DAW
Despliegue de aplicaciones web
4. Contenedores

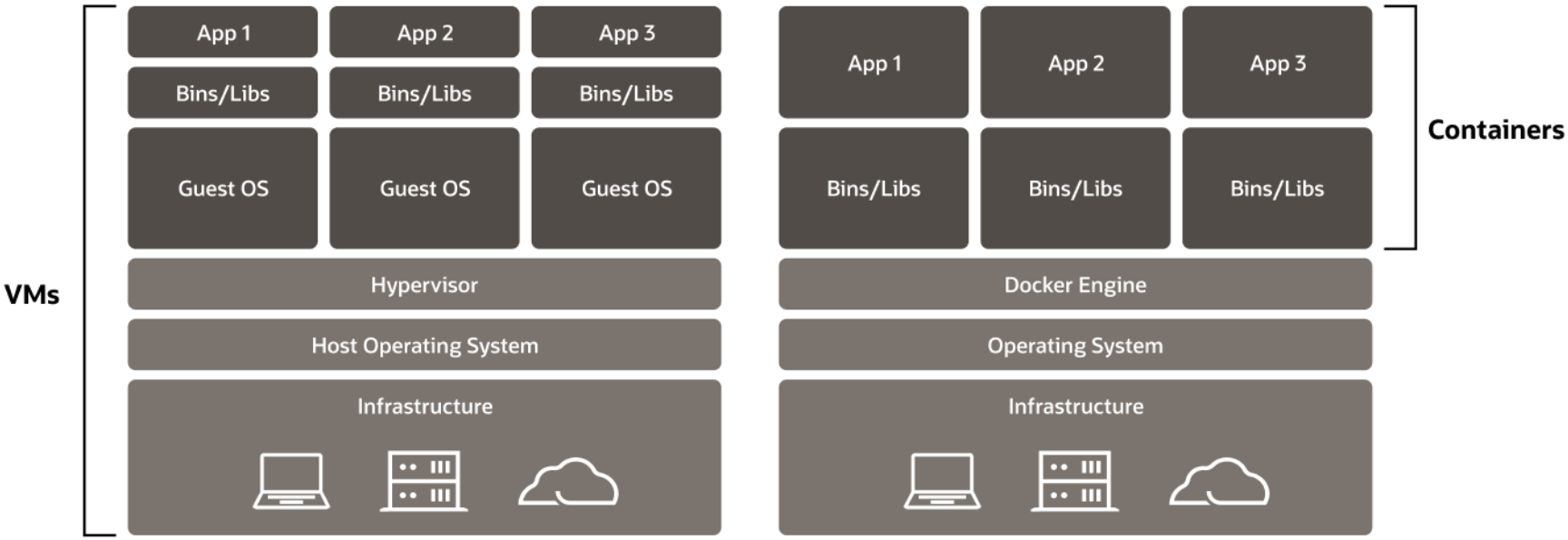
Juan Jesús Tortajada Cordero
jtortajada@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción (Docker)

Máquinas virtuales vs Contenedores



Virtual Machines

- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

Containers

- Containers include the app and all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host operating system.
- Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud.

oracle.com

1. Introducción (Docker)

Máquinas virtuales vs Contenedores: el problema…

Imagina que necesitas ejecutar dos aplicaciones en un equipo:

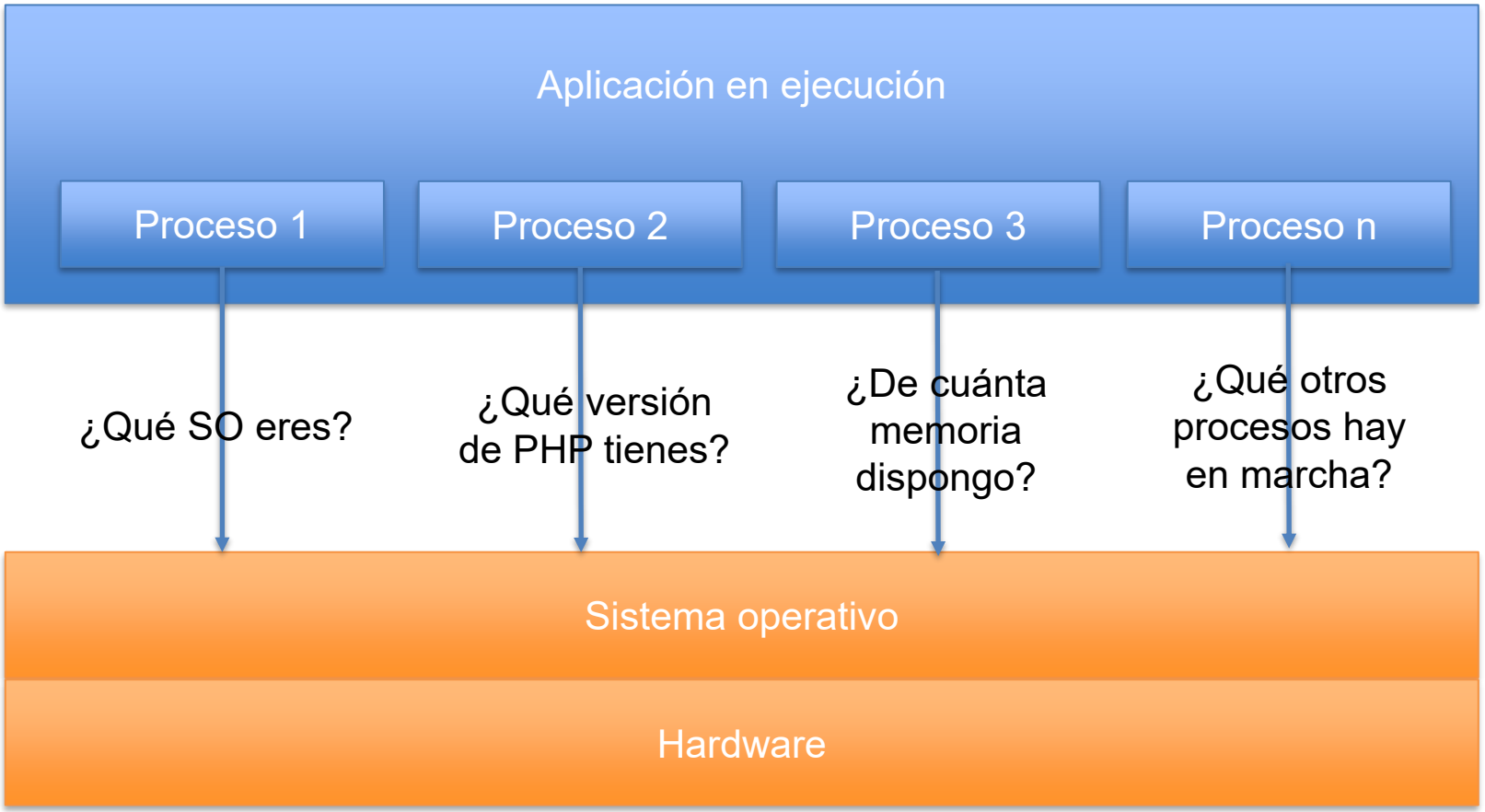
- Aplicación 1: requiere la última versión de {PHP/Java/…}
- Aplicación 2: requiere una versión antigua de PHP {PHP/Java/…}, con características incompatibles con la última versión

  ➔ Posible solución: máquinas virtuales separadas para cada aplicación, aparte del sistema operativo de la máquina huésped
  ➔ Posible problema: ineficiencia (sistemas operativos separados, reserva de HW, etc.)

Idea de la "**containerización**": utilizar solo un sistema operativo

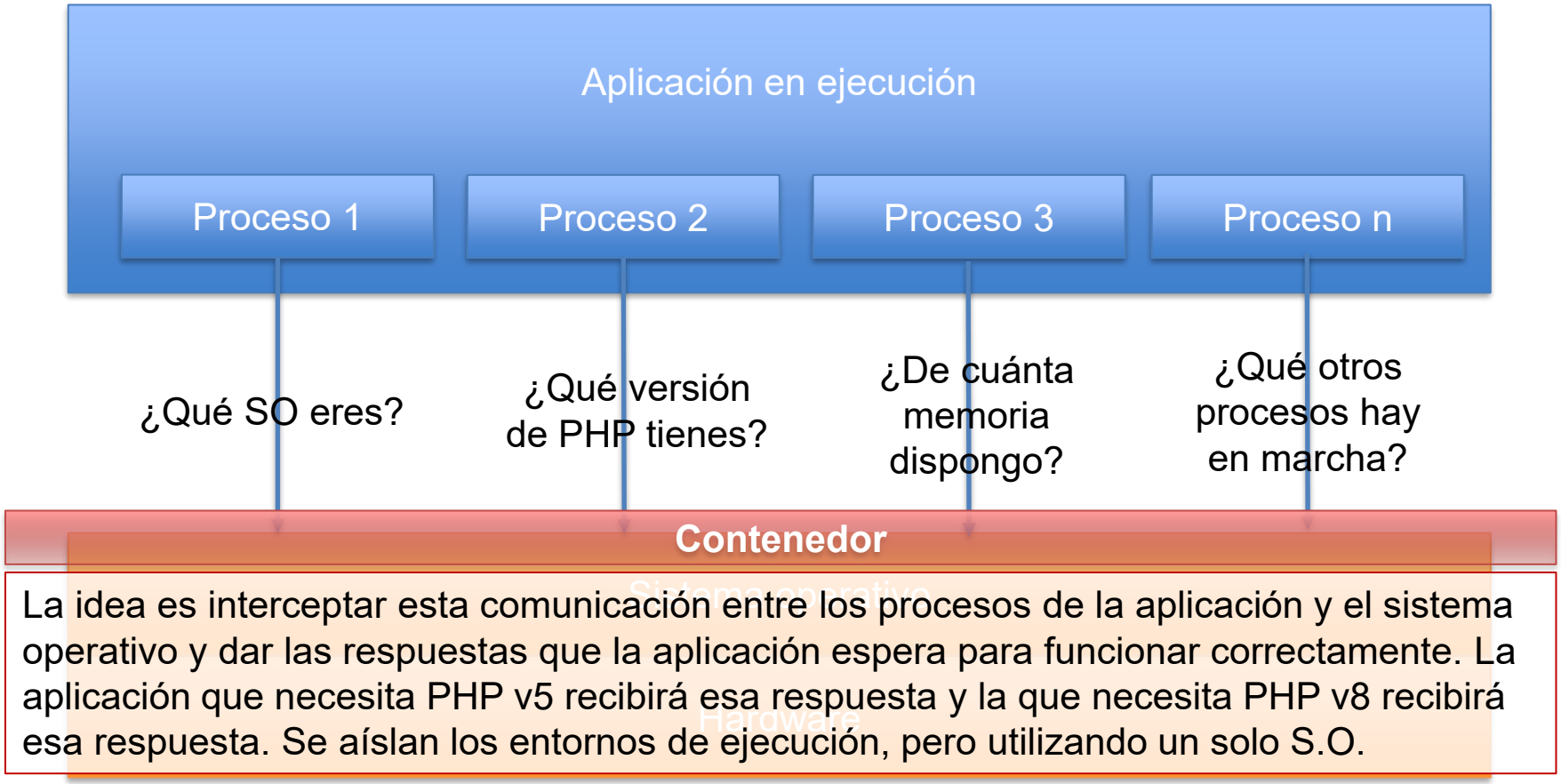
1. Introducción (Docker)

¿Cómo funciona Docker? En una aplicación normal (sin Docker)…



1. Introducción (Docker)

¿Cómo funciona Docker? La idea de Docker



La idea es interceptar esta comunicación entre los procesos de la aplicación y el sistema operativo y dar las respuestas que la aplicación espera para funcionar correctamente. La aplicación que necesita PHP v5 recibirá esa respuesta y la que necesita PHP v8 recibirá esa respuesta. Se aíslan los entornos de ejecución, pero utilizando un solo S.O.

1. Introducción (Docker)

¿Cómo funciona Docker?

Cada contenedor también tiene **recursos reservados** (memoria, usuarios, red, volúmenes,…) para poder ejecutar una aplicación.

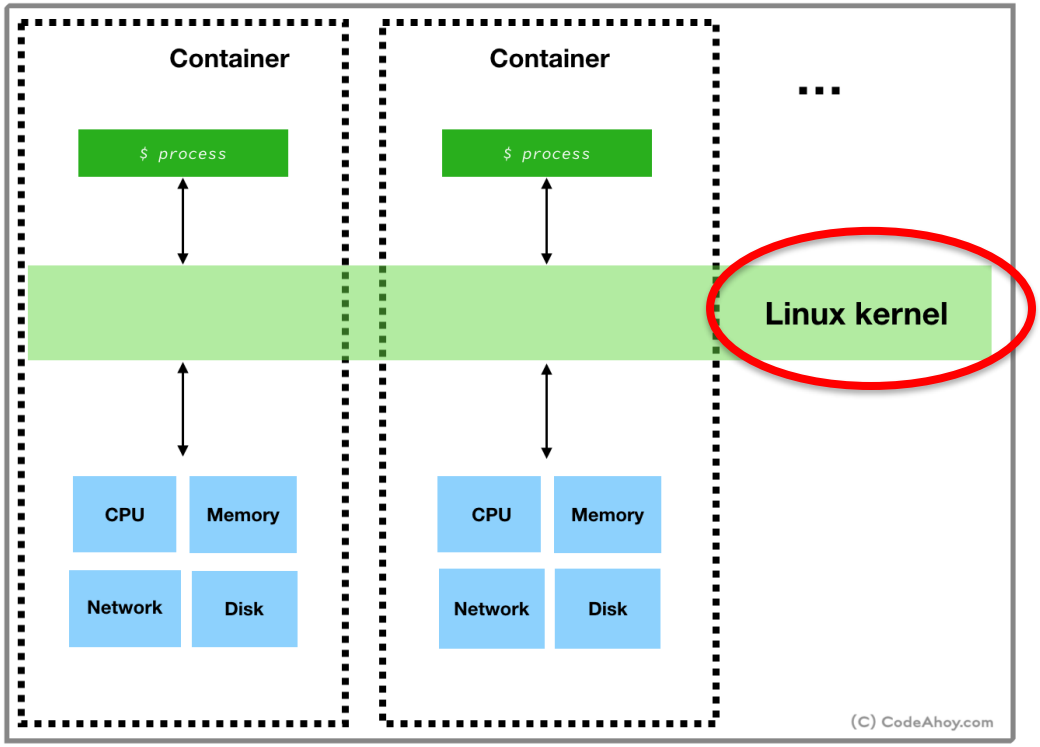
Este concepto es similar al de la máquina virtual, pero la diferencia está en que la manera de aislar estos recursos entre contenedores es mucho más **eficiente** que con una máquina virtual.

Para ello, los contenedores utilizan unas características del ***kernel* de Linux**. Estas son dos de las principales:

- **namespaces**: cada proceso tiene un espacio propio e independiente para interactuar con la CPU (p.ej. procesos en *namespaces* distintos pueden utilizar un mismo puerto, como el 80).
- **cgroups**: permiten poner límites a los recursos disponibles (p.ej. RAM, recursos de red, usuarios, espacio en disco duro, etc.).

1. Introducción (Docker)

¿Cómo funciona Docker?



El contenedor utiliza el *kernel* de Linux… ¿y si el SO es Win/Mac?

Se instala una ligera máquina virtual Linux que se ejecuta en Hyper-V (Win) o en HyperKit (Mac), hipervisores que incorporan estos SOs para ejecución de máquinas virtuales.

En Win también se utiliza WSL2 (Windows Subsystem for Linux), que permite ejecutar Linux de manera nativa en Windows.

También existen contenedores nativos de Windows, que utilizan directamente su *kernel*.

2. Instalación y conceptos básicos

Instalación

Windows: descargar e instalar Docker Desktop (igual que lo visto en el Tema 1) (https://www.docker.com/products/docker-desktop/)

Linux (Ubuntu): desde la MV Ubuntu o si estás ejecutando Ubuntu en nativo.

1. Actualizar paquetes de sistema: `sudo apt-get update  sudo apt-get upgrade`
2. Instalar dependencias: `sudo apt-get install apt-transport-https ca-certificates curl software-properties-common`
3. Añadir clave GPG de Docker: `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
4. Añadir repositorio: `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
5. Actualizar base de datos de paquetes: `sudo apt-get update`
6. Instalar Docker: `sudo apt-get install docker-ce`
7. Verificar la instalación (debe aparecer como activo): `sudo systemctl status docker`

2. Instalación y conceptos básicos

Conceptos

Demonio (*daemon*): proceso principal de Docker. Gestiona los objetos de Docker (imágenes, contenedores, redes, volúmenes, etc.) y se comunica con otros demonios Docker.

Cliente: herramienta para interaccionar con el demonio.

Registro: almacén de imágenes. Destaca Docker Hub como registro público.

Imagen: plantilla para crear un contenedor Docker. Puede estar basada en otras imágenes.

Contenedor: instancia en ejecución de una imagen. Parte de la imagen y admite diferentes opciones de configuración.

Volumen: permiten la persistencia de datos cuando el contenedor no está en ejecución. También se utilizan para compartir datos entre contenedores.

Servicio: define cómo se debe ejecutar un contenedor y permite escalar el contenedor a múltiples instancias (réplicas) distribuidas en un clúster de Docker.

3. Trabajo con imágenes y contenedores

Comandos básicos

Verificar instalación: `docker --version`

Descargar una imagen desde Docker Hub: `docker pull hello-world`

Listar imágenes descargadas: `docker images`

Ejecutar un contenedor (y descarga si no está en local): `docker run hello-world`

Listar contenedores en ejecución: `docker ps`

Listar todos los contenedores (ejecución y detenidos): `docker ps -a`

Detener un contenedor: `docker stop <id_contenedor>`

Eliminar un contenedor: `docker rm <id_contenedor>`

Eliminar una imagen: `docker rmi <id_imagen>`

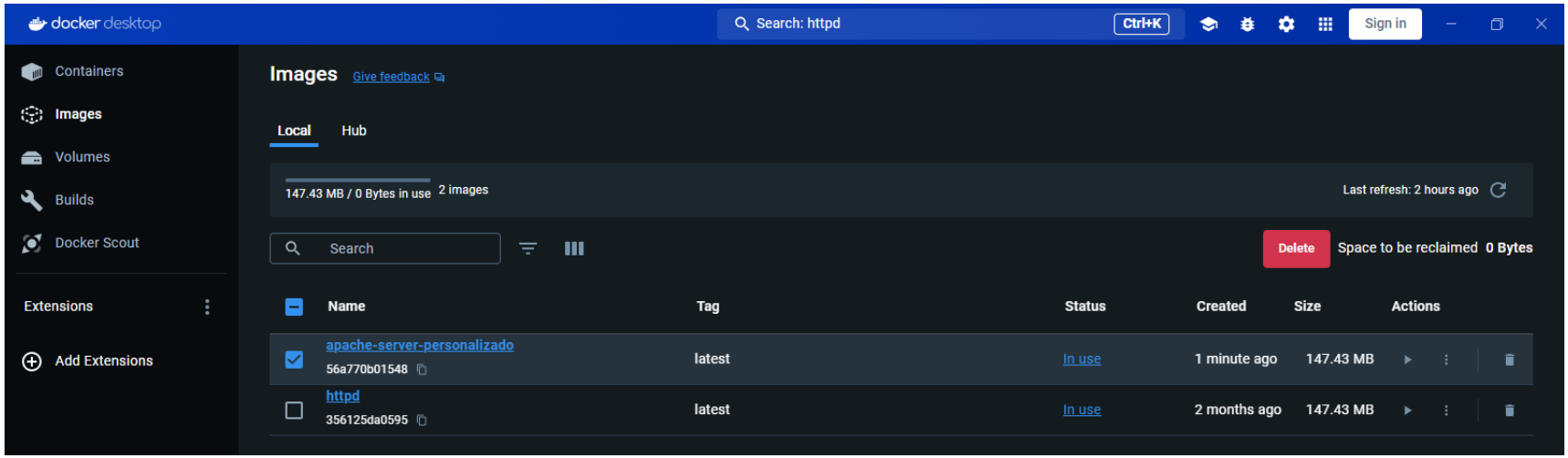
Eliminar **todo** lo que no esté en uso: `docker system prune -a`

https://docs.docker.com/reference/

3. Trabajo con imágenes y contenedores

Comandos básicos

Es aconsejable aprender a manejarse con la terminal o línea de comandos, ya que en muchas ocasiones (por ejemplo, servidores basados en Linux) no se dispone de interfaz gráfica. No obstante, en Windows puedes utilizar Docker Desktop, tal y como vimos en el tema 1.



3. Trabajo con imágenes y contenedores

Comandos básicos (ejercicio práctico)

1. Descarga y ejecuta la imagen “nginx”:

   ```
   docker pull nginx
   ```

   ```
   docker run -d -p 8080:80 nginx
   ```

2. Verifica que el contenedor está en ejecución: `docker ps`
3. En un navegador, introduce la URL: http://localhost:8080
4. O, si estás accediendo desde otra máquina: http://IP:8080

   NOTA: utiliza las instrucciones `ip a` o `ip addr` para averiguar la IP de la MV Linux

5. Detén y elimina el contenedor (utiliza el `id_contenedor` que corresponda):

   ```
   docker stop <id_contenedor>
   ```

   ```
   docker rm <id_contenedor>
   ```

3. Trabajo con imágenes y contenedores

`docker run` (también `docker container run`): ejecuta un contenedor.

Uso básico: `docker run [opciones] <nombreImagen>`

Opciones:

- `-d`: modo *daemon*: indicar que es un servidor (demonio).
- `-it`: modo interactivo: interactuar en la consola con la aplicación.
- `--restart always`: reinicia el contenedor si se reinicia la máquina real.
- `--name nombreContenedor`: asigna un nombre para tenerlo identificado.
- `--hostname nombreHost`: indica el nombre DNS que tendrá la máquina.
- `-p puertoExterno:puertoInterno`: mapea el puerto de la máquina real con el del software que se ejecuta en el contenedor.
- `-e nombreVariableEntorno:valorVariableEntorno`: asignar valores a las variables de entorno del contenedor (cada imagen tiene sus variables de entorno).
- `-v carpetaMaquinaReal:carpetaContenedor`: mapea una carpeta de la máquina real a una carpeta del contenedor (persistencia de datos).

https://docs.docker.com/reference/

3. Trabajo con imágenes y contenedores

`docker exec` (también `docker container exec`): ejecuta un comando en un contenedor que está en ejecución. Es importante que el comando sea un ejecutable.

Uso básico: `docker exec [opciones] <nombreContenedor> comando [argumentos]`

Opciones:

- `-d`: modo *detached*: ejecuta el comando en segundo plano.
- `-e`: indicar variables de entorno.
- `--env-file`: especifica un fichero donde leer variables de entorno.
- `-it`: modo interactivo: permite introducir varios comandos (sería el equivalente a una conexión SSH a otro sistema).

Comando: `/bin/bash` (Debian/Ubuntu) o `/bin/sh` (Alpine)

Ejemplo: descarga, ejecuta y accede a Linux Alpine

```
docker pull alpine
docker run -d -it --name alpine alpine
docker exec -it alpine /bin/sh
```

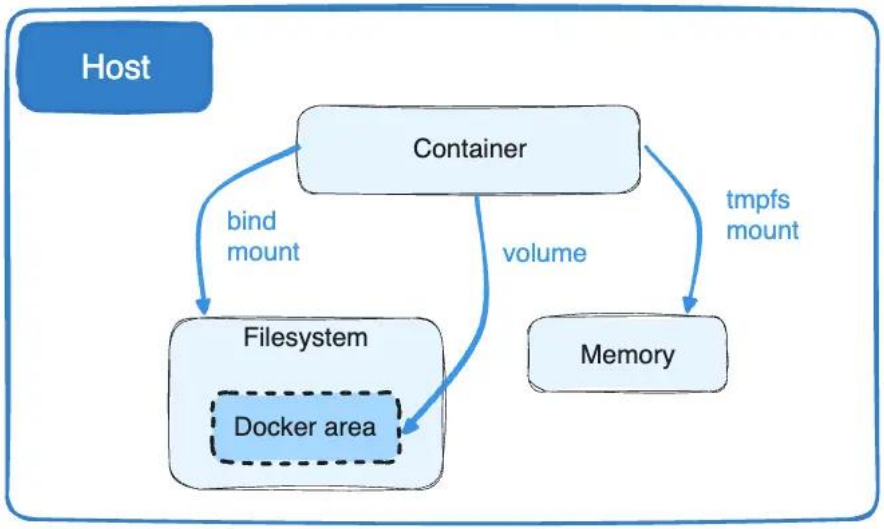
Estamos “dentro” de Alpine

```
/ # ls -l
total 56
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 bin
drwxr-xr-x    5 root     root           360 Jul 16 11:18 dev
drwxr-xr-x    1 root     root          4096 Jul 16 11:18 etc
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 home
drwxr-xr-x    6 root     root          4096 Jun 18 14:16 lib
drwxr-xr-x    5 root     root          4096 Jun 18 14:16 media
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 mnt
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 opt
dr-xr-xr-x  223 root     root             0 Jul 16 11:18 proc
drwx------    1 root     root          4096 Jul 16 11:20 root
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 run
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 sbin
drwxr-xr-x    2 root     root          4096 Jun 18 14:16 srv
dr-xr-xr-x   11 root     root             0 Jul 16 11:18 sys
drwxrwxrwt    2 root     root          4096 Jun 18 14:16 tmp
drwxr-xr-x    7 root     root          4096 Jun 18 14:16 usr
drwxr-xr-x   12 root     root          4096 Jun 18 14:16 var
/ #
```

4. Persistencia y volúmenes

Los contenedores son instancias en ejecución de una imagen. Por definición, cuando terminan su ejecución, todos los datos que no estuvieran en la imagen se perderán.

Los volúmenes son el mecanismo que emplea Docker para almacenar datos de forma persistente, de forma que estén disponible en futuras ejecuciones del contenedor.



En este esquema hay 3 formas en que el contenedor gestiona los datos:

***bind mount* (montaje de enlace)**

***volume* (volumen)**

tmpfs mount (montaje tmpfs): lo obviaremos porque hace referencia a datos que no requieren persistencia

https://docs.docker.com/storage/volumes/

4. Persistencia y volúmenes

***Volume* (volumen) vs *bind mount* (montaje de enlace)**

Característica	*Volume*	*Bind mount*
Gestión	Gestionado por Docker	Gestionado por el usuario
Ubicación	Ruta de Docker en el sistema de ficheros (/var/lib/docker/volumes)	Cualquier ruta en el equipo host
Portabilidad	Más portable	Menos portable
Desempeño	++	+
Seguridad	Mayor seguridad	Menor seguridad
Configuración	No requiere rutas absolutas del host	Requiere rutas absolutas del host
Uso compartido (entre contenedores)	Sencillo	Puede presentar riesgos

4. Persistencia y volúmenes

***Volume* (volumen) vs *bind mount* (montaje de enlace)**

Característica	*Volume*	*Bind mount*
Gestión	Gestionado por Docker	Gestionado por el usuario
Ubicación		ipo host
Portabilida		
Desemper		
Seguridad		
Configura		s del host
Uso compartido (entre contenedores)	Sencillo	Puede presentar riesgos

¿Cuándo usar volúmenes o montaje de directorio?
Volúmenes: cuando el contenedor necesita una base de datos.
Montaje de directorio: cuando tenemos el código de una aplicación (web o de otro tipo) en la máquina host y queremos que se ejecute en el contenedor.

NOTA: No es recomendable acceder a los volúmenes que cree Docker, especialmente cuando el contenedor esté en ejecución, ya que es el propio Docker quien los gestiona.

4. Persistencia y volúmenes

Volúmenes - Comandos básicos

Crear un volumen: `docker volume create <nombreVolumen>`

Listar volúmenes: `docker volume ls`

Obtener información de un volumen: `docker volume inspect <nombreVolumen>`

Borrar un volumen (se pierde toda la información): `docker volume rm <nombreVolumen>`

Eliminar volúmenes no utilizados: `docker volume prune`

https://docs.docker.com/storage/volumes/

4. Persistencia y volúmenes

Volúmenes - Ejemplo: volumen para alojar una base de datos MariaDB

Crear un volumen: `docker volume create volumenMariaDB`

Ejecutar el contenedor de MariaDB con el volumen que acabamos de crear:

```
docker run -d --name contenedorMariaDB -e
MYSQL_ROOT_PASSWORD=miContrasenya -e MYSQL_DATABASE=miBaseDeDatos -v
volumenMariaDB:/var/lib/mysql mariadb:latest
```

Detalle de los parámetros utilizados:

`-d`: ejecuta el contenedor en segundo plano.
`--name contenedorMariaDB`: asignamos un nombre al contenedor.
`-e MYSQL_ROOT_PASSWORD=miContrasenya`: contraseña de root para MariaDB.
`-e MYSQL_DATABASE=miBaseDeDatos`: crea una base de datos inicial.
`-v volumenMariaDB:/var/lib/mysql`: monta el volumen en el directorio `/var/lib/mysql` que es donde MariaDB (Docker) almacena sus datos.
`mariadb:latest`: usa la imagen de MariaDB más reciente que esté en Docker Hub.

4. Persistencia y volúmenes

Volúmenes - Ejemplo: persistencia de cambios en el volumen

Al levantar un contenedor con un volumen, los cambios que se realicen en la base de datos van a persistir mientras el contenedor no se elimine. En caso de eliminarlo y volverlo a levantar, los datos serán otra vez los que inicialmente tenía la imagen, por lo que es posible que se pierdan. Para evitarlo, se puede hacer lo siguiente:

1. Exportar la base de datos desde el contenedor MariaDB actual:
   ```
   docker exec CONTAINER_ID /usr/bin/mysqldump -u usuarioBDD -
   pcontraseñaBDD nombreBDD > ultimo_dump.sql
   ```

2. Crear un `Dockerfile` personalizado con el contenido del fichero `ultimo_dump.sql`:

```
FROM mariadb:latest

ENV MYSQL_ROOT_PASSWORD=contraseñaRoot
ENV MYSQL_DATABASE=nombreBDD
ENV MYSQL_USER=usuarioBDD
ENV MYSQL_PASSWORD=contraseñaBDD

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

EXPOSE 3306
```

➔ Más adelante veremos cómo crear y utilizar una imagen personalizada

NOTA: en la instrucción para exportar la base de datos aparece la contraseña precedida de la opción “-p” (sin espacio entre medias). Esta instrucción obliga a introducir la contraseña en el *shell,* lo que se considera no seguro, ya que se quedaría en el historial. Se puede utilizar la opción “-p” sin nada a continuación para que se solicite la contraseña de manera interactiva, evitando así este riesgo de seguridad.

```
docker exec CONTAINER_ID /usr/bin/mysqldump -u usuarioBDD -p nombreBDD > ultimo_dump.sql
```

4. Persistencia y volúmenes

Montaje de directorio (*bind mount*)
Ejemplo: contenedor con Apache en el que se ejecuta una app desarrollada en local

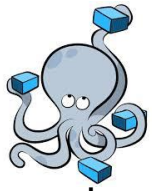
Imaginemos que el proyecto web local está alojado en `C:\xampp\htdocs\app`

Ejecutar el contenedor Apache con *bind mount*: `docker run -d --name contenedorApache -p 8080:80 -v C:\xampp\htdocs\app:/var/www/html httpd:latest`

Recordemos los parámetros:

`-d`: ejecuta el contenedor en segundo plano (*detached*).
`--name contenedorApache`: asigna un nombre al contenedor.
`-p 8080:80`: mapea el puerto 80 del contenedor al puerto 8080 del host para acceder a la web en http://localhost:8080.
`-v C:\xampp\htdocs\app:/var/www/html`: usa un *bind mount* para enlazar `C:\xampp\htdocs\app` en el host a `/var/www/html` en el contenedor.
`httpd:latest`: usa la imagen más reciente de Apache HTTP Server de Docker Hub.

5. Gestión de múltiples contenedores



Docker permite gestionar contenedores de manera individual, pero es habitual que el trabajo de desarrollo requiera el uso de diferentes aplicaciones y servicios. Por ejemplo, para desarrollar una aplicación web puede ser necesario el servidor web (p.ej. Apache), una base de datos (p.ej. MariaDB) y una aplicación de gestión de base de datos (p.ej. phpMyAdmin). Esto supone gestionar varios contenedores por separado.

➔ Docker Compose ←

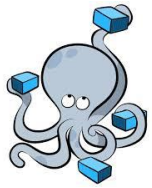
Herramienta que simplifica la definición y administración de aplicaciones multicontenedor. Incorporada en Docker Desktop (Win). En Linux: `sudo apt install docker-compose`

Ventajas:

- ✓ Definición de contenedores, redes y volúmenes en un único archivo YAML (.yaml/.yml).
- ✓ Disminución de redundancias y comandos repetitivos.
- ✓ Definir entornos de desarrollo coherentes para trabajo en equipo.
- ✓ Definir dependencias entre servicios, asegurando el orden adecuado.
- ✓ Definir redes personalizadas para que los servicios se comuniquen.
- ✓ Definir volúmenes compartidos.

https://docs.docker.com/compose/compose-application-model/

5. Gestión de múltiples contenedores



Ejemplo de uso

Tenemos una aplicación web desarrollada localmente (`C:\xampp\htdocs\test`) que queremos desplegar en un contenedor.

Vamos a emplear Docker Compose para instalar los 3 contenedores que necesitamos:

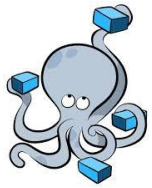
- PHP y Apache Server: PHP y servidor web
- MariaDB: base de datos
- phpMyAdmin: aplicación web para gestionar la base de datos

Crearemos un fichero `compose.yaml` (ver ejemplo para **Windows** en la siguiente diapositiva) en un directorio (en principio vale cualquiera).

Siguiendo en **Windows**: arrancar Docker Desktop
Para construir y levantar los servicios: `docker compose up -d`
Para borrar los servicios: `docker compose down`
Para acceder a los registros (*logs*) de los servicios: `docker compose logs`
Para ver todos los contenedores en servicio: `docker compose ps`

https://docs.docker.com/compose/reference/

5. Gestión de múltiples contenedores

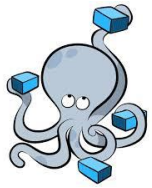


Ejemplo de `compose.yaml` (Windows)

NOTA: también se puede nombrar `compose.yml`, `docker-compose.yaml` o `docker-compose.yml`

```
services:
  web:
    image: php:8.2-apache
    container_name: contenedorApache
    ports:
      - "8080:80"
    volumes:
      - C:\xampp\htdocs\test:/var/www/html/test
    networks:
      - red1

  db:
    image: mariadb:latest
    container_name: contenedorMariaDB
    environment:
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
      MYSQL_DATABASE: miBaseDeDatos
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrasenya1
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - red1

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
    networks:
      - red1

volumes:
  db_data:
    name: db_data

networks:
  red1:
    name: red1
```

5. Gestión de múltiples contenedores



Ejemplo de `compose.yaml` (Windows)

NOTA: también se puede nombrar `compose.yml`, `docker-compose.yaml` o `docker-compose.yml`

```
services:
  web:
    image: php:8.2-apache
    container_name: contenedorApache
    ports:
      - "8080:80"
    volumes:
      - C:\xampp\htdocs\test:/var/www/html/test
    networks:
      - red1

  db:
    image: mariadb:latest
    container_name: contenedorMariaDB
    environment:
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
      MYSQL_DATABASE: miBaseDeDatos
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrasenya1
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - red1

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
    networks:
      - red1

volumes:
  db_data:
    name: db_data

networks:
  red1:
    name: red1
```

Define los servicios, donde cada servicio es un contenedor

Nombre del servicio para el servidor web

Imagen de la versión 8.2 de PHP/Apache

Nombre asignado

El pto 80 del contenedor se mapea al 8080 del host

Montaje de directorio (*bind mount*) para el código de la app

Nombre asignado a la red

Nombre del servicio para la base de datos

Última versión de MariaDB

Nombre asignado

Variables de entorno para MariaDB

BDD que se creará

Define un usuario (no usar 'root')

Y su contraseña

Monta un volumen para persistencia de datos

El volumen db_data se monta en /var/lib/mysql dentro del contenedor, que es donde MariaDB almacena sus datos

Nombre del servicio para phpMyAdmin

Nombre del servicio para la BDD

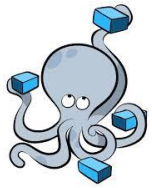
El puerto 80 de este contenedor se mapea al 8081 del host

Dependencia: phpMyAdmin solo se puede iniciar si el servicio db (base de datos) se ha iniciado previamente

Define los volúmenes que se usarán en los servicios (en este caso solo db_data)

Define las redes personalizadas por las que se comunicarán los servicios (en este caso solo red1)

5. Gestión de múltiples contenedores



Ejemplo de `compose.yaml` (Ubuntu)

NOTA: también se puede nombrar `compose.yml`, `docker-compose.yaml` o `docker-compose.yml`

```
services:
  web:
    image: php:8.2-apache
    container_name: contenedorApache
    ports:
      - "8080:80"
    volumes:
      - /var/www/html/test:/var/www/html/test
    networks:
      - red1

  db:
    image: mariadb:latest
    container_name: contenedorMariaDB
    environment:
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
      MYSQL_DATABASE: miBaseDeDatos
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrasenya1
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - red1

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
    networks:
      - red1

volumes:
  db_data:
    name: db_data
networks:
  red1:
    name: red1
```

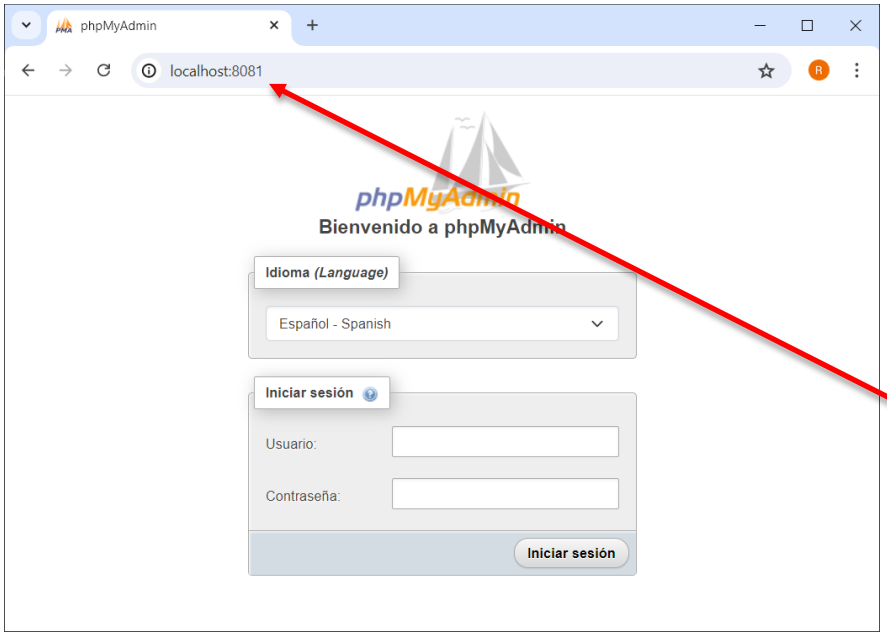
Antes de ejecutarlo, en un terminal cambiar los permisos para el lanzador `docker.sock`:

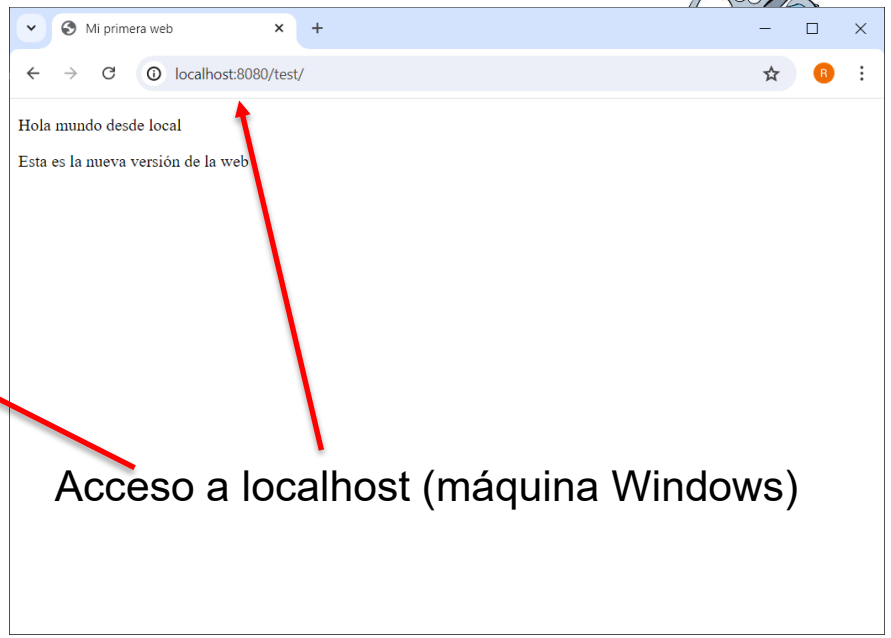
```
sudo chmod 666 /var/run/docker.sock
```

A continuación, levantar servicios:

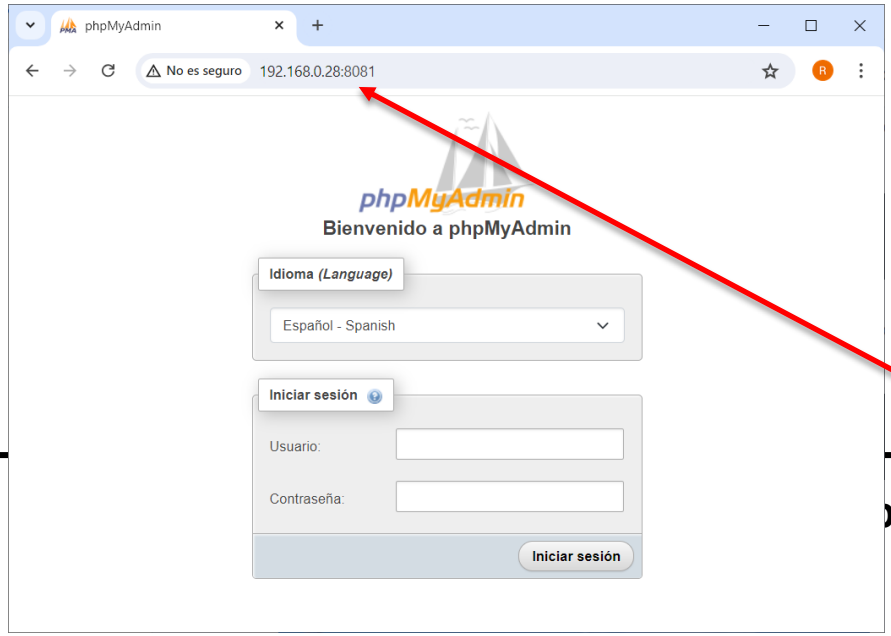
```
docker compose up -d
```

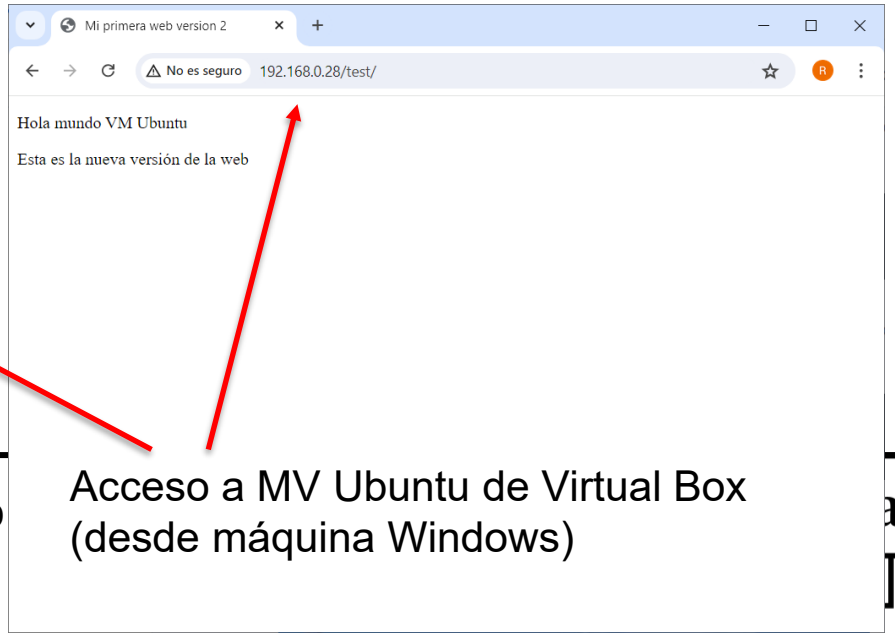
5. Gestión de múltiples contenedores





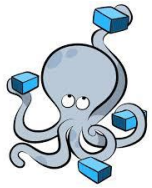
Acceso a localhost (máquina Windows)





Acceso a MV Ubuntu de Virtual Box
(desde máquina Windows)

5. Gestión de múltiples contenedores



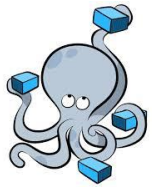
Conexión desde la aplicación web a la base de datos

Para conectarse a la base de datos, se puede configurar un fichero `config.php`:

```
<?php
    $servername = "db"; //nombre del servicio de MariaDB en Docker Compose
    $username = "usuario1"; //usuario que hemos creado (mejor no usar root)
    $password = "contrasenya1";  //contraseña del usuario1
    $dbname = "miBaseDeDatos";  //nombre de la base de datos

    // Crear conexión
    $conn = mysqli_connect($servername, $username, $password, $dbname);

    // Verificar conexión
    if ($conn->connect_error) {
        die("Error en la conexión: " . $conn->connect_error);
    }
    echo "Conexión correcta";
?>
```

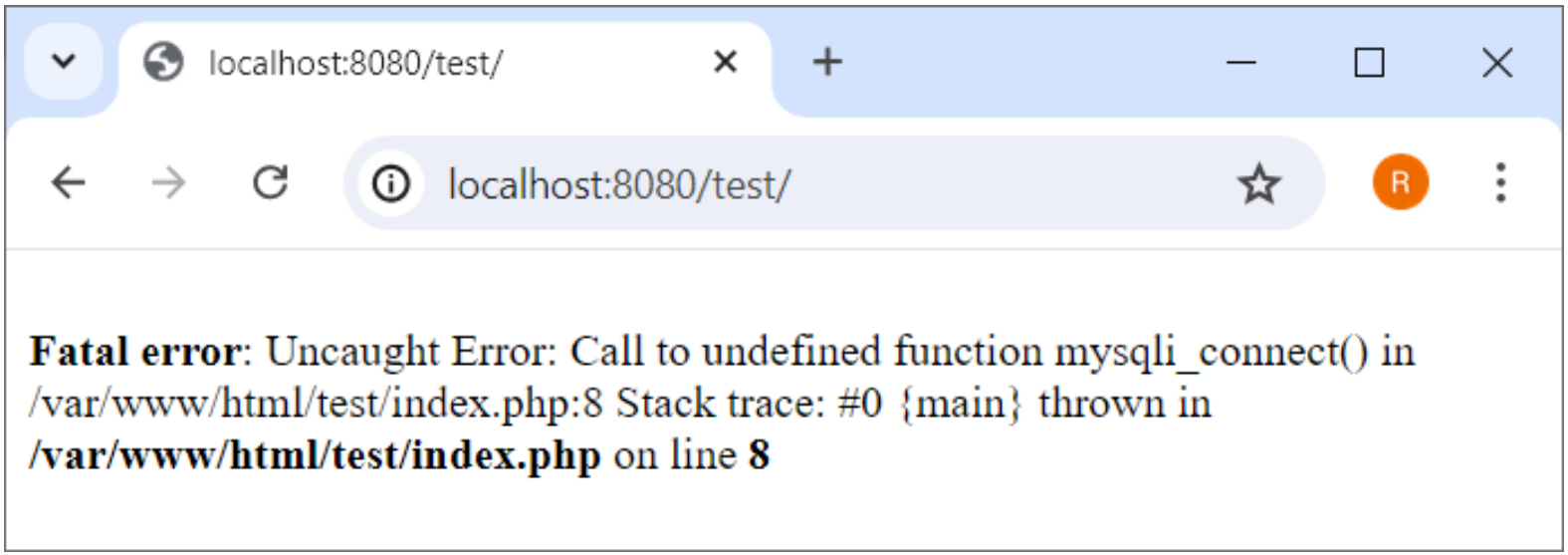
➔ Cada vez que se requiera utilizar la base de datos: `include 'config.php';`

5. Gestión de múltiples contenedores

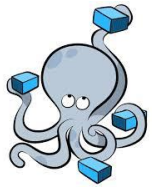


Conexión desde la aplicación web a la base de datos

Ejecución del script PHP de conexión a la base de datos:

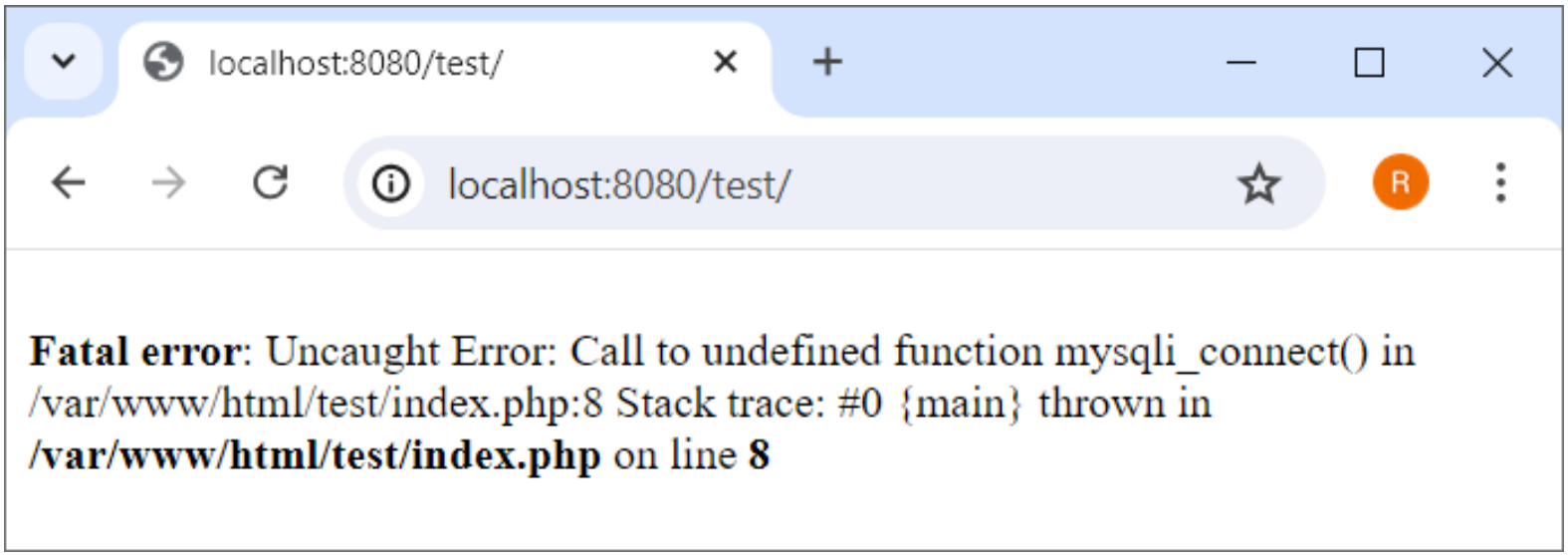


5. Gestión de múltiples contenedores



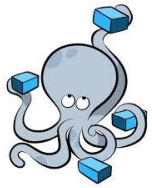
Conexión desde la aplicación web a la base de datos

Ejecución del script PHP de conexión a la base de datos:



Explicación: algunas imágenes de PHP no incluyen (curiosamente) la clase necesaria para conectarse a una base de datos MySQL, por lo que hay que crearse una imagen personalizada que la incorpore. En este caso sucede para el Docker que estamos utilizando en Windows (para la instalación en Ubuntu no pasa).

5. Gestión de múltiples contenedores



Conexión desde la aplicación web a la base de datos

Para crear nuestra imagen personalizada de PHP, creamos un fichero `Dockerfile` (sin extensión) en el mismo directorio donde tenemos el `compose.yaml`:

```
FROM php:8.2-apache
# Instala y habilita la extensión mysqli
RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
```

Y ahora modificamos la parte del fichero `compose.yaml` que define el servicio web:

Sustituir la línea:
`image: php:8.2-apache`

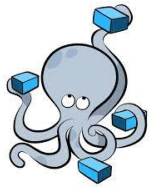
Por estas 3 líneas, que definen el uso de un `Dockerfile` personalizado:

```
build:
      context: .
      dockerfile: Dockerfile
```

`context: .` ← Indica el directorio donde está el fichero `Dockerfile`

`dockerfile: Dockerfile` ← Indica el nombre del fichero

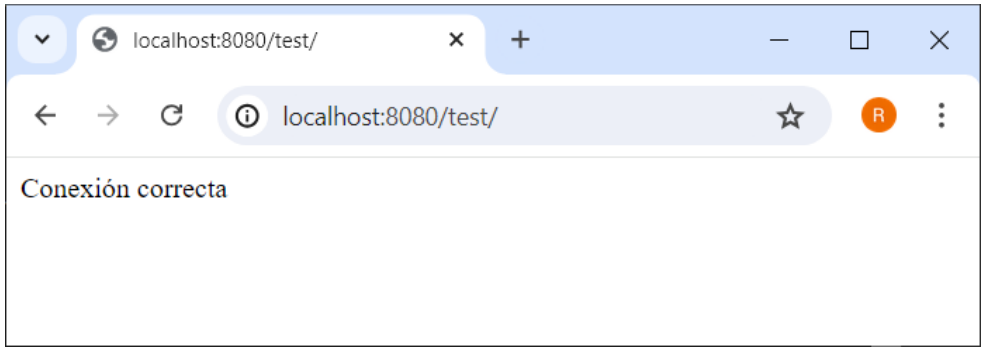
5. Gestión de múltiples contenedores

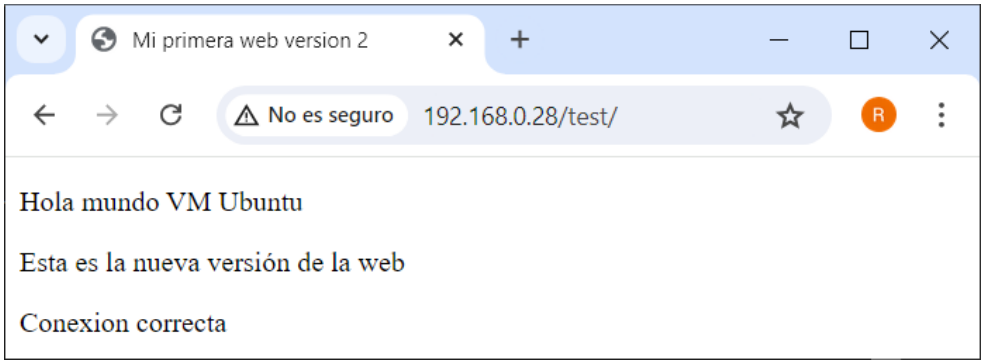


Conexión desde la aplicación web a la base de datos

Ahora solo hay que relanzar `docker compose` para aplicar los cambios:

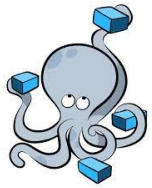
```
docker compose down
docker compose up -d
```





← En la versión Ubuntu no ha sido necesaria esta modificación

5. Gestión de múltiples contenedores



Nota sobre las rutas del servidor Apache en Linux

El servidor Apache puede utilizar por defecto dos rutas distintas para alojar las aplicaciones web: `/var/www/html` o `/usr/local/apache2/htdocs`.

En distribuciones Ubuntu o Debian la ruta es habitualmente `/var/www/html`.

En la imagen php:8.2-apache la ruta es habitualmente `/usr/local/apache2/htdocs`, aunque algunas imágenes pueden estar basadas en distribuciones Ubuntu o Debian y entonces utilizar como ruta `/var/www/html`.

En los ejemplos que hemos visto se ha utilizado la ruta `/var/www/html`.

5. Gestión de múltiples contenedores

Docker Swarm y Kubernetes

La gestión de múltiples contenedores también recibe el nombre de **orquestación**.

Docker Compose es una herramienta de orquestación limitada a un solo host, lo que hace que esté indicada para desarrollo, pruebas y producción a pequeña escala.

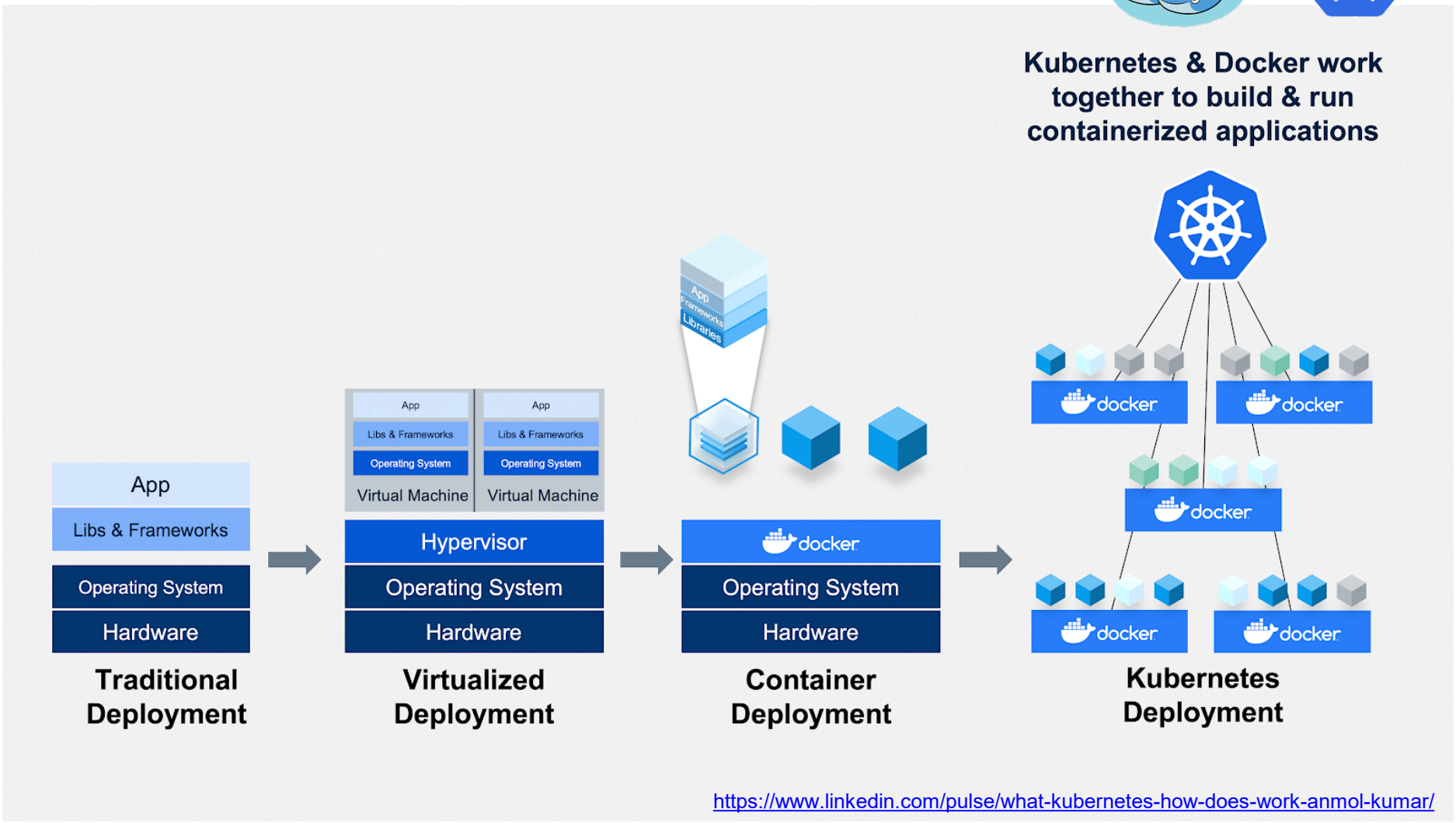
Por otro lado, **Docker Swarm** (Docker) y **Kubernetes** (Google) permiten orquestar muchos contenedores en clústeres de múltiples nodos, asegurando alta disponibilidad en entornos más complejos y de mayor escala.

Kubernetes es más complejo, pero ofrece más funciones y flexibilidad. Es la elección adecuada para despliegues en producción a gran escala.

https://kubernetes.io/

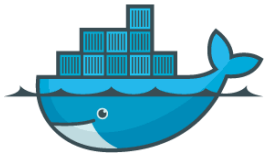
https://www.docker.com/resources/kubernetes-and-docker/

5. Gestión de múltiples contenedores



https://www.linkedin.com/pulse/what-kubernetes-how-does-work-anmol-kumar/

6. Imágenes personalizadas



En el tema 1 ya creamos una imagen personalizada y en este tema hemos visto que ha sido necesario crear también una imagen personalizada para incluir la clase `mysqli` en la imagen de PHP. En general, una imagen personalizada permite definir (y desplegar) un entorno específico para nuestra aplicación. Las instrucciones para su creación deben estar en un fichero **`Dockerfile`**.

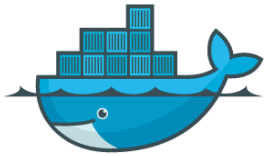
`FROM`: indica la imagen base a partir de la cual se construirá la nueva imagen.
`RUN`: ejecuta comandos en la imagen durante la construcción.
`COPY` o `ADD`: copia archivos o directorios desde el host a la imagen.
`WORKDIR`: establece el directorio de trabajo dentro de la imagen.
`CMD`: define el comando que se ejecutará cuando se inicie un contenedor de la imagen.
`EXPOSE`: declara el puerto en el que la aplicación escucha.
`ENV`: define variables de entorno dentro del contenedor.

Y para crear la imagen:

```
docker build -t nombreImagen:versión rutaDockerfile
```

`nombreImagen:versión` ➔ nombre y versión que queramos darle a nuestra imagen
`rutaDockerfile` ➔ ruta donde se encuentra el fichero `Dockerfile` que hemos creado (p.ej. “.” si estamos ejecutando la instrucción en el mismo directorio del fichero).

6. Imágenes personalizadas



Ejemplo: recordemos la imagen “Hola mundo” creada en el tema 1.

Estructura del proyecto:

```
/test_docker
|-- Dockerfile
|-- /public_html
    |-- index.html
```

```
FROM httpd:2.4
COPY ./public_html/ /usr/local/apache2/htdocs/
```

```
<!DOCTYPE html>
<html>
      <head>
            <title>Mi primera web</title>
      </head>
      <body>
            <p>Hola mundo</p>
      </body>
</html>
```

1. Abrir CMD/Terminal y situarse en el directorio `test_docker`.
2. Creación de la imagen: `docker build -t mi_imagen:v1 .`
3. Mostrar imágenes disponibles: `docker images`
4. Ejecución de la imagen: `docker run -d -p 8080:80 mi_imagen:v1`
5. Mostrar contenedores en ejecución: `docker ps`

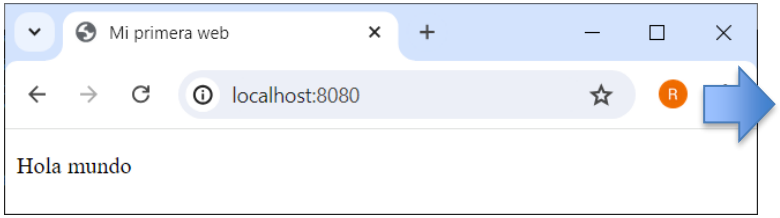
6. Imágenes personalizadas

Abrir CMD/Terminal y situarse en el directorio `test_docker`.
Creación de la imagen: `docker build -t mi_imagen:v1 .`
Mostrar imágenes disponibles: `docker images`

```
REPOSITORY              TAG          IMAGE ID       CREATED          SIZE
mi_imagen               v1           1e049ce31c99   57 seconds ago   148MB
despliegue-web          latest       367f5a5b4462   12 hours ago     503MB
php                     8.2-apache   5ccc0c72225d   7 days ago       503MB
mariadb                 latest       4486d64c9c3b   4 weeks ago      406MB
phpmyadmin/phpmyadmin   latest       933569f3a9f6   11 months ago    562MB
```

Ejecución de la imagen: `docker run -d -p 8080:80 mi_imagen:v1`
Mostrar contenedores en ejecución: `docker ps`

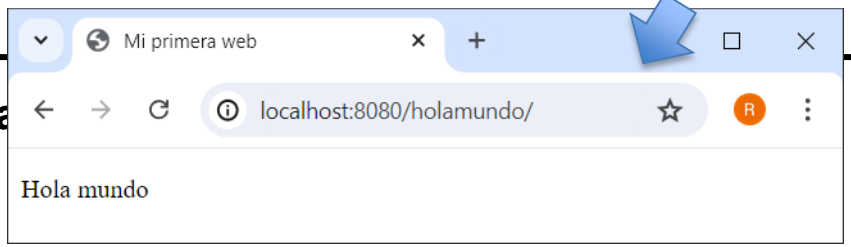
```
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
89f36eefa5f7   mi_imagen:v1   "httpd-foreground"       16 seconds ago   Up 15 seconds   0.0.0.0:8080->80/tcp     silly_banach
```



Vemos que la ruta es `localhost:8080`, pero si queremos añadir una subruta (p.ej. `localhost:8080/holamundo`), basta modificar el `Dockerfile`, añadiendo la subruta `holamundo` al final de la instrucción COPY

```
FROM httpd:2.4
COPY ./public_html/ /usr/local/apache2/htdocs/holamundo
```

Si ahora repetimos los pasos para crear la imagen y lanzarla, la ruta de navegación será la que queremos

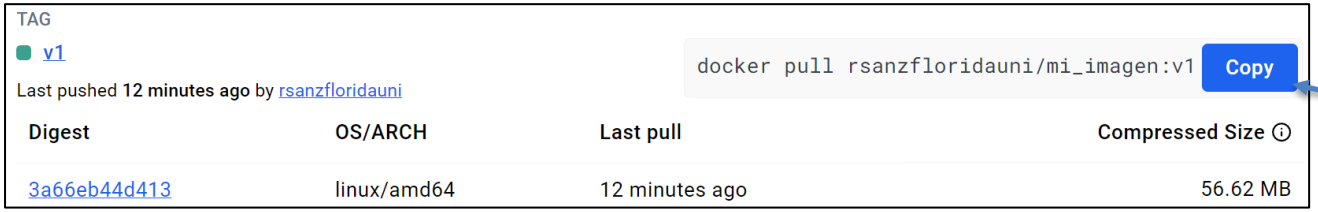


6. Imágenes personalizadas

Distribución de una imagen personalizada

Hemos creado una imagen (p.ej. `mi_imagen:v1`) que incluye nuestra aplicación y todas sus dependencias. Vamos a subirla a Docker Hub para que esté disponible para otros.

1. Crea una cuenta en Docker Hub si no la tienes aún.
2. Desde CMD/Terminal, inicia sesión en Docker Hub: `docker login`
3. Etiquetar la imagen: `docker tag mi_imagen:v1 tuUsuarioDockerHub/mi_imagen:v1`
4. Subir la imagen a Docker Hub: `docker push tuUsuarioDockerHub/mi_imagen:v1`
5. Verifica que la imagen aparece en tu cuenta de Docker Hub.



Instrucción para descargar la imagen (el usuario es en este caso `rsanzfloridauni`)

Desde cualquier otro equipo, una vez descargada la imagen, solo habría que arrancarla para tener la aplicación funcionando como en local:

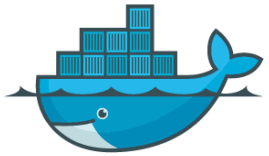
```
docker run -d -p 8080:80 rsanzfloridauni/mi_imagen.v1
```

NOTAS:

Por defecto, el repositorio se sube como público.

Se puede publicar una nueva versión de la imagen con otro tag (p.ej. `tuUsuarioDockerHub/mi_imagen:v2`).

6. Imágenes personalizadas



Ejemplo web cine (I): introducción

En este ejemplo vamos a crear dos imágenes personalizadas, con una base de datos MySQL y otra con nuestra aplicación PHP . A continuación, crearemos un fichero `docker-compose.yaml` que levantará las dos imágenes personalizadas más una imagen de phpMyAdmin.

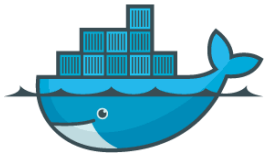
- **Imagen MySQL**: incluirá MySQL y las instrucciones necesarias para crear un volumen que incluya nuestra base de datos (ya rellenada).

- **Imagen PHP**: incluirá PHP 8.2, servidor Apache y una sencilla aplicación PHP propia que realizará una conexión a la base de datos y mostrará su contenido en una tabla HTML.

- **Imagen phpMyAdmin**: utilizaremos la imagen estándar, sin personalizar.

Recuerda que al tratarse de imágenes personalizadas tendremos que crear dos ficheros `Dockerfile`, uno para la imagen PHP y otro para la de MySQL.

6. Imágenes personalizadas



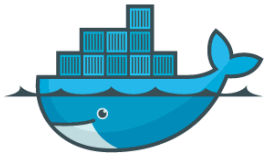
Ejemplo web cine (II): la base de datos

Se trata de una base de datos de películas llamada "`cine`", que tiene una sola tabla "`peliculas`", con los siguientes campos:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	**id** 🔑	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	**titulo**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		
☐	3	**director**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		
☐	4	**nota**	varchar(4)	utf8mb4_general_ci		No	*Ninguna*		
☐	5	**anyo**	int(4)			No	*Ninguna*		
☐	6	**presupuesto**	int(4)			No	*Ninguna*		
☐	7	**img_base64**	mediumtext	utf8mb4_general_ci		No	*Ninguna*		
☐	8	**url_trailer**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		

En el fichero `cine.sql` tienes todo lo necesario para crear la base de datos, la tabla y rellenarla con datos. Además, se incluye un usuario/contraseña con privilegios de acceso a la base de datos (`usuario1/contrasenyaUsuario1`).

6. Imágenes personalizadas



Ejemplo web cine (III): `Dockerfile` de la base de datos

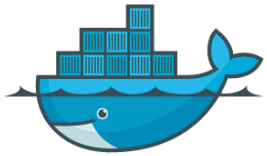
```
FROM mysql:latest
ENV MYSQL_ROOT_PASSWORD=contrasenyaRoot
ENV MYSQL_DATABASE=cine
ENV MYSQL_USER=usuario1
ENV MYSQL_PASSWORD=contrasenyaUsuario1

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

Tu usuario de Docker Hub

Construir la imagen: `docker build -t mi_cine:v1 .`
Etiquetar la imagen: `docker tag mi_cine:v1 rsanzfloridauni/mi_cine:v1`
Subir la imagen a Docker Hub: `docker push rsanzfloridauni/mi_cine:v1`

NOTA: recuerda que tienes que haber iniciado sesión en Docker Hub: `docker login`

6. Imágenes personalizadas

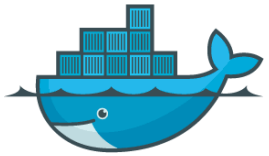


Ejemplo web cine (IV): fichero `index.php`

```
<?php
$servername = "db";
$username = "usuario1";
$password = "contrasenyaUsuario1";
$dbname = "cine";

// Crear la conexión
$conn = new mysqli($servername, $username, $password, $dbname);

// Verificar la conexión
if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

// Consulta SQL para seleccionar todo el contenido de la tabla peliculas
$sql = "SELECT * FROM peliculas";
$result = $conn->query($sql);

?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Listado de Películas</title>
</head>
```

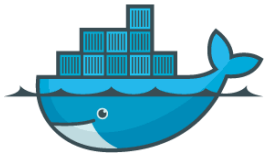
➔ CONTINUA

6. Imágenes personalizadas



```
<body>
    <h1>Listado de Películas</h1>
    <table>
        <thead>
            <tr>
<th>ID</th><th>Título</th><th>Director</th><th>Nota</th><th>Año</th><th>Ppto</th><th>Img</th><th>Trailer</th>
            </tr>
        </thead>
        <tbody>
            <?php
            if ($result->num_rows > 0) {
                    while($row = $result->fetch_assoc()) {
                    echo "<tr>";
                    echo "<td>" . htmlspecialchars($row["id"]) . "</td>";
                    echo "<td>" . htmlspecialchars($row["titulo"]) . "</td>";
                    echo "<td>" . htmlspecialchars($row["director"]) . "</td>";
                    echo "<td>" . htmlspecialchars($row["nota"]) . "</td>";
                    echo "<td>" . htmlspecialchars($row["anyo"]) . "</td>";
                    echo "<td>" . htmlspecialchars($row["presupuesto"]) . "</td>";
                    echo "<td><img src='" . htmlspecialchars($row["img_base64"]) . "'
alt='Imagen' width='100' height='100'></td>";
                    echo "<td><a href='" . htmlspecialchars($row["url_trailer"]) . "' target='_blank'>Ver
Trailer</a></td>";
                    echo "</tr>";
                }
            } else { echo "<tr><td colspan='8'>No hay registros</td></tr>";}
            ?>
        </tbody>
    </table>
</body>
</html>

<?php
$conn->close();
?>
```

6. Imágenes personalizadas



Ejemplo web cine (V): Dockerfile de PHP y servidor Apache

```
FROM php:8.2-apache
# Instala y habilita la extensión mysqli
RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
COPY . /var/www/html/mi_app_cine
```

Construir la imagen: `docker build -t mi_app_cine:v1 .`
Etiquetar la imagen: `docker tag mi_app_cine:v1 rsanzfloridauni/mi_app_cine:v1`
Subir la imagen a Docker Hub: `docker push rsanzfloridauni/mi_app_cine:v1`

NOTA: recuerda que tienes que haber iniciado sesión en Docker Hub: `docker login`

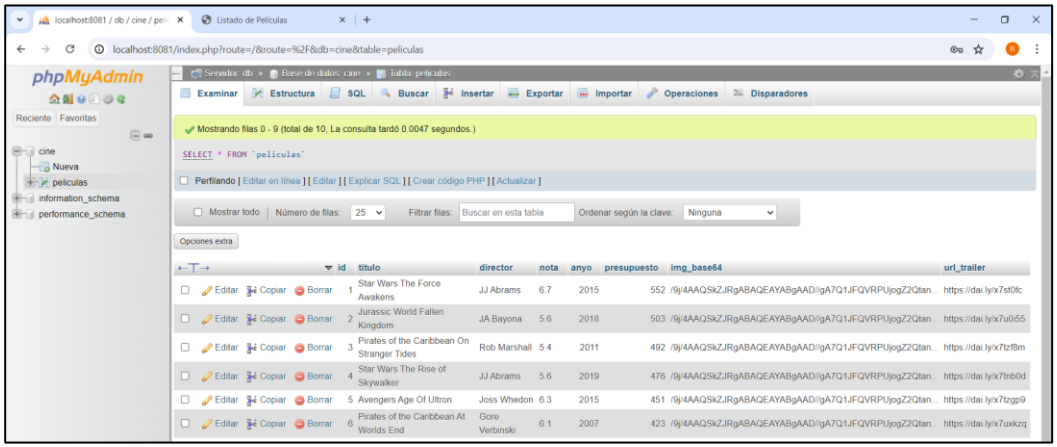
Tu usuario de Docker Hub

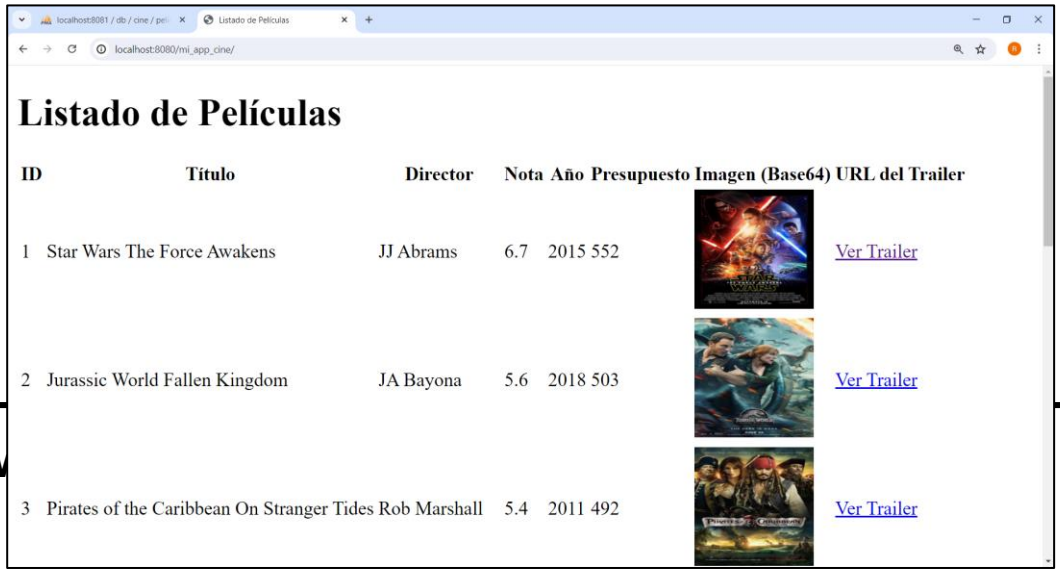
6. Imágenes personalizadas

Ejemplo web cine (VI): Docker Compose (compose.yaml)

```
services:
  db:
    image: rsanzfloridauni/mi_cine:v1
    container_name: contenedorMiCine
    environment:
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
      MYSQL_DATABASE: cine
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrasenyaUsuario1
    ports:
      - "3306:3306"

  php:
    image: rsanzfloridauni/mi_app_cine:v1
    container_name: contenedorMiApp
    ports:
      - "8080:80"
    depends_on:
      - db

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
```

```
docker compose up -d
```



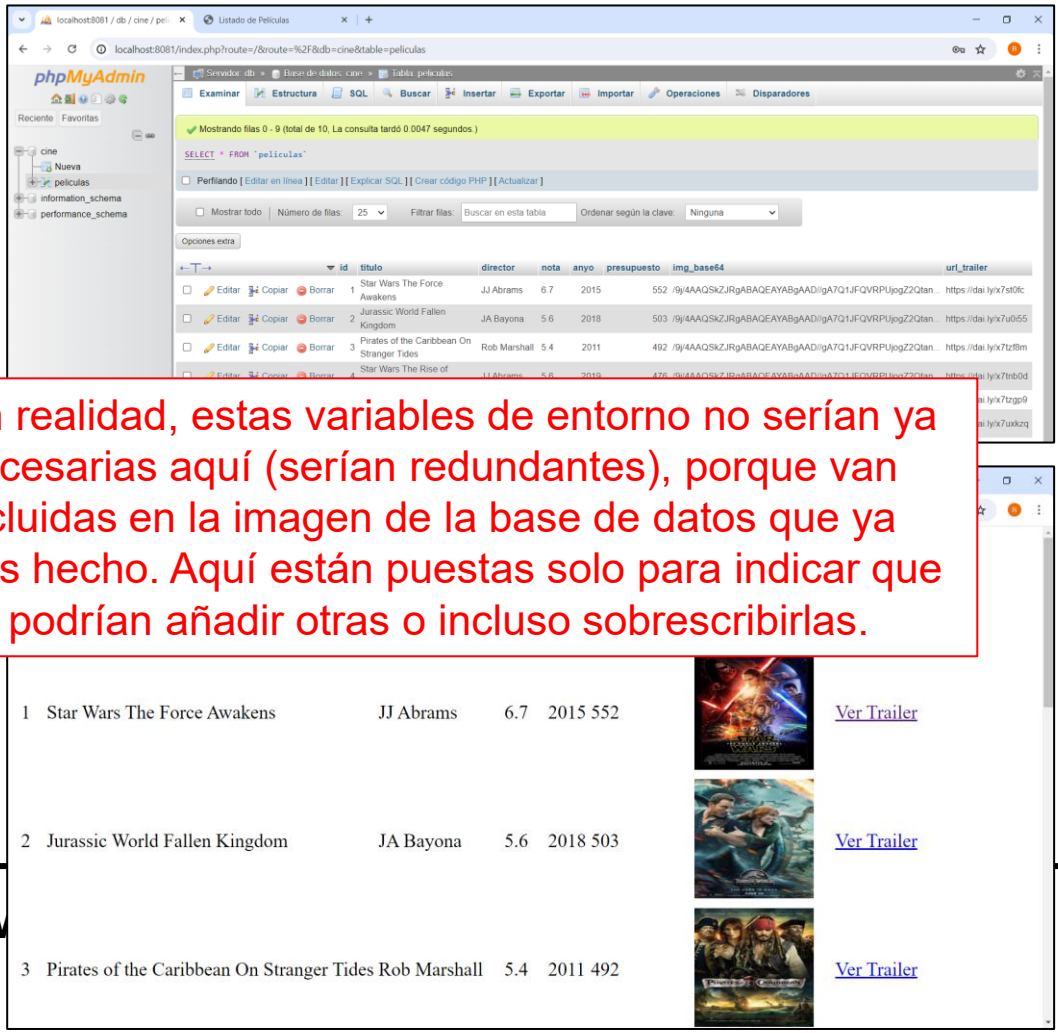


6. Imágenes personalizadas

Ejemplo web cine (VI): Docker Compose (compose.yaml)

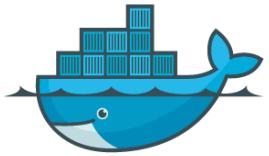
```
services:
  db:
    image: rsanzfloridauni/mi_cine:v1
    container_name: contenedorMiCine
    environment:
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
      MYSQL_DATABASE: cine
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrasenyaUsuario1
    ports:
      - "3306:3306"

  php:
    image: rsanzfloridauni/mi_app_cine:v1
    container_name: contenedorMiApp
    ports:
      - "8080:80"
    depends_on:
      - db

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrasenyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
```

```
docker compose up -d
```



En realidad, estas variables de entorno no serían ya necesarias aquí (serían redundantes), porque van incluidas en la imagen de la base de datos que ya has hecho. Aquí están puestas solo para indicar que se podrían añadir otras o incluso sobrescribirlas.

6. Imágenes personalizadas

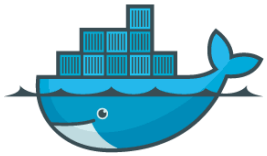


Estructura de directorios

```
/docker_cine
  |-- /mysql
  |     |-- cine.sql
  |     |-- Dockerfile
  |-- /php
  |     |-- Dockerfile
  |     |-- index.php
  |-- compose.yaml
```

MUY IMPORTANTE
Tener en cuenta que para ejecutar cada fichero (`Dockerfile` o `compose.yaml`) hay que estar en el directorio (o subdirectorio) correspondiente para que las rutas sean las correctas.

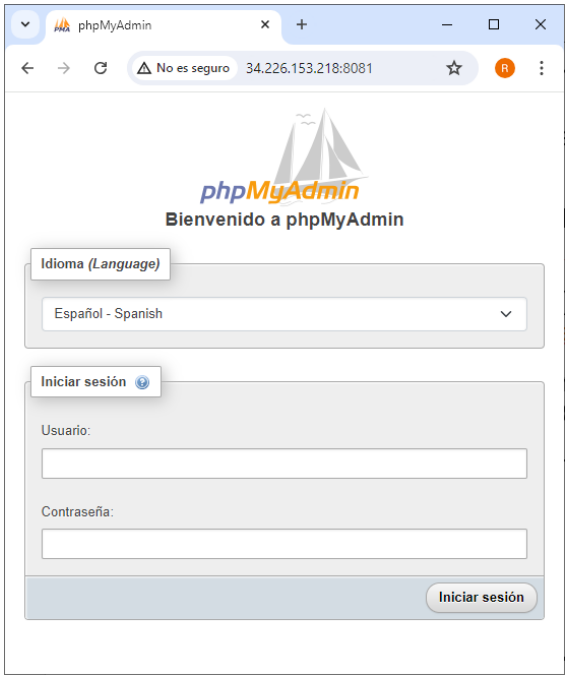
6. Imágenes personalizadas

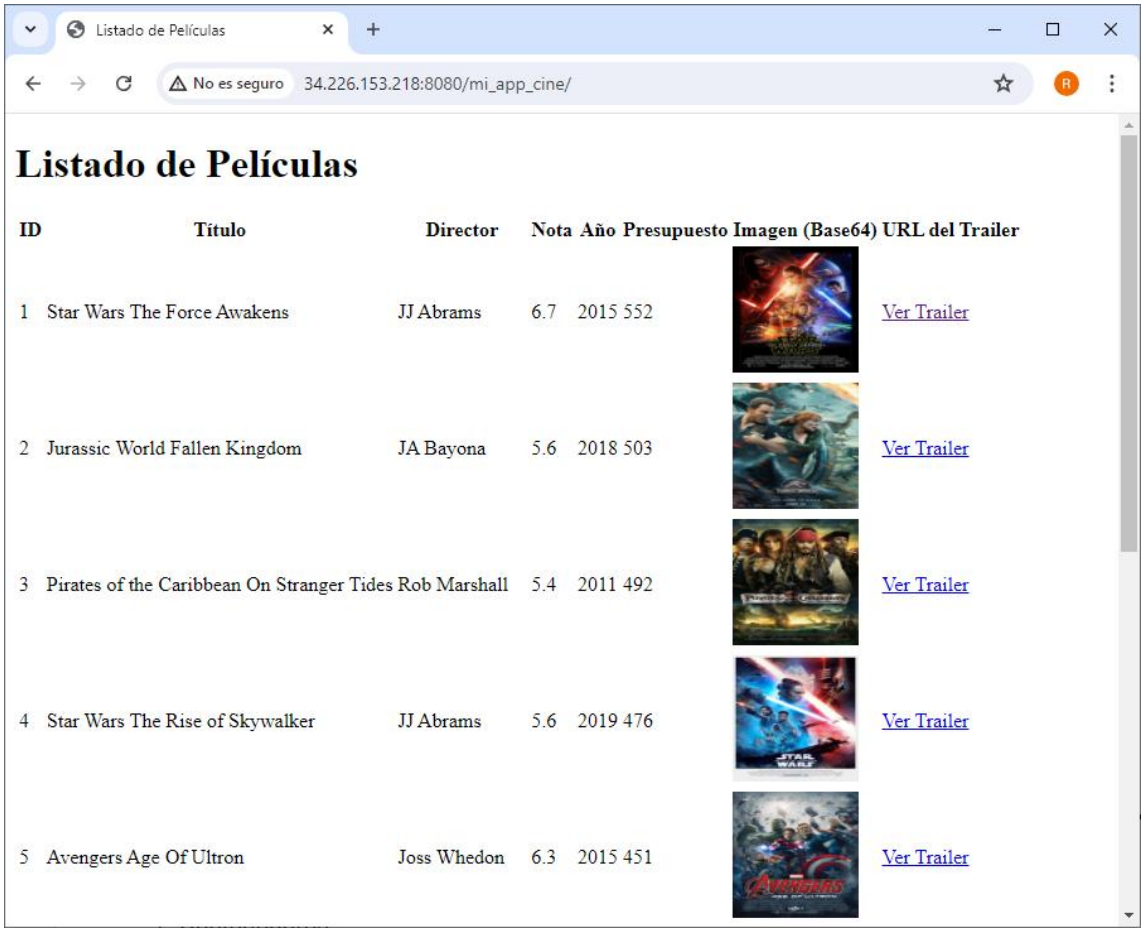


Prueba en otra máquina (MV Ubuntu, en Virtual Box o la instancia EC2 de AWS)

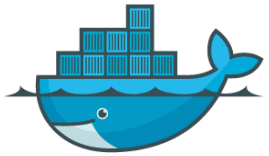
1. Copia el fichero `compose.yaml` a la máquina virtual con Ubuntu. Puedes hacerlo vía SFTP, como vimos en el tema 2.
2. Antes de ejecutarlo, en un terminal cambiar los permisos para el lanzador `docker.sock`: `sudo chmod 666 /var/run/docker.sock`
3. A continuación, levantar servicios (recuerda hacerlo desde el directorio donde tengas el fichero `.yaml`): `docker compose up -d`
4. Ejecuta un navegador en la máquina host (Windows) y accede a la URL correspondiente (`http://IP:8080/mi_app_cine`)
5. Prueba también introducir la URL `http://IP:8081` para acceder a phpMyAdmin.
6. Para detener los servicios: `docker compose down`

6. Imágenes personalizadas





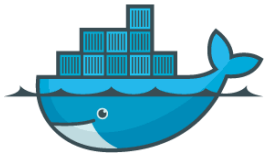
6. Imágenes personalizadas



¿Qué hace este `Dockerfile`?

```
FROM php:8.3-apache
WORKDIR /www/html
RUN apt-get update
RUN apt-get -y install nano
RUN apt-get -y install zip
RUN apt-get -y install unzip
RUN apt-get -y install git
RUN docker-php-ext-install mysqli pdo pdo_mysql
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php composer-setup.php
RUN php -r "unlink('composer-setup.php');"
RUN mv composer.phar /usr/local/bin/composer
RUN curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' | bash
RUN apt-get -y install symfony-cli
COPY ../ .
```

6. Imágenes personalizadas



¿Qué hace este `Dockerfile`?

```
FROM php:8.3-apache
WORKDIR /www/html
RUN apt-get update
RUN apt-get -y install nano
RUN apt-get -y install zip
RUN apt-get -y install unzip
RUN apt-get -y install git
RUN docker-php-ext-install mysqli pdo pdo_mysql
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php composer-setup.php
RUN php -r "unlink('composer-setup.php');"
RUN mv composer.phar /usr/local/bin/composer
RUN curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' | bash
RUN apt-get -y install symfony-cli
COPY ../ .
```

Parte de una imagen de PHP 8.3 con Apache

Establece `/www/html` como directorio de trabajo dentro del contenedor (alojará la aplicación)

Actualiza el índice de paquetes de Debian

Instala herramientas útiles para desarrollar y gestionar archivos

Instala extensiones `mysqli`, `pdo` y `pdo_mysql` para PHP

Descarga el script de instalación de Composer (gestor de dependencias de PHP)

Ejecuta el script (instala Composer)

Elimina el script para limpiar el entorno

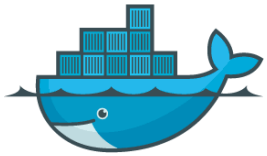
Instala Symfony CLI (interfaz de línea de comandos) para desarrollar aplicaciones

Descarga el script de instalación de Symfony

Mueve el archivo `composer.phar` a un directorio incluido en el PATH del sistema (`/usr/local/bin`), lo que hace que el comando `composer` esté disponible globalmente

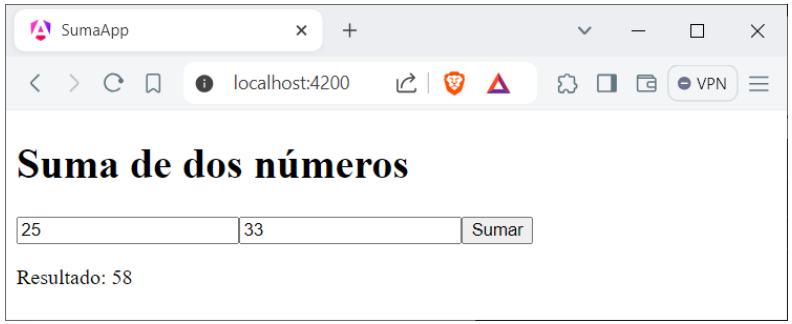
Copia la aplicación (ubicada en el directorio padre del Dockerfile en este ejemplo) a `/www/html`

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

Vamos a crear una sencilla aplicación en Angular que permita introducir dos números y devuelva la suma.



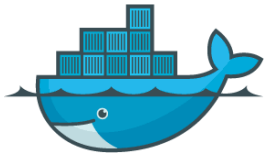
Si no tienes instalado Angular CLI, instálalo desde un terminal:

```
npm install -g @angular/cli
```

NOTA1
Debes tener instalado Node.js y el gestor de paquetes npm (tema 3). Puedes comprobarlo desde un terminal con `npm -v`
Windows: descarga e instala la última versión LTS de https://nodejs.org/en/download/prebuilt-installer
Linux: introduce estos comandos: `sudo apt-get install nodejs` `sudo apt install npm`

NOTA2
Es posible que el terminal de PowerShell de Windows te dé algún problema de ejecución de scripts. En ese caso, puedes utilizar CMD o GitBash.

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

Crea una aplicación nueva: `ng new suma-app`

Navega al directorio de la aplicación: `cd suma-app`

Genera un componente llamado `SumaComponent`: `ng generate component Suma`

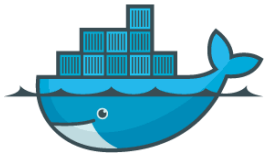
Implementar el componente (ver código en las siguientes diapositivas):

- `src/app/suma/suma.component.ts`
- `src/app/suma/suma.component.html`
- `src/app/app.component.ts`
- `src/app/app.component.html`

Ejecuta la aplicación: `ng serve --port 4200`

Accede a través del navegador: http://localhost:4200

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

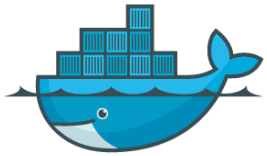
src/app/suma/suma.component.ts

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-suma',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './suma.component.html',
  styleUrl: './suma.component.css'
})
export class SumaComponent {
  numero1: number = 0;
  numero2: number = 0;
  resultado: number = 0;

  calcularSuma(): void {
    this.resultado = this.numero1 + this.numero2;
  }
}
```

src/app/suma/suma.component.html

```
<div>
    <h1>Suma de dos números</h1>
    <input type="number" [(ngModel)]="numero1" placeholder="Número 1">
    <input type="number" [(ngModel)]="numero2" placeholder="Número 2">
    <button (click)="calcularSuma()">Sumar</button>
    <p>Resultado: {{ resultado }}</p>
  </div>
```

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

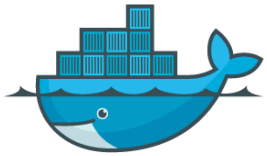
src/app/app.component.ts

```
import { Component } from '@angular/core';
import { SumaComponent } from './suma/suma.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [SumaComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'suma-app';
}
```

src/app/app.component.html

```
<app-suma></app-suma>
```

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

Construir la aplicación para producción: `ng build`

NOTA: se creará una carpeta llamada "**`dist`**" en el directorio del proyecto, con el código preparado para su despliegue en un servidor web.

Crear un fichero `Dockerfile` en el directorio raíz del proyecto:

```
FROM nginx:alpine
COPY ./dist/suma-app/browser /usr/share/nginx/html
```

Utilizamos la distribución de nginx de Linux Alpine (muy ligera)

NOTA: en alguna versión de Angular no hay que indicar la subcarpeta "`browser`", sería solo "`./dist/suma-app`"

Crear la imagen Docker: `docker build -t angular-suma-app:latest .`

Levantar contenedor: `docker run -d -p 8080:80 angular-suma-app`

Comprobar que funciona: http://localhost:8080

NOTA: recuerda que con **CTRL+F5** se actualiza la página completamente

Recuerda que puedes mapear otro puerto no usado (p.ej. 4200)

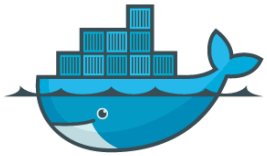
Sube la imagen a tu Docker Hub para poder descargarla en el servidor:

Autenticación: `docker login`

Etiquetar: `docker tag angular-suma-app tuUsuario/angular-suma-app:latest`

Subir: `docker push tuUsuario/angular-suma-app:latest`

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

Accede ahora al servidor (en este ejemplo accederemos a la máquina virtual Ubuntu), descarga de Docker Hub y ejecuta la imagen que acabas de crear:

```
docker pull tuUsuarioDockerHub/angular-suma-app:latest
docker run -d -p 8080:80 tuUsuarioDockerHub/angular-suma-app:latest
```

Con esto ya podrías desplegar tu aplicación de Angular en cualquier servidor que tenga instalado Docker.

7. Conclusiones

Aislamiento y portabilidad: Docker permite empaquetar aplicaciones con sus dependencias y versiones específicas, así como desplegarlas en cualquier sistema con Docker instalado. Se pueden utilizar imágenes de terceros o construir las propias (`Dockerfile`).

Eficiencia en el uso de recursos: consume menos recursos que una máquina virtual.

Simplificación del despliegue: Docker Compose permite definir y gestionar múltiples contenedores como un solo servicio y en un solo fichero YAML, facilitando así su despliegue.

Dependencias: Docker Compose permite definir el orden de inicio de los contenedores, para asegurar que las dependencias entre ellos estén bien gestionadas.

Comandos clave:

- **Docker**: `docker build`, `docker pull`, `docker run`, `docker ps`, `docker exec`, `docker stop`, `docker rm`, `docker rmi`
- **Docker Compose**: `docker compose up`, `docker compose down`