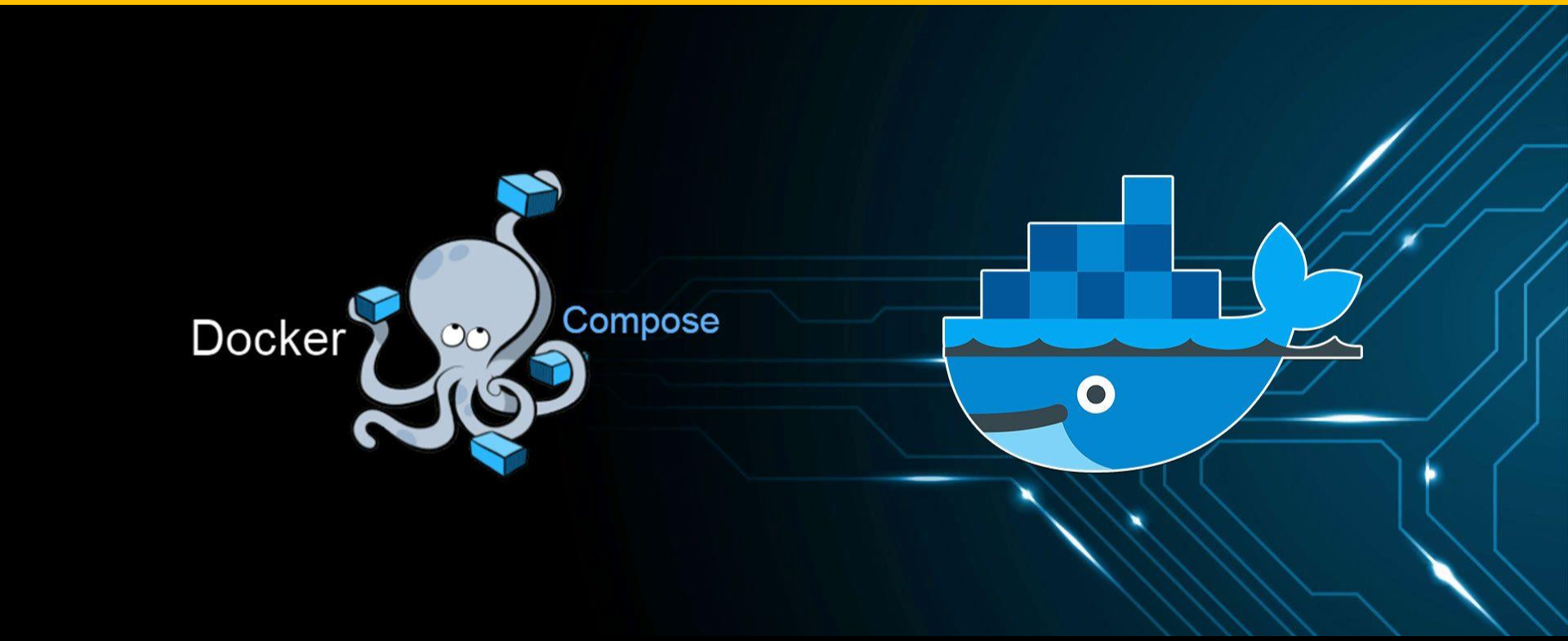
IES SAN ANDRÉS

ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL Y BIG DATA



CONTENEDORES CON

DOCKER

ÍNDICE

1.- Introducción a Docker

2.- Instalación de Docker Desktop

3.- Imágenes y contenedores

4.- Creación y gestión de imágenes

5.- Volúmenes en Docker

6.- Redes en Docker

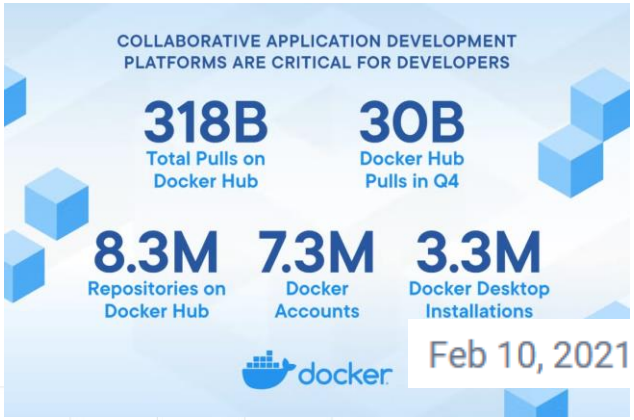
7.- Docker Compose

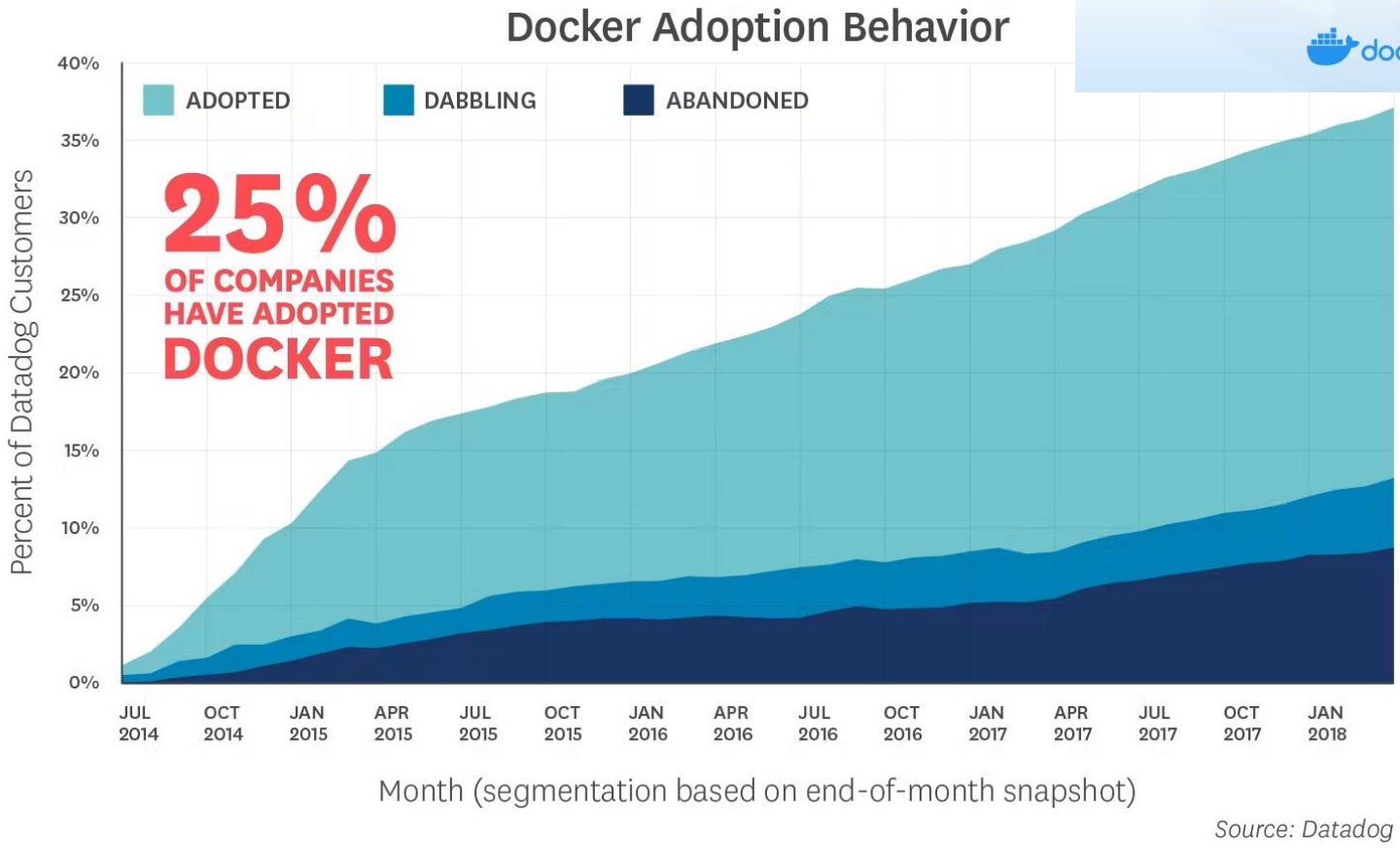
1

INTRODUCCIÓN A DOCKER

Docker es una plataforma de contenedores que permite a los desarrolladores y administradores de sistemas crear, empaquetar y ejecutar aplicaciones de manera portátil, ligera y escalable.

Es una tecnología madura ampliamente implantada y en continuo crecimiento.



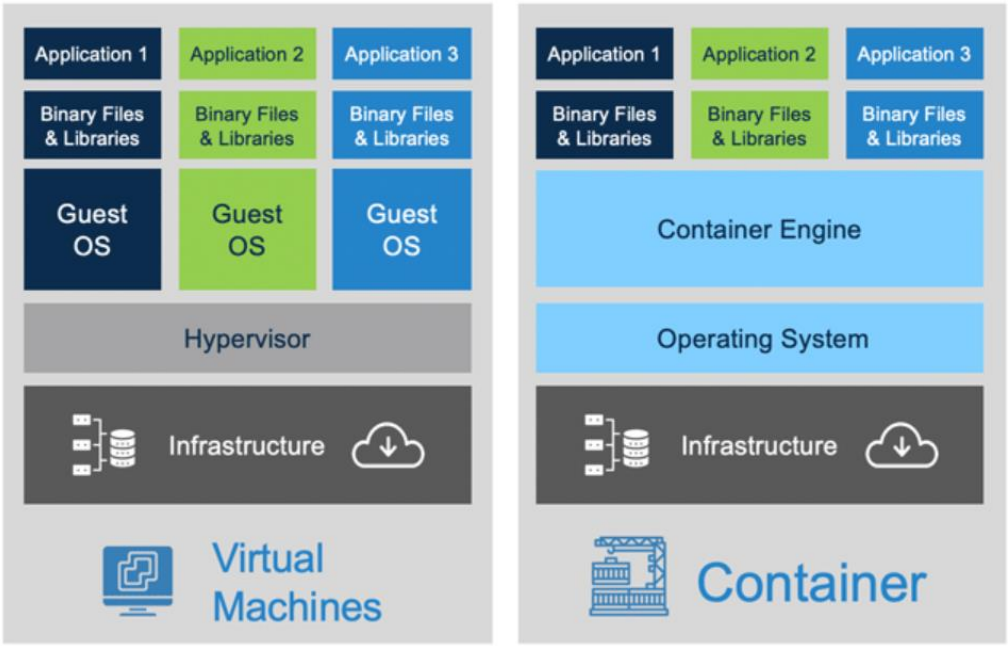


Source: Datadog

Aunque tiene similitudes con la virtualización, en realidad es un concepto totalmente diferente.

Virtualización: las máquinas virtuales proporcionan un hardware virtual sobre el que se instala un sistema operativo.

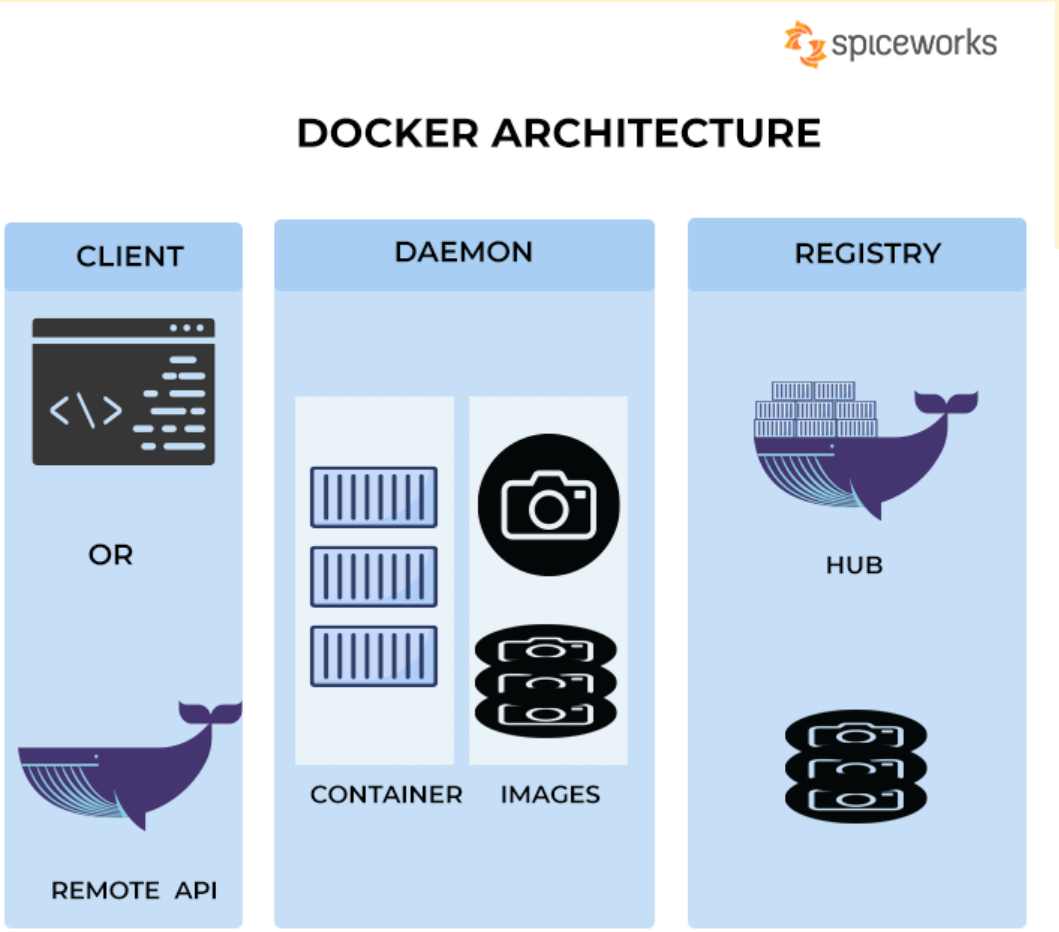
Contenerización: los contenedores son **entornos aislados** que contienen aplicaciones que acceden directamente al kernel del host.



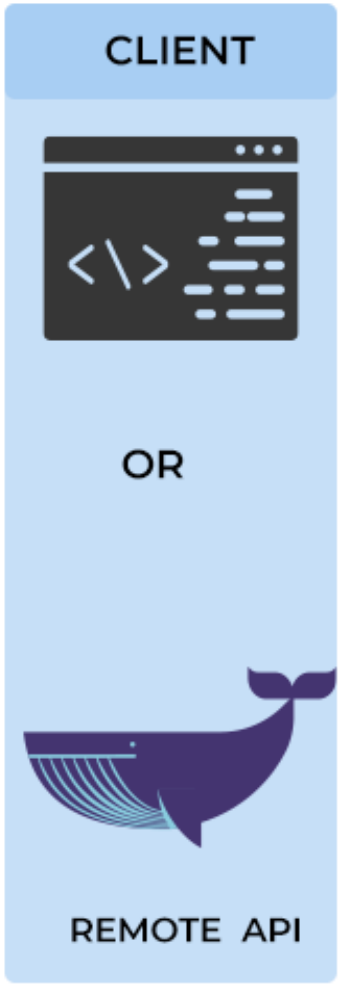
	Docker	Máquinas virtuales
Uso de recursos	Ligero	Alto
Tiempo de arranque	Segundos	Minutos
Aislamiento	Moderado, comparte kernel	Total, cada VM tiene su propio kernel
Portabilidad	Alta	Menos flexible, depende del hipervisor
Seguridad	Menos seguro, comparte recursos del host	Más seguro, aislamiento completo
Facilidad de despliegue	Rápido y ágil	Complejo y lento
Persistencia	Requiere configuración	Soporte nativo
Densidad	Alta, muchos contenedores por equipo	Baja
Uso en producción	Ideal para microservicios, DevOps y desarrollo y pruebas	Adecuado para entornos más tradicionales

La **arquitectura de Docker** se basa en un modelo cliente-servidor con tres componentes clave:

- **Cliente**
- **Demonio de Docker**
- **Registros de Docker**



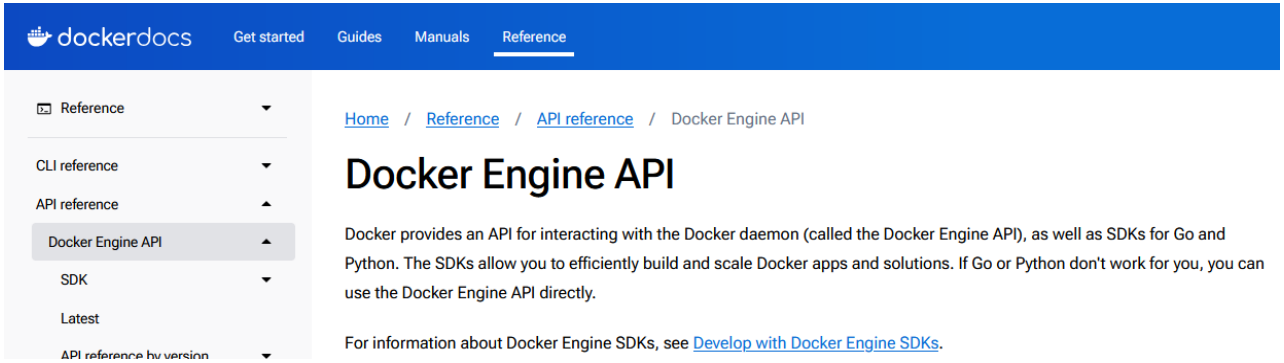
Cliente Docker | Demonio Docker | Registro de Docker



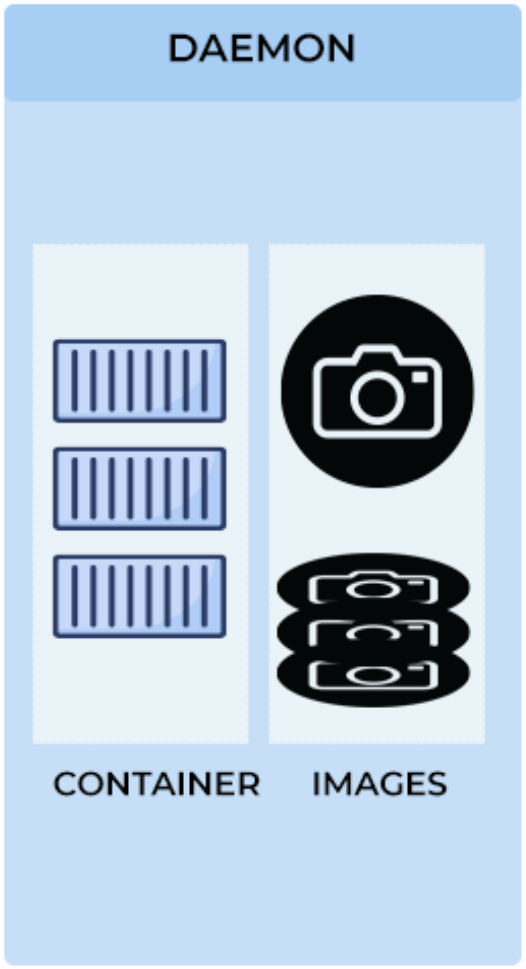
Docker CLI es la **interfaz de línea de comandos** que los usuarios utilizan para interactuar con Docker.

Envía comandos al Docker Deamon a través de la API de Docker

También es posible interactuar directamente con el Docker Deamon a través de una **API RESTful**



Cliente Docker | **Demonio Docker** | Registro de Docker



El **demonio de Docker** es el componente central de Docker, responsable de gestionar contenedores, imágenes, redes y volúmenes.

Se ejecuta en segundo plano en el sistema anfitrión.

Hay dos conceptos clave en Docker:

- Imágenes
- Contenedores

Cliente Docker | **Demonio Docker** | Registro de Docker

Imágenes | Contenedores

Una **imagen** es una plantilla de **solo lectura** que contiene todo lo necesario para ejecutar una aplicación:

- Un **sistema operativo** base (Ubuntu, Alpine, CentOS)
- Las **dependencias** necesarias para ejecutar la aplicación (p.e. Python, Node.js, …)
- El **código** de la aplicación
- **Configuraciones** y scripts de inicio

Las imágenes son **inmutables.** Una vez creadas no se puede modificar, se debe crear una nueva imagen.

Cliente Docker | **Demonio Docker** | Registro de Docker

Imágenes | Contenedores

Las imágenes se pueden **obtener de un registro** de Docker, o bien se pueden construir mediante código con un fichero denominado **Dockerfile**, que contiene las instrucciones para construir dicha imagen.

Dockerfile

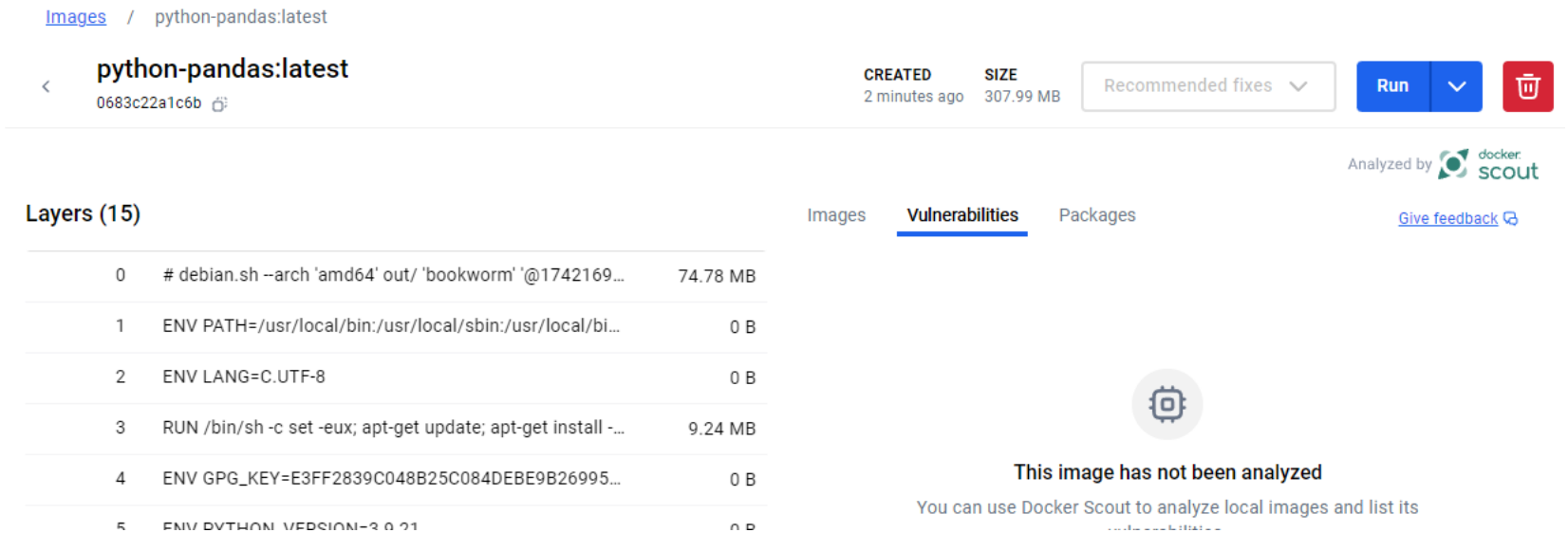
docker > Dockerfile > ...

```
1   FROM python:3.9-slim
2   WORKDIR /app
3   COPY requirements.txt .
4   RUN pip install -r requirements.txt
5   COPY . .
6   CMD ["python", "app.py"]
7
```

Cliente Docker | **Demonio Docker** | Registro de Docker

Imágenes | Contenedores

Una característica importante de las imágenes es que están **compuestas por capas**, cada una de las cuales representa un cambio en el sistema de archivos.



Cliente Docker | **Demonio Docker** | Registro de Docker

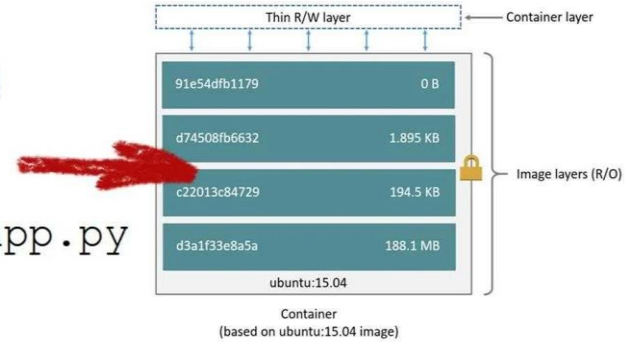
Imágenes | Contenedores

Todas las imágenes se construyen a partir de una imagen anterior (instrucción `FROM` del Dockerfile)

Cada instrucción en el Dockerfile (como `RUN`, `COPY` o `ADD`) crea una nueva capa

Dockerfile

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

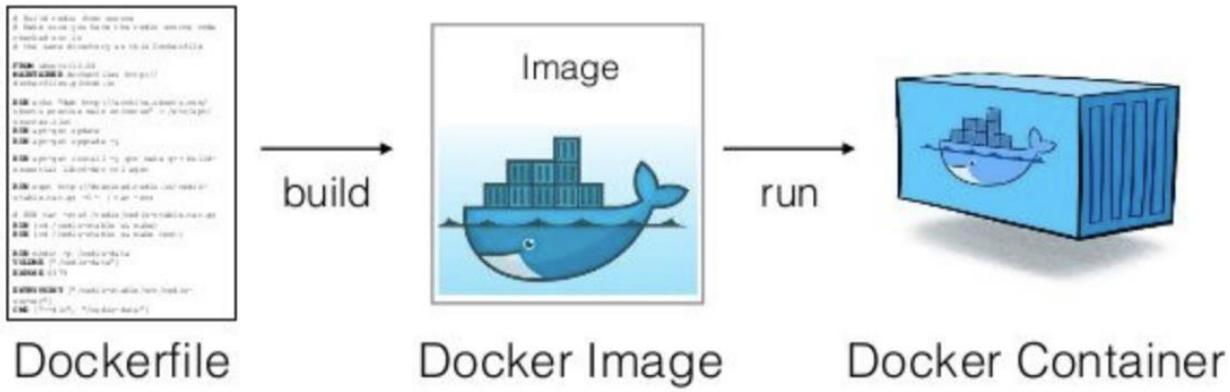


Ventaja: si varias imágenes tienen como base una misma imagen (p.e. Alpine), esta solo se descarga una vez y cada imagen solo ocupa lo que tenga de diferente respecto a la imagen base.

Cliente Docker | **Demonio Docker** | Registro de Docker

Imágenes | **Contenedores**

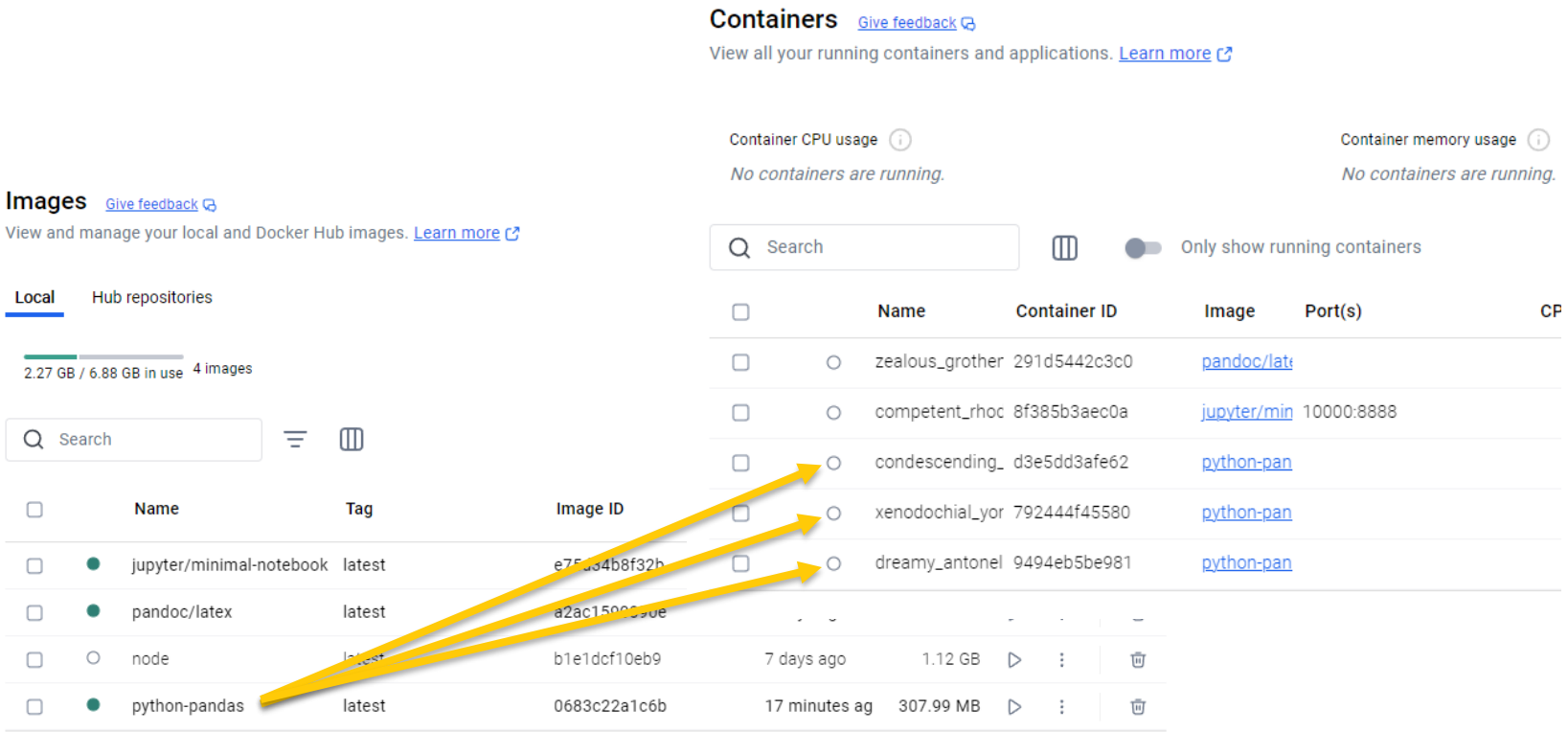
Los **contenedores** en Docker son **instancias en ejecución** de una imagen de Docker.



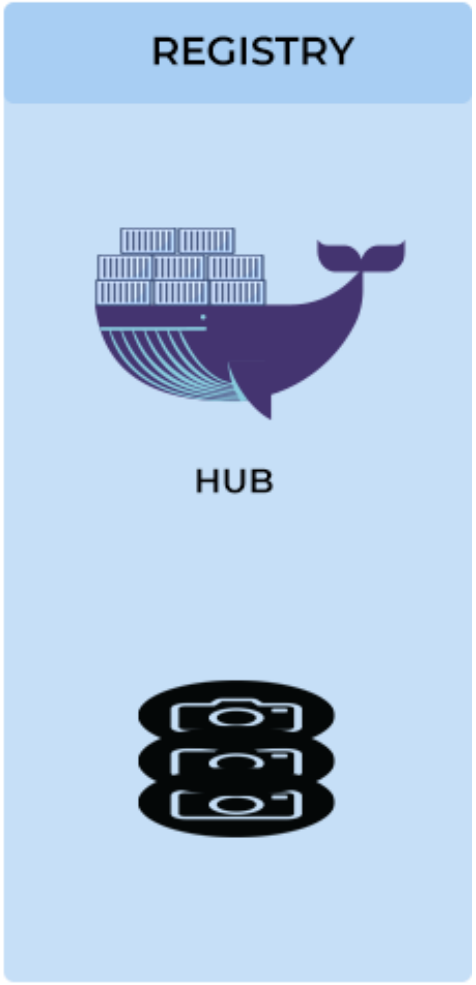
Cliente Docker | **Demonio Docker** | Registro de Docker

Imágenes | **Contenedores**

Ejemplo: de la imagen *python-pandas* he instanciado 3 contenedores



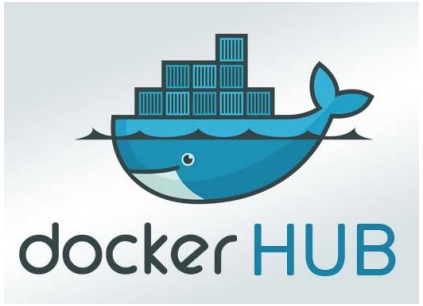
Cliente Docker | Demonio Docker | **Registro de Docker**



El **registro de Docker** es un repositorio donde se almacenan y comparten imágenes de Docker.

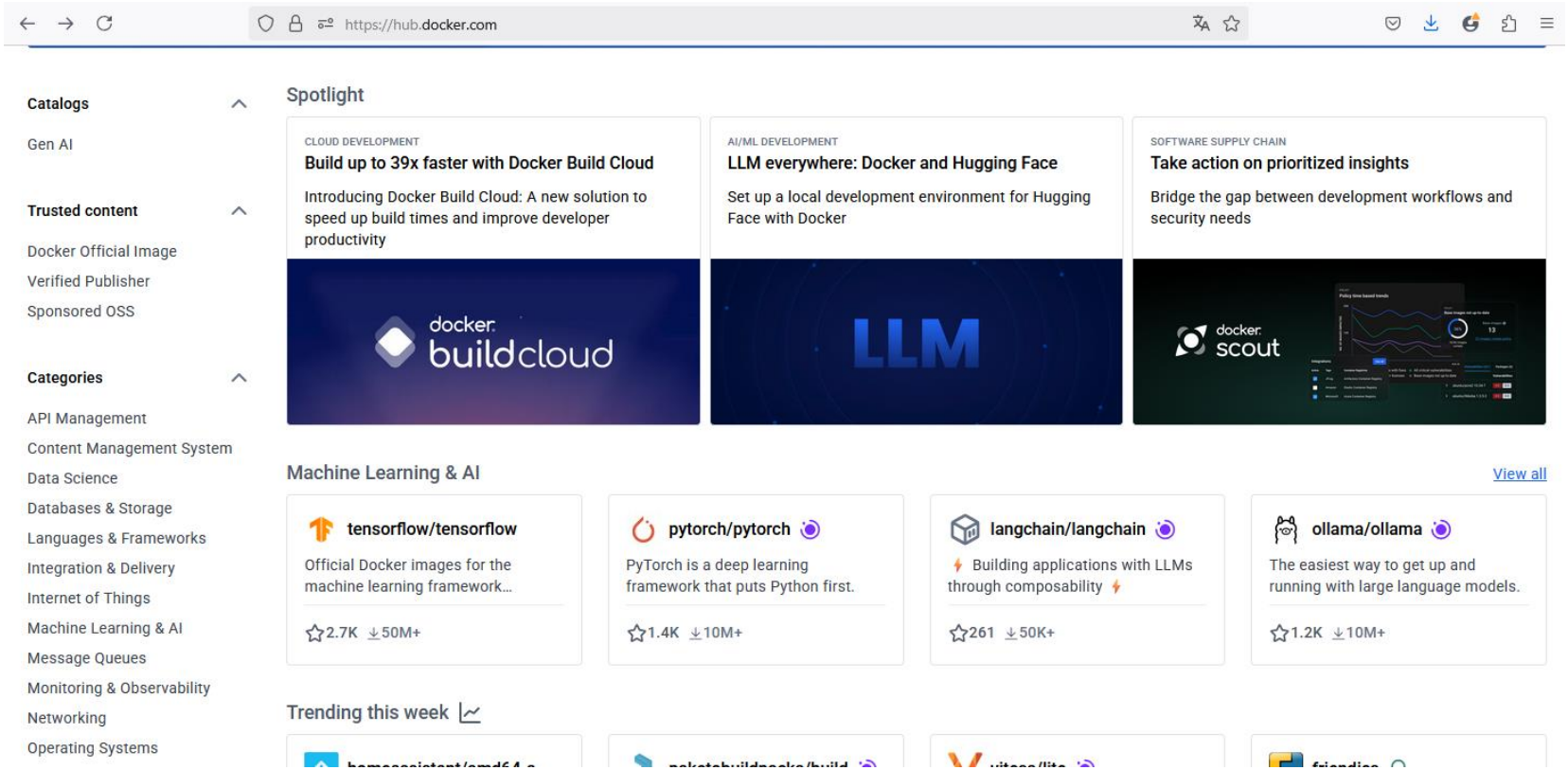
Permite **descargar imágenes** preconstruidas y compartir imágenes propias.

El registro público más conocido es **Docker Hub**, pero también existen registros privados como *Amazon ECR*, *Google Container Registry* o *Azure Container Registry*.



Cliente Docker | Demonio Docker | **Registro de Docker**

Podemos ver las imágenes disponibles en Docker Hub en la URL https://hub.docker.com/



2

INSTALACIÓN DE DOCKER DESKTOP

La aplicación de Docker para Windows se denomina **Docker Desktop**, y se puede obtener en https://www.docker.com/products/docker-desktop/

INSTALACIÓN DE DOCKER

Víctor J. González

Requisitos hardware para instalar Docker Desktop:

- CPU con juego de instrucciones de virtualización (Intel VT-x o AMD-V)
- Procesador de 64 bits
- Mínimo 4 GB de RAM (recomendado 8 GB)
- Mínimo 20 GB de espacio en disco (recomendado 40 GB o más)

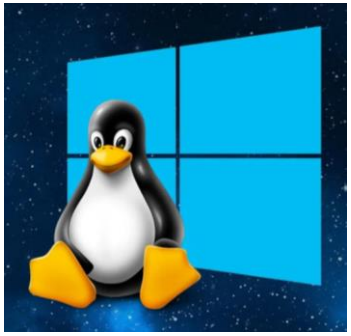
Requisitos software:

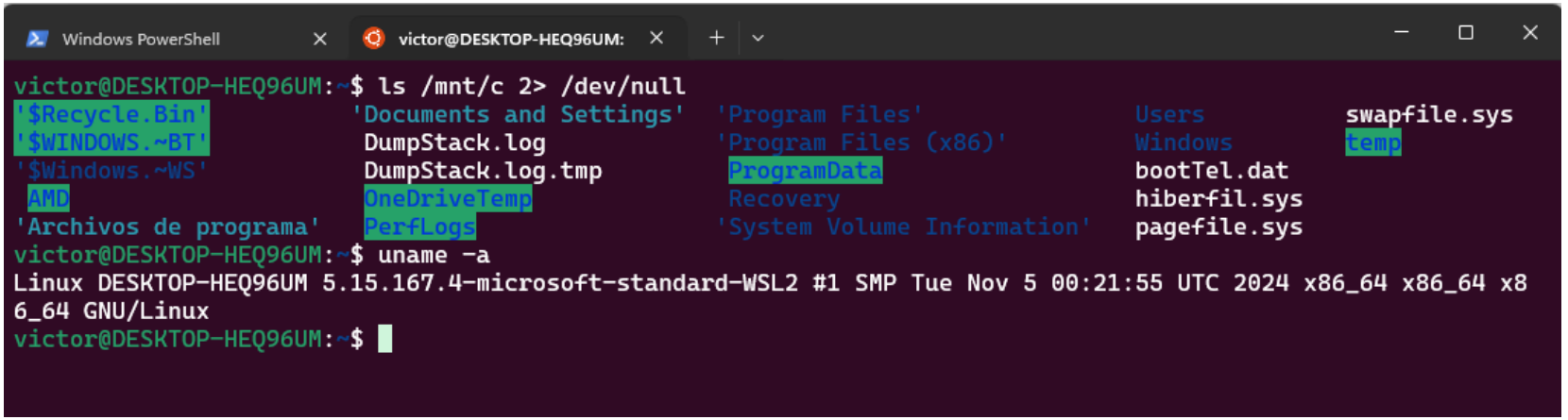
- Windows 10 64-bit Home o Pro versión 22H2
- Windows 11 64-bit Home o Pro versión 22H2
- WSL 2 o Hyper-V

INSTALACIÓN DE DOCKER

Víctor J. González

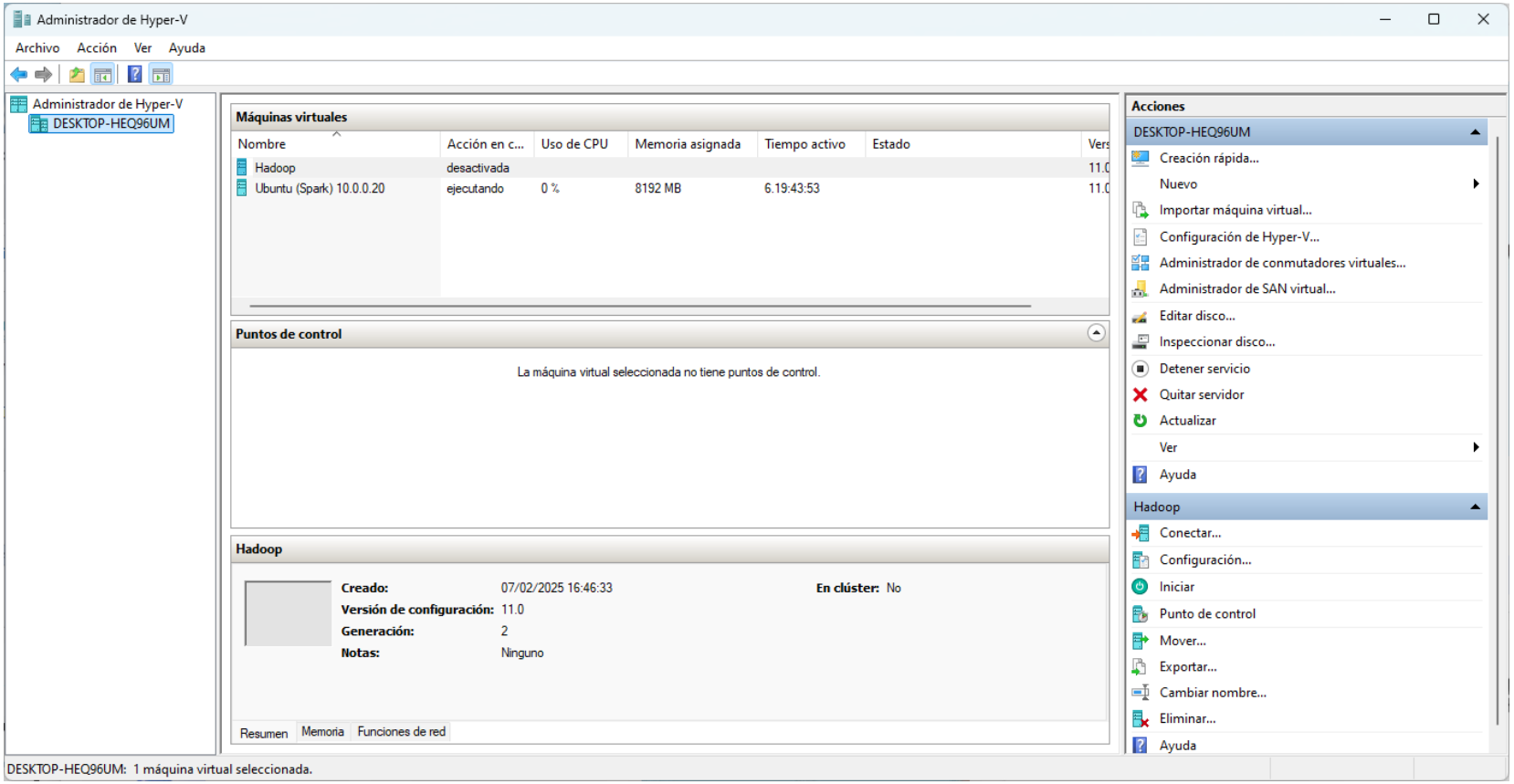
WSL 2 (Subsistema de Windows para Linux) es una capa de compatibilidad desarrollada por Microsoft para ejecutar binarios de Linux de forma **nativa**.





Para habilitar WSL 2 solo hay que ejecutar `wsl --install` en una terminal de Windows

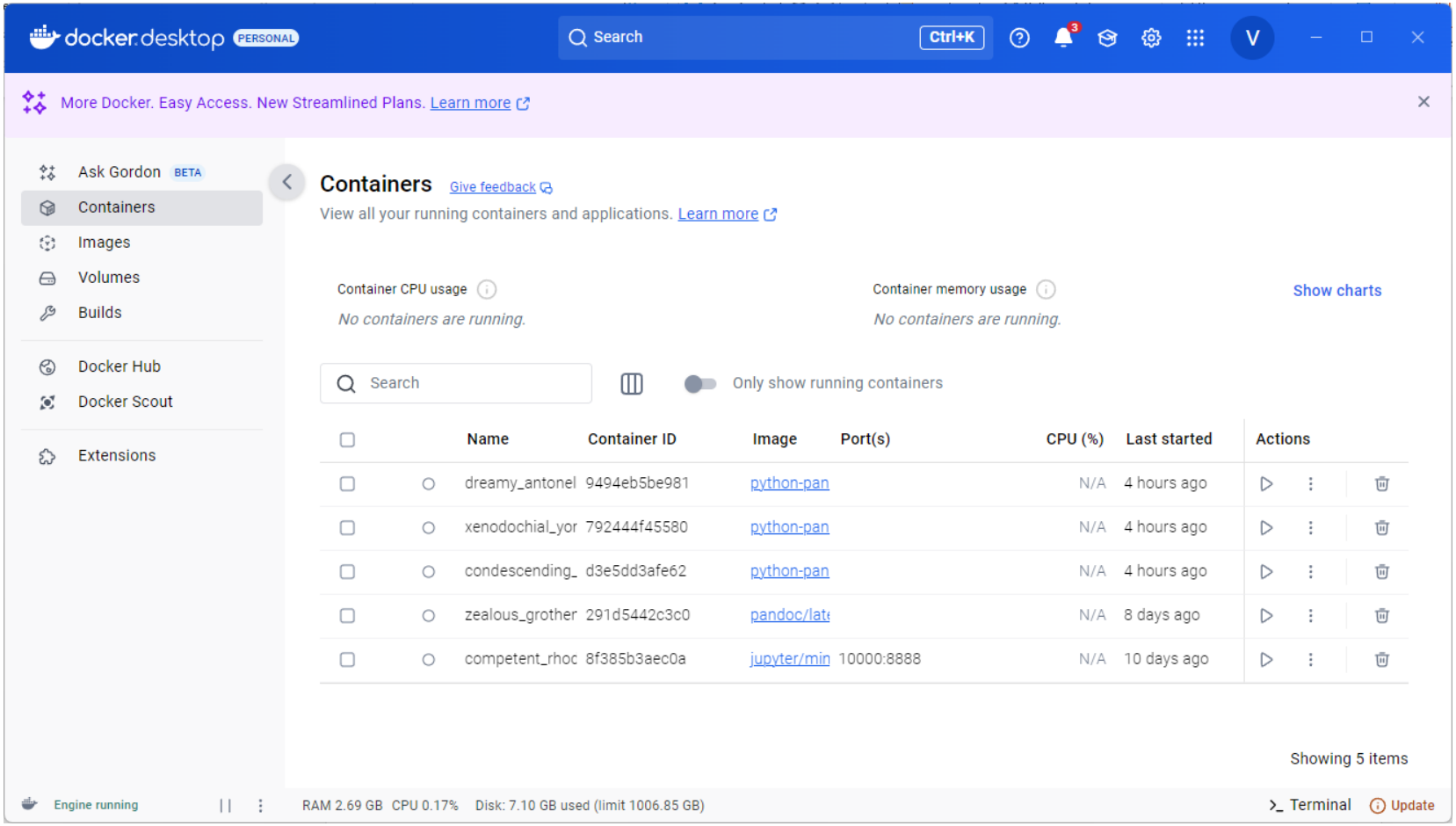
Hyper-V es el hipervisor de tipo 1 (*bare metal*) de Microsoft disponible en Windows 10/11 versión Pro, así como en las versiones de servidor.



¿Cuál de las dos opciones escoger? Realmente hay poca diferencia, aunque hay dos casos que nos obligarían a una opción u otra:

- Hyper-V no está disponible en la versión Home de Windows
- WSL 2 no está disponible en Windows Server

La instalación es muy sencilla. Una vez finalizada tendremos la **interfaz gráfica** disponible en nuestro equipo



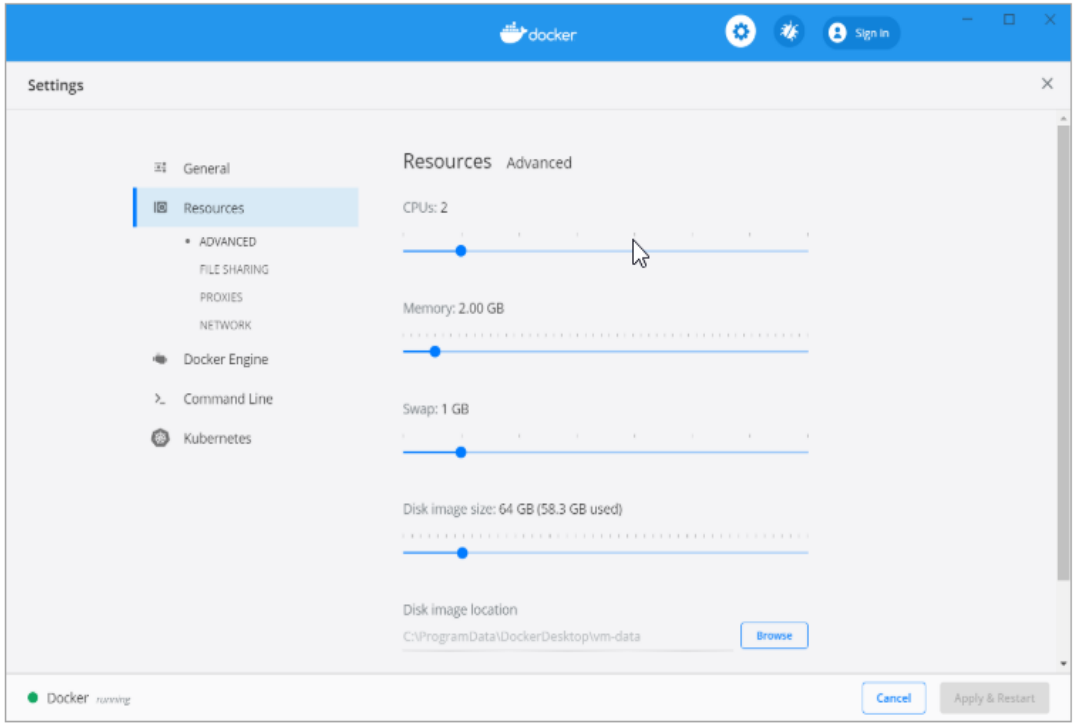
Aunque también podremos interactuar con Docker mediante la **línea de comandos**

```
PS G:\proyectos\docker> docker version
Client:
 Version:           28.0.1
 API version:       1.48
 Go version:        go1.23.6
 Git commit:        068a01e
 Built:             Wed Feb 26 10:41:52 2025
 OS/Arch:           windows/amd64
 Context:           desktop-linux

Server: Docker Desktop 4.39.0 (184744)
 Engine:
  Version:          28.0.1
  API version:      1.48 (minimum version 1.24)
  Go version:       go1.23.6
  Git commit:       bbd0a17
```

Hay algunas tareas de configuración que podemos realizar después de instalarlo.

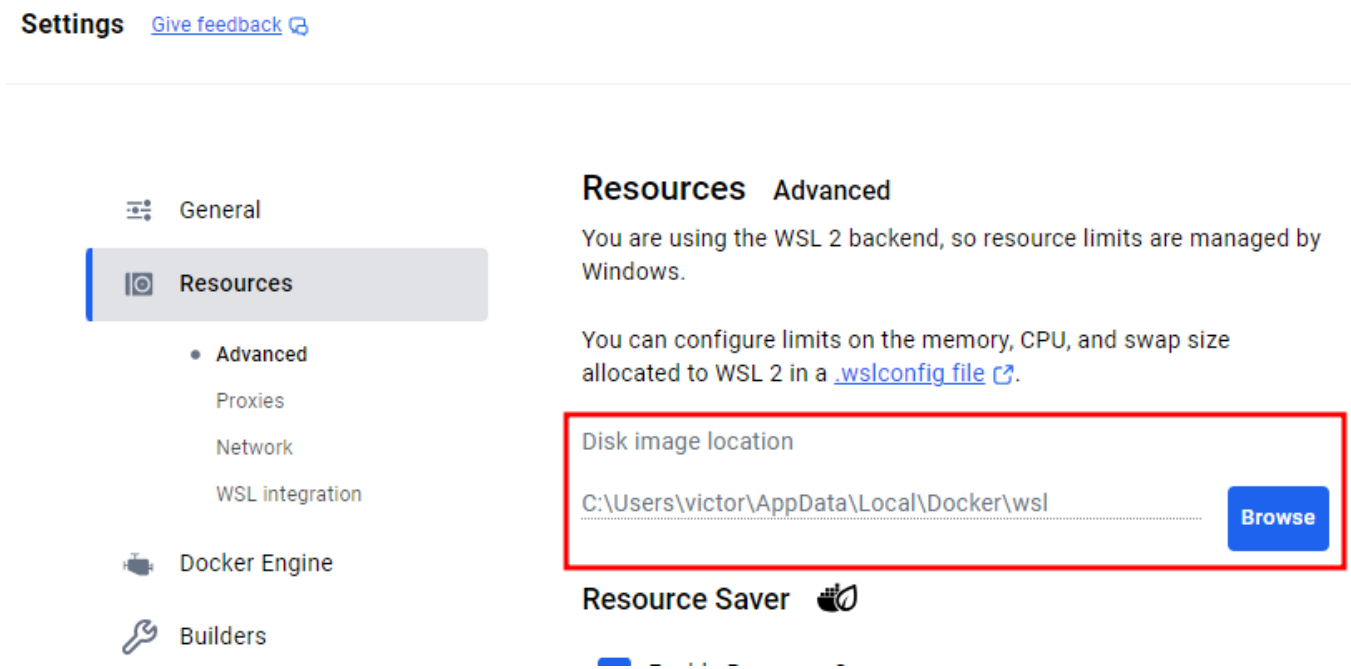
Configurar el uso de recursos

Si usamos Hyper-V como *backend* de Docker, lo que hace es crear una máquina virtual que será la que ejecute el demonio Docker. Podemos configurar los recursos de esta máquina en *Settings -> Resources*



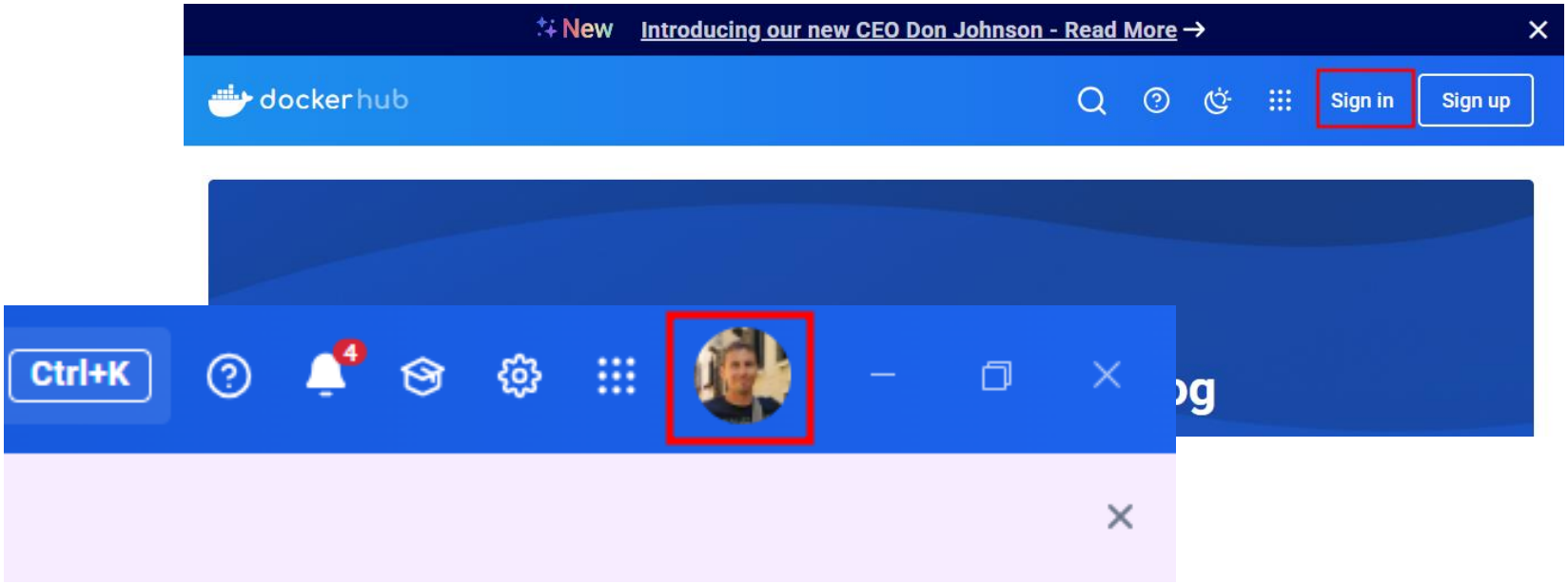
Configurar el almacenamiento

También en *Settings* -> *Resources* -> *Advanced* podemos cambiar el directorio donde se ubicarán los datos de Docker (imágenes, volúmenes, …)



Registro en Docker Hub

Si queremos acceder a toda la funcionalidad de Docker Hub, por ejemplo, para subir nuestras imágenes y compartirlas, debemos registrarnos e iniciar sesión en Docker Desktop.



3

IMÁGENES Y CONTENEDORES

Aunque podemos interactuar con Docker mediante la interfaz gráfica, por norma general tenemos más versatilidad si utilizamos la línea de comandos.

Todos los comandos de Docker tienen la misma estructura

```
docker <command> [--options] <object>
```

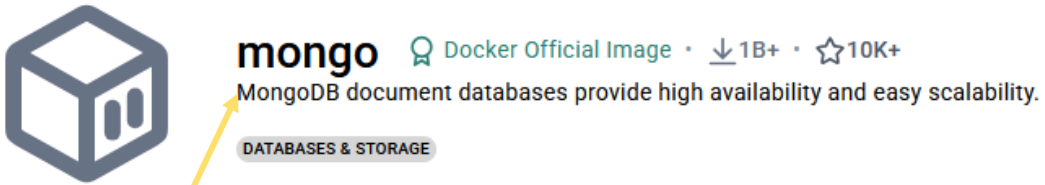
docker info

Muestra información sobre el entorno de ejecución de Docker

```
PS C:\proyectos\docker> docker info
Client:
 Version:    27.3.1
 Context:    desktop-linux
 Debug Mode: false
 Plugins:
  ai: Ask Gordon - Docker Agent (Docker Inc.)
    Version:  v0.1.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-ai.exe
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.18.0-desktop.2
    Path:     C:\Program Files\Docker\cli-plugins\docker-buildx.exe
  compose: Docker Compose (Docker Inc.)
    Version:  v2.30.3-desktop.1
    Path:     C:\Program Files\Docker\cli-plugins\docker-compose.exe
```

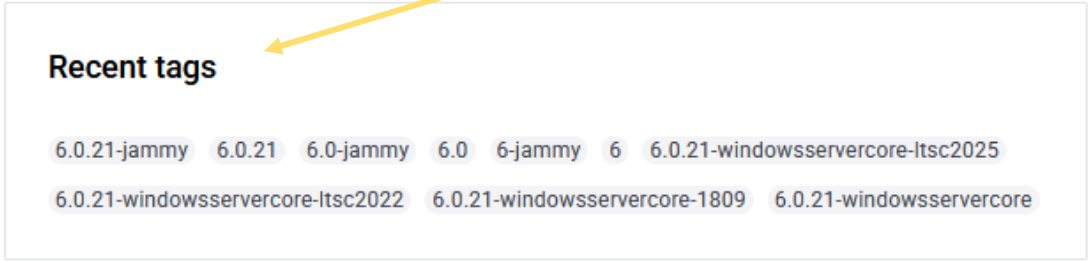
`docker [image] pull <nombre_imagen>`

Descarga una imagen en nuestro equipo

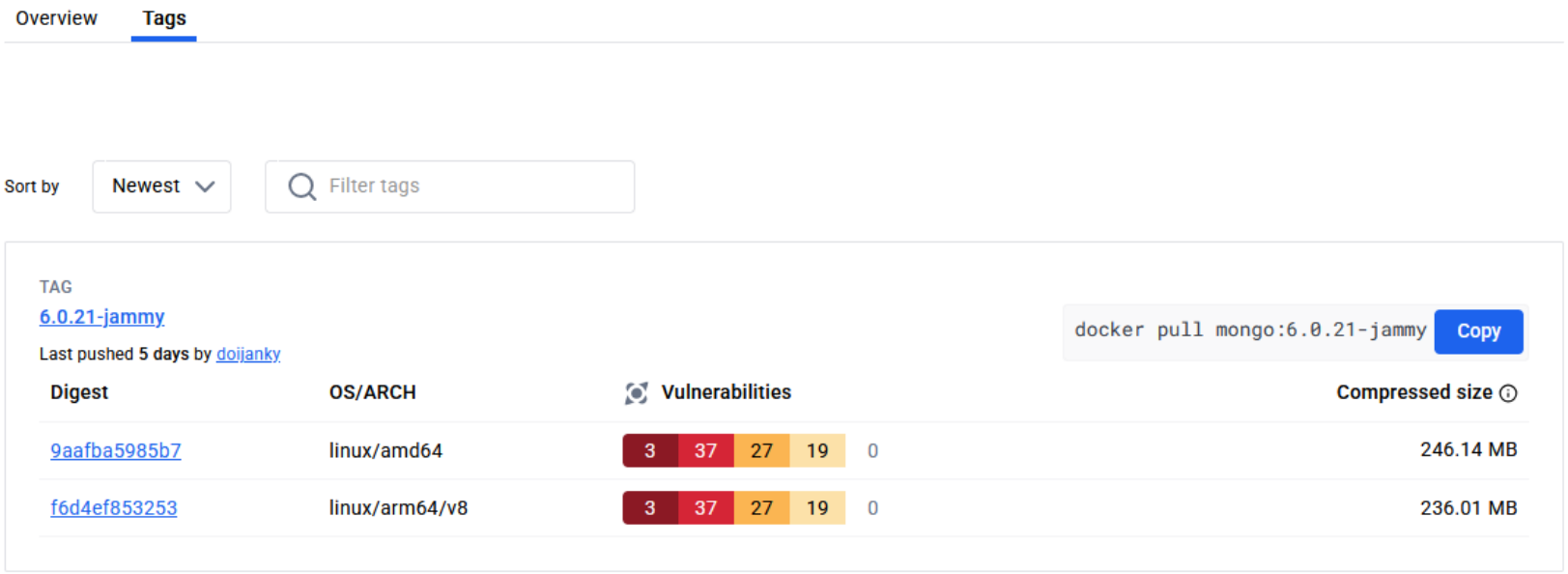


Para saber el nombre de la imagen vanos Docker Hub y buscamos la que queramos

Todas las imágenes tienen diferentes versiones. Se indica de la forma **mongo:6-jammy**. Si no ponemos nada o indicamos **latest** será la más reciente



En la pestaña *Tags* de Docker Hub podremos ver todas las vulnerabilidades que tiene cada versión



mongo:6.0.21-jammy MULTI-PLATFORM

DATABASES & STORAGE

INDEX DIGEST sha256:3a8d1155f89f0d1c6382a5bf7e60b196dbeb074a55391819383e13ab31e0c73b

OS/ARCH

linux/amd64

COMPRESSED SIZE	LAST PUSHED	TYPE	VULNERABILITIES	MANIFEST DIGEST
246.14 MB	5 days by doijanky	Image	3 37 27 19 0	sha256:9aafba59...

Layers (26)

ubuntu:78c5b6a20d04201e7ba513...			0	0	6	17	0
mongo:6.0.21-jammy			3	37	21	3	0
6	RUN /bin/sh -c set -eux; groupadd --gid 999 --system ...	1.78 KB					
7	RUN /bin/sh -c set -eux; apt-get update; apt-get install ...	1.51 MB					
8	ENV GOSU_VERSION=1.17	0 B					
9	ENV JSYAML_VERSION=3.13.1	0 B					
10	ENV JSYAML_CHECKSUM=662e32319bdd378e91f67...	0 B					

Vulnerabilities (88) Packages (220) Give feedback Analyzed by

Package or CVE name Fixable Show excepted

Reset filters

CVE ID	Severity	Fixable	Present in	Affected package(s
CVE-2023-24538	9.8 C	✓		golang / stdlib / 1.1
CVE-2024-24790	9.8 C	✓		golang / stdlib / 1.1
CVE-2023-24540	9.8 C	✓		golang / stdlib / 1.1
CVE-2025-30204	8.7 H	✓		golang / github.com

```
PS C:\proyectos\docker> docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
5a7813e071bf: Already exists
cf12757b6444: Pull complete
20cfb5e922d1: Pull complete
d11968535d8a: Pull complete
c711ee204b1d: Pull complete
4fc65ca4253f: Pull complete
dacd77ad2ef6: Pull complete
5fa69bd3db1e: Pull complete
Digest: sha256:1cb283500219e8fc0b61b328ea5a199a395a753d88b17351c58874fb425223cb
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview mongo
```

Al no indicar versión nos descarga la más reciente

Esta capa ya la descargaría para otra imagen, por lo que no vuelve a descargarla

```
PS C:\proyectos\docker> docker scout quickview mongo
    i New version 1.17.0 available (installed version is 1.15.0) at https://github.com/docker/scout-cli
      v SBOM of image already cached, 224 packages indexed
    v 1 exception obtained

    i Base image was auto-detected. To get more accurate results, build images with max-mode provenance
      Review docs.docker.com ↗ for more information.

  Target                 mongo:latest   3C   37H   27M   8L
    digest               b1d4ab2a0342
  Base image             ubuntu:24.04   0C   0H    5M    6L
  Updated base image     ubuntu:24.10   0C   0H    0M    0L
                                                     -5    -6

What's next:
    View vulnerabilities → docker scout cves mongo
    View base image update recommendations → docker scout recommendations mongo
    Include policy results in your quickview by supplying an organization → docker scout quickview mongo
>
```

Con este comando también podemos ver las vulnerabilidades de la imagen

`docker image list`

Muestra el listado de las imágenes que tenemos descargadas en nuestro equipo

```
PS C:\proyectos\docker> docker image list
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
mongo                               latest    b1d4ab2a0342   3 days ago       887MB
pandoc/latex                        latest    83b139f1ba53   6 days ago       563MB
ubuntu                              latest    a04dc4851cbc   8 weeks ago      78.1MB
ubuntu                              22.04     a24be041d957   8 weeks ago      77.9MB
quay.io/jupyter/base-notebook       latest    9fa59af4e48e   4 months ago     908MB
odoo                                16        91c2e3697b72   6 months ago     1.74GB
postgres                            14        480f26a07aa1   7 months ago     422MB
jupyter/minimal-notebook            latest    e75d34b8f32b   17 months ago    1.56GB
billryan/gitbook                    latest    e26d874c0165   7 years ago      1.34GB
fellah/gitbook                      latest    29087de21915   8 years ago      287MB
```

`docker image rm <nombre | id >`

Elimina localmente una imagen. Se puede identificar la imagen por el nombre o por el identificador

```
PS C:\proyectos\docker> docker image rm mongo
Untagged: mongo:latest
Untagged: mongo@sha256:1cb283500219e8fc0b61b328ea5a199a395a753d88b17351c58874fb425223cb
Deleted: sha256:b1d4ab2a034263b0e2306e41be1a596bd8600fb6fadef145a9088791c8d7c4b5
Deleted: sha256:a57c217d9c58bd2bfbe60623b7213c330f23be9c7330baac574155cb8a5a86db
Deleted: sha256:c4580b5f483b4eb88f97a4b7308454616cb308aa07ed53d5ad5cd15bf1597675
Deleted: sha256:42ceb8a2014a56082e6c5593ff9e0ad75506ec19c9c6e16d83b6e822178c9ad0
Deleted: sha256:2847a20f3a03acb0cfd8895fb854f979af01774a6942825e668e86a0507a59e0
Deleted: sha256:dcf3baa80dad3d4a9ce4a750c6766c2f1af83539f71dd93eb4249a0d5bd4f4ab
Deleted: sha256:c6a31bb5e4bb585b0666abd3fc7058a397943e9c024274989fd2a6f8c0fd75ff
Deleted: sha256:1267f6ee42550b1641ece2f0abb1b22ca4a55b3198a9b1e1f4ab7e131b4a58af
```

`docker [container] run <nombre>`

Ejecuta un contenedor

Mantiene abierta la entrada estándar del contenedor, permitiendo interactuar con él

Asigna una pseudoterminal al contenedor

```
docker run -i -t ubuntu /bin/bash
```

Nombre de la imagen que voy a instanciar

Comando que quiero ejecutar en el contenedor

ubuntu Docker Official Image · 1B+ · 10K+

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

OPERATING SYSTEMS

Como no he indicado versión, por defecto me descarga la última (latest)

Se podría indicar una versión diferente de la forma `ubuntu:22.04`

La imagen no se encuentra en nuestro equipo, por lo que la descarga de Docker Hub, si creara otro contenedor ya no la descargaría

```
PS C:\proyectos\docker> docker run -i -t ubuntu /bin/bash
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@0ba028ef2bd7:/#
```

Aquí tenemos el intérprete Bash que hemos dicho que nos ejecute en el contenedor

En el ejemplo anterior hemos ejecutado un comando interactivo, pero podemos ejecutar cualquier comando.

En este caso voy a ejecutar el comando `uname`

```
PS C:\proyectos\docker> docker run -i -t ubuntu:22.04 uname -a
Linux f32832516444 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55
PS C:\proyectos\docker>
```

Observa que, como ya descargamos la imagen, no necesita volver a descargarla

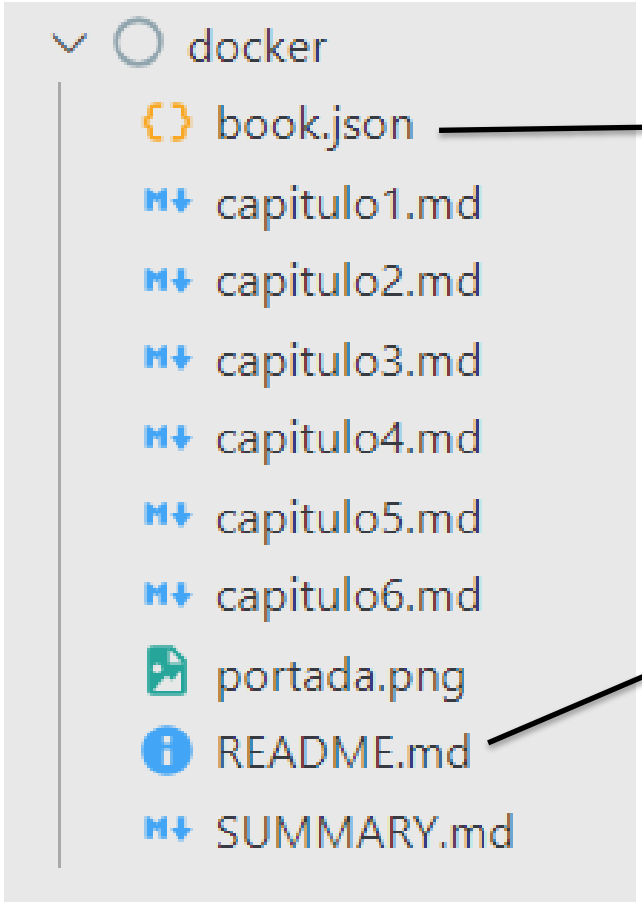
Modificadores más importantes:

Modificador	Función
`-it`	Ejecuta el contenedor en modo interactivo
`-d`	Ejecuta el contenedor en segundo plano
`--rm`	Elimina el contenedor automáticamente cuando se detiene
`--name <nombre>`	Asigna un nombre personalizado al contenedor
`-p <host>:<cont>`	Mapea puertos entre el host y el contenedor
`-v <host>:<cont>`	Mapea directorios entre el host y el contenedor
`--env <var>:<valor>`	Define variables de entorno
`--entrypoint <cmd>`	Sobrescribe el comando inicial definido en la imagen

Ejemplo práctico: quiero usar **Gitbook** para generar unos apuntes en PDF a partir de varios ficheros en formato Markdown.

Voy a usar esta imagen que contiene Gitbook

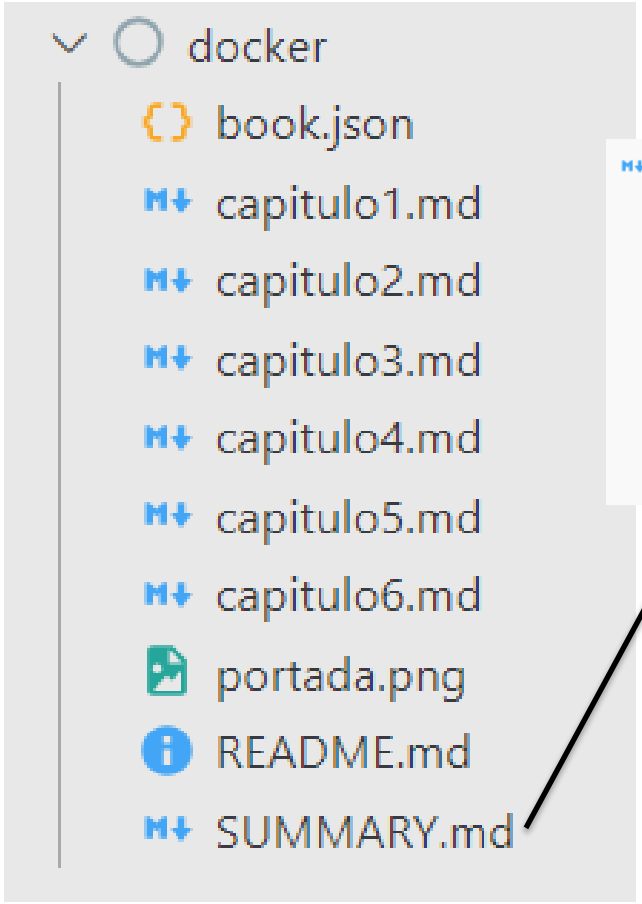
Comienzo creando la carpeta de proyecto con la estructura de ficheros




```json
{
  "title": "Apuntes Bash",
  "author": "Víctor J. González",
  "language": "es"
}
```

SUMMARY.md | book.json | README.md

```
1  # Apuntes Bash Scripting
2  ![Portada](portada.png)
3
4  ## Autor: Víctor J. González
5  ### Versión: 1.0
```



SUMMARY.md | book.json | README.md

```
1  # Índice
2
3  - [Introducción](README.md)
4  - [Conceptos básicos de creación de scripts](capitulo1.md)
5  - [Entrada y salida de datos](capitulo2.md)
6  - [Operaciones aritméticas y de cadenas](capitulo3.md)
7  - [Ejecución condicional](capitulo4.md)
8  - [Llaves, corchetes y paréntesis](capitulo5.md)
9  - [Funciones en Bash](capitulo6.md)
```

Y lanzo el contenedor

```
PS C:\proyectos\docker> docker run `
>> --rm `
>> --volume .:/gitbook `
>> billryan/gitbook `
>> gitbook pdf
```

Con `--rm` indicamos que se debe eliminar la máquina después de ejecutarse

La imagen a partir de la cual genero el contenedor

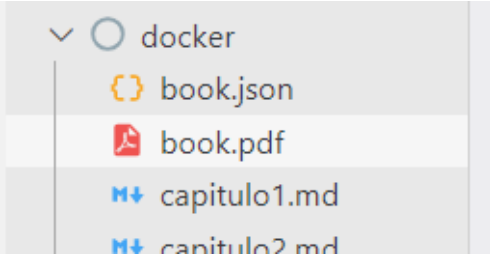
La opción `--volume` sirve para **mapear** un directorio de mi máquina en el contenedor, en este caso el directorio actual en el directorio `/gitbook`

Y, por último, el comando que voy a ejecutar dentro del contenedor

Ejecutamos la orden y comprobamos que no nos muestra ningún error.

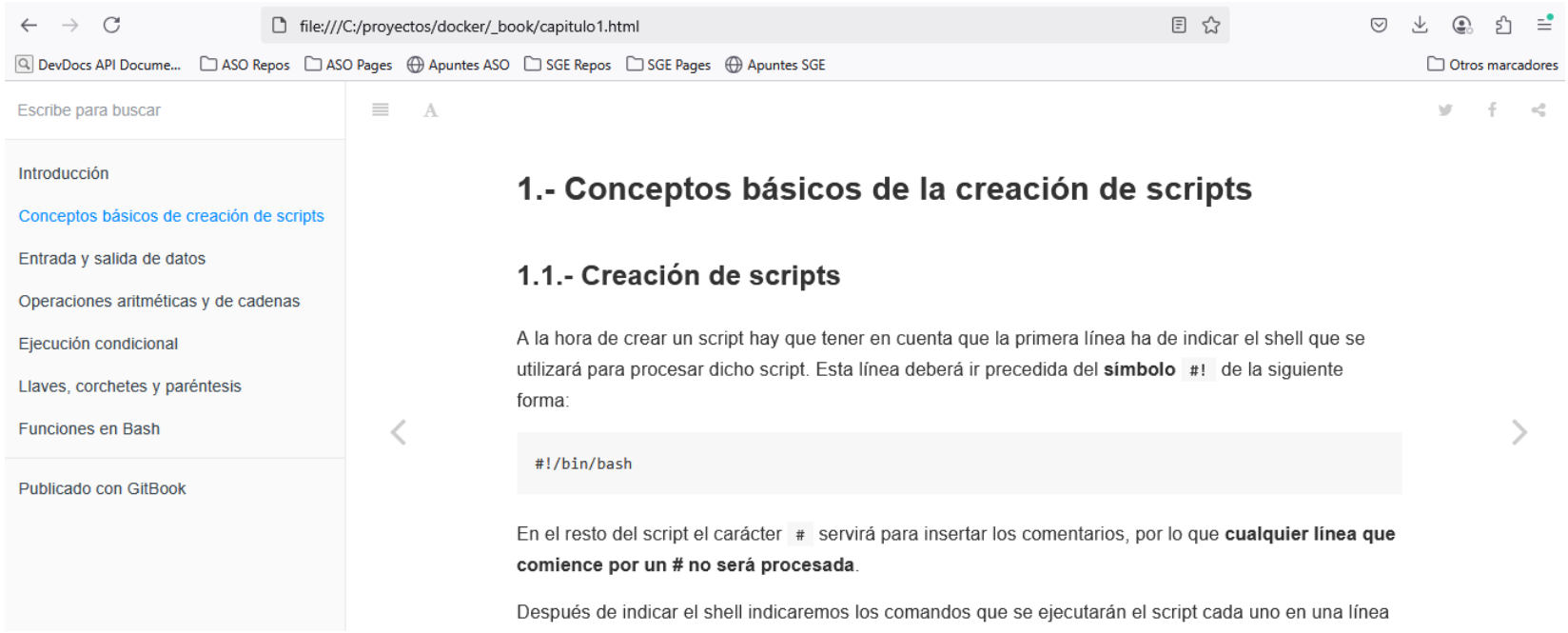
Y veremos que nos genera el fichero PDF

```
PS C:\proyectos\docker> docker run `
>> --rm `
>> --volume .:/gitbook `
>> billryan/gitbook `
>> gitbook pdf
info: 7 plugins are installed
info: 6 explicitly listed
info: loading plugin "highlight"... OK
info: loading plugin "search"... OK
info: loading plugin "lunr"... OK
info: loading plugin "sharing"... OK
info: loading plugin "fontsettings"... OK
info: loading plugin "theme-default"... OK
info: found 7 pages
info: found 2 asset files
info: >> generation finished with success in 4.0s !
info: >> 1 file(s) generated
PS C:\proyectos\docker>
```



```
PS C:\proyectos\docker> docker run `
>> --rm `
>> --volume .:/gitbook `
>> billryan/gitbook `
>> gitbook build
```

También podemos generar la página Web usando *gitbook build*



Otras situaciones en las que es útil usar un contenedor para realizar tareas puntuales.

Conversión de vídeos sin instalar software

```
PS G:\proyectos\docker> docker run `
>> --rm `
>> -v ".:/videos" `
>> jrottenberg/ffmpeg `
>> -i /videos/input.mp4 `
>> -vf "scale=640:480" `
>> /videos/output.mp4
```

FFmpeg Docker image

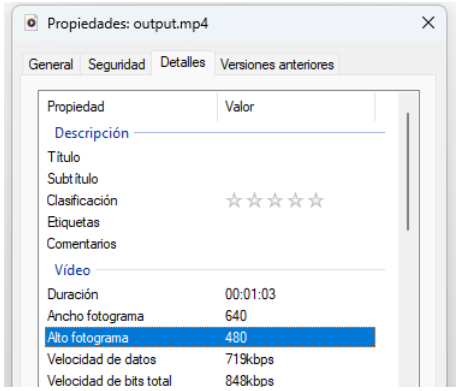
docker stars 452 | docker pulls 85M | pipeline passed | Azure Pipelines never built | docker build automated

This project prepares a minimalist Docker image with FFmpeg. It compiles FFmpeg from sources following i

You can install the latest build of this image by running `docker pull jrottenberg/ffmpeg` .

This image can be used as a base for an encoding farm.

Esto son parámetros de ffmpeg, en este caso indicamos los ficheros de entrada y salida y que queremos cambiar la escala del vídeo a 640x480



input.mp4	18/12/2022 11:52	Archivo MP4	133.466 KB
output.mp4	23/03/2025 8:37	Archivo MP4	6.598 KB

Conversión de imágenes sin instalar software

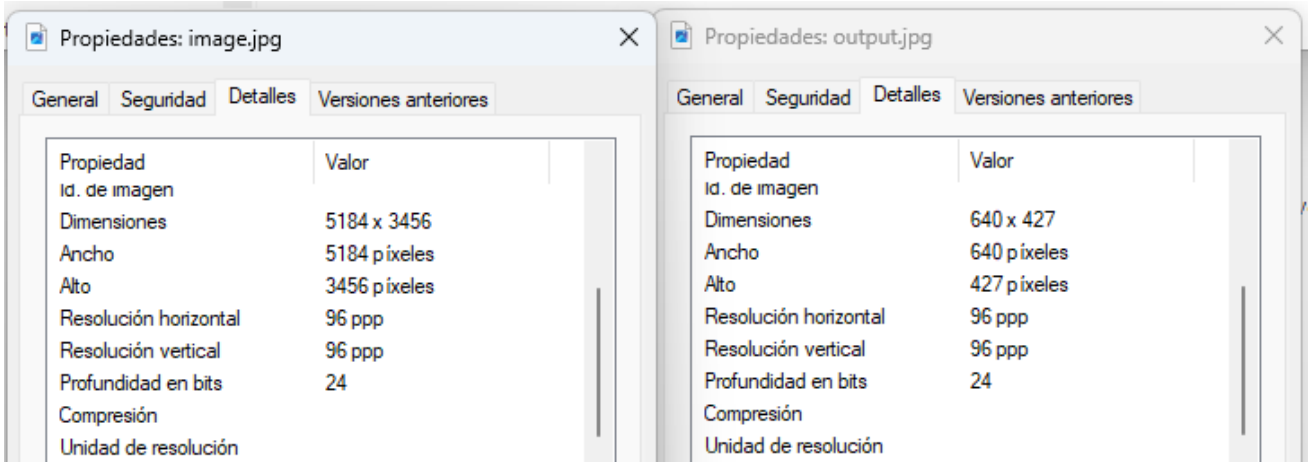
```
PS G:\proyectos\docker> docker run `
>> --rm `
>> -v .:/images `
>> dpokidov/imagemagick `
>> /images/image.jpg `
>> -resize 640x480 `
>> /images/output.jpg
```

imagemagick-docker

Overview

Delivers the latest version of the ImageMagick 7 to your environment in Docker container.

The purpose of that image is to be able to run the latest version of ImageMagick in stable Lin



Escaneo de seguridad y auditorías

```
PS G:\proyectos\docker> docker run `
>> --rm `
>> alpine/nikto `
>> -h https://www.iessanandres.com
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          142.250.184.19
+ Target Hostname:    www.iessanandres.com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=www.iessanandres.com
                   Altnames: www.iessanandres.com
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/O=Google Trust Services/CN=WR3
+ Message:            Multiple IP addresses found: 142.250.184.19, 142.250.184.19
+ Start Time:         2025-03-23 07:58:10 (GMT0)
---------------------------------------------------------------------------
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ The site uses SSL and Expect-CT header is not present.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ ERROR: Error limit (20) reached for host, giving up. Last error: opening stream:
+ Scan terminated:  20 error(s) and 6 item(s) reported on remote host
+ End Time:           2025-03-23 07:59:37 (GMT0) (87 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

nikto es un escáner de vulnerabilidades en sitios web

Servidor web temporal

```
PS G:\proyectos\docker> docker run `
>> --rm `
>> -v .:/usr/share/nginx/html `
>> -p 8080:80 `
>> nginx
```

Mapeo un directorio de la máquina física con la página web en el directorio del contenedor donde Nginx espera que esté

Con el modificador **–p** redirecciono un puerto del contenedor a otro de la máquina física. En este caso, el 80 del contenedor se mapeará sobre el 8080 de la máquina física

Imagen oficial de nginx

Hola mundo!!!

Bienvenidos al curso de Docker

Base de datos de pruebas

```
PS C:\proyectos\docker> docker run `
>> --rm `
>> -e MYSQL_ROOT_PASSWORD=1234 `
>> -e MYSQL_DATABASE=Pruebas `
>> -p 3306:3306 `
>> mysql
```

El modificador **–e** sirve para establecer el valor de variables de entorno dentro del contenedor. En la documentación de Docker Hub suele indicar qué variables hay que definir en la imagen

Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

`MYSQL_ROOT_PASSWORD`

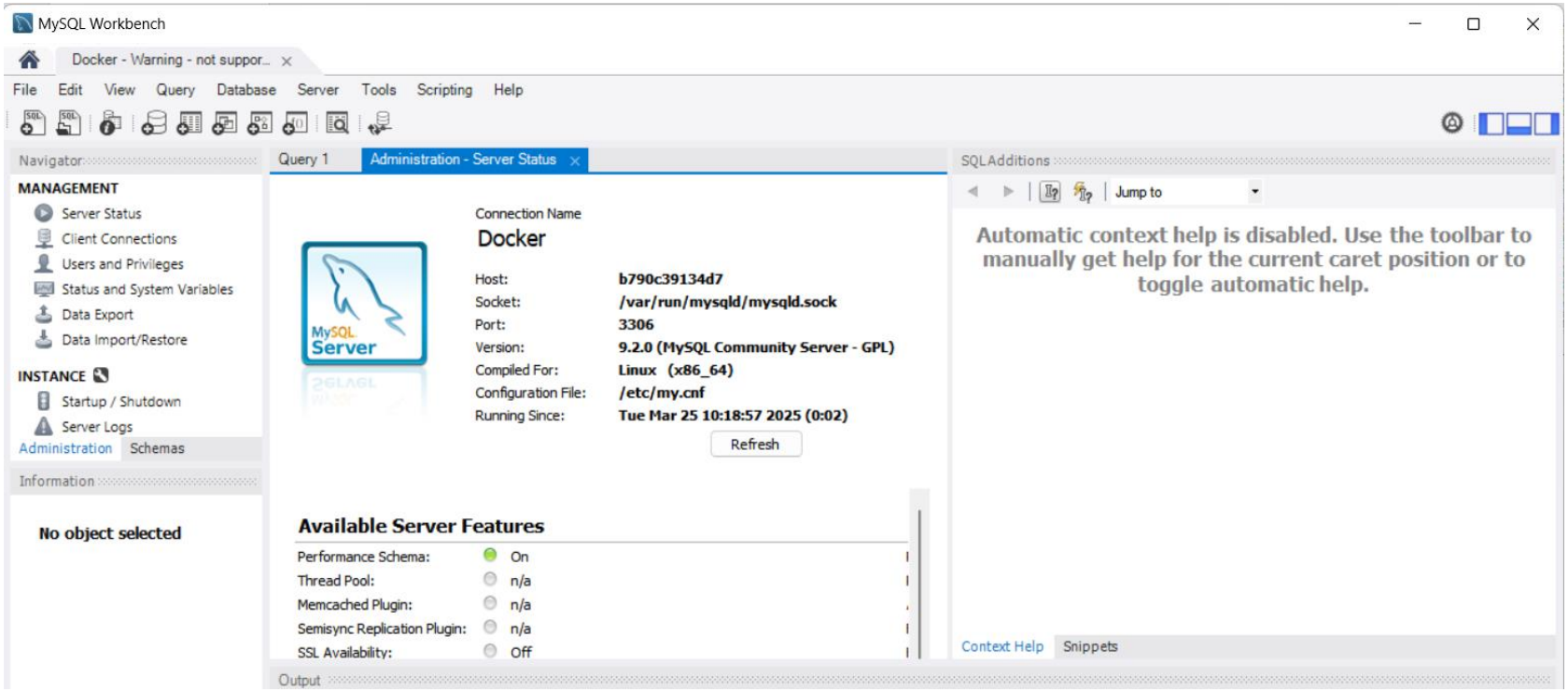
This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to `GRANT ALL`) to this database.

`MYSQL_USER`, `MYSQL_PASSWORD`

Y ya podemos acceder a la base de datos desde cualquier aplicación mediante el puerto 3306 de nuestro equipo



Pero también podemos lanzar el cliente de bases de datos en un contenedor. Por ejemplo, vamos a lanzar otro contenedor con PHPMyAdmin

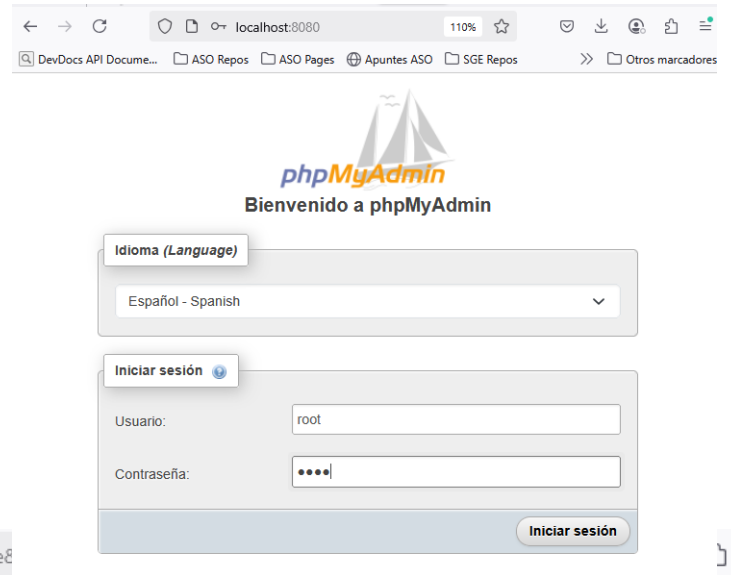
```
PS G:\proyectos\docker> docker run `
>> --rm `
>> -p 8080:80 `
>> -e PMA_HOST=host.docker.internal `
>> -e PMA_PORT=3306 `
>> phpmyadmin/phpmyadmin
```

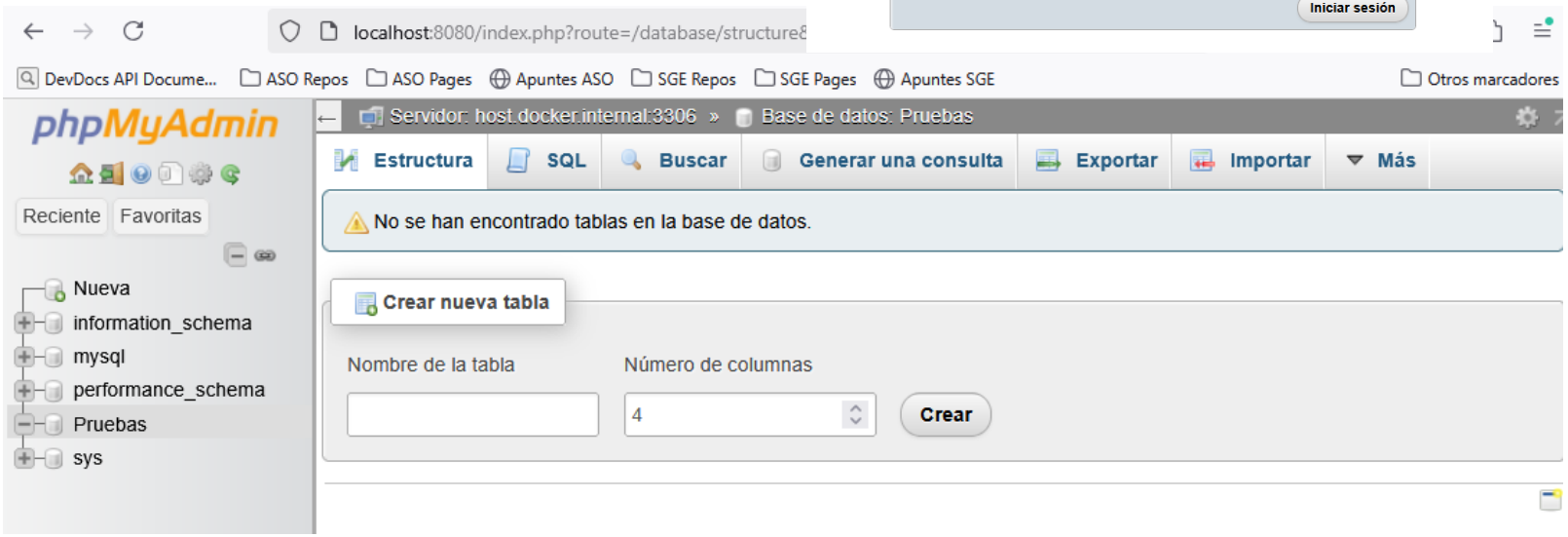
Si aquí pongo **localhost**, como es una variable interna del contenedor apuntará al propio contenedor. Por ello, debemos usar **host.docker.internal**

Environment variables summary

- `PMA_ARBITRARY` - when set to 1 connection to the arbitrary server will be allowed
- `PMA_HOST` - define address/host name of the MySQL server
- `PMA_VERBOSE` - define verbose name of the MySQL server
- `PMA_PORT` - define port of the MySQL server
- `PMA_HOSTS` - define comma separated list of address/host names of the MySQL servers
- `PMA_VERBOSES` - define comma separated list of verbose names of the MySQL servers

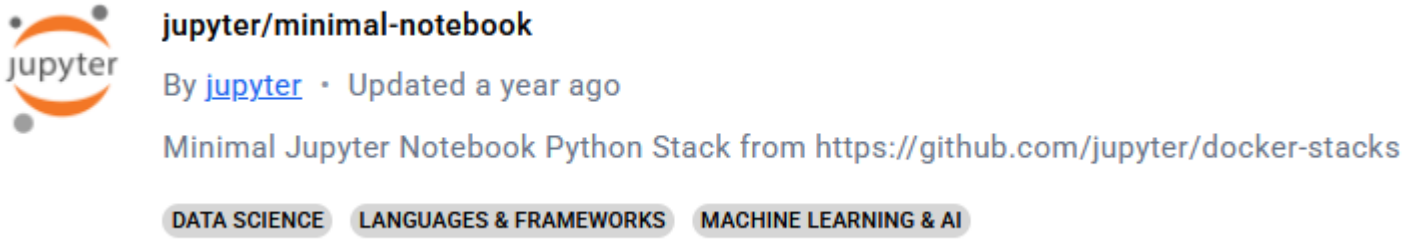
Si vamos en nuestra máquina al puerto 8080 podremos acceder al PHPMyAdmin que estará conectado a la base de datos





Ejecución de Python

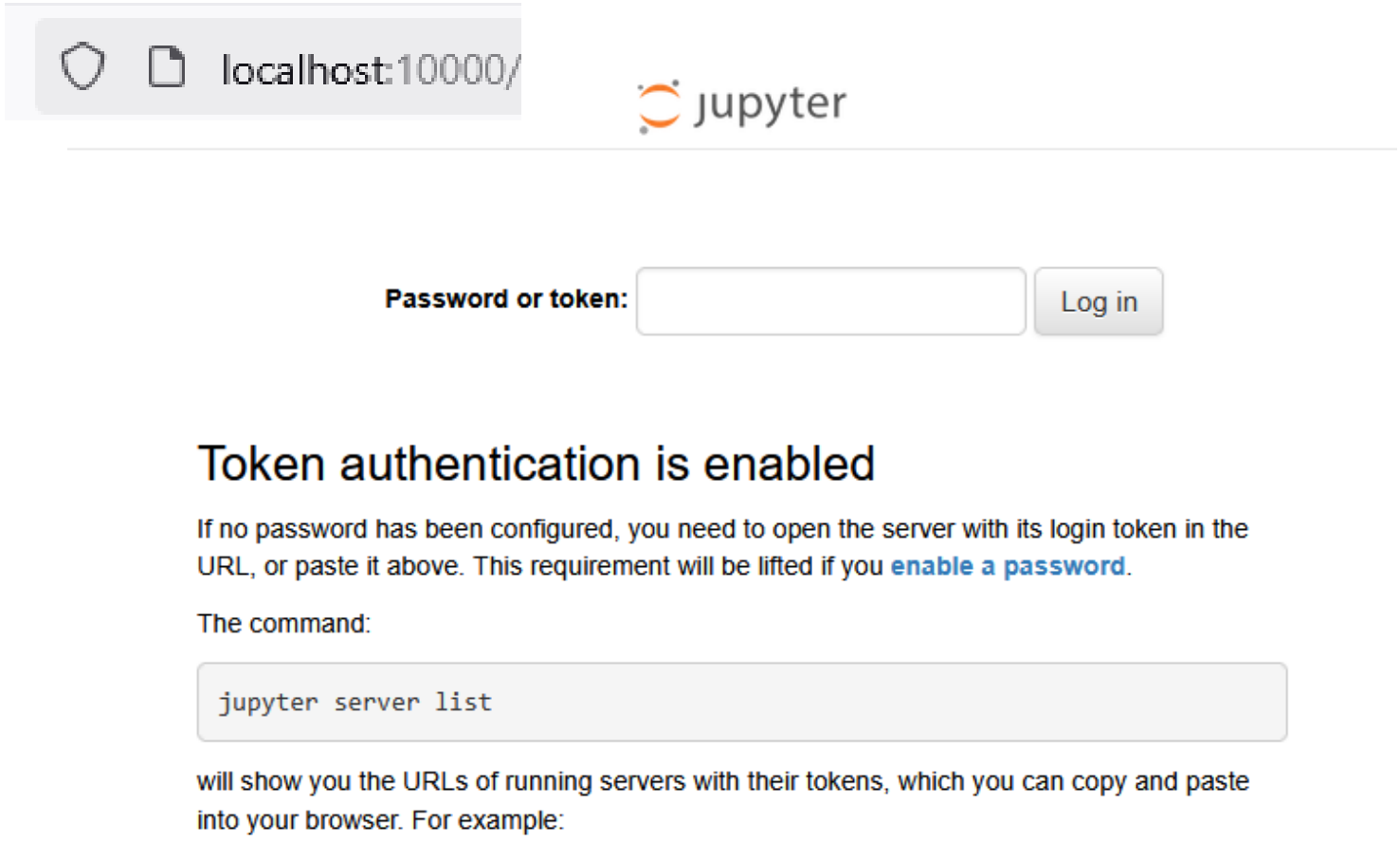
Jupyter es una aplicación web que permite crear y ejecutar scripts en Python combinándolos con textos (Markdown), gráficos y resultados de la ejecución de los scripts.



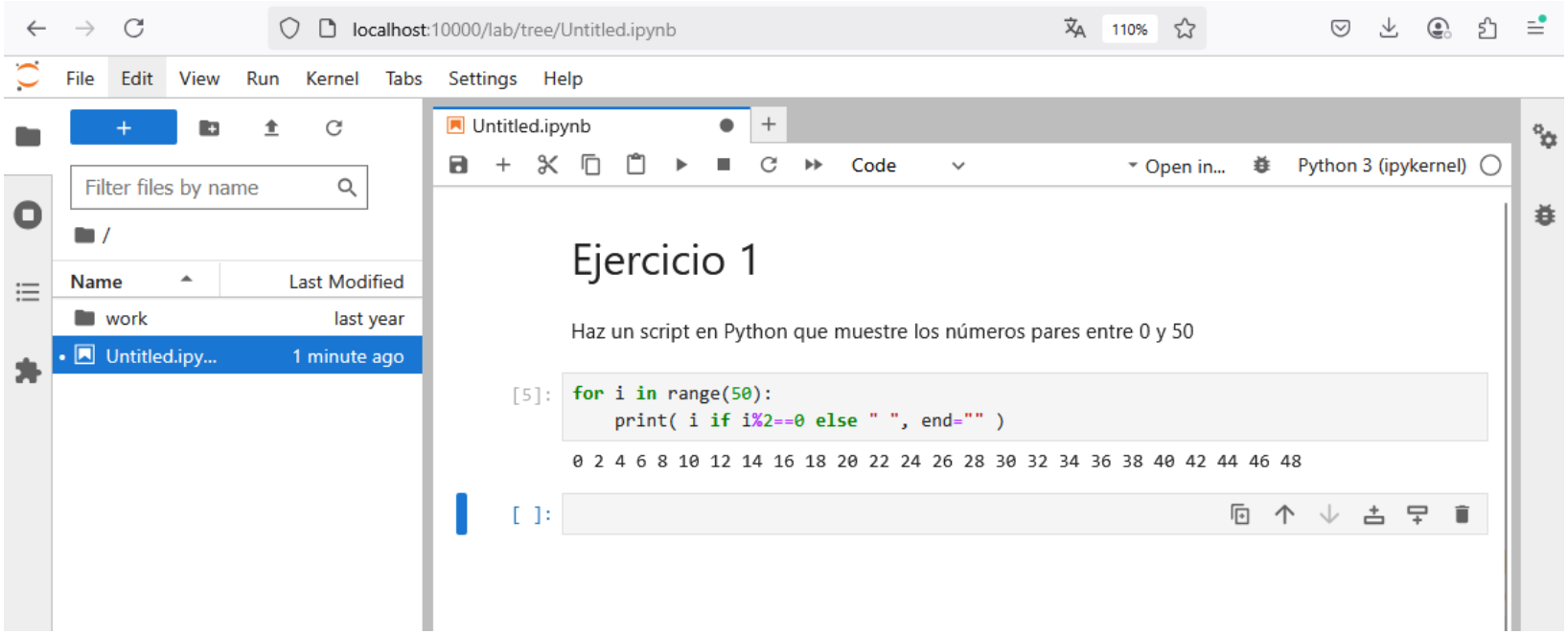

```
PS C:\proyectos\docker> docker run `
>> -d `
>> -p 10000:8888 `
>> jupyter/minimal-notebook
373fdd3580c32f0b6ca3d93323d688c1c94cb6ddb27349974b0dbf161c6e6353
PS C:\proyectos\docker>
```

En este caso lo estoy ejecutando en segundo plano

Al acceder me pide un token que me ha mostrado al inicializar, pero como lo he lanzado en segundo plano no veo los mensajes del contenedor.



Para solucionarlo, puedo acceder a los registros de log en la interfaz gráfica, haciendo click sobre el contenedor.


localhost:10000/lab/tree/Untitled.ipynb
110%
File Edit View Run Kernel Tabs Settings Help
Filter files by name
/
Name
Last Modified
work
last year
Untitled.ipy...
1 minute ago
Untitled.ipynb
Code
Open in...
Python 3 (ipykernel)
Ejercicio 1
Haz un script en Python que muestre los números pares entre 0 y 50
[5]:
for i in range(50):
print(i if i%2==0 else " ", end="")
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
[ ]:

docker [container] **ps**

Muestra una lista de contenedores en ejecución

```
PS C:\proyectos\docker> docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
373fdd3580c3   jupyter/minimal-notebook       "tini -g -- start-no…"   48 minutes ago
b790c39134d7   mysql                          "docker-entrypoint.s…"   2 hours ago
PS C:\proyectos\docker>
```

```
STATUS                    PORTS                                NAMES
Up 48 minutes (healthy)   0.0.0.0:10000->8888/tcp              jovial_burnell
Up 2 hours                0.0.0.0:3306->3306/tcp, 33060/tcp    goofy_hawking
```

Con el modificador **–a** también muestra los contenedores detenidos

```
PS C:\proyectos\docker> docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                      PORTS                                NAMES
373fdd3580c3   jupyter/minimal-notebook       "tini -g -- start-no…"   52 minutes ago   Up 52 minutes (healthy)     0.0.0.0:10000->8888/tcp              jovial_burnell
b790c39134d7   mysql                          "docker-entrypoint.s…"   2 hours ago      Up 2 hours                  0.0.0.0:3306->3306/tcp, 33060/tcp    goofy_hawking
85ec71834907   mongo                          "docker-entrypoint.s…"   2 hours ago      Exited (0) 2 hours ago                                           nice_ptolemy
a0bdf11bcc01   odoo:16                        "/entrypoint.sh --de…"   6 weeks ago      Exited (255) 5 weeks ago    0.0.0.0:8069->8069/tcp, 8071-8072/tcp   odoo_16
5aa2047d7228   postgres:14                    "docker-entrypoint.s…"   6 weeks ago      Exited (255) 5 weeks ago    0.0.0.0:5432->5432/tcp               db
78b116c4aa56   odoo:16                        "/entrypoint.sh --de…"   3 months ago     Exited (0) 3 months ago                                          odoo_examen
3c39100fed5c   postgres:14                    "docker-entrypoint.s…"   3 months ago     Exited (255) 3 months ago   0.0.0.0:2345->5432/tcp               postgres
d3f01d1a04af   jupyter/minimal-notebook       "tini -g -- start-no…"   3 months ago     Exited (255) 3 months ago   0.0.0.0:8888->8888/tcp               intelligent_sw
```

docker [container] **stop** <nombre|id>

Detiene un contenedor.

```
PS C:\proyectos\docker> docker stop jovial_burnell
jovial_burnell
```

docker [container] **restart** <nombre|id>

Reinicia un contenedor previamente detenido conservando la configuración

```
PS C:\proyectos\docker> docker start 373f
373f
```

docker [container] **rm** <nombre|id>

Elimina un contenedor

```
PS C:\proyectos\docker> docker rm 85ec
85ec
```

4

CREACIÓN Y GESTIÓN DE IMÁGENES

Hasta ahora hemos creado contenedores a partir de imágenes predefinidas, pero también podemos crear nuestras propias imágenes.

Esto se hace a partir del denominado **Dockerfile**, un archivo de texto plano que contiene un conjunto de instrucciones para construir imágenes de Docker personalizadas.

En este fichero se puede especificar:

- La base de la imagen
- Las dependencias
- Configuraciones
- Comandos a ejecutar

Cada línea en un Dockerfile es una **instrucción** seguida de una serie de argumentos.

Instrucción	Descripción
`FROM`	Define la imagen base
`RUN`	Ejecuta comandos desde la construcción
`COPY/ADD`	Copia archivos del host al contenedor
`WORKDIR`	Establece el directorio de trabajo
`ENV`	Define variables de entorno
`EXPOSE`	Indica qué puertos expone el contenedor
`CMD/ENTRYPOINT`	Define el comando por defecto al iniciar el contenedor

FROM

Define la imagen base sobre la que se construirá la nueva imagen

Dockerfile

```
1  FROM ubuntu:24.04
2
```

RUN

Permite ejecutar un comando durante el proceso de construcción de la imagen Docker. Cada instrucción `RUN` crea una nueva capa en la imagen.

Hay dos formas de usar `RUN`:

- **Shell form**: ejecuta el comando dentro de una Shell (/bin/sh –c)

Dockerfile

```
1   FROM ubuntu:24.04
2
3   RUN apt-get update && apt-get install -y curl
4
```

- **Forma exec**: ejecuta el comando directamente sin una Shell intermedia. Se especifica como un array JSON.

Dockerfile

```
1   FROM ubuntu:24.04
2
3   RUN ["apt-get", "update"]
4   RUN ["apt-get", "install", "-y", "curl"]
```

Característica	Formato shell	Formato exec
Sintaxis	CMD comando arg1 arg2	CMD [“comando”, “arg1”, “arg2”]
Uso de Shell	Sí (/bin/sh -c)	No, se ejecuta directamente
Variables de entorno	Expandidas automáticamente	No
Wildcards (*, ?, [ ])	Sí	No
Soporte para &&	Si	No
Eficiencia	Menos (proceso extra)	Más, no hay Shell extra
Recomendado para	Comandos complejos con pipes y redirecciones	Ejecución directa de programas

Ejemplo

Dockerfile

```
1   FROM ubuntu:latest
2
3   RUN apt-get update
4   RUN apt-get install -y curl
```

Con este subcomando generamos una nueva imagen a partir de un Dockerfile

Con **–t** indicamos el nombre de la nueva imagen

Etiqueta de la versión

```
docker build -t ubuntu-curl:1.0 .
```

Ruta donde se encuentra el Dockerfile

Y ya podemos ver la imagen que hemos creado

```
PS C:\proyectos\docker\dockerfile_tests> docker image list
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
ubuntu-curl                   1.0       9f089cc2e833   3 minutes ago    141MB
mongo                         latest    b1d4ab2a0342   5 days ago       887MB
```

Y crear contenedores a partir de dicha imagen donde vemos que tiene instalado curl

```
PS C:\> docker run `
>> --rm `
>> ubuntu-curl:1.0 `
>> curl -V
curl 8.5.0 (x86_64-pc-linux-gnu) libcurl/8.5.0 OpenSSL/3.0.13 zlib/1.3 brotli/1.1.0
0.21.2 (+libidn2/2.3.7) libssh/0.10.6/openssl/zlib nghttp2/1.59.0 librtmp/2.3 OpenLD
Release-Date: 2023-12-06, security patched: 8.5.0-2ubuntu10.6
```

Si miramos el historial de la imagen veremos que hemos añadido dos capas, uno por cada `RUN` del Dockerfile

```
PS C:\> docker image history ubuntu-curl:1.0
IMAGE          CREATED         CREATED BY                                      SIZE
9f089cc2e833   25 minutes ago  RUN /bin/sh -c apt-get install -y curl # bui…   15.8MB
<missing>      25 minutes ago  RUN /bin/sh -c apt-get update # buildkit        46.7MB
<missing>      8 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      8 weeks ago     /bin/sh -c #(nop) ADD file:6df775300d76441aa…   78.1MB
```

Podemos (y es aconsejable) optimizar la creación del Dockerfile usando un único `RUN`

Dockerfile

```
1   FROM ubuntu:latest
2
3   RUN apt-get update && apt-get install -y curl
```

Podemos (y es aconsejable) optimizar la creación del Dockerfile usando un único `RUN`

```
PS C:\proyectos\docker\dockerfile_tests> docker image list
REPOSITORY                   TAG       IMAGE ID       CREATED          SIZE
ubuntu-curl                  2.0       196f14bc896d   31 seconds ago   141MB
ubuntu-curl                  1.0       9f089cc2e833   32 minutes ago   141MB
```

```
PS C:\proyectos\docker\dockerfile_tests> docker image history ubuntu-curl:2.0
IMAGE          CREATED              CREATED BY                                      SIZE      COMMEN
196f14bc896d   About a minute ago   RUN /bin/sh -c apt-get update && apt-get ins…   62.4MB    buildk
<missing>      8 weeks ago          /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      8 weeks ago          /bin/sh -c #(nop) ADD file:6df775300d76441aa…   78.1MB
<missing>      8 weeks ago          /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
```

COPY

Este comando permite copiar archivos al interior de una imagen Docker.

La estructura es **COPY <src> .. <dest>**

Algunas reglas de este comando son:

- Si <src> es un directorio copia el contenido completo
- <src> tiene que encontrarse en el *build context*
- Si se indica más de un recurso <src>, el elemento <dest> tiene que ser un directorio y finalizar con una barra inclinada (“/”)

Ejemplo 1: vamos a crear una imagen con **nginx** y una página web precargada

Dockerfile × index.html

```
1   FROM ubuntu:latest
2
3   RUN apt-get update && \
4       apt-get install -y nginx
5
6   COPY ./web/* /var/www/html/
7
8   EXPOSE 80
9
10  CMD ["nginx", "-g", "daemon off;"]
```

Con EXPOSE indicamos que un puerto estará expuesto al exterior

Lanzamos nginx en primer plano

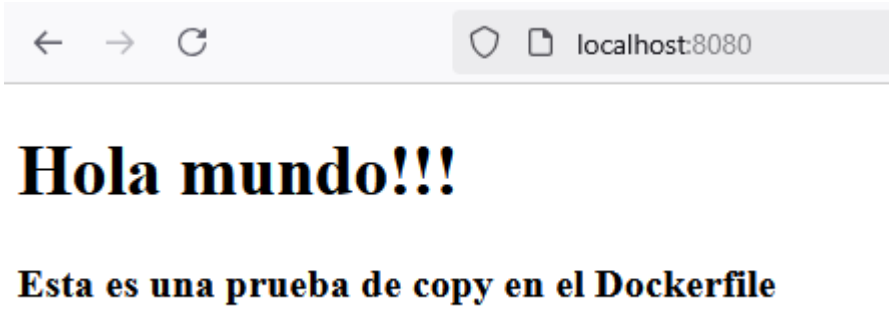
Construyo la imagen

```
docker build -t web-server:latest .
```

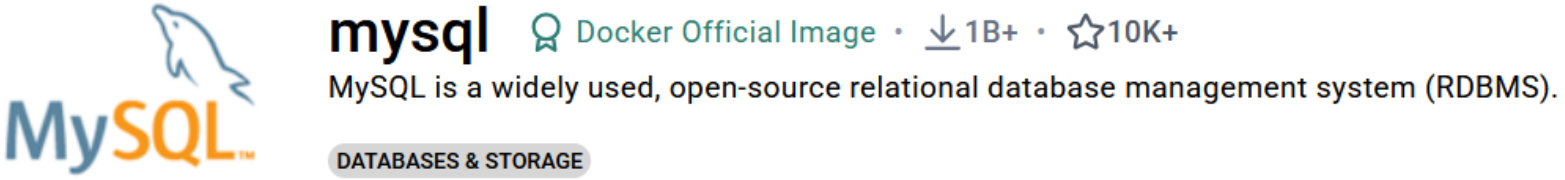
Levanto el contenedor

```
PS C:\> docker run `
>> -p 8080:80 `
>> -d `
>> web-server
c810c4f51864c3f5dd2387dc4b3fed34279d50d78e0e0899f7550a776c45c726
```

Y ya tendría mi web



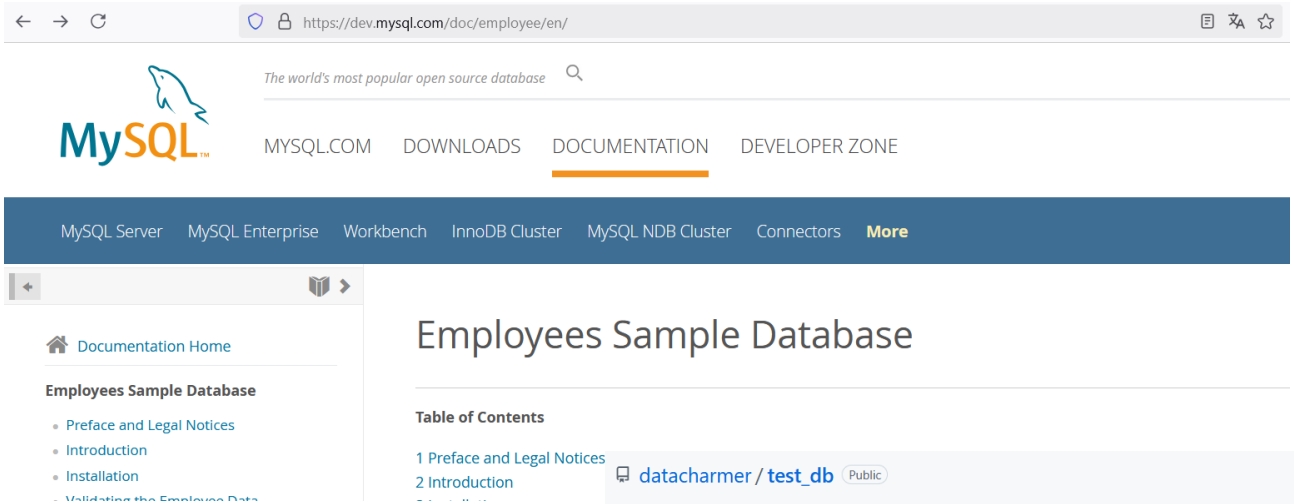
Ejemplo 2: vamos a crear una imagen con MySQL y que tenga una base de datos precargada.



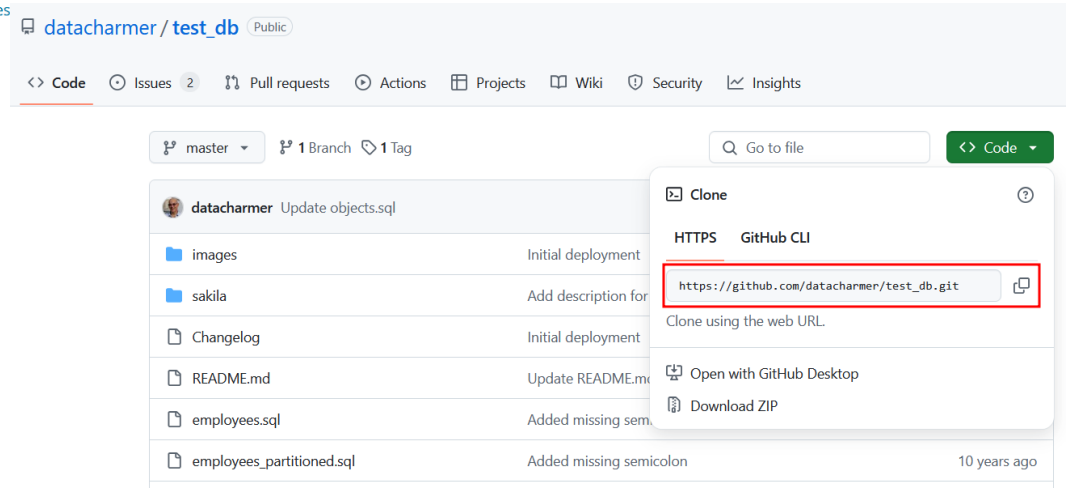
La idea es crear una imagen que contenga un script SQL que se ejecute la primera vez que se inicie el contenedor.

Utilizaremos el directorio **/docker-entrypoint-initdb.d/**, un directorio especial soportado por diversas imágenes de bases de datos (*MySQL, PostgreSQL, MariaDB, MongoDB*) para almacenar scritps de inicialización

Voy a utilizar la base de datos **Employees** disponible en https://dev.mysql.com/doc/employee/en/



La base de datos está alojada en Github, por lo que utilizo Git para descargarla

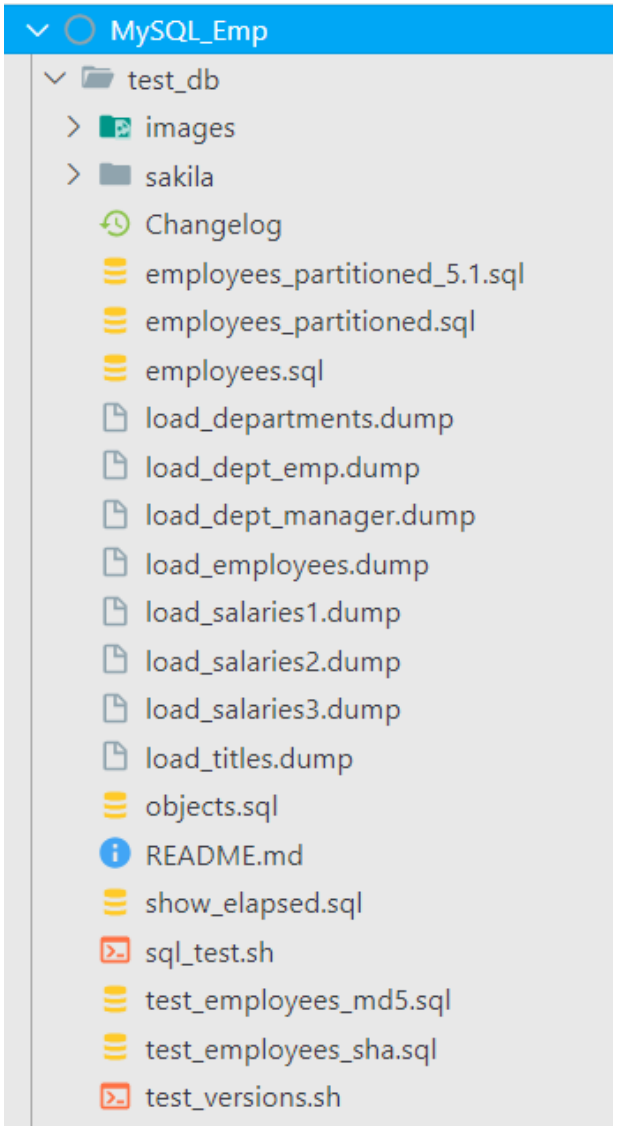


Comienzo creando un directorio que alojará mi proyecto y clono en él el repositorio con la base de datos

```
PS C:\MySQL_Emp> New-Item ./Dockerfile


    Directorio: C:\MySQL_Emp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         01/04/2025      9:11              0 Dockerfile


PS C:\MySQL_Emp> git clone https://github.com/datacharmer/test_db.git
Cloning into 'test_db'...
remote: Enumerating objects: 121, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (9/9), done.
Receiving objects:  39% (48/121), 14.12 MiB | 7.01 MiB/s
```

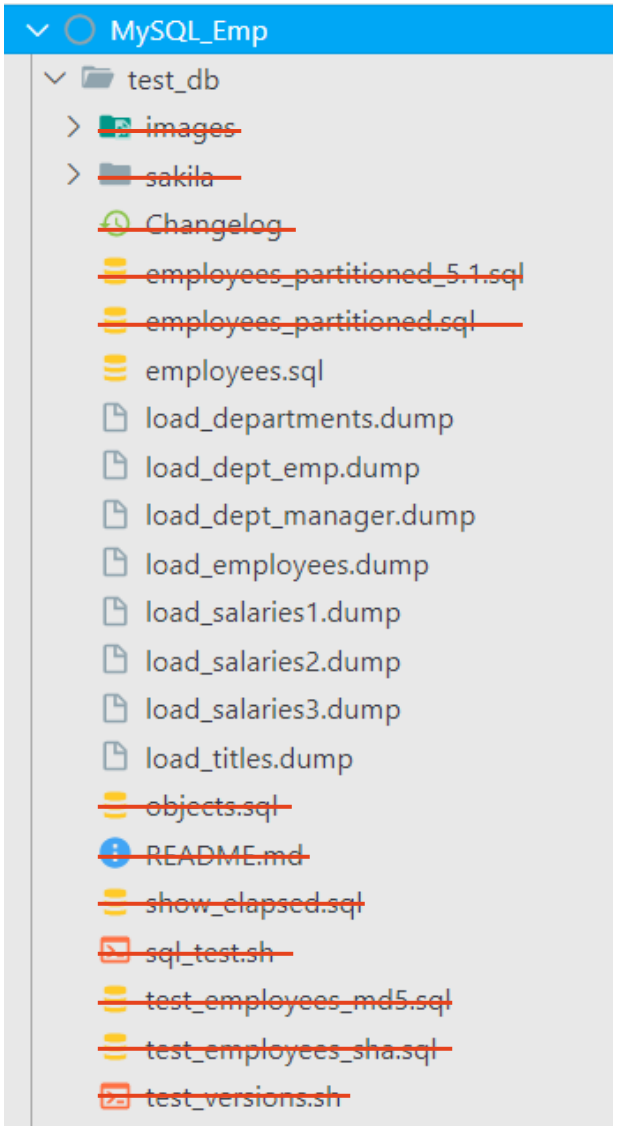
Creo el fichero Dockerfile, por ahora vacío

Y clono el repositorio con la base de datos



Este es el contenido del repositorio, pero hay que hacer limpieza para que funcione.

Docker ejecutará todos los scripts con extensión **.sql**, **.sql.gz** y **.sh** que haya en dicho directorio /docker-entrypoint-initdb.d/, así que me tengo que asegurar de que eso sea lo que contenga.

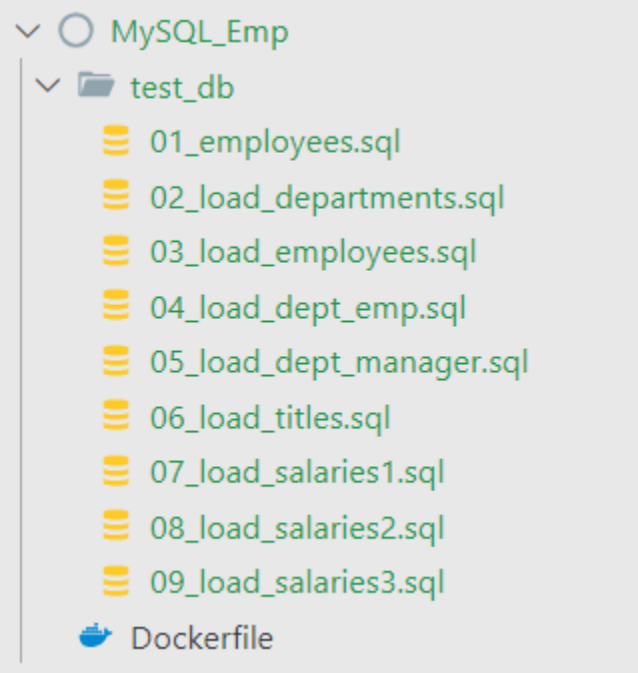


Elimino todos los ficheros que no pertenecen a la base de datos

El siguiente problema que tengo es que los ficheros .dump (insertados desde employees.sql) no se cargan.

Solución: cambiar la extensión a SQL y renombrarlos para que se mantengan el mismo orden

```
112  SELECT 'LOADING departments' as 'INFO';
113  source load_departments.dump ;
114  SELECT 'LOADING employees' as 'INFO';
115  source load_employees.dump ;
116  SELECT 'LOADING dept_emp' as 'INFO';
117  source load_dept_emp.dump ;
118  SELECT 'LOADING dept_manager' as 'INFO';
119  source load_dept_manager.dump ;
120  SELECT 'LOADING titles' as 'INFO';
121  source load_titles.dump ;
122  SELECT 'LOADING salaries' as 'INFO';
123  source load_salaries1.dump ;
124  source load_salaries2.dump ;
125  source load_salaries3.dump ;
126
127  source show_elapsed.sql ;
```

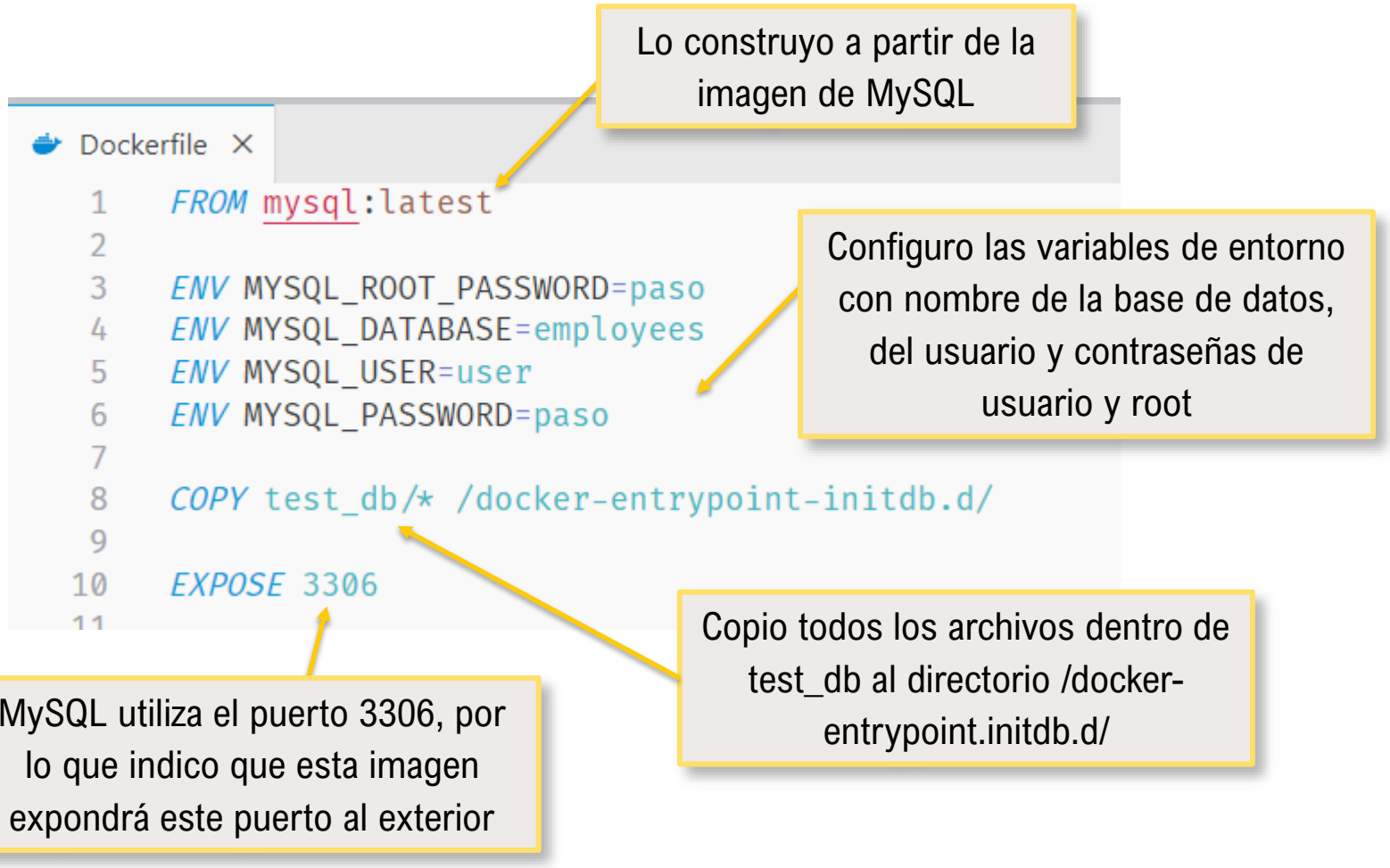


Los scripts dentro del directorio se ejecutarán por orden alfabético, así que conservo el orden con que se cargan en el fichero original

Y luego elimino estas líneas del fichero *employees.sql*

```
112  SELECT 'LOADING departments' as 'INFO';
113  source load_departments.dump ;
114  SELECT 'LOADING employees' as 'INFO';
115  source load_employees.dump ;
116  SELECT 'LOADING dept_emp' as 'INFO';
117  source load_dept_emp.dump ;
118  SELECT 'LOADING dept_manager' as 'INFO';
119  source load_dept_manager.dump ;
120  SELECT 'LOADING titles' as 'INFO';
121  source load_titles.dump ;
122  SELECT 'LOADING salaries' as 'INFO';
123  source load_salaries1.dump ;
124  source load_salaries2.dump ;
125  source load_salaries3.dump ;
126
127  source show_elapsed.sql ;
```

Y ya puedo crear el Dockerfile



Y construyo la imagen

Nombre que le voy a dar a la imagen

Directorio en que se encuentra el Dockerfile

```
PS C:\MySQL_Emp> docker build -t employees_mysql .
[+] Building 11.3s (5/7)
 => [1/2] FROM docker.io/library/mysql:latest@sha256:0596fa224c
```

```
PS C:\MySQL_Emp> docker image list
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
employees_mysql   latest    68c28b9b926f   49 seconds ago   973MB
```

Ahora ya podemos lanzar contenedores basados en esa imagen

Mapeamos el puerto 3306 del contenedor al puerto 3306 de la máquina física

```
PS C:\MySQL_Emp> docker run -p 3306:3306 employees_mysql
```

Si hemos hecho algo mal en la inicialización de la BD saltará aquí el error, ya que los scripts se ejecutan al crear el contenedor (al crear la imagen solo se copian)

Ejemplo: si dejo la carga de los otros ficheros en *employees.sql*

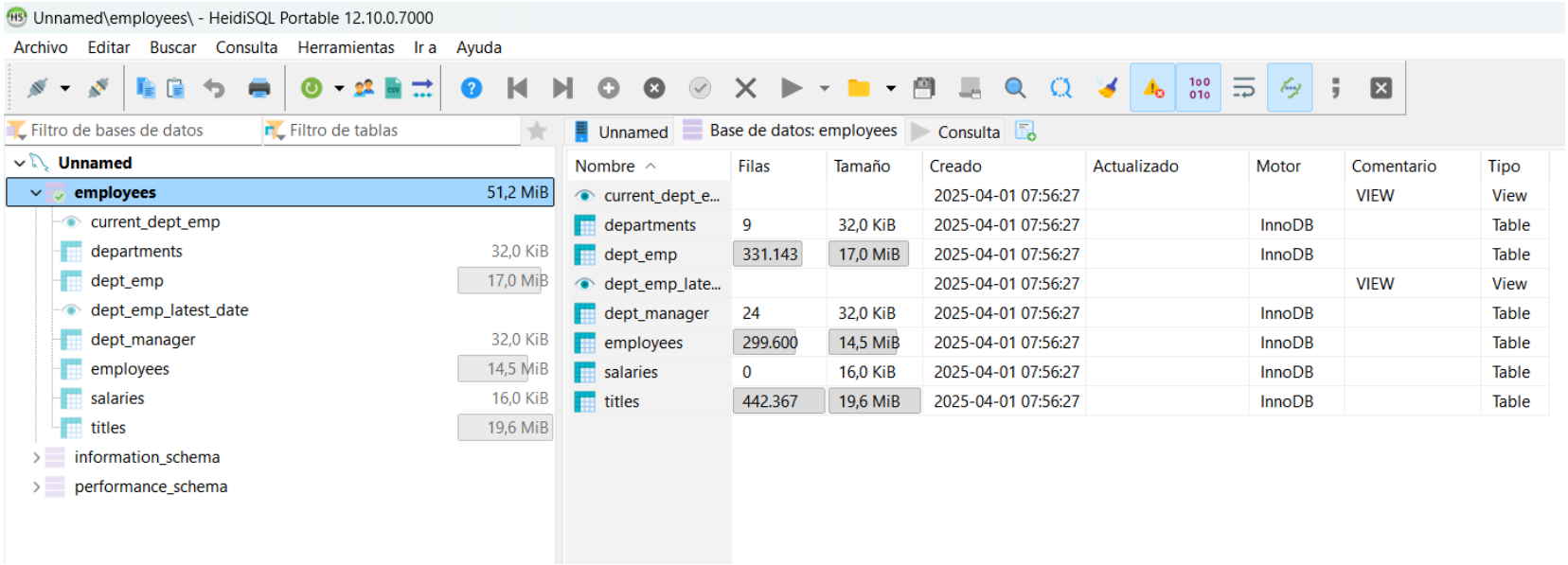
```
SELECT 'LOADING departments' as 'INFO';
source load_departments.dump ;
```

```
2025-04-01 07:50:21+00:00 [Note] [Entrypoint]: Creating database employees
2025-04-01 07:50:21+00:00 [Note] [Entrypoint]: Creating user user
2025-04-01 07:50:21+00:00 [Note] [Entrypoint]: Giving user user access to schema employees

2025-04-01 07:50:21+00:00 [Note] [Entrypoint]: /usr/local/bin/docker-entrypoint.sh: running
/docker-entrypoint-initdb.d/01_employees.sql
ERROR at line 113: Failed to open file 'load_departments.dump', error: 2
INFO
CREATING DATABASE STRUCTURE
INFO
storage engine: InnoDB
INFO
```

Si ha ido todo bien tendremos el contenedor funcionando

```
PS C:\MySQL_Emp> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS
 PORTS                                  NAMES
2cb840eccc0c   employees_mysql    "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes
 0.0.0.0:3306->3306/tcp, 33060/tcp   exciting_lederberg
```

Y podremos acceder a los datos a través del puerto 3306 del equipo



ADD

Este comando es similar a `COPY` con la diferencia de que es capaz de descomprimir archivos comprimidos (.tar, .tar.gz, .tgz) y también descargarlos desde una URL

```
14  # Añade y descomprime un tar.gz
15  ADD contenido.tar.gz /app/
16
17  # Descarga un archivo remoto
18  ADD https://example.com/logo.png /var/www/html/logo.png
```

WORKDIR

El comando `WORKDIR` sirve para definir el directorio de trabajo dentro del contenedor donde se ejecutarán los comandos posteriores (`RUN`, `CMD`, `COPY`, ...)

```
# Establece el directorio /app como espacio de trabajo
WORKDIR /app
# Copia el package.json al directorio /app
COPY package.json .
# Instala dependencias (se ejecuta en /app)
RUN npm install
# Copia el resto de los archivos
COPY . .
# El comando CMD también se ejecuta desde /app
CMD ["npm", "start"]
```

ENV

Se utiliza para definir **variables de entorno** que estarán disponibles durante la construcción y en los contenedores creados a partir de la imagen.

```
ENV JAVA_HOME=/opt/java

ENV APP_PORT=8080
EXPOSE ${APP_PORT}

ENV MYSQL_USER=victor
```

CMD / ENTRYPOINT

Definen cómo se ejecutará el contenedor una vez iniciado.

	CMD	ENTRYPOINT
Sobreescritura	Se puede sobrescribir con docker run	No se sobrescribe, solo se añaden argumentos
Uso típico	Parámetros por defecto	Comando principal del contenedor
Combinación	Si hay `ENTRYPOINT`, `CMD` actúa como sus argumentos	Si no hay `CMD`, se usan solo los argumentos de docker run

5

VOLÚMENES EN DOCKER

Los volúmenes en Docker son un mecanismo para **persistir datos** generados y utilizados por contenedores.

A diferencia del sistema de archivos efímero de los contenedores (que se borra al eliminarlos), los volúmenes mantienen la información de forma permanente y permiten compartirla entre contenedores.

Hay dos tipos de volúmenes:

- **Bind mounts**
- **Named volumes**

Bind mounts | Named volumes

Son un tipo de volumen que **vincula un directorio o archivos específico del sistema host** directamente a un contenedor.

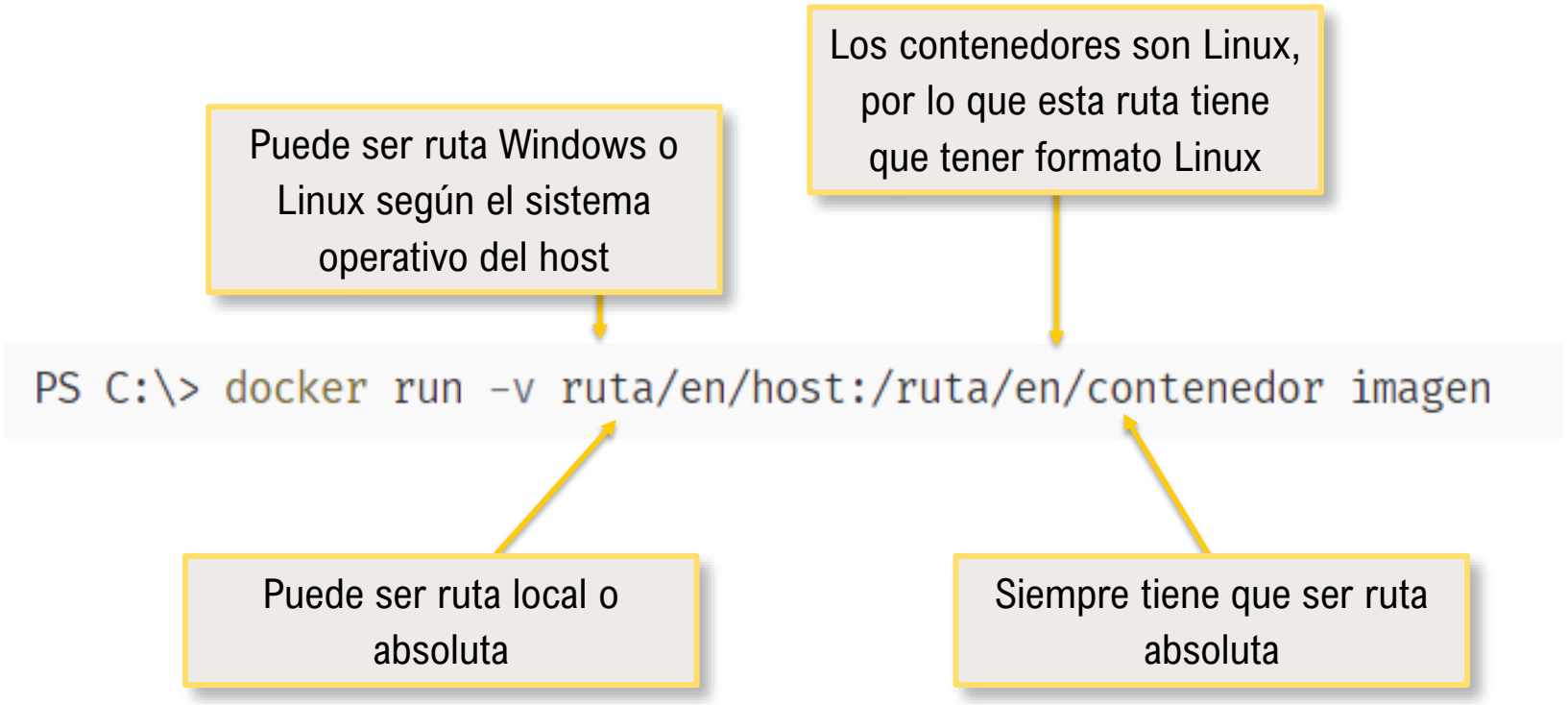
Permiten un control más manual sobre la ubicación de los datos.

Se suelen usar para:

- Desarrollo de aplicaciones (código sincronizado)
- Configuraciones personalizadas (archivos .env)
- Acceso a dispositivos del host en Linux (/dev)

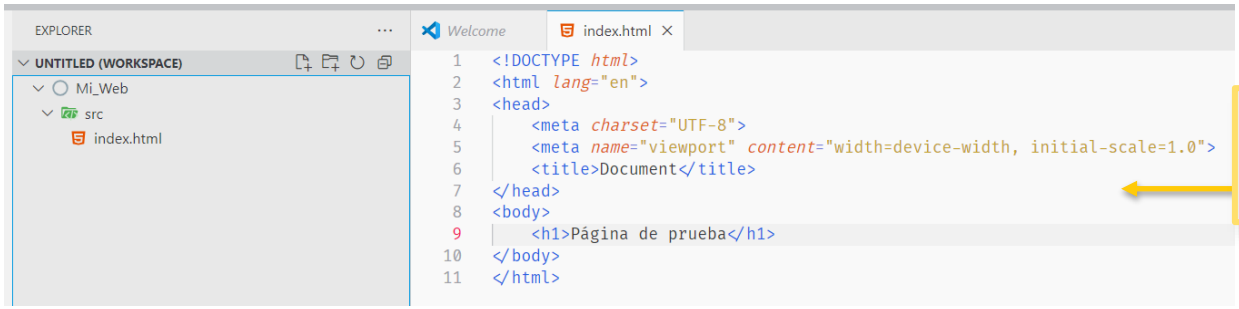
Bind mounts | Named volumes

Los puntos de montaje se indican al ejecutar el contenedor con docker run usando el **parámetro -v**



Bind mounts | Named volumes

Ejemplo: contenedor con Nginx que sirva una página alojada en host para desarrollo



Creo un directorio para mi proyecto

Lo lanzo en segundo plano

En Powershell esto se expande como directorio actual

```
PS C:\Mi_Web> docker run `
>> -d `
>> -v "$($PWD)\src:/usr/share/nginx/html/" `
>> -p 8088:80 `
>> nginx
09be730901e28d95ae4afa98ecdcc5ac813717db1c588c208255a5dbc5e5f830
```

nginx espera la página web en esta ruta

Bind mounts | Named volumes

Si vamos ahora al puerto en nuestro equipo veremos nuestra web

Cuando editemos los ficheros, al recargar la página veremos aplicados los cambios.

Bind mounts | **Named volumes**

Los **volúmenes con nombre** son completamente administrados por Docker, lo que los hace portables, seguros y eficientes.

Por defecto, en Windows se guardan en \\wsl.localhost\docker-desktop-data\data\docker\volumes

Sus ventajas son:

- **Portabilidad**, funcionan igual en cualquier host con Docker
- **Rendimiento**, están optimizados para E/S
- **Permisos automáticos**, ya que Docker gestiona los permisos
- **Backup fácil**, se pueden importar/exportar con herramientas estándar

Bind mounts | **Named volumes**

`docker volume create`

Crea un volumen con nombre

```
PS C:\Mi_Web> docker volume create mi_volumen
mi_volumen
```

Bind mounts | **Named volumes**

`docker volume list`

Muestra un listado de los volúmenes que tenemos en el equipo

```
PS C:\Mi_Web> docker volume list
DRIVER    VOLUME NAME
local     docker-elk_elasticsearch
local     docker-hadoop-spark_hadoop_datanode
local     docker-hadoop-spark_hadoop_historyserver
local     docker-hadoop-spark_hadoop_namenode
local     docker_data
local     mi_volumen
local     src
```

Bind mounts | **Named volumes**

`docker volume rm`

Elimina un volumen

```
PS C:\Mi_Web> docker volume rm src
src
```

Bind mounts | **Named volumes**

La forma de utilizarlos es análoga a los *bind mounts* mediante el parámetro **–v** del comando docker run

```
PS C:\> docker run `
>> -d `
>> -v mi_volumen:/ruta/en/contenedor `
>> imagen
```

6

REDES EN DOCKER

7

DOCKER COMPOSE

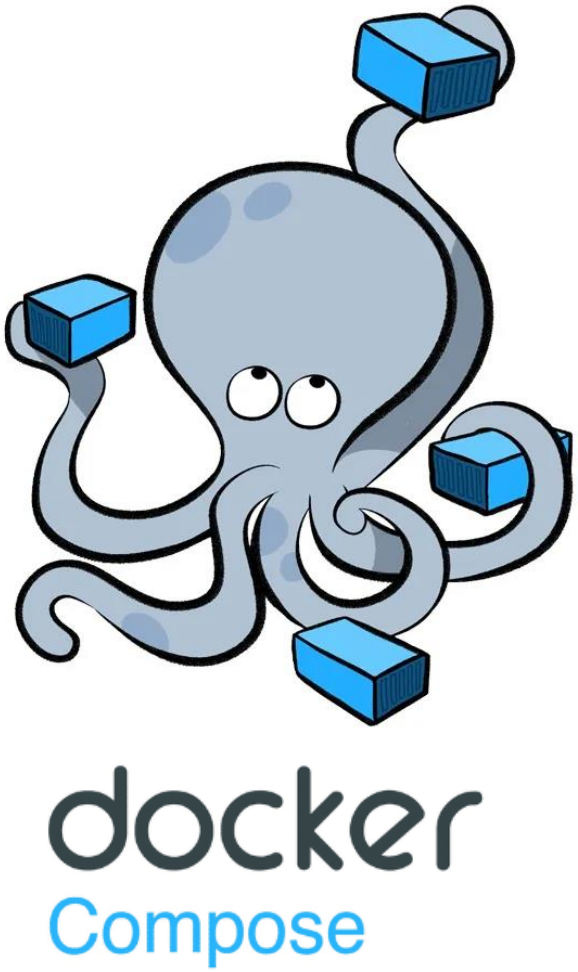
Docker Compose es una herramienta que permite definir y gestionar aplicaciones compuestas por **múltiples contenedores** Docker.

Se gestiona mediante un archivo de configuración en **formato YAML**.

YAML (*Yet Another Markup Languaje*) es un lenguaje de marcas ampliamente utilizado en herramientas como Kubernetes, Ansible, …

Es menos *verboso* que JSON o XML, con menos caracteres especiales.

Veamos sus características con un ejemplo:



```yaml
version: '3'
services:
  db:
    image: postgres:14
    container_name: db
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=odoo
      - POSTGRES_PASSWORD=paso
    ports:
      - '5432:5432'
    volumes:
      - ~/OdooDesarrollo/dataPGdev:/var/lib/postgresql/data
  odoo16:
    image: odoo:16
    container_name: odoo16
    environment:
      - HOST=db
      - USER=odoo
      - PASSWORD=paso
    ports:
      - '8069:8069'
```

Un fichero YAML está formado por pares clave-valor

El indentado determina el anidamiento, por ejemplo, *db* y *odoo16* están dentro de *services*

Importante: debe haber un espacio después de los dos puntos

No se puede usar tabulados, deben ser espacios

Las **listas** se indican con el símbolo guion (-)