TRABAJO FINAL CLOUD COMPUTING Y CONTENEDORES

# Parte 1: Entorno de desarrollo local utilizando docker

## Dockerfile

Primero se crea un Dockerfile de cada proyecto para generar las imágenes Docker de cada uno.

Se crea un dockerfile multietapa (2 Etapas). Donde la primera etapa se encarga de compilar y construir el programa y la segunda se encarga de ejecurlo y solo utiliza JRE que no incluye herramientas de compilado innecesarias, es más ligero y genera una imagen de menor tamaño.

|  |
| --- |
| # Etapa Construcción: Utilizar la imagen oficial de Maven para compilar el proyecto  FROM maven:3.9.9-eclipse-temurin-23 AS builder  # Establecer el directorio de trabajo dentro del contenedor  WORKDIR /app  # Copiar los archivos del proyecto al contenedor  COPY . .  # Compilar y construir el proyecto con Maven  RUN mvn clean package -DskipTests  # Etapa final: Crear la imagen final basada en OpenJDK  FROM eclipse-temurin:23-jre  # Establecer el directorio de trabajo dentro del contenedor  WORKDIR /app  # Copiar el JAR generado desde la fase de construcción  COPY --from=build /app/target/\*.jar app.jar  # Exponer el puerto definido en el application.properties  EXPOSE 9000  # Comando para ejecutar la aplicación  ENTRYPOINT ["java", "-jar", "app.jar"] |

*Se podría hacer en una sola etapa utilizando una sola imagen Maven+jdk para construir y ejecutar. Pero sería una imagen con un tamaño mayor (de 800MB a 490MB):*

|  |
| --- |
| FROM maven:3.9.9-eclipse-temurin-23  WORKDIR /app  COPY . .  RUN mvn clean package -DskipTests  EXPOSE 8080  CMD ["java", "-jar", "target/app.jar"] |

### Comandos

* **Construir Imagen**: docker build -t <nombre> <path>

Estando situado en la carpeta raiz de cada proyecto

docker build -t peliculas-frontend .

docker build -t peliculas-backend .

docker build -t usuarios-backend .

* **Crear contenedor**: docker run -p <puerto:puerto> <imagen> --name <nombre\_contenedor>

Con -d se ejecuta en segundo plano

docker run -d -p 9000:9000 --name peliculas-frontend peliculas-frontend

docker run -d -p 8001:8001 --name peliculas-backend peliculas-backend

docker run -d -p 8002:8002 --name usuarios-backend usuarios-backend

* **Parar un contenedor**: docker stop <nombre o id>
* **Eliminar un contendor**: docker rm <nombre o id>
* **Lanzar contenedor parado**: docker start <nombre o id>
* **Restart contenedor**: docker restart

SI SE HACE MODIFICACIÓN EN EL CÓDIGO

* Reconstruir la imagen
* Buscar y parar el contenedor
* Eliminar el contenedor
* Crear de nuevo el conenedor con la imagen actualizada

## Docker-compose

Todos los proyectos deben estar en la misma carpeta así como el propio docker-compose.yml

|  |
| --- |
|  |

### Configuración de nginx-proxy

Proxy inverso para que todos los servicios sean accesibles mediante un nombre de dominio (ej frontend.local, usuarios.local y peliculas.local).

* Configurar los hosts en el panel de navegación: http://localhost:81 (admin@example.com - changeme)
  + Domain Names: Nombre que se quiere dar al dominio: usuarios.local, peliculas.local, frontend.local
  + Forward Hostname / IP: Nombre del contenedor definido en docker-compose. Ej: peliculas\_backend, usuarios\_backend, frontend
  + Forward Port: Puerto utilizado por cada contendor.
  + Activar las opciones Cache Assets y Block Common Exploits.
* Configurar el archivos hosts para que los dominios creados apunten a la máquina local
  + /etc/hosts en Linux/Mac
  + C:\Windows\System32\drivers\etc\hosts en Windows
  + Se añaden los dominios:

127.0.0.1 frontend.local

127.0.0.1 usuarios.local

127.0.0.1 peliculas.local

PROBLEMA SURGIDO

En este punto había un problema: Ocurría el siguiente error solo al hacer llamadas entre el microservicio de frontend y los demas. Si se accedían a los dominios de backend desde el navegador sí funcionaban:

|  |
| --- |
| org.springframework.web.client.ResourceAccessException  Caused by: java.net.ConnectException: Connection refused |

Esto ocurría porque el contenedor del frontend no podía resolver o conectarse al dominio usuarios.local o peliculas.local porque los dominios configurados en el archivo hosts solo son accesibles desde la máquina anfitriona, no desde dentro de los contenedores de Docker. Esto ocurre porque los contenedores Docker no leen ni respetan el archivo hosts de tu sistema anfitrión.

Cuando se accede a http://usuarios.local desde el navegador, el archivo hosts redirige correctamente la solicitud al servicio backend (usuarios\_backend) a través de NGINX Proxy Manager.

Sin embargo, dentro del contenedor del frontend, el dominio usuarios.local no está configurado ni es resoluble porque los contenedores necesitan referencias válidas dentro de su red de Docker.

SOLUCIÓN

Se añaden "aliases" al servicio nginx-proxy en el fichero docker-compose.yml

|  |
| --- |
| nginx-proxy:  networks:  trabajo\_contenedores:  aliases:  - usuarios.local  - peliculas.local  - frontend.local |

Cuando defines un alias en la configuración de red de un contenedor, Docker registra ese alias como un nombre adicional para el contenedor dentro de la red. Es decir, los demás contenedores en la misma red pueden usar cualquiera de los alias configurados para comunicarse con el contenedor.

Esto significa que dentro de la red "trabajo\_contenedores", cualquier contenedor podrá acceder al contenedor nginx-proxy utilizando los nombres:

- usuarios.local

- peliculas.local

- frontend.local

Docker tiene un servicio interno de DNS para las redes. Cuando un contenedor intenta acceder a usuarios.local, Docker verifica si ese nombre (alias) está asignado a algún contenedor en la misma red. Si existe, lo redirige automáticamente al contenedor correspondiente (en este caso, nginx-proxy).

En este caso, el contenedor nginx-proxy actúa como un proxy inverso, redirigiendo el tráfico recibido en los alias (usuarios.local, peliculas.local, etc.) a los servicios backend correspondientes según las configuraciones que hayas creado en su panel de administración.

RESUMEN: Al utilizar cualquiera de esos nombres dentro de cualquier docker de la network "trabajo\_contenedores", se va a redirigir a nginx y esté lo va a redirigir al servicio backend que tenga configurado con ese nombre.

### Configuración Portainer

<http://localhost:9001> (admin -> portainerpass)

### Comandos

* **Construir las imágenes** de docker: docker-compose build
* **Construir solo un servicio**: docker-compose build <nombre\_del\_servicio>

Ej: docker-compose build peliculas\_backend

* **Iniciar contenedores**: docker-compose up (No reconstruye la imagen)
  + docker-compose up –build

La opción --build asegura que cualquier imagen actualizada se reconstruya automáticamente antes de levantar los contenedores. Utilizar si se ha cambiado el codigo fuente o el Dockerfile

* + docker-compose up --build --force-recreate

Reconstruye todas las imágenes. Crea nuevos contenedores desde cero.

* **Iniciar un servicio específico**: docker-compose up --build <nombre\_del\_servicio>
* **Consultar logs en tiempo real**: docker-compose logs -f
* **Consultar logs de un servicio**: docker-compose logs -f <nombre\_del\_servicio>
* **Parar los contenedores**: docker-compose down
* **Reiniciar un servicio**: docker-compose restart <nombre\_del\_servicio>

## Modificaciones en el código de los proyectos

- Como las bases de datos MySql están montadas en local, se actualiza la configuración de conexión de los servicios para que utilices estas bases de datos.

Como estas conexiones se han definido en variables de entorno (del contenedor docker) de cada servicio en el fichero docker-compose.yml, en los application.properties solo hay que utilizar estas variables:

|  |
| --- |
| spring.datasource.url=${SPRING\_DATASOURCE\_URL}  spring.datasource.username=${SPRING\_DATASOURCE\_USERNAME}  spring.datasource.password=${SPRING\_DATASOURCE\_PASSWORD} |

- Eliminar la dependencia con netflix-eureka-client del pom.xml y su definición en application.properties (eureka.client.service-url.defaultZone) de todos los proyectos

- En el proyecto de frontend:

Cambiar las url que utilizan los diferentes RestTemplates en los ServiceImpl (antes localhost:puerto) por los nuevos dominios creados con nginx. Para ello se añaden al application.properties:

|  |
| --- |
| api.peliculas.peliculas.url=http://peliculas.local/peliculas  api.peliculas.actores.url=http://peliculas.local/actores  api.usuarios.criticas.url=http://usuarios.local/criticas  api.usuarios.roles.url=http://usuarios.local/roles  api.usuarios.usuarios.url=http://usuarios.local/usuarios |

|  |
| --- |
| **Antes:**  String url = "http://localhost:8090/api/peliculas/actores";  **Ahora:**  @Value("${api.peliculas.actores.url}")  private String url;  **Antes:**  String url = "http://localhost:8090/api/usuarios/criticas";  **Ahora:**  @Value("${api.usuarios.criticas.url}")  private String url;  **Antes:**  String url = "http://localhost:8090/api/peliculas/peliculas";  **Ahora:**  @Value("${api.peliculas.peliculas.url}")  private String url;  **Antes:**  String url = "http://localhost:8090/api/usuarios/roles";  **Ahora:**  @Value("${api.usuarios.roles.url}")  private String url;  **Antes:**  String url = "http://localhost:8090/api/usuarios/usuarios";  **Ahora:**  @Value("${api.usuarios.usuarios.url}")  private String url; |

# PARTE 2: Despliegue de la aplicación