	Année Universitaire : 2024-2025 **Applications Web Distribuées** **UP_WEB**

Serveur de configuration

Mise en situation

Les microservices sont composés de nombreux petits services autonomes, chacun avec sa propre configuration. Plutôt qu'un fichier de propriétés centralisé (comme le monolithe), la configuration est dispersée sur plusieurs services, s'exécutant sur plusieurs serveurs. Dans un environnement de production, où il y'a probablement plusieurs instances de chaque service, la gestion de la configuration peut devenir une tâche lourde.

La configuration centralisée est un pattern dans lequel la configuration de tous les services est gérée dans un référentiel central plutôt que d'être dispersée sur des services individuels. Chaque service extrait sa configuration du référentiel central au démarrage.

Spring Cloud Config fournit un support côté serveur et côté client pour la configuration externalisée dans un système distribué. Avec le Config Server, on peut définir un emplacement central pour gérer les propriétés externes des applications dans tous les environnements.

Objectifs

- Créer un serveur de configuration
- Configurer les Clients
- Consulter les propriétés centralisées

Etapes

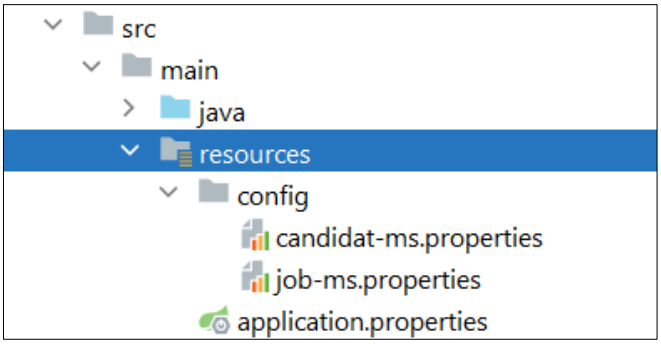
Etape 1: Créer le serveur de configuration - Config Server

1- Créer un projet spring en ajoutant les dependances **Server config** et **Eureka discovery client.**

2- Dans le main, ajouter les 2 annotations :

@enableConfigServer : pour le définir comme un service de configuration
@EnableDiscoveryClient : pour que ce serveur apparaisse dans la liste Eureka

3- Créer un dossier « **config** » ou autre nom sous resources, dans lequel on ajoutera 2 fichiers .properties **portant chacun le même nom de application.name du MS concerné** (candidat-ms.properties et job-ms.properties).
Les fichiers de configurations centralisées seront structurés ainsi :



- **application.properties** stockera les configurations de tous les microservices clients.
- **candidat-ms.properties** stockera les configurations du microservice Candidat.
- **job-ms.properties** stockera les configurations du microservice Job.

4- Ajouter les propriétés suivantes dans le fichier **application.properties** du serveur de configuration :

```
spring.application.name=config-server
server.port=8888
#eureka registration
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
eureka.client.register-with-eureka=true

#config server
spring.profiles.active=native #charger les fichiers de configuration localement
spring.cloud.config.server.native.searchLocations=classpath:/config

#spring.cloud.config.server.git.uri=https://github.com/my/servercloudconfig.git
```

NB : **classpath:/config** indique que les fichiers de configuration seront chargés depuis les ressources du serveur de configuration (c'est-à-dire le répertoire src/main/resources). config est le dossier déjà créé ci-dessus.

Etape 2: Configurer les Clients du Config Server(les micro-services)

Les microservices qui seront clients du config server (candidat et job) doivent être configurés comme suit, dans leurs fichiers **application.properties** :

```
spring.cloud.config.enabled=true

spring.config.import=optional:configserver:http://localhost:8888
```

Dans ces derniers fichiers, ajouter un message personnalisé pour chaque MS :

- **welcome.message= Welcome to Candidat MS**
- **welcome.message= Welcome to Job MS**

application.properties

```
# Job Service
spring.application.name=job-ms
server.port=8081

welcome.message=welcome to Job MS
# Datasource
spring.datasource.username=root
spring.datasource.password=root
```

application.properties

```
# Candidat Service
spring.application.name=candidat-ms
server.port=8060

welcome.message=Welcome to Candidat MS
# H2 console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

NB : **welcome.**message est une propriété clé-valeur utilisée pour stocker un message de bienvenue qui peut être injecté et affiché dans un contrôleur Spring Boot.

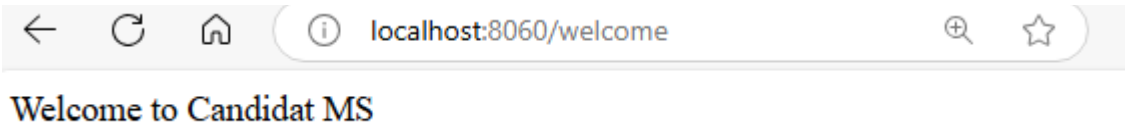
Sous le contrôleur de chaque MS, ajouter le suivant :

```
@Value("${welcome.message}")
private String welcomeMessage;


@GetMapping("/welcome")
public String welcome() {
    return welcomeMessage;
}
```

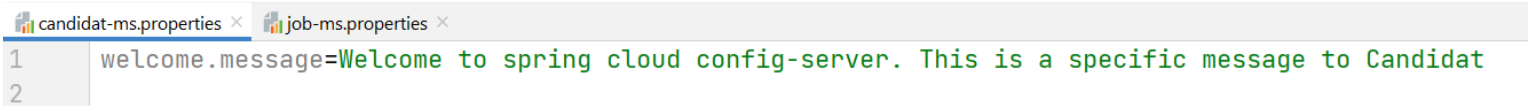
- Veuillez exécuter les projets suivants dans l'ordre:
  - **Eureka**
  - **Server de configuration**
  - **Candidat et Job**

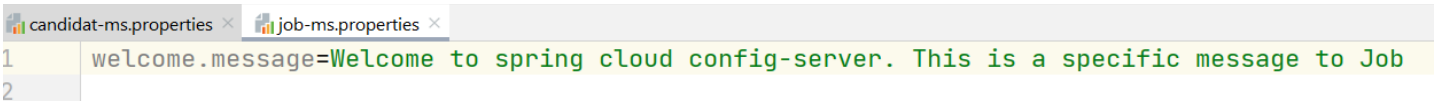
puis accéder à la nouvelle ressource identifié par **/welcome** :



Etape 3: Préparer les configurations centralisées

Les configurations centralisées des microservices, dans le serveur de configuration, seront stockées sous les fichiers **candidat-ms.properties** et **job-ms.properties** sous le répertoire **./src/main/resources/config**
Dans ces derniers fichiers, ajouter un message personnalisé pour chaque MS :





- Veuillez redémarrer le serveur de configuration.

Etape 4: Consulter les nouvelles configurations

Pour consulter les configurations prises du config server, par exemple celles du candidat on doit :

- Soit **redémarrer les microservices** pour qu'ils chargent à nouveau la nouvelle configuration.
- Soit appeler **l'endpoint** `/actuator/refresh` à partir du service pour rafraîchir la configuration sans redémarrer le service :

  Actuator est une fonctionnalité spécifique à Spring Boot utile pour observer l'état et les configurations d'une application en utilisant **`/actuator/health`**, **`/actuator/env`**, etc. Comme elle permet aussi de rafraîchir la configuration sans redéploiement en utilisant **`/actuator/refresh`**.

Dans notre projet utilisant **Config Server**, l'exposition de `/actuator/refresh` est essentielle afin de permettre aux micro-services de **recharger la configuration dynamiquement** après un changement dans le serveur de configuration.

Comment activer Actuator dans les micro-services ?

1 **Ajouter la dépendance** :

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

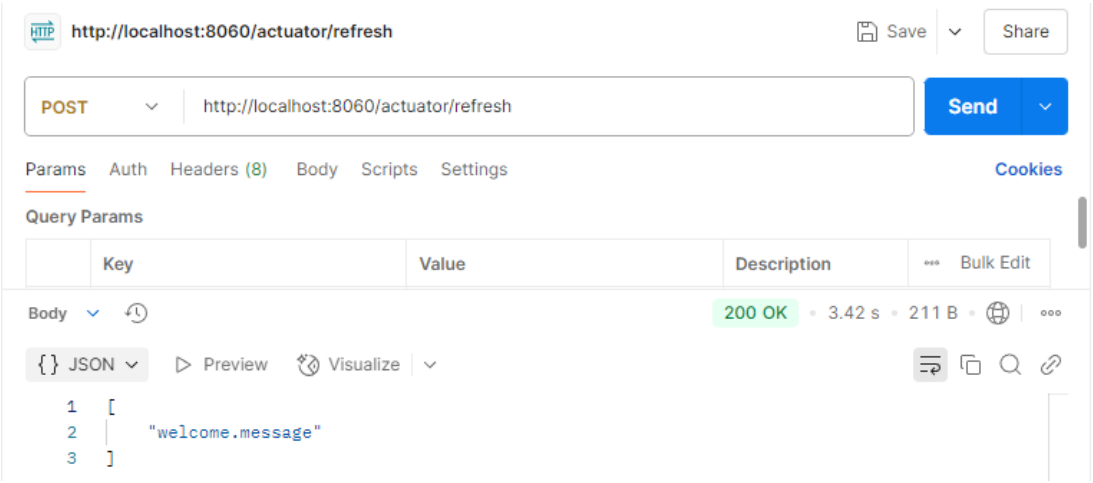
2 **Configurer application.properties** :

```
management.endpoints.web.exposure.include=refresh #ou bien * pour l'accès à tous les endpoints actuator
```

3 **Envoyer une requête POST pour rafraîchir la config** :

```
POST http://localhost:8060/actuator/refresh
```

Dans notre cas, la requête ci-dessus envoyée après la mise à jour de la propriété welcom.message donnera le suivant :



⇨ **La configuration est mise à jour dynamiquement.**

L’acces à la ressource **/welcome** donnera :

Le principal inconvénient de `/actuator/refresh` dans le cas de la configuration de la base de données est qu'il **ne rafraîchit pas les connexions à la base de données**. Par conséquent, pour que les nouveaux paramètres de la base de données soient pris en compte, tu as toujours besoin de redémarrer l'application. Cependant, des solutions comme la réinitialisation manuelle des connexions à la base de données ou l'usage de mécanismes comme **Spring Cloud Bus** assurant un rafraîchissement dynamique peuvent offrir des alternatives.