man7.org/linux/man-pages/man7/signal.7.html>

# Recopilar información

- **docker top**

- Se relaciona con el comando top de Linux para mostrar los procesos que se están ejecutando y cuantos recursos están consumiendo.
- Vemos cual es el proceso que más está consumiendo, muestra todos los procesos.
- Arrancar desde una ventana un contenedor interactivo
  - **docker run -it ubuntu bash**
- Desde otra ventana:
  - **docker top contenedor** (utilizar el id que tenga el contenedor anterior)
    - Se puede ver con: docker ps

# Recopilar información

- **docker stats**

- Nos permite ver información sobre ese contenedor.
- Que recursos está consumiendo
- Da información sobre el uso de CPU, de la memoria, etc.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
f9819616dee7	hopeful_vaughan	0.00%	1004KiB / 15.56GiB	0.01%	1.05kB / 0B	0B / 0B	1

# docker inspect

- Recuperar información de un **contenedor** o de una **imagen**.
- El último contenedor que hemos creado lo podemos ver:
  - **docker ps -l**
- **docker inspect contenedor / imagen**
  - Nos da mucha información sobre el contenedor, mejor redirigir a un fichero.
  - La información la vuelca en formato JSON
- **docker inspect contenedor > fichero**
- **Desde docker desktop también podemos verlo de una forma más cómoda**

# docker inspect

- Para una imagen es la misma sintaxis.
- Mostrar todas las imágenes: **docker images**
- **docker image id\_imagen / nombre**
- También se aconseja redirigir a un fichero
  - **docker image id\_imagen > fichero**



# Copiar archivos

- **docker cp**

- Se puede copiar del contenedor al S.O. y al revés.

- **Un fichero del S.O. copiarlo al contenedor**

- `docker cp fichero contenedor:path_destino`
  - Hay que indicar el contenedor donde vamos a copiar, dos puntos y la el path.
  - Ejemplo: **`docker cp ejemplo.txt ubuntu10:/tmp`**
  - Entrar al contenedor para comprobar el fichero:
  - **`docker exec -it ubuntu10 bash`**

- **Desde el contenedor copiar al S.O.**

- `docker cp Ubuntu10:fichero_origen_path path_destino`
  - Ejemplo: **`docker cp ubuntu10:/tmp .`** (. Directorio actual)

# Copias de Seguridad: import / export

- **Exportar el sistema de archivos del contenedor:**

- Se puede exportar el sistema de archivos completo del contenedor
  - `docker export <container_id> > container_backup.tar`

- **docker import**

- Crea una nueva imagen Docker a partir de un **archivo tar** que contiene un sistema de archivos.
- Restaurará el sistema de archivos del contenedor, pero **no preserva el historial de capas** ni la configuración original del contenedor, como variables de entorno, volúmenes o puertos expuestos.

- **Restaurar** el contenedor a partir del fichero tar:

- `docker import container_backup.tar <new_image_name>`

- **Esta forma tiene limitaciones**

# Limitaciones

- Con **docker export** y **docker import**, se **pierde la información del historial de imágenes y las capas de Docker**.
  - Esto puede llevar a imágenes menos eficientes en términos de almacenamiento.
- **No incluye los datos de volúmenes montados**, ya que estos no forman parte del sistema de archivos del contenedor. Para respaldar los datos de los volúmenes, deberás copiarlos manualmente, como expliqué antes.
- Copias más completas del contenedor, con configuraciones y metadatos, podemos usar **docker commit** junto con **docker save**.
  - Esto preserva más información, aunque no está diseñado específicamente para copias de seguridad completas.

# Copias de Seguridad más fiables

- **Copia de seguridad del contenedor como imagen:**

- Se puede guardar el estado actual del contenedor como una imagen de Docker
  - **`docker commit <container_id> <new_image_name>`**
- Cuando tenemos la nueva imagen se puede guardar en un archivo tar.
  - **`docker save -o backup.tar <new_image_name>`**

- **Copia de seguridad de los volúmenes**

- Si el contenedor utiliza volúmenes para almacenar datos, es importante respaldarlos por separado.
  - **`docker run --rm --volumes-from <container_id> -v $(pwd):/backup ubuntu tar cvf /backup/volume_backup.tar /path/to/volume`**

# Restaurar copias de Seguridad

- **Cargar la imagen respaldada:**

- `docker load -i container_backup.tar`

- **Crear un nuevo contenedor de la imagen restaurada:**

- `docker run -it <new_image_name>`

- **Restaurar los volúmenes:**

- `docker run --rm --volumes-from <new_container_id> -v $(pwd):/backup ubuntu tar xvf /backup/volume_backup.tar -C /path/to/volume`

# Pasar variables a los contenedores

- Se pueden pasar variables a los contenedores cuando lo estamos creando.
- Son útiles para pasarlas a las imágenes y crear los contenedores
- Utilizar la opción **-e**, se pueden pasar varias variables y si son muchas se pueden pasar en un fichero.
- Si el valor de la variable lleva espacios en blanco utilizar “”
- Por convención se pasan en mayúsculas.
- **`docker run --name ubuntu1 -d -it -e V1=10 -e V2=20 ubuntu`**

# Variables de entorno

- Dentro del contenedor podemos listar las variables.
- Antes hay que conectarse al contenedor:
  - **docker exec -it ubuntu1 bash**
- Dentro del contenedor, comandos de Linux:
  - **printenv**
  - **env**
  - Para una variable concreta: **echo \$V1**

# Variables de entorno

- Si creamos un contenedor basándonos en **mariadb** de esta forma:
  - `docker run -d --name mariadb1 mariadb`
  - Se crea, pero no arranca
  - Utilizar: **`docker logs mariadb1`**
  - El error se produce por no proporcionar el valor de la variable de entorno `MARIADB_ROOT_PASSWORD`
- Habría que borrarlo y volverlo a crear pasando un valor a esa variable de entorno:
  - **`docker run -d --name mariadb1 -e MARIADB_ROOT_PASSWORD=pass mariadb`**



# Variables de entorno

- Comprobar que se ha arrancado correctamente:
  - **docker ps**
  - Entrar en el contenedor: **docker exec -it mariadb1 bash**
  - Podemos comprobar el valor de la variable con **printenv**
- **Conectar con mariadb**
  - Dentro del contenedor: **mariadb -u root -p**
  - Probar un comando: **show databases;**

# Variables de entorno en ficheros

- Más cómodo crear un fichero (por convención puede tener extensión .properties)
- Por ejemplo:
  - **mariadb.properties**
    - MARIADB\_USER=usuario
    - MARIADB\_PASSWORD=pass
    - MARIADB\_DATABASE=base\_datos
  - En la llamada al comando se pasa el fichero:
    - **docker run --name mariadb2 -d --env-file mariadb.properties mariadb**
- Probar a entrar y visualizar las bases de datos

# Mapeo de puertos

- Al crear contenedores con el comando run o create por defecto no se publica ningún puerto.
- Hay que indicarlo de forma explícita.
- Por defecto, los puertos de un contenedor solo están accesibles desde:
  - Desde la misma red
  - Desde el host de Docker
  - Desde la red bridge de Docker (a través de la interfaz docker0 en los Linux)

# Mapeo de puertos

- Tenemos las opciones:
- -p, --publish
- -P, --publish-all
- Con -p se puede publicar un puerto o un rango de puertos
- Cuando el host de Docker recibe la petición en uno de sus puertos, la redirige al puerto correspondiente del contenedor.
- [host:]contenedor[/protocolo]

# Mapeo de puertos

- Por defecto, Docker publica en la dirección ip 0.0.0.0 que hace referencia a todas las interfaces de red que existan en el host.
- Por ejemplo:
  - `docker run -d --rm --name http_c -p 81:80`
  - El host de Docker recibe una petición por el puerto 81 y se redirige al 80 del contenedor.

# Mapeo de puertos

- Opción `-P`, `--publish-all`
- Con esta opción publicamos todos los puertos que se hayan expuesto en la imagen que se ha utilizado para crear el contenedor, en puertos aleatorios en el host.
  - **`docker run -d --rm --name http_c -P httpd`**
  - Se elige un puerto aleatorio, por ejemplo: 32770

# Estadísticas de uso

- **docker stats**

- Muestra las estadísticas de uso de los contenedores que están en marcha.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
912ea72fc28e	mysql_c	0.65%	389.6MiB / 15.56GiB	2.44%	1.57kB / 0B	0B / 0B	37

- **Las columnas:**

- ID del contenedor
- Uso de CPU (%)
- Uso de memoria y límite
- E/S de red (Net I/O)
- E/S de disco (Block I/O)
- Número de procesos (PIDs)

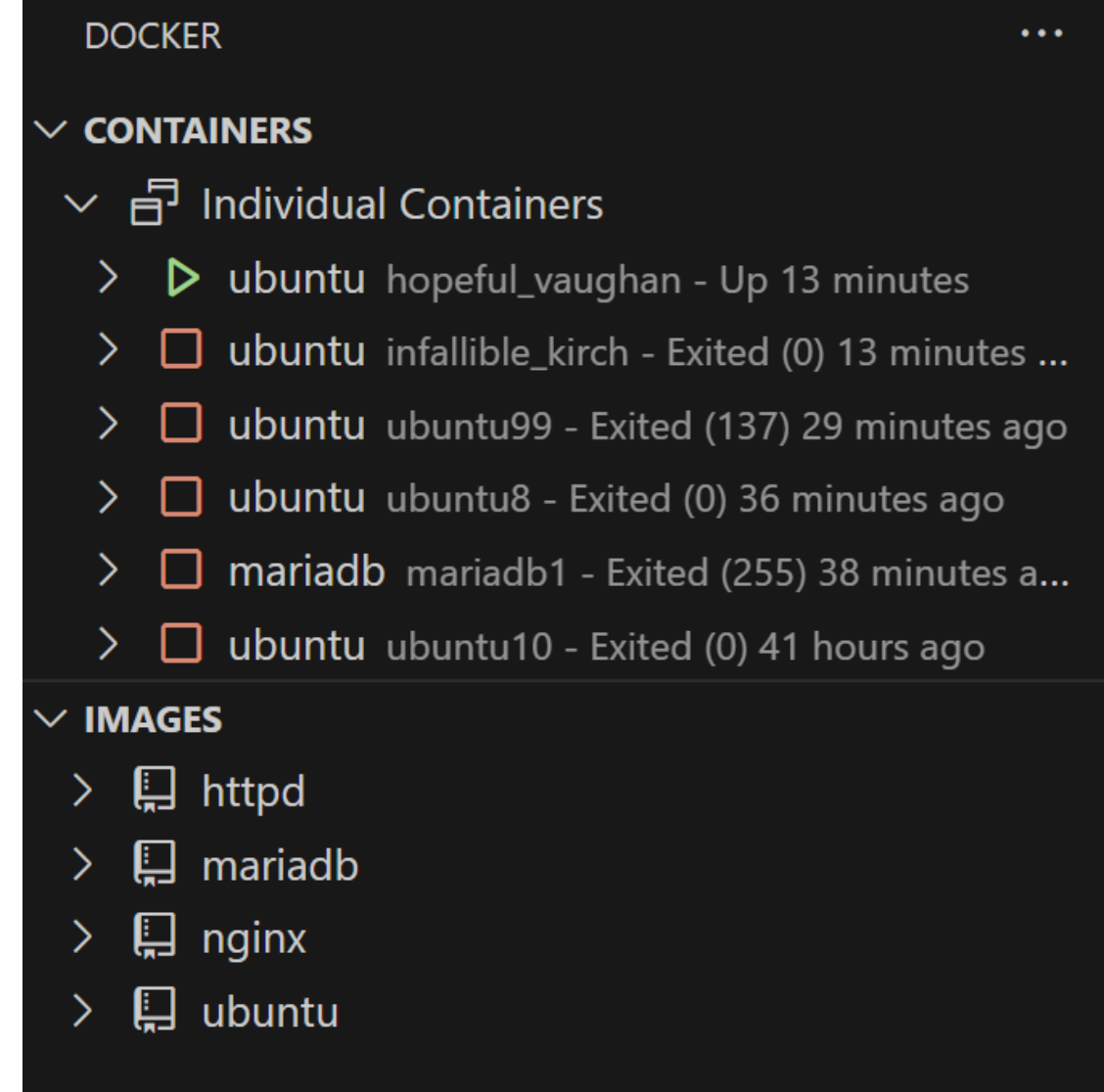
## Opciones:

**--no-stream** Las estadísticas se muestran una vez,  
En lugar de estar actualizando todo el rato.

**--all** incluye también los contenedores detenidos.

# Otras herramientas

- Podemos utilizar:
  - Visual Studio Code:
  - <https://code.visualstudio.com/download>
- Nos proporciona un plugin para Docker y en la vista de Docker nos muestra contenedores, imágenes, volúmenes, etc.





# Apéndice

- 1) **docker start** – To start a Docker container
- 2) **docker stop** – To stop a running docker container
- 3) **docker restart** – To restart docker container
- 4) **docker pause** – To pause a running container
- 5) **docker unpause** – To unpause a running container
- 6) **docker run** – Creates a docker container from docker image
- 7) **docker ps** – To list Docker containers

# Apéndice II

**8) docker exec** – To Access the shell of Docker Container

**9) docker logs** – To view logs of Docker container

**10) docker rename** – To rename Docker container

**11) docker rm** – To remove Docker container

**12) docker inspect** – Docker container info command

**13) docker attach** – Attach Terminal to Running container

**14) docker kill** – To stop and remove Docker containers

**15) docker cp** – To copy files or folders between a container and from local filesystem.