

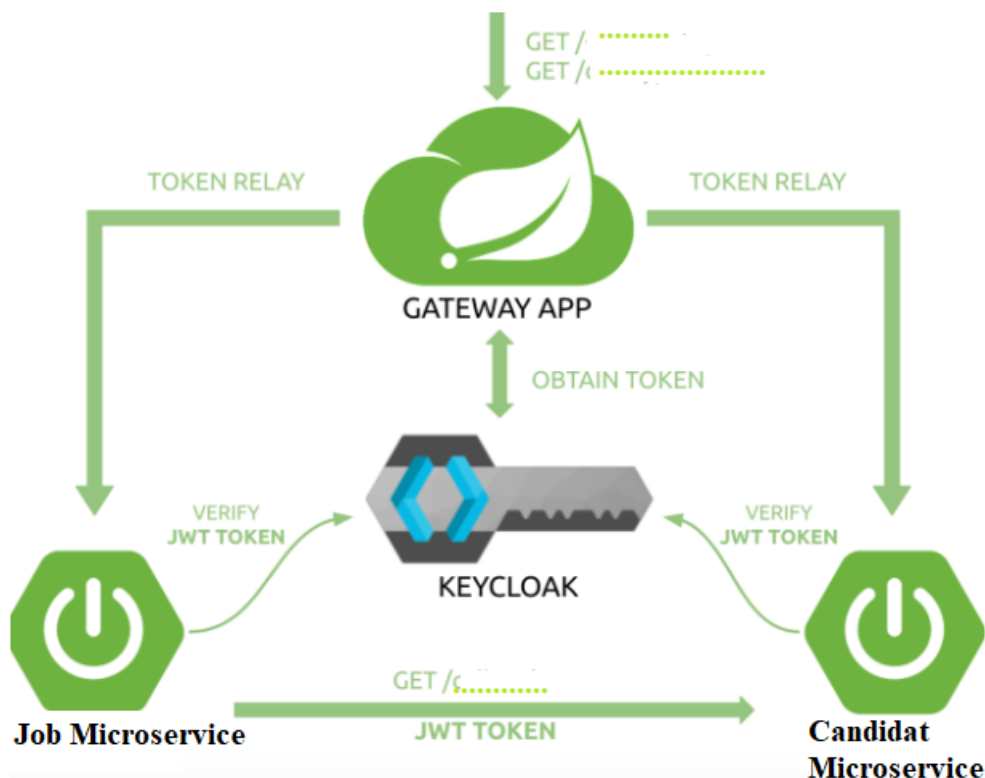
Sécurisation d'une API Gateway avec Keycloak

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**
 - clientID : gateway (nom de l'application Microservice Gateway)
 - RootURL, HomeURL et AdminURL : <http://localhost:8093> (Port de MS)

Gateway)

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

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

NB :

- L'API Gateway doit **uniquement valider les tokens** fournis par Keycloak, alors **Spring Boot Starter OAuth2 Resource Server** suffit. Cette bibliothèque Spring Boot supporte **OpenID Connect** et **OAuth2**, qui sont les protocoles utilisés par Keycloak.
- On aura besoin d'une dépendance Keycloak si on souhaite **utiliser le client Keycloak côté backend** pour interagir avec Keycloak (ex : créer des utilisateurs, gérer les sessions).

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.EnableWebFluxSecurity;
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. The browser address bar indicates the URL is localhost:8761. The page title is "spring Eureka".

System Status

| | | |
|-------------|---------|--------------------------|
| Environment | test | Current time |
| Data center | default | Uptime |
| | | Lease expiration enabled |
| | | Renews threshold |
| | | Renews (last min) |

DS Replicas

localhost

Instances currently registered with Eureka

| 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 postman, nous allons générer un token à travers Keycloak
- o Veuillez choisir Authorization-> OAuth 2.0
 - o Veuillez configurer un nouveau token

The screenshot shows the Postman interface for configuring a new OAuth 2.0 token. The 'Configure New Token' section is active, displaying the following configuration 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

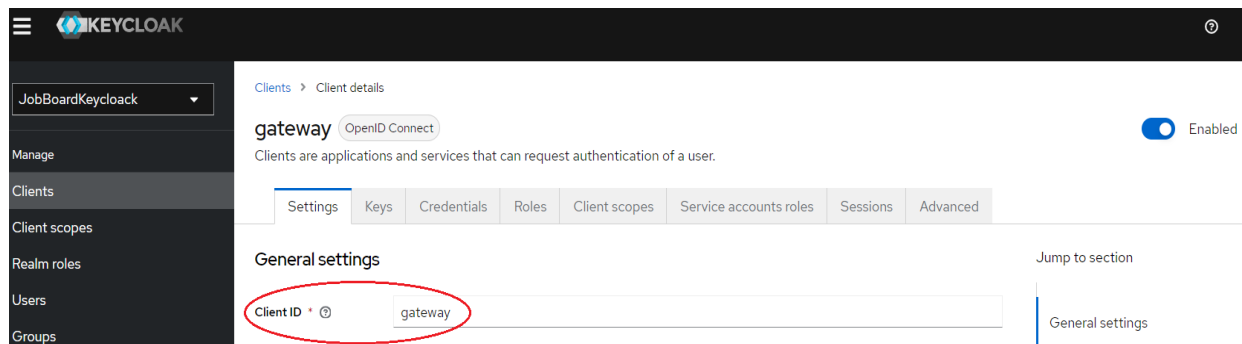
Red circles are drawn around the Grant Type, Access Token URL, Client ID, Client Secret, Scope, and Client Authentication fields in the original image.

Comme le montre la figure ci-dessus :

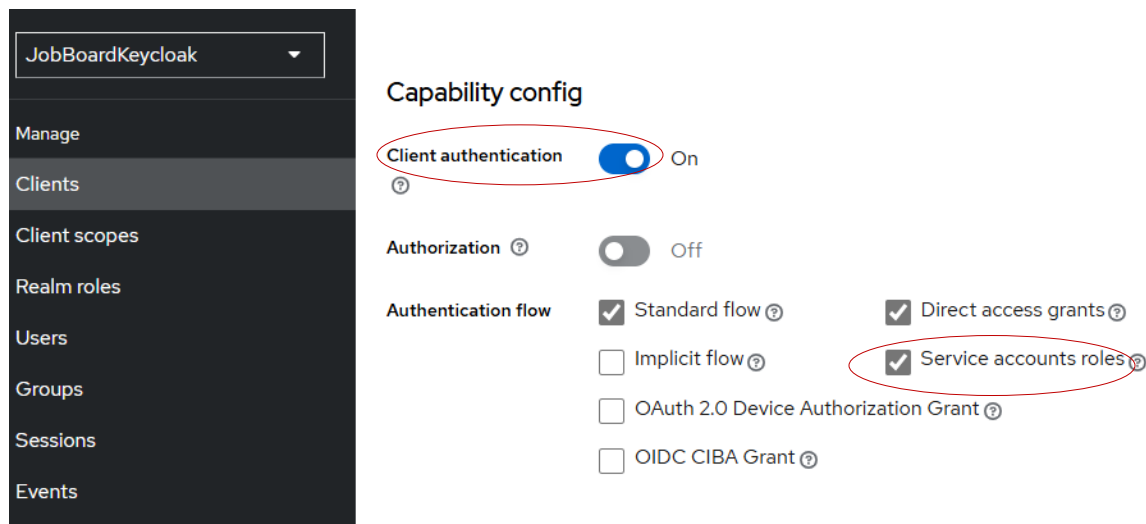
- o **Token Name** : veuillez choisir KeycloakToken
- o **Grant type** : Client Credentials
- o **Access Token URL** : Pour récupérer cette valeur à partir de l'interface de Keycloak : **Realm Settings** → **General** → **Endpoints**, puis cliquer sur le lien « OpenID EndPoint Configuration ». Récupérer ensuite 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-
```

- o **Client ID** : la valeur Client ID spécifiée dans Keycloak

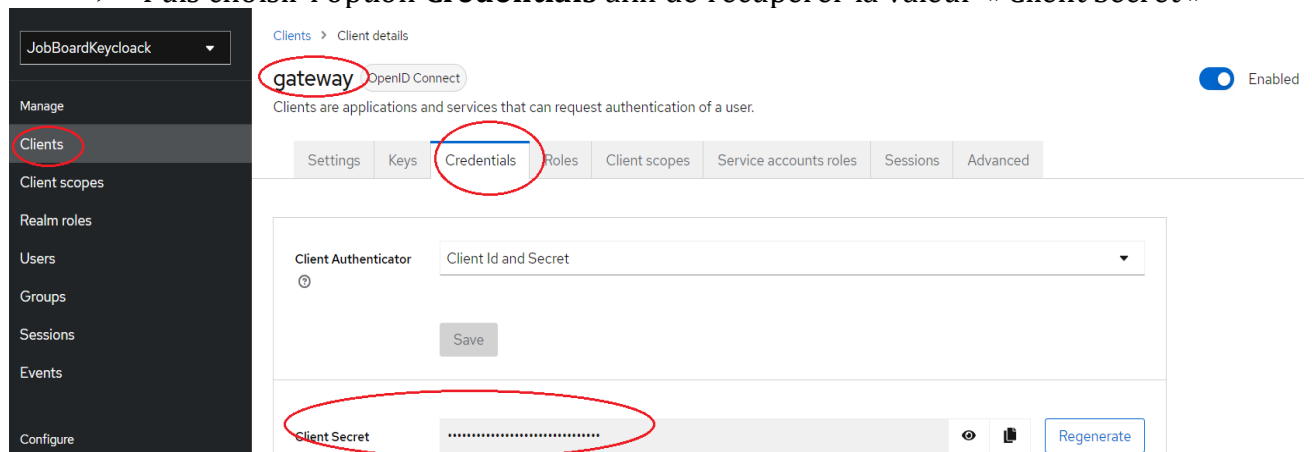


- **Client secret :**
 Au niveau du Client « gateway » :
 ➤ Activer l'authentification du client et « **Enregistrer** »:



NB: Dans l'onglet **Authentication Flow** du client, activer « **Service Account Roles** ». Cela permet au client d'utiliser un compte de service pour l'authentification.

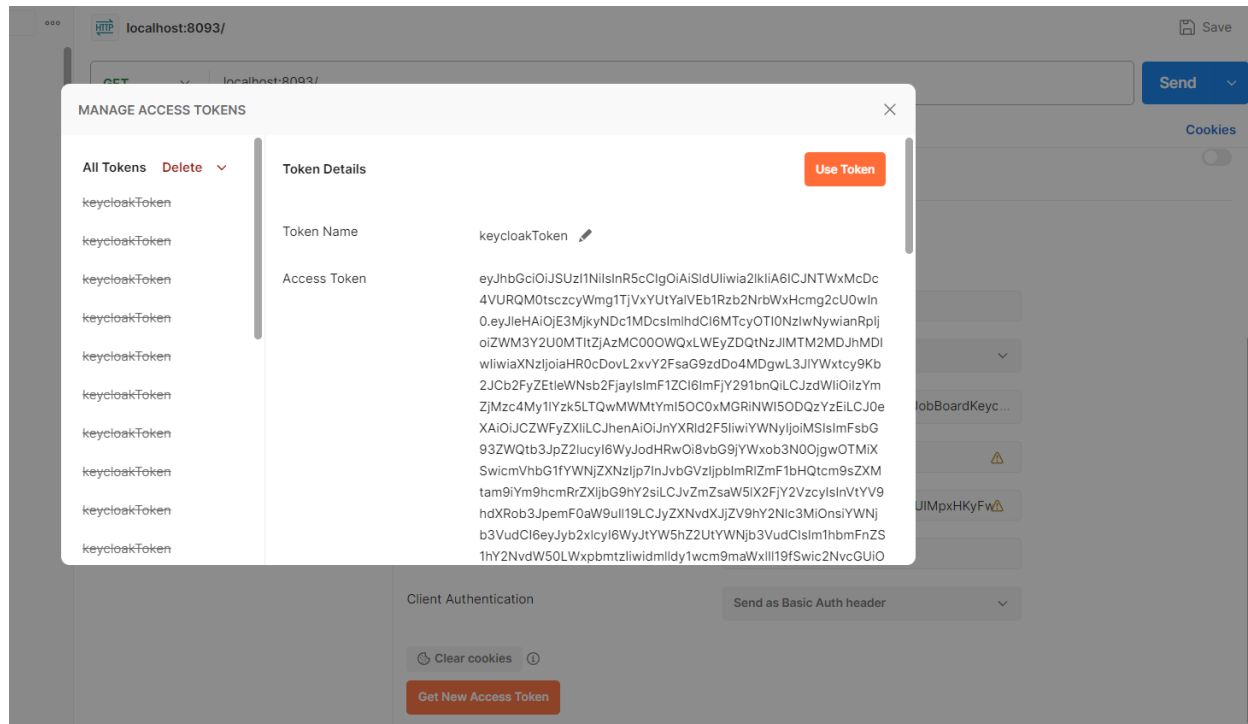
- Puis choisir l'option **Credentials** afin de récupérer la valeur « Client Secret »



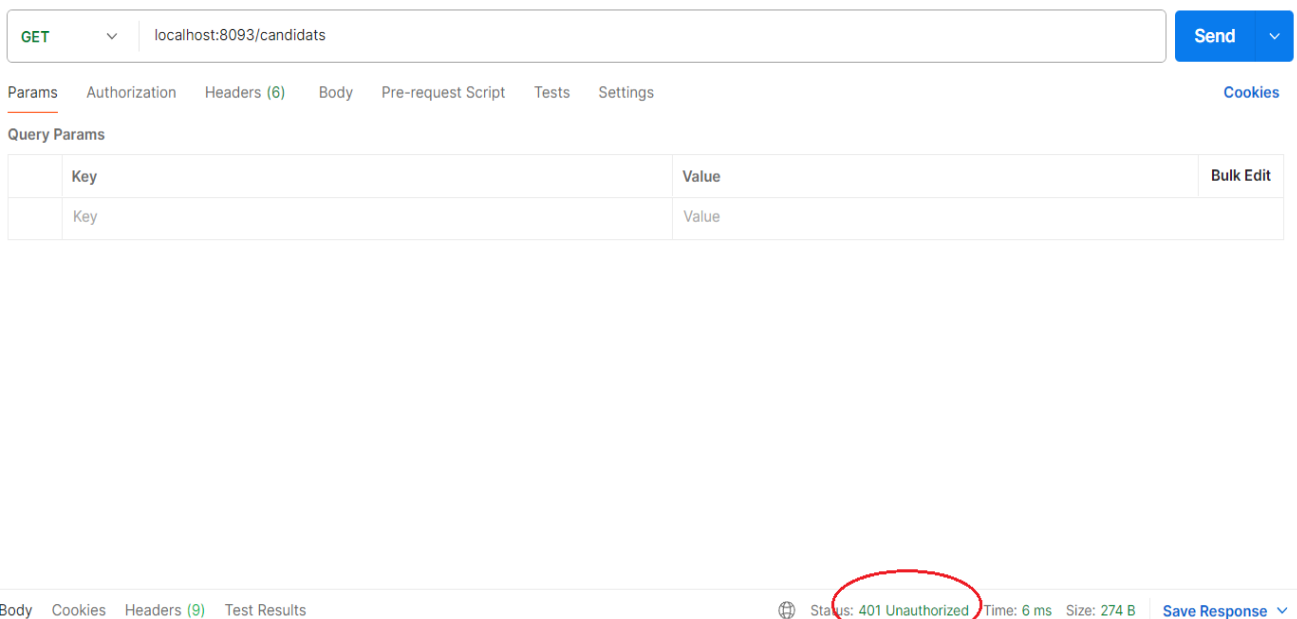
- **Scope :** openid offline_access (pour obtenir à la fois : un **ID Token** pour prouver l'identité de l'utilisateur et un **Refresh Token** pour un accès continu même après

la fin de la session utilisateur)

- **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



- Avec la clé de sécurité

The screenshot shows a REST client interface with the following details:

- URL:** `localhost:8093/candidats`
- Method:** `GET`
- Authorization:** `OAuth 2.0`
- Token:** `keycloakToken`
- Use Token Type:** `Access token`
- Header Prefix:** (empty)
- Auto-refresh token:** (checked)
- Status:** `200 OK`, Time: 31 ms, Size: 565 B
- Response Body (JSON):**

```
1 [
2   {
3     "id": 1,
4     "nom": "Mariem",
5     "prenom": "Ch",
6     "email": "ma@esprit.tn"
7   },
8   {
9     "id": 2,
10    "nom": "Sarrah",
11    "prenom": "ab",
```