**Introducción**

Este proyecto implementa una arquitectura de microservicios utilizando Spring Boot 3.3.10, Eureka Server para descubrimiento de servicios, Spring Cloud Gateway como puerta de enlace, y autenticación JWT en el usuarios-servicio. La comunicación entre servicios se realiza con Feign Client, y se simula una matrícula entre estudiantes y asignaturas.

**Tecnologías Utilizadas**

* Java 17
* Spring Boot 3.3.10
* Spring Cloud (2023.0.5)
* Spring Security
* JWT (JJWT 0.11.5)
* Spring Cloud Gateway
* Eureka Discovery Server
* Feign Client
* Maven

**El proyecto está dividido en los siguientes módulos/microservicios:**

1. **eureka-server**: Servicio de descubrimiento
2. **api-gateway**: Gateway que enruta el tráfico a los microservicios registrados
3. **usuarios-servicio**: Servicio de autenticación y gestión de usuarios
4. **asignaturas-servicio**: Servicio CRUD para asignaturas
5. **matriculas-servicio**: Servicio que consume los otros microservicios y simula una matrícula completa

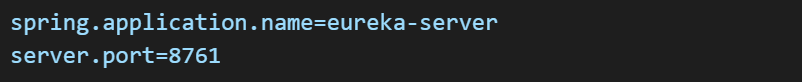
Cada microservicio cuenta con su propio **application.properties**, archivo **pom.xml**, estructura de paquetes y configuraciones independientes.

**Detalles por Microservicio**

**1. eureka-server**

* Puerto: 8761
* Rol: Registro centralizado de microservicios

**Main Config:**



**Dependencias Maven:**

<dependency>

<groupId>org.springframework.cloud</groupId>

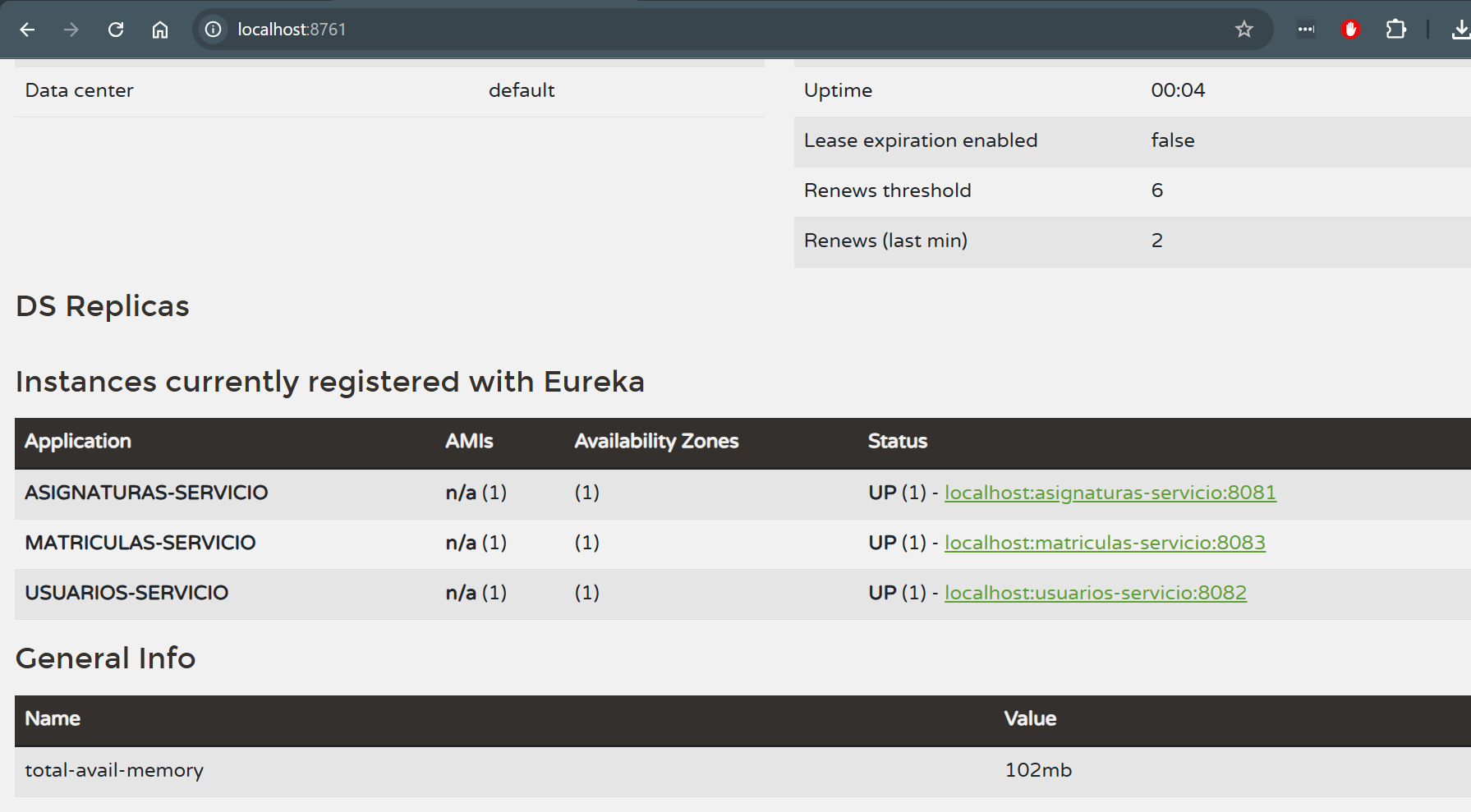
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

Funcionamiento:   
  
**A screenshot of a computer

AI-generated content may be incorrect.**

**Comunicación de los microservicion con el server**

****

**2. api-gateway**

* Puerto: 8080
* Rol: Enrutador principal de peticiones HTTP
* Rutas definidas por servicio (/api/usuarios/\*\*, /api/asignaturas/\*\*, /api/matriculas/\*\*)
* **Importante:** Usa spring-cloud-starter-gateway-mvc

**Configuracion del** **Aplication.properties:  
  
  
A screenshot of a computer program

AI-generated content may be incorrect.**

**Pruebas de funcionamiento:  
  
A screenshot of a computer program

AI-generated content may be incorrect.**

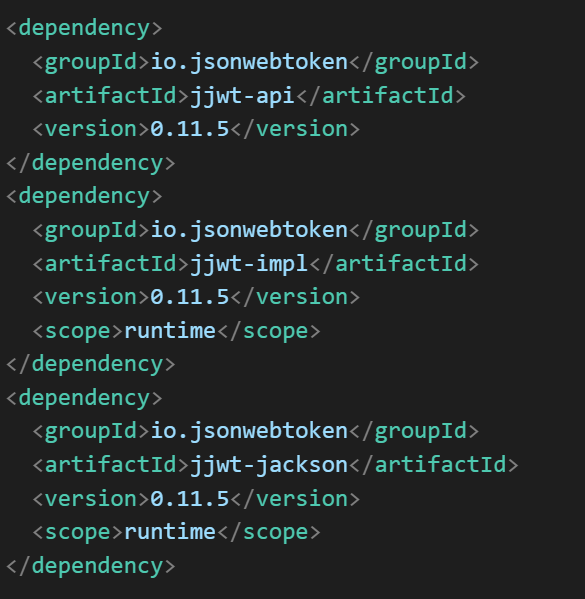
**3. usuarios-servicio**

* Puerto: 8082
* Rol:
  + Autenticación de usuarios (JWT)
  + Expone /auth/login para generar token JWT
  + Expuesto en Eureka y enrutado desde Gateway

**Estructura de Paquetes:**

* controller ➝ AuthController, UsuarioController
* model ➝ Usuario, AuthRequest, AuthResponse
* service ➝ JwtUtil, CustomUserDetailsService
* config ➝ SecurityConfig
* filter ➝ JwtRequestFilter

**Dependencias JWT:**



Pruebas en postman del microservicio   
  
**A black screen with white lines

AI-generated content may be incorrect.**

Ejecución del servicio  
  
**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**4. asignaturas-servicio**

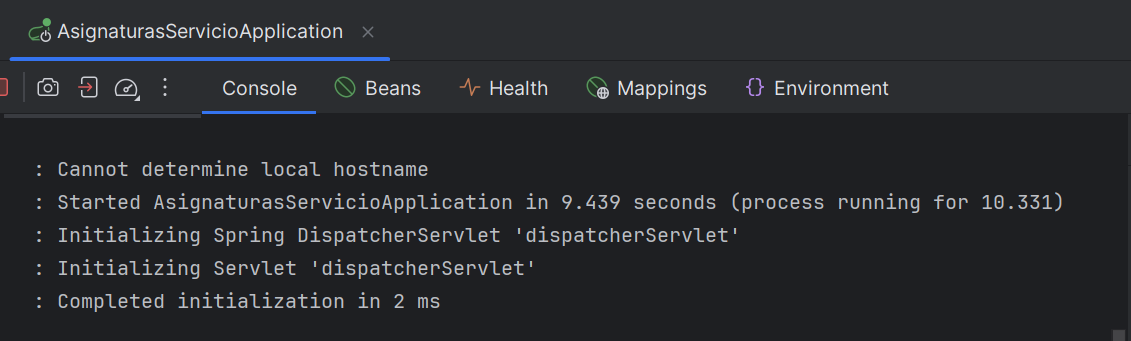
* **Puerto:** 8081
* **Rol:** CRUD para asignaturas
* **Ejemplo:**

@GetMapping("/{id}")

public Asignatura obtenerAsignatura(@PathVariable Long id) {

return new Asignatura(id, "Matemáticas", "Asignatura de cálculo");

}

**Ejecucion del servicio**  
  
****

**A screenshot of a computer

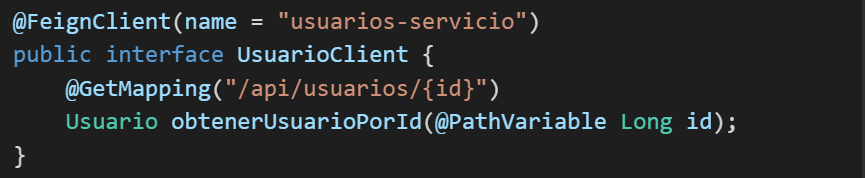
AI-generated content may be incorrect.**

**5.matriculas-servicio**

* Puerto: 8083
* Rol: Simula matrículas llamando a usuarios-servicio y asignaturas-servicio mediante Feign Client

**Paquetes:**

* controller ➝ MatriculaController
* dto ➝ MatriculaResponse, MatriculaCompleta
* feign ➝ UsuarioClient, AsignaturaClient
* service ➝ MatriculaService, FeignClientInterceptor
* model ➝ Usuario, Asignatura, Matricula

**Feign Clients:  
  
**

**Interceptor para JWT:**

@Component

public class FeignClientInterceptor implements RequestInterceptor {

private final TokenProvider tokenProvider;

public FeignClientInterceptor(TokenProvider tokenProvider) {

this.tokenProvider = tokenProvider;

}

@Override

public void apply(RequestTemplate template) {

String token = tokenProvider.getToken();

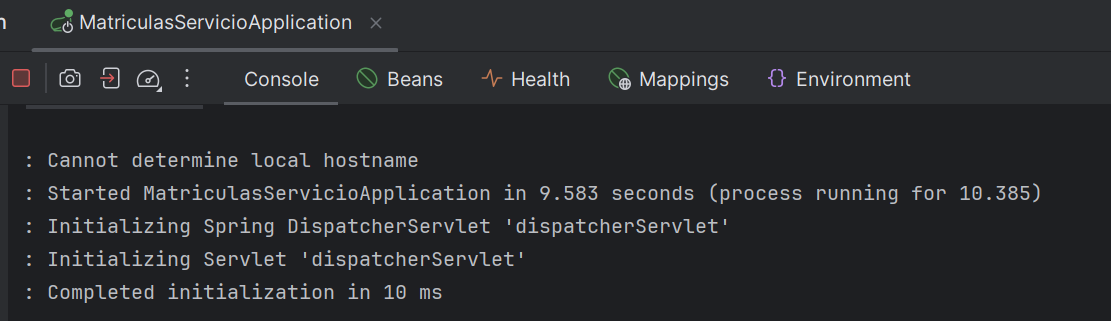
if (token != null) {

template.header("Authorization", "Bearer " + token);

}

}

}

**Ejecution del servicio   
  
**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Ejemplo de Flujo**

1. Se hace POST a /auth/login con admin/admin y se obtiene un JWT.
2. Se puede consultar /api/matriculas/completas pasando el JWT como Bearer en el header.
3. Este microservicio llama con Feign a los otros servicios inyectando el mismo token.

Ejecución de una matrícula complete (se enviaron datos estáticos)  
  
A screenshot of a computer

AI-generated content may be incorrect.

# **Pruebas Unitarias e Integración**

# **Microservicio: Usuarios-servicios**

Prueba unitaria para JwtUtil  
Prueba la generación y validación de tokens JWT.  
  
Texto

El contenido generado por IA puede ser incorrecto.

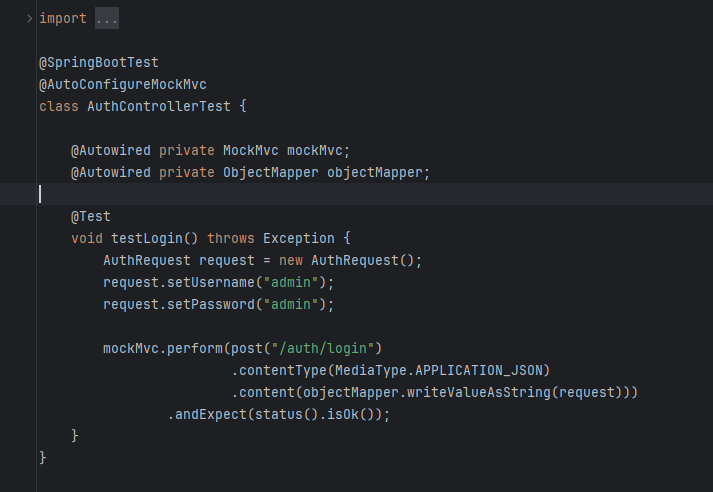
Prueba unitaria para UsuarioController

Prueba la consulta de usuario

Texto

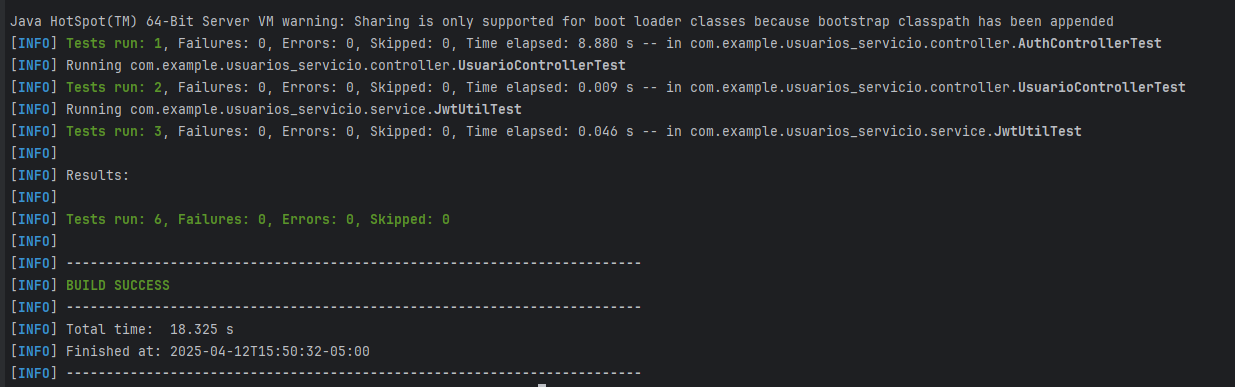
El contenido generado por IA puede ser incorrecto.

Pruebas de Integración con Spring Bot Test  
  
Estas pruebas validan el comportamiento de los endpoints REST usando MockMvc.



**Resultado:**

Se realizaron seis pruebas automatizadas con el microservicio usuarios-servicio. Todas las pruebas pasaron exitosamente sin errores ni fallos. Esto valida que los componentes clave del microservicio están funcionando correctamente de forma aislada y conjunta.



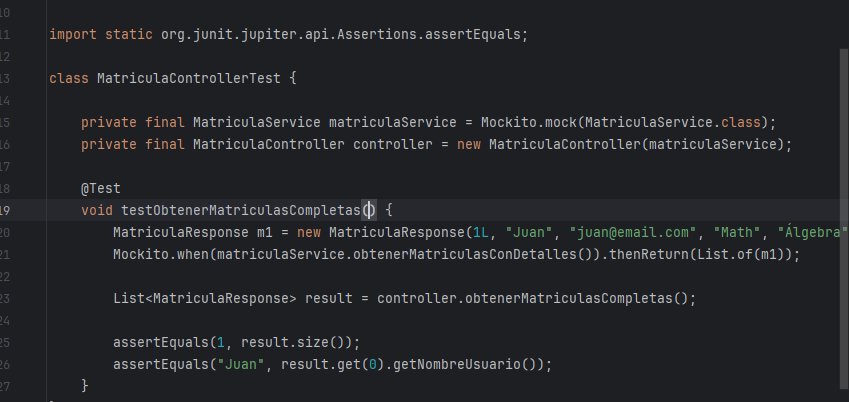
# Pruebas Unitarias e Integración

Microservicio: Matriculas-servicio

**Prueba unitaria:**

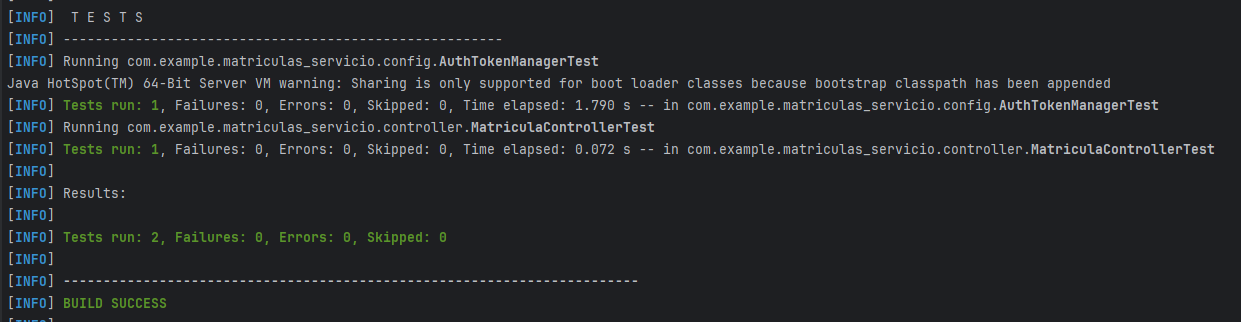


**Prueba unitaria:**



**Resultado:**

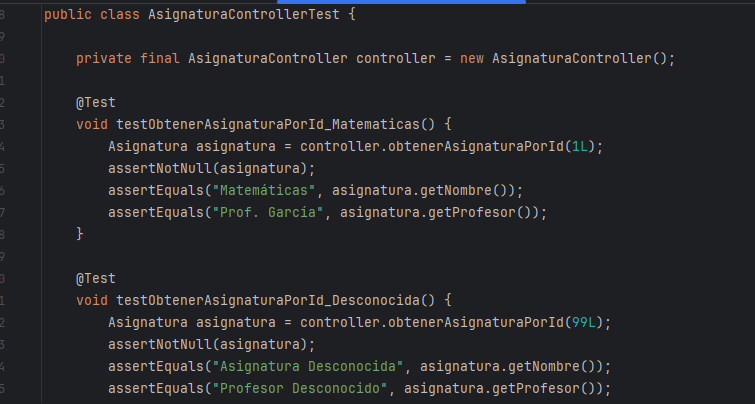
Se ejecuto dos pruebas, para authTokenManagertest y para MatriculaControllerTest,, con resultados exitosos.



# Pruebas Unitarias e Integración

Microservicio: Asignatura-servicio

**Pruebas unitarias**



**Prueba de integración:**

Texto

El contenido generado por IA puede ser incorrecto.

**Resultado:** se ejecuto una prueba unitara y una de integración con resultados exitosos

Texto

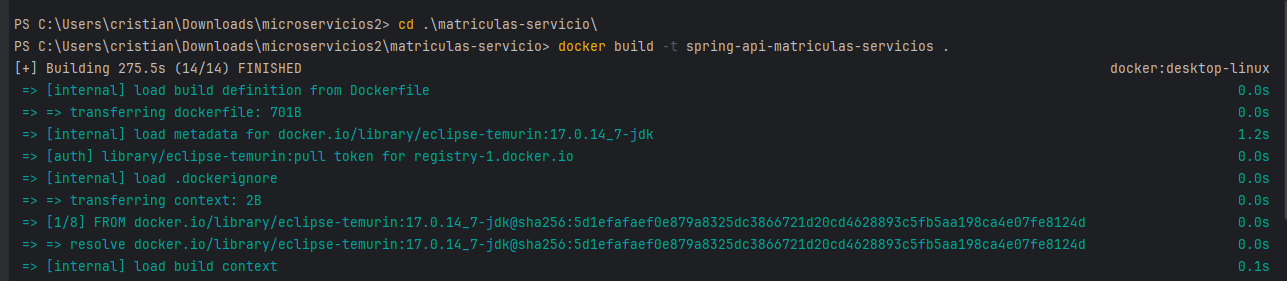
El contenido generado por IA puede ser incorrecto.

# Construir Docker.

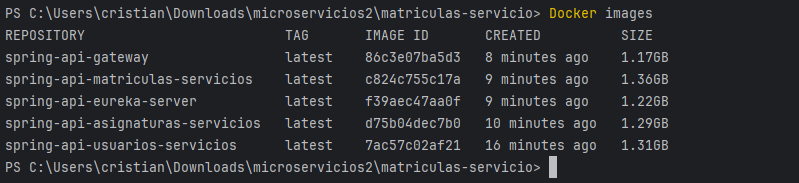
Construir Imagen:

Sobre cada microservicio se ejecuta el siguiente mensaje para crear la imagen del microservicio

docker build -t spring-api-matriculas-servicios .



Se realiza lo mismo para los demás Microservicios y se confirma las imágenes por consola o con el Docker desktop



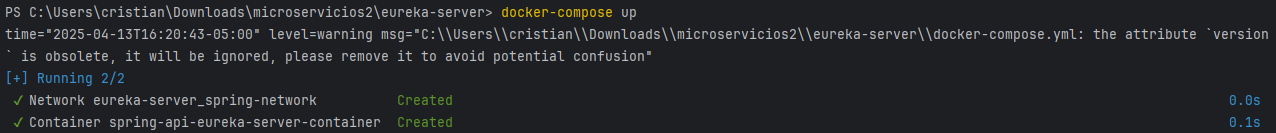
Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

**#crear contenedor:**

Sobre cada microservicio ejecutar el siguiente comando para crear el contenedor

docker-compose up



Se confirma la creación de las imágenes y contenedores con el siguiente comando:  
  
Docker ps  
  
Texto

El contenido generado por IA puede ser incorrecto.

Para que los Microservicios puedan comunicarse con los demás contenedores/microservicios es necesario crear una red de Docker que tengan en común

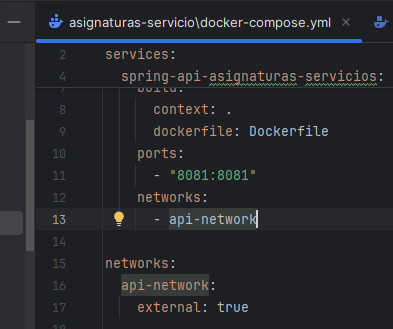
Se crea la red

docker network create --attachable api-network # se crea la red compartida

Se asigna la red

docker network connect api-network "image"

se asigna la red en la configuración del docker-compose, por cada microservicio



Si se realizo cambios en el código ejecutar los siguientes comandos para actualizar el contenedor y la imagen de Docker

Reconstruye la imagen

docker-compose build

docker-compose up

En application.properties cambiar la URL por la imagen creada en el microservicio

Texto

El contenido generado por IA puede ser incorrecto.



Se confirma el funcionamiento y comunicación de los microservicios con Docker.

Tabla

El contenido generado por IA puede ser incorrecto.

Se valida la comunicación con Postman:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Enlace Github:

Autores:

Cristian David Otalvaro Almanza

Roman Andres Urrego Peña