

# Tarea 5: Contenedores



**docker**

**vs**



**podman**

## Docker

[Principales operaciones](#)

[Instalación Docker Ubuntu](#)

[Persistencia de datos](#)

[Volúmenes](#)

[Bind mounts](#)

## Podman

[Docker rootfull-Podman rootless](#)

[Instalación](#)

[podman machine init](#)

[podman machine start](#)

[podman run](#)

[podman ps](#)

[podman machine ls](#)

[podman machine stop](#)

[Podman-servidor web](#)

[Podman logs](#)

[podman stop](#)

## MySQL en Docker

## Docker Hub

[Usar nuestras propias imágenes](#)

[Docker registry](#)

[Principales comandos](#)[Enlazando contenedores](#)[Usar --link \(obsoleto pero aún funciona\)](#)[Usando red personalizada \(network\)](#)[Enlazando tres contenedores](#)[Dockerfile](#)[Palabras reservadas](#)[Ejercicio dockerfile autoescuela](#)[Ejercicio dockerfile Mongodb](#)[Ejemplo dockerfile dos contenedores](#)[SQL Server](#)[Crear contenedor y arranque](#)[SQL Server con Dockerfile](#)[Ejemplo backup-restore](#)[MongoDB](#)[Acceso desde la bash del huésped](#)[Acceso desde MongoDB Compass](#)[Docker Compose](#)[docker-entrypoint.sh](#)[Ejercicio 1 Docker-compose \(mysql\)](#)[Ejercicio 2 Docker-compose \(Apache-Mysql\)](#)[Ejercicio 3 Docker-compose \(wordpress\)](#)[Ejercicio 4 Docker compose \(postgre y pgadmin\)](#)[Ejercicio 5 Docker-compose \(MongoDB\)](#)[Ejercicio 6 Docker-compose \(Mongodb-webapp\)](#)[Portainer](#)[Dockge](#)[Enlace a repositorio GitHub](#)

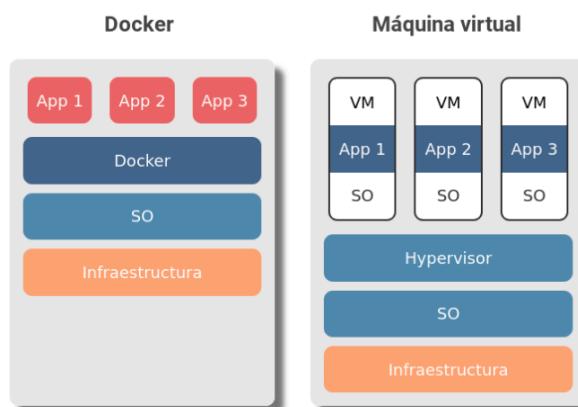
## Docker

**Docker** es una plataforma de contenedores que permite empaquetar, distribuir y ejecutar aplicaciones de manera **aislada**, rápida y eficiente.

En lugar de instalar software directamente en tu sistema operativo, **Docker encapsula** la aplicación junto con todas sus dependencias dentro de un **contenedor**. Esto facilita la portabilidad y evita problemas de compatibilidad.

- **Docker Engine** → Es el software que gestiona los contenedores.
- **Imagen Docker** → Es el "modelo" de un contenedor, incluye el sistema operativo y las dependencias.
- **Contenedor** → Es una instancia en ejecución de una imagen Docker.
- **Docker Hub** → Repositorio público donde se pueden descargar imágenes listas para usar.

La principal diferencia que hay entre un contenedor (Docker) y una máquina virtual es que el contenedor usa el SO del host y por tanto se ahorran muchísimos recursos.



Característica	Docker (Contenedores)	Máquinas Virtuales (VMs)
<b>SO completo</b>	✗ No, usa el del host	✓ Sí, cada VM tiene el suyo
<b>Uso de recursos</b>	⚡ Ligero (usa menos RAM/CPU)	🐢 Pesado (requiere más RAM/CPU)
<b>Tiempo de arranque</b>	🚀 Segundos	⌚ Minutos
<b>Portabilidad</b>	🌐 Muy fácil de mover y ejecutar en cualquier sistema	📦 Más complicado de migrar
<b>Aislamiento</b>	🔒 Parcial (comparten el núcleo del SO)	🔒 Completo (cada VM es independiente)

## Principales operaciones

### Gestión de Imágenes

Las imágenes son plantillas listas para ejecutar contenedores.

Comando	Descripción
<code>docker pull &lt;imagen&gt;</code>	Descarga una imagen desde Docker Hub.
<code>docker images</code>	Lista las imágenes disponibles en tu máquina.
<code>docker rmi &lt;imagen&gt;</code>	Elimina una imagen.
<code>docker build -t &lt;nombre&gt; .</code>	Crea una imagen a partir de un <code>Dockerfile</code> .

### Gestión de Contenedores

Los contenedores son instancias activas de una imagen.

Comando	Descripción
<code>docker run &lt;imagen&gt;</code>	Ejecuta un contenedor desde una imagen.
<code>docker run -d &lt;imagen&gt;</code>	Ejecuta el contenedor en segundo plano (modo <i>detached</i> ).
<code>docker run -it &lt;imagen&gt; bash</code>	Ejecuta el contenedor con acceso a consola interactiva.
<code>docker ps</code>	Lista los contenedores activos.
<code>docker ps -a</code>	Lista todos los contenedores (activos e inactivos).
<code>docker stop &lt;id/nombre&gt;</code>	Detiene un contenedor en ejecución.
<code>docker start &lt;id/nombre&gt;</code>	Inicia un contenedor detenido.
<code>docker rm &lt;id/nombre&gt;</code>	Elimina un contenedor.
<code>docker exec -it &lt;id/nombre&gt; bash</code>	Entra al terminal de un contenedor en ejecución.
<code>docker container prune</code>	Elimina todos los contenedores detenidos

## Volúmenes y persistencia

Los volúmenes se usan para guardar datos fuera del contenedor.

Comando	Descripción
<code>docker volume create &lt;nombre&gt;</code>	Crea un volumen.
<code>docker volume ls</code>	Lista los volúmenes.
<code>docker run -v &lt;volume&gt;:/ruta/en/contenedor &lt;imagen&gt;</code>	Usa un volumen en un contenedor.

## Redes

Docker crea redes virtuales para que los contenedores se comuniquen.

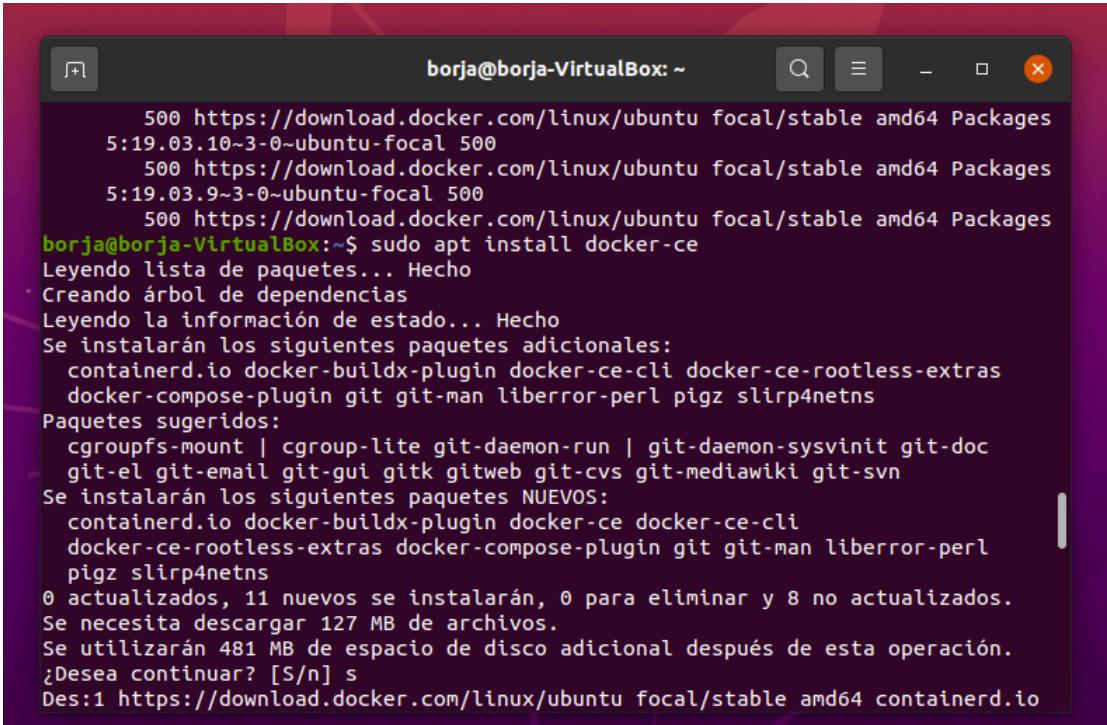
Comando	Descripción
<code>docker network ls</code>	Lista las redes disponibles.
<code>docker network create &lt;nombre&gt;</code>	Crea una red personalizada.
<code>docker run --network &lt;red&gt; &lt;imagen&gt;</code>	Conecta el contenedor a una red específica.

## Otros útiles

Comando	Descripción
<code>docker info</code>	Muestra información general sobre tu instalación de Docker.
<code>docker version</code>	Muestra la versión instalada.
<code>docker logs &lt;id/nombre&gt;</code>	Muestra los registros (logs) de un contenedor.
<code>docker inspect &lt;id/nombre&gt;</code>	Muestra detalles técnicos de un contenedor o imagen.
<code>docker login</code>	Inicia sesión en Docker Hub.
<code>docker push &lt;imagen&gt;</code>	Sube una imagen a Docker Hub (debes estar logueado).

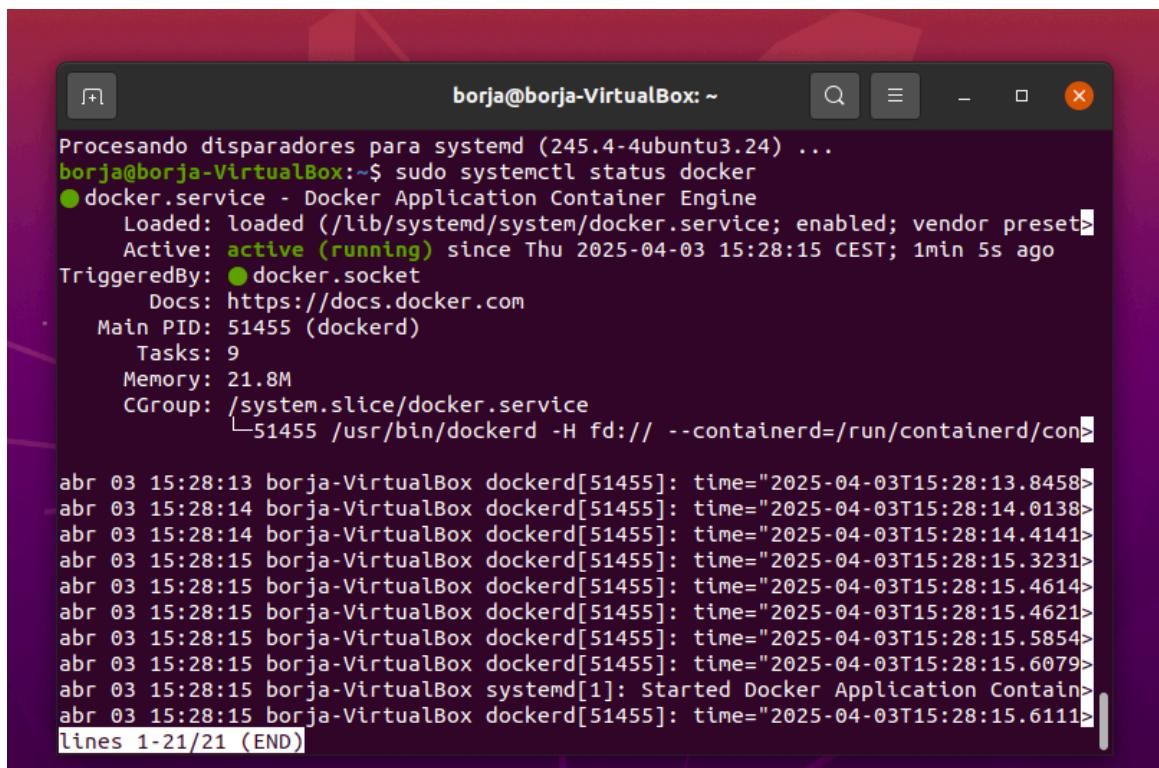
## Instalación Docker Ubuntu

```
-- Actualiza la lista de paquetes disponibles desde los repositorios configurados.  
sudo apt update  
  
-- Instala herramientas necesarias para poder trabajar con repositorios HTTPS y gestionar certificados  
sudo apt install apt-transport-https ca-certificates curl software-properties-common  
  
-- Descarga la clave GPG oficial de Docker y la añade al sistema.  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
  
-- Añade el repositorio oficial de Docker para Ubuntu 20.04 (nombre en clave: focal).  
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu focal stable"  
  
-- Actualiza nuevamente la lista de paquetes disponibles, esta vez incluyendo el repositorio de Docker que acabas de añadir.  
sudo apt update  
  
-- Te muestra información sobre el paquete docker-ce apt-cache policy docker-ce  
apt-cache policy docker-ce  
  
-- Instala Docker CE (Community Edition) desde el repositorio oficial.  
sudo apt install docker-ce  
  
-- Muestra el estado de Docker  
sudo systemctl status Docker
```



```
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
5:19.03.10-3~ubuntu-focal 500  
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
5:19.03.9-3~ubuntu-focal 500  
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
borja@borja-VirtualBox:~$ sudo apt install docker-ce  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras  
  docker-compose-plugin git git-man liberror-perl pigz slirp4netns  
Paquetes sugeridos:  
  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  
Se instalarán los siguientes paquetes NUEVOS:  
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli  
  docker-ce-rootless-extras docker-compose-plugin git git-man liberror-perl  
  pigz slirp4netns  
0 actualizados, 11 nuevos se instalarán, 0 para eliminar y 8 no actualizados.  
Se necesita descargar 127 MB de archivos.  
Se utilizarán 481 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n] s  
Des:1 https://download.docker.com/linux/ubuntu focal/stable amd64 containerd.io
```

Como vemos, el servicio se encuentra activo:

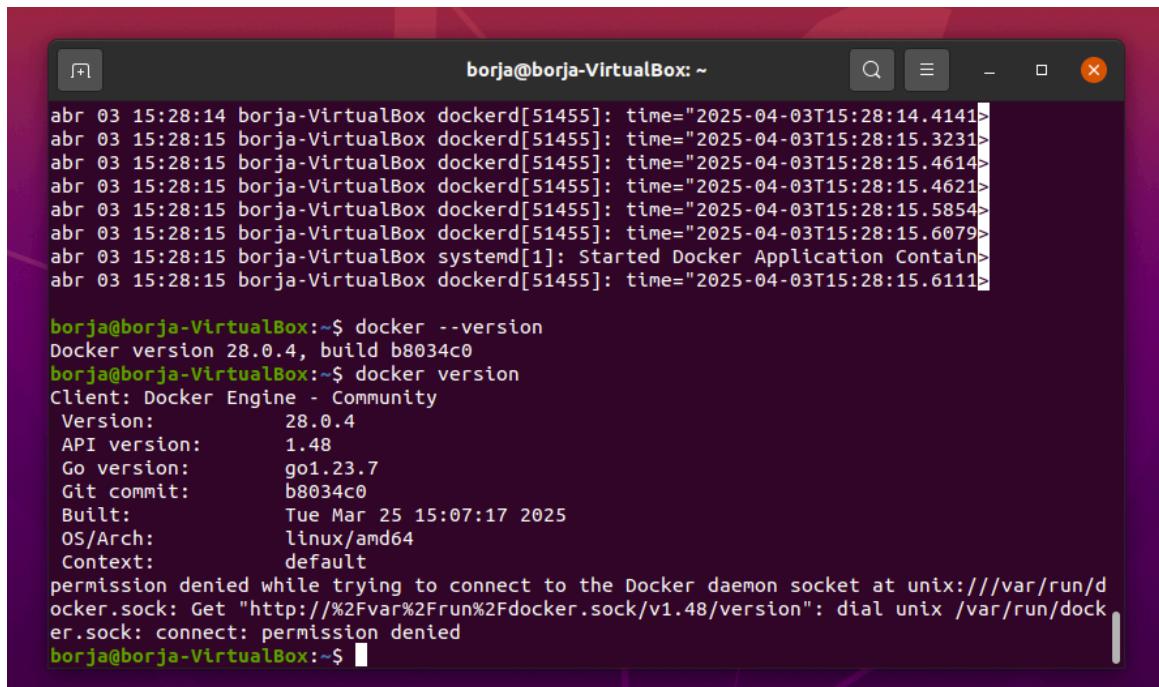


```
Procesando disparadores para systemd (245.4-4ubuntu3.24) ...
borja@borja-VirtualBox:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2025-04-03 15:28:15 CEST; 1min 5s ago
  TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 51455 (dockerd)
     Tasks: 9
    Memory: 21.8M
      CGroup: /system.slice/docker.service
              └─51455 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

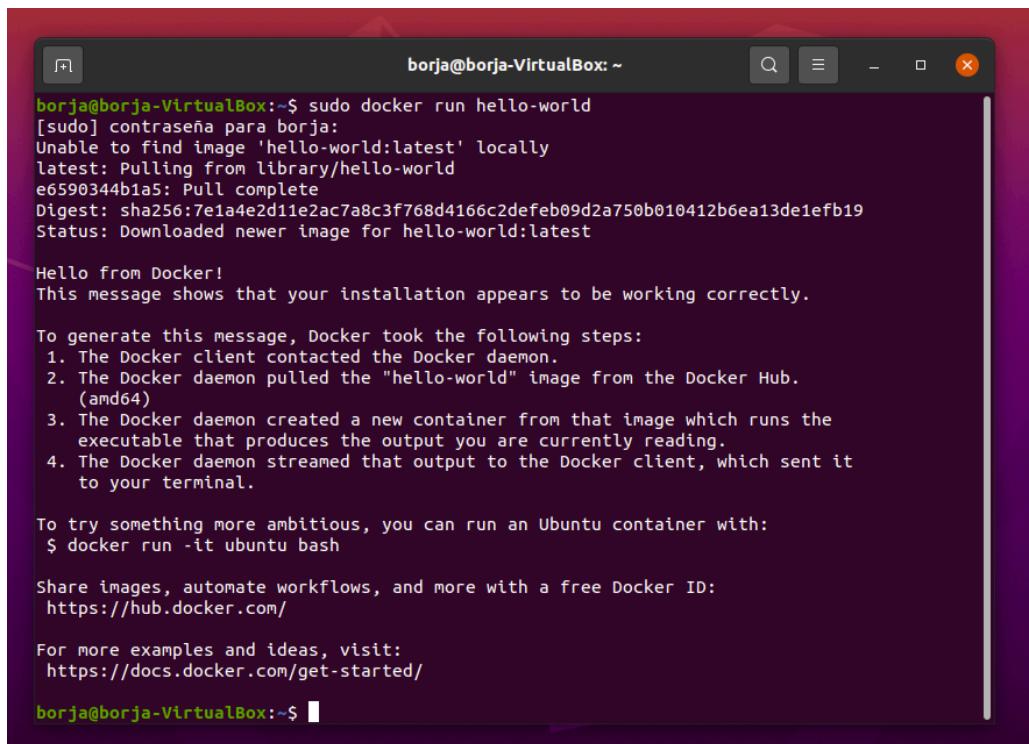
abr 03 15:28:13 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:13.845Z" level=info msg="Starting Docker Application Container Engine"
abr 03 15:28:14 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:14.013Z" level=info msg="Docker daemon is running"
abr 03 15:28:14 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:14.414Z" level=info msg="Docker daemon has initialized"
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.323Z" level=info msg="Listening on fd://"
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.461Z" level=info msg="Docker daemon is ready to receive API calls"
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.462Z" level=info msg="Docker daemon has initialized"
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.585Z" level=info msg="Docker daemon has initialized"
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.607Z" level=info msg="Docker daemon has initialized"
abr 03 15:28:15 borja-VirtualBox systemd[1]: Started Docker Application Container Engine
abr 03 15:28:15 borja-VirtualBox dockerd[51455]: time="2025-04-03T15:28:15.611Z" level=info msg="Docker daemon has initialized

lines 1-21/21 (END)
```

Comprobamos la versión:



```
borja@borja-VirtualBox:~$ docker --version
Docker version 28.0.4, build b8034c0
borja@borja-VirtualBox:~$ docker version
Client: Docker Engine - Community
  Version:          28.0.4
  API version:      1.48
  Go version:       go1.23.7
  Git commit:       b8034c0
  Built:            Tue Mar 25 15:07:17 2025
  OS/Arch:          linux/amd64
  Context:          default
  permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://var/run/docker.sock/v1.48/version": dial unix /var/run/docker.sock: connect: permission denied
borja@borja-VirtualBox:~$
```



```
borja@borja-VirtualBox:~$ sudo docker run hello-world
[sudo] contraseña para borja:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b010412b6ea13de1efb19
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/
borja@borja-VirtualBox:~$
```

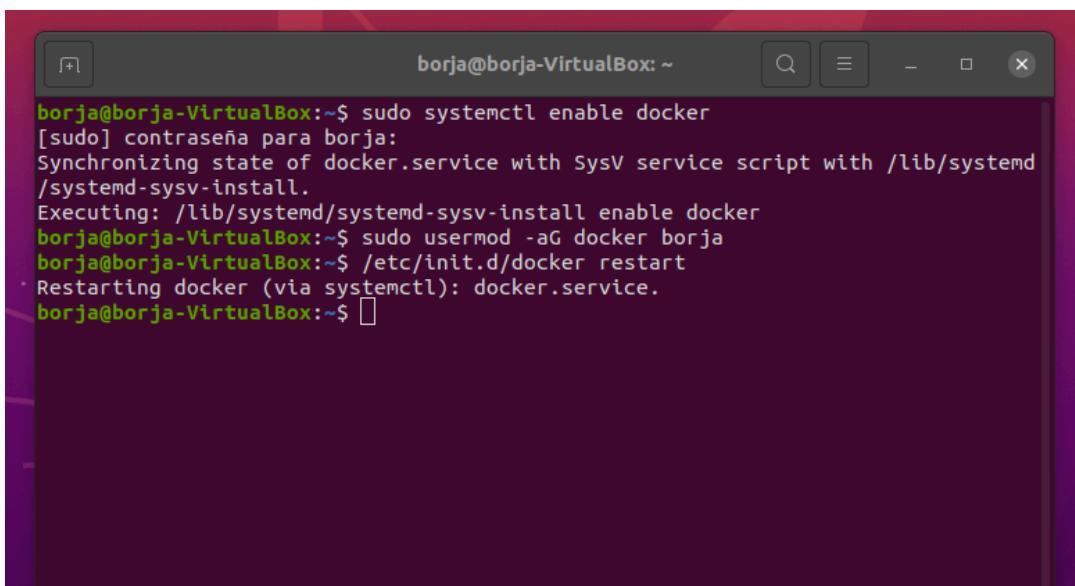
Ahora vamos a añadir un usuario al grupo docker:

Añadir un usuario al grupo, le da privilegio similares a root, lo que hace que no tengamos que estar escribiendo continuamente “sudo”, en entornos de prueba como este se suele usar para agilizar todo pero en producción se debe tener cuidado. Esta es una de las ventajas que tiene Podman sobre Docker, puesto que usa un usuario rootless (sin privilegios).

```
-- Iniciamos Docker
borja@borja-VirtualBox:~$ sudo systemctl enable Docker

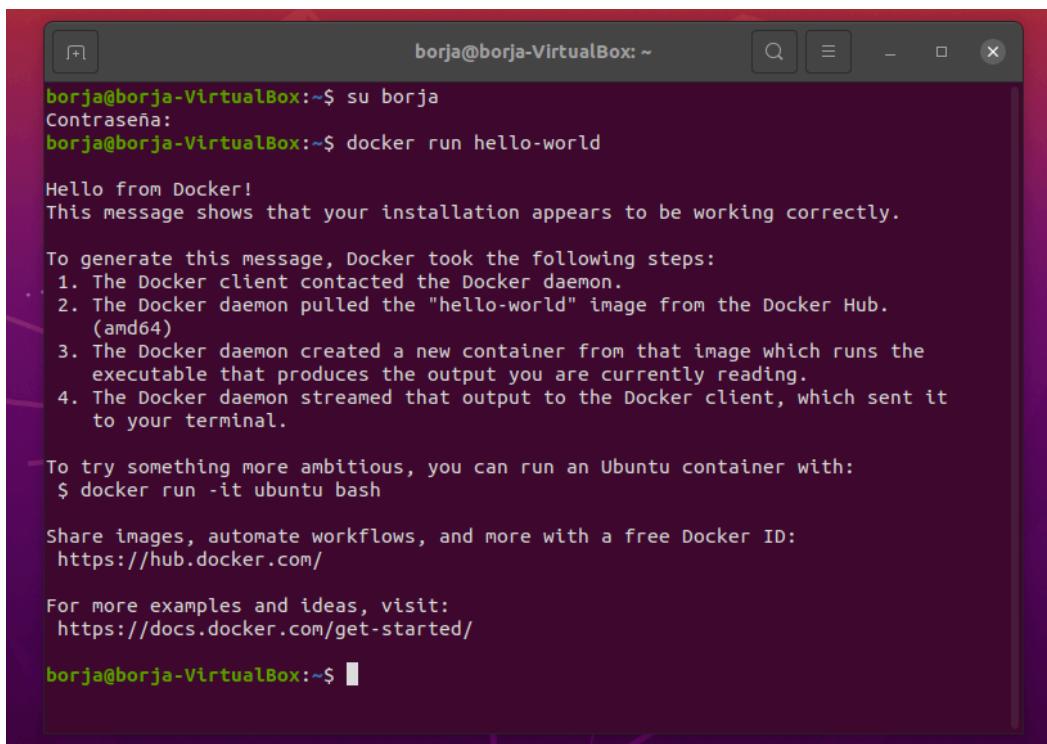
-- Añadimos el usuario
borja@borja-VirtualBox:~$ sudo usermod -aG docker Borja

-- Reiniciamos el demonio
borja@borja-VirtualBox:~$ /etc/init.d/docker restart
```



```
borja@borja-VirtualBox:~$ sudo systemctl enable docker
[sudo] contraseña para borja:
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
borja@borja-VirtualBox:~$ sudo usermod -aG docker borja
borja@borja-VirtualBox:~$ /etc/init.d/docker restart
* Restarting docker (via systemctl): docker.service.
borja@borja-VirtualBox:~$
```

Ahora iniciamos con el usuario creado:



```
borja@borja-VirtualBox:~$ su borja
Contraseña:
borja@borja-VirtualBox:~$ docker run hello-world

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/

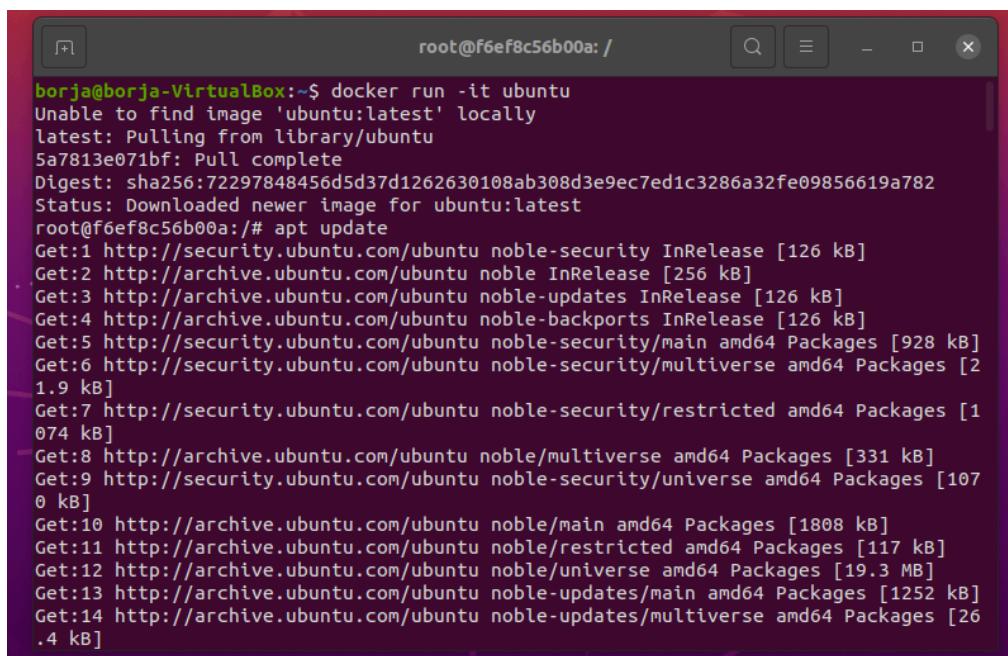
For more examples and ideas, visit:
https://docs.docker.com/get-started/
borja@borja-VirtualBox:~$
```

Vamos a probar a descargar una imagen de Ubuntu y a ejecutar parámetros dentro de el como por ejemplo instalar Apache:

```
-- Descargamos imagen de Ubuntu
      docker pull Ubuntu

-- Creamos y arrancamos un contenedor
      docker run -it Ubuntu

-- Una vez arrancado, ejecutamos parámetros ya desde la terminal del contenedor
      para instalar Apache
          apt update
          apt install curl
          apt install apache2
```



```
root@f6ef8c56b00a: /root
root@f6ef8c56b00a:~$ docker run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
5a7813e071bf: Pull complete
Digest: sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
Status: Downloaded newer image for ubuntu:latest
root@f6ef8c56b00a:~# apt update
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [928 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [2 1.9 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [1 074 kB]
Get:8 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [107 0 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1252 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [26 4 kB]
```

```

Running hooks in /etc/ca-certificates/update.d...
done.
root@f6ef8c56b00a:/# apt install apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  adduser apache2-bin apache2-data apache2-utils libapr1t64
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libaprutil1t64 libexpat1
  libgdbm-compat4t64 libgdbm6t64 libicu74 libjansson4 liblua5.4-0 libperl5.38t64
  libsqlite3-0 libxml2 media-types netbase perl perl-base perl-modules-5.38
  ssl-cert
Suggested packages:
  liblocale-gettext-perl cron quota cryptfs-utils apache2-doc
  apache2-suexec-pristine | apache2-suexec-custom www-browser ufw gdm libterm-readline-gnu-perl | libterm-readline-perl-perl make
  libtap-harness-archive-perl
The following NEW packages will be installed:
  adduser apache2 apache2-bin apache2-data apache2-utils libapr1t64
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libaprutil1t64 libexpat1
  libgdbm-compat4t64 libgdbm6t64 libicu74 libjansson4 liblua5.4-0 libperl5.38t64
  libsqlite3-0 libxml2 media-types netbase perl perl-modules-5.38 ssl-cert
The following packages will be upgraded:
  perl-base
1 upgraded, 23 newly installed, 0 to remove and 18 not upgraded.

```

Ahora salimos del contenedor y consultamos las imágenes que tenemos con *docker images* y *docker ps -a*:

```

Enabling conf charset.
Enabling conf localized-error-pages.
Enabling conf other-vhosts-access-log.
Enabling conf security.
Enabling conf serve-cgi-bin.
Enabling site 000-default.
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
root@f6ef8c56b00a:/# exit
exit
borja@borja-VirtualBox:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ubuntu          latest   a04dc4851cbc  2 months ago  78.1MB
hello-world     latest   74cc54e27dc4  2 months ago  10.1kB
borja@borja-VirtualBox:~$ docker ps -a
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS
      PORTS      NAMES
f6ef8c56b00a      ubuntu      "/bin/bash"   9 minutes ago   Exited (0) About a minute ago
      trusting_poincare
868105b5f9d9      hello-world  "/hello"     12 minutes ago   Exited (0) 12 minutes ago
      zealous_antonelli
cc2dd4677c58      hello-world  "/hello"     53 minutes ago   Exited (0) 52 minutes ago
borja@borja-VirtualBox:~$ 

```

## Persistencia de datos

- **Contenedor sin persistencia de datos:** todo lo que se almacena en el sistema de archivos del contenedor se pierde al detenerlo, eliminarlo o recrearlo.
- **Contenedor con persistencia de datos:** los datos se guardan fuera del contenedor (en el host o en otro servicio), por lo que sobreviven a reinicios, recreaciones o eliminaciones del contenedor.

Lo bueno de la persistencia es que podemos usar un volumen para guardar los datos y si ocurre cualquier cosa o borramos el contenedor, podremos crear otro contenedor y usar el mismo volumen en donde tenemos guardados los datos, funciona como un disco duro extraible por decirlo de alguna manera. Aunque hay dos métodos para usar la persistencia de datos (Volúmenes y bind mounts), lo que a mi juicio es más usado y mas sencillo para nosotros es usar volúmenes, ya que docker gestiona automáticamente dichos volúmenes y eso hace que la información sea más portable.

## Volúmenes

Vamos a realizar una prueba:

Creamos un volumen para guardar los datos:

```
docker volume create datos-mysql
```

Lanzamos un contenedor Mysql y le decimos que use el volumen:

```
docker run -d \
  --name mysql-persistente \
  -e MYSQL_ROOT_PASSWORD=Abcd1234. \
  -v datos-mysql:/var/lib/mysql \
  mysql:latest
```

Verificamos si el volumen se está usando:

```
docker volume ls
docker inspect datos-mysql
```

```
|borja@MacBook-Pro-de-Borja mysql % docker volume create datos-mysql
datos-mysql
borja@MacBook-Pro-de-Borja mysql % docker run -d \
  --name mysql-persistente \
  -e MYSQL_ROOT_PASSWORD=Abcd1234. \
  -v datos-mysql:/var/lib/mysql \
  mysql:latest
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
c2eb5d06bfea: Already exists
ba361f0ba5e7: Pull complete
0e83af98b000: Pull complete
770e931107be: Pull complete
a2be1b721112: Pull complete
68c594672ed3: Pull complete
cf201189145: Pull complete
e9f009c5b388: Pull complete
61a291920391: Pull complete
c8604ede059a: Pull complete
Digest: sha256:2247f6d47a59e5fa30a27ddc2e183a3e6b05bc045e3d12f8d429532647f61358
Status: Downloaded newer image for mysql:latest
9372ef710f46c6075081aaa083e00fb525f8970b2e2fd4f6f491823a986fd288
|borja@MacBook-Pro-de-Borja mysql % docker volume ls
DRIVER      VOLUME NAME
local      04c9f486fd56c9184b42b41ace2df69a7410bcf96bf1fc0e849601b26af79402
local      6c7584d33ecec787330bdd2a266e7b4a132c6f0a164360dd16c26ce511a43e74
local      6f8d16ae9fe10e0fe58a4baa4129cd4a1c0e0d6654e5533d2d7b9b9e22a8463f
local      60de05862146337ec212485cc6bb4a53a07ea2316ea429f35d26a908a278fe96
local      766a8ae37f0ca221e089831967b4433a6fd283101c1d99256e5ca7998afe19be
local      2262e46d1094b0d662f1aa242a57b66ff611d187b11bc0f2bb6a7bd459fc525b
local      cae68c055c2c8007fc5334b9766c1a534f42953be75a57343d7c75c9e6f9dde8
local      datos-mysql
|borja@MacBook-Pro-de-Borja mysql % docker inspect datos-mysql
[
  {
    "CreatedAt": "2025-05-06T11:48:27Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/datos-mysql/_data",
    "Name": "datos-mysql",
    "Options": {},
    "Scope": "local"
  }
]
borja@MacBook-Pro-de-Borja mysql %
```

Creamos una base de datos dentro del contenedor:

```
CREATE DATABASE autoescuela;
SHOW DATABASES;

[borja@MacBook-Pro-de-Borja mysql % docker exec -it mysql-persistente mysql -uroot -p

Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE autoescuela;
Query OK, 1 row affected (0.004 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| autoescuela |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.006 sec)

mysql> exit
Bye
borja@MacBook-Pro-de-Borja mysql %
```

Ahora borramos el contenedor y creamos otro con el mismo volumen, como podemos ver, sigue existiendo la base de datos creada en el contenedor anterior.

```
borja@MacBook-Pro-de-Borja mysql % docker stop mysql-persistente
mysql-persistente
borja@MacBook-Pro-de-Borja mysql % docker rm mysql-persistente

mysql-persistente
borja@MacBook-Pro-de-Borja mysql % docker run -d \
  --name mysql-restaurado \
  -e MYSQL_ROOT_PASSWORD=Abcd1234. \
  -v datos-mysql:/var/lib/mysql \
  mysql:latest

75c4bd2d7dd9b94b8aeb246e055535f559ab62cf3a65c0072d6e70d73931efbe
borja@MacBook-Pro-de-Borja mysql % docker exec -it mysql-restaurado mysql -uroot -p

Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| autoescuela |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.006 sec)

mysql>
```

## Bind mounts

**Bind mounts** conectan directamente una ruta del sistema de archivos del host con una ruta dentro del contenedor. A diferencia de los volúmenes de Docker, en los bind mounts controlamos la ubicación en el host y podemos ver y modificar los archivos directamente desde fuera del contenedor.

Vamos a ver un ejemplo, aunque nos quedaremos con que lo mas usado son los volúmenes.

Creamos una carpeta en nuestro ordenador para almacenar los datos que creemos en Mysql:

```
mkdir -p ~/docker_mysql_data
chmod 777 ~/docker_mysql_data
```

Ejecutamos el contenedor con bind mount:

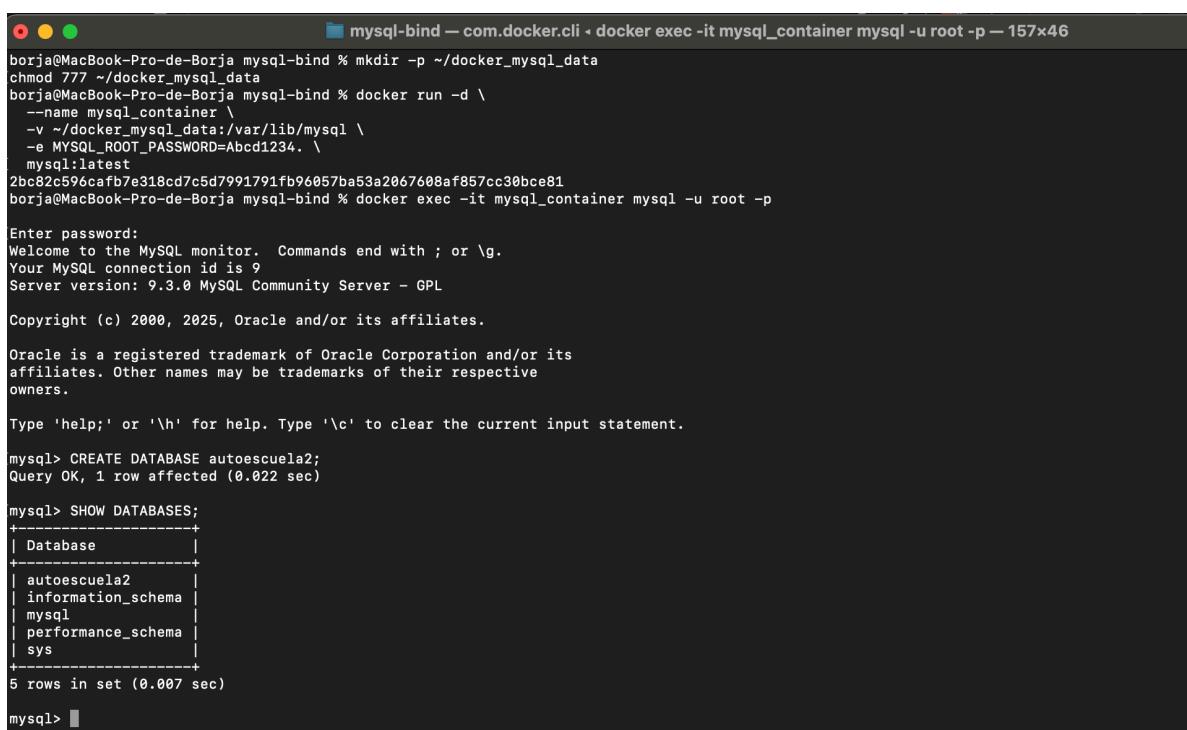
```
docker run -d \
--name mysql_container \
-v ~/docker_mysql_data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=Abcd1234 \
mysql:latest
```

Entramos al contenedor:

```
docker exec -it mysql_container mysql -u root -p
```

Creamos la base de datos:

```
CREATE DATABASE autoescuela2;
SHOW DATABASES;
```



```
mysql-bind -- com.docker.cli - docker exec -it mysql_container mysql -u root -p -- 157x46
borja@MacBook-Pro-de-Borja mysql-bind % mkdir -p ~/docker_mysql_data
chmod 777 ~/docker_mysql_data
borja@MacBook-Pro-de-Borja mysql-bind % docker run -d \
--name mysql_container \
-v ~/docker_mysql_data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=Abcd1234 \
mysql:latest
2bc82c596caf7e318cd7c5d7991fb96057ba53a2067608af857cc30bce81
borja@MacBook-Pro-de-Borja mysql-bind % docker exec -it mysql_container mysql -u root -p

Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

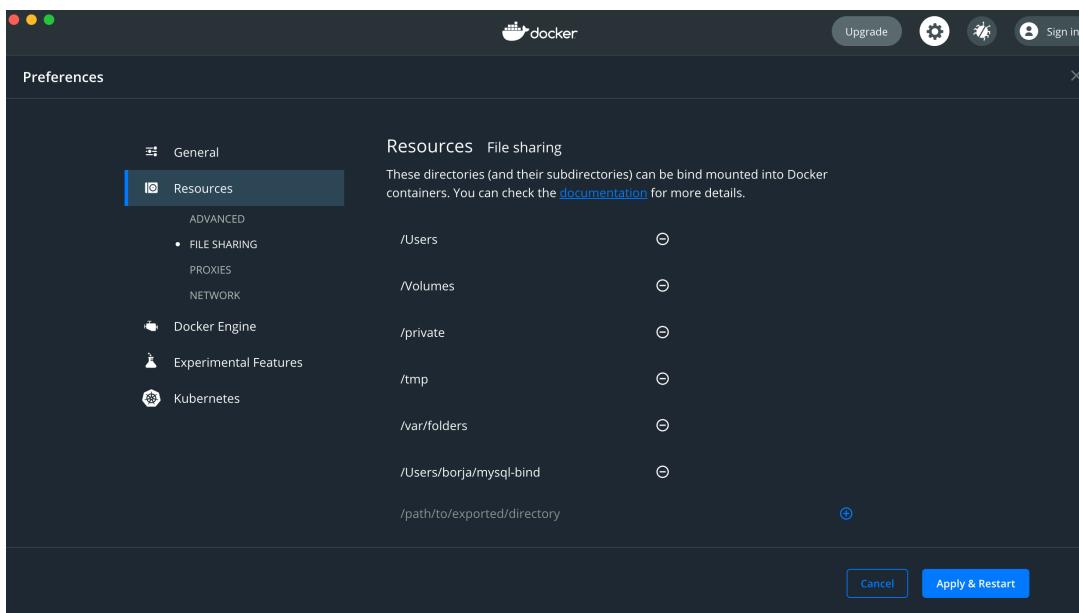
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE autoescuela2;
Query OK, 1 row affected (0.022 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| autoescuela2 |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.007 sec)

mysql> █
```

En ocasiones puede dar error de permisos. Esto es debido a que docker no tiene acceso de por si al sistema de archivos, debemos ir a Docker y acceder a preferencias→Resources y añadir la ruta a la que le queremos dar acceso.



Ahora borramos el contenedor y creamos uno nuevo usando el mismo directorio:

```
docker stop mysql_container
docker rm mysql_container
```

Creamos un nuevo contenedor:

```
docker run -d \
--name mysql_container2 \
-v ~/docker_mysql_data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=Abcd1234. \
mysql:latest
```

Entramos al contenedor:

```
docker exec -it mysql_container2 mysql -u root -p

borja@MacBook-Pro-de-Borja mysql-bind % docker stop mysql_container
docker rm mysql_container

mysql_container
mysql_container
borja@MacBook-Pro-de-Borja mysql-bind % docker run -d \
--name mysql_container2 \
-v ~/docker_mysql_data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=Abcd1234. \
mysql:latest
8777ad6b4487e92c7bb28e2d73d93799af5ee87b23a4ec22ffe1970c5e055722
|borja@MacBook-Pro-de-Borja mysql-bind % docker exec -it mysql_container2 mysql -u root -p
|Enter password:
|Welcome to the MySQL monitor. Commands end with ; or \g.
|Your MySQL connection id is 9
|Server version: 9.3.0 MySQL Community Server - GPL

|Copyright (c) 2000, 2025, Oracle and/or its affiliates.

|Oracle is a registered trademark of Oracle Corporation and/or its
|affiliates. Other names may be trademarks of their respective
|owners.

|Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[mysql> SHOW DATABASES;
+-----+
| Database      |
+-----+
| autoescuela2 |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.007 sec)

mysql> |
```

Como podemos comprobar, aunque es un nuevo contenedor, conserva los datos que guardamos en el directorio indicado.

Para borrar todos los contenedores detenidos usamos docker container prune.

## Podman

Podman es una herramienta para gestionar contenedores y pod (grupos de contenedores) que funciona como alternativa a Docker, pero con algunas diferencias importantes.

Su arquitectura inclusiva y sin daemons la convierte en una opción más segura y accesible para la gestión de los contenedores. Además, las funciones y las herramientas que la acompañan, permiten que los desarrolladores personalicen los entornos de contenedores según sus necesidades.

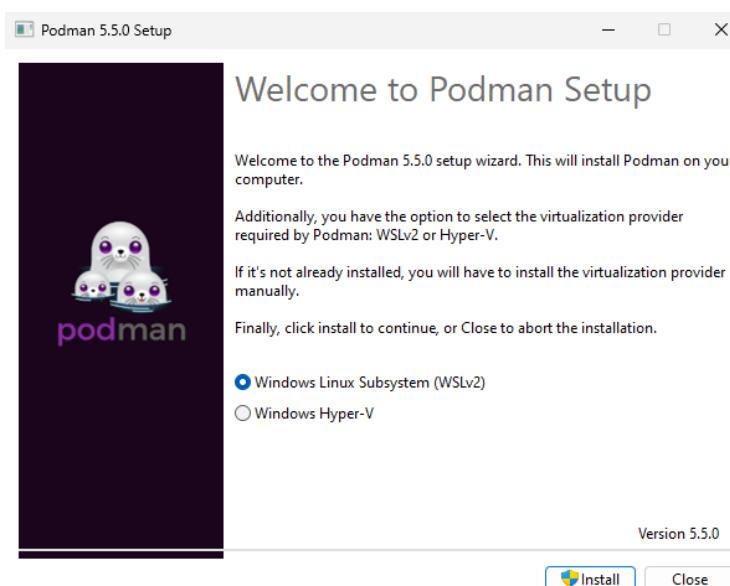
### Docker rootfull-Podman rootless

Docker requiere privilegios de root porque su demonio (dockerd) gestiona recursos del sistema, redes y almacenamiento a bajo nivel, lo que implica riesgos de seguridad si el demonio es comprometido. (rootfull)

Podman no necesita un demonio central y puede ejecutarse en modo rootless, es decir, sin privilegios de administrador. Esto reduce la superficie de ataque, mejora la seguridad y permite que usuarios sin permisos elevados gestionen contenedores sin riesgos asociados a la escalada de privilegios. (rootless)

## Instalación

Vamos a probar a instalar podman sobre Windows, ya al comienzo de la instalación nos dice que vamos a tener que usar o bien WLS o bien Hyper-V. En este caso hemos optado por WSL ya que lo teníamos ya instalado para realizar otras prácticas ([instalación WLS aquí](#))



## podman machine init

Crea un sistema (máquina virtual) para poder trabajar con contenedores, ya que podman no puede correr directamente sobre windows o mac, lo que hace este comando es levantar una especie de máquina virtual con un sistema linux minimalista. (en este caso ya la teníamos creada)

```
PS C:\Users\promi> podman machine init
podman : Error: podman-machine-default: VM already exists
En linea: 1 Carácter: 1
+ podman machine init
+ ~~~~~
+ CategoryInfo          : NotSpecified: (Error: podman-m... already exists:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

PS C:\Users\promi>
```

---

## podman machine start

Arranca la máquina virtual creada por podman machine init.

```
PS C:\Users\promi> podman machine start
Starting machine "podman-machine-default"
podman : your 131072x1 screen size is bogus. expect trouble
En linea: 1 Carácter: 1
+ podman machine start
+ ~~~~~
+ CategoryInfo          : NotSpecified: (your 131072x1 s... expect trouble:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

This machine is currently configured in rootless mode. If your containers
require root permissions (e.g. ports < 1024), or if you run into compatibility
issues with non-podman clients, you can switch using the following command:
podman machine set --rootful

API forwarding listening on: npipe://./pipe/podman-machine-default

Another process was listening on the default Docker API pipe address.
You can still connect Docker API clients by setting DOCKER HOST using the
following powershell command in your terminal session:
$Env:DOCKER_HOST = 'npipe://./pipe/podman-machine-default'

Or in a classic CMD prompt:
set DOCKER_HOST=npipe://./pipe/podman-machine-default

Alternatively, terminate the other process and restart podman machine.
Machine "podman-machine-default" started successfully

PS C:\Users\promi> |
```

---

## podman run

En este caso, usamos el comando podman run, igual que en docker, para levantar un contenedor con la imagen [quay.io/podman/hello](https://quay.io/podman/hello) que contiene una especie de prueba que al ejecutarse envía un mensaje “Hello Podman World”.

```
PS C:\Users\promi> podman run quay.io/podman/hello
podman : Trying to pull quay.io/podman/hello:latest...
En linea: 1 Carácter: 1
+ podman run quay.io/podman/hello
+ ~~~~~
+ CategoryInfo          : NotSpecified: (Trying to pull ...hello:latest...:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

Getting image source signatures
Copying blob sha256:81df7ff16254ed9756e27c8de9ceb02a9568228fccadbff080f41cc5eb5118a44
Copying config sha256:5dd467fce50b6951185da365b5feee75409968cbab5767b9b59e325fb2ecbc0
Writing manifest to image destination
!... Hello Podman World ...!

          .--"--
          /  \
          (O) (O) \
        ~~~| -=(Y,)=-| ~~~
        .--. / \ .--. ~
        / \ o o \ / \ / \
        | = (X) = | ~ / (O) (O) \
        ~~~~ ~~~~ | = (Y,) =-| ~~~
                    | U | ~~~

Project:  https://github.com/containers/podman
Website:  https://podman.io
Desktop:  https://podman-desktop.io
Documents:  https://docs.podman.io
YouTube:  https://youtube.com/@Podman
X/Twitter:  @Podman_io
Mastodon:  @Podman_io@fosstodon.org

PS C:\Users\promi>
```

Ahora realizamos otra prueba de podman run, levantamos un contenedor con la imagen ubi8-micro (Red Hat Universal Base Image 8 en versión mínima) y dentro de el ejecutamos el comando date para que nos devuelva la fecha:

```
PS C:\Users\promi> podman run ubi8-micro date
podman : Resolved "ubi8-micro" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
En línea: 1 Carácter: 1
+ podman run ubi8-micro date
+ ~~~~~
+ CategoryInfo          : NotSpecified: (Resolved "ubi8-...hortnames.conf):String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

Trying to pull registry.access.redhat.com/ubi8-micro:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob sha256:495c338ef5358d5a3b8ebd485fd2251fc4b99e4470f468451f703ab94d4c234c
Copying config sha256:00760263234b87f5ecc2f109df917685fd69fcc5b0a1e37d7508459c3869b13c
Writing manifest to image destination
Storing signatures
Sun May 18 23:35:47 UTC 2025

PS C:\Users\promi> |
```

1. **podman run**: Inicia un contenedor.
2. **ubi8-micro**: Es el nombre de la imagen base (una imagen oficial de Red Hat, muy mínima, basada en RHEL 8).
3. **date**: Es el comando que ejecuta dentro del contenedor. Muestra la fecha y hora actual en el sistema del contenedor.

## podman ps

Lista los contenedores en ejecución, igual que en docker, si queremos que nos liste también los detenidos, añadimos -a

## podman machine ls

Lista las máquinas virtuales administradas por podman.

```
PS C:\Users\promi> podman machine ls
NAME          VM TYPE   CREATED      LAST UP      CPUS      MEMORY     DISK SIZE
podman-machine-default*  ws1      17 minutes ago  Currently running  2          2GIB      100GIB
PS C:\Users\promi> podman machine stop
Machine "podman-machine-default" stopped successfully
PS C:\Users\promi>
```

## podman machine stop

Detiene las máquinas virtuales administradas por podman.

## Podman-servidor web

Esto lanzará un contenedor que expone el puerto 80 del contenedor al puerto 8080 de tu host y con una imagen de nginx:

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

Explicación de los parámetros:

- **-d** → Modo "detached", en segundo plano.
- **-p 8080:80** → Mapea el puerto 80 del contenedor al puerto 8080 del host.

- `-name webserver` → Le da un nombre al contenedor.
- `nginx` → Imagen base que se usará. Si no la tienes, se descarga automáticamente.



## Podman logs

Solo funciona con contenedores que hayan sido iniciados con `--detach` (modo en segundo plano) o que estén en ejecución. Muestra los logs del contenedor del cual pasamos el ID.

```
PS C:\Users\promi> podman logs b9a60ee44567
podman : 2025/05/18 23:49:51 [notice] 1#1: using the "epoll" event method
En línea: 1 Carácter: 1
+ podman logs b9a60ee44567
+ ~~~~~
+ CategoryInfo          : NotSpecified: (2025/05/18 23:4...l" event method:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
172.30.83.164 - - [18/May/2025:23:50:35 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36 Edg/136.0.0.0" "-"
172.30.83.164 - - [18/May/2025:23:50:36 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "http://localhost:8080/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36 Edg/136.0.0.0" "-"

2025/05/18 23:49:51 [notice] 1#1: nginx/1.27.5
2025/05/18 23:49:51 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/05/18 23:49:51 [notice] 1#1: OS: Linux 5.15.167.4-microsoft-standard-WSL2
2025/05/18 23:49:51 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/05/18 23:49:51 [notice] 1#1: start worker processes
2025/05/18 23:49:51 [notice] 1#1: start worker process 24
2025/05/18 23:49:51 [notice] 1#1: start worker process 25
2025/05/18 23:49:51 [notice] 1#1: start worker process 26
2025/05/18 23:49:51 [notice] 1#1: start worker process 27
2025/05/18 23:50:36 [error] 26#26: *3 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 172.30.83.164, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "localhost:8080", referer: "http://localhost:8080/"

PS C:\Users\promi> |
```

## podman stop

Podman stop detiene el contenedor que lleva ese ID y podman rm elimina el contenedor que lleva ese ID. Si usamos `podman rm -f <nombre_o_id_contenedor>` borramos el contenedor forzando su cierre.

```
PS C:\Users\promi> podman ps
CONTAINER ID  IMAGE                                     COMMAND           CREATED          STATUS           PORTS
             NAMES
b9a60ee44567  docker.io/library/nginx:latest  nginx -g daemon o...  6 minutes ago  Up 6 minutes (starting)  0.0.0.0:8080
0->80/tcp      webserver

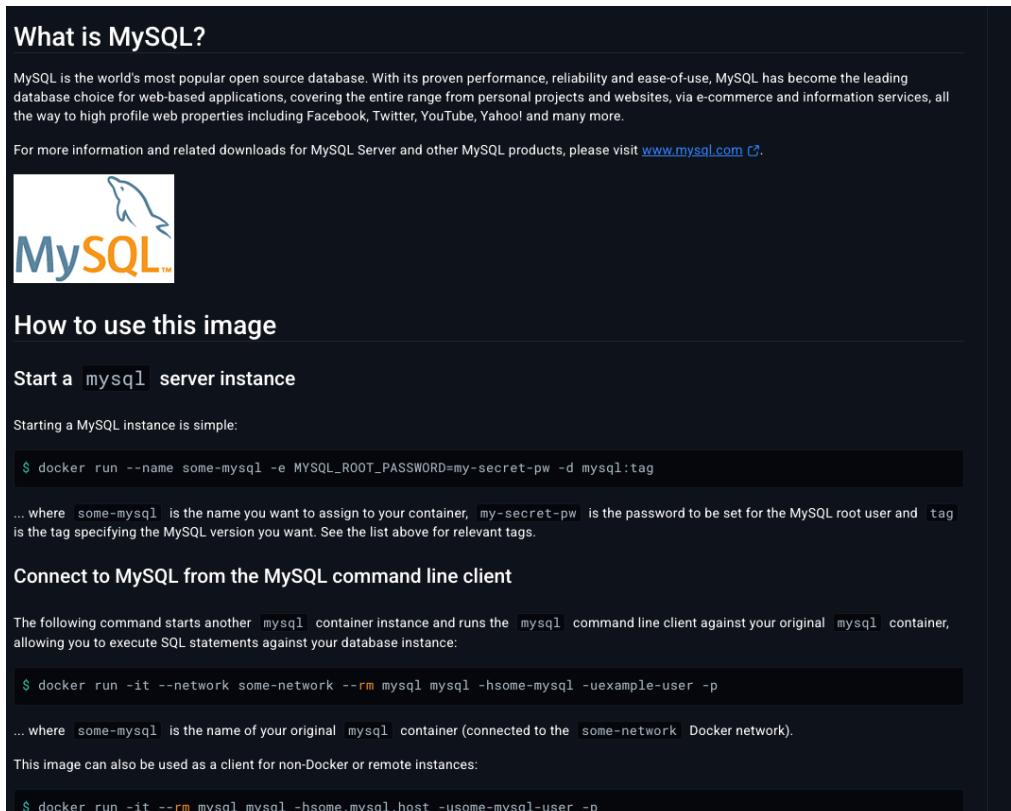
PS C:\Users\promi> podman stop b9a60ee44567
b9a60ee44567

PS C:\Users\promi> podman rm b9a60ee44567
b9a60ee44567

PS C:\Users\promi>
```

## MySQL en Docker

La mayoría de software tiene publicado un contenedor en DockerHub para que lo podamos descargar, en este caso vamos a probar a buscar el de MySQL y descargarlo. En la parte superior encontramos el buscador y escribirmos “MySQL”, una vez realiza la búsqueda, encontramos un apartado sobre como usar una imagen de mysql:



The screenshot shows the "How to use this image" section of the MySQL Docker image page. It includes instructions for starting a MySQL server instance, connecting from the command line, and using it as a client for non-Docker instances.

**How to use this image**

**Start a mysql server instance**

Starting a MySQL instance is simple:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

... where `some-mysql` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MySQL root user and `tag` is the tag specifying the MySQL version you want. See the list above for relevant tags.

**Connect to MySQL from the MySQL command line client**

The following command starts another `mysql` container instance and runs the `mysql` command line client against your original `mysql` container, allowing you to execute SQL statements against your database instance:

```
$ docker run -it --network some-network --rm mysql mysql -hsome-mysql -uexample-user -p
```

... where `some-mysql` is the name of your original `mysql` container (connected to the `some-network` Docker network).

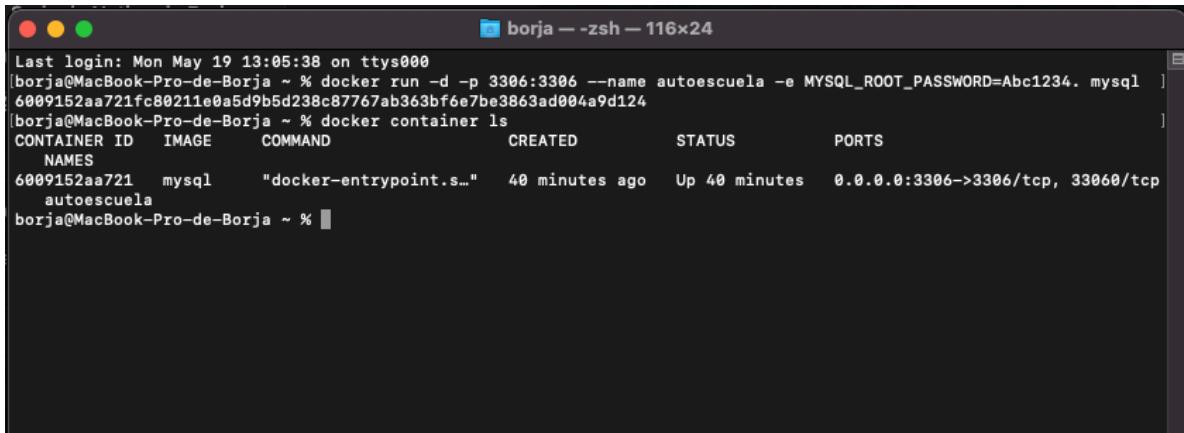
This image can also be used as a client for non-Docker or remote instances:

```
$ docker run -it --rm mysql mysql -hsome.mysql.host -usome-mysql-user -p
```

Seguimos los pasos que nos recomienda la propia web para levantar un contenedor con mysql:

```
docker run -d -p 3306:3306 --name autoescuela -e MYSQL_ROOT_PASSWORD=Abc1234. mysql
```

- `-d`: detach (se ejecuta en segundo plano)
- `-p`: port (puerto)
- `--name`: nombre del contenedor
- `-e`: variables de entorno.
- nombre de la imagen (mysql) al no poner versión se descarga la última.



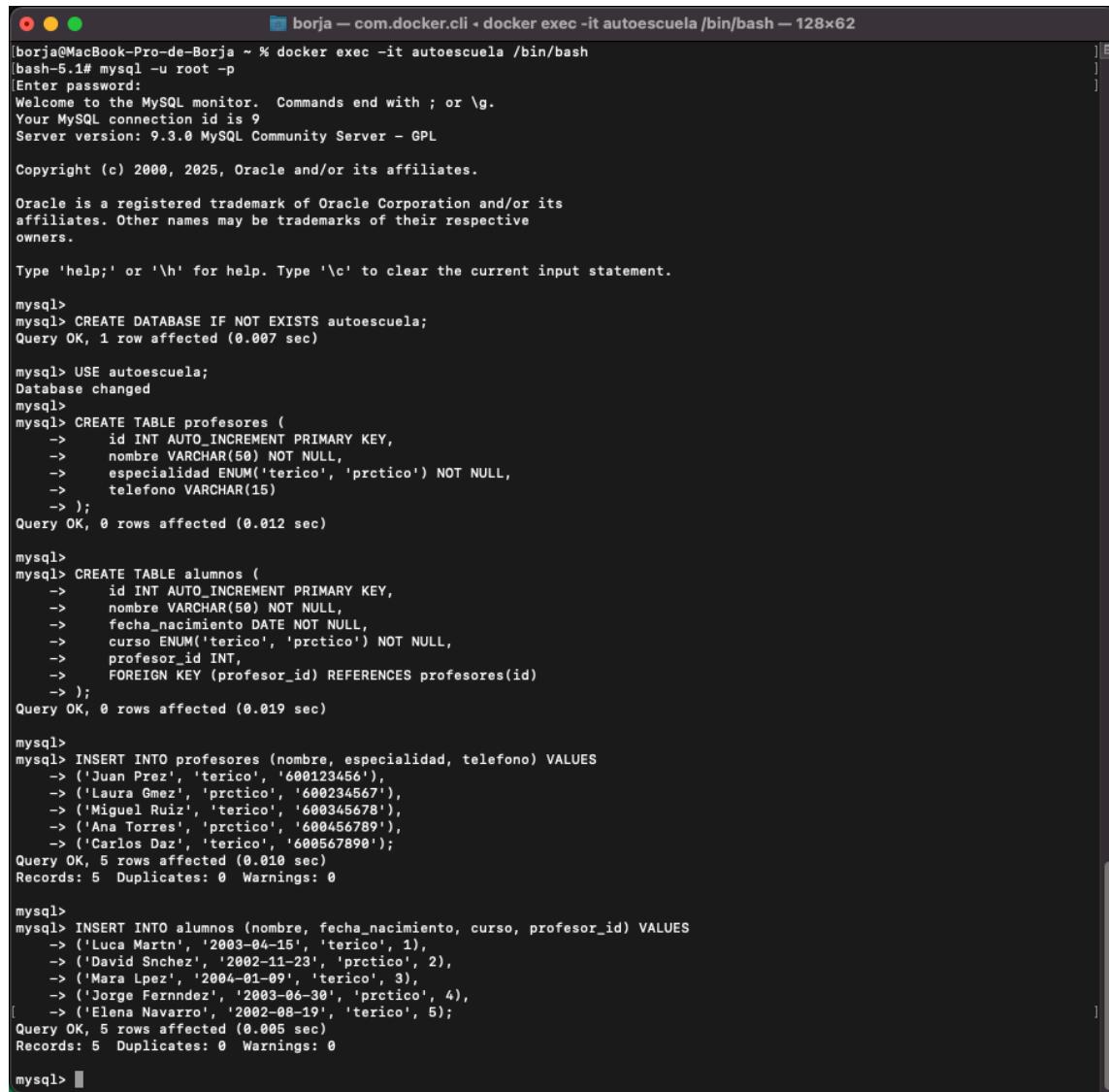
```
Last login: Mon May 19 13:05:38 on ttys000
[borja@MacBook-Pro-de-Borja ~ % docker run -d -p 3306:3306 --name autoescuela -e MYSQL_ROOT_PASSWORD=Abc1234. mysql
6009152aa721fc80211e0a5d9b5d238c87767ab363bf6e7be3863ad004a9d124
[borja@MacBook-Pro-de-Borja ~ % docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
6009152aa721 mysql "docker-entrypoint.s..." 40 minutes ago Up 40 minutes 0.0.0.0:3306->3306/tcp, 33060/tcp
autoescuela
[borja@MacBook-Pro-de-Borja ~ %
```

Para acceder al contenedor y abrir la terminal ejecutamos **docker exec**:

```
docker exec -it autoescuela /bin/bash
```

Desde la bash, entramos en mysql y trabajamos creando una base de datos.

```
mysql -u root -p
```



```
[borja@MacBook-Pro-de-Borja ~ % docker exec -it autoescuela /bin/bash
[bash-5.1# mysql -u root -p
[Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql> CREATE DATABASE IF NOT EXISTS autoescuela;
Query OK, 1 row affected (0.007 sec)

mysql> USE autoescuela;
Database changed
mysql>
mysql> CREATE TABLE profesores (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(50) NOT NULL,
    -> especialidad ENUM('terico', 'prctico') NOT NULL,
    -> telefono VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.012 sec)

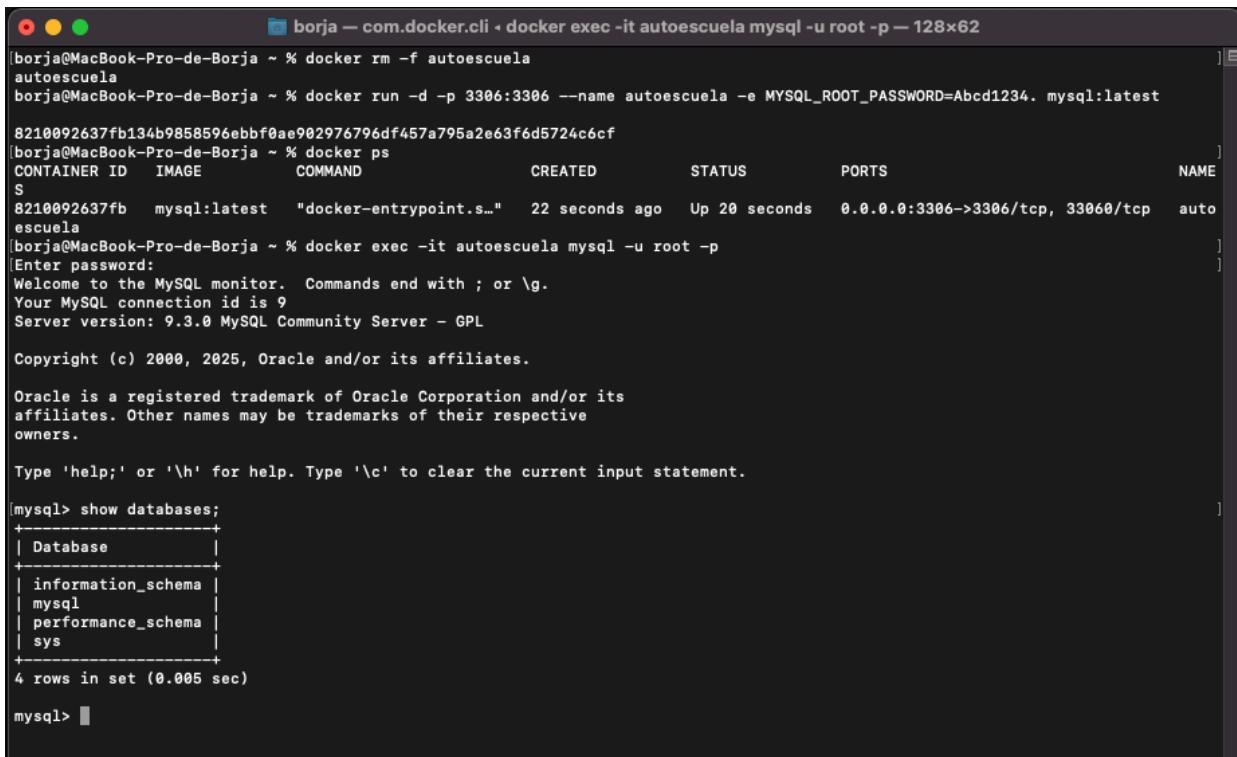
mysql>
mysql> CREATE TABLE alumnos (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(50) NOT NULL,
    -> fecha_nacimiento DATE NOT NULL,
    -> curso ENUM('terico', 'prctico') NOT NULL,
    -> profesor_id INT,
    -> FOREIGN KEY (profesor_id) REFERENCES profesores(id)
    -> );
Query OK, 0 rows affected (0.019 sec)

mysql>
mysql> INSERT INTO profesores (nombre, especialidad, telefono) VALUES
    -> ('Juan Prez', 'terico', '600123456'),
    -> ('Laura Gmez', 'prctico', '600234567'),
    -> ('Miguel Ruiz', 'terico', '600345678'),
    -> ('Ana Torres', 'prctico', '600456789'),
    -> ('Carlos Daz', 'terico', '600567890');
Query OK, 5 rows affected (0.010 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO alumnos (nombre, fecha_nacimiento, curso, profesor_id) VALUES
    -> ('Luca Martn', '2003-04-15', 'terico', 1),
    -> ('David Snchez', '2002-11-23', 'prctico', 2),
    -> ('Mara Lpez', '2004-01-09', 'terico', 3),
    -> ('Jorge Fernndez', '2003-06-30', 'prctico', 4),
    -> ('Elena Navarro', '2002-08-19', 'terico', 5);
Query OK, 5 rows affected (0.005 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
```

Como podemos ver, si salimos, borramos el contenedor y lo volvemos a levantar...los datos no se mantienen porque no hay persistencia de datos.



```

borja@MacBook-Pro-de-Borja ~ % docker rm -f autoescuela
autoescuela
borja@MacBook-Pro-de-Borja ~ % docker run -d -p 3306:3306 --name autoescuela -e MYSQL_ROOT_PASSWORD=Abcd1234. mysql:latest
8210092637fb134b9858596ebbf0ae902976796df457a795a2e63f6d5724c6cf
[borja@MacBook-Pro-de-Borja ~ % docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAME
8210092637fb mysql:latest "docker-entrypoint.s..." 22 seconds ago Up 20 seconds 0.0.0.0:3306->3306/tcp, 33060/tcp autoescuela
[borja@MacBook-Pro-de-Borja ~ % docker exec -it autoescuela mysql -u root -p
[Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.005 sec)

mysql> 

```

## docker volume

Para hacer que los datos se mantengan, tenemos varias opciones, una de ellas es crear un volumen de datos gestionado por docker (es como tener un disco duro extraible, si formateamos el disco principal de nuestro ordenador, los datos se mantienen en el externo).

Para ello, creamos el volumen

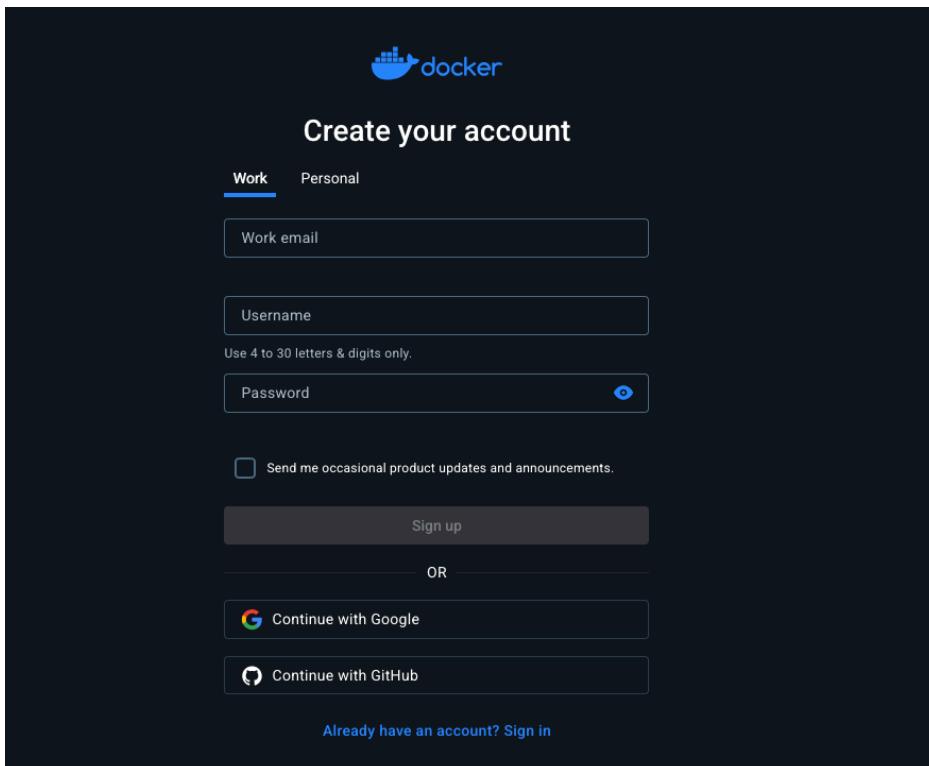
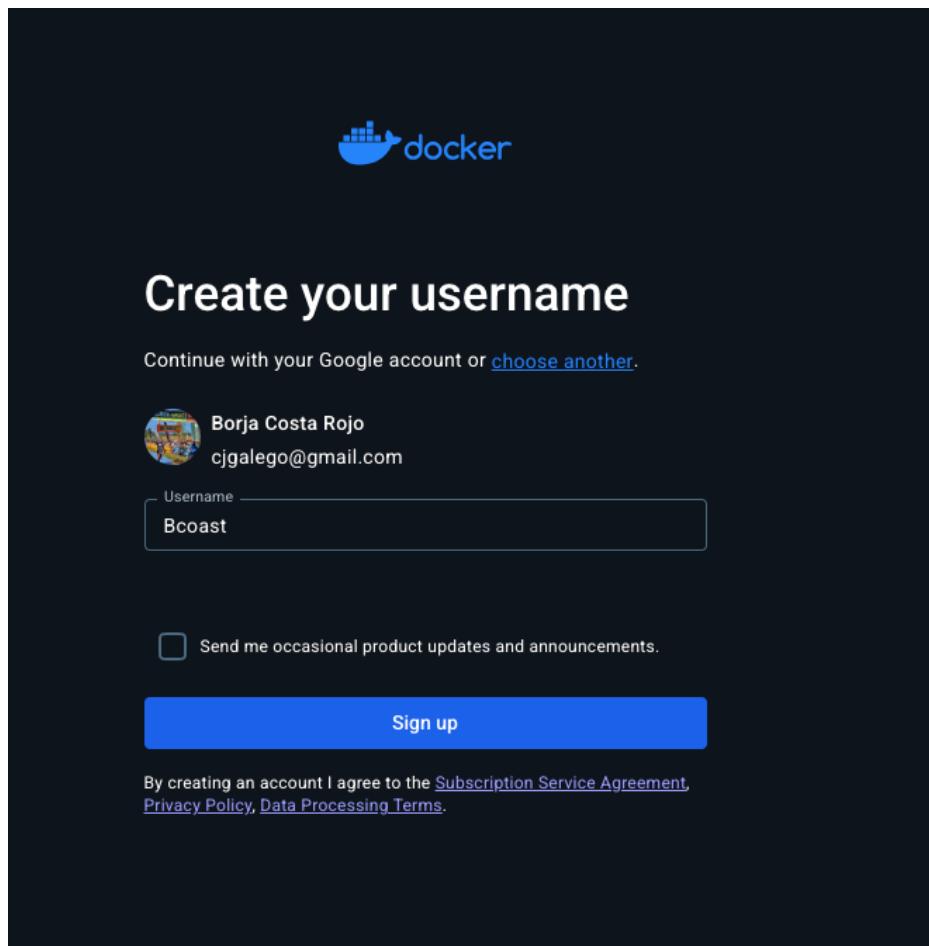
```
docker volume create autoescuela_data
```

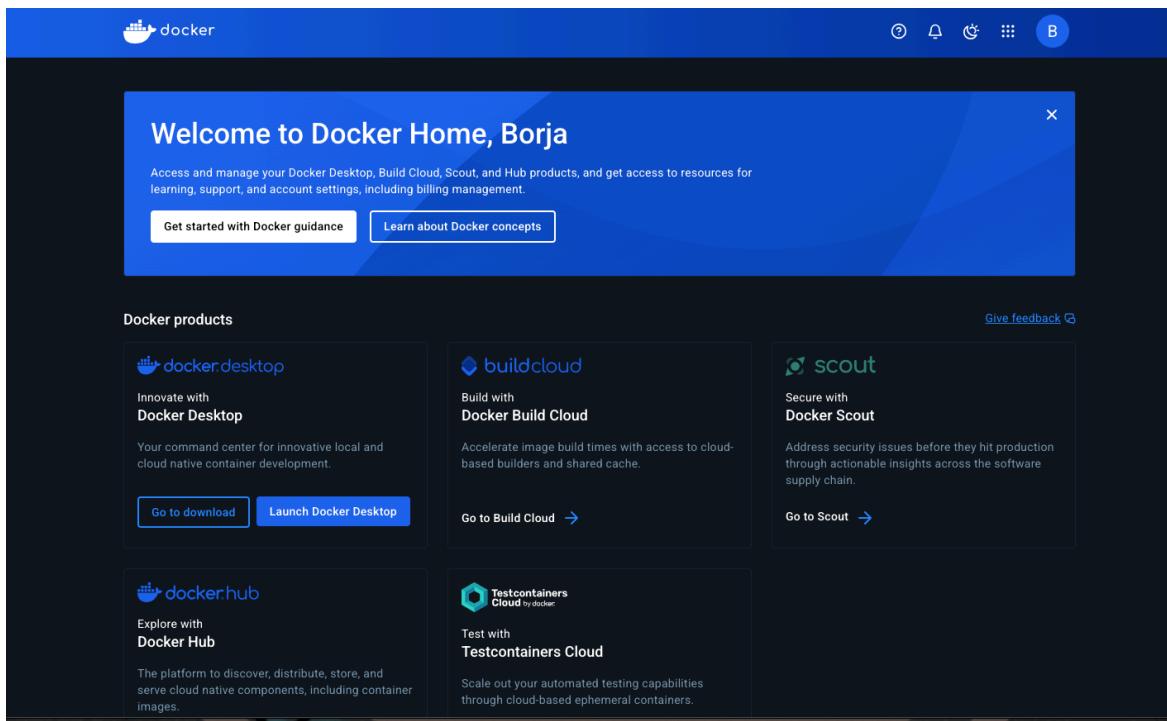
Mismo caso que explicamos en [persistencia de datos](#)

## Docker Hub

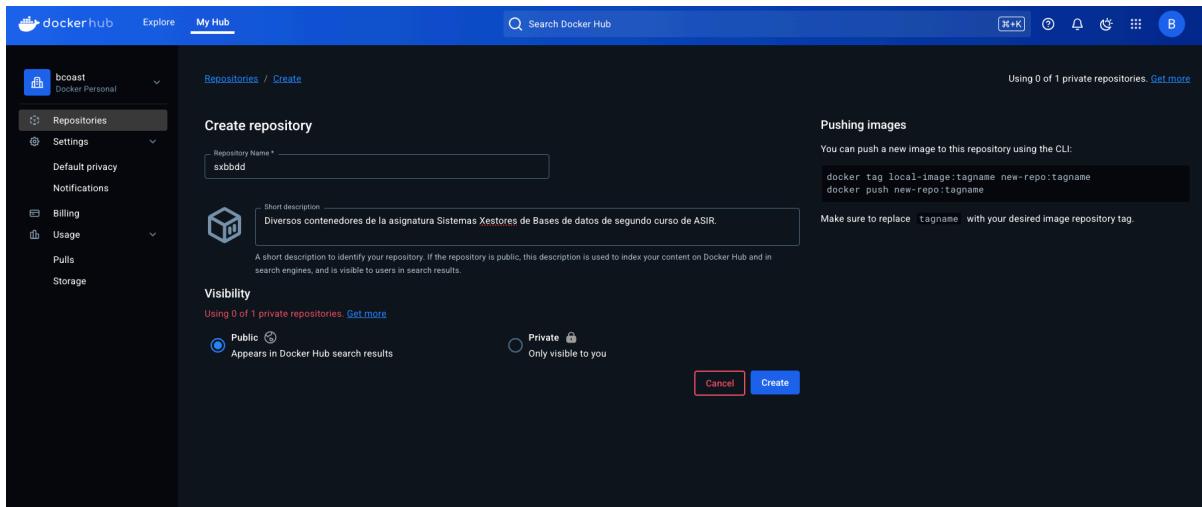
Dockerhub es un repositorio en la nube, tiene una estructura muy similar a Github y nos permite descargar imágenes listas para usar de multitud de aplicaciones (Ubuntu, MySQL, WordPress...). Además nos permitirá también tener nuestro propio repositorio de imágenes.

Accedemos a la web principal de Docker Hub: <https://hub.docker.com/> y en nuestro caso hemos optado por registrarnos con el correo de Gmail.





Ahora vamos a crear un repositorio:



## Usar nuestras propias imágenes

```

borjacosta ➤ □ ~ ○ ⟲master # ✘ 93 ➤ docker run -d -p 33060:3306 --name mysql-db -e MYSQL_ROOT_PASSWORD=Abcd1234. --mount src=mysql-db-data,dst=/var/lib/mysql mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
79f239a40e62: Pull complete
cb8acbf2440c: Pull complete
b2ead3e96e6b: Pull complete
548990e33276: Pull complete
cea172a6e83b: Pull complete
c11056354384: Pull complete
49978e7ccddf: Pull complete
daac2c594bdd: Pull complete
769c3ac51f88: Pull complete
fae51f7de1fb: Pull complete
Digest: sha256:0596fa224cdf3b3355ce3ddbf7ce77be27ec9e51841dfc5d2e1c8b81eea69d2
Status: Downloaded newer image for mysql:latest
907beda5d44ede63e344ccab85ca84d63e2addded4f3bc161893bf8f8399f13a
borjacosta ➤ □ ~ ○ ⟲master # ✘ 93 ➤ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
907beda5d44e mysql "docker-entrypoint.s..." About a minute ago Up About a minute 33060/tcp, 0.0.0.0:33060→3306/tcp mysql-db
borjacosta ➤ □ ~ ○ ⟲master # ✘ 93 ➤ docker exec -it mysql-mio mysql -p
Error response from daemon: No such container: mysql-mio
borjacosta ➤ □ ~ ○ ⟲master # ✘ 93 ➤ docker exec -it mysql-db mysql -p
in zsh at 19:08:06
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE FWIRTZ
      → ;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| FWIRTZ   |
| information_schema |
| mysql     |
| performance_schema |
| sys       |
+-----+
5 rows in set (0.01 sec)

mysql>

```

```

zsh in borjacosta
borjacosta ➤ □ ~ ○ ⟲master # ✘ 94 ➤ docker login --username bcoast
in zsh at 15:34:33

Info → A Personal Access Token (PAT) can be used instead.
To create a PAT, visit https://app.docker.com/settings

Password:
Login Succeeded
borjacosta ➤ □ ~ ○ ⟲master # ✘ 94 ➤ docker tag mysqlwirtz bcoast/sxbbdd:latest
in zsh at 15:34:48
borjacosta ➤ □ ~ ○ ⟲master # ✘ 94 ➤ docker push bcoast/sxbbdd:latest
in zsh at 15:39:41

The push refers to repository [docker.io/bcoast/sxbbdd]
c11056354384: Mounted from library/mysql
548990e33276: Mounted from library/mysql
769c3ac51f88: Mounted from library/mysql
b2ead3e96e6b: Mounted from library/mysql
daac2c594bdd: Mounted from library/mysql
cea172a6e83b: Mounted from library/mysql
cb8acbf2440c: Mounted from library/mysql
ea1dcbe2a6d8: Pushed
fae51f7de1fb: Mounted from library/mysql
79f239a40e62: Mounted from library/mysql
49978e7ccddf: Mounted from library/mysql
latest: digest: sha256:164b94c97d8f124a9a8ec92cc6dd6be6bc6607c53a6de57cb98c9a2a6c0b600b size: 2692
borjacosta ➤ □ ~ ○ ⟲master # ✘ 94 ➤
in zsh at 15:40:25

```

## Docker registry

Registry es una imagen que funciona como nuestro propio servidor de imágenes. Aunque dockerhub funciona muy bien, regirty nos permite no depender de la web o de dockerhub. Es sobretodo útil para empresas, para poder tener en una red local un servidor de imágenes.

Lo primero que debemos hacer es descargar la imagen de registry:

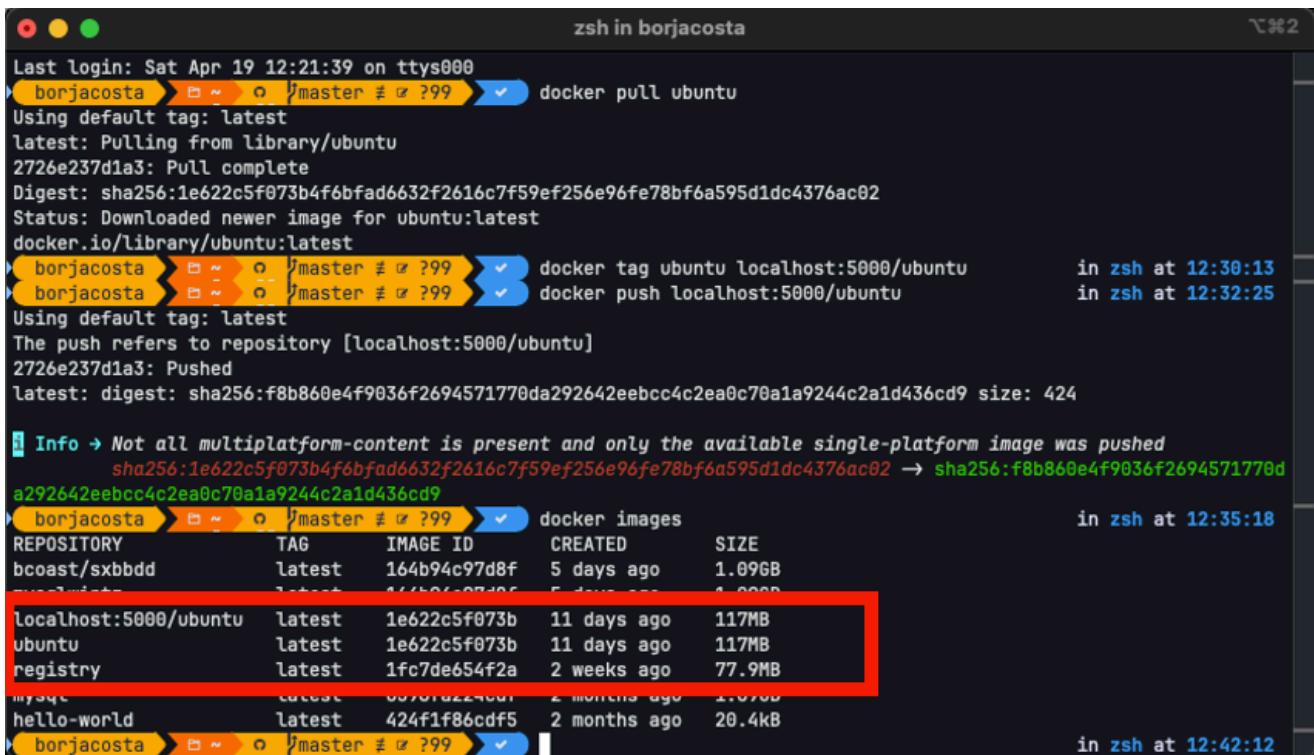
```
docker run -d -p 5000:5000 --name registry registry
```

Ahora abrimos otra terminal y traemos una imagen de Ubuntu:

```
docker pull ubuntu
```

Ahora le asignamos una etiqueta y hacemos un push al resgistro creado para subirlo

\*\*El warning lo que nos dice es que hay varias versiones dependiendo de la plataforma usada y que solo se subirá la imagen para la plataforma usada.

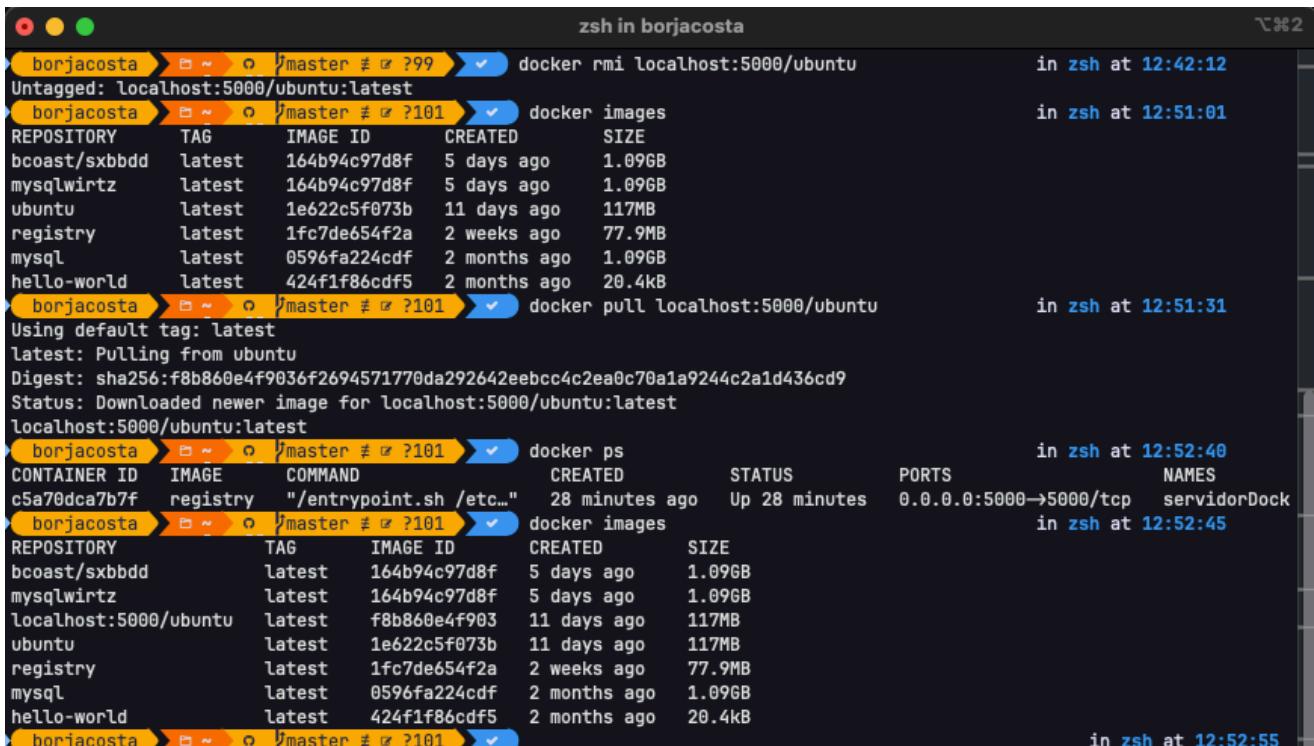


```

zsh in borjacosta
Last login: Sat Apr 19 12:21:39 on ttys000
borjacosta ▶ ~ o /master # ✘ ?99 docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
2726e237d1a3: Pull complete
Digest: sha256:1e622c5f073b4f6bfad6632f2616c7f59ef256e96fe78bf6a595d1dc4376ac02
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
borjacosta ▶ ~ o /master # ✘ ?99 docker tag ubuntu localhost:5000/ubuntu
borjacosta ▶ ~ o /master # ✘ ?99 docker push localhost:5000/ubuntu
Using default tag: latest
The push refers to repository [localhost:5000/ubuntu]
2726e237d1a3: Pushed
latest: digest: sha256:f8b860e4f9036f2694571770da292642eebcc4c2ea0c70a1a9244c2a1d436cd9 size: 424
Info → Not all multiplatform-content is present and only the available single-platform image was pushed
sha256:1e622c5f073b4f6bfad6632f2616c7f59ef256e96fe78bf6a595d1dc4376ac02 → sha256:f8b860e4f9036f2694571770da292642eebcc4c2ea0c70a1a9244c2a1d436cd9
borjacosta ▶ ~ o /master # ✘ ?99 docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
bcoast/sxbbdd latest 164b94c97d8f 5 days ago 1.09GB
mysqlwirtz latest 164b94c97d8f 5 days ago 1.09GB
localhost:5000/ubuntu latest 1e622c5f073b 11 days ago 117MB
ubuntu latest 1e622c5f073b 11 days ago 117MB
registry latest 1fc7de654f2a 2 weeks ago 77.9MB
mysql latest 0596fa224cdf 2 months ago 1.09GB
hello-world latest 424f1f86cdf5 2 months ago 20.4kB
borjacosta ▶ ~ o /master # ✘ ?99
in zsh at 12:35:18
in zsh at 12:32:25
in zsh at 12:30:13
in zsh at 12:42:12

```

Ahora para comprobar que funciona, borramos del repositorio local la imagen, comprobamos que se ha borrado, hacemos un pull para recuperarla del registry y volvemos a comprobar que tenemos de nuevo la imagen.



```

zsh in borjacosta
borjacosta ▶ ~ o /master # ✘ ?99 docker rmi localhost:5000/ubuntu
Untagged: localhost:5000/ubuntu:latest
borjacosta ▶ ~ o /master # ✘ ?101 docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
bcoast/sxbbdd latest 164b94c97d8f 5 days ago 1.09GB
mysqlwirtz latest 164b94c97d8f 5 days ago 1.09GB
ubuntu latest 1e622c5f073b 11 days ago 117MB
registry latest 1fc7de654f2a 2 weeks ago 77.9MB
mysql latest 0596fa224cdf 2 months ago 1.09GB
hello-world latest 424f1f86cdf5 2 months ago 20.4kB
borjacosta ▶ ~ o /master # ✘ ?101 docker pull localhost:5000/ubuntu
Using default tag: latest
latest: Pulling from ubuntu
Digest: sha256:f8b860e4f9036f2694571770da292642eebcc4c2ea0c70a1a9244c2a1d436cd9
Status: Downloaded newer image for localhost:5000/ubuntu:latest
localhost:5000/ubuntu:latest
borjacosta ▶ ~ o /master # ✘ ?101 docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c5a70dca7b7f registry "/entrypoint.sh /etc..." 28 minutes ago Up 28 minutes 0.0.0.0:5000→5000/tcp servidorDock
borjacosta ▶ ~ o /master # ✘ ?101 docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
bcoast/sxbbdd latest 164b94c97d8f 5 days ago 1.09GB
mysqlwirtz latest 164b94c97d8f 5 days ago 1.09GB
localhost:5000/ubuntu latest f8b860e4f903 11 days ago 117MB
ubuntu latest 1e622c5f073b 11 days ago 117MB
registry latest 1fc7de654f2a 2 weeks ago 77.9MB
mysql latest 0596fa224cdf 2 months ago 1.09GB
hello-world latest 424f1f86cdf5 2 months ago 20.4kB
borjacosta ▶ ~ o /master # ✘ ?101
in zsh at 12:42:12
in zsh at 12:51:01
in zsh at 12:51:31
in zsh at 12:52:40
in zsh at 12:52:45
in zsh at 12:52:55

```

para verlo más claro:

```

docker pull ubuntu (Docker Hub)
docker pull localhost:5000/ubuntu (Registry local)

```

## Principales comandos

Comando Docker	Descripción
docker login	Inicia sesión en Docker Hub con tus credenciales
docker logout	Cierra sesión de Docker Hub
docker search <nombre>	Busca imágenes públicas en Docker Hub
docker pull <imagen>	Descarga una imagen desde Docker Hub
docker push <usuario>/<imagen>	Sube una imagen al repositorio del usuario en Docker Hub
docker tag <img> <usuario>/<img>	Etiqueta una imagen local para poder subirla al Docker Hub
docker images	Lista todas las imágenes locales
docker rmi <imagen>	Elimina una imagen local

## Enlazando dos contenedores

Tenemos tres formas de enlazar contenedores:

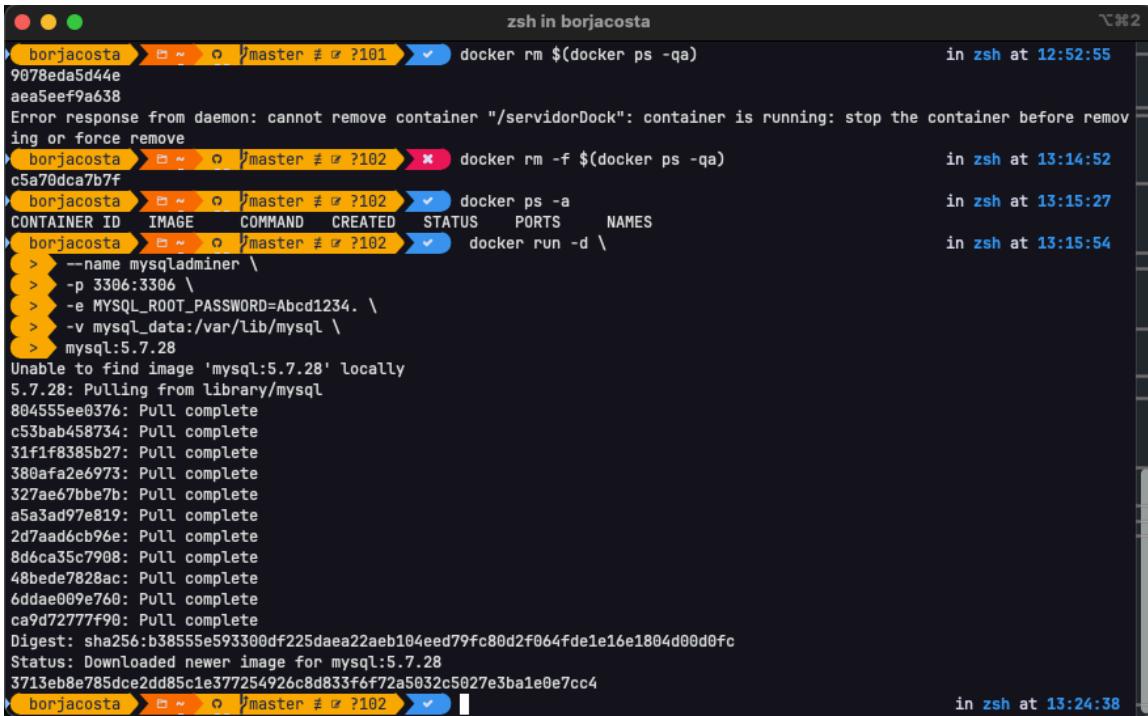
- Usando —link
- Usando red personalizada (network)
- Usando Docker Compose

### Usar --link (obsoleto pero aún funciona)

En este ejemplo vamos a enlazar un contenedor con MySQL con un contenedor con Adminer.

Lo primero que haremos es crear un contenedor con MySQL Server:

```
docker run -d \
--name mysqladminer \
-p 3306:3306 \
MYSQL_ROOT_PASSWORD=Abcd1234. \
-v mysql_data:/var/lib/mysql \
mysql:5.7.28
```



```

zsh in borjacosta
borjacosta ➜ master # docker rm $(docker ps -qa)
9078edda5d44e
aea5eef9a638
Error response from daemon: cannot remove container "/servidorDock": container is running: stop the container before removing or force remove
borjacosta ➜ master # docker rm -f $(docker ps -qa)
in zsh at 13:14:52
c5a70dca7b7f
borjacosta ➜ master # docker ps -a
in zsh at 13:15:27
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
borjacosta ➜ master # docker run -d \
in zsh at 13:15:54
  --name mysqladminer \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=Abcd1234. \
  -v mysql_data:/var/lib/mysql \
  mysql:5.7.28
Unable to find image 'mysql:5.7.28' locally
5.7.28: Pulling from library/mysql
804555ee0376: Pull complete
c53bab458734: Pull complete
31f1f8385b27: Pull complete
380afa2e6973: Pull complete
327aae67bbe7b: Pull complete
a5a3ad97e819: Pull complete
2d7aadcb96e: Pull complete
8d6ca35c7908: Pull complete
48bede7828ac: Pull complete
6ddae009e760: Pull complete
ca9d72777f90: Pull complete
Digest: sha256:b38555e593300df225daea22aeb104eed79fc80d2f064fde1e16e1804d00d0fc
Status: Downloaded newer image for mysql:5.7.28
3713eb8e785dce2dd85c1e377254926c8d833f6f72a5032c5027e3ba1e0e7cc4
borjacosta ➜ master # docker ps -a
in zsh at 13:24:38

```

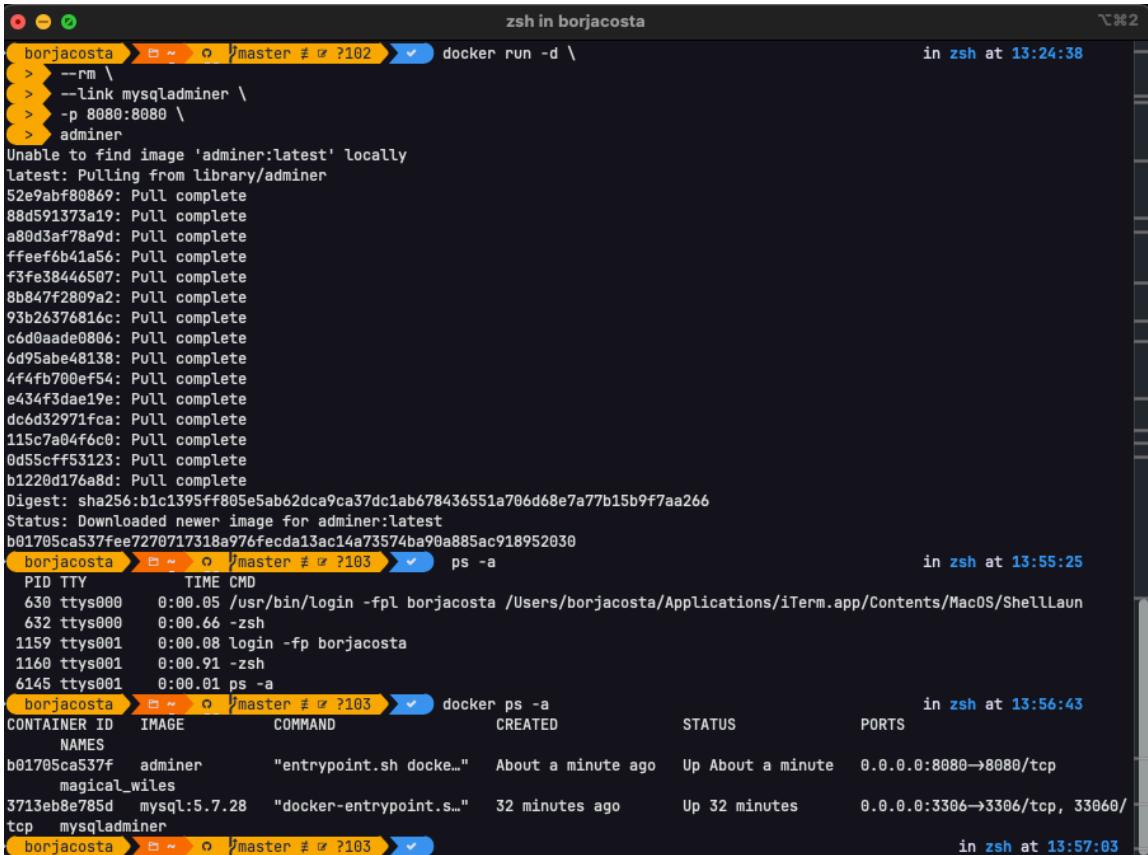
Ahora crearemos el contenedor con Adminer:

```

docker run -d \
--rm \
--link mysqladminer \
-p 8080:8080 \
adminer

```

Comprobamos:



```

zsh in borjacosta
borjacosta ➜ master # docker run -d \
in zsh at 13:24:38
  --rm \
  --link mysqladminer \
  -p 8080:8080 \
  adminer
Unable to find image 'adminer:latest' locally
latest: Pulling from library/adminer
52e9abf80869: Pull complete
88d591373a19: Pull complete
a80d3af78a9d: Pull complete
ffef6b41a56: Pull complete
f3fe38446507: Pull complete
8b847f2809a2: Pull complete
93b26376816c: Pull complete
c6d0aade8806: Pull complete
6d95abe48138: Pull complete
4f4fb790ef54: Pull complete
e434f3dae19e: Pull complete
dc4d32971fca: Pull complete
115c7a04f6c0: Pull complete
0d55cff53123: Pull complete
b1220d176a8d: Pull complete
Digest: sha256:b1c1395ff805e5ab62dca9ca37dc1ab678436551a706d68e7a77b15b9f7aa266
Status: Downloaded newer image for adminer:latest
b01705ca537fee7270717318a976fecda13ac14a73574ba90a885ac918952030
borjacosta ➜ master # ps -a
in zsh at 13:55:25
  PID TTY      TIME CMD
 630 ttys000  0:00.05 /usr/bin/login -fpl borjacosta /Users/borjacosta/Applications/iTerm.app/Contents/MacOS/ShellLaun
 632 ttys000  0:00.66 -zsh
1159 ttys001  0:00.08 login -fp borjacosta
1160 ttys001  0:00.91 -zsh
6145 ttys001  0:00.01 ps -a
borjacosta ➜ master # docker ps -a
in zsh at 13:56:43
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
  NAMES
b01705ca537f        adminer            "entrypoint.sh dock..."   About a minute ago   Up About a minute   0.0.0.0:8080→8080/tcp
  magical_wiles
3713eb8e785d        mysql:5.7.28      "docker-entrypoint.s..."  32 minutes ago     Up 32 minutes      0.0.0.0:3306→3306/tcp, 33060/tcp
  mysqladminer
borjacosta ➜ master #
in zsh at 13:57:03

```

Para comprobarlo, accedemos al navegador e intentamos acceder a [localhost:8080](http://localhost:8080) que es el puerto por el que escucha adminer, mediante adminer nos intentamos conectar a mysqladminer:

## Usando red personalizada (network)

Lo primero que debemos hacer es crear una red con la que relacionaremos los contenedores:

```
docker network create my-net
```

Una vez creada la red, creamos los contenedores:

```
docker run -d \
--rm \
--name mysqlc \
--network my-net \
-p 3306:3306 \
-v mysql_data:/var/lib/mysql \
mysql:5.7.28
docker run -d \
--rm \
--network my-net \
-p 8080:8080 \
adminer
```

```

zsh in borjacosta
borjacosta ➜ ~ ⚡ master # ✘ ?106 ➜ docker network create my-net
f0199f9a3c7445a9326a641cd1b8ca79d7f124039990048d70afdd3d3c993edd
in zsh at 14:23:28
borjacosta ➜ ~ ⚡ master # ✘ ?106 ➜ docker run -d \
> --rm \
> --name mysqlc \
> --network my-net \
> -p 3306:3306 \
> -v mysql_data:/var/lib/mysql \
> mysql:5.7.28
1f0a1190b3a56c931a0980304659a10d3789620eedd48df4c225ada32bd6283f
in zsh at 14:23:47
borjacosta ➜ ~ ⚡ master # ✘ ?106 ➜ docker run -d \
> --rm \
> --network my-net \
> -p 8080:8080 \
> adminer
1a683677a5dd92d548976bbf9d9d8902c81c62ae5350e4bf871ff3d846f46ee6
in zsh at 15:38:21
borjacosta ➜ ~ ⚡ master # ✘ ?106 ➜ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
1a683677a5dd adminer "entrypoint.sh docke..." About a minute ago Up About a minute 0.0.0.0:8080→8080/tcp
pensive_cannon
1f0a1190b3a5 mysql:5.7.28 "docker-entrypoint.s..." 6 minutes ago Up 6 minutes 0.0.0.0:3306→3306/tcp, 33060/tcp
mysqlc
in zsh at 15:43:33
borjacosta ➜ ~ ⚡ master # ✘ ?106 ➜
in zsh at 15:44:41

```

Comprobamos de nuevo desde el navegador:

## Enlazando tres contenedores

Vamos a probar otro ejemplo donde usaremos un contenedor con MySQL, otro con phpmyadmin y otro con el servidor web.

Como siempre, comenzamos por crear la conexión

```
docker network create my_network
```

Creamos una carpeta con el archivo index:

```
mkdir ~/my_web
cd ~/my_web
```

Editamos el archivo con un mensaje:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Todo Listo!</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background: linear-gradient(to right, #74ebd5, #acb6e5);
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .mensaje {
            background-color: white;
            padding: 40px 60px;
            border-radius: 12px;
            box-shadow: 0 10px 20px rgba(0,0,0,0.2);
            text-align: center;
        }
        .mensaje h1 {
            color: #2c3e50;
            font-size: 24px;
        }
    </style>
</head>
<body>
    <div class="mensaje">
        <h1>Está todo listo para aprobar la asignatura<br>
        <strong>Sistemas Gestores de Bases de Datos</strong><br>
        que imparte el profesor <strong>Manuel Corbelle</strong>.</h1>
    </div>
</body>
</html>
```

Creamos el contenedor con MySQL:

```
docker run -d \
--rm \
--name mysql-container \
--network my-net \
-p 3306:3306 \
-v mysql_data:/Users/borjacosta/Documents \
-e MYSQL_ROOT_PASSWORD=Abcd1234. \
-e MYSQL_DATABASE=wirtzdb \
-e MYSQL_USER=admin \
-e MYSQL_PASSWORD=Abcd1234. \
mysql:5.7.28
```

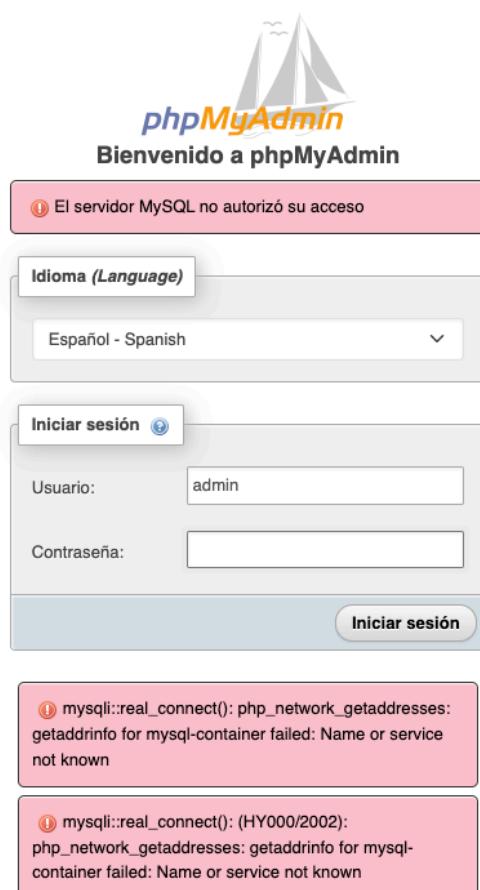
Creamos el contenedor con phpmyadmin, en este caso especificamos con “-e” el nombre y el puerto del contenedor mysql al que debe conectarse (si no lo hacemos puede que phpmyadmin intente buscar dentro de su propio contenedor y no funcione):

```
docker run -d \
--name phpmyadmin-container \
--network my_network \
-p 8080:80 \
-e PMA_HOST=mysql-container \
-e PMA_USER=admin \
-e PMA_PASSWORD=Abcd1234. \
phpmyadmin/phpmyadmin
```

Ahora creamos el contenedor con un servidor web:

```
docker run -d \
--name web-container \
--network my_network \
-p 80:80 \
-v ~/my_web:/var/www/html \
php:7.4-apache
```

Al comprobarlo e intentar acceder a phpmyadmin desde [localhost:8080](http://localhost:8080) nos daba error:



El error nos decía que no era capaz de resolver el nombre del host mysql-container. No nos dimos cuenta de que no estaba conectado a la red. Esto lo solucionamos conectándolo una vez hemos visto el error:

```
docker network connect my_network mysql-container
```

Averigua por qué.'"/&gt;

Ahora vamos a comprobar el acceso al servidor web:

## Dockerfile

Un Dockerfile es un archivo de texto con instrucciones para construir una imagen de Docker. Define paso a paso cómo debe configurarse un contenedor y nos permite crear nuestras propias imágenes. Los pasos lógicos serían:

**Construir Dockerfile → Docker build → Docker run**

Con Docker build se construye la imagen con las instrucciones del dockerfile y con docker run se arranca el contenedor.

Con Docker exec ejecutamos sentencias dentro del contenedor.

## Palabras reservadas

**FROM**: define la imagen base a partir de la cual se construirá la nueva imagen

**RUN**: ejecuta comandos en la imagen durante el proceso de construcción

**CMD**: define el comando por defecto que se ejecuta al iniciar un contenedor

**LABEL**: añade metadatos a la imagen (por ejemplo, autor, descripción)

**EXPOSE**: informa sobre los puertos que usará el contenedor en tiempo de ejecución

**ENV**: define variables de entorno en la imagen

**ADD**: copia archivos o directorios al contenedor y permite descomprimir archivos automáticamente

**COPY**: copia archivos o directorios desde el sistema host al contenedor (no descomprime)

**ENTRYPOINT**: define el ejecutable principal del contenedor, combinable con CMD

**VOLUME**: crea un punto de montaje para datos persistentes compartidos entre host y contenedor

**WORKDIR**: establece el directorio de trabajo para las siguientes instrucciones

**ARG**: define variables de entorno que solo existen durante la construcción de la imagen

**USER**: establece el usuario que ejecutará las instrucciones siguientes

**ONBUILD**: define instrucciones que se ejecutarán cuando otra imagen use esta como base

**STOP SIGNAL**: especifica la señal que se enviará para detener el contenedor correctamente

**HEALTHCHECK**: define cómo comprobar si el contenedor sigue funcionando correctamente

**SHELL**: permite especificar el intérprete de comandos por defecto para RUN, CMD y ENTRYPOINT

## Ejercicio dockerfile autoescuela

Vamos a crear un dockerfile que cree un contenedor con una imagen sql que genere una base de datos de nuestra autoescuela con una tabla profesores. Para ello, primero generamos el archivo dockerfile (recordamos que este archivo no debe tener extensión). Para no cometer errores, crearemos una carpeta para este proyecto, ya que todos los archivos deben estar en la misma carpeta para que no haya errores.

Primero creamos el dockerfile, luego, si queremos añadir setencias que se ejecuten al crear la imagen, creamos un archivo init.sql y por ultimo creamos la imagen con build.

```
FROM mysql:8.0

ENV MYSQL_ROOT_PASSWORD=Abcd1234.
ENV MYSQL_DATABASE=Autoescuela
ENV MYSQL_USER=borja
ENV MYSQL_PASSWORD=Abcd1234.

COPY init.sql /docker-entrypoint-initdb.d/init.sql
```

Ahora que tenemos el dockerfile, creamos el archivo init.sql

```

CREATE TABLE Profesores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    apellido VARCHAR(100),
    dni VARCHAR(20),
    especialidad VARCHAR(50)
);

INSERT INTO Profesores (nombre, apellido, dni, especialidad) VALUES
('Ana', 'López', '12345678A', 'Teórica'),
('Luis', 'Pérez', '23456789B', 'Práctica'),
('María', 'Gómez', '34567890C', 'Teórica'),
('Carlos', 'Ruiz', '45678901D', 'Práctica'),
('Laura', 'Santos', '56789012E', 'Teórica');

```

Ahora ya solo nos queda crear la imagen:

```
docker build -t autoescuela-sql .
```

El punto . al final indica que se busca el archivo Dockerfile **en el directorio actual**, y Docker lo usa por defecto **sin necesidad de nombrarlo**.

Solo si el archivo se llama distinto (por ejemplo, Dockerfile.mysql) o está en otra ruta, se necesita especificarlo:

```
docker run --name autoescuela-db -d autoescuela-sql
```

Ejecutamos el contenedor:

```
docker run --name autoescuela-db -e MYSQL_ROOT_PASSWORD=Abcd1234. -d
autoescuela-sql
```

Y ahora vamos a comprobar que todo ha funcionado correctamente:

```
exec -it autoescuela-db mysql -u root -p
```

```

autoescuela — com.docker.cli - docker exec -it autoescuela-db mysql -u root -p — 140x34
borja@MacBook-Pro-de-Borja autoescuela % docker exec -it autoescuela-db mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.42 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE Autoescuela;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM Profesores;
+----+-----+-----+-----+-----+
| id | nombre | apellido | dni | especialidad |
+----+-----+-----+-----+-----+
| 1 | Ana | López | 12345678A | Teórica |
| 2 | Luis | Pérez | 23456789B | Práctica |
| 3 | María | Gómez | 34567890C | Teórica |
| 4 | Carlos | Ruiz | 45678901D | Práctica |
| 5 | Laura | Santos | 56789012E | Teórica |
+----+-----+-----+-----+
5 rows in set (0.01 sec)

mysql> []

```

## Ejercicio dockerfile Mongodb

Creamos el directorio de trabajo:

```
mkdir mongodocker
cd mongodocker
```

Creamos el dockerfile:

```
FROM mongo

VOLUME ["/data/db"]

WORKDIR /data

CMD ["mongod"]

EXPOSE 27017
EXPOSE 28017
```

Ahora creamos la imagen:

```
docker build -t mongodocker .
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mongodocker	latest	5ffe29c1dcf1	4 minutes ago	890MB
mongoapp_webapp	latest	2b2851de2de8	4 hours ago	144MB
php	8.2-apache	b2ea8e26e2cd	2 days ago	502MB
custom2019image	latest	c6b596e5d8f	2 days ago	1.57GB
moodle19_docker_web	latest	90ad99e46347	3 days ago	624MB
<none>	<none>	b85d31b20162	3 days ago	624MB
postgres	latest	2258423a48c0	6 days ago	438MB
autoescuela-mysql	latest	d995ef91d5bd	9 days ago	772MB
mongo	latest	a7c5f35e6c6d	13 days ago	890MB
wordpress	latest	260d4fc3dbf	2 weeks ago	703MB
dpagent/pgadmin4	latest	077259320cb6	2 weeks ago	524MB
mysql	latest	2c849dee4ca9	4 weeks ago	859MB
mcr.microsoft.com/mssql/server	2022-latest	2b41d0be8283	2 months ago	1.65GB
mariadb	latest	e8147701838b	3 months ago	328MB
autoescuela-phpmyadmin	latest	d0a8c30d3d4c	3 months ago	571MB
mcr.microsoft.com/mssql-tools	latest	bc2b6cd40cb9	7 years ago	218MB
borja@MacBook-Pro-de-Borja mongodocker	latest			

Ahora ya solo nos quedaría probar la imagen levantando un contenedor con ella y ejecutar una bash en el:

```
docker run -p 27017:27017 -p 28017:28017 mongodocker
```

```
mongodocker -- mongosh mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000 -- zsh -- 139x29
[borja@MacBook-Pro-de-Borja mongodocker % docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
mongodocker          latest   5ffe29c1dcf1  4 minutes ago  890MB
mongoapp_webapp     latest   2b2851de2de8  4 hours ago   144MB
php                 8.2-apache  b2ea8e26e2cd  2 days ago   502MB
custom2019image     latest   c6b596e5d8f   2 days ago   1.57GB
moodle19_docker_web latest   90ad99e46347  3 days ago   624MB
<none>              <none>   b85d31b20162  3 days ago   624MB
postgres             latest   2258423a48c0  6 days ago   438MB
autoescuela-mysql   latest   d995ef91d5bd  9 days ago   772MB
mongo               latest   a7c5f35e6c6d  13 days ago  890MB
wordpress            latest   260d4fc3dbf  2 weeks ago  703MB
dpagent/pgadmin4    latest   077259320cb6  2 weeks ago  524MB
mysql               latest   2c849dee4ca9  4 weeks ago  859MB
mcr.microsoft.com/mssql/server  2022-latest  2b41d0be8283  2 months ago  1.65GB
mariadb              latest   e8147701838b  3 months ago  328MB
autoescuela-phpmyadmin latest   d0a8c30d3d4c  3 months ago  571MB
mcr.microsoft.com/mssql-tools  latest   bc2b6cd40cb9  7 years ago  218MB
borja@MacBook-Pro-de-Borja mongodocker %

mongodocker -- mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 -- com.docker.cli - docker exec -it nervous_hertz...
[borja@MacBook-Pro-de-Borja mongodocker % docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
d8e8c469a0a9        mongodocker        "docker-entrypoint.s..."   About a minute ago   Up 49 seconds   0.0.0.0:27017->27017/tcp, 0.0.0.0:28017->28017/tcp   nervous_hertz
[borja@MacBook-Pro-de-Borja mongodocker % docker exec -it nervous_hertz bash
[1] 0d8ebc40a9492/dash: mongo
bash: mongo: command not found
root@0d8ebc40a9492:/data# mongo
Current Mongosh Log ID: 6825ed92a659ef18b2d861df
Connecting to:          mongod://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.0
Using MongoDB:          8.0.9
Using Mongosh:          2.5.0
For mongosh info see: https://www.mongodb.com/docs/mongod-shell/
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2025-05-15T13:31:43.479+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-05-15T13:31:44.777+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2025-05-15T13:31:44.777+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T13:31:44.777+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T13:31:44.777+00:00: We suggest setting the contents of sysfsFile to 0.
2025-05-15T13:31:44.777+00:00: vm.max_map_count is too low
2025-05-15T13:31:44.777+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
-----

test> show dbs
admin 40.00 KiB
config 12.00 KiB
local 72.00 KiB
test> use wirtz
switched to db wirtz
wirtz> db.wirtz.insertOne({ asignatura: "Sistemas Xestores de Bases de Datos" });
{
  acknowledged: true,
  insertedId: ObjectId('6825ee07a659ef18b2d861e0')
}
wirtz> db.wirtz.find()
[
  {
    _id: ObjectId('6825ee07a659ef18b2d861e0'),
    asignatura: "Sistemas Xestores de Bases de Datos"
  }
]
wirtz>
```

## Ejemplo dockerfile dos contenedores

Vamos a crear como antes dos contenedores, uno con sql y el otro con phpmyadmin para administrar la base de datos, usaremos dockerfile en ambos y los uniremos mediante una red creada. Creamos todo dentro de la carpeta autoescuela.

Comenzamos por crear la red:

```
docker network create autoescuela-net
```

Ahora creamos la carpeta mysql dentro de autoescuela y en ella creamos el contenido de la base de datos init.sql

```
CREATE TABLE Profesores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    apellido VARCHAR(100),
    dni VARCHAR(20),
    especialidad VARCHAR(50)
);

INSERT INTO Profesores (nombre, apellido, dni, especialidad) VALUES
('Ana', 'López', '12345678A', 'Teórica'),
('Luis', 'Pérez', '23456789B', 'Práctica'),
('María', 'Gómez', '34567890C', 'Teórica'),
('Carlos', 'Ruiz', '45678901D', 'Práctica'),
('Laura', 'Santos', '56789012E', 'Teórica');
```

Creamos el dockerfile también:

```
FROM mysql:8.0

ENV MYSQL_ROOT_PASSWORD=Abcd1234.
ENV MYSQL_DATABASE=autoescuela
ENV MYSQL_USER=borja
ENV MYSQL_PASSWORD=Abcd1234.

COPY init.sql /docker-entrypoint-initdb.d/

EXPOSE 3306
```

Ahora creamos la carpeta phpmyadmin y dentro su dockerfile:

```
FROM phpmyadmin/phpmyadmin

ENV PMA_HOST=autoescuela-db
ENV PMA_PORT=3306
ENV PMA_USER=borja
ENV PMA_PASSWORD=Abcd1234.

EXPOSE 80
```

Ahora construimos las imágenes:

```
cd autoescuela/mysql
docker build -t autoescuela-mysql .

cd autoescuela/phpmyadmin
docker build -t autoescuela-phpmyadmin .
```

Y por último, ejecutamos los contenedores:

```
docker run -d \
-p 8080:80 \
--name autoescuela-phpmyadmin \
--network autoescuela-net \
-e PMA_HOST=autoescuela-mysql autoescuela-phpmyadmin

docker run -d \
--name autoescuela-mysql \
--network autoescuela-net \
```

```
-e MYSQL_ROOT_PASSWORD=root \
-p 3306:3306 autoescuela-mysql
```

Comprobamos que todo funciona:

\*\*Durante este ejercicio hemos tenido varios problemas por fallos en usuarios y contraseñas en los dockerfile, al no ponerlos exactamente iguales nos daba fallos al intentar acceder a phpmyadmin. También debemos especificar la misma red network.

Para entender mejor que debemos incluir en dockerfile y que debemos añadir a docker run:

✓ Lo que puedes OMITIR en `docker run` si está en el Dockerfile:

Configuración	Ejemplo en Dockerfile	¿Se puede omitir en <code>docker run</code> ?
Variables de entorno	ENV MSSQL_MEMORY_LIMIT_MB=2048	Sí (usará el valor por defecto)
Estructura de carpetas	RUN mkdir /data	Sí (la carpeta se creará)
Instalación de paquetes	RUN apt-get install -y nano	Sí (viene preinstalado)
Puerto expuesto	EXPOSE 1433	Sí (pero no se publicará externamente)

✖ Lo que DEBES seguir especificando en `docker run` (aunque esté en Dockerfile):

Configuración	Razón	Ejemplo obligatorio en <code>docker run</code>
<b>Publicar puertos</b>	EXPOSE solo documenta, no publica	<code>-p 1433:1433</code>
<b>Persistencia (volúmenes)</b>	Las carpetas existen pero no son persistentes	<code>-v ./data:/var/opt/mssql/data</code>
<b>Secretos/credenciales</b>	Por seguridad (ej: contraseñas)	<code>-e MSSQL_SA_PASSWORD=ClaveSegura</code>
<b>Recursos del sistema</b>	Depende del host	<code>--cpus=2 --memory=4G</code>

## SQL Server

### Crear contenedor y arranque

Lo primero que vamos hacer es crear un contenedor con Sql Server y arrancarlo, para ello ejecutamos:

```
docker run -e "ACCEPT_EULA=Y" \
-e "MSSQL_SA_PASSWORD=Abcd1234." \
--name sqlserver2022 \
-p 1433:1433 \
-v sql1data:/var/opt/mssql \
-d mcr.microsoft.com/mssql/server:2022-latest
```

—env ACCEPT\_EULA=Y: Indica que aceptamos la **licencia de uso** del software (EULA = End User License Agreement). Es obligatoria para que el contenedor arranque. Si no se incluye, el software se detiene o lanza un error indicando que no se aceptaron los términos

-e "MSSQL\_SA\_PASSWORD=Abcd1234.": Define la contraseña del usuario SA (System Administrator)

—name sqlserver2022: Nombre del contenedor (sino tendremos que usar el ID que genera Docker)

-p 1433:1433: Mapeo del puerto del host al puerto del contenedor

-v sql1data:/var/opt/mssql: volumen persistente para almacenar los datos, nombre del volumen y directorio

-d: (detached) para que el contenedor corra en segundo plano

mcr.microsoft.com/mssql/server:2022-latest: imagen usada para montar el contenedor

Ahora arrancamos el contenedor:

```
docker exec -it sqlserver2022 /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Abcd1234.
```

- docker exec: Ejecuta un comando dentro de un contenedor ya iniciado.
- -it: Asigna una terminal interactiva (permite ver la salida y escribir comandos).
- sqlserver2022: Nombre del contenedor donde se ejecutará el comando.
- /opt/mssql-tools/bin/sqlcmd: Ruta al cliente de línea de comandos `sqlcmd` dentro del contenedor, que permite conectarse al servidor SQL.
- S localhost: Conecta al servidor SQL usando `localhost` como dirección (dentro del contenedor significa el propio contenedor).
- -U sa: Usuario de autenticación, en este caso `sa` (System Administrator).
- -P Abcd1234.: Contraseña del usuario `sa`.

Pero esto nos da fallo, porque la imagen de `sqlserver2022` no contiene `sqlcmd`. Aquí tenemos dos opciones, o bien usar `sqlcmd` desde nuestro ordenador contra el contenedor de `sqlserver2022` o bien crear otro contenedor con `sql-tools` que tenga `sqlcmd` y ejecutarlo contra el contenedor que contiene `sqlserver2022`. En este caso vamos a ejecutar `sqlcmd` en el propio ordenador huesped contra el contenedor.

```
sqlcmd -S localhost -U sa -P Abcd1234.
borja@MacBook-Pro-de-Borja ~ % docker run -e "ACCEPT_EULA=Y" \
-e "MSSQL_SA_PASSWORD=Abcd1234." \
--name sqlserver2022 \
-p 1433:1433 \
-v sql1data:/var/opt/mssql \
-d mcr.microsoft.com/mssql/server:2022-latest
64d01651c079d50025d4ff39f64dfa192280fb56dc622bc7e86e3c1c76aa335e
borja@MacBook-Pro-de-Borja ~ % docker exec -it sqlserver2022 /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Abcd1234.
OCI runtime exec failed: exec failed: container_linux.go:367: starting container process caused: exec: "/opt/mssql-tools/bin/sqlcmd": stat /opt/mssql-tools/bin/sqlcmd: no such file or directory: unknown
borja@MacBook-Pro-de-Borja ~ % docker run -it --rm mcr.microsoft.com/mssql-tools \
/opt/mssql-tools/bin/sqlcmd -S host.docker.internal -U sa -P Abcd1234.

1> exit
borja@MacBook-Pro-de-Borja ~ % sqlcmd -S localhost -U sa -P Abcd1234.

1> █
```

Tras la consulta al profesor, ahora sabemos que el problema era la ruta en la que se encuentra `sqlcmd` dentro del contenedor que al cambiar la versión, esta ruta ha cambiado y ahora es: `/usr/bin/sqlcmd`.

Ahora podemos comenzar a trabajar con Sqlserver. Creamos una base de datos y consultamos las bases de datos existentes, conectamos Azzure Database y comprobamos que accede directamente al contenedor.

**Connection Details**

Connection type	Microsoft SQL Server
Input type	<input checked="" type="radio"/> Parameters <input type="radio"/> Connection String
Server *	...
Authentication type	SQL Login
User name *	sa
Password	*****
<input type="checkbox"/> Remember password	
Database	<Default>
Encrypt <small>?</small>	Mandatory
Trust server certificate	<input type="radio"/> True
Server group	<Default>
Name (optional)	
<input type="button" value="Advanced..."/>	
<input type="button" value="Connect"/> <input type="button" value="Cancel"/>	

Terminal Output (sqlcmd):

```

y: unknown
borja@MacBook-Pro-de-Borja ~ % docker run -it --rm mcr.microsoft.com/mssql-tools \
/opt/mssql-tools/bin/sqlcmd -S host.docker.internal -U sa -P Abcd1234.

1> exit
borja@MacBook-Pro-de-Borja ~ % sqlcmd -S localhost -U sa -P Abcd1234.

1> CREATE DATABASE autoescuela;
2> GO
1> SHOW DATABASES;
2> GO
Msg 2812, Level 16, State 62, Server 2842cb949b43, Line 1
Could not find stored procedure 'SHOW'.
1> SELECT name FROM sys.databases;
2> GO
name

-----
master
tempdb
model
msdb
autoescuela

(5 rows affected)
1> 

```

SSMS Screenshot:

The SSMS interface shows the 'Databases' node selected in the tree view. The right pane displays a list of databases: autoescuela, master, model, msdb, and tempdb. The 'Choose SQL Language' dropdown is set to 'master'.

Ahora vamos a probar a restaurar un backup de nuestra base de datos autoescuela desde nuestro repositorio de Github. Primero descargamos el backup del repositorio con curl:

```

curl -L -o autoescuela.bak
https://github.com/Bcoast84/SXBBDD/blob/main/Tarea%203%20-
%20Modelado%20SQL/Modelado/autoescuela.bak

```

Y ahora copiamos el bakcup en el contenedor y restauramos desde sqlcmd:

```

docker cp autoescuela.bak sqlserver2022:/var/opt/mssql/backup/autoescuela.bak

```

```

borja@MacBook-Pro-de-Borja /tmp % curl -L -o autoescuela.bak https://github.com/Bcoast84/SXBBDD/raw/refs/heads/main/Tarea%203%20-%20Modelado%20SQL/Modelado/autoescuela.bak
  % Total    % Received % Xferd  Average Speed   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
[  0     0     0     0     0     0      0  0:00:01  0:00:01  --:--:-- 0
100 6552k 100 6552k  0     0  4763k      0  0:00:01  0:00:01  --:--:-- 10.4M
borja@MacBook-Pro-de-Borja /tmp % docker cp autoescuela.bak sqlserver2022:/var/opt/mssql/backup/autoescuela.bak
borja@MacBook-Pro-de-Borja /tmp % sqlcmd -S localhost -U sa -P Abcd1234.
1> RESTORE FILELISTONLY FROM DISK = '/var/opt/mssql/backup/autoescuela.bak';
2> GO
LogicalName
[           PhysicalName
[           Type FileGroupName
[           Size      MaxSize
[           FileId      CreateLSN      DropLSN      UniqueId
[           ReadOnlyLSN  ReadWriteLSN  BackupSizeInBytes  SourceBlockSize FileGroupId
[           LogGroupGUID  DifferentialBaseLSN  DifferentialBaseGUID  IsReadOnly
y IsPresent TDEThumbprint      SnapshotUrl

autoescuela
C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\autoescuela.mdf
D      PRIMARY
8388608      351843720
80640          1          0          0 41F028D7-9DEA-443E-B22C-10D0F
41FE8D7          0          0          0 6488064      512      1
NULL
3900000021860001 DABBF32F-E76D-47CC-8394-E57368661C87
0          1 NULL          NULL

autoescuela_log
C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\autoescuela_log.ldf
L      NULL
8388608      21990232
55552          2          0          0 39BB68C8-6836-479D-8AA0-416C1
48BCBB9          0          0          0 512      0
NULL
0          1 NULL          NULL

(2 rows affected)
1> USE autoescuela:

```

## SQL Server con Dockerfile

Primero creamos una carpeta para realizar el ejercicio y nos posicionamos en ella:

```
mkdir sqlserver  
cd sqlserver
```

## Dockerfile

```
FROM mcr.microsoft.com/mssql/server:2019-GDR1-ubuntu-16.04
USER root

RUN mkdir /var/opt/sqlserver
RUN chown mssql /var/opt/sqlserver

ENV MSSQL_BACKUP_DIR="/var/opt/sqlserver"
ENV MSSQL_DATA_DIR="/var/opt/sqlserver"
ENV MSSQL_LOG_DIR="/var/opt/sqlserver"

USER mssql
CMD /opt/mssql/bin/sqlservr
```

Se construye la imagen con docker build:

```
docker build -t custom2019image .
```

Se inicia el contenedor con docker run:

```
docker container run -d \
-p 15789:1433 \
--volume sqlserver:/var/opt/sqlserver \
--env MSSQL_SA_PASSWORD=Abcd1234. \
--env ACCEPT_EULA=Y \
--name testcontainer \
custom2019image
```

Ejecutamos sentencia para poder ejecutar comandos desde la bash:

```
docker exec -it testcontainer "bash"  
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P "Abcd1234."
```

## Ejemplo backup-restore

Creamos la carpeta para trabajar:

```
mkdir backrestore
cd backrestore
```

Arrancamos el contendor:

```
sudo docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=Abcd1234.' \
--name 'sql1' -p 1401:1433 \
-v sql1data:/var/opt/mssql \
-d mcr.microsoft.com/mssql/server:2022-latest
```

Ejecutamos un SELECT para ver que funciona:

```
sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd \
-C -S localhost -U SA -P 'Abcd1234.' \
-Q 'SELECT name FROM sys.databases'
```

Intentamos cambiar la contraseña:

```
docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd \
-S localhost -U SA -P "Abcd1234." \
-Q "ALTER LOGIN SA WITH PASSWORD='NuevaClaveSegura123.'"
```

Nos da error por el certificado, probamos deshabilitando la validación de SSL:

```
docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd \
-S localhost -U SA -P "Abcd1234." \
-Q "ALTER LOGIN SA WITH PASSWORD='NuevaClaveSegura123.'"
-N -C
```

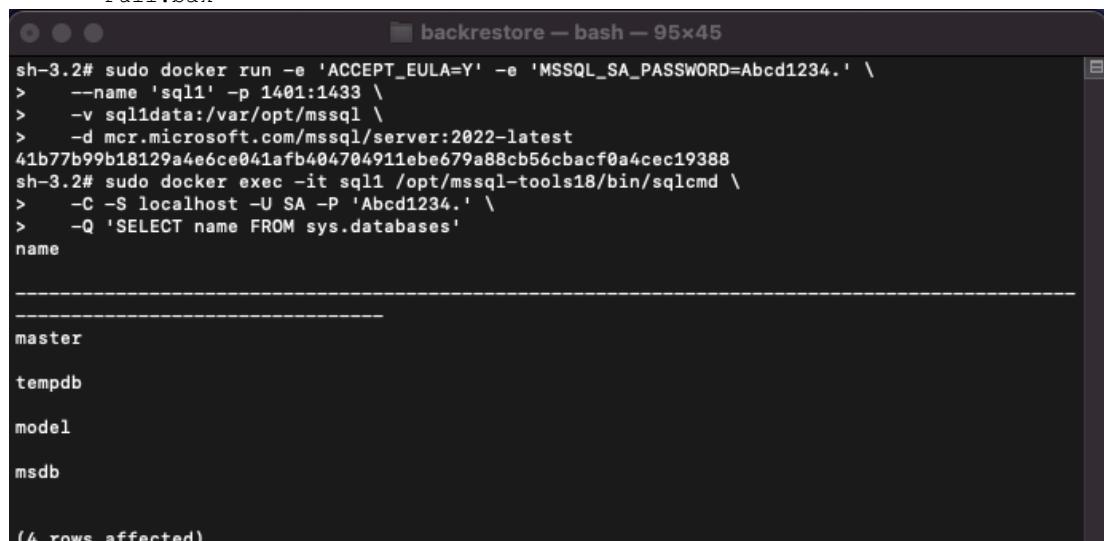
\*\*Después volvemos a realizar el cambio para evitar confusiones de contraseña.

Creamos el directorio para guardar el backup:

```
sudo docker exec -it sql1 mkdir /var/opt/mssql/backup
```

Descargamos el backup:

```
cd ~
curl -L -o wwi.bak 'https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak'
```



```
sh-3.2# sudo docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=Abcd1234.' \
> --name 'sql1' -p 1401:1433 \
> -v sql1data:/var/opt/mssql \
> -d mcr.microsoft.com/mssql/server:2022-latest
41b77b9b18129a4e6ce041afb404704911ebe679a88cb56cbacf0a4cec19388
sh-3.2# sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd \
> -C -S localhost -U SA -P 'Abcd1234.' \
> -Q 'SELECT name FROM sys.databases'
name
-----
master
tempdb
model
msdb

(4 rows affected)
```

```

sh-3.2# docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd \
> -S localhost -U SA -P "Abcd1234." \
> -Q "ALTER LOGIN SA WITH PASSWORD='NuevaClaveSegura123.'" \
> -N -C
sh-3.2# sudo docker exec -it sql1 mkdir /var/opt/mssql/backup
sh-3.2# curl -L -o wwi.bak 'https://github.com/Microsoft/sql-server-samples/releases/download/w
ide-world-importers-v1.0/WideWorldImporters-Full.bak'
      % Total      % Received % Xferd  Average Speed   Time     Time      Current
                                     Dload  Upload Total Spent   Left Speed
0     0     0     0     0     0      0  --:--:-- --:--:-- --:--:-- 0     0     0     0     0     0
0     0  121M     0     0     0     0      0  0:00:33 0:00:01  14 121M  14 17.3M  0     0 6912k  0     0
3737k     0  0:00:33 0:00:01  14 121M  14 17.3M  0     0 6241k  0 0:00:19 0:00:0
2    24 121M  24 29.9M  0     0 7976k  0 0:00:15 0:00:03  35 121M  35 42.9M  0
0  9082k     0  0:00:13 0:00:04  46 121M  46 56.6M  0     0 9782k  0 0:00:12 0:00
:05 54 121M  54 66.6M  0     0 9973k  0 0:00:12 0:00:06  63 121M  63 76.6M  0
0  9.7M     0  0:00:12 0:00:07  72 121M  72 87.8M  0     0 9.9M  0 0:00:12 0:
00:08 82 121M  82 99.8M  0     0 10.1M  0 0:00:11 0:00:09  91 121M  91 111M  0
0  10.2M     0  0:00:11 0:00:10 100 121M  100 121M  0     0 10.4M  0 0:00:11
0:00:11  --:--:-- 11.3M
sh-3.2# du -h wwi.bak
128M  wwi.bak
sh-3.2# 

```

Copiamos el backup en el contenedor:

```
sudo docker cp wwi.bak sql1:/var/opt/mssql/backup
```

Restauramos la base de datos

```

sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P
'Abcd1234.' -C -Q 'RESTORE DATABASE WideWorldImporters FROM DISK =
"/var/opt/mssql/backup/wwi.bak" WITH MOVE "WWI_Primary" TO
"/var/opt/mssql/data/WideWorldImporters.mdf",
MOVE "WWI_UserData" TO
"/var/opt/mssql/data/WideWorldImporters_userdata.ndf",
MOVE "WWI_Log" TO "/var/opt/mssql/data/WideWorldImporters.ldf",
MOVE "WWI_InMemory_Data_1" TO
"/var/opt/mssql/data/WideWorldImporters_InMemory_Data_1"'

```

Listamos las bases de datos:

```
sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P 'Abcd1234.' -Q 'SELECT name FROM sys.databases'
```

Nos da error por el cifrado, debemos añadir “-C”:

```
sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P 'Abcd1234.' -C -Q 'SELECT name FROM sys.databases'
```

Ahora realizamos consulta contra la base de datos:

```
sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P 'Abcd1234.' -C -Q 'SELECT TOP 10 StockItemID, StockItemName FROM
WideWorldImporters.Warehouse.StockItems ORDER BY StockItemID'
```

Creamos un nuevo backup:

```

sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -
P 'Abcd1234.' -C -Q
"BACKUP DATABASE [WideWorldImporters] TO DISK =
N'"/var/opt/mssql/backup/wwi_2.bak'
WITH NOFORMAT, NOINIT, NAME = 'WideWorldImporters-full', SKIP, NOREWIND,
NOUNLOAD, STATS = 10"

```

```
backrestore — bash — 146x57
Database 'WideWorldImporters' running the upgrade step from version 949 to version 950.
Database 'WideWorldImporters' running the upgrade step from version 950 to version 951.
Database 'WideWorldImporters' running the upgrade step from version 951 to version 952.
Database 'WideWorldImporters' running the upgrade step from version 952 to version 953.
Database 'WideWorldImporters' running the upgrade step from version 953 to version 954.
Database 'WideWorldImporters' running the upgrade step from version 954 to version 955.
Database 'WideWorldImporters' running the upgrade step from version 955 to version 956.
Database 'WideWorldImporters' running the upgrade step from version 956 to version 957.
RESTORE DATABASE successfully processed 58496 pages in 3.261 seconds (140.570 MB/sec).
sh-3.2# sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P 'Abcd1234.' -Q 'SELECT name FROM sys.databases'
Sqlcmd: Error: Microsoft ODBC Driver 18 for SQL Server : SSL Provider: [error:0A000086:SSL routines::certificate verify failed:self-signed certificate].
Sqlcmd: Error: Microsoft ODBC Driver 18 for SQL Server : Client unable to establish connection. For solutions related to encryption errors, see https://go.microsoft.com/fwlink/?linkid=2226722.
sh-3.2# sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P 'Abcd1234.' -C -Q 'SELECT name FROM sys.databases'
name
-----
master
tempdb
model
msdb
WideWorldImporters

(5 rows affected)
sh-3.2# sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P 'Abcd1234.' -C -Q 'SELECT TOP 10 StockItemID, StockItemName
FROM WideWorldImporters.Warehouse.StockItems ORDER BY StockItemID'
StockItemID StockItemName
-----
1 USB missile launcher (Green)
2 USB rocket launcher (Gray)
3 Office cube periscope (Black)
4 USB food flash drive - sushi roll
5 USB food flash drive - hamburger
6 USB food flash drive - hot dog
7 USB food flash drive - pizza slice
8 USB food flash drive - dim sum 10 drive variety pack
9 USB food flash drive - banana
10 USB food flash drive - chocolate bar

(10 rows affected)
sh-3.2# sudo docker exec -it sql1 /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P 'Abcd1234.' -C -Q "BACKUP DATABASE [WideWorldImporters] TO D
ISK = N'/var/opt/mssql/backup/wwi_2.bak' WITH NOFORMAT, NOINIT, NAME = 'WideWorldImporters-full', SKIP, NOREWIND, NOUNLOAD, STATS = 10"
10 percent processed.
20 percent processed.
30 percent processed.
40 percent processed.
50 percent processed.
60 percent processed.
Processed 1536 pages for database 'WideWorldImporters', file 'WWI_Primary' on file 1.
Processed 53112 pages for database 'WideWorldImporters', file 'WWI_UserData' on file 1.
70 percent processed.
Processed 3865 pages for database 'WideWorldImporters', file 'WWI_InMemory_Data_1' on file 1.
Processed 284 pages for database 'WideWorldImporters', file 'WWI_Log' on file 1.
100 percent processed.
BACKUP DATABASE successfully processed 58797 pages in 3.753 seconds (122.394 MB/sec).
sh-3.2#
```

Copiamos el backup realizado fuera del contenedor:

```
sudo docker cp sql1:/var/opt/mssql/backup/wwi_2.bak wwi_2.bak
```

## MongoDB

Lo primero que vamos hacer es borrar todos los volúmenes que tenemos creados en docker, para hacer limpieza y verlo todo mas claro

```
docker volume prune
```

Ahora, aunque no es necesario crear el volumen previamente ya que el propio docker lo crea al crear el contenedor, vamos a probarlo:

```
docker volume create mongodbdata
```

Ahora levantamos el contenedor:

```
docker run -d \
-p 27017:27017 \
--name servidormongolocal \
-v mongodbdata:/data/db \
mongo:latest
```

Para controlar errores tenemos:

```
docker logs servidormongolocal
```

```
docker inspect servidormongolocal
```

Nos conectamos al contenedor:

```
docker exec -it servidormongolocal bash
#o bien:
docker exec -it servidormongolocal mongosh
```

```
borja@MacBook-Pro-de-Borja ~ % 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:
mongodata
df2013f77e12fc4cadcc5fcf269d248b0ee76cb0a0b5952553b0679ba13f3e06

Total reclaimed space: 210MB
[borja@MacBook-Pro-de-Borja ~ % docker volume create mongodata
mongodata
borja@MacBook-Pro-de-Borja ~ % docker run -d \
-p 27017:27017 \
--name servidormongolocal \
-v mongodata:/data/db \
mongo:latest
278c49e10463c507075d45388f6fbe0b76c27560493e0e9bc3a4fccf92022931
[borja@MacBook-Pro-de-Borja ~ % docker exec -it servidormongolocal bash
root@278c49e10463:~# ls
bin  data  docker-entrypoint-initdb.d  home  lib  media  opt  root  sbin  sys  usr
boot  dev  etc  js-yaml.js  lib64  mnt  proc  run  srv  tmp  var
root@278c49e10463:~# exit
exit
[borja@MacBook-Pro-de-Borja ~ % docker exec -it servidormongolocal mongosh
Current Mongosh Log ID: 6825bf81b220326756d861df
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.0
Using MongoDB: 8.0.9
Using Mongosh: 2.5.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2025-05-15T10:17:58.884+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-05-15T10:17:59.627+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2025-05-15T10:17:59.627+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T10:17:59.628+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T10:17:59.628+00:00: We suggest setting the contents of sysfsFile to 0.
2025-05-15T10:17:59.628+00:00: vm.max_map_count is too low
2025-05-15T10:17:59.628+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.

-----
[test> use alumnos
switched to db alumnos
[alumnos> db.alumnos.insertOne({Nombre:"Borja Costa"})
{
  acknowledged: true,
  insertedId: ObjectId('6825c006b220326756d861e0')
}
[alumnos> db.alumnos.find()
[
  { _id: ObjectId('6825c006b220326756d861e0'), Nombre: 'Borja Costa' }
]
[alumnos> show collections;
alumnos
[alumnos> exit
borja@MacBook-Pro-de-Borja ~ %
```

## Acceso desde la bash del huesped

Podemos acceder a mongosh desde la terminal con si tenemos instalado mongosh en el huesped.

```
borja@MacBook-Pro-de-Borja ~ % mongosh "mongodb://localhost:27017"
Current Mongosh Log ID: 6825e0edeefc8590477331ad
Connecting to: mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.1
Using MongoDB: 8.0.9
Using Mongosh: 2.2.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2025-05-15T12:36:26.701+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-05-15T12:36:28.674+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2025-05-15T12:36:28.674+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T12:36:28.674+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-15T12:36:28.674+00:00: We suggest setting the contents of sysfsFile to 0.
2025-05-15T12:36:28.674+00:00: vm.max_map_count is too low
2025-05-15T12:36:28.674+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
-----
```

## Acceso desde MongoDB Compass

Ahora podemos probar a acceder desde MongoDB Compass a la base de datos que se encuentra dentro del contenedor. Suficiente con añadir una conexión al puerto que escucha el contenedor con Mongo.

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' sidebar is open, showing 'ContenedorMongo' is selected. Under 'ContenedorMongo', the 'alumnos' database is selected, and within it, the 'alumnos' collection is selected. The main pane displays the 'Documents' tab, which shows a single document. The document's '\_id' field is highlighted in red, showing the value: `_id: ObjectId('6825c006b220326756d861e0')`. The 'Nombre' field is also highlighted in red, showing the value: `Nombre: 'Borja Costa'`. The 'Find' button is visible at the bottom of the document list.

\*\*Esto lo que nos demuestra es la capacidad de interactuar con el contenido de un contenedor casi como si estuviésemos actuando en el propio host.

## Docker Compose

**Docker Compose** es una herramienta que permite definir y gestionar múltiples contenedores Docker como un solo servicio. Se usa principalmente para aplicaciones que requieren varios servicios interconectados (por ejemplo, una aplicación web + base de datos). **Usa un archivo único** (`docker-compose.yml`). Para instalar Docker Compose, o bien instalamos Docker Desktop o bien mediante los comandos:

```
sudo apt-get update
sudo apt install docker-compose-plugin
```

\*\*En mi caso estoy usando docker desktop

- Iniciar docker compose: `docker-compose up -d`
- Detener docker compose: `docker-compose down -v`

Ejemplo de archivo `docker-compose.yml` para levantar Moodle y MariaDB:

```
version: '3'
services:
  moodle:
    image: bitnami/moodle
    ports:
      - "8080:8080"
    environment:
      - MOODLE_DATABASE_HOST=mariadb
      - MOODLE_DATABASE_NAME=moodle
      - MOODLE_DATABASE_USER=bn_moodle
      - MOODLE_DATABASE_PASSWORD=moodlepass
    depends_on:
      - mariadb
    volumes:
      - moodle_data:/bitnami/moodle

  mariadb:
    image: bitnami/mariadb
    environment:
      - MARIADB_ROOT_PASSWORD=rootpass
      - MARIADB_DATABASE=moodle
      - MARIADB_USER=bn_moodle
      - MARIADB_PASSWORD=moodlepass
    volumes:
      - mariadb_data:/bitnami/mariadb

volumes:
  moodle_data:
  mariadb_data:
```

## docker-entrypoint.sh

Un archivo `docker-entrypoint.sh` es un script que se ejecuta cuando se inicia un contenedor Docker. Puedes usarlo para automatizar tareas como crear una base de datos, usuarios, tablas, importar datos, etc., justo después de que el contenedor se levanta. Con MySQL y Postgre se ejecuta automáticamente pero con SQL Server debemos hacerlo de forma manual desde un script que se lance como entrypoint o desde un cmd modificado.

Ejemplo de `docker-entrypoint.sh`:

```
set -e

# Espera a que MySQL esté listo
until mysqladmin ping -h"$MYSQL_HOST" --silent; do
  echo "Esperando a MySQL..."
```

```

    sleep 2
done

# Crear base de datos y usuario
mysql -u root -p"$MYSQL_ROOT_PASSWORD" <<EOF
CREATE DATABASE IF NOT EXISTS midb;
CREATE USER IF NOT EXISTS 'usuario'@'%' IDENTIFIED BY 'clave';
GRANT ALL PRIVILEGES ON midb.* TO 'usuario'@'%';
FLUSH PRIVILEGES;
EOF

exec "$@"

```

## Ejercicio 1 Docker-compose (mysql)

Lo primero que haremos será crear una carpeta de trabajo y darle permisos:

```

sh-3.2# mkdir compose1
sh-3.2# chmod 777 compose1
sh-3.2# cd compose1

```

Creamos el archivo yml:

```

services:
  mysql:
    image: mysql:latest
    ports:
      - 3306:3306
    environment:
      - MYSQL_ROOT_PASSWORD=Abcd1234.
      - MYSQL_DATABASE=Prueba
      - MYSQL_USER=user
      - MYSQL_PASSWORD=Abcd1234.
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
  mysql_data:

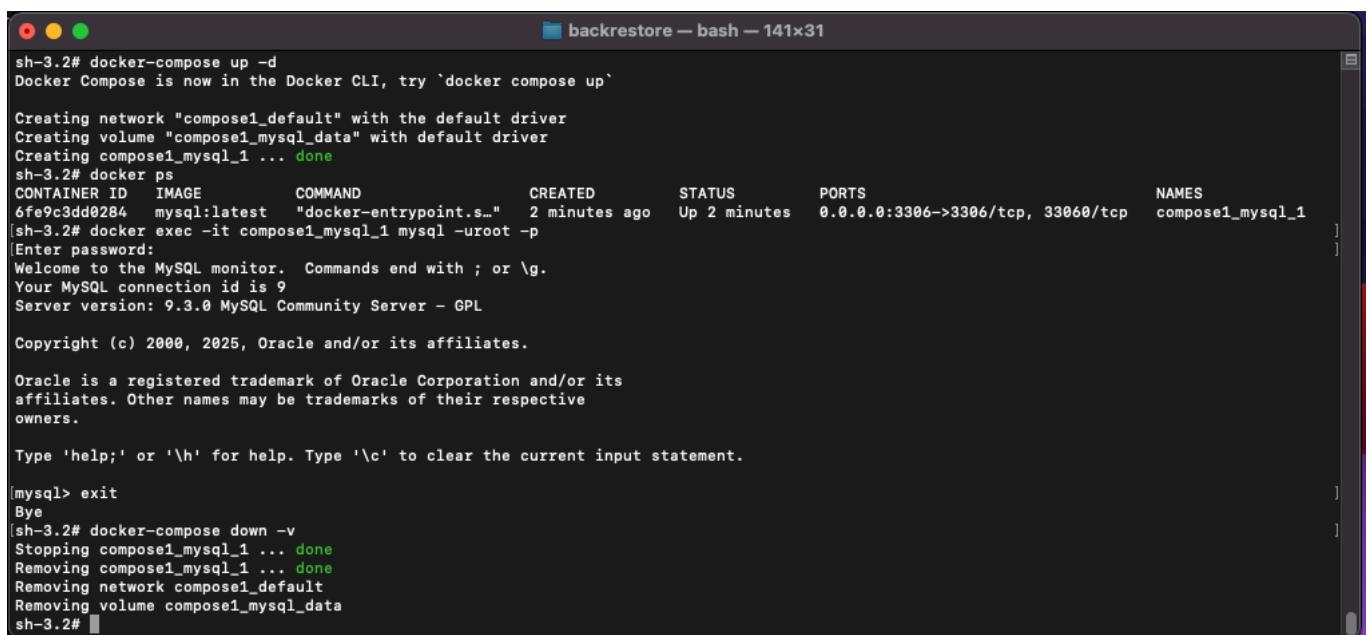
```

Ejecutamos docker compose, comprobamos que funciona y detenemos docker compose:

```

docker-compose up -d
docker-compose down -v

```



```

sh-3.2# docker-compose up -d
Docker Compose is now in the Docker CLI, try `docker compose up`

Creating network "compose1_default" with the default driver
Creating volume "compose1_mysql_data" with default driver
Creating compose1_mysql_1 ... done
sh-3.2# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
6fe9c3dd0284 mysql:latest "docker-entrypoint.s..." 2 minutes ago Up 2 minutes 0.0.0.0:3306->3306/tcp, 33060/tcp compose1_mysql_1
[sh-3.2# docker exec -it compose1_mysql_1 mysql -uroot -p
[Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
[sh-3.2# docker-compose down -v
Stopping compose1_mysql_1 ... done
Removing compose1_mysql_1 ... done
Removing network compose1_default
Removing volume compose1_mysql_data
sh-3.2#

```

## Ejercicio 2 Docker-compose (Apache-Mysql)

Levantar una aplicación web con **Apache + PHP** y **MySQL**

Lo primero que hacemos es crear la carpeta de trabajo:

```
mkdir appweb
cd appweb
```

Creamos el archivo docker-compose.yml, debemos definir el contenedor con apache+php y otro contenedor con mysql:

```
services:
  web:
    image: php:8.2-apache
    ports:
      - "8080:80"
    volumes:
      - ./src:/var/www/html
    depends_on:
      - db

  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: midb
      MYSQL_USER: user
      MYSQL_PASSWORD: userpass
    volumes:
      - db_data:/var/lib/mysql

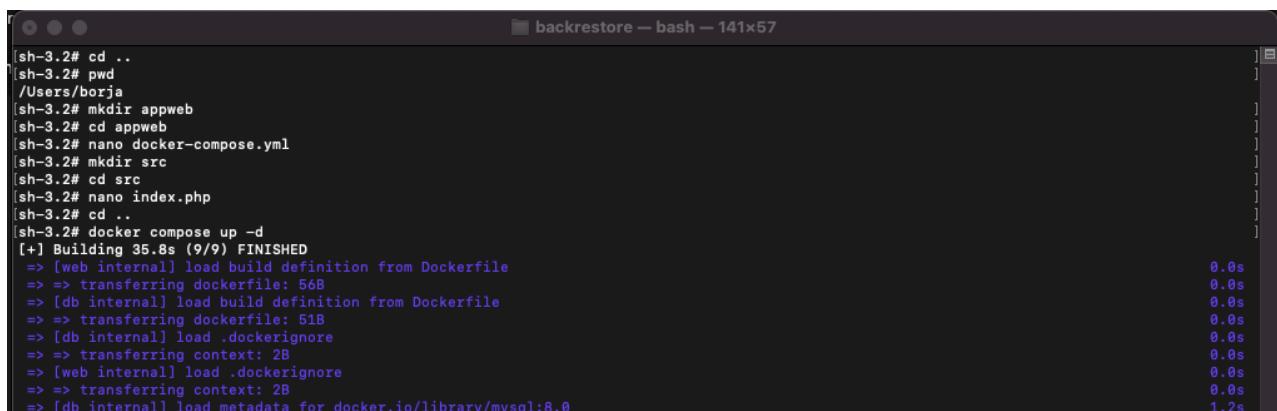
volumes:
  db_data:
```

Ahora creamos la carpeta src y dentro de ella el archivo index.php que será lo que nos servirá para comprobar que todo funciona correctamente:

```
<?php
$cexion = new mysqli("db", "user", "userpass", "midb");
if ($conexion->connect_error) {
    die("Error de conexión: " . $conexion->connect_error);
}
echo "Conexión exitosa a MySQL desde PHP en Docker";
?>
```

Ahora levantamos el docker-compose:

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



```
sh-3.2# cd ..
sh-3.2# pwd
/Users/borja
sh-3.2# mkdir appweb
sh-3.2# cd appweb
sh-3.2# nano docker-compose.yml
sh-3.2# mkdir src
sh-3.2# cd src
sh-3.2# nano index.php
sh-3.2# cd ..
sh-3.2# docker compose up -d
[+] Building 35.8s (9/9) FINISHED
=> [web internal] load build definition from Dockerfile          0.0s
=> => transferring dockerfile: 56B                                0.0s
=> [db internal] load build definition from Dockerfile          0.0s
=> => transferring dockerfile: 51B                                0.0s
=> [db internal] load .dockerignore                            0.0s
=> => transferring context: 2B                                0.0s
=> [web internal] load .dockerignore                            0.0s
=> => transferring context: 2B                                0.0s
=> [db internal] load metadata for docker.io/library/mysql:8.0  1.2s
```

=> writing image sha256:fb210f163092663603a79f4c9ffbd4762215cbfb84892948f8ab38a85e741ebf	0.0s
=> naming to docker.io/library/mysql:8.0	0.0s
=> writing image sha256:d888d3268680fa97adda9e2974f2eb043e09c85b792e558534e275c3169c1ce0	0.0s
=> naming to docker.io/library/php:8.2-apache	0.0s
=> [web 1/1] FROM docker.io/library/php:8.2-apache@sha256:e2408924aac97ed8dce0ba54adff30443fe7a940a87d7b0d083b36941d8aa431	33.6s
=> => resolve docker.io/library/php:8.2-apache@sha256:e2408924aac97ed8dce0ba54adff30443fe7a940a87d7b0d083b36941d8aa431	0.0s
=> sha256:4b8b3e98f906d144b5a1154d9a83de871e07a37a20d576c403c27f5339f12b 11.38KB / 11.38KB	0.0s
=> sha256:fceda96c6100027e1daaaec4b4c4f71798c88775c7ea365a5d01c2353166a19f 3.64KB / 3.64KB	0.0s
=> sha256:45d789c9d8451a765135948c716260205cb102d38818203f14f2143bd017f 226B / 226B	0.6s
=> sha256:1d3659077708e54d1b5f70b031e8095c72e3e81bdda16c103811d24b991b801 104.33MB / 104.33MB	16.7s
=> sha256:e2408924aac97ed8dce0ba54adff30443fe7a940a87d7b0d083b36941d8aa431 10.40KB / 10.40KB	0.0s
=> sha256:254e724d7862dc53abbd3bf76a29399cdd3d8aadeaf8e6a680659 28.23MB / 28.23MB	4.9s
=> sha256:b98ca3230ca2b3d3fc792c8668ba6ec5188696b7847a5cddb7b99ec47a5df8 226B / 226B	1.0s
=> sha256:a2de5d8c51569949760985d9ad7e071c4591332b0b2bd3c043219e80521554 20.12MB / 20.12MB	9.0s
=> sha256:ff5d5a2cec58651bc6c3e3ac76ad005036721b0f65932c8ad50d8cf3235a63 428B / 428B	5.2s
=> sha256:41763122f8aa0bc5ab13d5b12b1b160f50426e6cb824c06189824c9ed101e1 483B / 483B	5.6s
=> extracting sha256:254e724d7862dc53abbd3bf76a29399cdd3d8aadeaf8e6a680659 488B / 488B	11.9s
=> sha256:727efbdd2207dd74f4964b3c6b6551fb3506cdca11c3c753b5c7bf78555148 12.28MB / 12.28MB	9.2s
=> sha256:968a4c5f8e3b8548fa3ae2227c9a0c25b6828f0b19ebabd16d52e3627cb3c984 488B / 488B	9.3s
=> sha256:a6f9d43f5dbf7cf8e6229e60dea369c904c89db8745261796298b61335e60be5 2.45KB / 2.45KB	12.8s
=> sha256:cbe5f891bcf0d10edbe9f2cd08a7fc82c001c36c0e804e4b5e34c52abeee3d 243B / 243B	10.0s
=> sha256:d3ce667971450271619017d6a3240f83b7a2057327cd5f1820717bece3444efa 889B / 889B	10.4s
=> sha256:474fb780ef5461cfa02571ae0db9a0dc1e0cd5574a4ad75e08dc38e8acc1 32B / 32B	10.7s
=> extracting sha256:45d789cd9c8451a765135948c716260205cb162d38818203f14f2143bd017f	0.0s
=> extracting sha256:1d3659077708e54d1b5f70b031e8095c72e3ee81bdda16c103811d24b991b803	10.0s
=> extracting sha256:b98ca3230ca2b3d3fc792c8668ba6ec5188696b7847ae5cddb7b99ec47a5df50	0.0s
=> extracting sha256:a2de5d8c51569949760985d9ad7e071c4591332b0b2bd3c043219e80521554	1.6s
=> extracting sha256:ff5d5a2cec58651bc6c3e3ac76ad005036721b0f65932c8ad50d8cf3235a63	0.0s
=> extracting sha256:41763122f8aa0bc5ab13d5b12b1b160f50426e6cb824c06189824c9ed101e1	0.0s
=> extracting sha256:77efbdd2207dd74f4964b3c6b6551fb3506cdca11c3c753b5c7bf078555148	0.1s
=> extracting sha256:968a4c5f8e3b8548fa3ae2227c9a0c25b6828f0b19ebabd16d52e3627cb3c904	0.0s

Comprobamos que nos da error, consultada documentación vemos que puede ser por la extensión mysqli que en la imagen usada no viene con mysqli habilitada. Vamos a probar usando dockerfile para añadir mysqli a la imagen. Creamos el dockerfile en la carpeta de trabajo:

```
FROM php:8.1-apache
RUN docker-php-ext-install mysqli
```

Modificamos el docker-compose para añadir el build . :

```
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "80:80"
    volumes:
      - ./src:/var/www/html
    depends_on:
      - mysql

  mysql:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=Abcd1234.
      - MYSQL_DATABASE=Prueba
      - MYSQL_USER=user
      - MYSQL_PASSWORD=Abcd1234.
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
  mysql_data:
```

Archivo index.php:

```
<?php
$conexion = new mysqli("mysql", "user", "Abcd1234.", "Prueba");

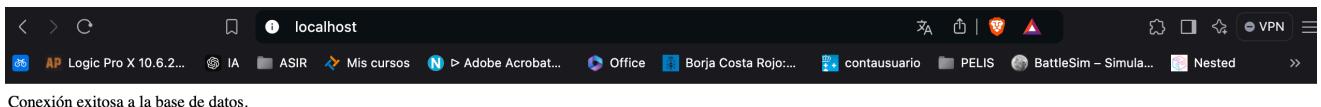
if ($conexion->connect_error) {
    die("Conexión fallida: " . $conexion->connect_error);
}

echo "Conexión exitosa a la base de datos.";
```

Levantamos el docker-compose, en este caso debemos añadir —build para que se ejecute el dockerfile y se reconstruya la imagen:

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

Comprobamos:



Resumen:

1. Creamos el docker-compose.yml con los dos contenedores que usaremos
2. Creamos dockerfile para añadir dependencias, en este caso msqli
3. Creamos el php que se mostrará en la web para la comprobación
4. Levantamos el contenedor forzando la reconstrucción de la imagen con —build
5. Comprobamos

---

## Ejercicio 3 Docker-compose (wordpress)

Levantar un docker-compose con la imagen de MariaDB y la imagen de wordpress.

Creamos la carpeta de trabajo y luego el docker-compose.yml:

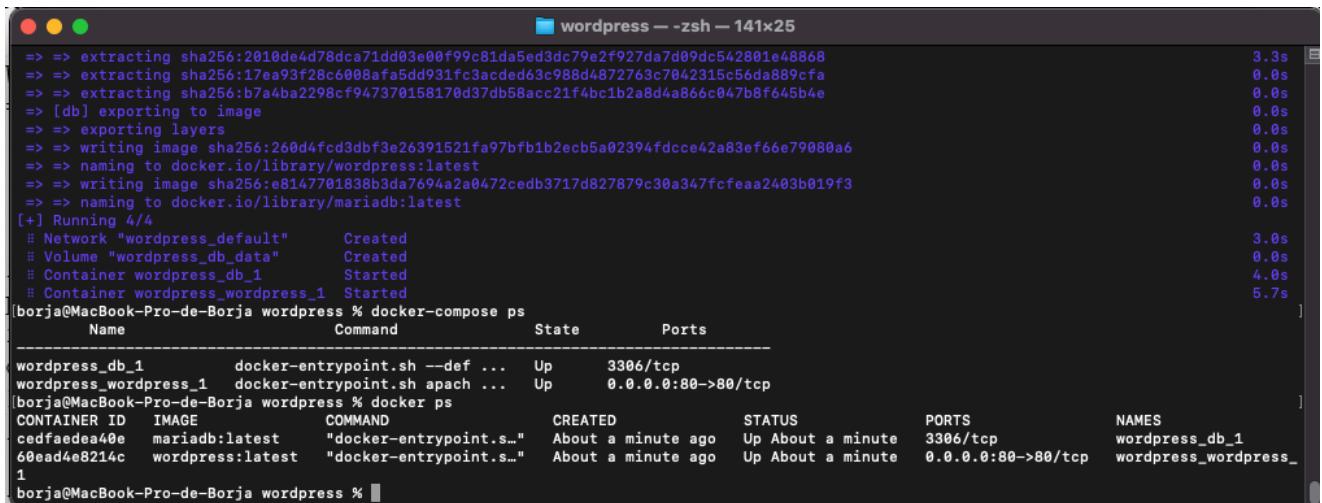
```
version: '3.3'
services:
  db:
    # Usando imagen mariadb
    image: mariadb:latest
    # Si quieres usar MySQL
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=Abcd1234.
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=Abcd1234.
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    ports:
```

```

- 80:80
restart: always
environment:
  - WORDPRESS_DB_HOST=db
  - WORDPRESS_DB_USER=wordpress
  - WORDPRESS_DB_PASSWORD=Abcd1234.
  - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:

```

Levantamos el docker:

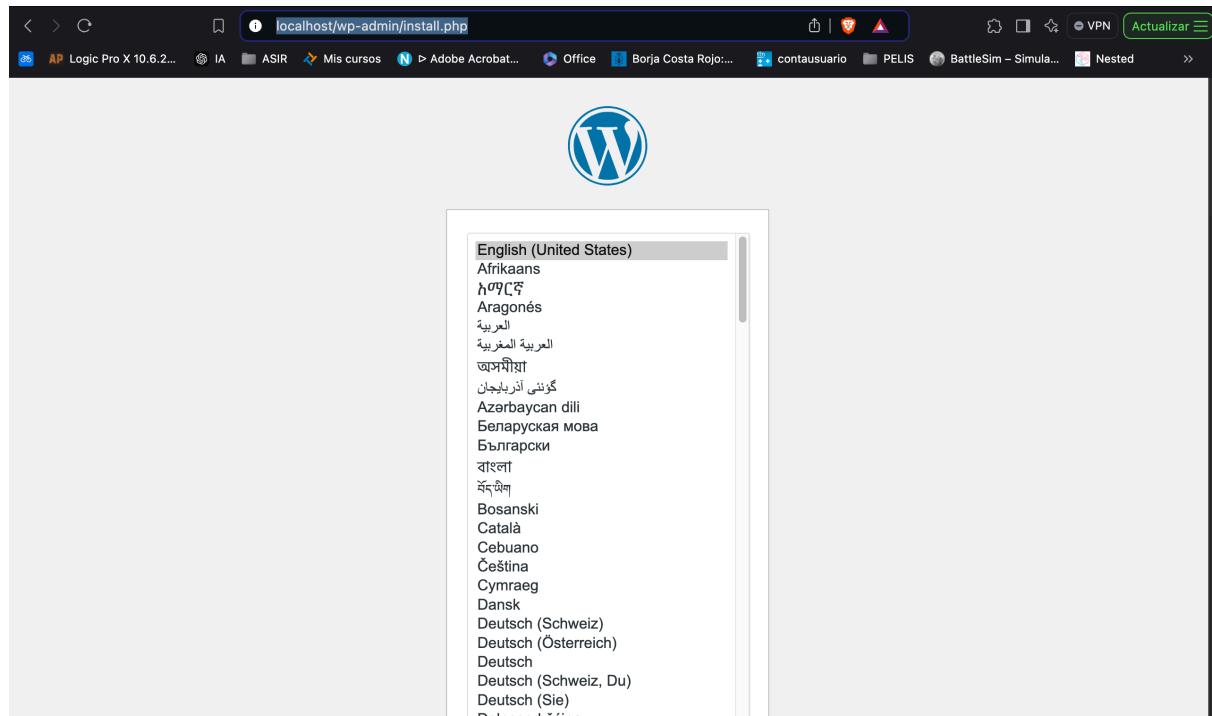


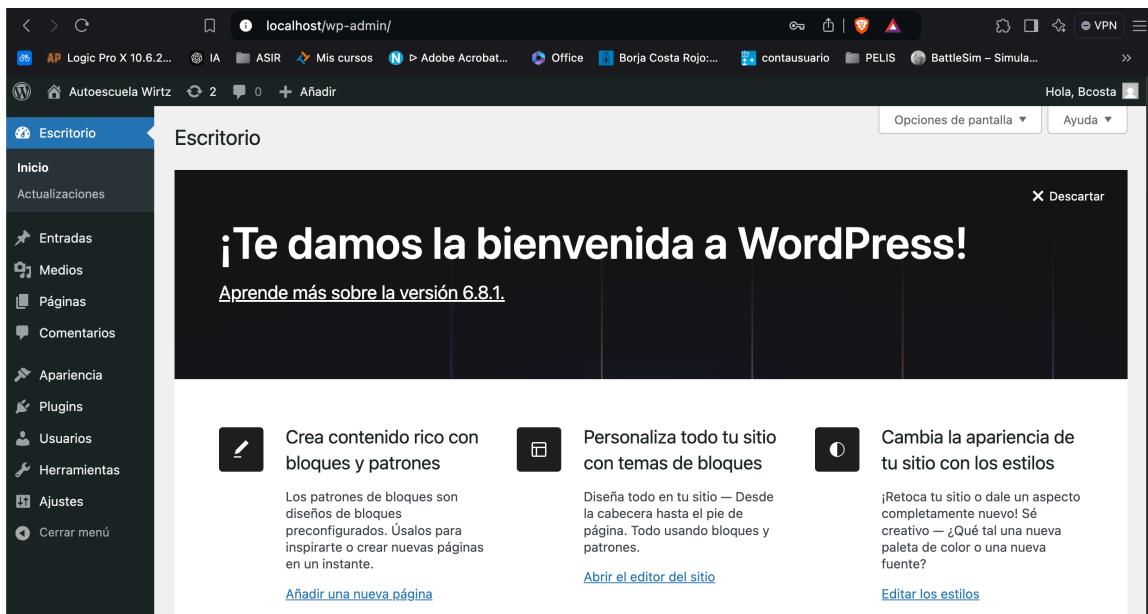
```

wordpress -- zsh - 141x25
=> => extracting sha256:2010de4d78dca71dd03e00f99c81da5ed3dc79e2f927da7d09dc542801e48868 3.3s
=> => extracting sha256:17ea93f28c6008a5dd931fc3acded63c988d4872763c7042315c56da889cfa 0.0s
=> => extracting sha256:b7a4ba2298cf947370158170d37db58acc21f4bc1b2a8d4a866c047b8f645b4e 0.0s
=> [db] exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:260d4fc3d3bf3e26391521fa97bfb1b2ecb5a02394fdcce42a83ef66e79080a6 0.0s
=> => naming to docker.io/library/wordpress:latest 0.0s
=> => writing image sha256:e8147701838b3da7694a2a0472cedb3717d827879c30a347fcfea2403b019f3 0.0s
=> => naming to docker.io/library/mariadb:latest 0.0s
[+] Running 4/4
  Network "wordpress_default"      Created          3.0s
  # Volume "wordpress_db_data"      Created          0.0s
  # Container wordpress_db_1       Started          4.0s
  # Container wordpress_wordpress_1 Started          5.7s
[borja@MacBook-Pro-de-Borja wordpress % docker-compose ps
  Name          Command          State          Ports
  wordpress_db_1  docker-entrypoint.sh --def ...  Up      3306/tcp
  wordpress_wordpress_1  docker-entrypoint.sh apach ...  Up      0.0.0.0:80->80/tcp
[borja@MacBook-Pro-de-Borja wordpress % docker ps
CONTAINER ID  IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
cedfaeeda40e  mariadb:latest  "docker-entrypoint.s..."  About a minute ago  Up About a minute  3306/tcp          wordpress_db_1
60ead4e8214c  wordpress:latest "docker-entrypoint.s..."  About a minute ago  Up About a minute  0.0.0.0:80->80/tcp  wordpress_wordpress_1
[borja@MacBook-Pro-de-Borja wordpress %

```

Comprobamos que accediendo a [localhost:80](http://localhost:80) se accede a la configuración de wordpress:





Tumbamos el docker:

```
docker-compose down -v
```

## Ejercicio 4 Docker compose (postgre y pgadmin)

Creamos la carpeta de trabajo y luego el docker-compose:

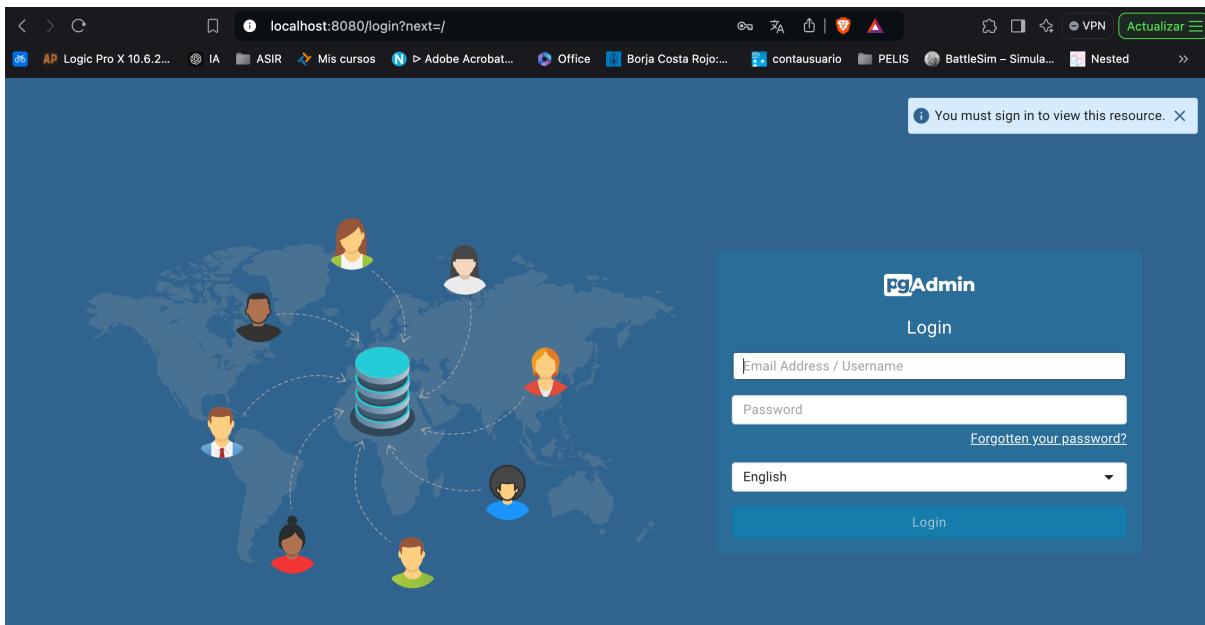
```
services:
  postgres:
    container_name: postgres_db
    image: postgres:latest
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: manuel
      POSTGRES_PASSWORD: Abcd1234.
      POSTGRES_DB: manuel
    volumes:
      - ./data:/var/lib/postgresql/data

  pgadmin:
    image: dpage/pgadmin4
    restart: always
    ports:
      - "8080:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: manuel@gmail.com
      PGADMIN_DEFAULT_PASSWORD: Abcd1234.
    depends_on:
      - postgres
```

Levantamos el docker:

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

Comprobamos:



```
pgadmin — -zsh — 141x26
=> => writing image sha256:2258423a48c0b6c20b7f0faa34c6759515e75cbda0896639b86f329268f7a92d
=> => naming to docker.io/library/postgres:latest
=> => writing image sha256:077259320cb6f83f5ea59f2c4f269d36e0d833669b26d6847eb0b63518e54a2b
=> => naming to docker.io/dpage/pgadmin4
[+] Running 1/2
 # Network "pgadmin_default" Created
 # Container postgres_db Creating
The new 'docker compose' command is currently experimental. To provide feedback or request new features please open issues at https://github.com/docker/compose-cli
Error response from daemon: invalid mount config for type "bind": bind source path does not exist: /Users/borja/pgadmin/data
[borja@MacBook-Pro-de-Borja pgadmin % docker compose up -d
[+] Running 2/2
 # Container postgres_db Started
 # Container pgadmin_pgadmin_1 Started
[borja@MacBook-Pro-de-Borja pgadmin % docker-compose ps
  Name          Command          State          Ports
-----          -----          -----          -----
pgadmin_pgadmin_1 /entrypoint.sh      Up      443/tcp, 0.0.0.0:8080->80/tcp
postgres_db      docker-entrypoint.sh postgres      Up      0.0.0.0:5432->5432/tcp
[borja@MacBook-Pro-de-Borja pgadmin % docker-compose down -v
Stopping pgadmin_pgadmin_1 ... done
Stopping postgres_db ... done
Removing pgadmin_pgadmin_1 ... done
Removing postgres_db ... done
Removing network pgadmin_default
[borja@MacBook-Pro-de-Borja pgadmin % ]
```

## Ejercicio 5 Docker-compose (MongoDB)

Creamos el docker-compose.yml:

```
version: '3'

services:
  mongodb:
    image: mongo:latest
    container_name: mongodb
    ports:
      - "27022:27017"
    environment:
      MONGO_INITDB_DATABASE: testdb
  mongo_seed:
    image: mongo:latest
    depends_on:
      - mongodb
    volumes:
```

```
- ./mongo-seed:/mongo-seed
  command:
    /mongo-seed/import.sh
```

Creamos la carpeta mongo-seed y dentro de ella el archivo [import.sh](#) donde irán las sentencias que se ejecutarán al iniciar el contenedor. En este caso se trata de un mongoimport que cogerá los datos de un archivo json:

```
mongoimport --host mongodb --db testdb --collection properties --type json --
  -file /mongo-seed/data.json --jsonArray
```

Creamos el archivo json con los datos que se añadirán a la base de datos en mongo:

```
[
  {
    "propertyType": "House",
    "bedrooms": 3,
    "created_by": 1
  },
  {
    "propertyType": "Apartment",
    "bedrooms": 2,
    "created_by": 2
  },
  {
    "propertyType": "Condo",
    "bedrooms": 1,
    "created_by": 3
  },
  {
    "propertyType": "Townhouse",
    "bedrooms": 4,
    "created_by": 4
  },
  {
    "propertyType": "Duplex",
    "bedrooms": 2,
    "created_by": 5
  },
  {
    "propertyType": "House",
    "bedrooms": 5,
    "created_by": 6
  },
  {
    "propertyType": "Apartment",
    "bedrooms": 1,
    "created_by": 7
  },
  {
    "propertyType": "Condo",
    "bedrooms": 3,
    "created_by": 8
  },
  {
    "propertyType": "Townhouse",
    "bedrooms": 4,
    "created_by": 9
  },
  {
    "propertyType": "Duplex",
    "bedrooms": 2,
    "created_by": 10
  }
]
```

Ahora levantamos los contenedores con docker-compose:

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

```
mongodb -- mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 -- com.docker.cli < docker exec -it...
```

```
borja@MacBook-Pro-de-Borja ~ % mkdir mongodb
borja@MacBook-Pro-de-Borja ~ % cd mongodb
borja@MacBook-Pro-de-Borja mongodb % nano docker-compose.yml
borja@MacBook-Pro-de-Borja mongodb % mkdir mongo-seed
borja@MacBook-Pro-de-Borja mongodb % cd mongo-seed
borja@MacBook-Pro-de-Borja mongo-seed % nano import.sh
borja@MacBook-Pro-de-Borja mongo-seed % nano data.json
borja@MacBook-Pro-de-Borja mongo-seed % cd ..
borja@MacBook-Pro-de-Borja mongodb % docker-compose up -d
Docker Compose is now in the Docker CLI, try `docker compose up`
```

```
[Creating network "mongodb_default" with the default driver
Pulling mongodb (mongo:latest)...
latest: Pulling from library/mongo
0622fac788ed: Already exists
c1aa6a7c13c0: Pull complete
82a30a244416: Pull complete
d2c82d0518f8: Pull complete
ab6be4e21c66: Pull complete
f6ae99e07c30: Pull complete
ca015e545994: Pull complete
54db8e3d8732: Pull complete
Digest: sha256:9f67b6bafda002f7bcad9939e4d84c3b4e9b11fffff6c1f9fab3f77e30c646304
Status: Downloaded newer image for mongo:latest
Creating mongodb ... done
Creating mongodb_mongo_seed_1 ... done
borja@MacBook-Pro-de-Borja mongodb % docker-compose ps
```

Name	Command	State	Ports
mongodb	docker-entrypoint.sh mongod	Up	0.0.0.0:27022->27017/tcp
mongodb_mongo_seed_1	docker-entrypoint.sh /mong ...	Exit 126	

Comprobamos:

```
mongodb -- mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 -- -zsh -- 139x54
borja@MacBook-Pro-de-Borja mongodb % docker exec -it mongodb mongosh
Current Mongosh Log ID: 6825ab3c03c132a6d0d861df
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.0
Using MongoDB:     8.0.9
Using Mongosh:     2.5.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
  The server generated these startup warnings when booting
  2025-05-15T08:52:04.500+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine.

  See http://dochub.mongodb.org/core/prodnotes-filesystem
  2025-05-15T08:52:05.993+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2025-05-15T08:52:05.993+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
  2025-05-15T08:52:05.993+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
  2025-05-15T08:52:05.993+00:00: We suggest setting the contents of sysfsFile to 0.
  2025-05-15T08:52:05.993+00:00: vm.max_map_count is too low
  2025-05-15T08:52:05.994+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.

-----
test> show dbs
admin   8.00 KiB
config  12.00 KiB
local   8.00 KiB
testdb  8.00 KiB
test> use testdb
switched to db testdb
testdb> show collections
properties
testdb> db.properties.find().limit(1)
[ {
  _id: ObjectId('6825ab37e1070a6dcbaaa293'),
  propertyType: 'Apartment',
  bedrooms: 2,
  created_by: 2
}
]
testdb> exit
borja@MacBook-Pro-de-Borja mongodb % docker-compose down -v
Stopping mongodb ... done
Removing mongodb_mongo_seed_1 ... done
Removing mongodb ... done
Removing network mongodb_default
borja@MacBook-Pro-de-Borja mongodb %
```

## Ejercicio 6 Docker-compose (Mongodb-webapp)

La idea es crear un contenedor con mongodb que ejecute un script de inicio y crear otro contenedor una app que interactue con la base de datos.

Creamos el directorio de trabajo:

```
mkdir mongoapp
cd mongoapp
```

Creamos el archivo .yml con los dos contenedores y la persistencia de datos:

```
version: '3.8'

services:
  mongo:
    image: mongo:6.0
    container_name: mongodb
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
      - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh
      - ./init.js:/docker-entrypoint-initdb.d/init.js

  webapp:
    build: ./webapp
    ports:
      - "5000:5000"
    environment:
      MONGO_URI: "mongodb://root@example@mongodb:27017"
    depends_on:
      - mongo

volumes:
  mongo_data:
```

Creamos el .sh para añadir el script de inicio:

```
#!/bin/bash

echo "Ejecutando script de inicialización..."
sleep 5 # Esperar a que MongoDB esté listo

mongo --username root --password example --authenticationDatabase admin
<<EOF
use testdb
db.createUser({
  user: "appuser",
  pwd: "Abcd1234",
  roles: [ { role: "readWrite", db: "testdb" } ]
})
EOF

echo "Inicialización completada."
```

Creamos la carpeta webapp y dentro de ella el dockerfile:

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY .
CMD ["python", "app.py"]
```

Creamos dentro también el [app.py](#) que es la propia aplicación web:

```
from flask import Flask, jsonify
from pymongo import MongoClient
import os

app = Flask(__name__)

# Conexión a MongoDB usando las variables de entorno
mongo_uri = os.getenv("MONGO_URI")
client = MongoClient(mongo_uri)
db = client.testdb

@app.route('/')
def hello():
    return "¡Hola desde el contenedor Flask!"

@app.route('/users')
def get_users():
    users = list(db.users.find({}, {'_id': 0}))
    return jsonify(users)

@app.route('/add/<name>')
def add_user(name):
    db.users.insert_one({"name": name})
    return f"Usuario {name} añadido"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

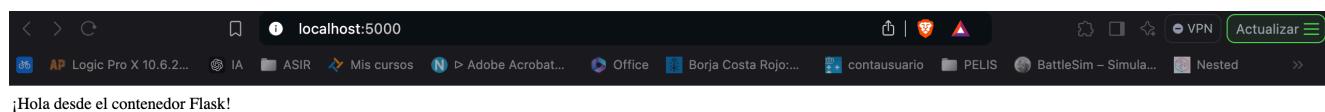
Levantamos los contenedores:

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

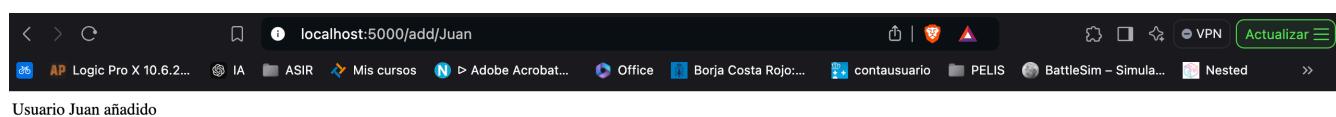
Nos da fallo y nos está diciendo que no encuentra el archivo requirements.txt, este archivo lista todas las dependencias de python que necesita la app para funcionar, lo creamos dentro de la carpeta webapp:

```
flask
pymongo
```

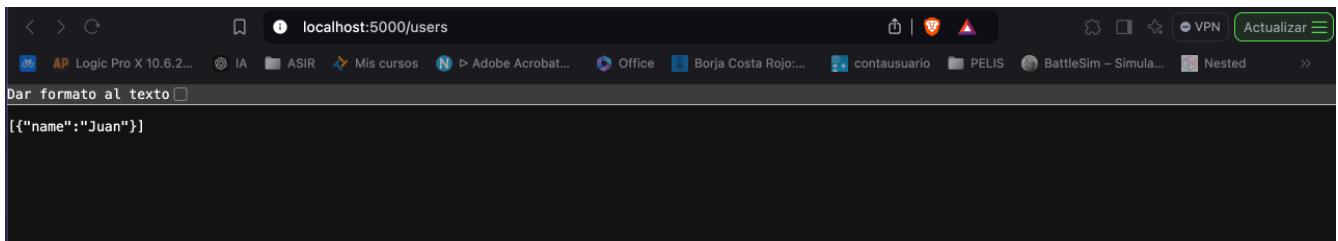
Levantamos de nuevo y ahora tendremos la webapp escuchando en el puerto 5000, vamos a probarlo:



Ahora si queremos añadir un usuario podemos acceder a la dirección:  
<http://localhost:5000/add/Juan>



Y si queremos listar los usuarios: <http://localhost:5000/users>



```
[{"name": "Juan"}]
```

Tendriamos el contenedor de Mongodbd escuchando por el puerto 27017.

Finalizamos tumbando los contenedores:

```
docker-compose down -v
```

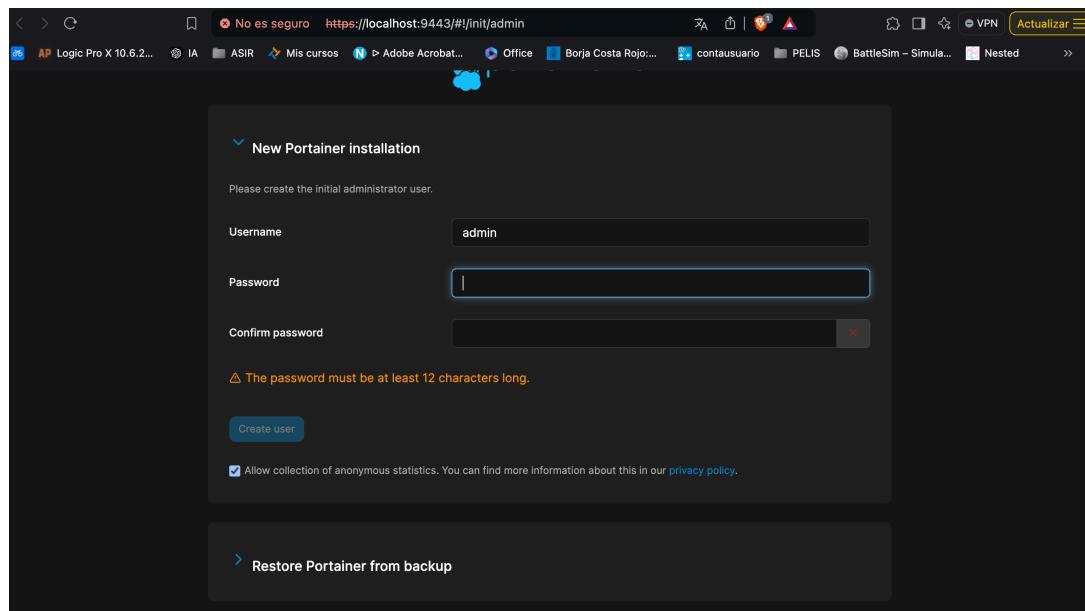
## Portainer

**Portainer** es una herramienta de gestión visual para Docker y Kubernetes que simplifica la administración de contenedores, imágenes, redes y volúmenes a través de una interfaz web intuitiva. Es ideal para usuarios que prefieren una GUI en lugar de la terminal.

La forma mas sencilla de usar portainer es ejecutarlo como contenedor:

```
docker run -d \
  -p 9443:9443 \
  --name portainer \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  portainer/portainer-ce:latest
```

Ahora accedemos a la interfaz web desde <https://localhost:9443> , en cuanto accedemos nos pide que creamos el usuario:



Una vez dentro vemos un interfaz muy cómodo:

Desde aquí tenemos un control total sobre contenedores e imágenes, consumo de recursos y todo bien organizado.

The screenshot shows the Portainer.io interface. On the left, a sidebar menu includes Home, Dashboard (selected), Templates, Stacks, Containers, Images, Networks, Volumes, Events, and Host. The main area displays an 'Environment summary' with details: Environment (local), URL (/var/run/docker.sock), GPU (none), and Tags (-). Below this are four summary cards: 0 Stacks, 2 Containers (1 running, 1 stopped, 0 healthy, 0 unhealthy), 17 Images (11.6 GB), and 3 Volumes.

The screenshot shows the 'Container list' page. The sidebar menu is identical to the previous screenshot. The main area lists two containers: 'nervous\_hertz' (exited - code 255) and 'portainer' (running). The table columns are: Name, State, Quick Actions, Stack, Image, Created, IP Address, and Pub. The 'portainer' container is highlighted in green.

**Podemos crear un contenedor**, hacemos clic en *"Containers" > "Add container"*:

- Nombre: mi-nginx
- Imagen: nginx:latest
- Mapeo de puertos: 8080:80

Para finalizar hacemos clic en *"Deploy the container"*.

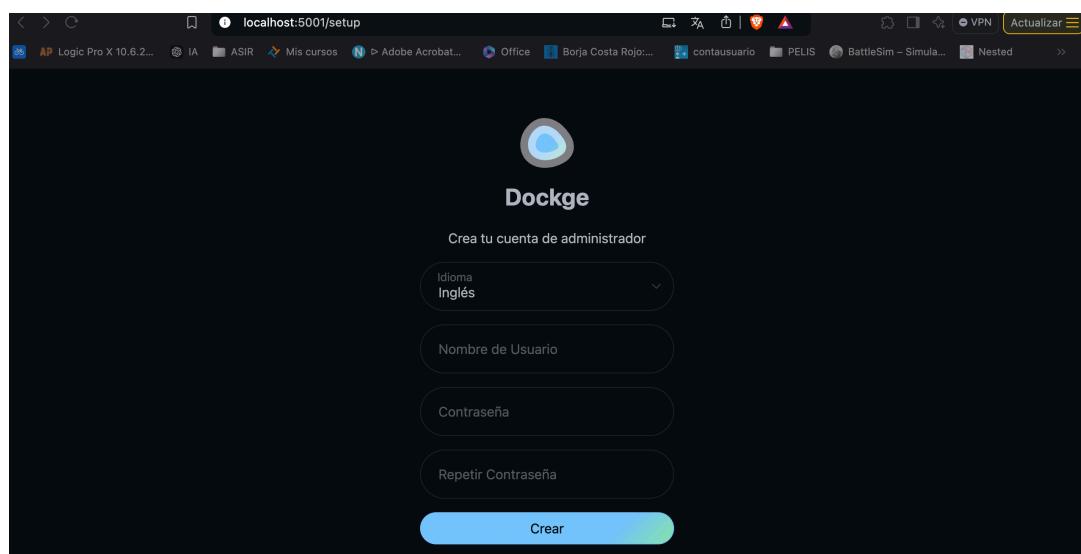
## Dockge

**Dockge** es un administrador visual para Docker basado en web, diseñado para simplificar el manejo de contenedores y stacks (grupos de servicios definidos en `docker-compose.yml`). Es una alternativa ligera y moderna a herramientas como Portainer, ideal para usuarios que buscan una interfaz sencilla y enfocada en la gestión de proyectos con Docker Compose.

Lo instalamos como contenedor, igual que hemos hecho con Portainer:

```
docker run -d \
--name dockge \
--restart=unless-stopped \
-p 5001:5001 \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /Users/borja/stacks:/opt/stacks \
-v dockge_data:/data \
louislam/dockge:latest
```

Ahora accedemos a la interfaz desde **http://localhost:5001**, igual que con portainer lo primero que nos pedirá es que creemos el usuario:



Como podemos ver, nos permite crear contenedores, editarlos y gestionarlos.

