Ciclo de vida de nuestras aplicaciones con docker

José Domingo Muñoz

IES Gonzalo Nazareno

Febrero 2021

Paso 1:Desarrollo de nuestra aplicación

En este ejemplo vamos a desarrollar una página web que va a ser servida por un servidor web que se ejecutará en un contenedor Docker.

Por lo tanto lo primero que debemos hacer es crear nuestra página web:

```
$ cd public_html
$ echo "<h1>Prueba</h1>" > index.html
```

Paso 2: Creación de la imagen Docker

Utilizando un fichero Dockerfile definimos como vamos a crear nuestra imagen:

- Qué imagen base vamos a utilizar.
- Qué paquetes vamos a instalar
- Donde copiamos nuestro código fuente (página web)
- Indicamos el servicio que va a ejecutar el contenedor (servidor apache)

Paso 2: Creación de la imagen Docker

El fichero Dockerfile:

```
FROM debian
RUN apt-get update -y && apt-get install -y \
    apache2 \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
ADD ./public_html /var/www/html/
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Podríamos usar una imagen base con apache2 ya instalado:

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

Paso 2: Creación de la imagen Docker

Creamos nuestra imagen, desde el directorio donde tenemos el `Dockerfile`, ejecutamos:

```
$ docker build -t josedom24/aplicacionweb:v1 .
Sending build context to Docker daemon  3.584kB
Step 1/4 : FROM debian
 ---> be2868bebaba
Step 2/4 : RUN apt-get update -y && apt-get install -y apache2 & apt-get clean && rm -r
 ...
Successfully built 518871c9fc0c
Successfully tagged josedom24/aplicacionweb:v1
```

Podemos comprobar que en nuestro entorno local tenemos la imagen que acabamos de crear:

```
$ docker image ls
REPOSITORY                TAG                 ...
josedom24/aplicacionweb   v1                  ...
debian                    latest              ...
```

Paso 3: Probamos nuestra aplicación en el entorno de desarrollo

Creamos un contenedor en nuestro entorno de desarrollo:

```
$ docker run --name aplweb -d -p 80:80 josedom24/aplicacionweb:v1
  fbdd73529e2bb2d9ee9c64150315137416 88e6d38509572251f5b624ed7dc23f

$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  ...
fbdd73529e2b        josedom24/aplicacionweb:v1     "/usr/sbin/apache2ct..."   ...
```

Probamos nuestra aplicación:

Figura 1: En desarrollo

Paso 4: Distribuimos nuestra imagen

Vamos a subir nuestra imagen al registro Docker Hub:

```
$ docker login
...
$ docker push josedom24/aplicacionweb:v1
The push refers to repository [docker.io/josedom24/aplicacionweb]
ac126159496f: Pushed
cc15ec5f0c43: Pushed
...
```

Comprobamos que está subida al repositorio:

```
$ docker search josedom24/aplicacionweb
NAME                      DESCRIPTION...
josedom24/aplicacionweb:v1
```

Paso 5: Implantación de la aplicación

En el el entorno de producción, bajamos la imagen de Docker Hub y creamos el contenedor:

```
$ docker pull josedom24/aplicacionweb:v1
v1: Pulling from josedom24/aplicacionweb
9a029d5ca5bb: Pull complete
...
$ docker run --name aplweb_prod -d -p 80:80 josedom24/aplicacionweb:v1
```

Paso 6: Modificación de la aplicación

Al modificar el código de la aplicación tenemos que generar una nueva imagen.

```
$ cd public_html
echo "<h1>Prueba 2</h1>" > index.html
$ docker build -t josedom24/aplicacionweb:v2 .
```

Podemos probarla en el entorno de desarrollo, eliminando el contenedor anterior:

```
$ docker rm -f aplweb
$ docker run --name aplweb2 -d -p 80:80 josedom24/aplicacionweb:v2
```

Y probamos la aplicación:

Figura 2: Ha cambiado...

Paso 6: Modificación de la aplicación

Subimos la nueva versión de la aplicación. En el entorno de producción: bajamos la nueva versión, eliminamos el contenedor de la versión antigua y creamos un nuevo contenedor con la nueva imagen:

```
$ docker push josedom24/aplicacionweb:v1
...
```

En producción:

```
$ docker pull josedom24/aplicacionweb:v2
...
$ docker rm -f aplweb_prod
$ docker run --name aplweb2_prod -d -p 80:80 josedom24/aplicacionweb:v2
```