

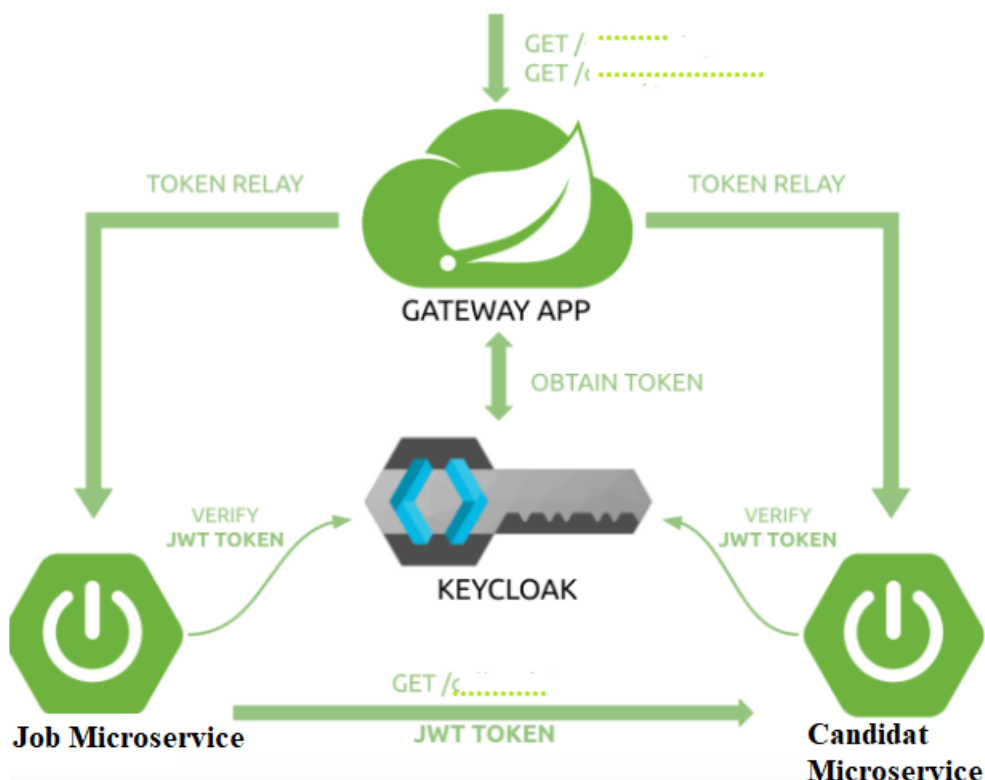
Sécurisation d'une API Gateway avec Keycloak

Objectif

- Assurer la sécurité d'une API Gateway en utilisant Auth2.0.
- Assurer la sécurité de l'accès vers l'ensemble des microservices.

Principe de fonctionnement

Lors de l'accès aux Microservices à travers le Gateway, celle-ci va renvoyer automatiquement l'utilisateur vers Keycloak pour récupérer un Token. Keycloak authentifiera l'utilisateur si besoin, puis renverra des informations sur l'utilisateur et le fameux Token à notre Viewer. Ce Token sera ensuite utilisé pour l'accès vers le microservice en question.



Les étapes à suivre

1. Créer un realm « **JobBoardKeycloak** »
2. Créer un client **gateway**
 - o clientID : gateway (nom de l'application Microservice Gateway)
 - o RootURL, HomeURL et AdminURL : <http://localhost:8093> (Port de MS Gateway)

- Vérifier la configuration ajoutée selon les figures ci-dessous :

General settings

Client ID * ⓘ gateway

Name ⓘ gateway

Description ⓘ

Always display in UI ⓘ ☐ Off

Access settings

Root URL ⓘ http://localhost:8093

Home URL ⓘ http://localhost:8093

Admin URL ⓘ http://localhost:8093

Capability config

Client authentication ⓘ ☒ On

Authorization ⓘ ☐ Off

Authentication flow

- ☐ Standard flow ⓘ
- ☐ Implicit flow ⓘ
- ☒ Service accounts roles ⓘ
- ☐ OAuth 2.0 Device Authorization Grant ⓘ
- ☐ OIDC CIBA Grant ⓘ

Logout settings

Front channel logout ⓘ ☒ On

Front-channel logout URL ⓘ

Backchannel logout URL ⓘ

Backchannel logout session required ⓘ ☒ On

Backchannel logout revoke offline sessions ⓘ ☐ Off

Jump to section

General settings

Access settings

Capability config

Login settings

Logout settings

3. Dans le fichier pom.xml de votre projet Gateway, ajouter le code suivant :

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>
```

```

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-oauth2-resource-
server</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>
  <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>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <repositories>

```

```

<repository>
  <id>netflix-candidates</id>
  <name>Netflix Candidates</name>
  <url>https://artifactory-
oss.prod.netflix.net/artifactory/maven-oss-candidates</url>
  <snapshots>
    <enabled>>false</enabled>
  </snapshots>
</repository>
</repositories>

```

4. Sous le projet Gateway, ajouter la classe **SecurityConfig**

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecu
rity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.server.SecurityWebFilterChain;

@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity
serverHttpSecurity) {
        return serverHttpSecurity.csrf(ServerHttpSecurity.CsrfSpec::disable)
            .authorizeExchange(exchange -> exchange.pathMatchers("/eureka/**"))
            .permitAll()
            .anyExchange().authenticated()
            .oauth2ResourceServer((oauth) -> oauth
                .jwt(Customizer.withDefaults()))
            .build();
    }

}

```

5. Dans le fichier **application.properties**, ajouter la propriété suivante :

```

spring.security.oauth2.resourceserver.jwt.issuer-
uri=http://localhost:8080/realms/JobBoardKeycloak

```

```

spring.application.name=gateway
server.port=8093

# eureka registration
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
eureka.client.register-with-eureka=true

spring.security.oauth2.resourceserver.jwt.issuer-uri=http://localhost:8080/realms/JobBoardKeycloak

```

6. Maintenant, nous allons tester l'accès sécurisé vers le MS de G. Candidats.

Exécuter les projets :

- Gateway
- Candidat
- Eureka

7. Accéder au serveur eureka

The screenshot shows the Spring Eureka web interface. At the top, there's a header with the 'spring Eureka' logo. Below it, the 'System Status' section displays environment details like 'test' and 'default'. To the right, there are status indicators for 'Current time', 'Uptime', 'Lease expiration enabled', 'Renews threshold', and 'Renews (last min)'. The 'DS Replicas' section shows 'localhost'. The main part of the interface is 'Instances currently registered with Eureka', which contains a table with the following data:

Application	AMIs	Availability Zones	Status
CANDIDAT-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:candidat-service:8088
GATEWAY	n/a (1)	(1)	UP (1) - host.docker.internal:Gateway:8093

NB :

- 8093 : est le port de l'API Gateway
- Le path d'accès au MS de Candidat est candidats
`r->r.path("/candidats/**")`

8. Sur le postman, nous allons générer un token à travers le Keycloak

- Veuillez choisir Authorization-> OAuth 2.0
- Veuillez configurer un nouveau token

GET localhost:8093/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

The authorization data will be automatically generated when you send the request.
[Learn more about authorization](#)

Add authorization data to Request Hea... v

Share token
Sharing this token will allow anyone with access to this request to view and use it.

Configure New Token
Configuration Options Advanced Options

Token Name keycloakToken

Grant Type Client Credentials

Access Token URL http://localhost:8080/realms/JobBoardKeyc...

Client ID gateway

Client Secret fRnY5s71GBqMtCkBwNVSmBUImPxHKyFw

Scope openid offline_access

Client Authentication Send as Basic Auth header

Clear cookies

Get New Access Token

```
{
  "issuer": "http://localhost:8080/realms/JobBoardKeycloak",
  "authorization_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/auth",
  "token_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/token",
  "introspection_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/token"
}
```

Comme le montre la figure ci-dessus :

- Token Name : veuillez choisir KeycloakToken
- Grant type : Client credentials
- Access Token URL : Pour récupérer cette valeur, au niveau du Realm Settings du keycloak, veuillez cliquer sur le lien OpenID EndPoint Configuration. Puis, récupérer la valeur de token-endpoint

```
{
  "issuer": "http://localhost:8080/realms/JobBoardKeycloak",
  "authorization_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/auth",
  "token_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/token",
  "introspection_endpoint": "http://localhost:8080/realms/JobBoardKeycloak/protocol/openid-connect/token"
}
```

- Client ID : la valeur Client ID spécifié dans Keycloak

KEYCLOAK

JobBoardKeycloak

Manage Clients Client scopes Realm roles Users Groups

Clients > Client details

gateway OpenID Connect Enabled

Clients are applications and services that can request authentication of a user.

Settings Keys Credentials Roles Client scopes Service accounts roles Sessions Advanced

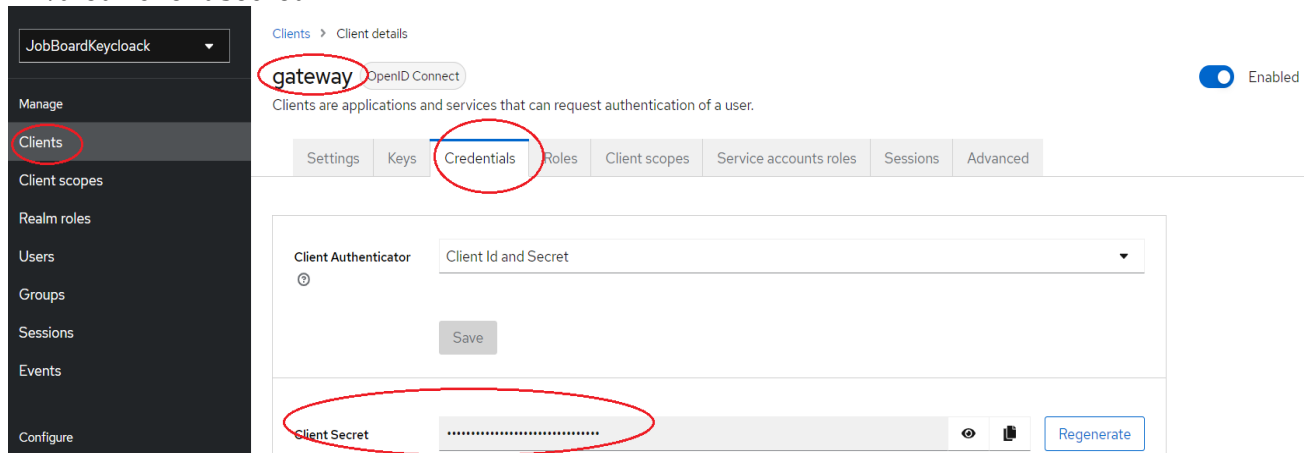
General settings

Client ID * @ gateway

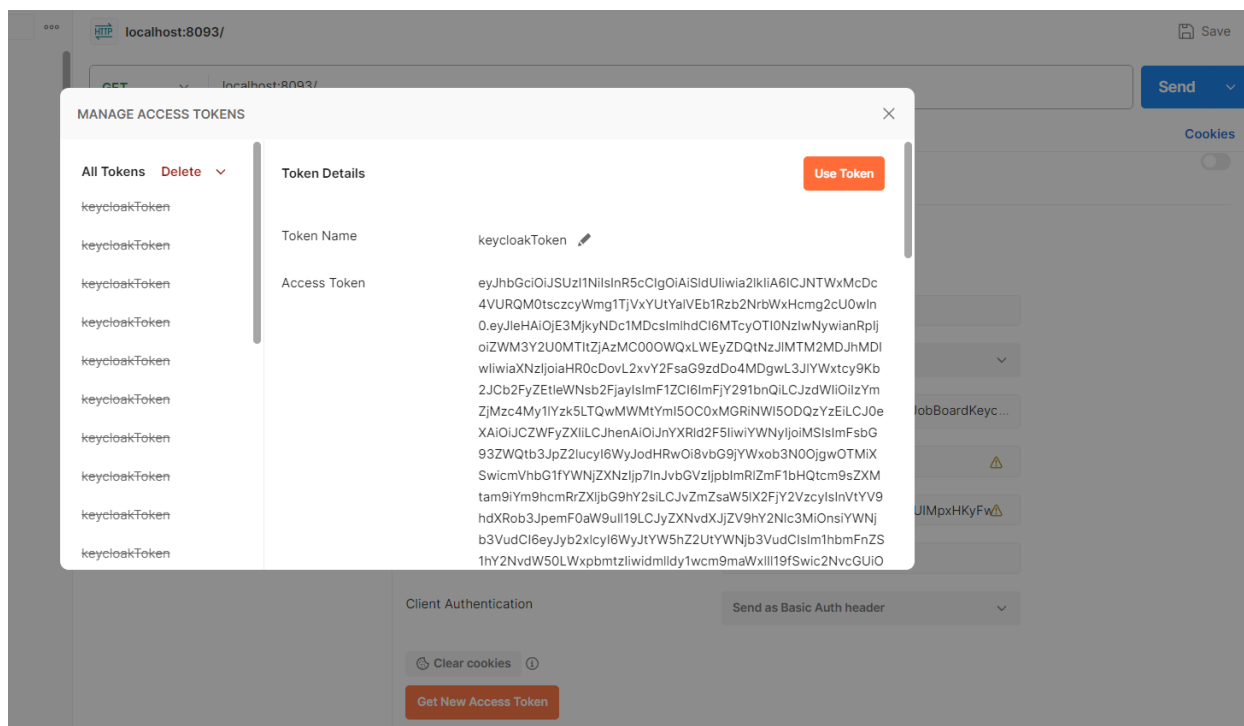
Jump to section General settings

- Client secret :

Au niveau du Gateway Client, veuillez choisir l'option Credentials. Puis, récupérer la valeur client secret



- Scope : openid offline_access
- Client Authentication : Send a Basic Auth header
- Cliquer sur Get New Access Token. Vous pouvez voir la Valeur de token générée.



- Veuillez choisir use token pour utiliser cette valeur comme clé de sécurité.
- Tester l'accès vers le Microservice Candidat comme suit :
 - Sans clé de sécurité

