DIGITALHOUSE

Comunicación segura entre microservicios

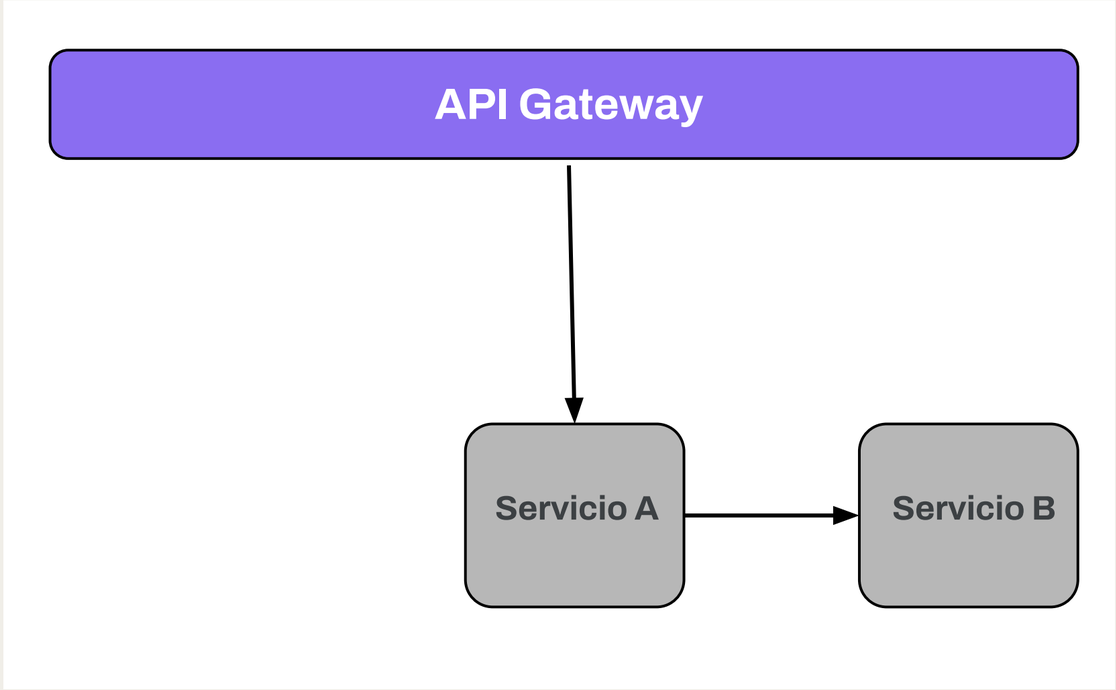
Certified Tech Developer
The Ultimate Degree

Índice

Introducción

Supongamos que tenemos la siguiente arquitectura de microservicios, en donde todos los servicios requieren un JWT para permitir el consumo de sus API.

Si necesitamos consumir internamente un servicio que está disponible solo para usuarios autenticados, tenemos que encontrar la forma de generar un token en el microservicio que origina la petición o reenviar el token que llega del gateway para poder obtener una respuesta.



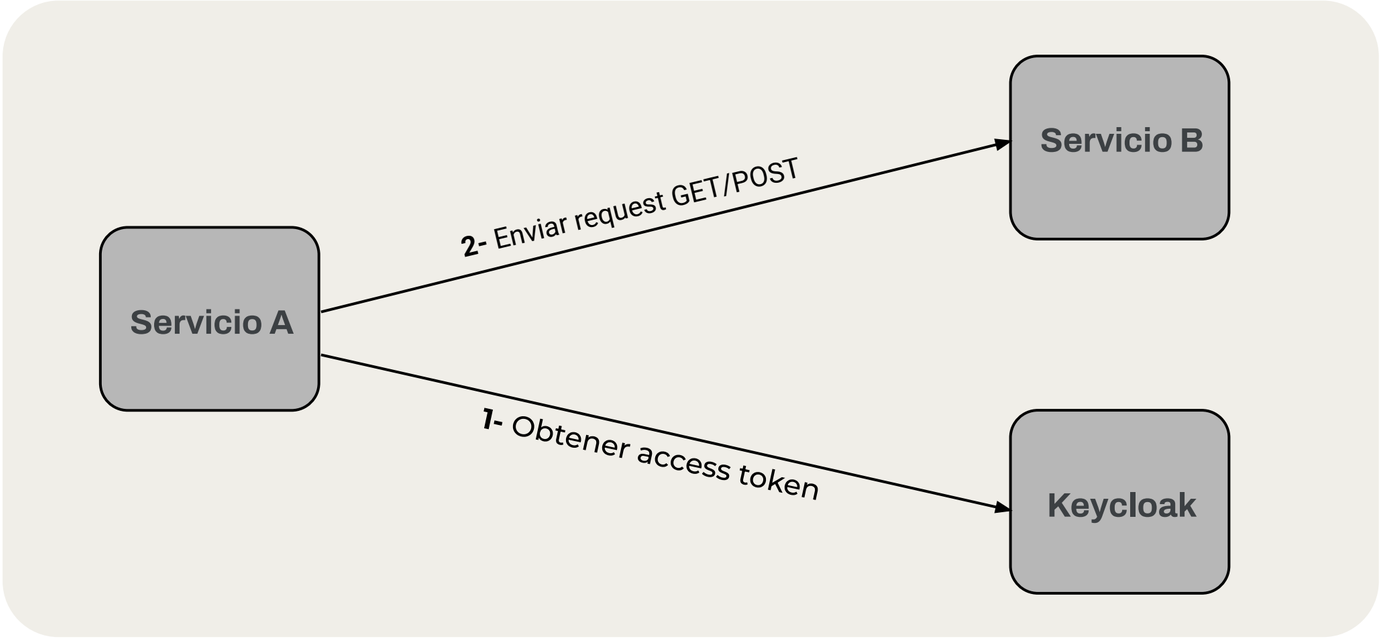
En el caso del ejemplo, necesitamos que el Servicio A pueda **obtener un token** para luego consumir los endpoints del Servicio B, o **reenviar el token que llega a través del gateway**.

01

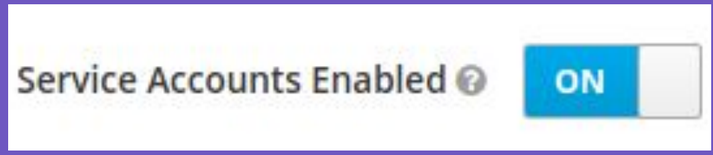
Obtener un token nuevo

Para obtener un token nuevo, necesitamos autenticarnos con Keycloak antes de realizar la llamada y tomar el token.

Utilizando las credenciales del cliente (**client credentials**) obtenemos el access token y luego enviamos el token en el request para poder obtener los recursos solicitados. El Servicio B valida el token y, si es válido, atiende nuestra solicitud.



Para obtener un token a partir de las credenciales del cliente tenemos que habilitar la opción “Service Accounts Enabled” en el cliente que vayamos a utilizar para el microservicio.



02

Configuración en el servicio que origina la solicitud

En Servicio A, vamos a utilizar Feign para enviar las solicitudes al Servicio B.

Aquí podemos ver las **dependencias necesarias**:

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency

<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-oauth2-client</artifactId>
</dependency>


<dependencyManagement>
   <dependencies>
       <dependency>
           <groupId>org.springframework.cloud</groupId>
           <artifactId>spring-cloud-dependencies</artifactId>
           <version>${spring-cloud.version}</version>
           <type>pom</type>
           <scope>import</scope>
       </dependency>
   </dependencies>
</dependencyManagement>
```

Agregamos en la clase principal:

```
@EnableFeignClients

@SpringBootApplication
@EnableFeignClients
public class ServiceA {
    public static void main(String[] args) {
        SpringApplication.run(ExampleApplication.class, args);
    }
}
```

03 Configuración para obtener el token

El objetivo es agregar el access token automáticamente a todas los requests que salgan del Servicio A mediante Feign. Para realizar esto vamos a usar interceptores, específicamente **RequestInterceptor**, que inyecta el access token en todas los requests de Feign, agregándolo en el header.

```
@Configuration
public class OAuthFeignConfig {

 public static final String CLIENT_REGISTRATION_ID = "keycloak";

 private final OAuth2AuthorizedClientService oAuth2AuthorizedClientService;
 private final ClientRegistrationRepository clientRegistrationRepository;

publicOAuthFeignConfig(OAuth2AuthorizedClientService oAuth2AuthorizedClientService,
                       ClientRegistrationRepository clientRegistrationRepository) {
   this.oAuth2AuthorizedClientService = oAuth2AuthorizedClientService;
   this.clientRegistrationRepository = clientRegistrationRepository;
 }

@Bean
public RequestInterceptor requestInterceptor() {
 ClientRegistration clientRegistration =
clientRegistrationRepository.findByRegistrationId(CLIENT_REGISTRATION_ID);
 OAuthClientCredentialsFeignManager clientCredentialsFeignManager =
     new OAuthClientCredentialsFeignManager(authorizedClientManager(),
clientRegistration);
 return requestTemplate -> {
   requestTemplate.header("Authorization", "Bearer " +
clientCredentialsFeignManager.getAccessToken());
 };
}
}
```

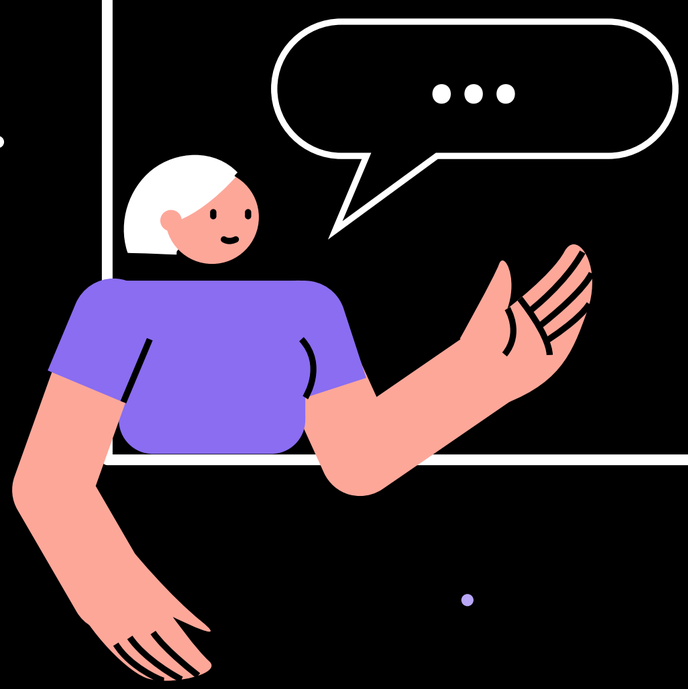
En el bean **requestInterceptor()**, usamos las clases **ClientRegistration** y **OAuthClientCredentialsFeignManager** para registrar el cliente oauth2 y obtener un token de acceso del servidor de autorización (Keycloak). Para hacer esto, necesitamos definir las propiedades del cliente oauth2 en nuestro archivo **application.properties**.

```
spring.security.oauth2.client.registration.keycloak.authorization-grant-type=client_credentials
spring.security.oauth2.client.registration.keycloak.client-id=backend
spring.security.oauth2.client.registration.keycloak.client-secret=TLP9Bau4NLouo8Cb0I8yzydfxq35AIHC
spring.security.oauth2.client.provider.keycloak.token-uri=http://localhost:8082/realms/digital-house/protocol/openid-connect/token
```

La clase **OAuthClientCredentialsFeignManager** es una clase que tenemos que crear nosotros para obtener el token.

```
public String getAccessToken() {
 try {
   OAuth2AuthorizeRequest oAuth2AuthorizeRequest = OAuth2AuthorizeRequest
       .withClientRegistrationId(clientRegistration.getRegistrationId())
       .principal(principal)
       .build();
                          OAuth2AuthorizedClient         client         =
manager.authorize(oAuth2AuthorizeRequest);
   if (isNull(client)) {
        throw new IllegalStateException("client credentials flow on " +
clientRegistration.getRegistrationId() + " failed, client is null");
   }
   return client.getAccessToken().getTokenValue();
 } catch (Exception exp) {
   System.out.println("client credentials error " + exp.getMessage());
 }
 return null;
}
```

Este método se encarga de enviar la solicitud a Keycloak para autenticarnos y obtener el token. De esta manera, cada vez que enviemos una solicitud utilizando Feign, el interceptor ejecutará este método y agregará el token al header del request.



04 Implementación de Feign

Ahora solo nos queda crear la interfaz para utilizar Feign.

Aquí agregamos la clase que creamos anteriormente, **OAuthFeignConfig**, como una clase de configuración. De esta manera, se ejecutarán sus métodos.

```
@FeignClient(name = "service-b",url = "http://localhost:8083/",configuration = OAuthFeignConfig.class)
public interface IFeignServiceBRepository {

 @GetMapping("/product/all")
 List<Map<String,String>> findAll();
}
```

05

Reenviar el token que llega desde el gateway

Para reenviar el token también vamos a utilizar **RequestInterceptor**, pero esta vez vamos a obtener el token del usuario autenticado en lugar de autenticarnos con Keycloak.

```java
@Component
public class FeingInterceptor implements RequestInterceptor {
 private final String AUTHORIZATION_HEADER = "Authorization";
 private final String TOKEN_TYPE = "Bearer";

 @Override
 public void apply(RequestTemplate requestTemplate) {
   String token = getAccessToken();
   if (token != null) {
     requestTemplate.header(AUTHORIZATION_HEADER, String.format("%s %s", TOKEN_TYPE, token));
   }
 }

 private String getAccessToken() {
   Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
   String token = null;
   if (authentication != null) {
     try {
       token = ((JwtAuthenticationToken) authentication).getToken().getTokenValue();
     } catch (Exception ignored) {

     }

   }
   return token;
 }
}
```

Como podemos ver en el método **Apply**, agregamos el token que nos devuelve el método **getAccessToken()** en el header. De esta manera, todas las solicitudes enviadas con Feign ejecutarán este método.

Para cambiar de un interceptor a otro, tenemos que modificar la clase de configuración en la interfaz de Feign.

De esta manera, cambiamos **OAuthFeignConfig** por **FeingInterceptor**.

```java
@FeignClient(name = "product-service",url =
"http://localhost:8083/",configuration = FeingInterceptor.class)
public interface IFeignProductRepository {

 @GetMapping("/product/all")
 List<Map<String,String>> findAll();
}
```

DIGITALHOUSE

¡Muchas gracias!