Desafío 3: Rueda de coches (Frontend)

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 (**Frontend**), 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 cliente (Frontend)

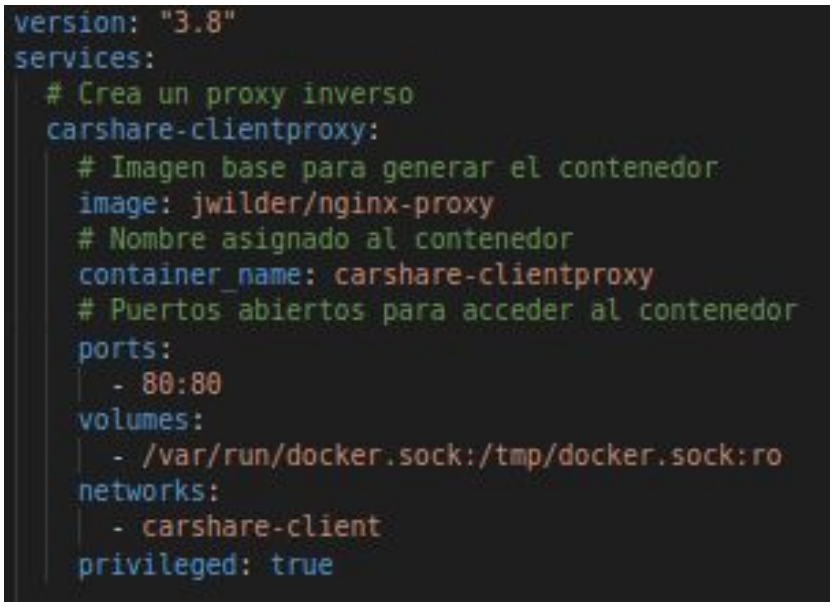
Para el despliegue de la parte de Frontend lanzaremos un script según el tipo de sistema operativo que tengamos, que se podrá encontrar en este mismo directorio.

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 cliente de la aplicación CarShare
version: "3.8"
services:
  # Crea un proxy inverso
  carshare-clientproxy:
    # Imagen base para generar el contenedor
    image: jwilder/nginx-proxy
    # Nombre asignado al contenedor
    container_name: carshare-clientproxy
    # Puertos abiertos para acceder al contenedor
    ports:
      - 80:80
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - carshare-client
    privileged: true

  # Crea un contenedor con la api de la aplicación
  carshare-client:
    # La imagen del contenedor se construye sobre una adecuando las necesidades
    build: .
    # Nombre asignado al contenedor
    container_name: carshare-client
    # Variable de entorno para el contendor
    environment:
      # Esta ruta debe estar incluida en el fichero hosts del anfitrión
      - VIRTUAL_HOST=carshare.client.local
      - VIRTUAL_PORT=4200
    # Para la versión de desarrollo se mapea la carpeta con una local
    volumes:
      - ./:/app/
    networks:
      - carshare-client
    depends_on:
      - carshare-clientproxy


networks:
    carshare-client:
        driver: bridge
```

3.1.2 Proxy Inverso



```yaml
version: "3.8"
services:
  # Crea un proxy inverso
  carshare-clientproxy:
    # Imagen base para generar el contenedor
    image: jwilder/nginx-proxy
    # Nombre asignado al contenedor
    container_name: carshare-clientproxy
    # Puertos abiertos para acceder al contenedor
    ports:
      - 80:80
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    networks:
      - carshare-client
    privileged: true
```

Este contenedor de nombre: **carshare-clientproxy**, 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.

- 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 Node

```
# Crea un contenedor con la api de la aplicación
carshare-client:
  # La imagen del contenedor se construye sobre una adecuando las necesidades
  build: .
  # Nombre asignado al contenedor
  container_name: carshare-client
  # Variable de entorno para el contendor
  environment:
    # Esta ruta debe estar incluida en el fichero hosts del anfitrión
    - VIRTUAL_HOST=carshare.client.local
    - VIRTUAL_PORT=4200
  # Para la versión de desarrollo se mapea la carpeta con una local
  volumes:
    - ./:/app/
  networks:
    - carshare-client
  depends_on:
    - carshare-clientproxy
```

Lo que hace este contenedor con nombre: **carshare-client** 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, en este caso es la misma que la red del Proxy inverso, ya que se tiene que comunicar entre ellos.

- Por último le decimos que depende del contenedor **carshare-clientproxy** (Proxy inverso), de manera que, hasta que el contenedor del Proxy Inverso no haya sido lanzado, este no se lanzará

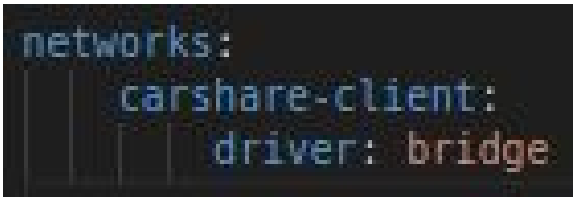
3.1.4 Dockerfile

```
#Descarga la imagen node para instalar angular
FROM node:alpine

WORKDIR /app

# Instala nodejs y npm
RUN apk add nodejs npm &&\
    npm install -g @angular/cli

CMD ["npm" ,"start"]

# Abre el puerto
EXPOSE 4200
```

En este Dockerfile lo que hacemos es partir de una imagen base de **Node** con Alpine (menor tamaño), le decimos cual va a ser su ruta de trabajo y posteriormente le aplicamos una serie de comandos que serán los encargados de instalar NodeJS y el gestor de paquetes NPM.

Para finalizar, lanzamos mediante el comando **CMD** que arranque el gestor NPM y por último, abrimos el puerto 4200.

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/ClienteRuedas

Gitkraken

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