Uso de Docker-Compose para la creación de aplicaciones

Introducción

Después de haber creado la aplicación de Spring Boot con MongoDB, se nos recalcó la dificultad del proceso de lanzamiento e interconexión de los diversos contenedores que constituían dicha aplicación. Es por ello que se nos encomendó la tarea de buscar si existe una forma en Docker de gestionar estas aplicaciones multicontenedor.

La forma recomendada es **Docker Compose**

Docker Compose

Es una herramienta ***externa*** *al* ***docker engine***, por lo que es necesario instalarla.

Se fundamenta en la existencia de un archivo, *docker-compose.yml*, que es el encargado de declarar la estructura de la aplicación. Luego, con los comandos propios de esta aplicación, se levanta la estructura deseada en docker.

Uso

Hemos usado los siguientes comandos:

- **docker-compose up**: levanta la aplicación según el *docker-compose.yml* Notar que al igual que con Docker, si se quiere **su ejecución en segundo plan**, hay que especificar el parámetro *dettached* mediante la opción **-d**
- **docker-compose down**: para los contenedores que componen la aplicación y realiza algunas tareas de limpieza relativas

Así mismo, durante la realización de la tarea experimenté varios problemas con la conexión. Tras analizar los logs, observé que había errores en la cadena de conexión del MongoDB, que asocié a problemas de *caché*. Investigando, resolví mis problemas con el siguiente comando:

- **docker-compose build –no-cache**: permite reflejar los cambios en el Dockerfile o en el directorio de trabajo en la aplicación

docker-compose.yml

Es el fichero que describe la configuración de la aplicación. Su estructura básica consiste en definir la versión de la sintaxis usada e ir definiendo ***servicios***, que podrían asociarse a los distintos contenedores que se levantarán. Además de

poder especificarse a nivel de servicio, también pueden definirse a nivel global aspectos tales como los **volumenes** y las **redes**.

Un aspecto a tener en cuenta de este archivo es que es **incompatible con las tabulaciones**, por lo que si el documento contiene el carácter ***/t***, no podrá ser utilizado.

A continuación expongo mis dos documentos utilizados:

Construyendo a partir de un Dockerfile

```
 0  version: '3'
 1  services:
 2  spring-guestbook-compose:
 3   build: .
 4   ports:
 5    - 40010:8080
 6  spring-mongo-compose:
 7   image: mongo
 8   volumes:
 9    - /home/centos/docker/persistence/mongodb/
          spring-guestbook-compose:/data/db
10
```

Puede verse que se crean dos servicios:

1. Un servicio de mongo que, a partir de la imagen oficial, levanta un contenedor con persistencia en el directorio deseado. Notar que el nombre dado al servicio es el equivalente al *'localhost'* de la aplicación Java.
2. Un servicio de la aplicación. Esta aplicación se construye con las especificaciones del Docekerfile, especificándose los puertos que se usarán en la aplicación.

Notar que no es necesario crear ninguna red de forma manual, ya que docker-compose crea un network básico por defecto

Constuyendo a partir de una imagen oficial

```
 0  version: '3'
 1  services:
 2  spring-guestbook-compose-variation:
 3   image: maven:alpine
 4   ports:
 5    - 40011:8080
 6   volumes:
 7    - ./app:/home/
 8   entrypoint: /bin/bash
```

```
 9    command: "/home/start.sh"
10   spring-mongo-compose-variation:
11    image: mongo
12    volumes:
13     - /home/centos/docker/persistence/mongodb/
            spring-guestbook-compose-variation:/data/db
14
```

Este archivo es análogo al anterior, con la salvedad de que en vez de usar un Dockerfile, usa una imagen ya hecha. Como se ha partido de una imagen oficial, se ha especificado otro volumen que contiene el código de la aplicación y un script para compilarlo y ejecutarlo con los parámetros adecuados. Notar la forma de especificar el *entrypoint* y *cmd* de la aplicación, muy similar a lo que haríamos en un Dockerfile.