Practica Docker

# Objectivo

El objetivo de esta práctica es montar en un entorno local de Docker compose y en un entorno de Kubernetes, 2 versiones de una aplicación que lista los coches de una lista en mongo db a través. La solución tiene 2 imágenes, una de base de datos (mongodb) y otra donde hay que compilar una solución web que consulta la base de datos y expone esa información en una api consultable desde un navegador. En la api se expone la versión de la solución, un valor del entorno de la aplicación web y el listado de coches de la base de datos.

## Overview Solucion

La solución se implementó siguiendo los siguientes pasos, que se detallarán a continuación:

- Ajuste de archivos Dockerfile

- Ajuste de archivos Controllers PracticaController.cs

- Creación de archivos de script inicialización db

- Creación de archivos docker-compose

- Creación de archivos . dockerignore

- Build y run con docker-compose

En general, la solución permite crear y ejecutar 2 servicios "web" y "base de datos mongo" dentro de docker, configurándolos adecuadamente para que puedan comunicarse entre sí y que se pueda conectar al servicio "web" desde localhost.

Docker

Database-mongo container

web container

mongodb

web app

Port: 27107

Port: 7007

Port: 7007



Archivos entregados

*practica\_docker*: Carpeta que contiene definicion del entorno de Docker y local

# Solucion

## Step 1: Ajuste file Dockerfile

El comando ARG usa valores disponibles en la fase de compilación y no después de que se crea la imagen, asigno a través de ARG la variable "Entorno" resaltada con mi nombre:

1. ARG Entorno="Giovanni Quattrocchi"

Utilizo el comando ENV para exponer la variable anterior configurándola como variable de entorno del contenedor de tal manera que esté disponible una vez creada la imagen:

1. ENV Entorno=${Entorno}

Asigno a la variable global MONGOHOST el nombre del servicio que aloja el contenedor mongodb, esto para permitir que el contenedor webapp pueda conectarse directamente al servicio mongo

1. ENV MONGOHOST=database-mongo

## Step 2: Ajuste file \Controllers\PracticaController.cs

Cambio la línea asignando el valor de la variable de entorno Version con la variable de clase “result” para que pueda devolver la versión configurada:

1. Version = Environment.GetEnvironmentVariable("Version") ?? "1"

## Step 3: Creación de archivos de script inicialización db

Creo la carpeta “script” con el archivo mongo-init.js dentro que contiene instrucciones para conectarse a la db " testdb" de mongo e insertar las líneas en la colección "cars"

Una posible alternativa de inicialización sería entrar dentro del contenedor que contiene mongo y ejecutar manualmente los comandos:

1. docker exec -it mongo-container /bin/bash
3. mongo testdb
4. db.cars.insert({name: "Audi", price: 52642})
5. db.cars.insert({name: "Mercedes", price: 57127})
6. db.cars.insert({name: "Skoda", price: 9000})
7. db.cars.insert({name: "Volvo", price: 29000})
8. db.cars.insert({name: "Bentley", price: 350000})
9. db.cars.insert({name: "Citroen", price: 21000})
10. db.cars.insert({name: "Hummer", price: 41400})
11. db.cars.insert({name: "Volkswagen", price: 21600})

## Step 4: Creación docker-compose

Creo el docker compose, que me permite configurar los servicios de las dos aplicaciones, de tal manera que con un solo comando creo y ejecuto todos los servicios con sus respectivas configuraciones

A través del archivo docker-compose se definirán dos servicios “web” y “database-mongo”.

### Servicio Mongo

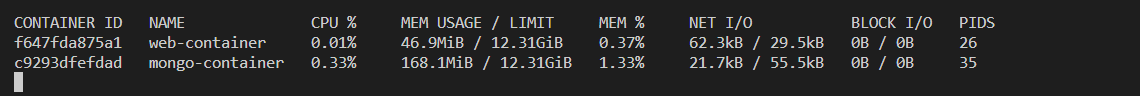
El servicio mongo utiliza una imagen pública desde el docker hub del repositorio oficial de mongo (descarga la última versión).

Monto la carpeta local script que contiene el script de inicialización de la base de datos en la carpeta docker-entrypoint-initdb. d, de esta manera en la fase de arranque del contenedor.

El código dentro de la carpeta docker-entrypoint-initdb. d solo se ejecuta si la base de datos nunca se ha inicializado antes.

Monto un volumen interno "volume-mongo" en la ruta del contenedor “data/db” (aquí se quedan los archivos de datos de mongo) para que la información sobre db sea persistente (en la primera invocación del comando docker-compose up se creará el volumen).

Al ejecutar el comando docker stats de antemano, analicé el consumo real de recursos de los dos contenedores en ejecuciones



Configuro una limitación de recursos basada en el análisis anterior para cada contenedor

En el detalle de la memoria, he configurado un hard limit desde el que no se puede ir más allá y un soft limit (reservation). De esta manera, docker intentará mantener la asignación de memoria por debajo del límite blando impuesto y, en caso de exceso, aumentará, pero solo hasta el límite duro impuesto

### Servicio Web

El servicio web utiliza una imagen que se construye desde el Dockerfile existente y que se ha modificado previamente en el directorio actual.

Connecta el contenedor y la máquina host al puerto expuesto 7007. El servicio web usa el puerto 7007 en su interior.

Creo una variable de entorno llamada “Version” con el valor 1 asignado, para permitir que el servicio la use dentro de ella.

Teniendo en cuenta que el servicio web necesita leer los datos de la base de datos, creé una dependencia con el servicio base de databes-mongo de tal manera que solo inicié el servicio web después de iniciar el servicio base de database-mongo.

El servicio web utiliza una imagen construida por Dockerfile esistente y anteriormente modificado in the current directory. Luego vincula el contenedor y la máquina host al puerto expuesto, 7007.

## Step 4: Creacion file .dockerignore

Creo que el archivo . dockerignore che serve per escludere file (docker-compose, Dockerfile) o directory (script) al momento della esecuzione della build del DockerFile.

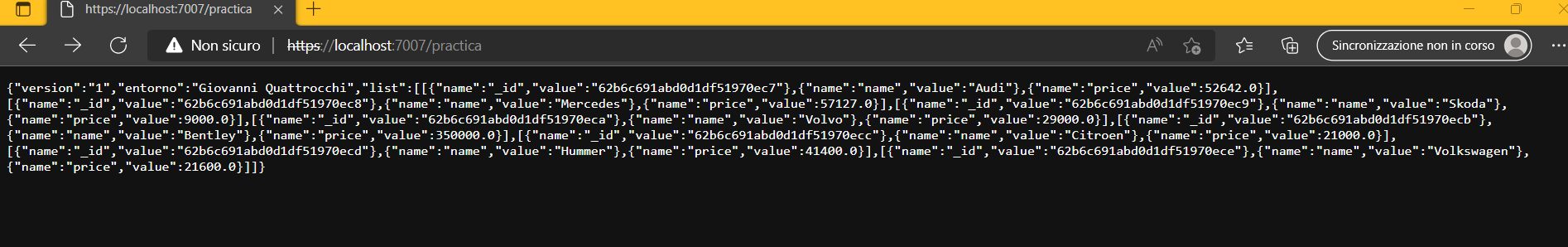
## Step 5: Build y run con docker-compose

Con el siguiente comando se ejecuta la extracción de mongodb y la compilación de la aplicación web (a través de DockerFile) e inicia los servicios definidos con las configuraciones adecuadas

1. docker compose up -d

Verificación del resultado

Accedo desde borwser por <https://localhost:7007/practica>



## Extra: export de imágenes y publicación

He exportado las imágenes con los siguientes comandos:

1. docker save webapp -o webapp.tar
2. docker save mongo -o mongo.tar

Asigno correctamente una etiqueta de publicación:

1. docker tag webapp giovanniq/webapp:v1
2. docker tag mongo giovanniq/mongo:v1

Realizo el pull para publicar la imagen:

1. docker push giovanniq/webapp:v1
2. docker push giovanniq/mongo:v1