

# Docker

# Docker

## Tabla de contenidos

- [Docker](#)
- [Tabla de contenidos](#)
  - [Introducción](#)
  - [Palabras clave y sus analogías](#)
  - [Recursos:](#)
  - [Instalación](#)
    - [Instalación en Windows](#)
    - [Docker Desktop](#)
    - [Docker Hub](#)
  - [Comandos Docker](#)
  - [Crear Contenedor](#)
    - [Mapeo de Puertos](#)
  - [Docker Run](#)
    - [Parámetros Docker Run](#)
    - [Usar VSCode y Docker](#)
    - [Dockerfile](#)
  - [Docker Compose](#)
    - [Pila LAMPP personalizada](#)
    - [Seccion1](#)

# Introducción

[Tabla de contenidos](#)

## Palabras clave y sus analogías

| Término Docker        | ¿Qué es? (Técnico)  | Analogía (Puerto Marítimo)                                  |
|-----------------------|---|---|
| Docker                | Plataforma que gestiona contenedores                        | 🚢 El barco portacontenedores que transporta todo            |
| Imagen                | Plantilla (read-only) para crear contenedores               | 📦 La carga estandarizada que irá dentro del contenedor      |
| Contenedor            | Instancia aislada creada desde una imagen                   | Un TEU (contenedor físico) en el barco                      |
| Dockerfile            | Script con instrucciones para construir una imagen          | La lista de embalaje o especificación de qué va en la caja  |
| <b>docker build</b>   | Crea una imagen a partir de un Dockerfile                   | 🏗 Empacar la caja según la lista                            |
| <b>docker pull</b>    | Descarga una imagen desde Docker Hub                        | ⬇ Recoger una caja ya embalada desde un depósito            |
| <b>docker run</b>     | Crea y arranca un contenedor desde una imagen               | Cargar y usar la caja en el barco, ya lista para trabajar   |
| <b>docker start</b>   | Enciende un contenedor que ya existe                        | ▶ Reutilizar un contenedor guardado, sin volver a armarlo   |
| <b>docker stop</b>    | Detiene un contenedor                                       | ▬ Apagar el contenedor, como cerrar la caja temporalmente   |
| <b>docker rm</b>      | Elimina un contenedor (no la imagen)                        | 🗑 Descartar una caja usada, pero no tirar el diseño         |
| <b>docker rmi</b>     | Elimina una imagen  | ✖ Eliminar el modelo de caja del catálogo                   |
| <b>docker ps</b>      | Lista los contenedores en ejecución                         | ⌚ Ver qué contenedores están cargados en el barco y activos |
| <b>docker images</b>  | Lista las imágenes disponibles localmente                   | 📦 Ver qué modelos de cajas tienes en tu almacén             |
| <b>docker exec</b>    | Ejecuta un comando dentro de un contenedor                  | 👉 Abrir una caja ya cerrada para revisar algo dentro        |
| <b>docker-compose</b> | Orquesta múltiples contenedores como servicios relacionados | El plan de carga del barco que dice qué cajas van juntas    |

## Recursos:

# Instalación

## [Tabla de contenidos](#)

Componentes necesarios: - Docker Desktop - Docker Compose - Docker CLI (Command Line Interface)

### Instalación en Windows

Windows no tiene Docker de forma nativa, así que Docker Desktop utiliza una máquina virtual con Linux (usando WSL2) para correr contenedores. ¿Qué necesitamos en Windows? - Windows 10/11 Pro o Home - WSL 2 (Subsistema de Windows para Linux) - Docker Desktop para Windows

Pasos: 1. Instalar WSL 2

```
wsl --install  
wsl -l -v
```

#### 2. Descargar Docker Desktop

- Desde: <https://www.docker.com/products/docker-desktop>
- Instalar como siempre (next, next, etc). Se integrará con WSL2 automáticamente. Podemos usarlo con terminal de Windows, PowerShell, o WSL (Ubuntu) (yo recomiendo PowerShell)

Y listo. El resto será igual que en Linux... (que es desde donde hago la documentación)

# Docker Desktop

## [Tabla de contenidos](#)

IMPORTANTE: Tenemos que tener arrancado Docker Desktop para poder usar los comandos que veremos mas adelante.

- <https://docs.docker.com/desktop/install/linux-install/>
- <https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>

```
# Debemos instalar el Terminal de GNOME
sudo apt install gnome-terminal

# Añadimos el repositorio de Docker
sudo apt update
sudo apt upgrade
sudo apt install apt-transport-https ca-certificates curl software-properties-common lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

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

# Volvemos a actualizar los repositorios e instalamos
sudo apt update
cd ~/Descargas
sudo apt install -y docker-ce docker-ce-cli containerd.io

# Damos permisos para ejecutar Docker sin sudo
sudo usermod -aG docker $USER
newgrp docker

# Ahora vamos a instalar Docker Desktop
sudo apt install gnome-terminal
# Comprobamos si el módulo KVM está habilitado
ls /dev/kvm
# (opcional) Habilitamos manualmente KVM
# modprobe kvm

# Nos vamos a esta URL y nos descargamos el .deb
# https://docs.docker.com/desktop/setup/install/linux/ubuntu/
# Instalamos el .deb
sudo apt-get install ./docker-desktop-amd64.deb
```

# Docker Hub

## [Tabla de contenidos](#)

- <https://hub.docker.com/> En este página tenemos todas las imágenes de Docker Si queremos ir a uno concreto, bajamos y le damos a la tecnología que queramos (por ejemplo NodeJS)
- [https://hub.docker.com/\\_/node](https://hub.docker.com/_/node)

```
# Para empezar debemos elegir una carpeta del equipo
cd ~/Plantillas
mkdir Docker
cd Docker

# Para bajar la ULTIMA IMAGEN de Node
docker pull node:current
# Este es mas liviano:
# node:current-alpine

# Si queremos una versión concreta
# Por ejemplo: nodeJS v18
# docker pull node:18

# Y ahora vamos a iniciararlo y probarlo:
echo "<h1>Hola desde Node</h1>" > index.html

# Y ejecutamos el contenedor (ojo, 1 comando):
# IMPORTANTE! Es la mejor opción. No usar varios pasos...
# Explico cada línea:
# docker run -> Inicia contenedor
# -d -> En segundo plano
# --name -> Nombre contenedor (se lo pongo yo)
# -v "$PWD":/web -> Monta carpeta actual DENTRO DEL CONTENEDOR COMO /web para meterlo archivos HTML
# -p 3000:3000 -> Mapea los puertos (local:docker)
# node:current -> Nombre imagen que usamos
# sh -c -> Comandos de consola a ejecutar en el contenedor
docker run -d --name nodeweb \
-v "$PWD":/web \
-p 3000:3000 \
node:current \
sh -c "npm install -g http-server && http-server /web -p 3000"

# Ya solo tenemos que probarlo en el navegador:
http://localhost:3000/

# Si quiero parar el contenedor:
docker stop nodeweb
# Ahora en el navegador da un error...
# Y si quiero borrar el contenedor
docker rm nodeweb

# Si repetimos el proceso con node:18, podemos ver como se visualiza la imagen y el contenedor en el docker desktop
```

# Comandos Docker

## [Tabla de contenidos](#)

Ahora vamos a ver todos los comandos de la introducción (tabla) usando MySQL5.7 y exponiendo el puerto 3305 (no el 3306 del MySQL8 que tengo en local) y añadiendo el uso de un cliente para poder usar ese MySQL en la consola local (ElementaryOS7|Mint) > IMPORTANTE: Docker Desktop TIENE que estar EJECUTÁNDOSE!

```
# 📦 Paso1: Descargamos la imagen (ver DockerHub)
cd ~/Plantillas/Docker
docker pull mysql:5.7

# Paso2: Creamos y ejecutamos el contenedor
docker run -d \
--name mysql57 \
-e MYSQL_ROOT_PASSWORD=clave123 \
-e MYSQL_DATABASE=tienda \
-p 3304:3306 \
mysql:5.7

# 🔍 Paso 3: Verificar que está corriendo (docker ps)
docker ps
# Paso 4: Entrar al contenedor y usar el cliente MySQL interno (docker exec)
docker exec -it mysql57 mysql -u root -p

# ✅ Paso 5: Conectarte desde consola local al contenedor (⚙️ Cliente Externo)
mysql -h 127.0.0.1 -P 3304 -u root -p

# 🗑️ Paso 6: Opcional - limpiar recursos
docker ps -a          # Ver contenedores detenidos
docker images         # Ver imágenes
docker stop mysql57   # Detener el contenedor
docker rm mysql57     # Eliminar el contenedor
docker rmi mysql:5.7  # Eliminar la imagen
```

# Crear Contenedor

## Tabla de contenidos

Para crear un contenedor lo primero es elegir una imagen que nos servirá de base y de ahí podremos agregar las tecnologías que queramos: > ADVERTENCIA: Cuidado con las imágenes en nuestro sistema. Ocupan bastante espacio.

```
# Bajamos la imagen de mongo
docker pull mongo

# Vemos las imágenes bajadas (ojo a la columna Repository)
docker images
# Para ver detalles de una imagen:
docker image inspect mongo

# Creamos un nuevo contenedor a partir del bajado
docker create mongo
# Para ponerle un nombre:
docker create --name miMongo mongo:latest

# Aparece una ID similar a esta...
# 971902f1b63debab44d7ae1a3358aabdc2b14a5a43b5ea016d21432b0b11cf76

# Ahora arrancamos el contenedor (usamos la ID de antes)
docker start 971902f1b63debab44d7ae1a3358aabdc2b14a5a43b5ea016d21432b0b11cf76
# arrancar un contenedor con nombre
docker start miMongo

# Ver TODOS los contenedores
docker ps -a
# Para visualizar los contenedores arrancados
docker ps
# Vemos que sale la ID recortada. Es PERFECTAMENTE VÁLIDA
# 971902f1b63d    mongo      "docker-entrypoint.s..."    friendly_volhard

# Para detener el contenedor podemos usar la ID recortada
docker stop 971902f1b63d
docker stop miMongo

# Para ver todos los contenedores creados, arrancados o no
docker ps -a

# Para borrar el contenedor también podemos usar su nombre
docker rm friendly_volhard
docker ps -a  # Ahora NO aparece nada

# OJO, la imagen sigue ahí...
docker images
# Podemos asignarle un nombre a nuestro contenedor creado
# docker create --name <nombre> <imagen_origen>
docker create --name miMongoDB mongo

# Ahora arrancamos nuestro contenedor con el nombre personalizado
docker start miMongoDB
docker ps  # Visualizamos contenedores arrancados

# ATENCIÓN: Aún NO podemos visualizar MongoDB. Ver siguiente apartado.
# Detenemos y borramos el contenedor
docker stop miMongoDB
docker rm miMongoDB
docker ps

# Ahora retomamos la creación del contenedor pero esta vez con parámetros adicionales:
docker run -d \
--name miMongoAvante \
-p 27017:27017 \
-v mongo_data:/data/db \
mongo
```

```

# Accedemos a Mongo desde el cliente del contenedor
docker exec -it miMongoAvante mongosh

# Ver BBDD en Mongo
show dbs
# Crear BBDD
use Avante

# Meter datos
db.alumnos.insertMany([
  { nif: "11A", nombre: "Fran", edad: 30, genero: false },
  { nif: "22B", nombre: "Noelia", edad: 25, genero: true },
  { nif: "33C", nombre: "Elias", edad: 35, genero: false },
  { nif: "44D", nombre: "Angela", edad: 25, genero: true }
])

# Ver las tablas
show collections

# Visualizar los datos
db.alumnos.find().pretty()

# Ahora salimos
exit

#borramos el contenedor y la imagen
docker stop miMongoAvante
docker rm miMongoAvante
docker images
docker rmi mongo

# (Avanzado) Con -v mongo_data hemos creado un volumen de datos persistidos (se mantienen independientes del contenedor). Podemos visualizarlos montando un contenedor temporal con:
docker run --rm -it -v mongo_data:/data busybox sh
cd /data
ls -la
# OJO, son datos binarios, que sólo MONGO puede ver. Pero no quita que si mas adelante vuelvo a montar el contenedor mongo, puedo visualizar esos datos
exit

# Si repito el proceso anterior con otro contenedor (por ejemplo mongoRecuperado), veré los datos
docker pull mongo
docker run -d \
  --name mongoRecuperado \
  -p 27017:27017 \
  -v mongo_data:/data/db \
  mongo

# Y vemos los datos!
docker exec -it mongoRecuperado mongosh
show dbs
use Avante
db.alumnos.find().pretty()
exit

# No olvidemos parar el contenedor, eliminarlo, y borrar la imagen si no la vamos a usar. Esta vez borramos también los datos
docker stop mongoRecuperado    # Para contenedor
docker rm mongoRecuperado      # Borra contenedor
docker ps -a                     # Ver contenedores
docker rmi mongo                 # Borra imagen
docker images                     # Ver imágenes
docker volume rm mongo_data     # Borra volumen (adios datos!)
docker volume ls                  # Ver volúmenes

```

MUY IMPORTANTE: Pasar al Dockerfile Si da tiempo, dar los apartados intermedios como práctica

# Mapeo de Puertos

## [Tabla de contenidos](#)

Una cosa es el puerto que usa el contenedor (en este caso 27017 de Mongo) y otra el puerto que vamos a emplear desde el Anfitrión (Host) Debemos mapear puertos:

```
# La estructura es -p<puerto_host>:<puerto:contenedor>
docker create -p27017:27017 --name miMongoDB mongo

# Arrancamos
docker start miMongoDB
docker ps
# Visualizamos en el navegador: http://localhost:27017/

# ¿Podemos dejar a docker que mapee por nosotros? SI, pero NO es recomendable
docker create -p27017 --name miMongoDB2 mongo
docker start miMongoDB2
docker ps

# Vemos como en PORTS sale algo asi:
# PORTS          NAMES
# 0.0.0.0:44427->27017/tcp  miMongoDB2
# 0.0.0.0:27017->27017/tcp  miMongoDB
# Y lo comprobamos en el navegador: http://localhost:44427/

# Detenemos y eliminamos
docker stop miMongoDB2
docker rm miMongoDB2

# Para visualizar los logs del contenedor
docker logs miMongoDB
# Si queremos que se MANTENGA A LA ESCUCHA
docker logs --follow miMongoDB
# Si entramos en el navegador, se añaden nuevas líneas...
# Para salir CTRL+C

# Y como en ocasiones anteriores, paramos y borramos
docker stop miMongoDB
docker rm miMongoDB
# Y borramos la imagen de Mongo
docker image rm mongo
```

# Docker Run

## [Tabla de contenidos](#)

Docker Run es un comando muy interesante de Docker porque comprende varios comandos que ya hemos visto: - Docker pull - Docker create - Docker start - docker logs –follow

```
docker run mongo
# Nos saldrá los logs. Para mos con CTRL+C

# Para inciar el contenedor sin logs
docker run -d mongo
# No descarga!!, ya tenemos la imagen bajada y el contendor creado
# Visualisamos el contenedor arrancado
docker ps

# Evidentemente podemos personalizar el comando
# primro paramos y borramos por la ID que nos devuelve el ps
docker stop b6955cef3485
docker rm b6955cef3485

# Y ahora si el comando personalizado
# docker run --name <nombre> -p<puerto_host>:<puerto_contenedor> <imagen>
docker run --name miMongoDB -p27017:27017 -d mongo
docker ps
```

# Parámetros Docker Run

## [Tabla de contenidos](#)

Con todo lo que ya sabemos podemos hacer una práctica interesante donde vamos a ver como crear un contenedor de MySQL y usar el SGBD por línea de comandos con el puerto que deseemos.

1. En primer lugar nos vamos a <https://hub.docker.com/>
2. Buscamos MySQL (<https://hub.docker.com/search?q=mysql>)
3. Pulsamos en el contenedor oficial ([https://hub.docker.com/\\_/mysql](https://hub.docker.com/_/mysql))
4. Vemos lo siguiente de la documentación (creada con MarkDown, el mismo lenguaje que uso aquí mismo)
  - some-mysql -> El nombre que queramos para nuestro contenedor
  - my-secret-pw -> Contraseña acceso mysql
  - tag -> Versión mysql.

```
# docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
docker run --name contenedorMySQL -e MYSQL_ROOT_PASSWORD=root -d mysql:latest
```

5. Con lo anterior, ya tendremos corriendo mysql en nuestro equipo dentro de su contenedor correspondiente. Si ya hemos instalado previamente MySQL en local, veremos la diferencia con los siguientes comandos:

```
# Conexión con el MySQL local
mysql -h 127.0.0.1 -P 3306 -u root -p
# Contraseña root
# Y creamos la BBDD de prueba
CREATE DATABASE mysqlHost;
SHOW DATABASES;
exit
```

6. Ahora nos conectaos con el bash (la terminal) de nuestro contenedor usando el comando

```
# docker exec -it <nombre_contenedor> bash

# Conexión con el MySQL del CONTENEDOR!
docker exec -it contenedorMySQL bash
mysql -u root -p
CREATE DATABASE mysqlContenedor;
SHOW DATABASES;
exit
# Y para salir del bash del contenedor
exit
# NOTA: para hacer un clear, hacemos lo siguiente
# EN EL BASH GENERAL del contenedor:
# alias clear='printf "\033c"'
# clear
```

7. Vemos que las BBDD de ambos son distintos.
8. Para acabar, procedemos a parar y eliminar el contenedor además de su imagen de origen:

```
docker stop contenedorMySQL
docker rm contenedorMySQL
docker images # para sacar la ID del contenedor
docker image rm 2d9aad1b5856 # OJO! la id puede ser distinta
```

# Usar VSCode y Docker

## [Tabla de contenidos](#)

Recurso: - <https://www.howtogeek.com/devops/how-to-edit-code-in-docker-containers-with-visual-studio-code/>

Vamos a ver como crear un contenedor de Apache (HTTPD) y editar un archivo HTML dentro desde VSCode (En local) 1. Nos vamos a Docker hub y buscamos apache 2. Vemos la página oficial ([https://hub.docker.com/\\_/httpd](https://hub.docker.com/_/httpd))

```
# descargo la imagen
docker pull httpd
```

3. Creamos el contenedor:

```
# dit -> Crear y ejecuta el contenedor en segundo plano
docker run -dit --name miApache -p8080:80 httpd:2.4
# En el navegador ya podemos verlo: http://localhost:8080/
# No confundir con el apache local: http://localhost/
```

4. Abrimos el VSConde e instalamos este plugin: Dev Containers (de Microsoft)
5. Pulsamos en CTRL + SHIFT + P (ejecutar comando)
6. Elegimos: "Explorador Remoto: foco en vista contenedores en desarrollo. (empezamos a escribir remoto)
7. Pulsamos en el icono de abajo a la izquierda de VS [Abrir una ventana remota] [><]
8. Pulsamos en [Adjuntar al contenedor que está en ejecución]
9. Seleccionamos el contenedor (el único que hay)
10. Se abre una nueva ventana y le damos a Archivo > Abrir carpeta
11. Abrimos esta ruta: /usr/local/apache2/htdocs/ > [Aceptar] > OJO: No es la normal de /var/www/html
12. Editamos el archivo html con lo que queramos y listo! Ya tendremos nuestro primer HTML dentro del contenedor.
13. Ahora, DESDE VS CODE podemos borrar el contenedor. Simplemente en la pestaña de Explorador Remoto pulsamos en [Eliminar Contenedor]
14. Para borrar la imagen, procedemos por consola:

```
docker images
# Esta vez borramos por repository:tag
docker image rm httpd:2.4
```

## Dockerfile

### [Tabla de contenidos](#)

Recurso: - <https://iescelia.org/ciberseguridad/serie-docker-como-crear-dockerfile/>

Vamos a ver como crear un contenedor que personalice una imagen base descargada de Docker Hub. Para ello usaremos el archivo Dockerfile (con la primera D en mayusculas)

1. Creamos el archivo Dockerfile en una carpeta propia

```
cd ~/Plantillas/Docker  
mkdir Contenedores-Docker  
cd Contenedores-Docker  
code Dockerfile # Nos abre el VS Code
```

2. Dentro del VSCode, procedemos a la personalización de la imagen en un contenedor propio:

```
#Imagen origen  
FROM ubuntu:22.04  
  
# Ejecutamos dentro del Ubuntu esto...  
RUN apt update && apt -y upgrade  
RUN apt install -y git
```

[!TIP] Si pregunta para instalar la extensión Docker de Microsoft, pulsar [Si].

3. Ahora nos toca construir la imagen

```
# Comando: docker build -t nombre_imagen archivo_dockerfile  
# OJO! Si estamos en la misma carpeta, pondremos:  
# Importante, poner el punto al final!  
docker build -t ubuntu-git .
```

4. Y nos creamos el contenedor a partir de esa imagen. Además ejecutamos el bash directamente (-it)

```
docker images  
# OJO, con --rm para el contenedor y lo BORRA cuando escribimos exit  
# Si no queremos ser tan radicales, omitir --rm  
docker run -it --rm --name prueba-ubuntu-git ubuntu-git  
  
# Vemos la versión de Ubuntu  
cd ~  
cat /etc/os-release  
# Vemos la versión de Git  
git --version  
  
# Volvemos al host y vemos las diferencias  
exit  
git --version # Será la misma que en el contendor!  
cat /etc/os-release
```

# Docker Compose

## Tabla de contenidos

Recursos: - <https://iescelia.org/ciberseguridad/serie-docker-estructura-de-configuracion-de-un-archivo-docker-compose/> - <https://www.kodetop.com/crea-tu-ambiente-de-desarrollo-php-mysql-con-docker/>

En esta práctica vamos a crearnos un archivo de docker-compose.yml que nos permitirá crear un contenedor con las imágenes de: - Apache2 - PHP 7.4 - MySQL5 > En el apartado siguiente veremos como crear un contenedor con las versiones mas nuevas...

1. Lo primero es visualizar que tenemos el docker-compose. Este se instala con el Docker desktop. Tan solo tenemos que visualizar su versión:

```
docker compose version
# Docker Compose version v2.22.0-desktop.2
# Visualizamos la ayuda
docker compose
```

2. Nos creamos una carpeta en nuestro directorio de proyectos de Docker y un archivo llamado docker-compose.yml

```
mkdir $HOME/Contenedores-Docker/lampp
cd $HOME/Contenedores-Docker/lampp
code docker-compose.yml
```

3. Incluimos lo siguiente en el archivo

```
version: '3'

volumes:
  mysql-data:
    name: lamp-data

# Ambos servicios irán conectados a la misma red
networks:
  network-id:
    name: lamp-network

services:
  mysql:
    image: mysql:5.6
    container_name: lamp-mysql
    environment:
      MYSQL_DATABASE: prueba
      MYSQL_USER: usuario
      MYSQL_PASSWORD: clave
      MYSQL_ROOT_PASSWORD: admin
    volumes:
      - mysql-data:/var/lib/mysql
    ports:
      - "3305:3306"
    restart: always
    networks:
      - network-id

  php:
    image: php:7.4-apache
    container_name: lamp-php
    ports:
      - "8080:80"
    volumes:
      - ./www:/var/www/html
    links:
      - mysql
    networks:
      - network-id
```

5. Ahora creamos la imagen a partir del docker-compose

```
docker compose up -d
```

## 6. E iniciamos una instancia de la imagen (el contenedor)

```
docker exec -ti lamp-mysql /bin/bash
```

## 7. Ahora toca probar el contenedor de PHP

```
# Nos creamos el archivo info.php por línea de comandos con la configuración del servidor local:  
echo "<?php phpinfo(); ?>" >> info.php  
# Ponemos en el navegador del host: http://localhost:8080/info.php
```

## 8. Y por último, toca probar el contenedor de MySQL

```
docker exec -ti lamp-php /bin/bash
```

```
# Ya dentro del contenedor  
mysql -u root -p # Clave admin  
# Visualizamos los usuarios:  
SELECT Host, User, Password FROM mysql.user;  
# Salimos de Mysql y del bash  
exit  
exit
```

## 9. Puedo personalizar el contenedor resultante del docker-compose, creando en el mismo directorio un Dockerfile. Por ejemplo, para instalar las extensiones mysqli y pdo\_mysql.

```
cd $HOME/Contenedores-Docker/lampp  
code Dockerfile
```

- Pongo esto en el Dockerfile

```
#Partimos de la imagen base del docker-compose  
FROM php:7.4-apache  
  
# Ejecuto los comandos de instalación de las extensiones  
# Puedo poner TODAS las extensiones que yo quiera separadas por espacios  
RUN docker-php-ext-install mysqli pdo_mysql
```

- Modifico el docker-compose

## 10. OJO, para nuestras prácticas siempre es bueno eliminar contenedores e imágenes:

```
# Detenemos y eliminamos los contenedores  
docker ps -a  
docker stop lamp-php && docker rm lamp-php  
docker stop lamp-mysql && docker rm lamp-mysql  
  
# Eliminamos las imágenes  
docker images  
  
# Si queremos saber los datos de una imagen:  
docker inspect image php:7.4-apache  
# Información general de Docker  
docker info  
  
# Y ahora si, borramos las imágenes:  
docker image rm php:7.4-apache  
docker image rm mysql:5.6
```

## Pila LAMPP personalizada

### [Tabla de contenidos](#)

- Recursos:
  - <https://blog.tkav.dev/running-your-lamp-stack-on-docker-containers>

En este práctica vamos a ver la utilidad final de los contenedores: - 1. Diseñar una imagen que contenga todas las tecnologías que necesitemos. - 2. Agregar un archivo de configuración que nos permita personalizar dicho entorno en función de los requerimientos de los Devops -<https://es.wikipedia.org/wiki/DevOps> - 3. Iniciar distintas instancias de dicha imagen (o conjunto de imágenes) para nuestro trabajo como desarrolladores.

- Los pasos a seguir serán los siguientes:

- 1. Clonamos el proyecto del recurso que he tomado como base (OJO, podemos hacerlo por nuestra cuenta, tan solo necesitaríamos tener archivos apache.conf y php.ini que son fácilmente localizables por internet)

```
git clone https://github.com/tkav/docker-lamp-stack
```

- 2. Evidentemente, este proyecto lo vamos a personalizar a nuestro gusto. Para ello nos vamos al archivo docker-compose.yml, donde pondremos lo siguiente:
    - En /docker-compose.yml ``yml version: "3.5" services:

## Servicio PHP

### He personalizado el nombre de las imágenes (image)

```
php: build: context: './php/' args: PHP_VERSION: ${PHP_VERSION}

# Si no se pone esto, toma como nombre el de la carpeta image: php:${PHP_VERSION:+${PHP_VERSION}}fpm-alpine
networks: - backend volumes: - ${PROJECT_ROOT}:/var/www/html/

# Variables de entorno para comunicar PHP -> MySQL environment: MYSQL_HOST: "${DB_HOST}" MYSQL_DATABASE: "${DB_NAME}" MYSQL_USER: "${DB_USERNAME}" MYSQL_PASSWORD: "${DB_PASSWORD}" INSTANCE_NAME: "${INSTANCE_NAME}" PROJECT_COLOR: "${PROJECT_COLOR}" container_name: ${DOCKER_IMAGE_PREFIX}php restart: always
```

## Servicio apache (incluyo redes FrontEnd y BackEnd)

```
apache: build: context: './apache/' args: APACHE_VERSION: ${APACHE_VERSION} image:
httpd:${APACHE_VERSION:+${APACHE_VERSION}}alpine depends_on: - php networks: - frontend - backend ports: - "${EXTERNAL_APACHE_PORT}:80" volumes: - ${PROJECT_ROOT}:/var/www/html/ container_name: ${DOCKER_IMAGE_PREFIX}apache restart: always
```

## Servicio de MySQL

```
mysql: image: mysql:${MYSQL_VERSION} command: - mysqld restart: always ports: -
"${EXTERNAL_MYSQL_PORT}:3306" volumes: - ./mysql:/data:/var/lib/mysql networks: - backend environment: TZ: "${TZ}" MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}" MYSQL_DATABASE: "${DB_NAME}" MYSQL_USER: "${DB_USERNAME}" MYSQL_PASSWORD: "${DB_PASSWORD}" container_name: ${DOCKER_IMAGE_PREFIX}mysql
```

## Redes y Volúmenes

```
networks: frontend: name: ${DOCKER_IMAGE_PREFIX}frontend backend: name: ${DOCKER_IMAGE_PREFIX}backend volumes:
data:
```

- 3. Aunque no lo vamos a tocar, al menos visualizamos el Dockerfile del Apache (he incluido algunos comentarios, no hace falta tocar nada):

```

- En /apache/Dockerfile
```dockerfile
# Definimos el argumento y lo ponemos el FROM (la imagen a descargar)
ARG APACHE_VERSION=""
FROM httpd:${APACHE_VERSION:+$APACHE_VERSION}-alpine

# Aprovechamos para actualizar EN ESA VERSIÓN!
# (Se quitan vulnerabilidades)
RUN apk update; \
    apk upgrade;

# Copiamos del archivo apache.conf el vhost para enviar solicitudes de php al contenedor php-fpm
COPY demo.apache.conf /usr/local/apache2/conf/demo.apache.conf
RUN echo "Include /usr/local/apache2/conf/demo.apache.conf" \
>> /usr/local/apache2/conf/httpd.conf

# Permitimos la reescritura de las URLs
RUN sed -i '/LoadModule rewrite_module/s/^#//g' /usr/local/apache2/conf/httpd.conf

RUN { \
    echo 'IncludeOptional conf.d/*.conf'; \
} >> /usr/local/apache2/conf/httpd.conf \
&& mkdir /usr/local/apache2/conf.d

# Exponemos "publicamente" los puertos 80 y 443 (del Contenedor!)
EXPOSE 80
EXPOSE 443

```

- 4. Lo mismo, para PHP
  - En /php/Dockerfile ```dockerfile # Definimos el argumento y lo ponemos el FROM (la imagen a descargar) ARG PHP\_VERSION="" # Aquí las etiquetas del PHP oficial de Docker FROM php:\${PHP\_VERSION:+\$PHP\_VERSION}-fpm-alpine

```
RUN apk update;
apk upgrade;
```

## Aquí podríamos poner la instalación de mas extensiones

```
RUN docker-php-ext-install mysqli
```

## Copiamos NUESTRO php.ini al del contenedor (en /usr/local)

```
COPY php.ini /usr/local/etc/php/conf.d/php.ini
```

- 5. Con todo listo, ya solo nos toca definir las variables de entorno que nos habrán pasado los devops:

```

> NOTA: Si .env no existe, copiamos y pegamos del sample.env
> Comando: cp sample.env .env
- En /.env
```env
PHP_VERSION=8.1
MYSQL_VERSION=8.1.0
APACHE_VERSION=2.4
TZ=Spain/madrid

DB_HOST=mysql
DB_ROOT_PASSWORD=admin
DB_NAME=prueba
DB_USERNAME=usuario
DB_PASSWORD=clave

PROJECT_ROOT=~/public_html

```

```
INSTANCE_NAME=DEV
PROJECT_COLOR=green
DOCKER_IMAGE_PREFIX=contenedor_dev_

EXTERNAL_APACHE_PORT=8080
EXTERNAL_MYSQL_PORT=8306
```

- 6. Una de las mejores cosas que tienen estos contenedores que vamos a crear es que los archivos de trabajo lo tendremos en el host, con los que se comunicarán directamente. De este modo, en la carpeta public\_html nos creamos el siguiente archivo info.php
  - En /public\_html/info.php

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

- 7. Creamos las imágenes y los contenedores:

```
# Me meto en la raiz del proyecto: docker-lamp-stack
docker compose build
docker compose up -d
```

- 8. Y ya solo nos queda empezar a probar (en el navegador):
  - http://localhost:8080/
  - http://localhost:8080/info.php

9. Lo genial de esta configuración es que podemos acceder al MySQL del contenedor desde el Cliente MySQL del Host

```
mysql -h 127.0.0.1 -P 8306 -u root -p      # Contraseña: admin
SHOW DATABASES;      # veremos prueba, creado en el .env del paso 5
```

10. OJO, para nuestras prácticas siempre es bueno eliminar contenedores e imágenes una vez que dejemos de usarlas: > [!NOTE]

> Podemos también hacer lo mismo con la interfaz gráfica de Docker.

```
# Detenemos y eliminamos los contenedores
docker ps -a

# Del tirón! (poniendo / podemos concatenar varias líneas...)
docker stop contenedor_dev_apache && docker rm contenedor_dev_apache /
docker stop contenedor_dev_mysql && docker rm contenedor_dev_mysql /
docker stop contenedor_dev_php && docker rm contenedor_dev_php

# Eliminamos las imágenes
docker images

# Borramos las imágenes:
# Del tirón en una sola línea!
docker image rm httpd:2.4-alpine /
docker image rm php:8.1-fpm-alpine /
docker image rm mysql:8.1.0
```

# Seccion1

[Tabla de contenidos](#)

#...