DOCKER

Docker ejecuta los procesos de forma aislada, pero en tu propia maquina, de forma "mágica" con los Cgroups y namespaces

Dockerfiles y docker-compose

Sentencias

FROM	Crear la imagen desde otra imagen oficial	
RUN	Va seguido de los comanados a ejecutar dentro del contenedor	
CMD	Comando que se ejecutara al hacer *docker run*	Si no se indica, se ejecuta el de la imagen del **FROM.** Se ejecuta como argumento de lo que se indica en el **ENTRYPOINT**
LABEL	Etiquetar la imagen	
EXPOSE	Exponer puertos	
ADD	Copia en el contenedor, permite poner URL's	
COPY	Copia en el contenedor, solo rutas locales	
ENV	Variables de entorno a incluir en el conteredor	
ENTRYPOINT	Se ejecuta siempre que se arranca un contenedor, lo que se indique aquí	default: *"/bin/sh", "c"* Si no se indica, se ejecuta el de la imagen del **FROM**

Construir imagen (Repositorio de imagenes hub.docker.com)

```
docker build -t name-image .
```

- `name-image`: Nombre de la imagen a construir
- `.`: Donde esta el dockerfile desde donde se va a construir la imagen

Docker cache

La 1º vez que se construye la imagen tarda un poco en ejecutar todo lo necesario.

Si construimos de la imagen por 2º vez solo se ejecutaran las sentencias que no estén cacheadas, es decir, que hayan sido cambiadas.

Si hacemos justo después del **FROM** un **COPY** de nuestro código fuente, cada vez que este cambie se ejecutara el **COPY** y todas las sentencias que se encuentren por debajo de nuestro **COPY.**

Por esto el orden en que escribimos nuestro *dockerfile* es muy importante. Deben ir arriba las sentencias que menos cambian.

Ya que cada sentencia FROM, RUN, COPY ... crea una nueva capa que se cachea, desde que una de esta capas se modifique/cambie, se tienen que volver a crear la capa y cada una de las que están por debajo.

1 sentencia - 1 capa

Esto solo crea una capa de cache y hace que ocupe menos espacio. Si tenemos dos RUN como arriba nos crearía 2 capas.

Imagenes pequeñas

Las imagenes deben ocupar lo menos posible.

Una imagen pesara más según las sentencias que tenga que ejecutar. Por ejemplo es mejor concatenar las sentencias del **RUN** con **&&**

```
RUN apt-get update
RUN apt-get install -y \
    bzr \
    cvs \
    git \
    mercurial \
    subversion
```

```
RUN apt-get update && apt-get install -y \
    bzr \
    cvs \
    git \
    mercurial \
    subversion
```

Multistage build

Es un *dockerfile* con varias sentencias **FROM.** Nos permiten separar la imagen de ***build***, donde generamos nuestra aplicación, de la imagen que finalmente ejecutaremos, haciendo que ocupe menos y sea más segura.

build de la aplicación dejando en */app* el código a desplegar

```
FROM composer as backend
WORKDIR /app
COPY . /app/
RUN composer install\
    --ignore-platform-reqs \
    --no-ansi \
    --no-dev \
    --no-interaction \
    --no-scripts
```

- `composer`: Imagen oficial de composer
- `as backend`: alias

despliegue de la aplicación construida en la imagen anterior

```
FROM php:7.2-apache
RUN pecl install xdebug-2.6.0 \
    && docker-php-ext-enable xdebug \
    && docker-php-ext-install pdo pdo_mysql
WORKDIR /app
COPY --from=backend /app /var/www/html/
COPY php.ini /usr/local/etc/php/php.ini
```

Copia desde la imagen **backend** Tambien te puede indicar **--from=0** hacendo referencia la primera imagen

Good practice

```
FROM composer as backend
WORKDIR /app
COPY composer.json composer.lock /app/
RUN composer install\
    --ignore-platform-reqs \
    --no-ansi \
    --no-dev \
    --no-interaction \
    --no-scripts
COPY . /app/
RUN composer dump-autoload --no-dev --optimize --classmap-
authoritative

FROM php:7.2-apache
RUN pecl install xdebug-2.6.0 \
    && docker-php-ext-enable xdebug \
    && docker-php-ext-install pdo pdo_mysql
WORKDIR /app
COPY --from=backend /app /var/www/html/
COPY php.ini /usr/local/etc/php/php.ini
```

Solo copia el fichero de las dependencias. Solo se necesita este para descargar las dependencias, por lo que nos ahorramos copiar toda la aplicación. **Esta linea solo se ejecutara si cambia el fichero composer.ison**

Solo se ejecutara si cambia el composer.json

docker-compose

Es un fichero de configuración **.yml** en el que se especifica como se levantan los contenedores evitando hacerlo de forma manual (*docker run ...*).

docker-compose	build	Construye las imágenes definidas en el fichero docker-compose
	up	Levanta (y construye, si no lo está) los contenedores especificados en el
	up -d	Igual modo **detach**
	up --build	Fuerza a construir la imagen
	ps	Lista los contenedores levantados
	logs -f nameService	Muestra los logs del servicio indicando el nombre del servicio
	top	Muestra los procesos en ejecución
	stop	Para los contenedores especificados en el fichero
	down	Elimina los contenedores parados

Volumen

Sirve para persistir los datos de por ejemplo una base de datos, de forma que si el contenedor se para o se elimina no perdemos la información ya que la tendremos en local.

También vale para publicar aplicaciones de lenguajes interpretados, ya que simplemente tenemos que tener en el volumen los ficheros generados para desplegar la aplicación.

docker-compose.yml

```
version: '3'
services:
 app:
    image: php:7.2-apache
    ports:
    - 8000:80
    volumes:
    - ${PWD}:/var/www/html
```

- `image: php:7.2-apache`: Indicar que imagen se va a construir de una ya existente
- `ports:`: Exponer puerto. ***xxxx:yyyy*** donde x es el puerto de la maquina local e y es el puerto del contenedor
- `volumes:`: En este caso montamos como volumen el código fuente

```
version: '3'
services:
 web:
   build:
     context: ${PWD}
     dockerfile: Dockerfile
   enviroment:
     - DISPLAY_ERRORS=On
     - MYSQL_HOST=mysql
     - MYSQL_USER=root
     - MYSQL_PASSWORD=root
   ports:
     - 8000:80
   depends_on:
     - mysql

 mysql:
   image: mysql:5.7
   enviroment:
      - MYSQL_ROOT_PASSWORD=root
   volumes:
     - ${PWD}/migrations:docker-entrypoint-initdb.d
```

- `services:`: Dentro se indican los servicios que vamos a levantar
- `enviroment:`: Variables de entorno definidas para el contenedor
- `depends_on:`: Indica que este servicio (***web***) depende del servicio ***mysql***. El servicio web no se lanazará hasta que el docker del servicio mysql se haya arrancado. no es lo mismo que el servicio dentro del docker de mysql está escuchando
- `volumes:`: A la hora de arrancar la imagen se ejecutará todo lo que tenemos en la carpeta *migrations. (.sql, .sh ...)* Estos script se ejecutan en orden alfabético *(revisar la documentación oficial de cada base de datos)*

@yodralopez