Desafío 3: Rueda de coches (Backend)

Autores:

- Rodrigo Navas.
- Jorge Olmo.
- Alejandro Martín

Índice

1. Objetivo

El objetivo es explicar el proceso de despliegue del proyecto basado en el **Desafío 3** usando la tecnología de Dockers.

2. Requisitos previos

Para el despliegue de la aplicación es necesario tener docker instalado en nuestro sistema.

El desarrollo y pruebas de este despliegue han sido realizadas en los sistemas: **MacOS, Kubuntu y Ubuntu.**

3. Despliegues

El despliegue de esta parte de la aplicación (**Backend**), se realizará mediante el uso de Docker.

¿Qué es Docker?

“Docker es una tecnología basada en contenedores que permite aislar la infraestructura donde se ejecuta nuestra aplicación de la infraestructura que contiene los contenedores de la aplicación”

Los contenedores se pueden ejecutar en cualquier sistema sin necesidad de tener que instalar / modificar archivos de configuración no deseados que puedan provocar el mal funcionamiento de otras aplicaciones de nuestro sistema.

3.1 Despliegue para el entorno servidor (Backend)

Para el despliegue de la parte de Backend lanzaremos un script según el tipo de sistema operativo que tengamos, que se podrá encontrar en este mismo directorio.

3.1.1 Docker-compose

La ejecución de este fichero generará los contenedores con los datos y una vez generados los contenedores lanzarán su ejecución para poder empezar a usar la aplicación.

```yaml
# Fichero de configuración para los contenedores del servidor de la aplicación CarShare
version: "3.7"
services:

  # Crea un proxy inverso
  carshare-serverproxy:
    # Imagen base para generar el contenedor
    image: jwilder/nginx-proxy
    # Nombre asignado al contenedor
    container_name: carshare-serverproxy
    # Puertos abiertos para acceder al contenedor
    # Asigno el 81 para que no haya conflictos con el ciente
    ports:
      - 81:80
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - carshare-server
    privileged: true


  # Contenedor para la persistencia de datos
  carshare-db:
    # La imagen del contenedor está preparada en dockerhub
    image: rodrigo81/carshare-db
    # Nombre asignado al contenedor
    container_name: carshare-db
    # Red para el conjunto de contenedores
    ports:
      - 3306:3306
    #La ruta del volumen debe apuntar a la carpeta que contiene los datos de la base de datos
    volumes:
      - database-carshare:/var/lib/mysql
    #Variables de entorno  para el contenedor
    env_file:
      - .env.example
    networks:
      - carshare-server
    depends_on:
        - carshare-serverproxy
```

```yaml
# Contenedor para el servidor laravel
  carshare-server:
    # La imagen del contenedor se construye sobre una adecuando las necesidades
    build: .
    # Nombre asignado al contenedor
    container_name: carshare-server
    # Variable de entorno para el contendor
    environment:
      # Esta ruta debe estar incluida en el fichero hosts del anfitrión
      VIRTUAL_HOST: carshare.server.local
    depends_on:
        - carshare-db
        - carshare-serverproxy
    networks:
        - carshare-server
    # Se mapea la carpeta con una local
    volumes:
        - .:/var/www/html/
    # restart: always

networks:
    carshare-server:
        driver: bridge
volumes:
    database-carshare:
```

3.1.2 Proxy Inverso

```
# Crea un proxy inverso
carshare-serverproxy:
  # Imagen base para generar el contenedor
  image: jwilder/nginx-proxy
  # Nombre asignado al contenedor
  container_name: carshare-serverproxy
  # Puertos abiertos para acceder al contenedor
  # Asigno el 81 para que no haya conflictos con el ciente
  ports:
    - 81:80
  volumes:
    - /var/run/docker.sock:/tmp/docker.sock:ro
  networks:
    - carshare-server
  privileged: true
```

Este contenedor de nombre: **carshare-serverproxy**, es el encargado de redirigir el tráfico, de manera que llevará los paquetes desde un origen a un destino en concreto según a donde vayan dirigidos.

- Lo que hacemos es partir de una imagen base de **Nginx**, este servidor web será el que actuará de proxy inverso.

- Mapeamos los puertos de manera que podamos acceder al contenedor. Hay que tener cuidado de que no sean los mismos puertos que el cliente

- Mapeamos el volumen de datos que será donde se almacenarán los datos.

- Lo añadimos a una red para aislarlo del resto de contenedores que tengamos y pueda comunicarse entre miembros de su red.

- Le decimos que también va a ser privilegiado, lo que hace es que va a estar al mismo nivel que nuestro sistema operativo y escogerá este contenedor como proxy inverso por defecto.

3.1.3 Base de datos

```
# Contenedor para la persistencia de datos
carshare-db:
  # La imagen del contenedor está preparada en dockerhub
  image: rodrigo81/carshare-db
  # Nombre asignado al contenedor
  container_name: carshare-db
  # Red para el conjunto de contenedores
  ports:
    - 3306:3306
  #La ruta del volumen debe apuntar a la carpeta que contiene los datos de la base de datos
  volumes:
    - database-carshare:/var/lib/mysql
  #Variables de entorno  para el contenedor
  env_file:
    - .env.example
  networks:
    - carshare-server
  depends_on:
      - carshare-serverproxy
```

Este contenedor será el encargado de la base de datos, será el que se comunique con la aplicación en laravel mediante el proxy inverso, y este será el encargado de redirigir la respuesta hacia el cliente.

- Lo que hacemos es partir de una imagen ya creada.

- Mapeamos los puertos para que puedan usarse por nuestra aplicación servidor

- Montamos el volumen, hay que tener en cuenta que el volumen tiene que apuntar a la carpeta donde están almacenados los datos de la base de datos.

- Ponemos la red a la que va a estar conectada y por ultimo le decimos que depende del proxy inverso

3.1.3 Laravel

```
# Contenedor para el servidor laravel
carshare-server:
  # La imagen del contenedor se construye sobre una adecuando las necesidades
  build: .
  # Nombre asignado al contenedor
  container_name: carshare-server
  # Variable de entorno para el contendor
  environment:
    # Esta ruta debe estar incluida en el fichero hosts del anfitrión
    VIRTUAL_HOST: carshare.server.local
  depends_on:
      - carshare-db
      - carshare-serverproxy
  networks:
      - carshare-server
  # Se mapea la carpeta con una local
  volumes:
      - .:/var/www/html/
  # restart: always
```

Lo que hace este contenedor con nombre: **carshare-server** es lanzar la API de nuestro cliente web. Partiremos una imagen base a la cual mediante un archivo **Dockerfile** le instalamos unos ciertos programas y lanzamos unos comandos para que sea accesible desde nuestro navegador y se haga un despliegue correcto.

- Primero mandaremos la instrucción **build** que será la encargada de compilar este archivo **Dockerfile**.

- Le añadimos unas variables de entorno a nuestro contenedor.

- Mapeamos el volumen de datos, de manera que todos los cambios que hagamos se vean reflejados en el contenedor.

- Lo añadimos a la red, ya que se tienen que comunicar entre la base de datos y el proxy inverso.

- Por último le decimos que depende del contenedor **carshare-serverproxy** (Proxy inverso), y de **carshare-db** (base de datos) de manera que, hasta que el contenedor del Proxy Inverso y la base de datos no se hayan sido lanzado, este no se lanzará.

3.1.4 Dockerfile

```
#Desacrga la imagen
FROM php:apache-buster

#Fichero para solucionar los problemas de rutas en Laravel
COPY ./000-default.conf /etc/apache2/sites-available/

    # Actualiza el sistema
RUN apt-get --yes update && apt-get --yes upgrade && apt --yes autoremove && apt-get --yes clean\
    # Instala Composer
    curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer &&\
    # Instala las dependencias necesarias
    apt-get install -y libmcrypt-dev openssl zip unzip git &&\
    # Instala las necesidades para la persistencia de datos
    docker-php-ext-install mysqli pdo_mysql  &&\
    docker-php-ext-enable mysqli pdo_mysql  &&\
    # Recarga el vhost para aplicar los cambios necesarios
    a2enmod headers  &&\
    a2enmod rewrite  &&\
    service apache2 restart  &&\
    # Limpia los residuos de las instalaciones
    apt-get clean

# Directorio de trabajo
WORKDIR /var/www/html/

# Abre el puerto para el acceso
EXPOSE 80
```

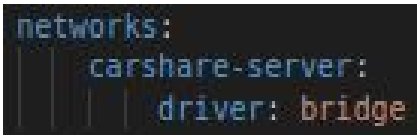
En este Dockerfile lo que hacemos es partir de una imagen base de **Apache con PHP**, le decimos que copie la configuración nueva de nuestro servidor Apache preparado para lanzar nuestra aplicación. Lanzamos las órdenes para actualizar el sistema y seguidamente que limpie los archivos en desuso.

Una vez actualizado y limpiado el sistema de ficheros en desuso procedemos a instalar **composer**, las dependencias necesarias y las utilidades para la persistencia de datos, en este caso, usaremos un sistema **MySQL**. Recargamos el Virtual Host para aplicar los cambios necesarios y por último limpiamos los residuos de las instalaciones.

Le decimos cuál será el espacio de trabajo con la orden **WORKDIR**

Para finalizar, ponemos el puerto de escucha que nos permitirá el acceso.

3.1.5 Network



Una vez llegados a este punto del **docker-compose.yml,** lo que hacemos es crear una red, la cual será la que vamos a usar en nuestros contenedores, y le decimos que actúe en modo puente.

4. Github y Gitkraken

Github

https://github.com/rnavas81/ServidorRuedas

Gitkraken

https://app.gitkraken.com/glo/board/YAgdJxxU7wARZgGv