CLOUD COMPUTING: SERVICIOS Y APLICACIONES

MÁSTER EN INGENIERÍA INFORMÁTICA

Uso de contenedores (Docker)

Práctica 2

Autor

Juan Pablo Porcel Porcel - juanpiporcel@correo.ugr.es

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 9 de abril de 2017

Índice

1. Introducción

El objetivo de esta práctica es familiarizarse con el uso de una plataforma PaaS y desarrollar habilidades de despliegue de contenedores y configurar aplicaciones sencillas en los mismos.

Para ello el alumno deberá realizar las tareas que se describen a continuación y entregar documentación describiendo con el mayor detalle posible todas las actividades realizadas.

- Crear un contenedor docker con un servidor web con SSL.
  1. ¿Cuál es el puerto SSL?
  2. ¿Cómo redirigir el puerto SSL a vuestro puerto asignado?
- Crear una página en el servidor web que se conecte a un servicio de base de datos en otro contenedor.
- Duplicar los contenedores y discutir o mostrar qué pasaría si uno de ellos cayese.
- Desplegar un servicio OwnCloud o NewCloud en otro contenedor y chequear su correcto funcionamiento almacenando archivos.
- Elaborar un breve documento detallando el trabajo realizado.

2. Configuración de los contenedores docker

Para esta práctica se han creado tres contenedores docker para desplegar la aplicación web creada para la primera práctica. En la Figura **??** se puede ver un esquema de la conexión entre los tres contenedores.

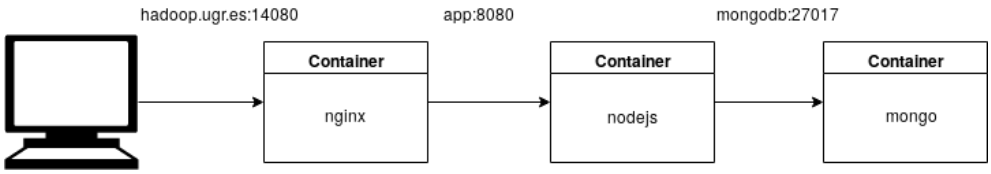

Figura 2.1: Conexión entre los contenedores.

El despliegue de la aplicación se puede llevar a cabo con docker-compose o siguiendo los pasos que se detallan en las siguientes subsecciones para trabajar sólamente con Docker.

El archivo de configuración para docker-compose se muestra a continuación.

```yaml
version: '3'

services:
  jpblo_nginx:
    build:
      context: ./nginx
      dockerfile: Dockerfile
    container_name: jpblo_nginx
    ports:
      - "14080:80"
      - "14083:443"
    links:
      - "jpblo_app:app"

  jpblo_app:
    build:
      context: ./app
      dockerfile: Dockerfile
    container_name: jpblo_app
    ports:
      - "14082:8080"
    links:
      - "jpblo_mongo:mongodb"

  jpblo_mongo:
    image: mongo
    container_name: jpblo_mongo
    ports:
      - "14081:27017"

  mongo-seed:
    build: ./mongo
    links:
      - "jpblo_mongo:mongodb"
```

Mediante este fichero de configuración se han especificado la redirección de puertos para cada contenedor y la conexión entre ellos mediante la etiqueta **links**. Los ficheros dockerfile de configuración para los dos primeros contenedores se detallan más adelante. El contenedor mongo-seed se usa para cargar en el contenedor con el sistema gestor de base de datos la base de datos de prueba.

2.1. SGBD

Antes de nada vamos a clonar el repositorio de la asignatura con la aplicación web y los archivos de configuración y provisionamiento.

```
git clone https://github.com/JPPorcel/CCSA.git ./app
```

El primer contenedor tendrá alojado un sistema gestor de base de datos. Para desplegar este contenedor se ha elegido la imagen de **mvertes/alpine-mongo** alojada en Docker-Hub debido a que la versión oficial de mongo producía un error al desplegar el contenedor en *hadoop.ugr.es*. Podemos crear el contenedor con el siguiente comando:

```
docker run -d -p 14081:27017 --name jpblo_mongo mvertes/alpine-
    mongo
```

Una vez que el contenedor esté corriendo podemos conectarnos a la base de datos mongo para comprobar que todo ha ido bien con:

```
docker exec -i -t jpblo_mongo mongo
```

Ahora vamos a importar la base de datos de prueba a el contenedor con **mongoimport**. Tenemos que conectarnos por el puerto que hemos especificado en docker (14081). El archivo restaurantes.json se encuentra en la ruta P2/mongo/ dentro del repositorio.

```
mongoimport --host 127.0.0.1 --port 14081 --db restaurants --
    collection restaurants --drop --file restaurantes.json
```

2.2. Aplicación Web

Ahora vamos a desplegar el contenedor con la aplicación web. Para ello usaremos el **Dockerfile** que se encuentra en P2/app/:

```
FROM keymetrics/pm2-docker-alpine

# Create app directory
RUN mkdir -p /opt/app
WORKDIR /opt/app

# Install app dependencies
COPY package.json /opt/app/
RUN npm install

```

```
# Bundle app source
COPY . /opt/app

EXPOSE 8080

CMD pm2 start --no-daemon server.js
```

La imagen base para este contenedor es **keymetrics/pm2-docker-alpine** que tiene ya instalado NodeJS y pm2 para administrar la aplicación. Mediante este dockerfile añadimos los fuentes de la aplicación y exponemos el puerto 8080 que será el puerto de escucha de la aplicación. Por último lanzamos la aplicación con $pm2 start --no-daemon server.js$.

Desde el mismo directorio donde se encuentra este dockerfile podemos contruir la imagen con:

```
cd app/P2/app && docker build -t jpblo/webapp .
```

Y lanzarlo especificando la redirección del puerto 8080 a uno de los puertos disponibles y especificando el nombre del host donde se encuentra la base de datos mongo **jpblo_mongo:mongodb**:

```
docker run -d --link jpblo_mongo:mongodb -p 14082:8080 --name
    jpblo_app jpblo/webapp
```

2.3. Servidor Web

Para desplegar el contenedor con el servidor web **nginx** usaremos el siguiente Dockerfile que se encuentra en la ruta P2/nginx/ dentro del repositorio.

```
FROM nginx

# Copy custom configuration file from the current directory
COPY nginx.conf /etc/nginx/nginx.conf
```

Este dockerfile carga la imagen oficial de nginx y añade el archivo de configuración de nginx al contenedor.

```
worker_processes auto;

events { worker_connections 1024; }

```

```
http {
  server {
      listen 80;

      location / {
      proxy_pass http://app:8080;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
      }
  }
}
```

Podemos contruir la imagen para este contenedor con:

```
cd app/P2/nginx && docker build -t jpblo/nginx .
```

Y lanzarlo especificando de nueco la redirección del puerto 80 a uno de los puertos disponibles, así como el puerto SSL (**14083:443**), y el nombre del host donde se encuentra la aplicación web **jpblo_app:app**:

```
docker run -d --link jpblo_app:app -p 14080:80 -p 14083:443 --
   name jpblo_nginx jpblo/nginx
```

El servidor web nginx escuchará en el puerto 80 y redireccionará las peticiones a donde se encuentra nuestra aplicación corriendo **app:8080**.

Con nginx también podemos realizar un balanceo de carga replicando los contenedores que contienen la aplicación y añadiendo sus direcciones al archivo de configuración de nginx. Un ejemplo podría ser el que se muestra en el siguiente fichero de configuración de nginx con tres contenedores con la aplicación replicados.

```
worker_processes 4;

events { worker_connections 1024; }

http {
  upstream node-app {
    least_conn;
    server node1:8080 weight=10 max_fails=3 fail_timeout=30s;
    server node2:8080 weight=10 max_fails=3 fail_timeout=30s;
```

```
10      server node3:8080 weight=10 max_fails=3 fail_timeout=30s;
11    }
12
13    server {
14      listen 80;
15
16      location / {
17        proxy_pass http://node-app;
18        proxy_http_version 1.1;
19        proxy_set_header Upgrade $http_upgrade;
20        proxy_set_header Connection 'upgrade';
21        proxy_set_header Host $host;
22        proxy_cache_bypass $http_upgrade;
23      }
24    }
25  }
```

De esta manera conseguimos que el trabajo esté repartido entre los tres contenedores y que si falla alguno no deje de estar la aplicación inoperativa.

3. Aplicación web

En la Figura 3.1 se puede ver una captura del servidor corriendo en la dirección hadoop.ugr.es:14080.

La aplicación está escrita en NodeJS con Express y Pug y usa una base de datos Mongo para almacenar la información. Es una aplicación muy sencilla. Se trata de un gestor de restaurantes donde se pueden añadir nuevos restaurante desde la opción de añadir que nos mostrará un formulario como el de la Figura 3.2.

En la sección de Listar nos aparecen los primeros 20 restaurantes de la base de datos. Podemos filtrar los restaurantes por nombre, ciudad o tipo de cocina. El buscador que se encuentra implementado en la página buscará por tipo de cocina pero se pueden añadir más parámetros en la url, por ejemplo *cocina=Granaina&nombre=Julio&ciudad=Granada*. En las Figura 3.3 y Figura 3.4 vemos la búsqueda y consulta de un restaurante añadido a la base de datos.

4. OwnCloud

Para poner en marcha OwnCloud en hadoop.ugr.es con docker usaremos dos contenedores. El primero contendrá la aplicación de OwnCloud y el segundo alojará una base

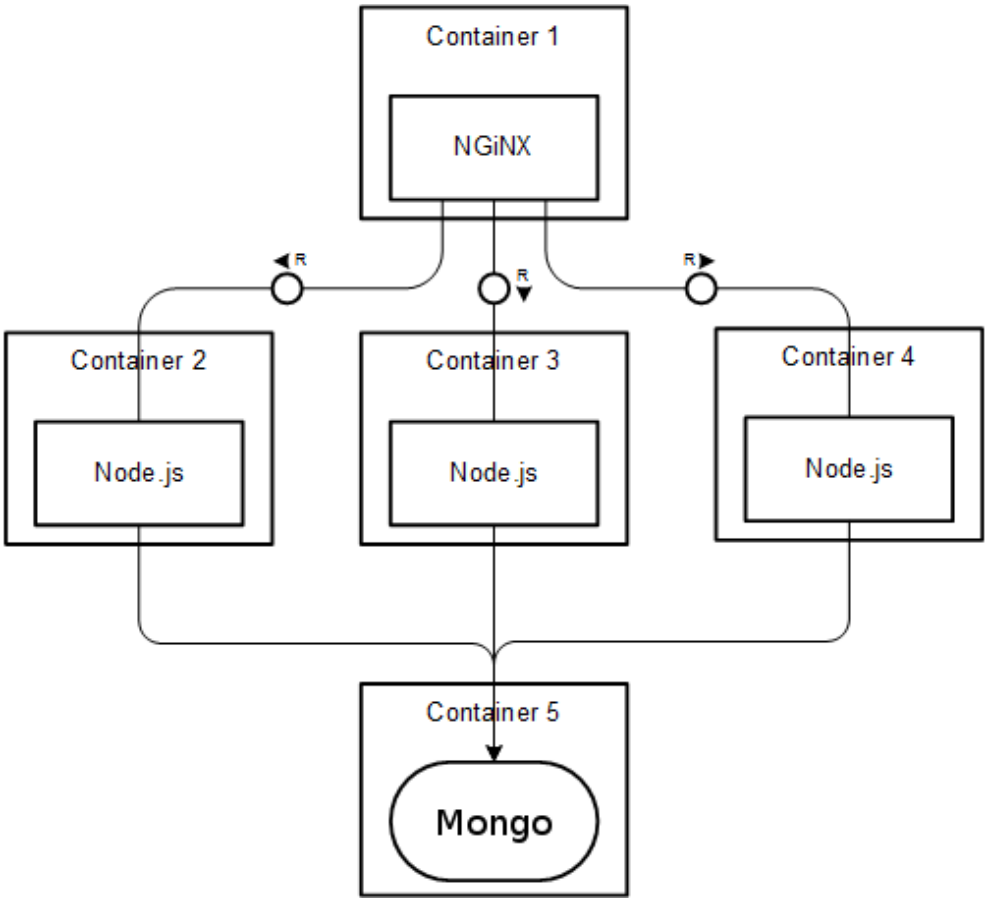

Figura 2.2: Aplicación web replicada en tres contenedores

de datos PostgreSQL para conseguir persistencia en los datos de la aplicación, como las cuentas de usuario y los índices a los archivos.

Podemos desplegar ambos contenedores con:

```
1 docker run -d --name jpblo_postgres -e POSTGRES_PASSWORD=
    password postgres
2 docker run -d -p 14084:80 --name jpblo_owncloud --link
    jpblo_postgres:owncloud-db owncloud
```

Ahora podemos acceder a `hadoop.ugr.es:14084` y configurar el usuario administrador de OwnCloud y la base de datos que se va a utilizar como aparece en la Figura 4.1.

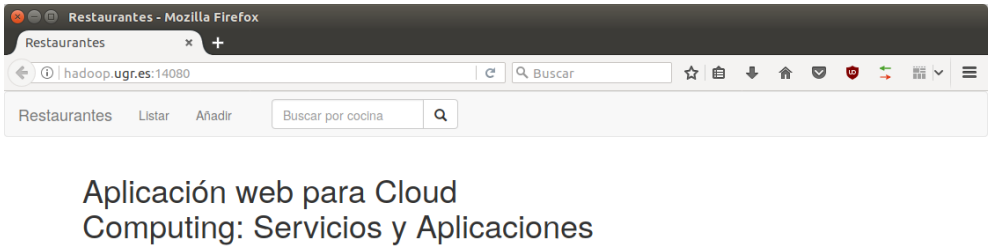

Figura 3.1: Aplicación web corriendo en hadoop.ugr.es:14080

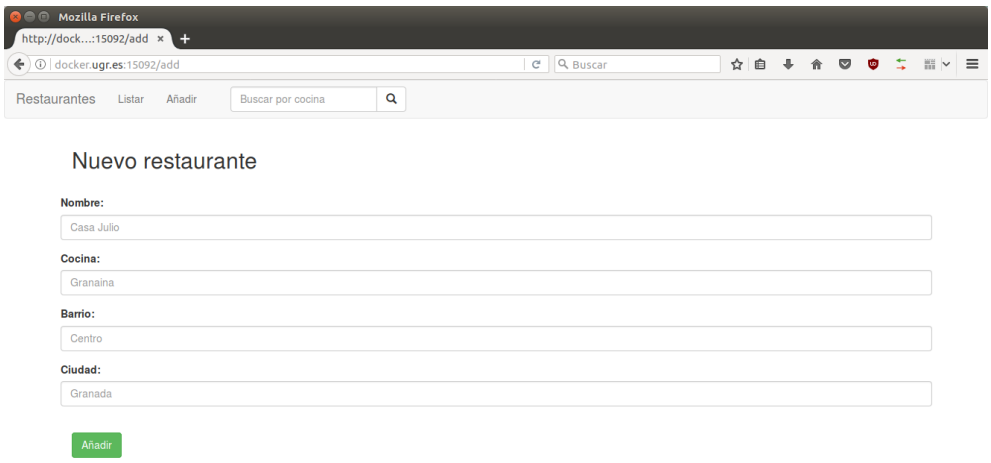

Figura 3.2: Formulario para añadir un nuevo restaurante

5. Conclusiones

Mediante esta práctica se ha intentado familiarizar al alumno con el uso de una plataforma PaaS. Al contrario de la primera práctica, en esta segunda práctica no se han tenido problemas de conexión con la plataforma y se ha podido realizar la práctica correctamente.

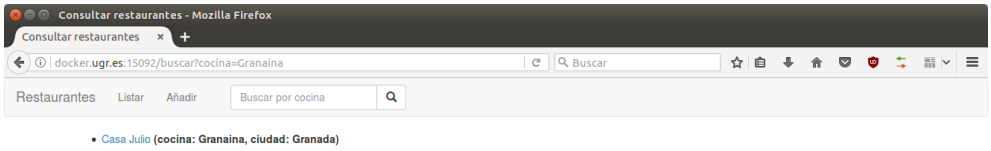

Figura 3.3: Buscar un restaurante

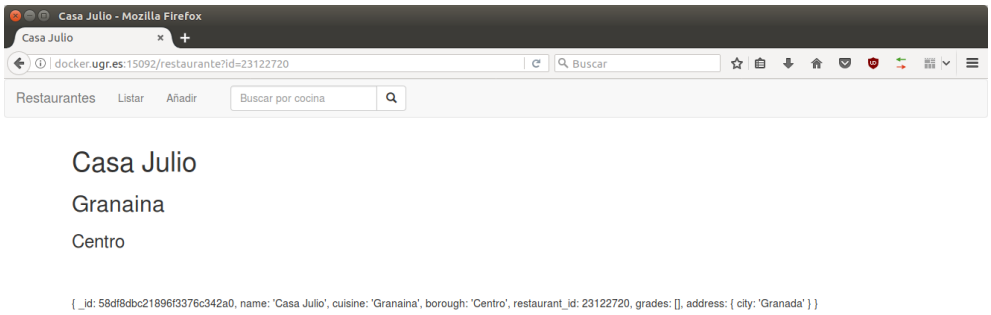

Figura 3.4: Consultar un restaurante

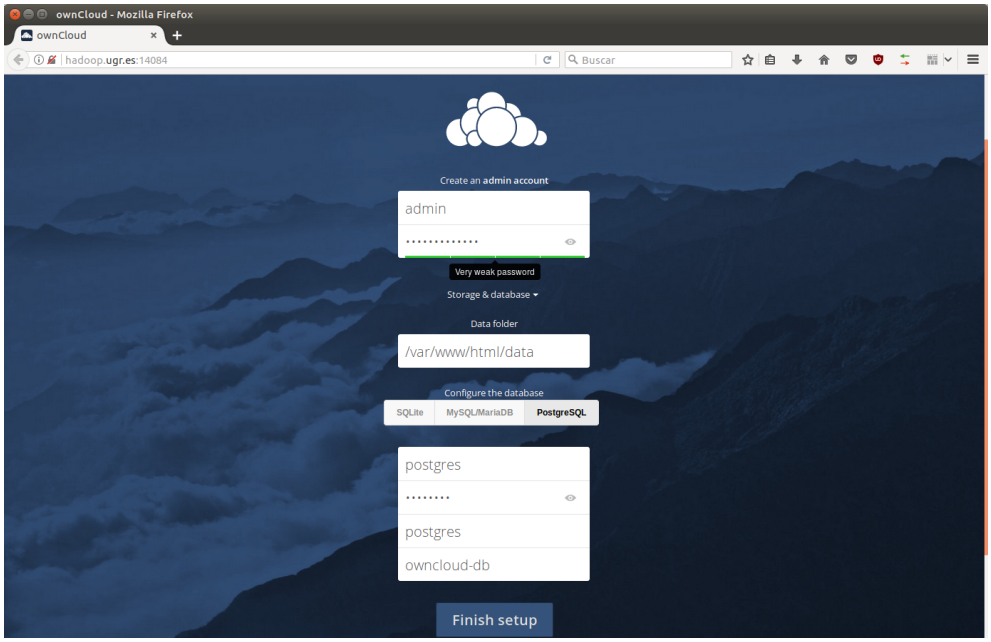

Figura 4.1: Configuración de la base de datos PostgreSQL en OwnCloud

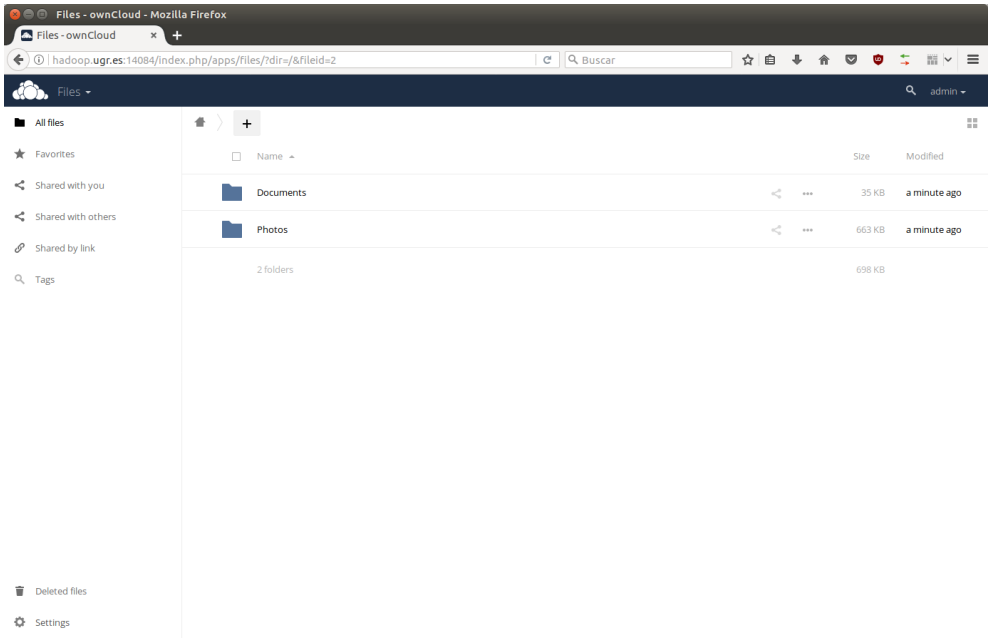

Figura 4.2: Aplicación OwnCloud corriendo en hadoop.ugr.es:14084

El uso de contenedores es una tecnología muy usada hoy en día para el despliegue de aplicaciones web ya que conseguimos abstraernos del sistema operativo donde se ejecutará la aplicación quitándonos problemas de compatibilidad y provisionamiento. También, cabe destacar que el balanceo de carga es mucho más sencillo llevarlo a cabo con contenedores que con máquinas virtuales.