VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria

Máster Universitario en Tecnologías Web, Computación en la Nube y Aplicaciones Móviles

Computación en la Nube

Trabajo final

Curso 2018-2019

LUIS ARAUJO

Contenido

Introducción

El siguiente trabajo, describe detalladamente el proceso para desplegar servicios en contenedores Docker que a su vez están conectados a otros contenedores para gestionar la persistencia de datos. Para llevar dicho escenario a un ambiente más real, todo el despliegue se ejecutará en una infraestructura como servicio, en concreto OpenStack.

La arquitectura del despliegue consistirá en dos redes, la primera que da acceso al exterior de la infraestructura (red externa) y la segunda que se comunica con una instancia o máquina virtual que tiene instalado Docker. Esa máquina virtual tendrá 4 contenedores:

- El primero será un servicio que requiere acceso a persistencia relacional y que tiene un puerto abierto para poder acceder.
- Otro contenedor que se utiliza como servidor de una base de datos relacional (MySQL).
- Luego, un contenedor que será otro servicio que requiere acceso a persistencia no relacional y que tiene un puerto abierto para poder acceder.
- Finalmente, un contenedor que se utiliza como servidor de una base de datos no relacional (MongoDB).

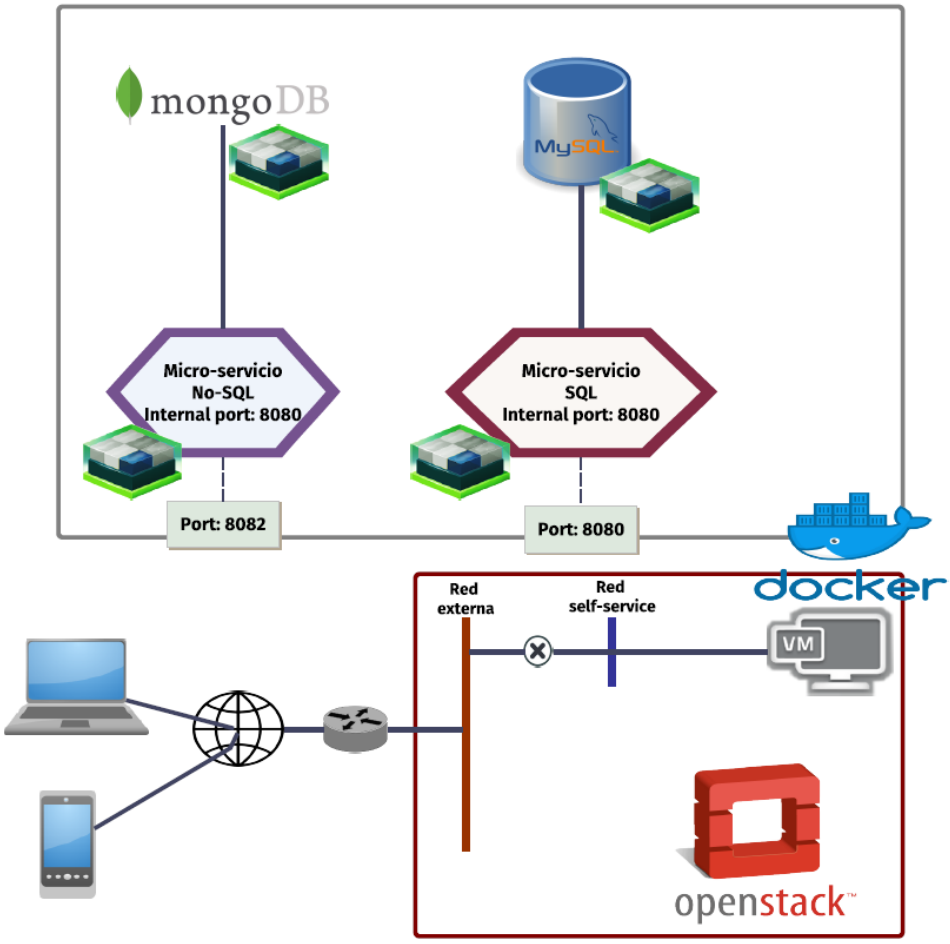

Figura 1: Aquitectura del despliegue

Procedimiento

Empaquetar Servicios

En primer lugar, se han desarrollado dos aplicaciones en Spring Boot (API REST) que acceden a persistencia relacional (MySQL) y no relacional (MongoDB). Ambas aplicaciones se han gestionado con Maven y para su empaquetamiento se ha definido configuración relevante en sus ficheros *application.properties*. Entre ellos, la cadena de conexión a la base de datos que será el nombre del servicio que se ha definido para los contenedores de MySQL y MondoDB respectivamente y el número de puerto de cada servicio.

```
spring.datasource.url=jdbc:mysql://mysql-
container:3306/airlinesdb
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-
platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.generate-ddl=true
server.port=8082
```

Figura 2: fichero *aplication-properties* del servicio mysql-app

```
spring.data.mongodb.host=mongo-server-docker
spring.data.mongodb.port=27017
spring.data.mongodb.database=shops
server.port=8083
```

Figura 3: fichero *aplication-properties* del servicio mysql-app

Crear imágenes de servicios

Luego, se han creado imágenes Docker de los servicios *mysql-app* y *mongo-app*. En cada contenedor se copia el fichero .jar y luego se define el comando para ejecutar la aplicación.

```
FROM openjdk:8
LABEL maintainer="Luis Araujo"
ADD target/airlines-app-0.0.1-SNAPSHOT.jar airlines-app-
0.0.1.jar
EXPOSE 8082
ENTRYPOINT [ "java", "-jar", "airlines-app-0.0.1.jar" ]
```

Figura 4: Dockerfile de servicio *mysql*-app

```
FROM openjdk:8
LABEL maintainer="Luis Araujo"
ADD target/shops-app-0.0.1-SNAPSHOT.jar shops-app-0.0.1.jar
EXPOSE 8083
ENTRYPOINT [ "java", "-jar", "shops-app-0.0.1.jar" ]
```

Figura 5: Dockerfile de servicio *mongo-app*

Cada imagen de los servicios se ha subido al repositorio que se encuentra en *twcammaster.uv.es* que luego, serán descargadas por *docker swarm* cuando se desplieguen los servicios.

```
# Comando ejecutado en directorio del servicio mysql-app
$ docker build -t twcammaster.uv.es/proyecto-8-mysql-app .
$ docker push twcammaster.uv.es/proyecto-8-mysql-app

#Comando ejecutado en directorio del servicio mongo-app
$ docker build -t twcammaster.uv.es/proyecto-8-mongo-app .
$ docker push twcammaster.uv.es/proyecto-8-mongo-app
```

Figura 6: Comandos para generar imágenes Docker de los servicios y subirlas al repositorio

Fichero docker-cloud

Una vez creada las imágenes de los servicios, se creó un fichero *docker-cloud.yml* que será utilizado por *docker swarm* para desplegar los servicios:

```
version: "3.7"
services:
  mongo-app:
    image: twcammaster.uv.es/proyecto-8-mongo-app
    ports:
      - target: 8083
        published: 8083
    depends_on:
      - mongo-server-docker
    networks:
      - internal
  mongo-server-docker:
    image: mongo:3.4.2
    ports:
      - "27017:27017"
    command: mongod
    networks:
      - internal
```

```
  mysql-app:
    image: twcammaster.uv.es/proyecto-8-mysql-app
    ports:
      - target: 8082
        published: 8082
    depends_on:
      - mysql-container
    networks:
      - internal
  mysql-container:
    image: mysql:5.6
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=airlinesdb
    ports:
      - "3306:3306"
    networks:
      - internal
networks:
  internal:
```

Figura 7: Fichero docker-cloud.yml

Principalmente, este fichero define los 4 servicios necesarios con sus respectivas imágenes, puertos y red. Para los servicios *mysql-app* y *mongo-app* se establecen los servicios dependientes para ser desplegados, en este caso dependen del servicio *mysql-container* y *mongo-server-docker* respectivamente.

Crear infraestructura en OpenStack

Luego, se ha creado la infraestructura en OpenStack definida en la *Figura 1*. Para eso, se ha utilizado un contenedor *ocata-cli* para tener acceso a la API de OpenStack.

```
$ openstack network create $OS_NETWORK
$ openstack subnet create --subnet-range 10.2.0.0/24 --network
  $OS_NETWORK --dns-nameserver 8.8.4.4 $OS_SUBNET
$ openstack router create $OS_ROUTER
$ openstack router add subnet $OS_ROUTER $OS_SUBNET
$ neutron router-gateway-set $OS_ROUTER external-network
$ openstack security group create $OS_SECURITY_GROUP
$ openstack security group rule create --proto tcp --dst-port
22 $OS_SECURITY_GROUP
$ openstack security group rule create --proto tcp --dst-port
8082 $OS_SECURITY_GROUP
```

```
$ openstack security group rule create --proto tcp --dst-port
8083 $OS_SECURITY_GROUP
$ openstack security group rule create --proto tcp --dst-port
2376 $OS_SECURITY_GROUP
```

Figura 8: Comandos para crear infraestructura.

Los comandos definen la creación de una red y una subred, también un *router* que está conectado a la red externa y la red que se ha creado. Se establece un grupo se seguridad y se abren los puertos necesarios, el 22 para hacer SSH con docker-machine, el 8082 que es el puerto del servicio mysql-app y 8083 del servicio mongo-app. Además, se ha abierto el puerto 2376, porque hubo problemas de generación de certificados al cargar las variables de entorno de *docker machine*[1].

Docker Machine y Docker Swarm

Finalmente, se crea la máquina virtual con *docker-machine*. Previamente se establecieron las variables de entorno de la API de OpenStack.

```
$ docker-machine create \
                  --openstack-flavor-name $OS_FAVLOR \
                  --openstack-image-name $OS_IMAGE \
                  --openstack-net-name $OS_NETWORK \
                  --openstack-floatingip-pool external-network \
                  --openstack-ssh-user ubuntu \
                  --openstack-tenant-name proyecto8 \
                  --openstack-sec-groups $OS_SECURITY_GROUP \
                  --openstack-domain-name Default \
                  --driver openstack \
                  $VM_NAME
```

Figura 9: Creación de máquina virtual con docker machine.

Seguidamente, se cargan las variables de entorno de la máquina virtual creada, se inicia *docker swarm* con la dirección IP de la máquina y se despliegan los servicios con el fichero *docker-cloud* creado previamente.

```
$ docker-machine env $VM_NAME
$ ip=$(docker-machine ssh $VM_NAME ifconfig ens3 | grep -Eo "inet
(addr:)?([0-9]*\.){3}[0-9]*" | grep -Eo "([0-9]*\.){3}[0-9]*")
$ docker swarm init --advertise-addr $ip
$ docker stack deploy --compose-file docker-cloud.yml services-
app
```

Figura 10: Comandos para desplegar servicios.

[1] https://stackoverflow.com/questions/48602329/creating-new-docker-machine-instance-always-fails-validating-certs-using-opensta

Conclusiones

- Las tecnologías de virtualización crecen rápidamente y docker es una herramienta muy potente que hace más fácil el despliegue de múltiples servicios.
- La contenedorización gana cada vez más atención porque es posible desarrollar software más rápido, más barato y mejor.
- Este enfoque de despliegue tiene en cuenta la integración, entrega e implementación continua y que ahora se conoce como DevOps.