Aplicación de SpringBoot con MongoDB Dockerizada

Introducción

En esta tarea se nos encargó realizar una aplicación sencilla en **Spring Boot** que sea ejecutable mediante el comando *java -jar*. Esta aplicación debía tener además una conexión a una base de datos **MongoDB**, con unas características simples de *listado* e *inserción*.

MongoDB

Conexión

Este motor de base de datos nos es totalmente desconocido, así que una parte del tiempo fue dedicado a familiarizarse con él.

En primer lugar, es muy importante entender el proceso de la conexión de la aplicación de *Spring* a esta base de datos. Al lanzar la aplicación, es imperativo pasar una url de conexión a la base de datos. Esta configuración está bastante **integrada en Spring Boot**, quedando el parámento resultante en:

```
shell  -Dspring.data.mongodb.uri=mongodb://spring-demo-mongo/guestbook
```

Notar en el comando anterior que se le ha pasado **guestbook**, que sería el nombre de la base de datos y **spring-demo-mongo**, que sería el **servidor** de la base de datos. En nuestras pruebas de desarrollo era *localhost*, pero al dockerizar la aplicación y el sistema de base de datos, es el nombre del **contenedor de mongo**.

Estos parámetros de conexión pueden configurarse dentro de la aplicación. Creamos la clase `MongoConfig` que *extiende* a `AbstractMongoConfiguration` y sobreescribiendo unos métodos, conseguimos personalizar el nombre de la base de datos y la ruta.

Shell

Con estas configuraciones, se conectan la aplicación y la base de datos. A veces puede ser interesante acceder a a la *shell* de la base de datos. Con la base de datos *dockerizada*, podemos usar comandos de Docker para interactuar con el contenedor. En concreto usamos el comando exec. De forma resumida, ejecutamos un:

```
docker exec -it contenedor mongo
```

Con este comando nos loguearemos a la base de datos y tras un `exit`, saldremos de la shell, no habiendo parado el contenedor en ningún momento.

Persistencia

Al ser una base de datos, es especialmente interesante disponer de algo que permita que la volatilidad de Docker por su naturaleza no impida la persistencia de los datos. Para este fin existen en Docker los **volúmenes**, que permiten crear una carpeta en el entorno que corre el demonio de Docker y usarla como ruta de almacenamiento, para disponer de los datos de forma independiente a lo que pase con el contenedor.

En el caso de mongo, lo conseguimos con el comando:

```
docker run  -v $DBPATH:/data/db -d mongo
```

Donde `DBPATH` es la ruta física (en nuestro caso `/home/centos/docker/bd-guestbook`, por ejemplo) y `/data/db` es la ruta dentro del contenedor a persistir; en este caso, es la ruta que usa MongoDB para almacenar los datos.

Red

Para que los distintos contenedores se puedan 'ver', es necesario que estén en la misma red. Para gestionar las redes, está la utilidad docker network. Para esta aplicación no ha sido necesario configurar muchos aspectos de la red, sólamente:

- Crear la red: `docker network create red`
- Configurar los puertos de forma acorde:
- No se abre ningún puerto en la base de datos
- Se habre el puerto 8080 en la aplicación
- Lanzar los contenedores especificando la red:
- Simplemente hay que especificar el parámetro `--network=red` al crear el **contenedor**

Aplicación

La aplicación está realizada con **Spring Boot**. Ha sido basada en un paradigma **MVC** usando para las vistas **JSPs**. El desarrollo ha sido bastante inmediato debido a la sencillez de la aplicación, pero es interesante destacar algunos errores que surgen al dockerizar la aplicación, estando estos causados por ciertas incompatibilidades entre *Spring Boot* y *JSP*.

Además de adecuar las dependencias a lo especificado en nuestro `pom.xml`, es importante:

- Empaquetar la aplicación como `.war` aunque la ejecutemos con un `java -jar`
- Que el directorio de todo lo relacionado con las vistas esté debajo del directorio `webapps`
- Guardar **css** y **js** en sendas carpetas en un subdirectorio `static`
- Guardas los **jsp* en un subdirectorio del tipo `WEB-INF/jsp`
- Tener el archivo `application.properties` debidamente configurado en `resources`

Dockerización

En este primer paso, lanzamos ambos contenedores por separado. Para automatizar este paso, realicé un script que:

- Lanza el contenedor de mongo según los nombres especificados
- Modifica el archivo de configuración java con las conexiones apropiadas mediante `sed`
- Crea un Dockerfile con la conexión correcta
- Crea e instancia la aplicación