

Índice

1. Introducción.....	1
2. Contenedores vs Virtualización.....	1
3. Containerización con Docker.....	2
3.1 Imágenes vs contenedores.....	3
3.2 Docker Cli.....	4
3.3 Creación de un contenedor.....	5
3.4 Sintaxis del Docker File.....	8
3.5 Volúmenes.....	9
3.6 Entorno de desarrollo.....	9
4. Trabajo a Realizar.....	9
5. Bibliografía / webgrafía.....	10

Actividad 5 – Containerización de aplicaciones

1. Introducción

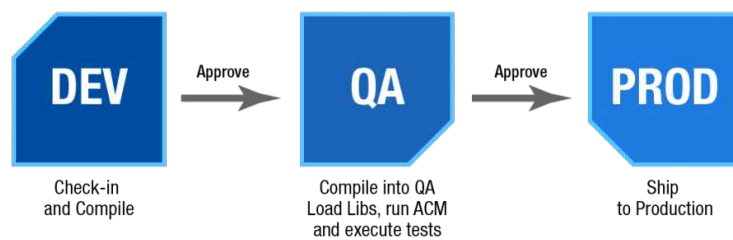
Cuando se desarrolla una Aplicación Web **el entorno de despliegue** de producción **no es el mismo** que el entorno de **desarrollo** o el el entorno de **test** (Diferentes sistemas operativos, versiones de los servicios que dan soporte a las aplicaciones,...), **Estas diferencias acostumbra a producir errores** durante el despliegue de la aplicación

Además, otro problema que se nos presenta es que la puesta en marcha del entorno de desarrollo de las aplicaciones se complica a medida que necesitamos trabajar con diferentes proyectos que poseen requerimientos muy distantes; Podemos estar trabajando con código legado de una aplicación que necesita **php5.6** y a la vez estar desarrollando un proyecto en la versión LTS de Laravel en la que necesitamos **php7.3.1**. En estas circunstancias, se recurre a **técnicas de virtualización y/o containerización** en las que **se lleva a cabo la paquetización** de todo el STACK de la aplicación.

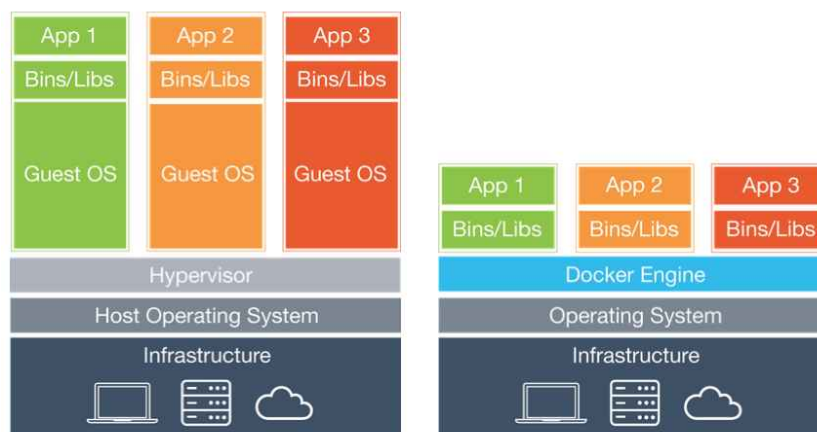
2. Contenedores vs Virtualización

Las técnicas de containerización permiten configurar los entornos de despliegue mediante código, generalmente en un fichero de configuración en formato **yaml** o **json**. Esto es lo que denominamos Infraestructura como Código (IaC).

De esta forma **podemos reproducir en cualquier máquina** e independientemente del sistema operativo **la misma arquitectura** en todos los entornos; **desarrollo, pruebas y producción**.



- La **virtualización**, se basa en la creación de una máquina completa, incluyendo el sistema operativo. Para facilitar esta tarea, hay programas como **Vagrant** que permiten crear y configurar el entorno de despliegue y aprovisionarlo a partir de un fichero de configuración.
- La **containerización**, por su parte, permite aprovechar el **núcleo del sistema operativo anfitrión** para desplegar la arquitectura, siendo mucho más ligero y no afectando al rendimiento del sistema, lo que nos permite, además de utilizarlo en los entornos de desarrollo, su utilización en entornos de producción. La tecnología Docker surge en 2013, y actualmente se ha convertido en una herramienta fundamental para muchas empresas, entre las que podemos destacar **Google**.



3. Containerización con Docker

Docker es un sistema de containerización basado en el núcleo de linux. Esto implica que solo podamos llevar a cabo la containerización de arquitecturas basadas en este sistema operativo. Algunas ventajas de su utilización son:

- Tenemos el **mismo entorno de ejecución** de la aplicación en desarrollo y producción
- Cada **aplicación** se encuentra **aislada** de las demás en su propio contenedor/es.
- Solo se necesita tener instalado el demonio de docker (**Docker Engine**) en el sistema operativo donde necesitemos hacer correr la aplicación para su despliegue. El administrador de sistemas no necesita conocer la arquitectura de la aplicación, solo se encarga de asignar suficientes recursos a la máquina.

El primer paso para trabajar con docker es instalarlo, podemos hacer uso de la guía presente en el [siguiente](#) enlace o utilizar la versión de los repositorios de **ubuntu**.

```
$ sudo apt install docker.io
```

La instalación de **docker**, también incluye la herramienta **Docker Cli**. Se trata de un cliente en línea de comandos (CLI) que nos permite gestionar el **Docker Engine**. Una vez instalado el resultado debe ser similar al siguiente

```
root@ddaw:~$ docker -v
Docker version 18.09.7, build 2d0083d
```

3.1 Imágenes vs contenedores

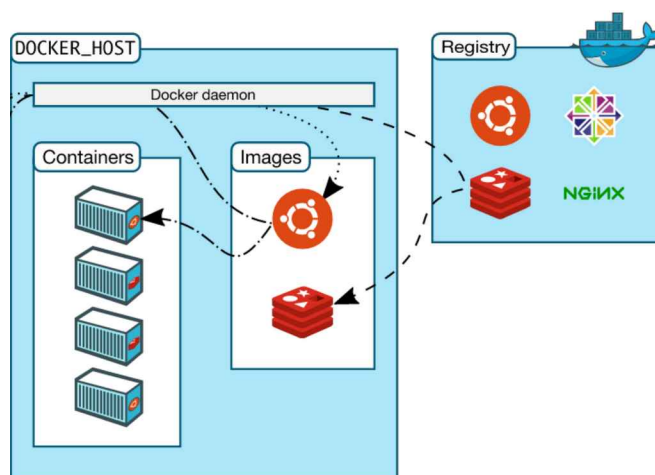
Docker encapsula las **aplicaciones en contenedores**. Un **contenedor** es el equivalente a una **máquina virtual en la virtualización clásica**, pero mucho más ligera porque utiliza el sistema operativo del host anfitrión. Las aplicaciones de cada contenedor "ven" un sistema operativo, que puede ser diferente en cada contenedor, pero quien realiza el trabajo es el sistema operativo común que hay por debajo.

Docker **crea** los **contenedores** a **partir** de **imágenes**. Las imágenes son una especie de **plantillas que contienen todo el software** que necesita la aplicación para ponerse en marcha. Las imágenes se pueden crear a partir de otras imágenes más básicas incluyendo software adicional en forma de **capas**. Todos los contenedores creados a partir de una imagen contienen el mismo software, aunque en el momento de su creación se pueden personalizar algunos detalles.

De esta forma, **un contenedor no es más que una instancia de una imagen** en la que

se han definido el sistema operativo base, los servicios y las configuraciones a aplicar necesarias para el aplicativo. Las imágenes pueden estar guardadas en forma local mediante un fichero **.tar** o utilizar [Docker Hub](#).

Docker Hub nos ofrece, a modo de repositorio **github/gitlab**, una herramienta para mantener las versiones de las distintas imágenes que vayamos creando y distribuirlas a la comunidad.



Podemos encontrar **imágenes oficiales** mantenidas directamente por los propios fabricantes de software de servicios como: *apache*, *nginx*, *php*, *mysql*,... y para diferentes distribuciones como: *ubuntu*, *debian*...

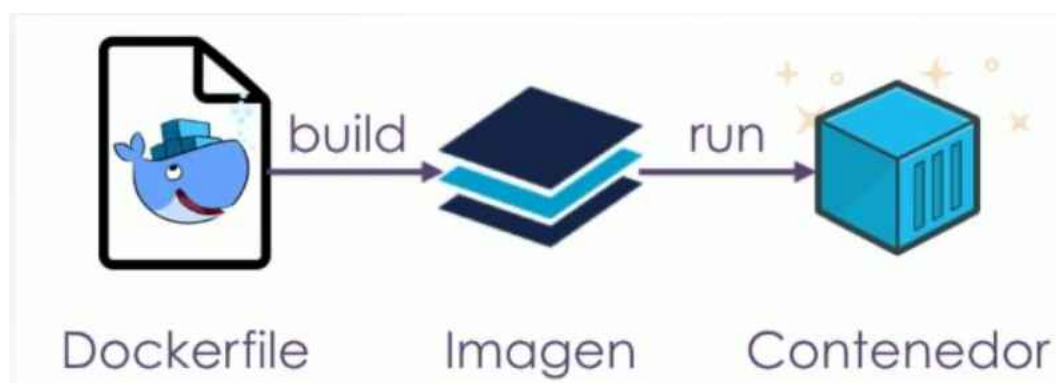
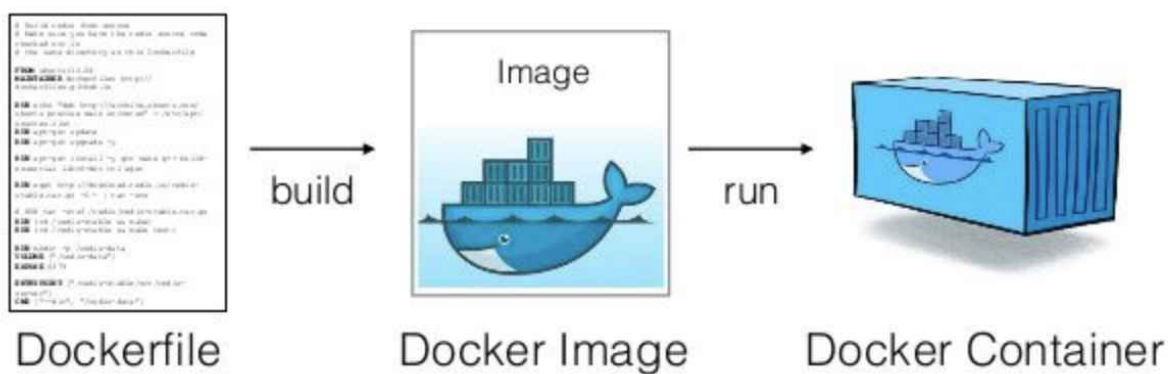
3.2 Docker Cli

La herramienta **Docker Cli** nos permite gestionar todos los contenedores desde la terminal. A continuación explicaremos los principales comandos:

<code>docker ps -a</code>	Muestra todos los contenedores de nuestra máquina. La opción -a nos muestra también aquellos que están detenidos.
<code>docker image ls</code>	Muestra las imágenes descargadas y disponibles en nuestra máquina
<code>docker build -t <id imagen> <path_fichero_dockerfile></code> <code>docker build -t myAppImage .</code>	Permite construir una imagen a partir de un dockerfile y que será la base para la construcción de contenedores.
<code>docker run IMAGEN</code>	Docker crea los contenedores a partir de imágenes

<code>docker run ubuntu:18.04</code>	locales (ya descargadas), pero si al crear el contenedor no se dispone de la imagen local, descargará la imagen.
<code>docker rm id</code>	Para y elimina un contenedor
<code>sudo docker rmi id</code>	Elimina una imagen local
<code>docker exec -it <container name> comando</code> <code>docker exec -it myContenedor /bin/sh</code>	Ejecuta un comando dentro del contenedor

3.3 Creación de un contenedor



En el [siguiente](#) enlace puedes encontrar las imágenes oficiales mantenidas por los desarrolladores de **PHP**. Seleccionaremos aquella que más se adapte a nuestras necesidades. Para nuestro caso, y dado que queremos ejecutar una aplicación web seleccionaremos como base una imagen de PHP que contenga también **apache**, por

ejemplo **7.3-apache**.**Supported tags and respective Dockerfile links**

- 7.4.1-cli-buster, 7.4-cli-buster, 7-cli-buster, cli-buster, 7.4.1-buster, 7.4-buster, 7-buster, buster, 7.4.1-cli, 7.4-cli, 7-cli, cli, 7.4.1, 7.4, 7, latest
- 7.4.1-apache-buster, 7.4-apache-buster, 7-apache-buster, apache-buster, 7.4.1-apache, 7.4-apache, 7-apache, apache
- 7.4.1-fpm-buster, 7.4-fpm-buster, 7-fpm-buster, fpm-buster, 7.4.1-fpm, 7.4-fpm, 7-fpm, fpm
- 7.4.1-zts-buster, 7.4-zts-buster, 7-zts-buster, zts-buster, 7.4.1-zts, 7.4-zts, 7-zts, zts
- 7.4.1-cli-alpine3.11, 7.4-cli-alpine3.11, 7-cli-alpine3.11, cli-alpine3.11, 7.4.1-alpine3.11, 7.4-alpine3.11, 7-alpine3.11, alpine3.11, 7.4.1-cli-alpine, 7.4-cli-alpine, 7-cli-alpine, cli-alpine, 7.4.1-alpine, 7.4-alpine, 7-alpine, alpine
- 7.4.1-fpm-alpine3.11, 7.4-fpm-alpine3.11, 7-fpm-alpine3.11, fpm-alpine3.11, 7.4.1-fpm-alpine, 7.4-fpm-alpine, 7-fpm-alpine, fpm-alpine
- 7.4.1-zts-alpine3.11, 7.4-zts-alpine3.11, 7-zts-alpine3.11, zts-alpine3.11, 7.4.1-zts-alpine, 7.4-zts-alpine, 7-zts-alpine, zts-alpine
- 7.4.1-cli-alpine3.10, 7.4-cli-alpine3.10, 7-cli-alpine3.10, cli-alpine3.10, 7.4.1-alpine3.10, 7.4-alpine3.10, 7-alpine3.10, alpine3.10
- 7.4.1-fpm-alpine3.10, 7.4-fpm-alpine3.10, 7-fpm-alpine3.10, fpm-alpine3.10
- 7.4.1-zts-alpine3.10, 7.4-zts-alpine3.10, 7-zts-alpine3.10, zts-alpine3.10
- 7.3.13-cli-buster, 7.3-cli-buster, 7.3.13-buster, 7.3-buster, 7.3.13-cli, 7.3-cli, 7.3.13, 7.3
- 7.3.13-apache-buster, 7.3-apache-buster, 7.3.13-apache, 7.3-apache

A) El primer paso será la creación de un fichero Dockerfile que nos permitirá configurar la instancia de la imagen que vamos a lanzar. Un fichero Dockerfile básico sería el siguiente:

Dockerfile
<pre>#indicamos la imagen base FROM php:7.3-apache #Copiamos el contenido local de myApp/ a var/www/html del contenedor COPY myApp/ /var/www/html #Abrimos el puerto 80 EXPOSE 80</pre>

En el fichero DockerFile estamos indicando a DockerEngine que utilice la imagen base seleccionada en el paso anterior, que abra el **puerto 80** del contenedor y que copie el contenido de la carpeta local myApp/ (la web que queremos desplegar) en el **DocumentRoot** del servidor apache del contenedor.

Nota: en el ejemplo el directorio **myApp** contiene una página **index.php** con la función

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

B) Una vez creado el fichero, solo nos queda generar la **imagen** que utilizaremos para crear todos los contenedores que necesitemos.

```
$ sudo docker build -t my-first-container .
```

El parámetro `-t` sirve para indicar el nombre de la imagen.

```
root@ddaw ~/docker-test$ sudo docker build -t my-first-container .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM php:7.3-fpm-alpine3.11
7.3-fpm-alpine3.11: Pulling from library/php
e6b0cf9c0882: Pull complete
3c7a574e8632: Pull complete
2a6c65f587e9: Pull complete
ba33e4b639bb: Pull complete
6cb91a91a20b: Pull complete
cfac268afcd6: Pull complete
129dc26ebe8f: Pull complete
a3de864ebbab: Pull complete
5c198efec1ca: Pull complete
45211ed8ec9d: Pull complete
Digest: sha256:a2787aaf21a76e47495fc26c771552a9ac6ebe454dd55d479ad618c1f54e381d
Status: Downloaded newer image for php:7.3-fpm-alpine3.11
--> c64e8e698685
Step 2/3 : COPY myApp/ /var/www/html
--> a963b9b549ff
Step 3/3 : EXPOSE 80
--> Running in 80caa5f54502
Removing intermediate container 80caa5f54502
--> 50ee02d241a1
Successfully built 50ee02d241a1
Successfully tagged my-first-container:latest
```

Como podemos ver, ya tenemos disponible la imagen en nuestro ordenador, para crear tantas instancias (contenedores) como necesitemos.

```
root@ddaw ~/docker-test$ sudo docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-first-container	latest	50ee02d241a1	4 minutes ago	80.2MB
php	7.3-fpm-alpine3.11	c64e8e698685	3 days ago	80.2MB

C) El último paso será la creación del contenedor:

```
$sudo docker run -p 8080:80 my-first-container
```


Con la opción **run** indicamos que queremos lanzar una instancia de la imagen construida y con la opción **-p** que el puerto **8080** de nuestra máquina se redirija al puerto 80 del contenedor.

Si ejecutamos la orden `$ sudo docker ps`, veremos un nuevo contenedor en ejecución.

```
alecogi@ddaw ~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2fc28c31ce72	my-first-container	"docker-php-entrypoi..."	21 minutes ago	Up 21 minutes	0.0.0.0:8080->80/tcp	elated_cray

Si accedemos a **ip-servidor:8080**, estaremos accediendo directamente al puerto 80 del contenedor y por tanto a la página desplegada en **/var/www/html**.

PHP Version 7.3.13 	
System	Linux 2tc28c31ce72 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
Build Date	Dec 28 2019 21:15:05
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=usr' '--with-sqlite3=usr' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731.NTS
PHP Extension Build	API20180731.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled

3.4 Sintaxis del Docker File.

En este apartado vamos a hacer una introducción al uso de las instrucciones más importantes que podemos definir dentro de un fichero **Dockerfile**, para una descripción más detallada consulta la [documentación oficial](#).

FROM FROM <imagen>:<tag>	Indica la imagen base que se va a utilizar para crear la imagen
MAINTAINER <nombre> <correo> MAINTAINER ddae@cipfpbatoi.com	Permite configurar datos del autor de la imagen
RUN RUN ["apt","install","apache2"]	Ejecuta cualquier comando en una imagen y hace un commit de los resultados. Esa nueva imagen, será utilizada para el siguiente paso en el Dockerfile
ENV ENV password 1234567890	Esta instrucción configura las variables de entorno. Estos valores estarán disponibles para todos los comandos que sigan en el Dockerfile.
EXPOSE EXPOSE 80 443	Especifica a docker que el contenedor escuche en los puertos indicados.
CMD CMD ["-p", "80"] CMD ["echo", "Hola Mundo"]	Nos permite especificar argumentos para el ENTRYPOINT. También dispone de la forma exec que permite especificar un ejecutable
ENTRYPOINT ENTRYPOINT ["/bin/ping"]	Nos permite especificar un comando que siempre se ejecutará al iniciar el contenedor. De esta forma configuramos un contenedor como un ejecutable. (Solo ejecutará el proceso pasado como argumento).

3.5 Volúmenes

Los contenedores en docker son elementos efímeros que se crean cuando se necesitan y que no importa que se destruyan ya que pueden ser reconstruidos a partir de la imagen. Se trata de **elementos sin estado**, lo que implica que si la aplicación o aplicaciones incluidas en el contenedor generan datos y esos datos se guardan en el propio contenedor, se perderán en el momento en que se destruyera el contenedor. Para conseguir la persistencia de los datos, se pueden emplear dos técnicas:

- **Directorios enlazados:** la información se guarda fuera de Docker, en la máquina *host* (en nuestro caso, en la máquina virtual de Ubuntu).
- **Volúmenes:** La información se guarda mediante Docker, pero en unos elementos llamados *volúmenes*, independientes de las imágenes y de los contenedores.

Los **volúmenes** son la mejor solución cuando la información es generada por el propio contenedor y los directorios enlazados pueden ser más adecuados cuando la información no es generada por el contenedor, como podría ser un entorno de desarrollo.

3.6 Entorno de desarrollo

Cuando estamos desarrollando una aplicación, necesitamos que los ficheros se encuentren siempre sincronizados en el `documentRoot` del contenedor. Para ello se hace uso de los **directorios enlazados (bind mount)**.

Este se lleva a cabo mediante la opción **--mount**, indicando el nombre del **directorio host** de la máquina, seguida de ":" y el nombre del directorio del contenedor.

```
$ sudo docker run -d -p 8001:80 --mount  
type=bind,source=${PWD}/myApp/,target=/var/www/html/ my-first-container
```

4. Trabajo a Realizar (2 puntos)

1. Crea una imagen tomando como base una imagen **oficiales** ofrecida por los desarrolladores de php que contenga el servidor **apache** y una versión **7.4 de php**. Incluye la instalación de la **extensión mysqli** en la imagen que crees a partir del DockerFile (`RUN docker-php-ext-install mysqli`).

2. Crea un contenedor a partir de la imagen. Este será utilizado para el **desarrollo local** de una aplicación que se encontrará en tu directorio local **~/prueba-docker**. Enlaza esta carpeta con el **document root** del virtualHost por defecto configurado en el contenedor. De esta forma, podremos desarrollar en local y ejecutar en el contenedor sin tener que reiniciarlo.
3. Crea una página **index.php**, con la función `phpinfo()`, en el directorio del proyecto y accede a ella a través del navegador. (Realiza una captura de pantalla del resultado).

Documenta con capturas de pantalla y explicaciones los pasos que has dado.

5. Bibliografía / webgrafía

- Prieto Vega, M. Servidores de aplicaciones Web.
https://ioc.xtec.cat/materials/FP/Materials/ICC0_DAW/DAW_ICC0_M08/web/html/media/fp_daw_m08_u2_pdfindex.pdf. Institut Obert de Catalunya - IOC
- Docker.com. Documentación oficial. <https://docs.docker.com>.
- Mclibre.com. Introducción a Docker.
<https://www.mclibre.org/consultar/webapps/lecciones/docker-1.html>.