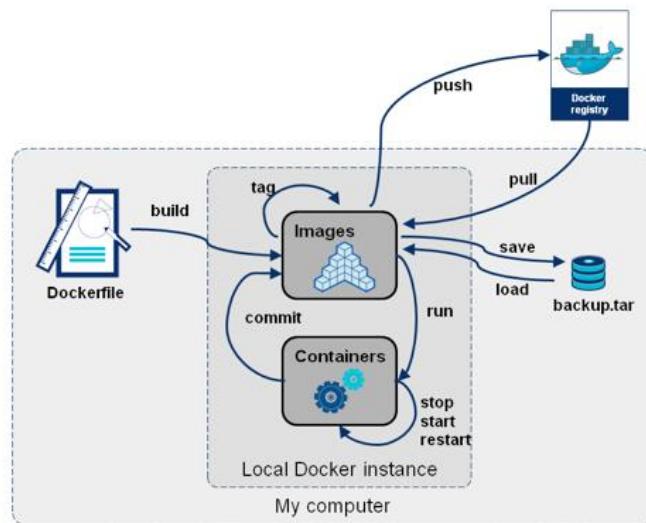


Actividades Docker. Creación de imágenes personalizadas

1. Apuntes previos sobre gestión de imágenes personalizadas



- Opción 1: descargar imagen genérica → crear contenedor → instalación de servicios y copia de ficheros → se debe hacer docker commit del contenedor a una imagen
- Opción 2: uso de ficheros Dockerfile. Se edita el Dockerfile donde se agregan archivos y se lanzan comandos → se crea la imagen nueva con docker build

El fichero Dockerfile

Un fichero Dockerfile es un conjunto de instrucciones que serán ejecutadas de forma secuencial para construir una nueva imagen docker

Hay varias instrucciones que podemos usar en la construcción de un Dockerfile, pero la estructura fundamental del fichero es:

- indicamos imagen base: FROM
- Instrucciones de construcción: RUN, COPY, ADD, WORKDIR
- Configuración: variables de entorno, usuarios, puertos: USER, EXPOSE, ENV
- Instrucciones de arranque: CMD, ENTRYPOINT

Detalle de las opciones del fichero

- FROM:** Sirve para especificar la imagen sobre la que voy a construir la mía.
Ejemplo: FROM php:7.4-apache

- **LABEL:** Sirve para añadir metadatos a la imagen mediante clave=valor.
Ejemplo: LABEL company=naranco
- **COPY:** Para copiar ficheros desde mi equipo a la imagen. Esos ficheros deben estar en el mismo contexto (carpeta o repositorio). Su sintaxis es:
`COPY [--chown=<usuario>:<grupo>] origen destino`
Ejemplo: COPY --chown=www-data:www-data myapp /var/www/html
- **ADD:** Es similar a COPY pero tiene funcionalidades adicionales como especificar URLs y tratar archivos comprimidos.
- **RUN:** Ejecuta una orden en la imagen. Su sintaxis es
`RUN orden ó RUN ["orden", "param1", "param2"]`
Ejemplo: RUN apt update && apt install -y git. En este caso es muy importante que pongamos la opción -y porque en el proceso de construcción no puede haber interacción con el usuario.
- **WORKDIR:** Establece el directorio de trabajo dentro de la imagen que estoy creando para posteriormente usar las órdenes RUN,COPY,ADD,CMD o ENTRYPOINT
Ejemplo: WORKDIR /usr/local/apache/htdocs
- **EXPOSE:** Nos da información acerca de qué puertos tendrá abiertos el contenedor cuando se cree uno en base a la imagen que estamos creando. Es meramente informativo.
Ejemplo: EXPOSE 80
- **USER:** Para especificar (por nombre o UID/GID) el usuario de trabajo para todas las órdenes RUN,CMD Y ENTRYPOINT posteriores.
Ejemplos: USER alumno ó USER 1001:10001
- **ENV:** Para establecer variables de entorno dentro del contenedor. Puede ser usado posteriormente en las órdenes RUN añadiendo \$ delante del nombre de la variable de entorno.
Ejemplo: ENV WEB_DOCUMENT_ROOT=/var/www/html
No se puede usar con ENTRYPOINT Y CMD.
- **ENTRYPOINT:** Para establecer el ejecutable que se lanza siempre cuando se crea el contenedor con docker run, salvo que se especifique expresamente algo diferente con el flag --entrypoint.
Su sintaxis es la siguiente:
`ENTRYPOINT <command> / ENTRYPOINT
["executable", "param1", "param2"]`
Ejemplo: ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
- **CMD:** Para establecer el ejecutable por defecto (salvo que se sobrescriba desde la orden docker run) o para especificar parámetros para un ENTRYPOINT. Si tengo varios sólo se ejecuta el último.
Su sintaxis es CMD param1 param2 / CMD ["param1", "param2"] ó CMD["command", "param1"].
Ejemplo: CMD ["-c" "/etc/nginx.conf"] / ENTRYPOINT ["nginx"]

2. **Creación de una imagen de Apache2.** Se pide crear una imagen a partir de este archivo Dockerfile:

```
FROM httpd:2.4
ADD public_html /usr/local/apache2/htdocs/
EXPOSE 80
```

Una vez creado se debe guardar en una carpeta junto con el directorio public_html (donde guardamos nuestro sitio web con los ficheros HTML necesarios)

A continuación lanzamos un nuevo contenedor mapeando el puerto 80 y verificamos desde un navegador que se carga el contenido web de nuestro contenedor

3. Realiza el mismo ejercicio anterior pero esta vez empleando una imagen de **nginx**.
NOTA: el directorio web de nginx es /usr/share/nginx/html
4. Realiza un Dockerfile para crear una imagen personalizada desde la imagen php:7.4-apache. En este caso el fichero Dockerfile debe copiar los scripts PHP a la carpeta /var/www/html.

De nuevo se debe crear la imagen, lanzar un contenedor desde la imagen y verificar desde el navegador el acceso al contenido

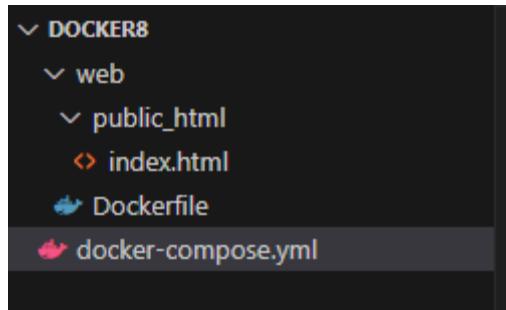
5. Finalmente vamos a realizar el mismo proceso pero en este caso desde una imagen de base de debían. Usaremos el siguiente archivo Dockerfile:

```
FROM debian
RUN apt-get update && apt-get install -y apache2 libapache2-mod-php7.3 php7.3 && apt-get clean && rm -rf /var/lib/apt/lists/*
RUN rm /var/www/html/index.html
ADD app /var/www/html/
EXPOSE 80
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Igualmente crearemos la nueva imagen, lanzamos un contenedor y verificamos el acceso desde el navegador

NOTA: en este caso el contenido PHP lo guardamos en la carpeta app

1.- Para el primer ejercicio, esta es la estructura.



El docker-compose.yml es el siguiente:

```
docker-compose.yml
services:
  web:
    build: ./web
    ports:
      - 8000:80
    volumes:
      - datos_web:/usr/local/apache2/htdocs
```

Y el Dockerfile con httpd:

```
Dockerfile
FROM httpd:2.4
ADD public_html /usr/local/apache2/htdocs/
EXPOSE 80
```

(No le saqué foto al resultado, pero funcionaba, ya que el primer ejercicio era sencillo)

2.- En el segundo ejercicio utilizaremos un contenedor de nginx, asi que cambiamos la ruta del volumen

```
  Dockerfile  X  index.html  docker-compose.yml  X
  docker-compose.yml
  ►Run All Services
  1 services:
  2   web:
  3     build: ./web
  4     ports:
  5       - 8000:80
  6     volumes:
  7       - datos_web:/usr/share/nginx/html
  8   volumes:
  9     datos_web:
```

Este sería el Dockerfile:

```
  Dockerfile  X  index.html  docker-compose.yml
  web > Dockerfile > ...
  1 FROM nginx
  2 ADD public_html /usr/share/nginx/html
  3 EXPOSE 80
  4
```

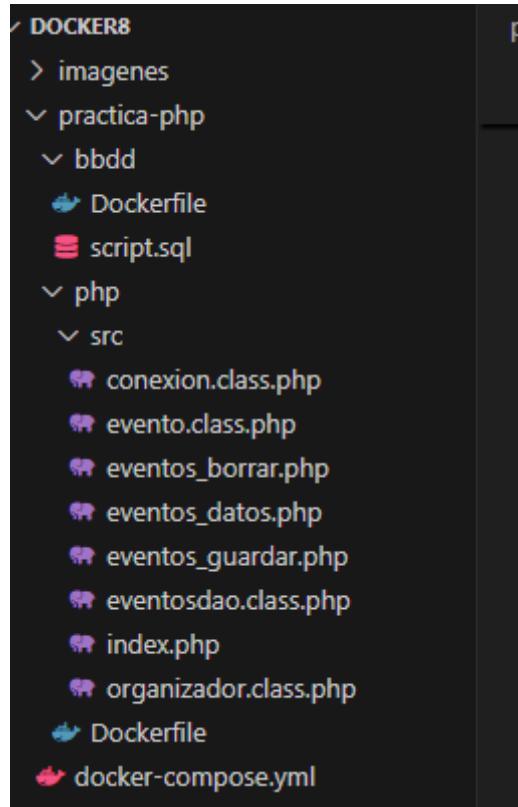
Y tras lanzarlo funcionaría:

localhost:8000

Hola

3.- Ahora vamos a desplegar una página de php.

Esta es la estructura:



Este es el docker-compose, que más adelante modificaremos:

```
1  services:
2    php:
3      build: ./php
4      ports:
5        - 8000:80
6        # Este puerto tiene que ser compatible con el del dockerfile (el 80)
7      volumes:
8        - scripts:/var/www/html
9
10   bbdd:
11     build: ./bbdd
12     ports:
13       - 4000:3306
14       # El puerto 3306 puede cambiar
15     volumes:
16       - datos_bbdd:/var/lib/mysql
17     environment:
18       - MYSQL_ROOT_PASSWORD=root
19     volumes:
20       - scripts:
21         - datos_bbdd:
```

Y este es el dockerfile de la base de datos:

```
practica-php > bbdd > Dockerfile > ...
1  FROM mysql
2  COPY script.sql /docker-entrypoint-initdb.d/
```

Este es el dockerfile del backend:

```
practica-php > php > Dockerfile > ...
1  FROM php:8.4-apache
2  RUN docker-php-ext-install pdo pdo_mysql
3  ADD ./src /var/www/html
4  EXPOSE 80
5  # Este puerto tiene que ser compatible con el del docker compose
```

Aquí necesitamos 2 Dockerfiles, uno para la base de datos y otro para el backend.

4.- Ahora añadiremos el phpmyadmin, con el siguiente docker-compose.

```
practica-php > docker-compose.yml
  ▷ Run All Services
1  services:
2    php:
3      build: ./php
4      ports:
5        - 8000:80
6        # Este puerto tiene que ser compatible con el del dockerfile (el 80)
7      volumes:
8        - scripts:/var/www/html
9    bbdd:
10      build: ./bbdd
11      ports:
12        - 4000:3306
13        # El puerto 3306 puede cambiar
14      volumes:
15        - datos_bbdd:/var/lib/mysql
16      environment:
17        - MYSQL_ROOT_PASSWORD=root
18    phpmyadmin:
19      image: phpmyadmin
20      ports:
21        - 9000:80
22      environment:
23        - PMA_HOST=bbdd
24      volumes:
25        - scripts:
26          - datos_bbdd:
```

Antes de levantar el contenedor hay que asegurarse de que nuestra clase conexión apunte a nuestra base de datos del contenedor de la siguiente manera.

```
<?php
4 references | 0 implementations
class Conexion
{
    3 references
    private static $instancia = null;
    3 references
    private $conexion;
    1 reference
    private $host = 'bbdd';
    1 reference
    private $usuario = 'root';
    1 reference
    private $password = 'root';
    1 reference
    private $basedatos = 'dwes_01_gestion_eventos';

    //Constructor privado
    0 references
    private function __construct()
    {
```

El host es igual al nombre del servicio de la base de datos

Como se puede comprobar, el ejercicio funciona:



Inserción de datos del evento

Nombre del evento.*

Fecha.* dd/mm/aaaa

Ubicación.*

Número de asistentes.*

Organizador:

[Mostrar los eventos guardados](#)

[Borrar eventos](#)

5.- Ahora para el último ejercicio cambiaremos el Dockerfile del backend y el docker-compose:

Este es el Dockerfile:

```
1  FROM debian
2  RUN apt-get update && apt-get install -y apache2 libapache2-mod-php php php-mysql && apt-get clean && rm -rf /var/lib/apt/lists/*
3  RUN rm /var/www/html/index.html
4  ADD ./src /var/www/html/
5  EXPOSE 80
6  CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
7  # Este puerto tiene que ser compatible con el del docker-compose
```

Y este el docker-compose:

```
docker-compose.yml  Dockerfile  connexion.class.php

docker-compose.yml
  services:
    php:
      build: ./php
      ports:
        - 8000:80
      volumes:
        - ./php/src:/var/www/html
    bbdd:
      build: ./bbdd
      ports:
        - 4000:3306
      volumes:
        - datos_bbdd:/var/lib/mysql
      environment:
        - MYSQL_ROOT_PASSWORD=root
    phpmyadmin:
      image: phpmyadmin
      ports:
        - 9000:80
      environment:
        - PMA_HOST=bbdd
      volumes:
        - scripts:
          - datos_bbdd:
```

El ejercicio volvería a funcionar, pero corriendo desde un Debian.



localhost:8000

Inserte los datos del evento

Nombre del evento.*

Fecha.* dd/mm/aaaa

Ubicacion.*

Número de asistentes.*

Organizador: Eventos Corporativos S.L.

[Mostrar los eventos guardados](#)

[Borrar eventos](#)