Desarrollo de microservicios con Spring Boot y Docker

Por **Jorge Pacheco Mengual** - 7 junio, 2016

Siguiendo con la serie de tutoriales dedicados a Docker, vamos a ver cómo desplegar un microservicio desarrollado con Spring Boot en un contenedor Docker.
Posteriormente veremos como escalar y balancear este microservicio a través de HAProxy.

0. Índice de contenidos

1. Introducción

En el tutorial Introducción a los microservicios del gran José Luis
vimos una introducción al concepto de microservicios, cuales son sus ventajas e inconvenientes y cuando podemos utilizarlos.

El objetivo que perseguimos con el presente tutorial es desarrollar un microservicio con Spring Boot,
empaquetarlo dentro de una imagen Docker, dentro de la fase de construcción de maven, y una vez podamos
levantarlo, ver una posibilidad de escalabilidad gracias a **Docker Compose** y **HAProxy**.

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: MacBook Pro 15' (2.3 GHz Intel Core i7, 16GB DDR3 SDRAM)
- Sistema Operativo: Mac OS X El Capitan 10.11
- Software: Docker 1.11.1, Docker Machine 0.7.0, Docker Compose 1.7.1
- Software: Spring Boot 1.4.0.M3

3. El Microservicio

El objetivo del tutorial no es tanto el desarrollo de microservicios con Spring Boot, sino su empaquetamiento y despliegue,
por tanto vamos a implementar un microservicio
'tonto' cuya única funcionalidad es devolvernos un mensaje de hola.

El código lo podeís encontrar en mi cuenta de github aquí, lo primero que deberíamos implementar es un
@RestControler como el que describimos a continuación:

```
Shell
```

```
package com.autentia;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MicroServiceController {


    private final AddressService service;

    @Autowired
    public MicroServiceController(AddressService service) {
        this.service = service;
    }

    @RequestMapping(value = "/micro-service")
    public String hello() throws Exception {

        String serverAddress = service.getServerAddress();
        return new StringBuilder().append("Hello from IP address: ").append(serverAddress)
    }


}
```

Como podemos observar es un ejemplo muy sencillo, hemos declarado un controlador rest, al cual le hemos inyectado un servicio que recupera la IP del servidor, y devuelve un string del tipo

«Hello from, IP address xx.xx.xx.xx»

La clase principal de nuestro microservicio encargada levantar un tomcat embebido con nuestro microservicio tendría un aspecto parecido a este:

Shell

```
package com.autentia;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MicroServiceSpringBootApplication {

  public static void main(String[] args) {

      SpringApplication.run(MicroServiceSpringBootApplication.class, args);
  }
}
```

Podemos levantar nuestro servicio de la siguiente manera:

Shell

```
  mvn clean spring-boot run
```

Una vez levantado el microservicio podemos invocarlo de la siguiente manera:

```Shell
curl http://localhost:8080/micro-service
```

Hasta ahora nada impresionante … pasemos al siguiente punto

4. Dockerizar el microservicio

En este apartado vamos a ver como podemos 'empaquetar' nuestro microservicio dentro de un contenedor docker,
para ello vamos a usar el plugin de maven spotify/docker-maven-plugin.

Antes de meternos de lleno en el uso de este plugin, vamos a generar un Dockerfile de nuestro microservicio,
para ello nos creamos un directorio **src/main/docker** y creamos nuestro Dockerfile de la siguiente manera:

```Shell
FROM frolvlad/alpine-oraclejdk8:slim
MAINTAINER jpacheco@autentia.com
ADD micro-service-spring-boot-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Repasemos el Dockerfile:

- **FROM:** Tomamos como imagen base frolvlad/alpine-oraclejdk8
  esta imagen está basada en Alpine Linux que es una distribución Linux de sólo 5 MB, a la cual se le ha añadido la OracleJDK 8.
- **ADD:** Le estamos indicando que copie el fichero micro-service-spring-boot-0.0.1-SNAPSHOT.jar al contenedor con el nombre app.jar
- **EXPOSE:** Exponemos el puerto 8080 hacia fuera (es el puerto por defecto en el que escuchará el tomcat embebido de nuestro microservicio)
- **ENTRYPOINT:** Le indicamos el comando a ejecutar cuando se levante el contenedor, como podemos ver es la ejecución de nuestro jar

El siguiente paso en añadir el plugin a nuestro pom.xml de la siguiente manera

```Shell
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>1.8</java.version>
  <docker.image.prefix>autentia</docker.image.prefix>
</properties>
 .....

 <plugins>
   <plugin>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
   <plugin>
     <groupId>com.spotify</groupId>
     <artifactId>docker-maven-plugin</artifactId>
     <version>0.4.10</version>
     <configuration>
       <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
       <dockerDirectory>src/main/docker</dockerDirectory>
       <serverId>docker-hub</serverId>
       <registryUrl>https://index.docker.io/v1/</registryUrl>
       <forceTags>true</forceTags>
       <imageTags>
         <imageTag>${project.version}</imageTag>
       </imageTags>
       <resources>
         <resource>
           <targetPath>/</targetPath>
           <directory>${project.build.directory}</directory>
           <include>${project.build.finalName}.jar</include>
         </resource>
       </resources>
     </configuration>
     <executions>
       <execution>
         <id>build-image</id>
         <phase>package</phase>
         <goals>
           <goal>build</goal>
         </goals>
       </execution>
       <execution>
         <id>push-image</id>
         <phase>install</phase>
         <goals>
           <goal>push</goal>
         </goals>
         <configuration>
           <imageName>${docker.image.prefix}/${project.artifactId}:${project.version}</ima
         </configuration>
       </execution>
     </executions>
   </plugin>
```

En el apartado **properties** definimos:

- **docker.image.prefix:** que indica el prefijo de la imagen a generar

En el apartado **configuration** de la sección de plugins definimos los siguiente parámetros :

- **imageName:** Nombre de la imagen (prefijo + artifactId del proyecto)
- **dockerDirectory:** Directorio en el que se encuentra el Dockerfile definido anteriormente
- **serverId:** Identificador del registry de Docker (opcional: si queremos realizar un docker push a nuestro registry)
- **registryUrl:** URL del registry de Docker (opcional: si queremos realizar un docker push a nuestro registry)
- **imageTag:** Definimos las tags de nuestra imagen
- **resource:** Le indicamos el recurso que vamos a empaquetar dentro de la imagen ('targetPath' path base de los recursos, 'directory' directorio de los recursos, 'include' incluimos el jar resultante)

En el apartado **executions** vinculamos los goals del plugin a las fases de maven:

- **build-image:** Vinculamos a la fase package de maven, el goal docker:build que construye la imagen con el microservicio
- **build-image:** Vinculamos a la fase de install de maven, el goal docker:push que sube nuestra imagen al registro de docker

Una vez configurado podemos ejecutar alguno de los goals del plugin:

```shell
mvn clean package
```

```
[INFO] Building image jpacheco/micro-service-spring-boot
Step 1 : FROM frolvlad/alpine-oraclejdk8:slim
 ---> 7f321b30ec66
Step 2 : MAINTAINER jpacheco@autentia.com
 ---> Running in afea6b316da8
 ---> ac603658e8ce
Removing intermediate container afea6b316da8
Step 3 : ADD micro-service-spring-boot-0.0.1-SNAPSHOT.jar app.jar
 ---> 0ec77aba1beb
Removing intermediate container 701720a3929a
Step 4 : EXPOSE 8080
 ---> Running in cc4e8310e47b
 ---> 196f7135ca0d
Removing intermediate container cc4e8310e47b
Step 5 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
 ---> Running in 77cec2b3a7d3
 ---> 635454a48c9c
Removing intermediate container 77cec2b3a7d3
Successfully built 635454a48c9c
[INFO] Built jpacheco/micro-service-spring-boot
[INFO] Tagging jpacheco/micro-service-spring-boot with 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10.729 s
[INFO] Finished at: 2016-05-30T17:09:03+02:00
[INFO] Final Memory: 40M/375M
[INFO] ------------------------------------------------------------------------
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$
```

Como se puede ver en los logs, después de realizar el empaquetado se construye la imagen. Comprobamos si se han generado la imagen:

```shell
docker images
```

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$ docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
jpacheco/micro-service-spring-boot  0.0.1-SNAPSHOT      635454a48c9c        3 minutes ago       181.3 MB
jpacheco/micro-service-spring-boot  latest              635454a48c9c        3 minutes ago       181.3 MB
frolvlad/alpine-oraclejdk8          slim                7f321b30ec66        3 days ago          167.4 MB
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$
```

Podemos observar que en nuestro registro local, están disponibles tanto la imagen base de la que hemos partido **frolvlad/alpine-oraclejdk8:slim**,
como nuestra imagen con los tags 0.0.1-SNAPSHOT y latest. El siguiente paso es arrancar un contenedor a partir de nuestra imagen

```Shell
1 docker run -d -p 8080:8080 --name microservicio jpacheco/micro-service-spring-boot:0.0.1-SNA
```

Con esto arrancamos nuestro contenedor, podemos comprobarlo ejecutando

```Shell
1 docker ps
```

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$ docker ps
CONTAINER ID        IMAGE                                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
afe6a36dedb3        jpacheco/micro-service-spring-boot:0.0.1-SNAPSHOT   "java -Djava.security"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp   microservicio
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$
```

Una vez levantado el contenedor accedemos a nuestro servicio de manera análoga a la anterior sustituyendo 'localhost' por la IP de nuestro docker-machine

```Shell
1 curl http://192.168.99.100:8080/micro-service
```

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$ curl http://192.168.99.100:8080/micro-service
Hello from IP address: 172.17.0.2
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker
$
```

Podemos observar que la IP que devuelve es la IP interna del contenedor 172.17.0.2.

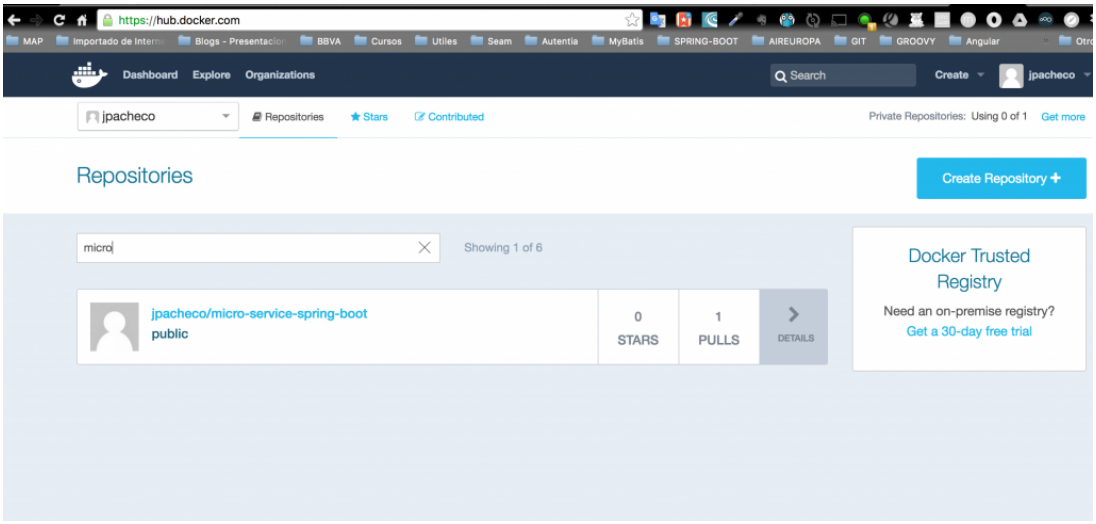
El último paso que nos quedaría para completar el ciclo seria realizar el 'push' de nuestra imagen a nuestro docker registry (es este ejemplo usaremos docker-hub,
como hemos definido en el pom.xml con los parámetros: serverId, registryUrl), nos faltaría añadir nuestras credenciales de docker-hub en el settins.xml de maven

```Shell
1     <server>
2       <id>docker-hub</id>
3       <username>myuser</username>
4       <password>mypassword</password>
5       <configuration>
6         <email>user@company.com</email>
7       </configuration>
8     </server>
```

Ya estamos listos para realizar un 'push' de nuestra imagen. Recordar que hemos vinculado el push a la tarea maven 'install'

```Shell
1 mvn install
```

Podemos acceder a nuestra cuenta de docker-hub y comprobar que se ha creado nuestra imagen



La cosa se empieza a poner interesante … ya tenemos nuestro microservicio empaquetado en un contenedor
y disponible en nuestro registry

5. Escalando la solución

El siguiente paso que vamos a estudiar, es como podemos lanzar varias instancias de nuestro microservicio
y como podemos balancear el trafico entre ellas.

Para esto vamos a usar **HAProxy** que es una herramienta open source que actúa como balanceador de carga (load balancer) ofreciendo alta disponibilidad,
balanceo de carga y proxy para comunicaciones TCP y HTTP.

Como no podía ser de otra manera, vamos a usar Docker para levantar el servicio HAProxy
Para ello vamos a usar docker-compose para definir tanto el microservicio, como el balanceador

```Shell
  microservice1:
    image: 'jpacheco/micro-service-spring-boot:latest'
    expose:
      - "8080"
  microservice2:
    image: 'jpacheco/micro-service-spring-boot:latest'
    expose:
      - "8080"
  loadbalancer:
    image: 'dockercloud/haproxy:latest'
    links:
      - microservice1
      - microservice2
    ports:
      - '80:80'
```

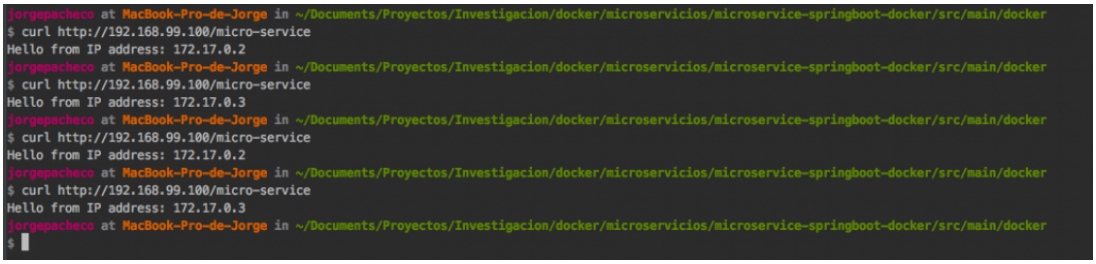
Como podemos ver en el fichero docker-compose.yml, hemos definido 2 instancias de nuestro microservicio (microservice1, microservice2) y un balanceador (loadbalancer) con enlaces a
los microservicios definidos anteriormente. Lo que conseguimos con esta imagen de HAProxy es exponer el puerto 80 y

redirigir a los 2 microservicios expuestos en el 8080 usando una estrategia round-robin. Levantamos nuestro entorno con:

```Shell
docker-compose up -d
```

y podemos observar como se levantan los 3 contenedores

Vamos a invocar a nuestro microservicio a través del balanceador:



Como podemos observar en la consola, el balanceador va accediendo cada vez a una instancia del microservicio, logrando el balanceo de carga que íbamos buscando … No está mal no?, el ejemplo va tomando 'cuerpo'.

Pero.. ¿y si queremos levantar más instancias de nuestro microservicio? ¿Tenemos que modificar el docker-compose.yml, y añadir 'microservice3…microserviceN'?

Revisando la documentación de la imagen HAProxy encontramos una solución a esta problemática, la idea es levantar una primera instancia del balanceador y del microservicio y posteriormente en función de las necesidades, levantar más instancias del microservicio y que el balanceador se reconfigure para añadirlas. Veamos como quedaría el docker-compose.yml

```Shell
version: '2'
services:
   microservice:
    image: 'jpacheco/micro-service-spring-boot:latest'
    expose:
      - '8080'
   loadbalancer:
    image: 'dockercloud/haproxy:latest'
    links:
      - microservice
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - '80:80'
```

Repasemos las lineas más destacadas:

- **version: '2'** estamos indicando que use la v2 de docker-compose (necesaria para este ejemplo)
- **service:** Tag raíz del que cuelgan nuestros contenedores
- **microservice: loadbalancer** Definición de nuestros contenedores
- **volumes:** El contenedor de HAProxy necesita acceder al 'docker socket' para poder detectar nuevas instancias y reconfigurarse

Vamos a levantar nuestros contenedores:

```Shell
docker-compose -f docker-composeV2.yml up -d
```

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$ docker-compose -f docker-composeV2.yml up -d
Creating docker_microservice_1
Creating docker_loadbalancer_1
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$
```

Vemos como se han levantado una instancia del balanceador y otra del microservicio. Ahora vamos a escalar nuestro microservicio añadiendo 2 instancias más:

```Shell
docker-compose -f docker-composeV2.yml scale microservice=3
```

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$ docker-compose -f docker-composeV2.yml scale microservice=3
Creating and starting docker_microservice_2 ... done
Creating and starting docker_microservice_3 ... done
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$
```

Comprobamos que se han creado 2 nuevas instancias de nuestro microservicio, ahora vamos a probar que estas instancias se han añadido al balanceador:

```
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$ curl http://192.168.99.100/micro-service
Hello from IP address: 172.19.0.2
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$ curl http://192.168.99.100/micro-service
Hello from IP address: 172.19.0.4
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$ curl http://192.168.99.100/micro-service
Hello from IP address: 172.19.0.5
jorgepacheco at MacBook-Pro-de-Jorge in ~/Documents/Proyectos/Investigacion/docker/microservicios/microservice-springboot-docker/src/main/docker
$
```

Como podemos ver, cada petición es atendida por una instancia distinta .... podríamos ir añadiendo instancias según vayamos necesitando, bastaría con ejecutar **docker-compose -f docker-composeV2.yml scale microservice=<Instancias_vivas+Nuevas>**

6. Conclusiones

Como hemos podido ver a lo largo del tutorial, la combinación de Spring Boot y Docker nos permite desarrollar facilmente microservicios, incluso montar infraestructuras que permitan su escalabilidad. El siguiente paso sería investigar la posibilidad de escalarlo a través de un cluster de máquinas usando herramientas como **Docker Swarn** o **Kubernetes**, pero eso os lo dejo a vosotros 😉

Un saludo.

7. Referencias

- HAProxy
- Imagen HAProxy
- docker-maven-plugin
- Spring Boot
- Fuentes

Jorge Pacheco Mengual

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/Java EE