

DAW

Despliegue de aplicaciones web

4. Contenedores

Juan Jesús Tortajada Cordero

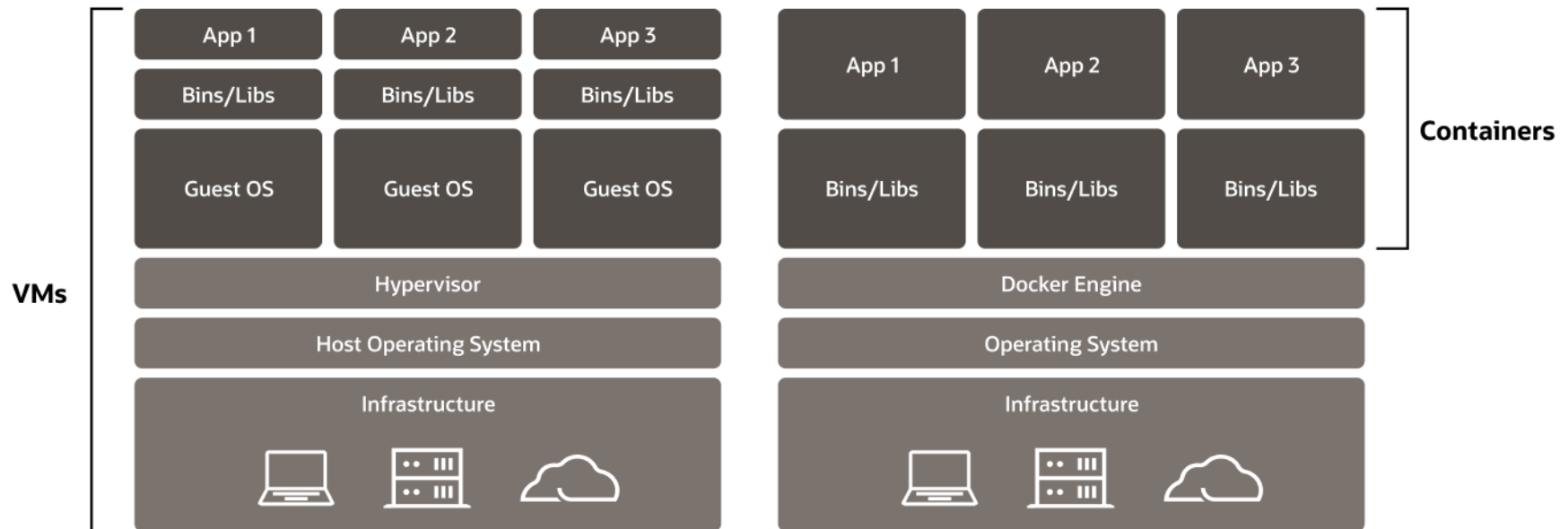
jtortajada@florida-uni.es

Índice

- 1. Introducción (Docker)**
- 2. Instalación y conceptos básicos**
- 3. Trabajo con imágenes y contenedores**
- 4. Persistencia y volúmenes**
- 5. Gestión de múltiples contenedores**
- 6. Imágenes personalizadas**
- 7. Conclusiones**

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.)

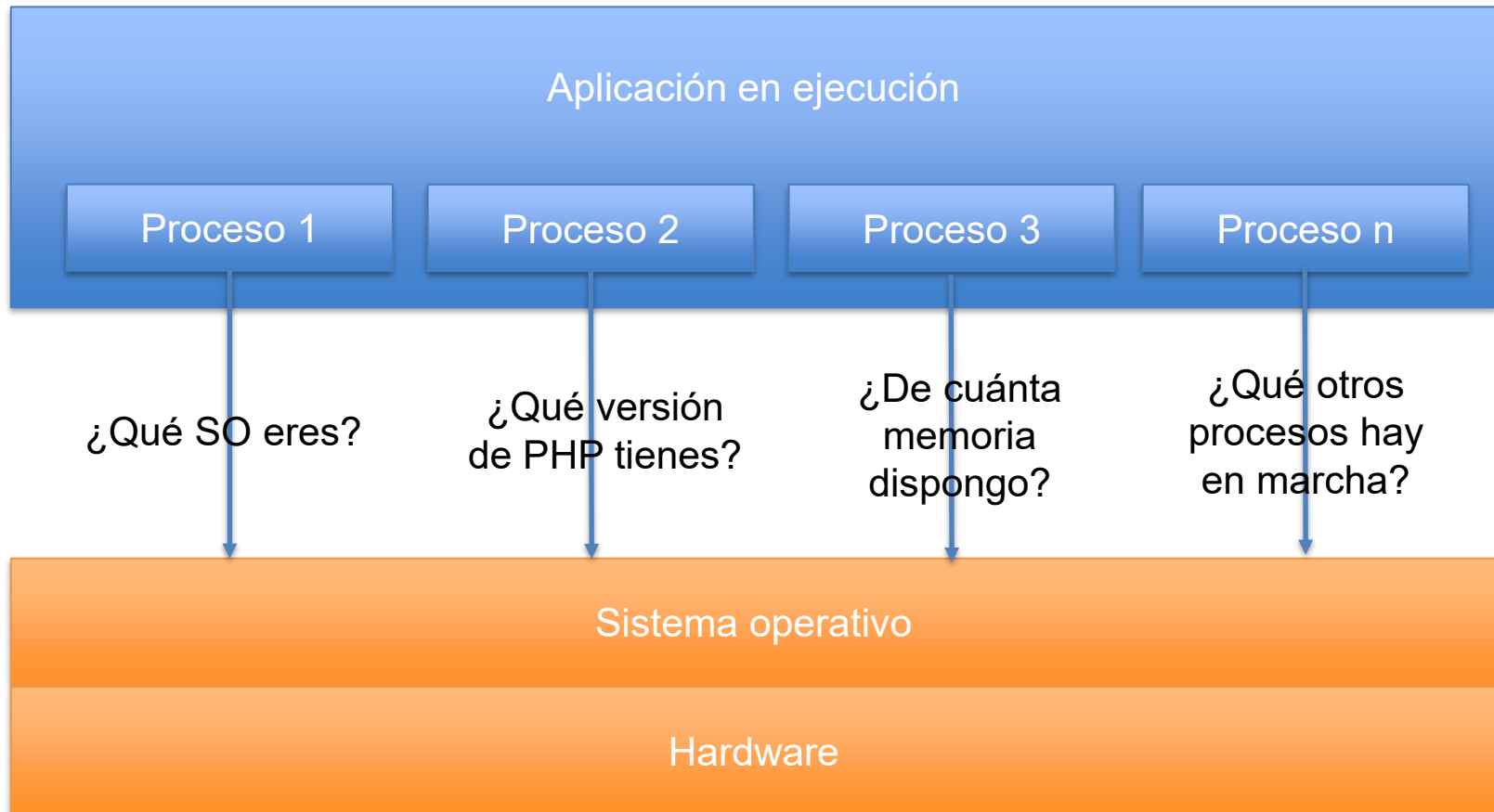


Idea de la “**containerización**”: utilizar solo un sistema operativo



1. Introducción (Docker)

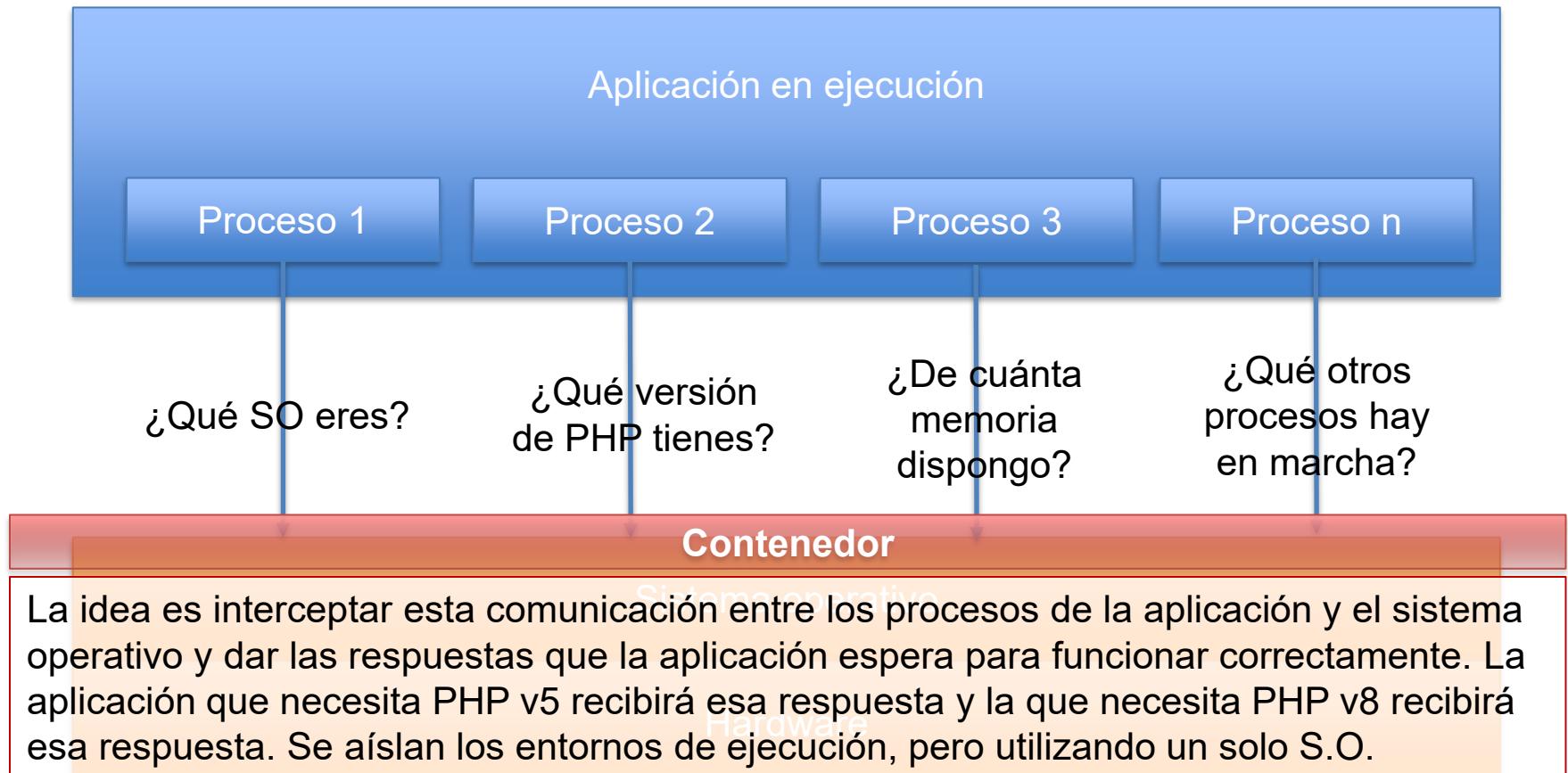
¿Cómo funciona Docker? En una aplicación normal (sin Docker)...





1. Introducción (Docker)

¿Cómo funciona Docker? La idea de Docker





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.

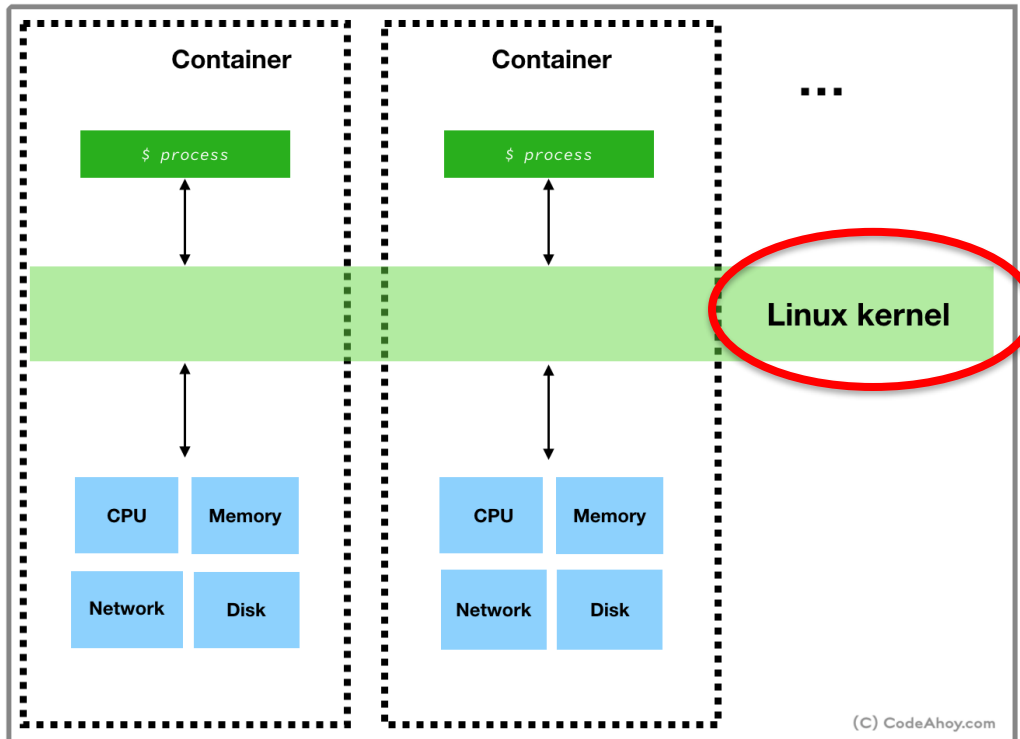
Para ello, los contenedores utilizan unas características del **kernel de Linux**. Estas son dos de las principales:

- **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](https://hub.docker.com/) 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`

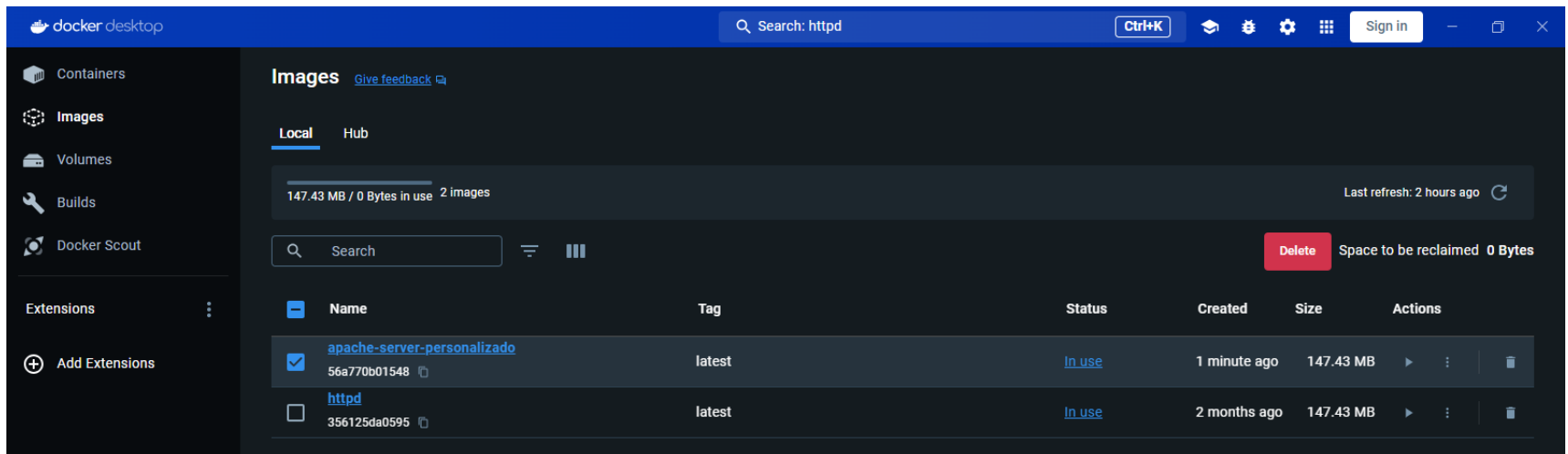
<https://docs.docker.com/reference/>



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
/ #
```

DAW - Despliegue de aplicaciones web

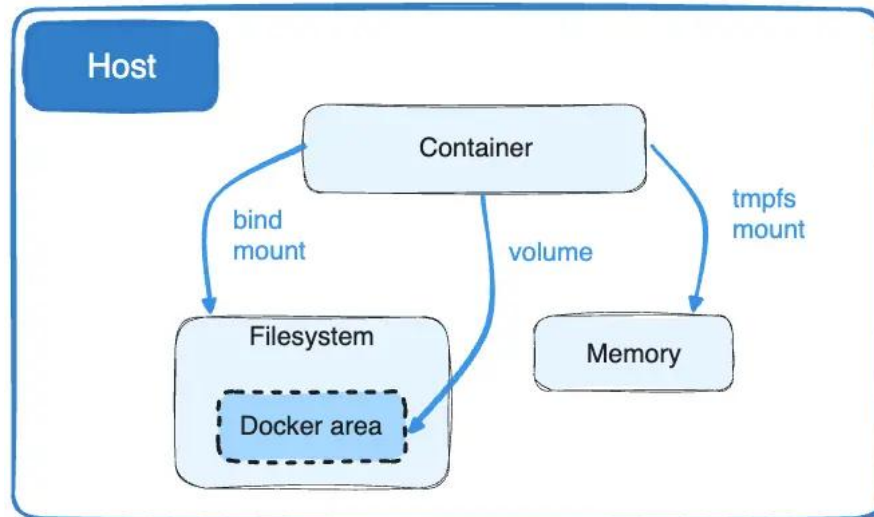
4. Contenedores



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	<i>Volume</i>	<i>Bind mount</i>
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	<i>Volume</i>	<i>Bind mount</i>
Gestión	Gestionado por Docker	Gestionado por el usuario
Ubicación	<div>¿Cuándo usar volúmenes o montaje de directorio? <u>Volúmenes</u>: cuando el contenedor necesita una base de datos. <u>Montaje de directorio</u>: 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.</div>	
Portabilidad		
Desempeño		
Seguridad		
Configuración		
Uso compartido (entre contenedores)	Sencillo	Puede presentar riesgos



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

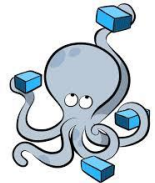
Imaginemos que el proyecto web local está alojado en `C:\xampp\htdocs\app`

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 ➔

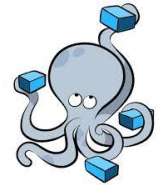
Herramienta que simplifica la definición y administración de aplicaciones multicontenedor. Incorporada en Docker Desktop (Win). En Linux: `sudo apt install docker-compose`

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`

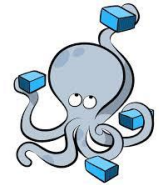
Para borrar los servicios: `docker compose down`

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

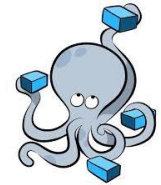


Ejemplo de compose.yaml (Windows)

NOTA: también se puede nombrar compose.yaml, 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.yaml, 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: contrase nyaRoot
      MYSQL_DATABASE: miBaseDeDatos
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: contrase nya1
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - red1
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: contenedorPhpMyAdmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: contrase nyaRoot
    ports:
      - "8081:80"
    depends_on:
      - db
    networks:
      - red1
  db_data:
    name: db_data
  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

contrase nyaRoot

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

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)

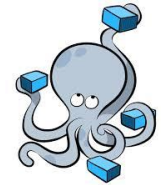
Nombre del servicio para phpMyAdmin

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

5. Gestión de múltiples contenedores



Ejemplo de compose.yaml (Ubuntu)

NOTA: también se puede nombrar compose.yaml, 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
```

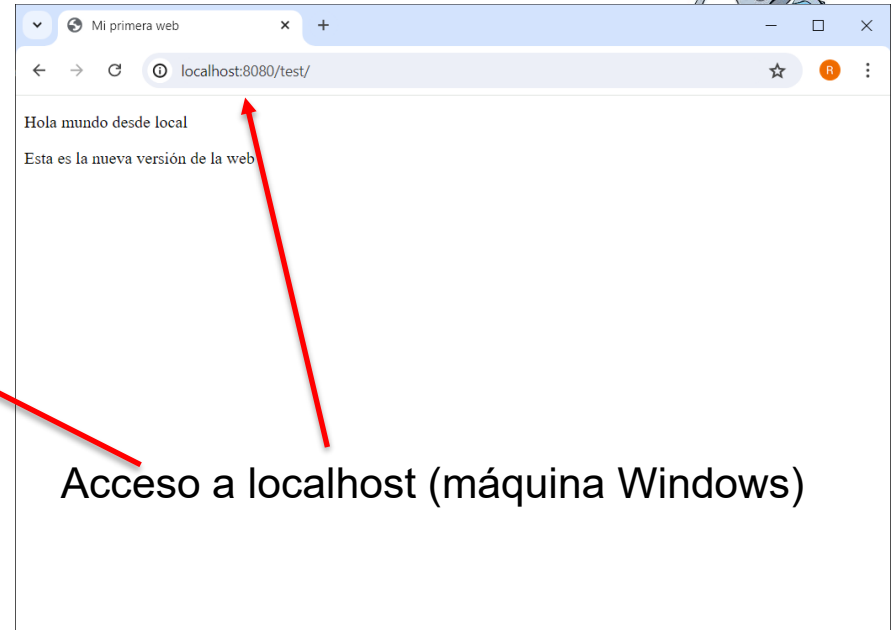
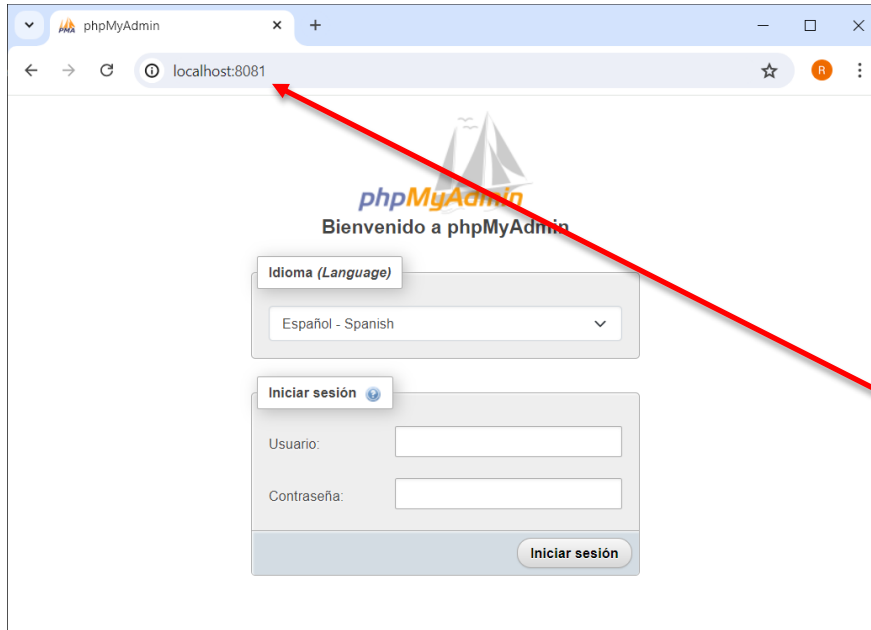
Antes de ejecutarlo, en un terminal cambiar los permisos para el lanzador docker.sock:

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

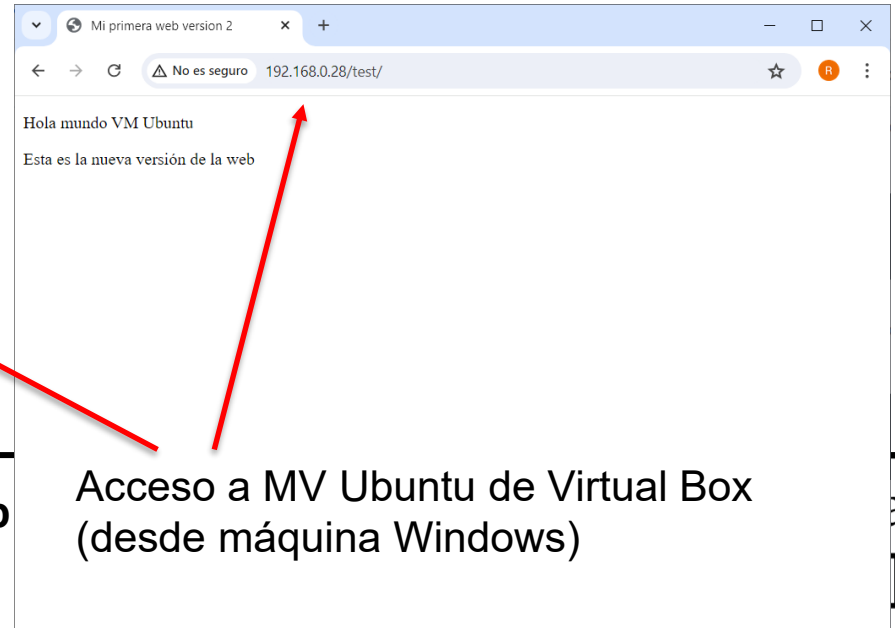
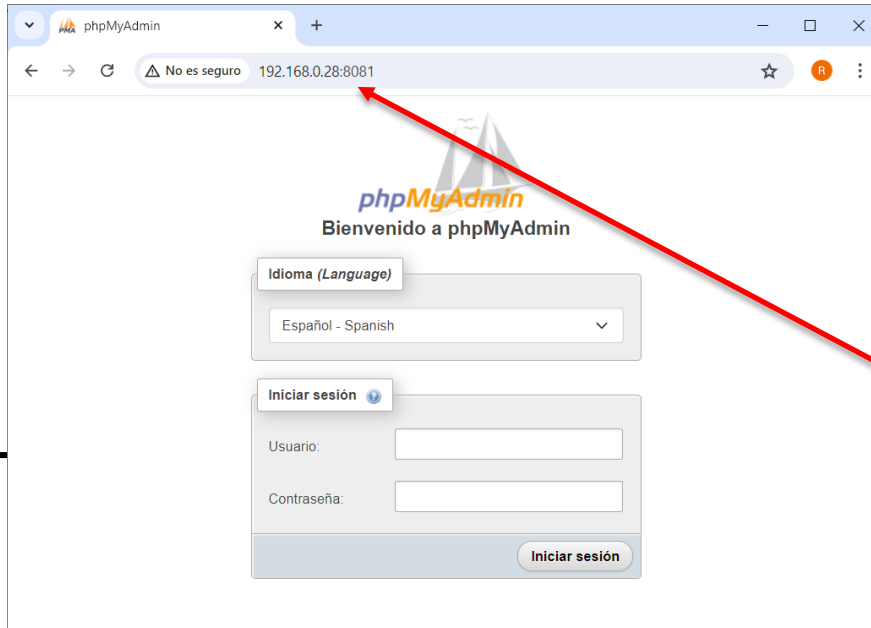
A continuación, levantar servicios:

```
docker compose up -d
```

5. Gestión de múltiples contenedores

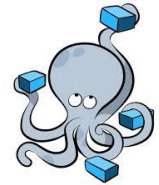


Acceso a localhost (máquina Windows)



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:

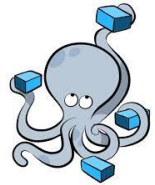
```
<?php
    $servername = "db"; //nombre del servicio de MariaDB en Docker Compose
    $username = "usuario1"; //usuario que hemos creado (mejor no usar root)
    $password = "contrasena1"; //contraseña del usuario1
    $dbname = "miBaseDeDatos"; //nombre de la base de datos

    // 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";
?>
```

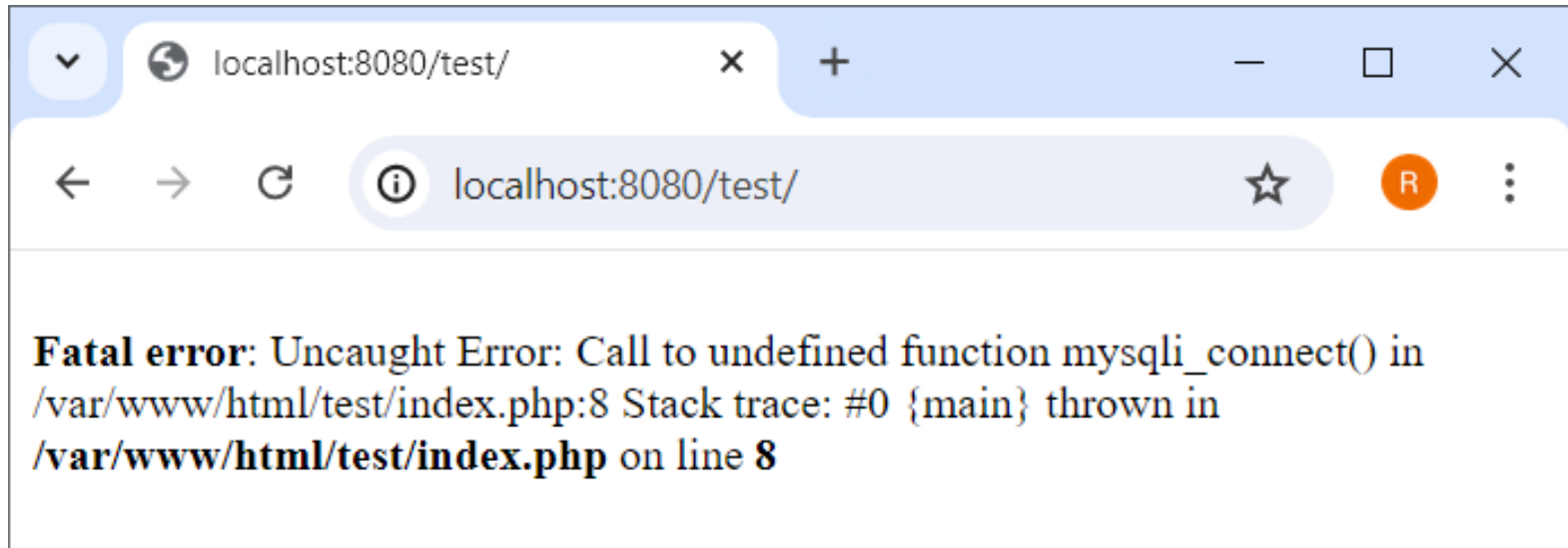
➔ Cada vez que se requiera utilizar la base de datos: include 'config.php';

5. Gestión de múltiples contenedores

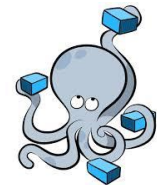


Conexión desde la aplicación web a la base de datos

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

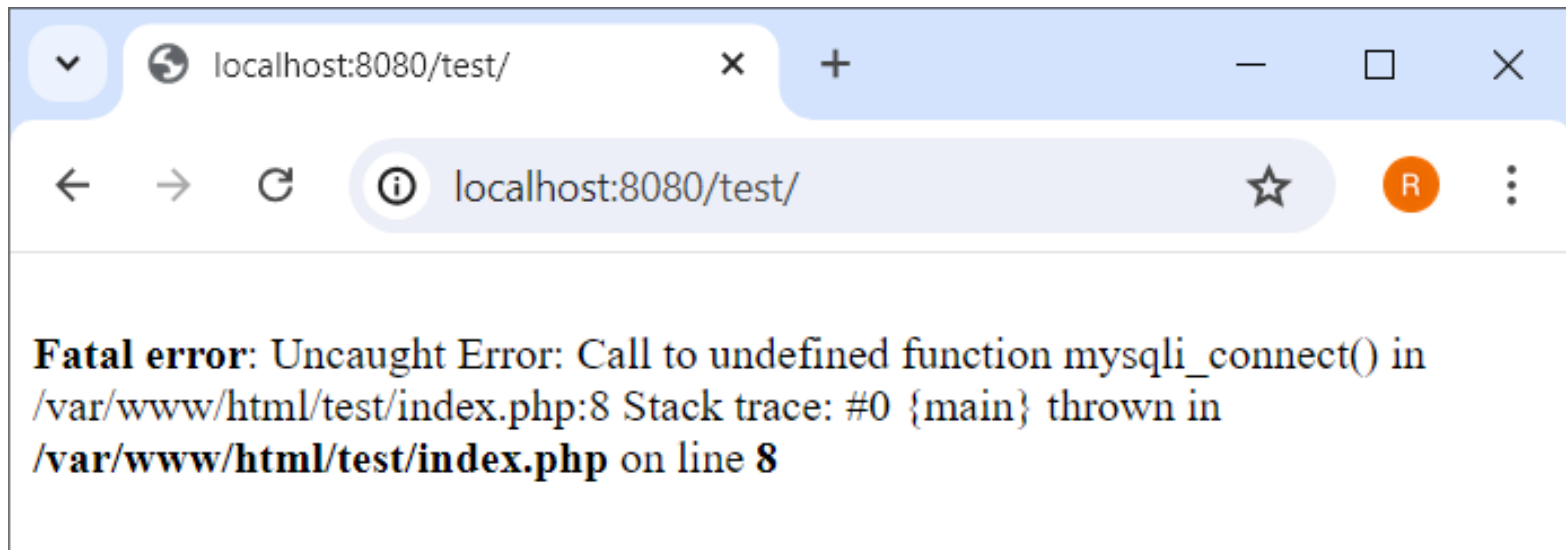


5. Gestión de múltiples contenedores



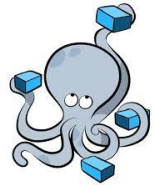
Conexión desde la aplicación web a la base de datos

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:

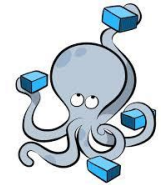
```
image: php:8.2-apache
```

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

```
build:
```

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


5. Gestión de múltiples contenedores

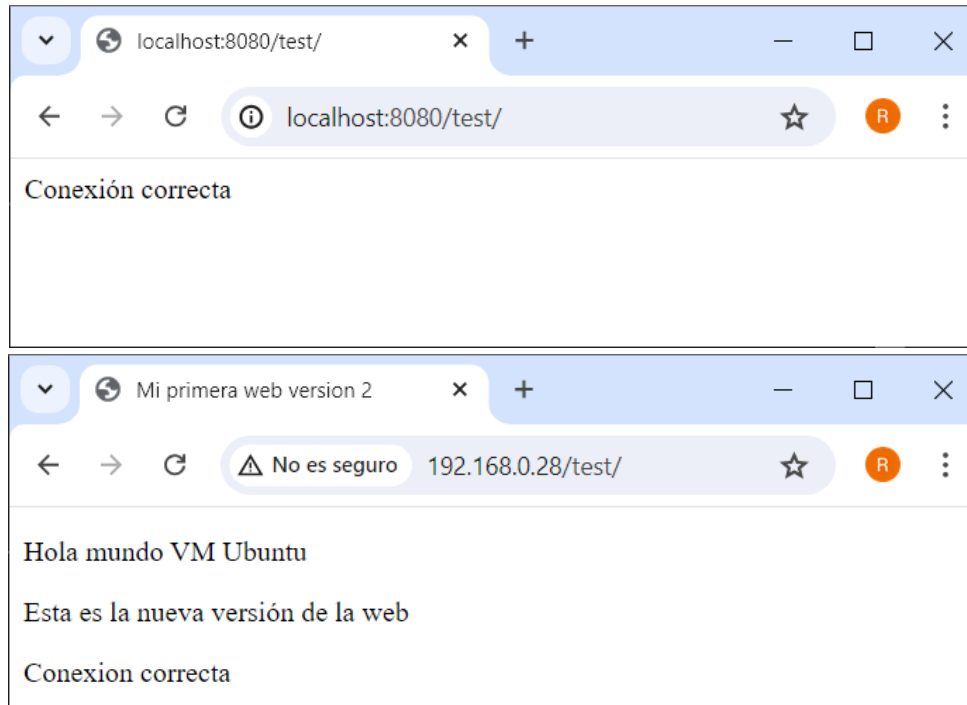


Conexión desde la aplicación web a la base de datos

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

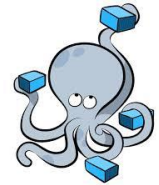
`docker compose down`

`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

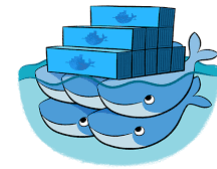
El servidor Apache puede utilizar por defecto dos rutas distintas para alojar las aplicaciones web: `/var/www/html` o `/usr/local/apache2/htdocs`.

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.

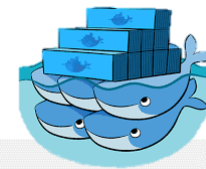
Por otro lado, **Docker Swarm** (Docker) y **Kubernetes** (Google) permiten orquestar muchos contenedores en clústeres de múltiples nodos, asegurando alta disponibilidad en entornos más complejos y de mayor escala.

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

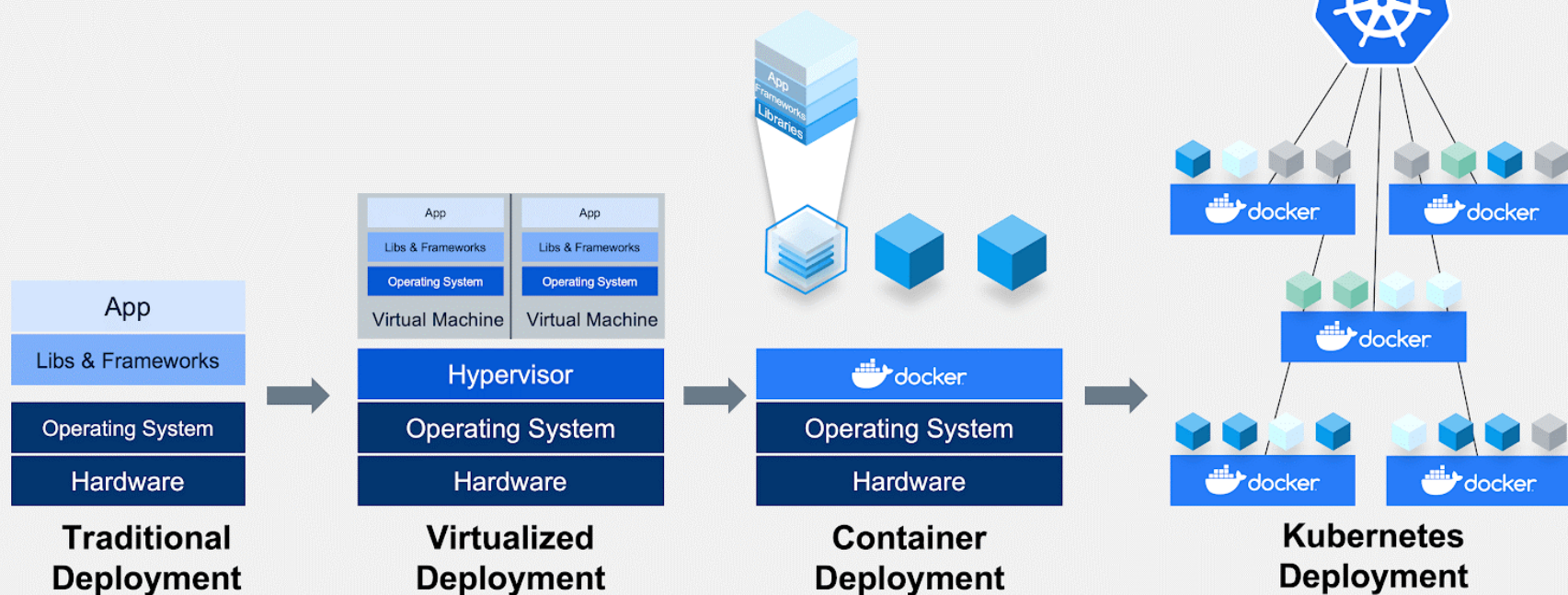
<https://kubernetes.io/>

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

5. Gestión de múltiples contenedores

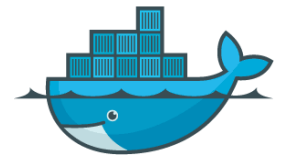


Kubernetes & Docker work together to build & run containerized applications



<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.

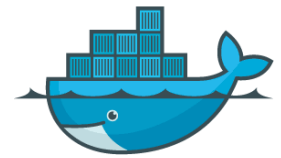
Y para crear la imagen:

```
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

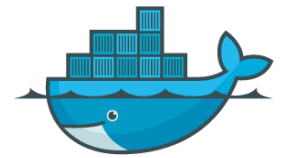
| -- index.html

```
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 .`

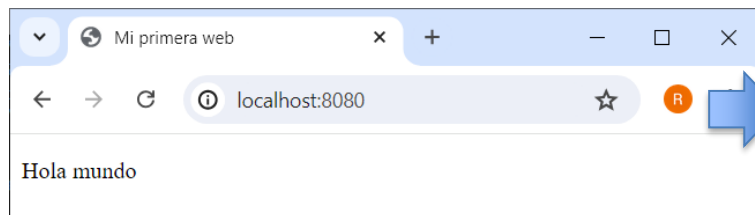
Mostrar imágenes disponibles: `docker images`

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`

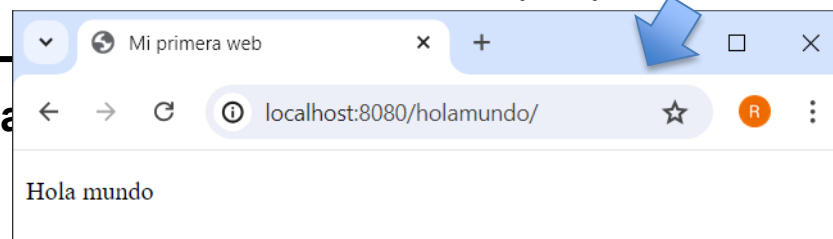
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
89f36eefa5f7	mi_imagen:v1	"httpd-foreground"	16 seconds ago	Up 15 seconds	0.0.0.0:8080->80/tcp	silly_banach



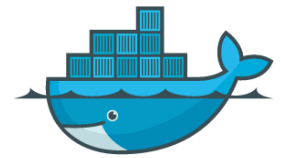
Vemos que la ruta es localhost:8080, pero si queremos añadir una subruta (p.ej. localhost:8080/holamundo), basta modificar el Dockerfile, añadiendo la subruta holamundo al final de la instrucción COPY

```
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.

TAG			
v1			
Last pushed 12 minutes ago by rsanzfloridauni			
Digest	OS/ARCH	Last pull	Compressed Size ⓘ
3a66eb44d413	linux/amd64	12 minutes ago	56.62 MB

```
docker pull rsanzfloridauni/mi_imagen:v1
```

[Copy](#)

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:

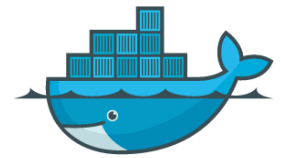
```
docker run -d -p 8080:80 rsanzfloridauni/mi_imagen.v1
```

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



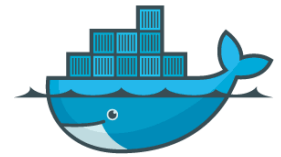
Ejemplo web cine (I): introducción

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 rellena).
- **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



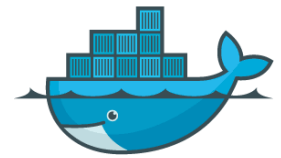
Ejemplo web cine (II): la base de datos

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
<input type="checkbox"/>	1	id 	int(11)			No	Ninguna		AUTO_INCREMENT
<input type="checkbox"/>	2	titulo	varchar(100)	utf8mb4_general_ci		No	Ninguna		
<input type="checkbox"/>	3	director	varchar(100)	utf8mb4_general_ci		No	Ninguna		
<input type="checkbox"/>	4	nota	varchar(4)	utf8mb4_general_ci		No	Ninguna		
<input type="checkbox"/>	5	anyo	int(4)			No	Ninguna		
<input type="checkbox"/>	6	presupuesto	int(4)			No	Ninguna		
<input type="checkbox"/>	7	img_base64	mediumtext	utf8mb4_general_ci		No	Ninguna		
<input type="checkbox"/>	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=contrasenyRoot
ENV MYSQL_DATABASE=cine
ENV MYSQL_USER=usuario1
ENV MYSQL_PASSWORD=contrasenyUsuario1

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

Tu usuario de
Docker Hub

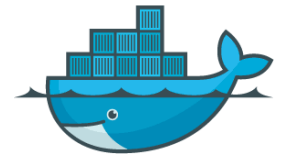
Construir la imagen: `docker build -t mi_cine:v1 .`

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 = "contrasenaUsuario1";
$dbname = "cine";

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

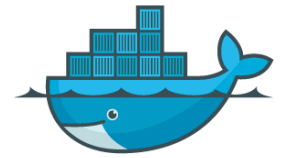
// Verificar la conexión
if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

// 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

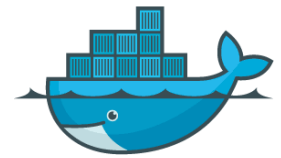
6. Imágenes personalizadas



```
<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($row["presupuesto"]) . "</td>";
            echo "<td><img src='data:image/jpeg;base64,'" . htmlspecialchars($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 .`

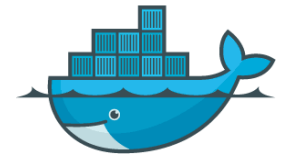
Etiquetar la imagen: `docker tag mi_app_cine:v1 rsanzfloridauni/mi_app_cine:v1`

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



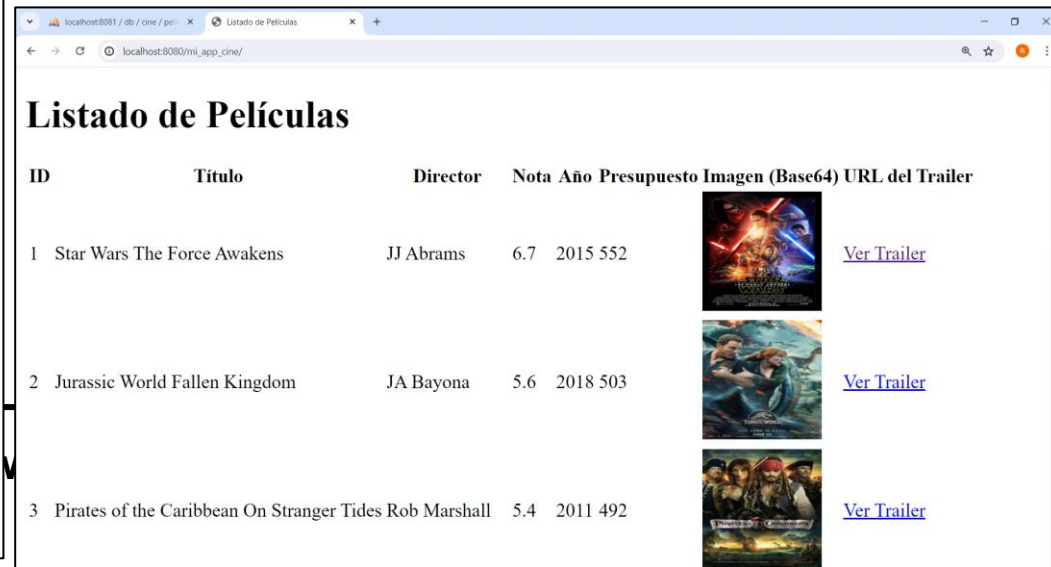
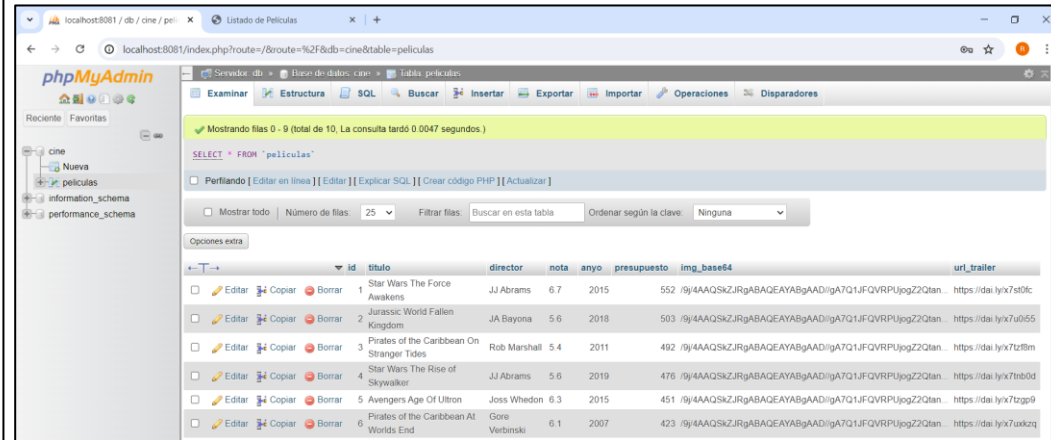
Ejemplo web cine (VI): Docker Compose (compose.yaml)

```
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"
```

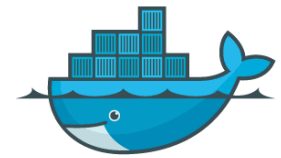
```
php:
  image: rsanzfloridauni/mi_app_cine:v1
  container_name: contenedorMiApp
  ports:
    - "8080:80"
  depends_on:
    - db
```

```
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



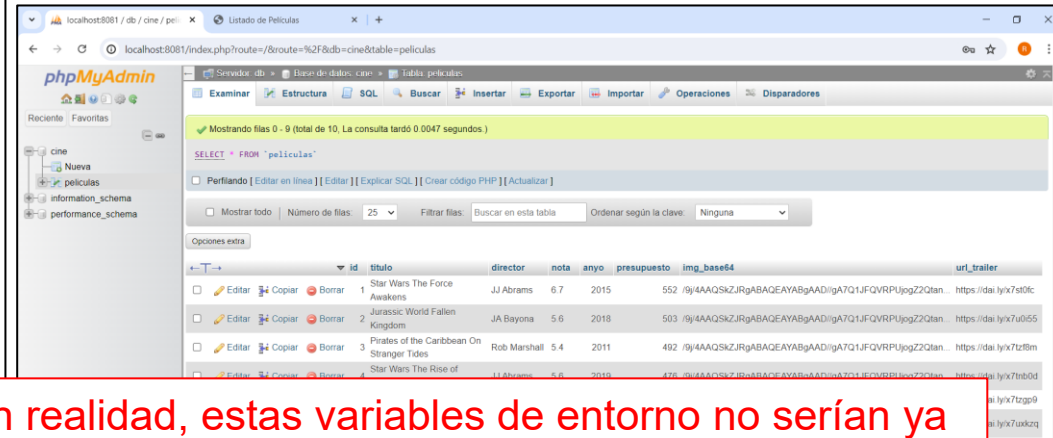
Ejemplo web cine (VI): Docker Compose (compose.yaml)

```
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"
```




`docker compose up -d`

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

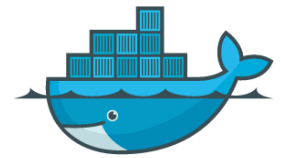
```
phpmyadmin:
  image: phpmyadmin/phpmyadmin:latest
  container_name: contenedorPhpMyAdmin
  environment:
    PMA_HOST: db
    MYSQL_ROOT_PASSWORD: contrasenyaRoot
  ports:
    - "8081:80"
  depends_on:
    - db
```



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.

1	Star Wars The Force Awakens	JJ Abrams	6.7	2015	552		Ver Trailer
2	Jurassic World Fallen Kingdom	JA Bayona	5.6	2018	503		Ver Trailer
3	Pirates of the Caribbean On Stranger Tides	Rob Marshall	5.4	2011	492		Ver Trailer

6. Imágenes personalizadas



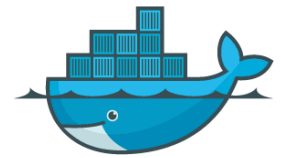
Estructura de directorios

```
/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.

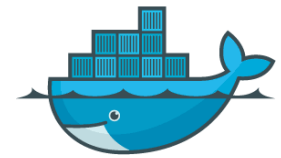
6. Imágenes personalizadas



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



phpMyAdmin

No es seguro 34.226.153.218:8081

phpMyAdmin
Bienvenido a phpMyAdmin

Idioma (Language)

Español - Spanish






Iniciar sesión

Usuario:

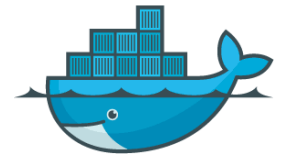
Contraseña:

Iniciar sesión

Listado de Películas

ID	Titulo	Director	Nota	Año	Presupuesto	Imagen (Base64)	URL del Trailer
1	Star Wars The Force Awakens	JJ Abrams	6.7	2015	552		Ver Trailer
2	Jurassic World Fallen Kingdom	JA Bayona	5.6	2018	503		Ver Trailer
3	Pirates of the Caribbean On Stranger Tides	Rob Marshall	5.4	2011	492		Ver Trailer
4	Star Wars The Rise of Skywalker	JJ Abrams	5.6	2019	476		Ver Trailer
5	Avengers Age Of Ultron	Joss Whedon	6.3	2015	451		Ver Trailer

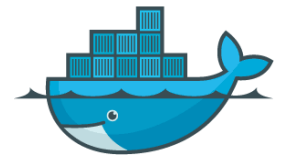
6. Imágenes personalizadas



¿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



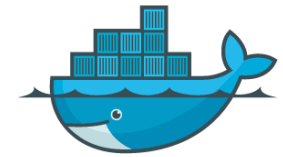
¿Qué hace este Dockerfile?

```
FROM php:8.3-apache Parte de una imagen de PHP 8.3 con Apache
WORKDIR /www/html Establece /www/html como directorio de trabajo dentro del contenedor (alojará la aplicación)
RUN apt-get update Actualiza el índice de paquetes de Debian
RUN apt-get -y install nano Instala herramientas útiles para desarrollar y gestionar archivos
RUN apt-get -y install zip Instala extensiones mysqli,
RUN apt-get -y install unzip pdo y pdo_mysql para PHP
RUN apt-get -y install git Descarga el script de
RUN docker-php-ext-install mysqli pdo pdo_mysql instalación de
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');" Composer (gestor de
RUN php composer-setup.php Ejecuta el script (instala Composer) dependencias de PHP)
RUN php -r "unlink('composer-setup.php');" Elimina el script para limpiar el entorno
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 Instala Symfony CLI (interfaz de línea de
COPY ../ 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`

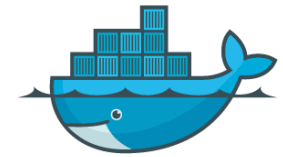
Windows: descarga e instala la última versión LTS de <https://nodejs.org/en/download/prebuilt-installer>

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
```

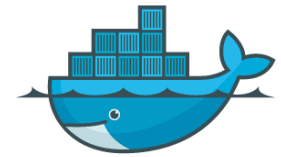
```
src/app/app.component.ts
```

```
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

```
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',
  styleUrls: ['./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.ts

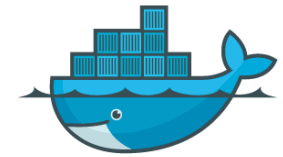
src/app/suma/suma.component.html

```
<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>
```

DAW - Despliegue de

4. Contenedores

6. Imágenes personalizadas



Ejemplo de Docker para una app desarrollada en Angular

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

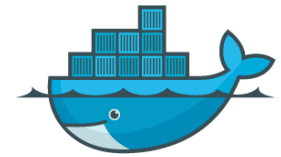
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [SumaComponent],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'suma-app';
}
```

src/app/app.component.ts

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

src/app/app.component.html

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

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

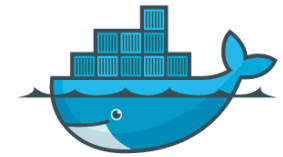
Autenticación: `docker login`

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

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

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

6. Imágenes personalizadas

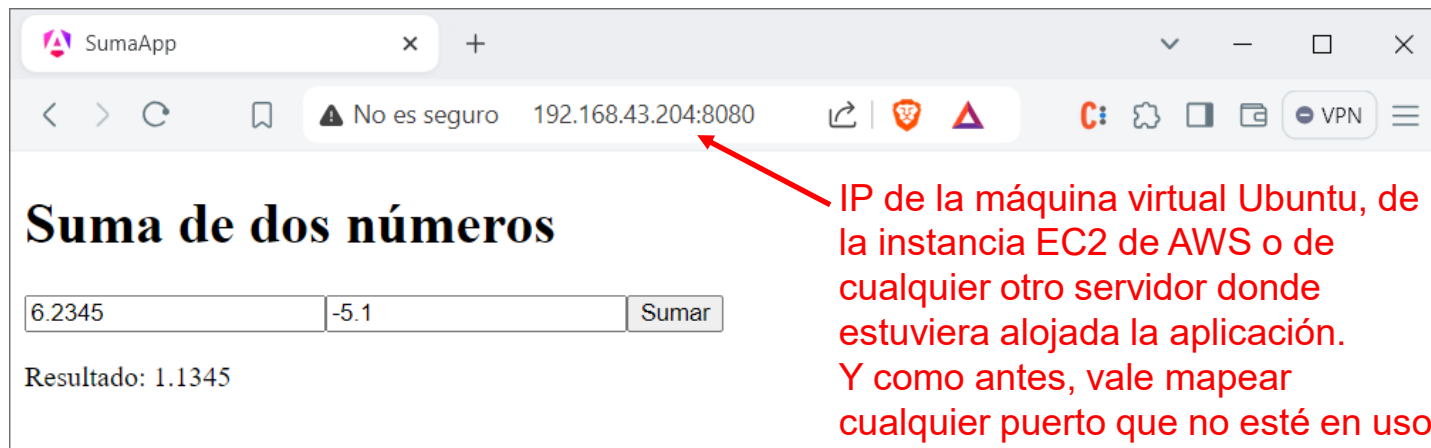


Ejemplo de Docker para una app desarrollada en Angular

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

DAW

Despliegue de aplicaciones web

5. Integración y despliegue continuo

Roberto Sanz Requena

rsanz@florida-uni.es

Índice

1. Introducción
2. Integración continua (CI)
3. Despliegue continuo (CD)
4. Servidores de automatización
5. Conclusiones
6. Anexo: ejemplo de *pipeline* CI/CD para Angular

1. Introducción

Pongamos por caso...

Un cliente pide a tu empresa de desarrollo una aplicación web con 5 funcionalidades.

Acordáis un plazo de 2 meses para entregar una primera versión estable que pueda estar ya en producción. Es un plazo muy ajustado, pero es un cliente potencial muy importante y no quieres negociar.

Dada la importancia del cliente, destinas 4 desarrolladores (2 sénior y 2 juniors) al proyecto.

Por otro lado, en tu empresa cuentas con un titulado DAW que se suele encargar de la parte de operaciones (puesta en producción de las aplicaciones, monitorización de servidores, control de estabilidad, seguridad, etc.).

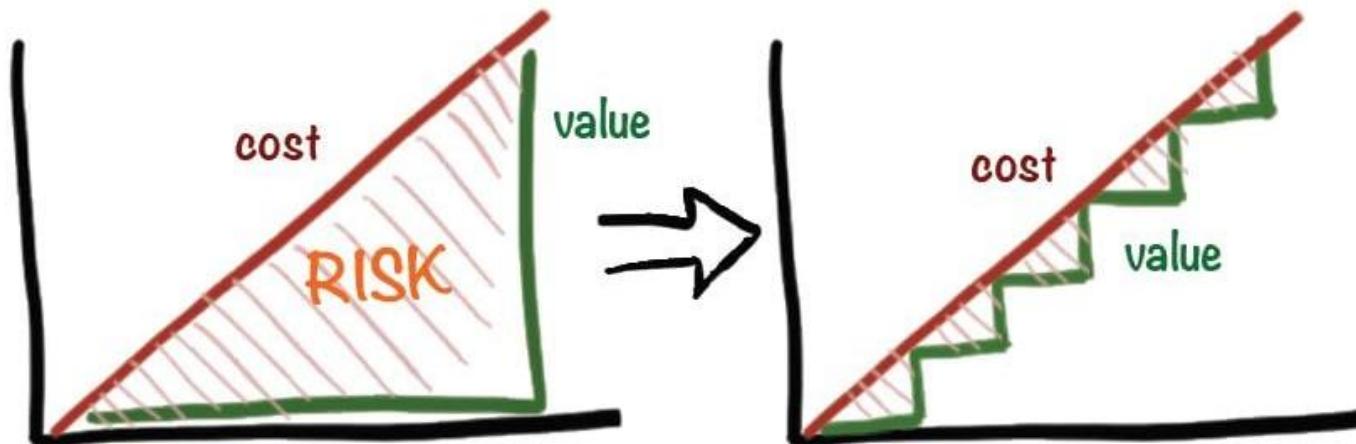
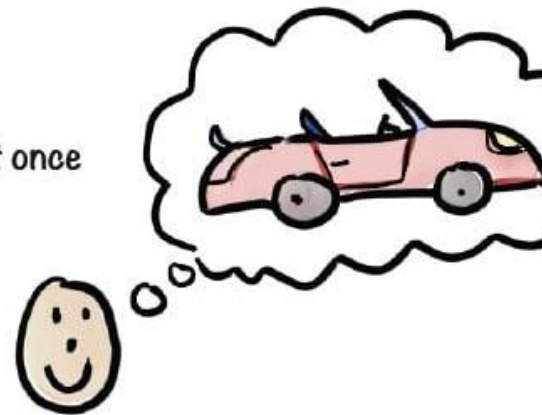
¿Cómo organizarías el trabajo?

1. Introducción

Agile = Iterative + Incremental

Don't try to get it all right from the beginning

Don't build it all at once



<https://www.equinox.co.nz/blog/agile-practices-manage-project-risk>

1. Introducción

El tipo de software también influye...

Aplicación web en producción

Al primer día de producción se detecta un error en una vista web.

Se corrige un fichero HTML en el propio servidor de la aplicación, sin necesidad de recompilar la aplicación ni reiniciar el servicio.

Coste: una disculpa y un café.

Software de la sonda Mars Climate Orbiter

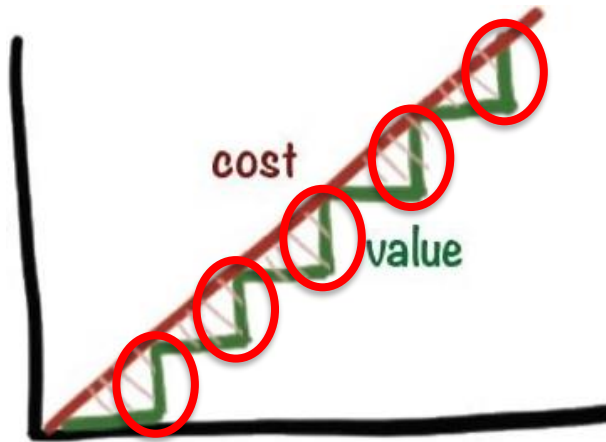
Nueve meses y medio después del lanzamiento, en vez de quedarse en la órbita de Marte, se acerca demasiado y se desintegra en la atmósfera.

El software de control terrestre funcionaba con el Sistema Anglosajón de Unidades y el de la nave con el Sistema Métrico Decimal (Sistema Internacional).

Coste: 327.6 millones de dólares (y la humillación mundial).

<https://blog.cdemi.io/analyzing-software-failure-on-the-nasa-mars-climate-orbiter/>

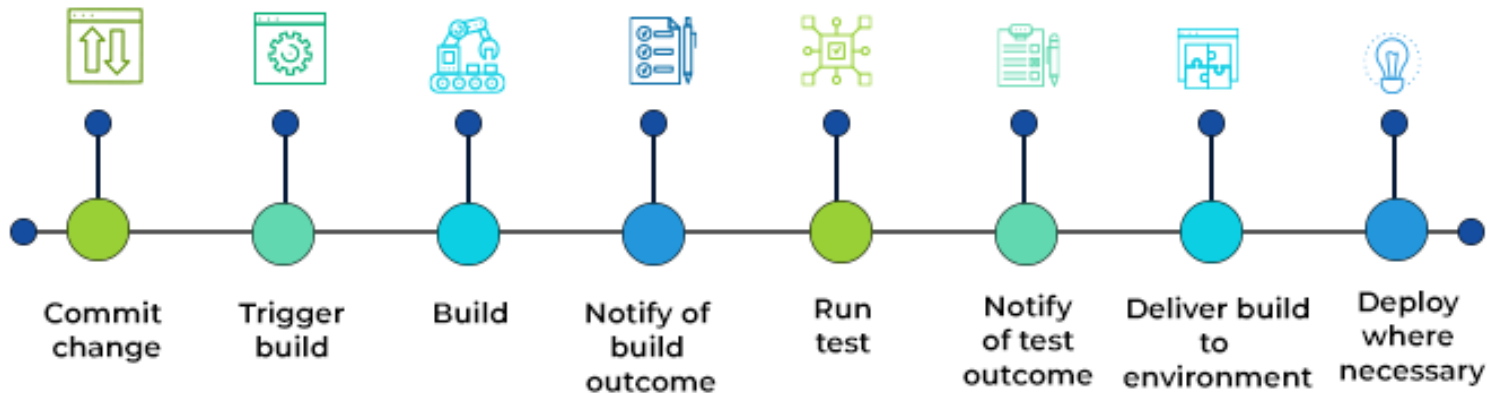
1. Introducción



Iteraciones

En cada iteración se integran los cambios que van realizando los desarrolladores y el técnico de operaciones puede ir revisando y dando *feedback* sobre los resultados del testeo y el despliegue a (pre)producción (*staging*).

Un paso más allá → Automatizar todo el proceso en una *pipeline*

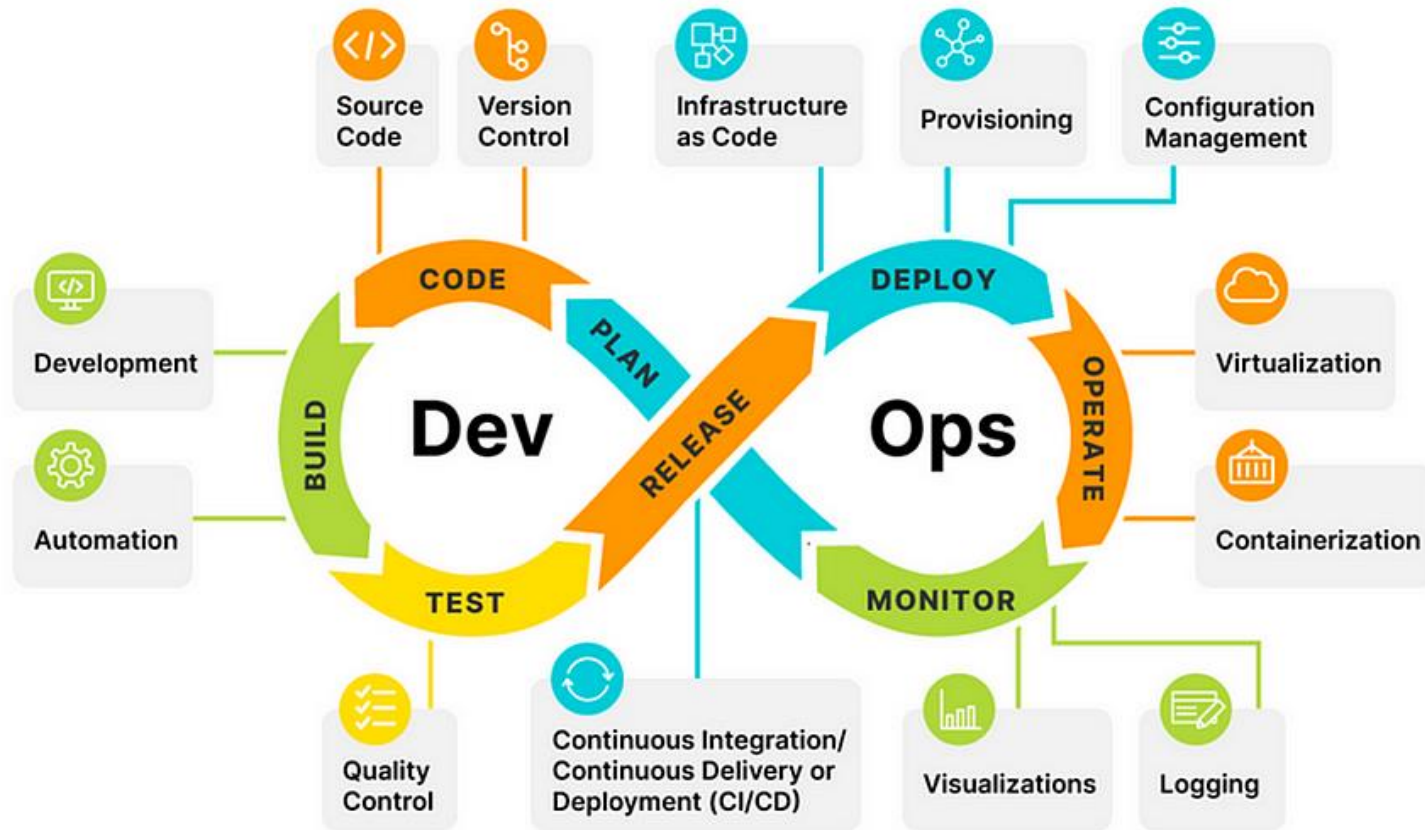


<https://www.spiceworks.com/tech/devops/articles/what-is-ci-cd/>

DAW - Despliegue de aplicaciones web

1. Introducción

DevOps: combinar elementos típicos de desarrollo y de operaciones



<https://zentagroup.medium.com/devops-en-la-era-de-las-ia-generativa-8a2e7badf49e>

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

1. Introducción

Componentes clave en integración y despliegue continuo **(Continuous Integration / Continuous Deployment)**

- ✓ Repositorios de código con control de versiones: Git y GitHub/GitLab
- ✓ Pruebas automatizadas: PHPUnit (PHP), JUnit (Java), Mocha (Javascript), etc.
- ✓ Herramientas de automatización: Jenkins, Travis CI, GitHub Actions, GitLab CI/CD
- ✓ Entornos de prueba y producción: servidores en la nube (AWS, Azure, etc.).

2. Integración continua

Práctica de desarrollo de software consistente en integrar frecuentemente los cambios en el código a un repositorio común (varias veces al día).

El objetivo es detectar de forma automatizada y resolver de manera rápida los posibles errores que haya en el código.

Pilares de la integración continua:



Ver tema 1

- ✓ **Control del código fuente (control de versiones):** entorno colaborativo (trabajo en ramas, *merging*, resolución de conflictos, etiquetado de versiones, etc.).
- ✓ **Compilación (*build*) automática:** compilar el código, gestionar dependencias y empaquetar el ejecutable (artefacto binario → carpetas “bin”, “dist”, “target”,...).
- ✓ **Testeo automático:** pruebas unitarias, de integración, funcionales, de eficiencia, etc.
- ✓ **Seguridad:** identificar vulnerabilidades, como contraseñas débiles, configuraciones inadecuadas o dependencias obsoletas.

2. Integración continua

Compilación (*build*) automática

Depende del tipo de lenguaje: lenguajes compilados vs interpretados

Compile



+ dependencias

C, C++, C#, Java,...

make dotnet maven/gradle

Interpreted



PHP, Javascript, Python,...

PHP es interpretado directamente por un servidor web (Apache/Nginx) a través de un módulo de PHP o a través de la línea de comandos (CLI).

Por su parte, Javascript es interpretado por el navegador.

<https://airoserver.com/dedicated/interpreted-vs-compiled-languages/>

2. Integración continua

Compilación (*build*) automática en lenguajes interpretados → *Bundle* (paquete)

PHP

Existe la opción de empaquetar toda la aplicación en un único fichero .phar (el servidor Apache/Nginx debe estar configurado para ejecutar estos ficheros).

Hay que asegurarse que en el fichero quedan incluidas también las dependencias necesarias (lo puedes hacer mediante Composer).

Javascript

También permite crear un *bundle* de la aplicación mediante herramientas como webpack, parcel o rollup, utilizando además npm o yarn para la gestión de las dependencias del proyecto. El *framework* Angular utiliza webpack.

NOTA: el “bundling” no es tan común en PHP. Sin embargo, en Javascript es aconsejable, sobre todo en web frontend, ya que el código Javascript se descarga al cliente y esta estrategia supone menos solicitudes HTTP y una carga más eficiente.



<https://snipcart.com/blog/javascript-module-bundler>

2. Integración continua

Testeo

Utilizar herramientas y scripts para ejecutar pruebas automáticas del software.

Tipos de pruebas automatizadas:

- **Pruebas unitarias**: verifican unidades individuales de código, como funciones o métodos.
- Pruebas de integración: evalúan cómo interactúan diferentes módulos del software.
- Pruebas funcionales: aseguran que las funciones del software cumplan con los requisitos especificados.
- Pruebas de sistema: comprueban el sistema en su totalidad para verificar su comportamiento en un entorno completo.

2. Integración continua

Testeo - Pruebas unitarias con PHPUnit

PHPUnit es un *framework* de testeo para realizar pruebas unitarias en PHP.

Su funcionamiento se basa en el uso de **aserciones** (*assertions*), que son métodos para verificar que el comportamiento del código es el esperado. Una aserción compara el resultado de la ejecución del código con el resultado esperado. Si coinciden, toma el valor “true” y la prueba pasa. En caso contrario, el valor es “false” y no pasa la prueba.

Consideraciones de las pruebas unitarias:

- Solo se prueban métodos públicos de cada clase.
- No se debe hacer uso de las dependencias de la clase bajo prueba.
- El test no debe implementar lógica de negocio (if...else, for, etc.).
- La estructura de un test unitario debe ser:
 - Preparación de los datos de entrada.
 - Ejecución del test.
 - Comprobación del test (*assertion*), con solo una comprobación por test.

<https://phpunit.de/>

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Instalación de Composer/PHPUnit

Primero vamos a instalar **Composer** (si no lo tienes ya instalado)...

Asumiendo que tienes instalado PHP 8.2 o superior, ejecuta las siguientes instrucciones desde una ruta que reconozca el comando php:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'dac665fdc30fdd8ec78b38b9800061b4150413ff2e3b6f88543c636f7cd84f6db9189d43a81e5503cda447da73c7
e5b6') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-
setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

La instalación habrá creado un fichero `composer.phar` en el directorio correspondiente. Puedes comprobar que se ha instalado correctamente: `composer --version`

La idea de utilizar Composer con PHPUnit es que cada proyecto incorpore su módulo de testeo como una dependencia más en la fase de desarrollo. Por tanto, deberemos instalar PHPUnit con Composer para cada proyecto. Esto facilita el trabajo.

<https://phpunit.de/>

<https://getcomposer.org/download/>

NOTA: Si no reconoce la instrucción `composer`, puedes utilizar la instrucción `php composer.phar` en su lugar (p.ej. `php composer.phar --version`).

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Ejemplo “Calculadora”

Crea una carpeta nueva llamada “calculadora”. Sitúate en la carpeta e instala PHPUnit:

```
composer require --dev phpunit/phpunit
```

Se han creado dos ficheros (composer.json y composer.lock) y una carpeta (vendor). En la carpeta /vendor/bin está el ejecutable phpunit.

Arranca VSCode u otro editor para desarrollar el proyecto. La estructura final del proyecto deberá ser la siguiente:

```
calculadora
|-- /src
|   |-- Calculadora.php
|-- /tests
|   |-- CalculadoraTest.php
|-- /vendor
|-- composer.json
|-- composer.lock
```

<https://phpunit.de/>
<https://getcomposer.org/download/>

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Ejemplo “Calculadora”

```
{
    "require-dev": {
        "phpunit/phpunit": "^11.4"
    },
    "autoload": {
        "psr-4": {
            "App\\": "src"
        }
    }
}
```

composer.json

```
<?php

namespace App;

class Calculadora {
    public function suma($a, $b) {
        return $a + $b;
    }
}
```

Calculadora.php

```
<?php

use PHPUnit\Framework\TestCase;
use App\Calculadora;

class CalculadoraTest extends TestCase {
    public function testSuma() {
        $calc = new Calculadora();
        $this->assertEquals(5, $calc->suma(3,2));
    }
}
```

CalculadoraTest.php

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Ejemplo “Calculadora”

```
{
    "require-dev": {
        "phpunit/phpunit": "^11.4"
    },
    "autoload": {
        "psr-4": {
            "App\\": "src"
        }
    }
}
```

composer.json

Indica a composer cómo cargar automáticamente las clases necesarias (estándar PSR-4)

Importante: mismo método que en Calculadora.php pero con “test” delante (y camelCase)

```
<?php
namespace App;

class Calculadora {
    public function suma($a, $b) {
        return $a + $b;
    }
}
```

Calculadora.php

```
<?php
use PHPUnit\Framework\TestCase;
use App\Calculadora;

class CalculadoraTest extends TestCase {
    public function testSuma() {
        $calc = new Calculadora();
        $this->assertEquals(5, $calc->suma(3,2));
    }
}
```

CalculadoraTest.php

Método para evaluar igualdad

Indicamos que es una clase de testeo

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Ejemplo “Calculadora”

Definido el proyecto y su código, vamos a realizar el testeo. Desde el terminal:

1. Cargar las clases necesarias con Composer: `composer dump-autoload`
2. Ejecución del test: `vendor/bin/phpunit tests`

```
PHPUnit 11.0.0 by Sebastian Bergmann and contributors.  
Runtime:      PHP 8.2.12  
.  
Time: 00:00.035, Memory: 6.00 MB  
OK (1 test, 1 assertion)                                1 / 1 (100%)
```

NOTA: es conveniente acostumbrarse a ejecutar estas instrucciones desde el directorio raíz del proyecto, para evitar confusiones con las rutas a los ficheros necesarios.

2. Integración continua



Testeo - Pruebas unitarias con PHPUnit - Asserts

```
<?php

use PHPUnit\Framework\TestCase;
use App\Calculadora;

class CalculadoraTest extends TestCase {
    public function testSuma() {
        $calc = new Calculadora();
        $resultado = $calc -> suma(3, 2);
        $this -> assertEquals(5, $resultado);
        $this -> assertSame(5, $resultado);
        $this -> assertGreaterThan(6, $resultado);
        $this -> assertNotNull($resultado);
    }
}
```

There was 1 failure:

1) CalculadoraTest::testSuma
Failed asserting that 5 is greater than 6.

...\calculadora\tests\CalculadoraTest.php:12

FAILURES!

Tests: 1, Assertions: 3, Failures: 1.

Para una lista completa de las pruebas que permite PHPUnit, puedes consultar la documentación (<https://docs.phpunit.de/en/11.2/assertions.html>).

2. Integración continua



PHPUnit

Testeo - Pruebas unitarias con PHPUnit - Desarrollo y despliegue

Es importante no perder de vista que la fase de testeo es parte del proceso de desarrollo de la aplicación, por lo que los archivos necesarios para el testeo no deben desplegarse en un entorno de producción.

Interesa, por tanto, que lo que llegue al servidor de producción sea solo lo necesario para que nuestra aplicación funcione.

Pasos:

1. Gestión del control de versiones (si desplegamos a través de GitHub)
2. Preparación local del código que irá a producción
3. Despliegue en servidor web de la aplicación “limpia”

2. Integración continua



PHPUnit

Testeo - Pruebas unitarias con PHPUnit - Desarrollo y despliegue

Gestión del control de versiones

El despliegue de la aplicación se puede configurar a partir de un *push* realizado a GitHub. Por ello, es importante que el código que llegue a GitHub esté “limpio” y preparado para producción.

Por ello, hay que utilizar el fichero `.gitignore` para indicar lo que no queremos que se suba a GitHub. Para el ejemplo de la Calculadora, deberías incluir en `.gitignore`:

```
/vendor/  
/tests/
```

2. Integración continua



PHPUnit

Testeo - Pruebas unitarias con PHPUnit - Desarrollo y despliegue

Preparación local del código que irá a producción

1. Crear un directorio para el despliegue: `mkdir despliegue`
2. Copiar al directorio todos los archivos con el código fuente:
`cp -R src despliegue`
`cp -R public despliegue`
`cp index.php despliegue`
3. Situar en el directorio de despliegue e instalar las dependencias de producción:
`composer install --no-dev --optimize-autoloader`
4. Transferir el contenido de la carpeta de despliegue al servidor correspondiente.

3. Despliegue continuo

El **despliegue continuo** es el conjunto de prácticas que permiten el proceso de entrega de cambios en el código, desde la fase de integración hasta el entorno de producción, de manera automática.

Es una forma de asegurarse que nuestra aplicación está siempre a punto de pasar a producción, de manera rápida y fiable, ya sea una versión final o una versión de prueba.

Ventajas:

- ✓ Reducción de riesgos: cambios pequeños y frecuentes.
- ✓ Mejora la calidad del software: pruebas automatizadas.
- ✓ Acelera el paso a la fase de producción: comercialización.
- ✓ Facilita el *feedback* de los usuarios.



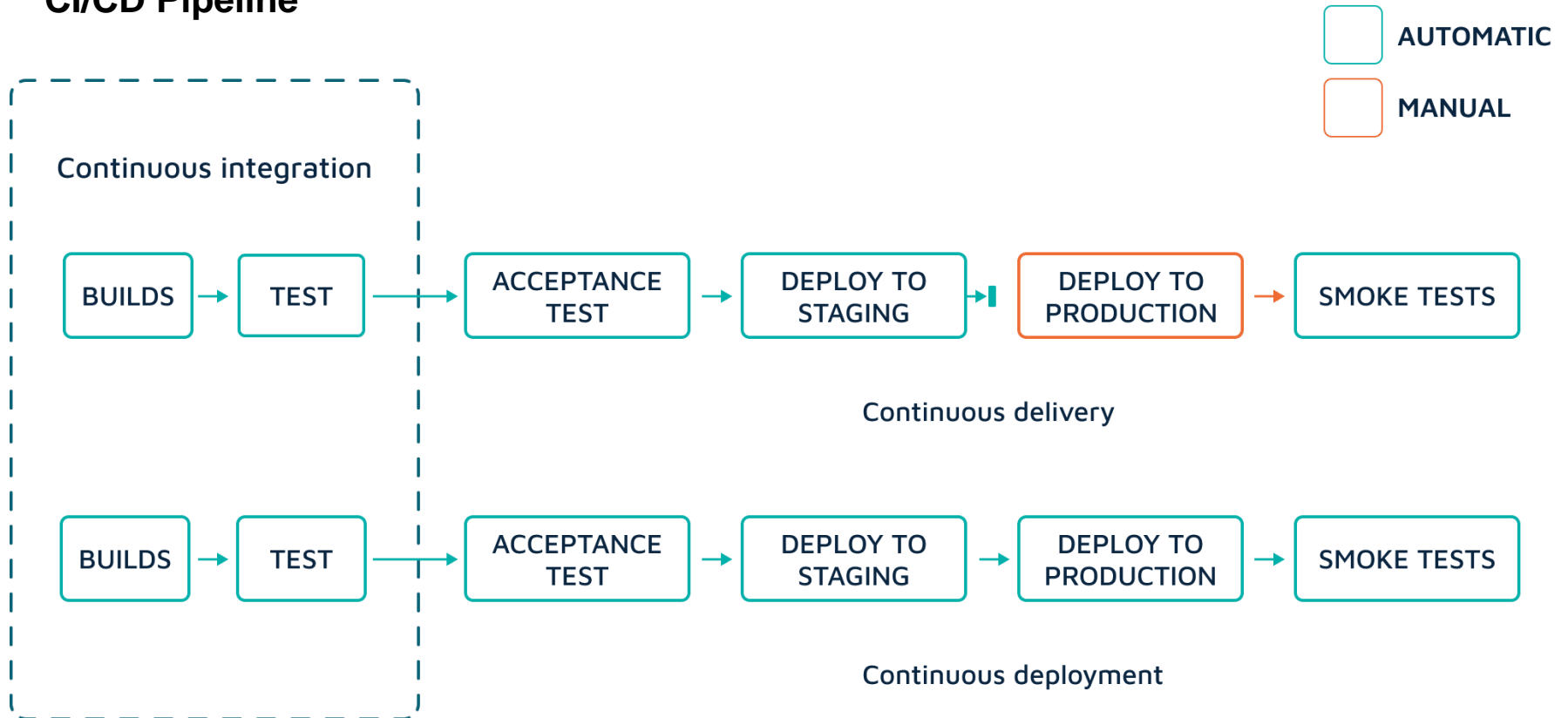
“Canary release”

Entrega continua

A diferencia del despliegue continuo, en la entrega continua, el despliegue no se realiza de manera automática (por ejemplo, cada vez que se hace un *push* a GitHub), sino que se debe iniciar manualmente.

3. Despliegue continuo

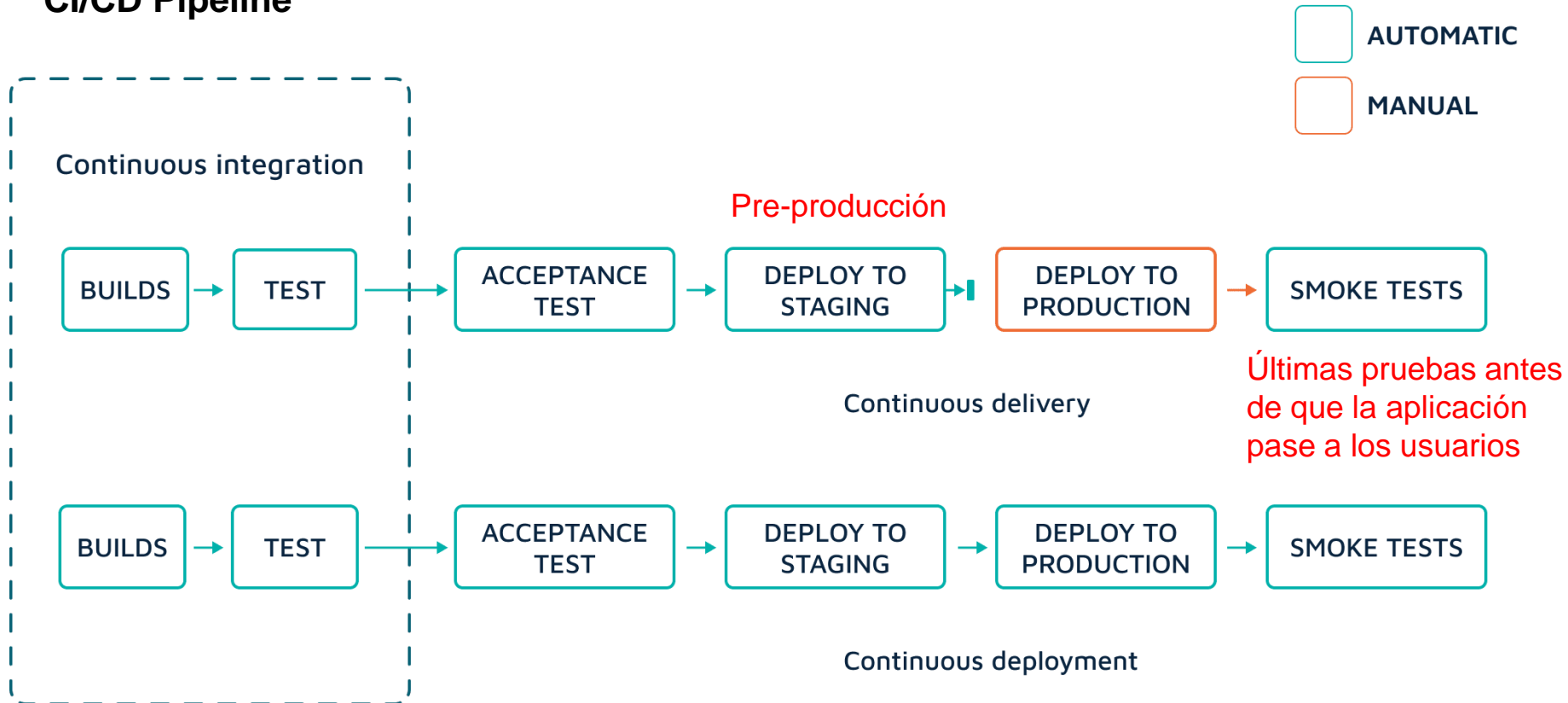
CI/CD Pipeline



<https://codefresh.io/learn/continuous-delivery/continuous-deployment-only-for-unicorns/>

3. Despliegue continuo

CI/CD Pipeline



<https://codefresh.io/learn/continuous-delivery/continuous-deployment-only-for-unicorns/>

4. Servidores de automatización

Los **servidores de automatización** son plataformas que facilitan la automatización de tareas repetitivas relacionadas con los procesos de integración continua (CI) y despliegue/entrega continuos (CD).

Se basan en scripts predefinidos que cubren la integración del código fuente, la realización de pruebas automáticas y el despliegue en diferentes entornos.

Ejemplos:



Infraestructura	Gestionada por el usuario	Gestionada por GitHub
Configuración	UI y Jenkinsfile (Groovy)	Archivos YAML
Integración	Multiplataforma	Nativa con GitHub
Escalabilidad	Gestión del usuario	Gestión automática
Extensibilidad	Amplio soporte de plugins	Marketplace de acciones
Coste	Infraestructura del usuario	Gratis (repositorio público)
Curva de aprendizaje	Alta	Media

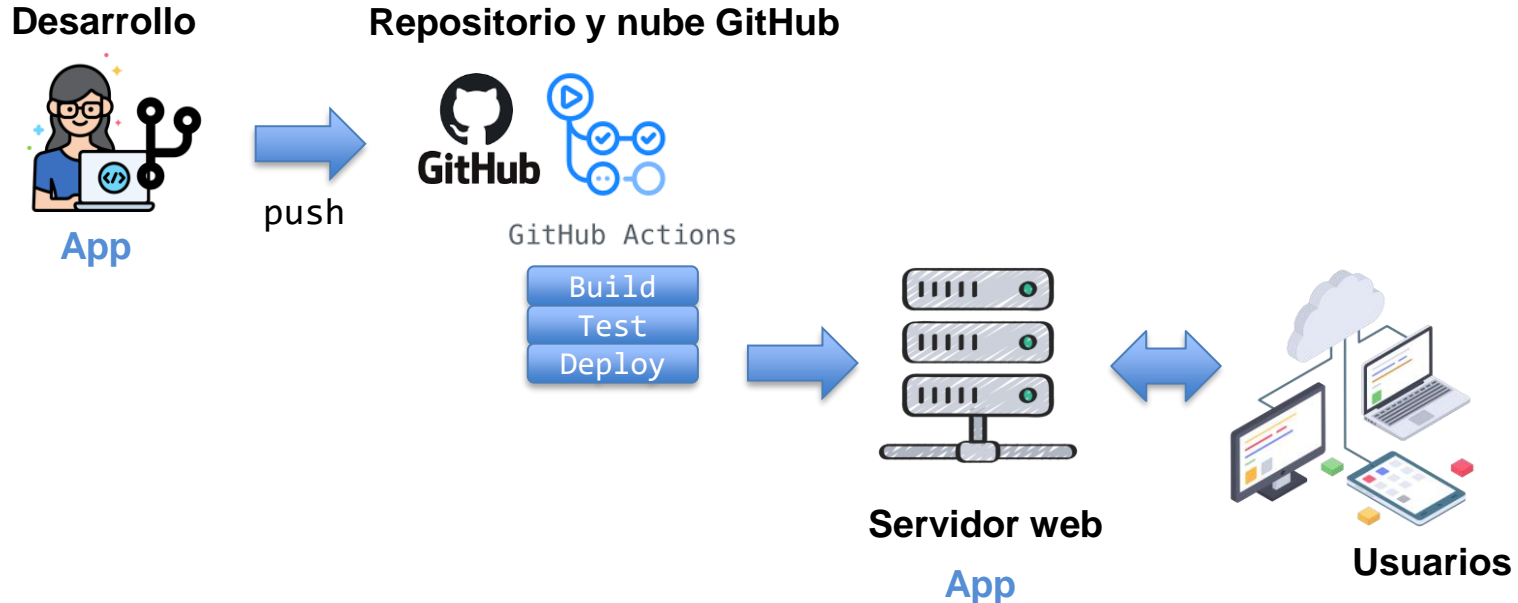
4. Servidores de automatización



GitHub Actions

GitHub Actions

GitHub Actions es una plataforma de CI/CD que permite ejecutar flujos de trabajo (compilación, testeo, despliegue, etc.) cuando sucede algún evento en el repositorio de GitHub.



<https://github.com/features/actions>

4. Servidores de automatización

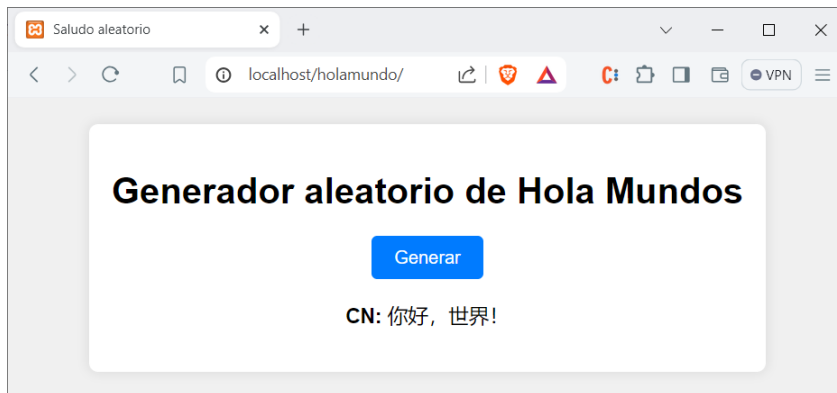


GitHub Actions

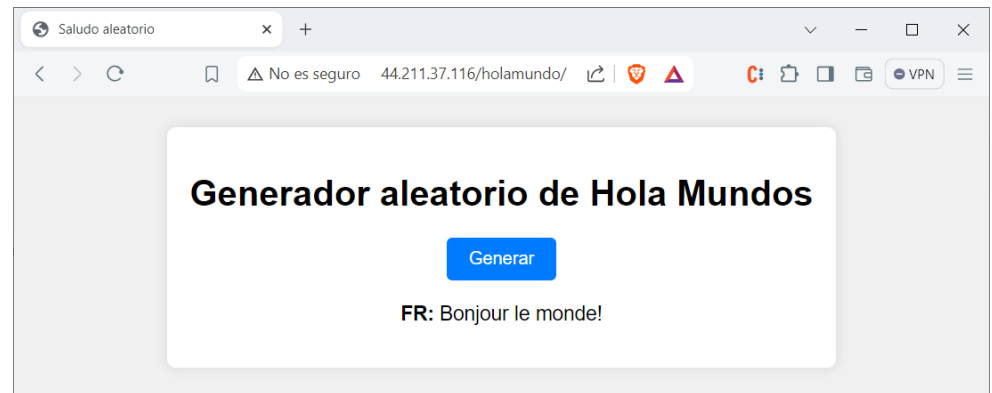
Ejemplo - Hola Mundo multilinguaje

Vamos a desarrollar una aplicación en local del tipo “Hola Mundo”, que mostrará una página con un botón. Al hacer clic, aparecerá el saludo “Hola Mundo” en un idioma aleatorio.

A través de GitHub Actions, configuraremos un *pipeline* de despliegue que se activará cada vez que se haga un *push* a la rama *main* de nuestro repositorio de GitHub. Este *pipeline* incluye una fase de testeo y una fase de despliegue, que transferirá la última versión de la aplicación (rama *main*) a un servidor web Apache alojado en una instancia EC2 de AWS con Ubuntu e IP pública.



Aplicación desarrollada en local



Aplicación desplegada automáticamente en AWS

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en local

1. Abre un terminal, crea una carpeta vacía llamada “holamundo_multi” y sitúate en ella.
2. Igual que hicimos anteriormente, instala PHPUnit: `composer require --dev phpunit/phpunit ^11` (se han creado los dos ficheros `-composer.json` y `composer.lock` y la carpeta `-vendor`).
3. En el directorio raíz del proyecto, crea un fichero `.gitignore` con el siguiente contenido:
vendor/
composer.phar
4. Crea dos carpetas en el directorio raíz: `src` y `tests`. En `src` irá el fichero de la clase que vamos a utilizar en la aplicación (`Saludo.php`) y en `tests`, la clase para testeo (`SaludoTest.php`).
5. Finalmente, crea un fichero `index.php` en el directorio raíz del proyecto.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en local

```
{
  "require-dev": {
    "phpunit/phpunit": "11"
  },
  "autoload": {
    "psr-4": {
      "HolaMundoMultilenguaje\\": "src"
    }
  }
}
```

composer.json

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en local

```
<?php

namespace HolaMundoMultilenguaje;

class Saludo
{
    private $saludos;

    public function __construct()
    {
        $this->saludos = [
            "es" => ["saludo" => "¡Hola Mundo!", "codigo" => "ES"], // Español
            "zh" => ["saludo" => "你好, 世界!", "codigo" => "CN"], // Chino
            "en" => ["saludo" => "Hello, World!", "codigo" => "US"], // Inglés
            "hi" => ["saludo" => "नमस्ते दुनिया!", "codigo" => "IN"], // Hindi
            "ar" => ["saludo" => "مرحباً بالعالم!", "codigo" => "AR"], // Árabe
            "pt" => ["saludo" => "Olá Mundo!", "codigo" => "PT"], // Portugués
            "ru" => ["saludo" => "Привет, мир!", "codigo" => "RU"], // Ruso
            "ja" => ["saludo" => "こんにちは、世界!", "codigo" => "JP"], // Japonés
            "de" => ["saludo" => "Hallo Welt!", "codigo" => "DE"], // Alemán
            "fr" => ["saludo" => "Bonjour le monde!", "codigo" => "FR"], // Francés
        ];
    }

    public function generarSaludoAleatorio()
    {
        $indice = array_rand($this->saludos);
        return $this->saludos[$indice];
    }
}
```

Saludo.php

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en local

```
<?php

use PHPUnit\Framework\TestCase;
use HolaMundoMultilenguaje\Saludo;

class SaludoTest extends TestCase
{
    public function testGenerarSaludoAleatorio()
    {
        $saludo = new Saludo();
        $resultado = $saludo->generarSaludoAleatorio();

        // Verifica que el resultado es un string
        $this->assertIsString($resultado['saludo']);

        // Verifica que el resultado contiene un saludo válido
        $this->assertNotEmpty($resultado['saludo']);

        // Verifica que el código también es un string
        $this->assertIsString($resultado['codigo']);

        // Verifica que el código no esté vacío
        $this->assertNotEmpty($resultado['codigo']);
    }
}
```

SaludoTest.php

4. Servidores de automatización



```
<?php

require_once __DIR__ . '/../src/Saludo.php';
use HolaMundoMultilenguaje\Saludo;
$saludo = new Saludo();
$mensaje = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $resultado = $saludo->generarSaludoAleatorio();
    $mensaje = $resultado['saludo'];
    $codigo = $resultado['codigo'];
}

?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Saludo aleatorio</title>
    <style>
        body { font-family: Arial, sans-serif; display: flex; justify-content: center; align-items: center;
            height: 100vh; background-color: #f0f0f0; margin: 0; }
        .container {text-align:center; background-color:#fff; padding:20px; border-radius:8px; box-shadow: 0 0 10px rgba(0,0,0,0.1);}
        button {padding: 10px 20px;font-size:16px;cursor:pointer;border:none;border-radius:5px;background-color:#007bff;color:#fff;}
        p { font-size: 18px; margin-top: 20px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Generador aleatorio de Hola Mundos</h1>
        <form method="post">
            <button type="submit">Generar</button>
        </form>
        <?php if (!empty($mensaje))
            echo "<p><b>".$codigo."</b> ".htmlspecialchars($mensaje)."</p>";
        ?>
    </div>
</body>
</html>
```

index.php

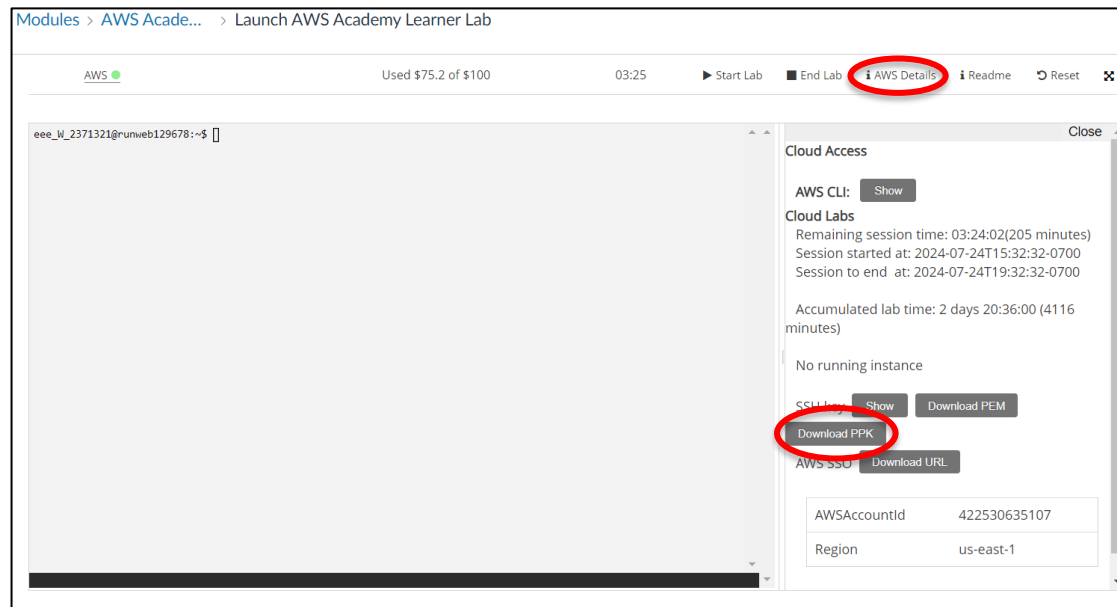
4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Preparación del servidor en AWS

1. Si no la has creado con anterioridad, crea una instancia EC2 (máquina virtual) de AWS con Ubuntu (puedes seguir la guía que hay disponible en Florida Oberta).
2. Desde la página de AWS Academy Learner Lab, en AWS Details, descarga la clave SSH con formato PPK (la necesitarás para acceder por PuTTY → ver tema 2 ←).



DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Preparación del servidor en AWS

3. Accede por SSH a la instancia EC2 (ver **tema 2**)
4. Actualiza la lista de paquetes: `sudo apt update`
5. Instala Apache: `sudo apt install apache2`
6. Instala PHP: `sudo apt install php libapache2-mod-php`
7. Habilita el módulo PHP en Apache: `sudo a2enmod php8.3` (o la versión que tengas)
8. Reinicia Apache: `sudo systemctl restart apache2`
9. Instala Composer: ver diapositivas previas
10. Prueba que el servidor funciona correctamente accediendo a la IP desde el navegador de tu PC local (debería mostrarse la página de bienvenida de Apache).

NOTA: recuerda que el directorio por defecto donde se alojará nuestra aplicación web en este servidor es `/var/www/html`.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en GitHub

1. Crea un repositorio público en GitHub (es necesario que sea público para el plan gratuito de GitHub Actions). En este ejemplo lo he llamado holamundomultilenguaje.
2. En “*Settings*”, accede a “*Secrets and variables → Actions*”. Este apartado permite almacenar de manera segura variables con información sensible, como nombres de usuario, contraseñas, IPs, etc. Dentro de “*Repository secrets*”, vamos a crear los siguientes:

Name: DEPLOY_PATH **Secret:** ruta de despliegue (p.ej. /var/www/html/holamundo)

Name: HOST **Secret:** IP de la instancia EC2 de AWS*

Name: SSH_PRIVATE_KEY **Secret:** contenido de la clave .pem de AWS**

Name: USERNAME **Secret:** nombre de usuario (p.ej. ubuntu)

* Ten en cuenta que si no configuras la IP pública de tu EC2 como elástica (“fija”), cada vez que inicies la EC2 cambiará, con lo que tendrás que cambiar también este secreto. Por tanto, es aconsejable configurar siempre tu máquina con IP elástica para evitar errores.

** No confundir la clave .pem con la clave .ppk, que se utiliza solo para PuTTY. El contenido del fichero .pem se debe copiar completo.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Trabajo en Local ↔ GitHub

1. En el directorio local del proyecto crea el fichero YAML (crea también la ruta) `./.github/workflows/despliegue.yml` (ver diapositiva siguiente).
2. Iniciar el repositorio: `git init`
3. Añadir el remoto: `git remote add origin https://github.com/rsanzfloridauni/holamundomultilenguaje.git`
4. Añade todos los ficheros para preparar el commit: `git add .`
5. Haz el commit: `git commit -m "Primer commit"`
6. Haz el push a GitHub: `git push -u origin main`

Si el *push* se realiza correctamente, se enviará todo al repositorio remoto y automáticamente se desencadenará la “Action” programada en el fichero `despliegue.yml`.

En GitHub, accede a la pestaña “Actions” del repositorio. Si lo haces inmediatamente después de hacer el *push*, te dará a tiempo a ver que la “Action” está aún en proceso.

```
name: CI/CD Pipeline for HolaMundoMultilenguaje
```

```
on:
```

```
  push:  
    branches:  
      - main
```

```
jobs:
```

```
  test:  
    runs-on: ubuntu-latest  
  
    steps:  
      - name: Checkout code  
        uses: actions/checkout@v2  
  
      - name: Set up PHP  
        uses: shivammathur/setup-php@v2  
        with:  
          php-version: '8.3'  
  
      - name: Install dependencies  
        run: composer install  
  
      - name: Run tests  
        run: ./vendor/bin/phpunit tests
```

```
  deploy:  
    runs-on: ubuntu-latest  
    needs: test
```

```
    steps:  
      - name: Checkout code  
        uses: actions/checkout@v2  
  
      - name: Set up PHP  
        uses: shivammathur/setup-php@v2  
        with:  
          php-version: '8.3'  
  
      - name: Install dependencies  
        run: composer install  
  
      - name: Deploy to server  
        env:  
          SSH_PRIVATE_KEY: ${ secrets.SSH_PRIVATE_KEY }  
          HOST: ${ secrets.HOST }  
          USERNAME: ${ secrets.USERNAME }  
          DEPLOY_PATH: ${ secrets.DEPLOY_PATH }  
        run: |  
          mkdir -p ~/.ssh  
          echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa  
          chmod 600 ~/.ssh/id_rsa  
  
          # Use SSH to create the deploy path if it doesn't exist  
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "mkdir -p $DEPLOY_PATH"  
  
          # Sync the application files to the server  
          rsync -avz --delete --no-t --exclude 'tests' --exclude '.git' . $USERNAME@$HOST:$DEPLOY_PATH  
  
          # Optionally, run Composer install on the server  
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "cd $DEPLOY_PATH && composer install --no-dev --optimize-autoloader"  
  
          # Restart Apache if needed  
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "sudo systemctl restart apache2"
```

```

name: CI/CD Pipeline for HolaMundoMultilenguaje Nombre del pipeline o workflow.

on: Evento (trigger) que lanza el pipeline o workflow. En este caso se lanza cuando se hace un push a la rama main (o master, si fuera el caso) del repositorio. Otra opción sería tener una rama específica de deploy que fuera solo para despliegue. O cualquier otra opción que nos venga bien para nuestro flujo de trabajo desarrollo-testeo-producción.
  push:
    branches:
      - main

jobs: Jobs o tareas que se realizan en el pipeline: en este caso hay dos (test y deploy).
  test: Tarea de test: se ejecuta en Ubuntu (última versión) y tiene varios pasos
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code Este paso usa la acción checkout para clonar el código fuente del repositorio en Ubuntu.
        uses: actions/checkout@v2

      - name: Set up PHP Este paso usa la acción setup-php (obtenida del repositorio de GitHub shivamathur) para instalar y configurar PHP en Ubuntu. Especifica que se utiliza la versión 8.3.
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'

      - name: Install dependencies Este paso ejecuta el comando composer install en el terminal de Ubuntu para instalar las dependencias del proyecto que estén definidas en el archivo composer.json.
        run: composer install

      - name: Run tests Este paso utiliza PHPUnit para ejecutar las pruebas que estén definidas en el directorio tests.
        run: ./vendor/bin/phpunit tests

  deploy: Tarea de deploy: se ejecuta en Ubuntu (última versión) y requiere para su ejecución (needs) que el trabajo de test se haya completado con éxito.
    runs-on: ubuntu-latest
    needs: test

    steps:
      - name: Checkout code Este paso usa la acción checkout para clonar el código fuente del repositorio en Ubuntu.
        uses: actions/checkout@v2

      - name: Set up PHP Este paso usa la acción setup-php (obtenida del repositorio de GitHub shivamathur) para instalar y configurar PHP en Ubuntu. Especifica que se utiliza la versión 8.3.
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'

      - name: Install dependencies Este paso ejecuta el comando composer install en el terminal de Ubuntu para instalar las dependencias del proyecto que estén definidas en el archivo composer.json.
        run: composer install

      - name: Deploy to server Este paso ejecuta el despliegue propiamente. Primero define unas variables de entorno (env), que son las que previamente habrás almacenado en los secretos del repositorio.
        env:
          SSH_PRIVATE_KEY: ${ secrets.SSH_PRIVATE_KEY }
          HOST: ${ secrets.HOST }
          USERNAME: ${ secrets.USERNAME }
          DEPLOY_PATH: ${ secrets.DEPLOY_PATH }
        run: |
          mkdir -p ~/.ssh
          echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa

          "run: |" ejecuta un bloque de comandos: "mkdir ..." crea el directorio "~/.ssh" si no existe;
          "echo ..." guarda la clave privada SSH en un archivo "id_rsa"; "chmod ..." ajusta los permisos de la clave privada para que solo el propietario pueda leer o escribir en ella.

          # Use SSH to create the deploy path if it doesn't exist Conexión SSH al servidor para crear el directorio de despliegue. La opción "StrictHostKeyChecking=no" previene interrupciones de la conexión.
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "mkdir -p $DEPLOY_PATH"

          # Sync the application files to the server
          rsync -avz --delete --no-t --exclude 'tests' --exclude '.git' . $USERNAME@$HOST:$DEPLOY_PATH

          # Optionally, run Composer install on the server
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "cd $DEPLOY_PATH && composer install --no-dev --optimize-autoloader"

          # Restart Apache if needed
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "sudo systemctl restart apache2"

```

```

name: CI/CD Pipeline for HolaMundoMultilenguaje

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'

      - name: Install dependencies
        run: composer install

      - name: Run tests
        run: ./vendor/bin/phpunit tests

  deploy:
    runs-on: ubuntu-latest
    needs: test

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up PHP
        uses: shivammathur/setup-php@v2
        with:

```

Sincroniza (transfiere) archivos entre repositorio y servidor con las siguientes opciones:

- a: preserva propiedades de los archivos;
- v: muestra información de progreso;
- z: comprime datos para la transferencia;
- delete: borra archivos en destino que no existan en el repositorio;
- no-t: no preserva tiempos de modificación (puede dar problemas en el servidor)
- exclude 'tests' -exclude '.git': excluyen los directorios tests y .git de la sincronización (no estarán en el código desplegado)

```

echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa
chmod 600 ~/.ssh/id_rsa

# Use SSH to create the deploy path if it doesn't exist
ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "mkdir -p $DEPLOY_PATH"

# Sync the application files to the server
rsync -avz --delete --no-t --exclude 'tests' --exclude '.git' $USERNAME@$HOST:$DEPLOY_PATH

# Optionally, run Composer install on the server
ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "cd $DEPLOY_PATH && composer install --no-dev --optimize-autoloader"

# Restart Apache if needed
ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "sudo systemctl restart apache2"

```

Conexión vía SSH al servidor y ejecución de la instrucción `composer install` en el directorio de despliegue, pero excluyendo las dependencias de desarrollo (`--no-dev`) (no instala PHPUnit, en este caso) y optimizando el cargador automático.

Conexión vía SSH al servidor y reinicio del servidor Apache para aplicar cualquier cambio en la configuración o en los archivos.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - GitHub Action en proceso

The screenshot shows the GitHub Actions interface for the repository 'rsanzfloridauni / holamundomultilenguaje2'. The 'Actions' tab is selected, showing a list of workflows on the left and a detailed view of a workflow run on the right. The workflow run is titled 'Primer commit' and is currently in the 'In progress' state. The commit hash is '1665dce' and it was pushed by 'rsanzfloridauni'. The branch is 'main'. The workflow is identified as 'CI/CD Pipeline for PHP #1: Commit 1665dce pushed by rsanzfloridauni'.

Esta captura muestra el *workflow* justo después de hacer el *push* desde local a GitHub. Se puede ver que está en proceso (“*In progress*”) y que aparece identificado por el *commit* que hemos hecho (“Primer commit”).

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - GitHub Action finalizada

The screenshot shows the GitHub Actions interface for a workflow named 'CI/CD Pipeline for PHP'. The workflow is triggered by a push to the 'main' branch. The run is successful, with a total duration of 50s. The workflow consists of two jobs: 'test' (11s) and 'deploy' (20s). The 'despliegue.yml' file is shown with the trigger 'on: push'.

Trigger	Status	Total duration	Artifacts
Triggered via push 12 minutes ago	Success	50s	-

Jobs:

- test (11s)
- deploy (20s)

Workflow finalizado. Al hacer clic en el título del *commit* (“Primer *commit*”) se muestra el detalle. Se puede ver que aparecen dos partes: *test* y *deploy*, que se corresponden con los *jobs* que se indicaban en el fichero `despliegue.yml`. En este caso, ambos han finalizado correctamente, por lo que aparecen en verde. En caso de error, aparecerían en rojo y al hacer clic en ellos nos mostrarían el detalle del error.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - GitHub Action finalizada

The screenshot shows the GitHub Actions interface for a workflow run. On the left, the 'Summary' tab is active, showing a list of jobs: 'test' and 'deploy'. The 'test' job is selected. The main panel displays the details of the 'test' job, which succeeded 15 minutes ago in 11s. The job steps are listed on the right, with 'Run tests' expanded to show the output logs. The logs indicate that the tests passed successfully, with a runtime of 00:00.004 and memory usage of 8.00 MB. The final status is 'OK (1 test, 4 assertions)'.

```
test
succeeded 15 minutes ago in 11s

> Set up job 1s
> Checkout code 0s
> Set up PHP 4s
> Install dependencies 2s
> Run tests 0s
  1 ▶ Run ./vendor/bin/phpunit tests
  8 PHPUnit 11.2.8 by Sebastian Bergmann and contributors.
  9
 10 Runtime:      PHP 8.3.9
 11
 12 .
 13                                     1 / 1 (100%)
 14 Time: 00:00.004, Memory: 8.00 MB
 15
 16 OK (1 test, 4 assertions)

> Post Checkout code 0s
> Complete job 0s
```

Detalle del *job* de *test*, donde se muestran los tests realizados de manera correcta. Para el *job* de *deploy* también se puede ver todo lo que se ha realizado automáticamente.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Error de testeo

Puedes probar de forzar un error de testeo para ver que el *workflow* da error y no termina. En este caso, he modificado la última línea de `SaludoTest.php` para que compruebe que `$resultado['código']` sea vacío (`assertEmpty`). Hago commit y push, y ahora vemos que el proceso no acaba correctamente y, por tanto, no hay despliegue.

Actions

New workflow

All workflows

CI/CD Pipeline for PHP

Management

Caches

Attestations

Runners

CI/CD Pipeline for PHP

despliegue.yml

2 workflow runs

Event Status Branch Actor

Forzar error de testeo

CI/CD Pipeline for PHP #2: Commit `ad728f3` pushed by `rsanzfloridauni`

main

3 minutes ago

23s

Run tests

0s

1 ▶ Run `./vendor/bin/phpunit tests`

8 PHPUnit 11.2.8 by Sebastian Bergmann and contributors.

9

10 Runtime: PHP 8.3.9

11

12 F 1 / 1 (100%)

13

14 Time: 00:00.006, Memory: 8.00 MB

15

16 There was 1 failure:

17

18 1) SaludoTest::testGenerarSaludoAleatorio

19 Failed asserting that a string is empty.

20

21 /home/runner/work/holamundomultilenguaje2/holamundomultilenguaje2/tests/SaludoTest.php:23

22

23 FAILURES!

24 Tests: 1, Assertions: 4, Failures: 1.

25 Error: Process completed with exit code 1.

DAW - Despliegue

5. Integración y despliegue

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje

Al finalizar, el proyecto deberá tener la siguiente estructura:

```
/holamundo
|-- /.github
|   |-- /workflows
|       |-- despliegue.yml
|-- /src
|   |-- Saludo.php
|-- /tests
|   |-- SaludoTest.php
|-- /vendor
|-- .gitignore
|-- composer.json
|-- composer.lock
|-- index.php
```

4. Servidores de automatización



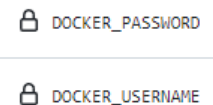
GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Automatizar imágenes Docker

Como un plus al despliegue realizado, también puedes automatizar que se cree una imagen Docker de tu aplicación y se guarde en Docker Hub.

Hay que realizar los siguientes pasos:

1. En el repositorio, crear un fichero `Dockerfile` con las instrucciones para crear una imagen personalizada a partir del código de la aplicación.
2. En GitHub, añadir los secretos necesarios para iniciar sesión en Docker Hub.
3. En el fichero YAML, crear la imagen a partir del fichero `Dockerfile`.
4. En el fichero YAML, iniciar sesión en Docker Hub.
5. En el fichero YAML, enviar la imagen a Docker Hub.



🔒 DOCKER_PASSWORD

🔒 DOCKER_USERNAME

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Automatizar imágenes Docker

Dockerfile

```
FROM php:8.3-apache

# Instala extensiones de PHP necesarias
RUN docker-php-ext-install mysqli pdo pdo_mysql

# Copia el código fuente de tu aplicación al contenedor
COPY . /var/www/html/

# Establece los permisos adecuados para el directorio de la aplicación
RUN chown -R www-data:www-data /var/www/html
RUN chmod -R 755 /var/www/html

# Expone el puerto 80 para que Apache pueda recibir tráfico web
EXPOSE 80

# Comando para mantener el contenedor en ejecución mientras Apache funcione
CMD ["apache2-foreground"]
```

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Automatizar imágenes Docker

Añadir al final del fichero despliegue.yml (en el bloque deploy)

- name: Build Docker image
run: docker build -t \${{ secrets.DOCKER_USERNAME }}/hola-mundo-multilenguaje:\${{ github.sha }} .
- name: Log in to Docker Hub
uses: docker/login-action@v2
with:
 username: \${{ secrets.DOCKER_USERNAME }}
 password: \${{ secrets.DOCKER_PASSWORD }}
- name: Push Docker image to Docker Hub
run: docker push \${{ secrets.DOCKER_USERNAME }}/hola-mundo-multilenguaje:\${{ github.sha }}

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Automatizar imágenes Docker

Añadir al final del fichero despliegue.yml (en el bloque deploy)

- name: Build Docker image
run: docker build -t \${{ secrets.DOCKER_USERNAME }}/**hola-mundo-multilenguaje:\${{ github.sha }}** .
- name: Log in to Docker Hub
uses: docker/login-action@v2
with:
 username: \${{ secrets.DOCKER_USERNAME }}
 password: \${{ secrets.DOCKER_PASSWORD }}
- name: Push Docker image to Docker Hub
run: docker push \${{ secrets.DOCKER_USERNAME }}/**hola-mundo-multilenguaje:\${{ github.sha }}**

Nombre de la imagen

NOTA: `github.sha` es el identificador del *commit* que activa el pipeline de GitHub Actions. Es un identificador único de GitHub que resulta práctico para etiquetar la versión de imagen de Docker que se generó a partir de este *commit*. De esta forma se facilita la identificación y la trazabilidad.

4. Servidores de automatización



GitHub Actions

Ejemplo - Hola Mundo multilenguaje - Automatizar imágenes Docker

deploy

succeeded 1 minute ago in 45s

- > ✓ Set up job
- > ✓ Checkout code
- > ✓ Set up PHP
- > ✓ Install dependencies
- > ✓ Deploy to server
- > ✓ Build Docker image
- > ✓ Log in to Docker Hub
- > ✓ Push Docker image to Docker Hub
- > ✓ Post Log in to Docker Hub
- > ✓ Post Checkout code
- > ✓ Complete job

dockerhub Explore Repositories Organizations

rsanzfloridauni Search by repository name All Content Create repository

rsanzfloridauni / hola-mundo-multilenguaje	☆ 0	↓ 0	Public	Scout inactive
Contains: Image • Last pushed: less than a minute ago				
rsanzfloridauni / mi_app_cine	☆ 0	↓ 9	Public	Scout inactive
Contains: Image • Last pushed: 18 days ago				
rsanzfloridauni / mi_cine	☆ 0	↓ 10	Public	Scout inactive
Contains: Image • Last pushed: 18 days ago				
rsanzfloridauni / mi_imagen	☆ 0	↓ 6	Public	Scout inactive
Contains: Image • Last pushed: 18 days ago				

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker

Vamos a construir un **workflow (o pipeline) nuevo** basado solamente en Docker, es decir, no hay que instalar Apache, PHP, MySQL,... en el servidor, solo hay que instalar Docker y Docker Compose y asegurarse de que los puertos SSH (22) y de la aplicación (p.ej. 8080, 8081,...) estén abiertos.

Pasos:

- En la máquina remota: instalar Docker y Docker Compose y abrir puerto SSH.
- En el repositorio de GitHub: crear los secretos correspondientes.
- En local:
 - Desarrollar la aplicación: en este caso basada en PHP y MySQL.
 - Crear un Dockerfile personalizado para generar la imagen de la aplicación.
 - Crear un Dockerfile personalizado para generar la imagen de la base de datos.
 - Crear un fichero Docker Compose para orquestar los servicios.
 - Crear el fichero `.github/workflows/deploy.yml`.

NOTA: Por simplicidad vamos a prescindir en este ejemplo de la parte de testeo automático.

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en la máquina remota

Pasos:

- En la máquina remota: instalar Docker y Docker Compose (igual que en el tema de Contenedores) y abrir puertos SSH, 8080 y 8081 (revisar la guía de AWS disponible en el F.O.).

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64]
```

```
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
sudo apt-get update
```

```
sudo apt-get install docker-ce
```

```
sudo systemctl status docker
```


4. Servidores de automatización









GitHub Actions

Automatizar despliegue con Docker - Trabajo en GitHub

Pasos:

- En el repositorio de GitHub: crear los secretos correspondientes:

Name 
 DOCKER_PASSWORD
 DOCKER_USERNAME
 REMOTE_HOST
 REMOTE_KEY
 REMOTE_USER

Contraseña de tu cuenta de DockerHub

Usuario de tu cuenta de DockerHub

IP de tu instancia EC2 (recuerda que si no la configuras como elástica -fija-, cambiará)

Tu clave PEM (debes copiarla/pegarla exactamente el contenido del fichero)

Usuario de la máquina Ubuntu de EC2 (habitualmente es “ubuntu” por defecto)

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en local

Del tema anterior, Dockerfile para la aplicación y Dockerfile para la base de datos:

```
/docker_cine
```

```
| -- /mysql
```

```
|   | -- cine.sql
```

```
|   | -- Dockerfile
```

```
| -- /php
```

```
|   | -- Dockerfile
```

```
|   | -- index.php
```

```
| -- compose.yml
```

```
FROM mysql:latest
```

```
ENV MYSQL_ROOT_PASSWORD=contrasenaRoot
```

```
ENV MYSQL_DATABASE=cine
```

```
ENV MYSQL_USER=usuario1
```

```
ENV MYSQL_PASSWORD=contrasenaUsuario1
```

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

```
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
```

DAW - Despliegue de aplicaciones web

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en local

Del tema anterior, Docker Compose (compose.yml) para orquestar los servicios:

```
services:
  db:
    image: rsanzfloridauni/mi_cine:v1
    container_name: contenedorMiCine
    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
```

```
name: Deploy to Remote Server
```

```
on:
```

```
  push:
```

```
    branches:
      - main
```

```
jobs:
```

```
  build-and-deploy:
```

```
    runs-on: ubuntu-latest
```

```
  steps:
```

```
    # Clonar el repositorio
```

```
    - name: Checkout code
      uses: actions/checkout@v3
```

```
    # Iniciar sesión en DockerHub
```

```
    - name: Log in to DockerHub
      uses: docker/login-action@v2
      with:
        username: ${ secrets.DOCKER_USERNAME }
        password: ${ secrets.DOCKER_PASSWORD }
```

```
    # Construir y subir la imagen de la base de datos
```

```
    - name: Build and push DB Docker image
      run: |
        cd mysql
        docker build -t ${ secrets.DOCKER_USERNAME }}/mi_cine:v1 .
        docker push ${ secrets.DOCKER_USERNAME }}/mi_cine:v1
```

```
    # Construir y subir la imagen de la aplicación
```

```
    - name: Build and push PHP Docker image
      run: |
        cd php
        docker build -t ${ secrets.DOCKER_USERNAME }}/mi_app_cine:v1 .
        docker push ${ secrets.DOCKER_USERNAME }}/mi_app_cine:v1
```







```
    # Transferir el archivo compose.yml al servidor remoto
```

```
    - name: Transfer compose.yml to remote server
      uses: appleboy/scp-action@master
      with:
        host: ${ secrets.REMOTE_HOST }
        username: ${ secrets.REMOTE_USER }
        key: ${ secrets.REMOTE_KEY }
        source: ./compose.yml
        target: ~/deploy/
```

```
    # Desplegar la aplicación en el servidor remoto usando docker-compose
```

```
    - name: Deploy with Docker Compose
      uses: appleboy/ssh-action@master
      with:
        host: ${ secrets.REMOTE_HOST }
        username: ${ secrets.REMOTE_USER }
        key: ${ secrets.REMOTE_KEY }
        script: |
          cd ~/deploy
          sudo chmod 666 /var/run/docker.sock
          docker compose down || true
          docker compose pull
          docker compose up -d
```

Fichero .github/workflows/deploy.yml

Name 
 DOCKER_PASSWORD
 DOCKER_USERNAME
 REMOTE_HOST
 REMOTE_KEY
 REMOTE_USER

Recuerda definir correctamente los secretos en GitHub

name: Deploy to Remote Server

Define el nombre del flujo de trabajo como "Deploy to Remote Server".

on:

push:

branches:
- main

Especifica que el flujo de trabajo se activará automáticamente cuando haya un push a la rama main (o master, si fuera el caso) del repositorio.

jobs:

build-and-deploy:

runs-on: ubuntu-latest

Define un trabajo llamado build-and-deploy que se ejecutará como parte del flujo de trabajo.

Especifica que este trabajo se ejecutará en un entorno virtual basado en Ubuntu (el más reciente disponible).

steps:

Clonar el repositorio

- name: Checkout code
uses: actions/checkout@v3

Clona el código del repositorio actual en el entorno de trabajo.

Iniciar sesión en DockerHub

- name: Log in to DockerHub
uses: docker/login-action@v2
with:
username: \${ secrets.DOCKER_USERNAME }
password: \${ secrets.DOCKER_PASSWORD }

Inicia sesión en DockerHub usando las credenciales almacenadas como secretos (DOCKER_USERNAME y DOCKER_PASSWORD).

Construir y subir la imagen de la base de datos

- name: Build and push DB Docker image
run: |
cd mysql
docker build -t \${ secrets.DOCKER_USERNAME }/mi_cine:v1 .
docker push \${ secrets.DOCKER_USERNAME }/mi_cine:v1

Construye una imagen Docker de la base de datos desde el directorio mysql y la sube a DockerHub con el nombre mi_cine:v1.

Construir y subir la imagen de la aplicación

- name: Build and push PHP Docker image
run: |
cd php
docker build -t \${ secrets.DOCKER_USERNAME }/mi_app_cine:v1 .
docker push \${ secrets.DOCKER_USERNAME }/mi_app_cine:v1

Construye una imagen Docker de la aplicación PHP desde el directorio php y la sube a DockerHub con el nombre mi_app_cine:v1.

Transferir el archivo compose.yml al servidor remoto

- name: Transfer compose.yml to remote server
uses: appleboy/scp-action@master
with:
host: \${ secrets.REMOTE_HOST }
username: \${ secrets.REMOTE_USER }
key: \${ secrets.REMOTE_KEY }
source: ./compose.yml
target: ~/deploy/

Usa la acción scp-action para transferir el archivo compose.yml al servidor remoto EC2.

- Se conecta al servidor utilizando las credenciales (REMOTE_HOST, REMOTE_USER y REMOTE_KEY).
- Copia el archivo desde la máquina local a la ubicación remota ~/deploy/.

Desplegar la aplicación en el servidor remoto usando docker-compose

- name: Deploy with Docker Compose
uses: appleboy/ssh-action@master
with:
host: \${ secrets.REMOTE_HOST }
username: \${ secrets.REMOTE_USER }
key: \${ secrets.REMOTE_KEY }
script: |
cd ~/deploy
sudo chmod 666 /var/run/docker.sock
docker compose down || true
docker compose pull
docker compose up -d

Usa la acción ssh-action para ejecutar comandos remotos en el servidor:

- Navega al directorio ~/deploy.
- Cambia permisos del socket de Docker (/var/run/docker.sock) para evitar problemas de permisos.
- Baja los contenedores existentes si están corriendo (docker compose down).
- Descarga las últimas versiones de las imágenes (docker compose pull).
- Levanta los contenedores en modo desacoplado (docker compose up -d).

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en local ↔ GitHub

Queda crear el repositorio de Git local (si no lo has hecho ya), vincularlo al remoto y realizar la acción de push.

```
git init
```

```
git remote add origin https://github.com/tuGitHub/nombreDelRepo.git
```

```
git add .
```

```
git commit -m "Primer commit"
```

```
git push -u origin main
```

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en GitHub

Al realizar la acción de push desde el repositorio local se desencadena la acción en GitHub Actions. Y si todo funciona correctamente...

build-and-deploy
succeeded 6 minutes ago in 1m 27s

Search logs

> ✓ Set up job

> ✓ Build appleboy/scp-action@master

> ✓ Checkout code

> ✓ Log in to DockerHub

> ✓ Build and push DB Docker image

> ✓ Build and push PHP Docker image

> ✓ Transfer compose.yml to remote server

> ✓ Deploy with Docker Compose

> ✓ Post Log in to DockerHub

> ✓ Post Checkout code

> ✓ Complete job

1s

7s

0s

0s

13s

18s

2s

45s

0s

0s

0s

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

4. Servidores de automatización









GitHub Actions

Automatizar despliegue con Docker - Trabajo en local

Y si accedemos desde el navegador a la URL:


Browser address bar: 35.153.231.210:8080/mi_app_cine/

Listado de Películas

ID	Título	Director	Nota	Año	Presupuesto	Imagen (Base64)	URL del Trailer
1	Star Wars The Force Awakens	JJ Abrams	6.7	2015	552		Ver Trailer
2	Jurassic World Fallen Kingdom	JA Bayona	5.6	2018	503		Ver Trailer
3	Pirates of the Caribbean On Stranger Tides	Rob Marshall	5.4	2011	492		Ver Trailer
4	Star Wars The Rise of Skywalker	JJ Abrams	5.6	2019	476		Ver Trailer
5	Avengers Age Of Ultron	Joss Whedon	6.3	2015	451		Ver Trailer
6	Pirates of the Caribbean At Worlds End	Gore Verbinski	6.1	2007	423		Ver Trailer

Recuerda que los puertos **8080** y **8081** (o los que quieras utilizar) deben estar abiertos en la configuración de seguridad de la instancia EC2.

Browser address bar: 35.153.231.210:8081



Bienvenido a phpMyAdmin

Idioma (Language)

Español - Spanish

Iniciar sesión

Usuario:

Contraseña:

Iniciar sesión

a

4. Servidores de automatización



GitHub Actions

Automatizar despliegue con Docker - Trabajo en local ↔ máquina remota

Puedes acceder desde local vía SSH (PuTTY) al servidor EC2 de AWS y comprobar que las imágenes y contenedores funcionan correctamente en la máquina remota:

```
ubuntu@ip-172-31-52-235:~/deploy$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rsanzfloridauni/mi_app_cine	v1	3ebc375a8428	About a minute ago	501MB
rsanzfloridauni/mi_cine	v1	3dfd68659c3e	2 minutes ago	604MB
rsanzfloridauni/angular-suma-app	latest	d4d7ffb86093	2 months ago	47.3MB
phpmyadmin/phpmyadmin	latest	933569f3a9f6	17 months ago	562MB

```
ubuntu@ip-172-31-52-235:~/deploy$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
77b0975fbb5	phpmyadmin/phpmyadmin:latest	"/docker-entrypoint..."	9 minutes ago	Up 9 minutes	0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
5ca08b44741a	rsanzfloridauni/mi_app_cine:v1	"docker-php-entrypoi..."	9 minutes ago	Up 9 minutes	0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
69097db0ba29	rsanzfloridauni/mi_cine:v1	"docker-entrypoint.s..."	9 minutes ago	Up 9 minutes	0.0.0.0:3306->3306/tcp, ::3306->3306/tcp

DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

5. Conclusiones

- La integración y despliegue continuos (CI/CD) permiten optimizar la entrega de valor en la cadena de desarrollo y producción de software, automatizando flujos de trabajo que incluyen tareas de desarrollo y operaciones.
- Las partes fundamentales en CI/CD son:
 - Repositorios de código con control de versiones (Git/GitHub)
 - Compilación (*build*) automática (*bundles*/Maven/Make)
 - Implementación de pruebas automatizadas (PHPUnit)
 - Uso de herramientas de automatización (GitHub Actions)
 - Uso de entornos de desarrollo, pruebas y producción (local/MVs/AWS)

6. Anexo: *pipeline* CI/CD para Angular



Vamos a recuperar el ejemplo de la aplicación suma del tema anterior...



El objetivo es replicar el *pipeline/workflow* de GitHub Actions que hemos hecho para la aplicación PHP, pero ahora para una aplicación cliente desarrollada con Angular. Vamos a ver dos *frameworks* de testeo basados en Javascript muy utilizados: Jasmine y JEST.



[Jasmine](#) es el *framework* que lleva incorporado Angular [por defecto](#) cuando se instala Angular CLI. Es para Javascript, con lo que también se puede utilizar en React u otros *frameworks* de desarrollo cliente de Javascript.



[JEST](#) es un *framework* de testeo muy utilizado en Javascript (funciona en proyectos de Angular, Babel, TypeScript, Node, React, Vue, etc.). Hay que instalarlo aparte.

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local

En el tema anterior se había implementado el componente con los ficheros:

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

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

```
src/app/app.component.ts
```

```
src/app/app.component.html
```

Para implementar las pruebas hay que editar el fichero `src/app/suma.component.spec.ts` incluyendo las pruebas del componente. En este caso van a ser dos pruebas muy sencillas: la primera es para ver que la suma funciona correctamente y la segunda para ver que el resultado se renderiza en la página correctamente.

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { SumaComponent } from './suma.component';

describe('SumaComponent', () => {
  let component: SumaComponent;
  let fixture: ComponentFixture<SumaComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [SumaComponent]
    })
    .compileComponents();
    fixture = TestBed.createComponent(SumaComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
  it('debe crear', () => {
    expect(component).toBeTruthy();
  });
  it('debe sumar correctamente', () => {
    component.numero1 = 5;
    component.numero2 = 10;
    component.calcularSuma();
    expect(component.resultado).toBe(15);
  });
  it('debe actualizar el resultado en la plantilla HTML', () => {
    component.numero1 = 3;
    component.numero2 = 7;
    component.calcularSuma();
    fixture.detectChanges(); // Trigger change detection
    const compiled = fixture.nativeElement;
    const resultParagraph = compiled.querySelector('p');
    expect(resultParagraph.textContent).toContain('Resultado: 10');
  });
});
```

suma.component.spec.ts

```

import { ComponentFixture, TestBed } from '@angular/core/testing';
import { SumaComponent } from './suma.component'; Importa componente a testear

describe('SumaComponent', () => { Descripción del bloque de pruebas para SumaComponent
  let component: SumaComponent; Declara variable component (para facilitar y reutilizar código)
  let fixture: ComponentFixture<SumaComponent>; Variable para manejar el componente en el entorno de pruebas
  beforeEach(async () => { Función que inicializa el entorno de pruebas para cada prueba
    await TestBed.configureTestingModule({
      imports: [SumaComponent]
    })
    .compileComponents();
    fixture = TestBed.createComponent(SumaComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
  it('debe crear', () => { Definición de una prueba: comprobar que el componente se ha creado y no es nulo
    expect(component).toBeTruthy();
  });
  it('debe sumar correctamente', () => { Definición de una prueba: comprobar que la suma se realiza correctamente, definiendo dos números y llamando al método "calcularSuma()" del componente. Se utilizan los nombres de los atributos del componente (numero1, numero2 y resultado).
    component.numero1 = 5;
    component.numero2 = 10;
    component.calcularSuma();
    expect(component.resultado).toBe(15);
  });
  it('debe actualizar el resultado en la plantilla HTML', () => {
    component.numero1 = 3;
    component.numero2 = 7;
    component.calcularSuma();
    fixture.detectChanges(); // Trigger change detection
    const compiled = fixture.nativeElement;
    const resultParagraph = compiled.querySelector('p');
    expect(resultParagraph.textContent).toContain('Resultado: 10');
  });
});

```

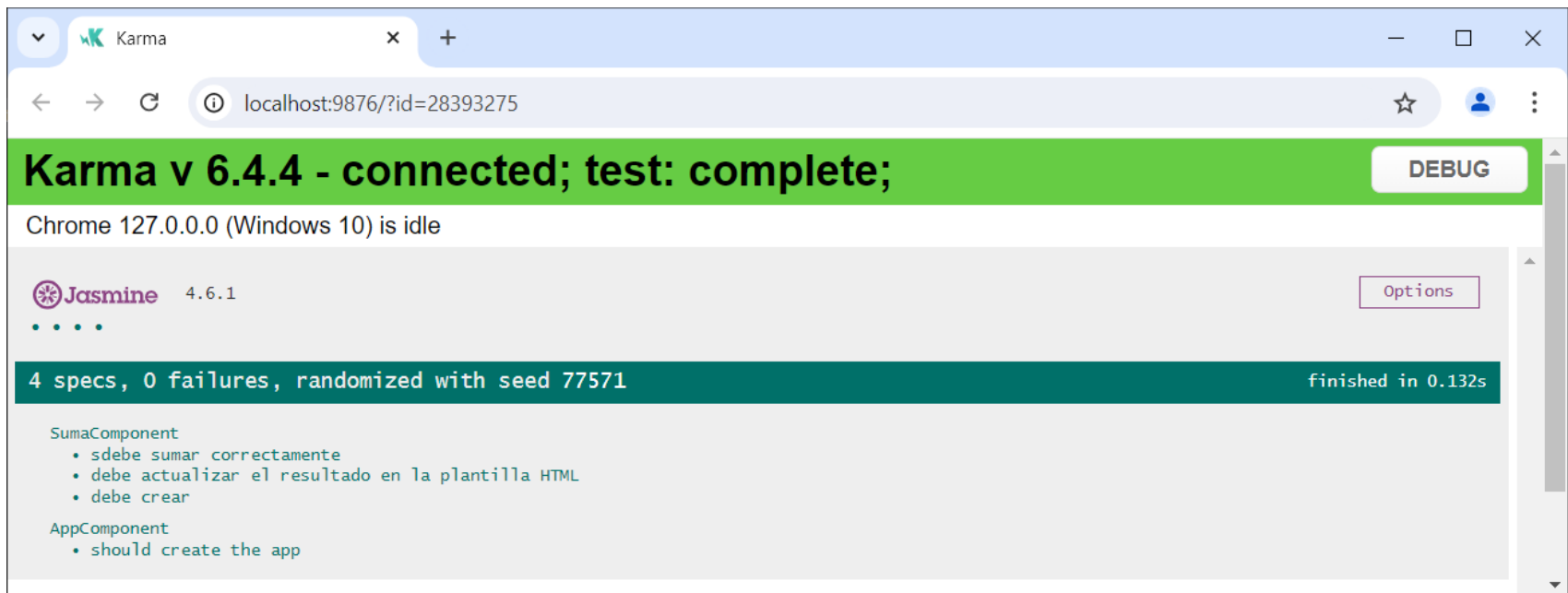
Definición de una prueba: comprobar que el resultado de la suma se actualiza correctamente en el DOM (Document Object Modelo) o página web HTML. `fixture.detectChanges()` detecta los cambios y actualiza la vista, `fixture.nativeElement` obtiene el elemento DOM que ha cambiado, concretamente se trata de un elemento de tipo `<p>` (ver `suma.component.html`). Finalmente se verifica que el elemento `<p>` tiene como contenido "Resultado 10" para la suma de los números de esta prueba.

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con Jasmine/Karma

Ejecutar el test para ver que funcione correctamente desde el terminal: `npm test`



Ejemplo de test correcto

NOTA: El *framework* Jasmine lleva incorporado un servidor web llamado [Karma](#) para visualizar el resultado de las pruebas. El servidor se ejecuta tras realizar la prueba.

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con Jasmine/Karma

Ejecutar el test para ver que funcione correctamente desde el terminal: `npm test`

The screenshot shows the Karma test runner interface in a web browser. At the top, it says 'Karma v 6.4.4 - connected; test: complete;'. Below that, it says 'Chrome 127.0.0.0 (Windows 10) is idle'. The Jasmine logo and version '4.6.1' are shown. A summary bar indicates '4 specs, 1 failure, randomized with seed 51908' and 'finished in 0.121s'. Below this, there are links for 'Spec List' and 'Failures'. The failure details for 'SumaComponent' are shown: 'debe actualizar el resultado en la plantilla HTML'. The error message is 'Expected 'Resultado: 10' to contain 'Resultado: 9''. The stack trace shows the error occurred in 'UserContext.apply' at 'src/app/suma/suma.component.spec.ts:32:41'. A red arrow points from the error message to a text box on the right.

Ejemplo de test incorrecto (se ha forzado que el Resultado sea 9 en vez de 10)

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con JEST

Dado que JEST no viene incorporado por defecto en Angular, hay que instalarlo y realizar algunas tareas adicionales de configuración en el proyecto.

1. Instalar dependencias:

```
npm install --save-dev jest jest-preset-angular @types/jest ts-jest
```

2. Añadir al fichero package.json la configuración de JEST:

```
"jest": {  
  "preset": "jest-preset-angular",  
  "setupFilesAfterEnv": ["<rootDir>/setup-jest.ts"],  
  "testPathIgnorePatterns": [  
    "<rootDir>/node_modules/",  
    "<rootDir>/dist/",  
    "<rootDir>/e2e/"  
  ],  
  "globals": {  
    "ts-jest": {  
      "tsconfig": "<rootDir>/tsconfig.spec.json",  
      "stringifyContentPathRegex": "\\\\.html$",  
      "isolatedModules": true  
    }  
  },  
  "moduleNameMapper": {  
    "^src/(.*)$": "<rootDir>/src/$1"  
  }  
}
```

DAW - Despliegue de aplic

5. Integración y despliegue continuo

da

tiu

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con JEST

3. En la raíz del proyecto, crear el fichero `setup-test.js` con el siguiente contenido:

```
import 'jest-preset-angular/setup-jest';
```

4. Modificar el fichero `tsconfig.spec.json` para que incluya el compilador JEST/Node:

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "outDir": "./out-tsc/spec",
    "types": ["jest", "node"]
  },
  "files": ["src/test.ts"],
  "include": ["src/**/*.spec.ts", "src/**/*.d.ts"]
}
```

5. Modifica la entrada “test” del atributo “scripts” de `package.json` para que sea “jest”:

```
"scripts": {
  "test": "jest",
  ...
}
```

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con JEST

6. Ejecutar el test: `npm test`

```
PASS src/app/app.component.spec.ts (6.876 s)
PASS src/app/suma/suma.component.spec.ts (6.923 s)

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        10.107 s
Ran all test suites.
```

Y si forzamos que haya un error... ➔

NOTA: A diferencia del *framework* Jasmine, que lleva incorporado el servidor web Karma para visualizar el resultado de las pruebas, con JEST toda la información de las pruebas aparece en el terminal.

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en local - Testeo con JEST

```
PASS src/app/app.component.spec.ts
FAIL src/app/suma/suma.component.spec.ts
  ● SumaComponent > debe actualizar el resultado en la plantilla HTML

    expect(received).toContain(expected) // indexOf

    Expected substring: "Resultado: 9"
    Received string:    "Resultado: 10"

      30 |     const compiled = fixture.nativeElement;
      31 |     const resultParagraph = compiled.querySelector('p');
    > 32 |     expect(resultParagraph.textContent).toContain('Resultado: 9');
    > 32 |     expect(resultParagraph.textContent).toContain('Resultado: 9');
        |                                         ^
      33 |   });
      34 | });
      35 |

      at src/app/suma/suma.component.spec.ts:32:41
      at _ZoneDelegate.invoke (node_modules/zone.js/bundles/zone.umd.js:416:32)
      at ProxyZoneSpec.Object.<anonymous>.ProxyZoneSpec.onInvoke (node_modules/zone.js/bundles/zone-testing.umd.js:2164:43)
      at _ZoneDelegate.invoke (node_modules/zone.js/bundles/zone.umd.js:415:38)
      at ZoneImpl.run (node_modules/zone.js/bundles/zone.umd.js:147:47)
      at Object.wrappedFunc (node_modules/zone.js/bundles/zone-testing.umd.js:450:38)

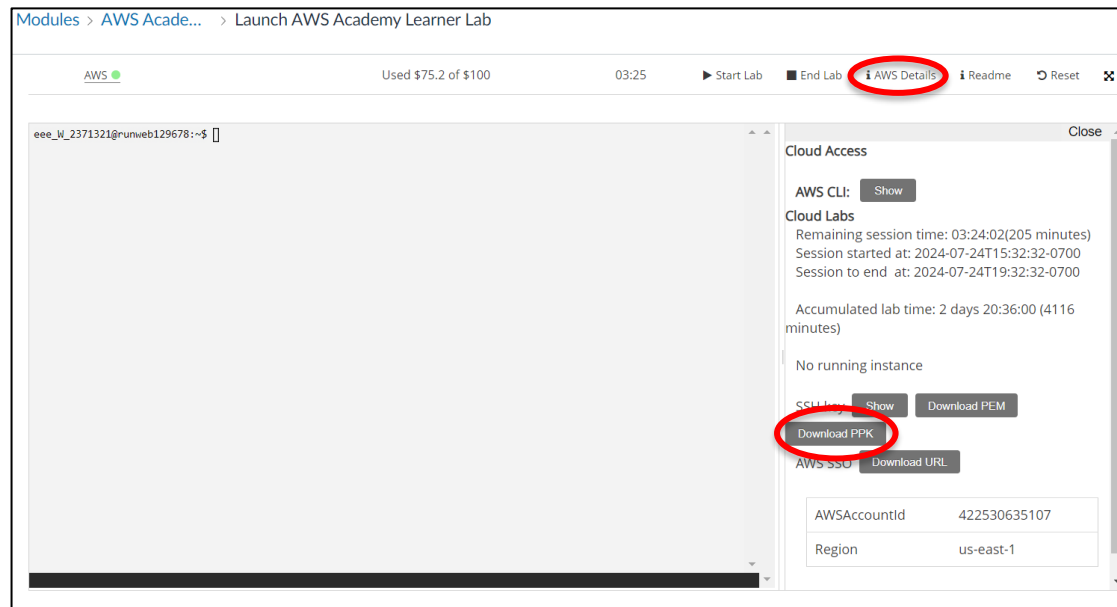
Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 3 passed, 4 total
Snapshots:   0 total
Time:        4.578 s, estimated 7 s
Ran all test suites.
```

6. Anexo: *pipeline* CI/CD para Angular



Preparación del servidor en AWS

1. Si no la has creado con anterioridad, crea una instancia EC2 (máquina virtual) de AWS con Ubuntu (puedes seguir la guía que hay disponible en Florida Oberta).
2. Desde la página de AWS Academy Learner Lab, en AWS Details, descarga la clave SSH con formato PPK (la necesitarás para acceder por PuTTY → ver tema 2 ←).



DAW - Despliegue de aplicaciones web

5. Integración y despliegue continuo

6. Anexo: *pipeline* CI/CD para Angular



Preparación del servidor en AWS

3. Accede por SSH a la instancia EC2 (ver **tema 2**)
4. Actualiza la lista de paquetes: `sudo apt update`
5. Instala Nginx: `sudo apt install nginx`
6. Comprueba que funciona poniendo en el navegador: http://IP_EC2_AWS
7. El directorio por defecto donde se alojan las aplicaciones es `/var/www/html`. Vamos a crear un directorio específico para nuestro proyecto `/var/www/html/suma-angular`.
Accede a `./html` y crea el directorio: `sudo mkdir suma-angular`

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en GitHub

1. Crea un repositorio público en GitHub (es necesario que sea público para el plan gratuito de GitHub Actions). En este ejemplo lo he llamado suma-angular.
2. En “Settings”, accede a “Secrets and variables → Actions”. Este apartado permite almacenar de manera segura variables con información sensible, como nombres de usuario, contraseñas, IPs, etc. Dentro de “Repository secrets”, vamos a crear los siguientes:

Name: DEPLOY_PATH **Secret:** ruta de despliegue (p.ej. /var/www/html/suma-app)

Name: HOST **Secret:** IP de la instancia EC2 de AWS*

Name: SSH_PRIVATE_KEY **Secret:** contenido de la clave .pem de AWS**

Name: USERNAME **Secret:** nombre de usuario (p.ej. ubuntu)

* Ten en cuenta que si no configuras la IP pública de tu EC2 como elástica (“fija”), cada vez que inicies la EC2 cambiará, con lo que tendrás que cambiar también este secreto. Por tanto, es aconsejable configurar siempre tu máquina con IP elástica para evitar errores.

** No confundir la clave .pem con la clave .ppk, que se utiliza solo para PuTTY.

6. Anexo: *pipeline* CI/CD para Angular



Trabajo en Local ↔ GitHub

1. Crea un fichero `.gitignore` para que no se sincronicen dependencias y otros ficheros que no es necesario tener en GitHub (ver diapositiva siguiente).
2. En el directorio local del proyecto crea el fichero YAML (crea también la ruta) `./.github/workflows/despliegue.yml` (ver diapositiva siguiente).
3. Iniciar el repositorio: `git init`
4. Añadir el remoto: `git remote add origin https://github.com/rsanzfloridauni/suma-angular.git`
5. Añade todos los ficheros para preparar el commit: `git add .`
6. Haz el *commit*: `git commit -m "Primer commit"`
7. Haz el *push* a GitHub: `git push -u origin main`

Si el *push* se realiza correctamente, se enviará todo al repositorio remoto y automáticamente se desencadenará la “Action” programada `despliegue.yml`. En GitHub, accede a la pestaña “Actions” del repositorio. Si lo haces inmediatamente después de hacer el *push*, te dará a tiempo a ver que la “Action” está aún en proceso.

.gitignore

```
# Compiled output
/dist
/tmp
/out-tsc
/bazel-out

# Node
/node_modules
npm-debug.log
yarn-error.log

# IDEs and editors
.idea/
.project
.classpath
.c9/
*.launch
.settings/
*.sublime-workspace

# Visual Studio Code
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json
.history/*

# Miscellaneous
/.angular/cache
.sass-cache/
/connect.lock
/coverage
/libpeerconnection.log
testem.log
/typings

# System files
.DS_Store
Thumbs.db
```

despliegue.yml

```
name: Deploy to Server
on:
  push:
    branches:
      - main # o cualquier rama en la que desees activar este pipeline
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '20' # version LTS

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test

      - name: Build Angular application
        run: npm run build

      - name: Deploy to server
        env:
          SSH_PRIVATE_KEY: ${ secrets.SSH_PRIVATE_KEY }
          HOST: ${ secrets.HOST }
          USERNAME: ${ secrets.USERNAME }
          DEPLOY_PATH: ${ secrets.DEPLOY_PATH }
        run: |
          echo "Deploying to server..."
          mkdir -p ~/.ssh
          echo "$SSH_PRIVATE_KEY" | tr -d '\r' > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa
          ssh -o StrictHostKeyChecking=no $USERNAME@$HOST "mkdir -p $DEPLOY_PATH"
          ssh -i ~/.ssh/id_rsa $DEPLOY_USER@$DEPLOY_SERVER "rm -rf $DEPLOY_PATH/*"
          rsync -avz --delete ./dist/suma-app/browser/ $USERNAME@$HOST:$DEPLOY_PATH
          ssh $USERNAME@$HOST 'sudo systemctl restart nginx'
```

6. Anexo: *pipeline* CI/CD para Angular

GitHub Action finalizada y despliegue realizado



Summary

Jobs

- build-and-deploy

Run details

- Usage
- Workflow file

build-and-deploy
succeeded 3 minutes ago in 32s

Search logs

- Set up job 0s
- Checkout repository 1s
- Set up Node.js 0s
- Install dependencies 11s
- Run tests 4s
- Build Angular application 11s
- Deploy to server 2s
- Post Set up Node.js 0s
- Post Checkout repository 0s
- Complete job 0s

Despliegue correcto

PERO...

SumaApp

No es seguro 3.89.244.7/suma-app/

Suma de dos números

2 1 Sumar

Resultado: 0

El botón "Sumar" no funciona

Error al cargar recursos

DevTools is now available in Spanish! Always match Chrome's language Switch DevTools to Spanish Don't show again

Elements Console Sources Network Performance Memory Application >>

Styles Computed Layout >>

Failed to load resource: the server responded with a status of 404 (Not Found) main-ZUU4QFZ2.js:1

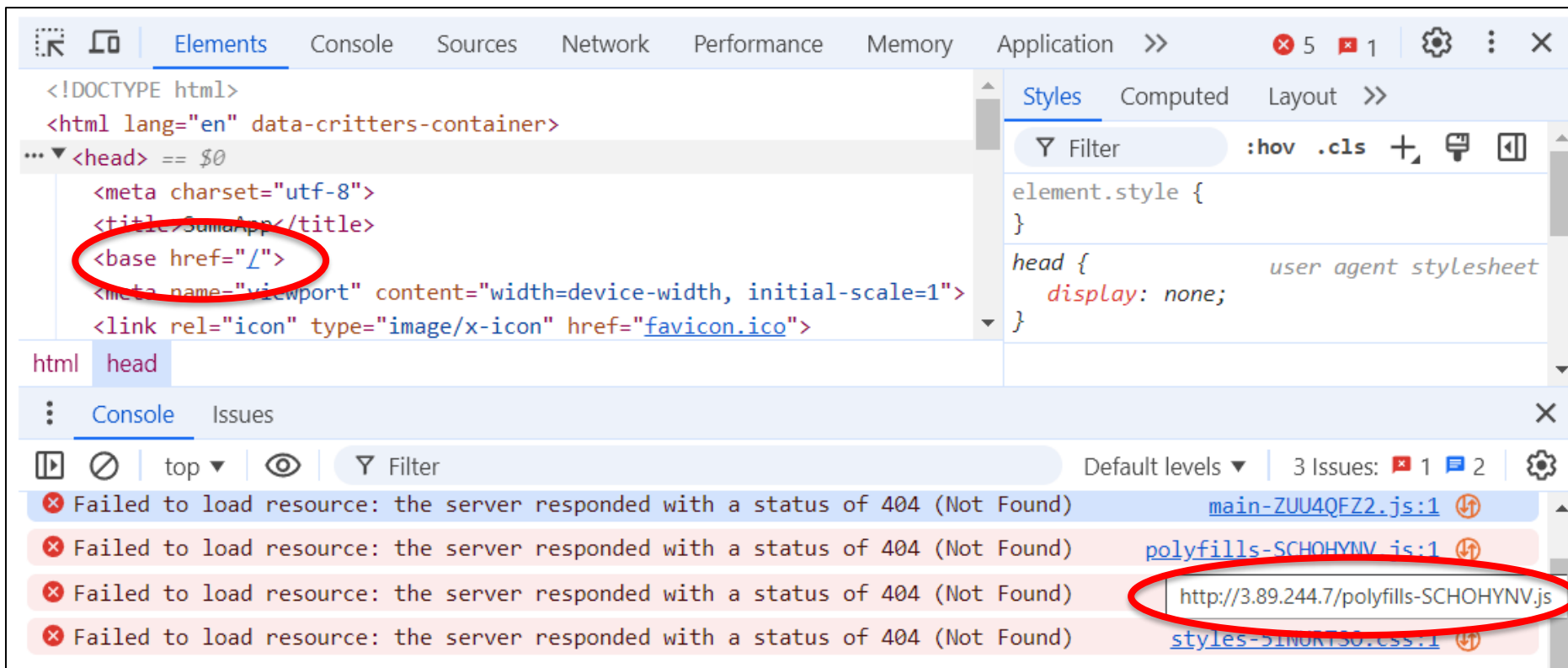
Failed to load resource: the server responded with a status of 404 (Not Found) polyfills-SCHOHYWV.js:1

Failed to load resource: the server responded with a status of 404 (Not Found) favicon.ico:1

Failed to load resource: the server responded with a status of 404 (Not Found) styles-5INURTS0.css:1

6. Anexo: *pipeline* CI/CD para Angular

GitHub Action finalizada y despliegue realizado



Hemos desplegado nuestra aplicación en <http://IP/suma-app>, pero está buscando los recursos en <http://IP/>. El fichero index.html que crea Angular por defecto no tiene en cuenta que se puede desplegar la aplicación en un subdirectorio del servidor, por lo que hay que indicarlo manualmente. Basta editar el campo <base href="/"> y cambiarlo por <base href="/suma-app/"> (o el que hayas definido en la ruta de despliegue). Este campo href debe coincidir con el subdirectorio del servidor Nginx (o Apache) (ruta /var/www/html/**subdirectorio**).

6. Anexo: *pipeline* CI/CD para Angular



GitHub Action finalizada y despliegue realizado

The screenshot shows the GitHub Actions interface for a workflow named 'build-and-deploy'. The workflow has succeeded 3 minutes ago in 32s. The left sidebar shows the 'Summary' tab selected, with 'Jobs' listed below it, including 'build-and-deploy' which is marked as successful. The main area shows the 'Run details' for the 'build-and-deploy' job, listing the steps and their durations: 'Set up job' (0s), 'Checkout repository' (1s), 'Set up Node.js' (0s), 'Install dependencies' (11s), 'Run tests' (4s), 'Build Angular application' (11s), 'Deploy to server' (2s), 'Post Set up Node.js' (0s), 'Post Checkout repository' (0s), and 'Complete job' (0s).

Step	Duration
Set up job	0s
Checkout repository	1s
Set up Node.js	0s
Install dependencies	11s
Run tests	4s
Build Angular application	11s
Deploy to server	2s
Post Set up Node.js	0s
Post Checkout repository	0s
Complete job	0s

The screenshot shows a web browser window with the title 'SumaApp'. The address bar shows the URL '3.89.244.7/suma-app/'. The page content is titled 'Suma de dos números'. It features two input fields: the first contains '1324' and the second contains '1134'. To the right of these fields is a button labeled 'Sumar'. Below the input fields, the text 'Resultado: 2458' is displayed.

Suma de dos números

1324	1134	Sumar
------	------	-------

Resultado: 2458

Tras editar el fichero `index.html` del proyecto Angular compilado (en la carpeta `dist` del proyecto) y desplegar de nuevo, ya funciona correctamente.

6. Anexo: *pipeline* CI/CD para Angular



Sugerencia...

Por último, puedes completar el *pipeline* creando de manera automática una imagen personalizada de Docker con tu aplicación y subiéndola a tu cuenta de Docker Hub, tal y como hemos hecho para el caso de la aplicación Hola Mundo Multilenguaje de PHP.

Puedes partir del fichero Dockerfile para una aplicación Angular que desarrollamos en el tema anterior.

