INSA de Toulouse
guermouc@insa-toulouse.fr

Consommer un service de Configuration

Précédemment, vous avez mis en place un service de configuration. Vous allez maintenant voir comment l'utiliser. Vous allez donc voir comment développer un microservice client du microservice de configuration.

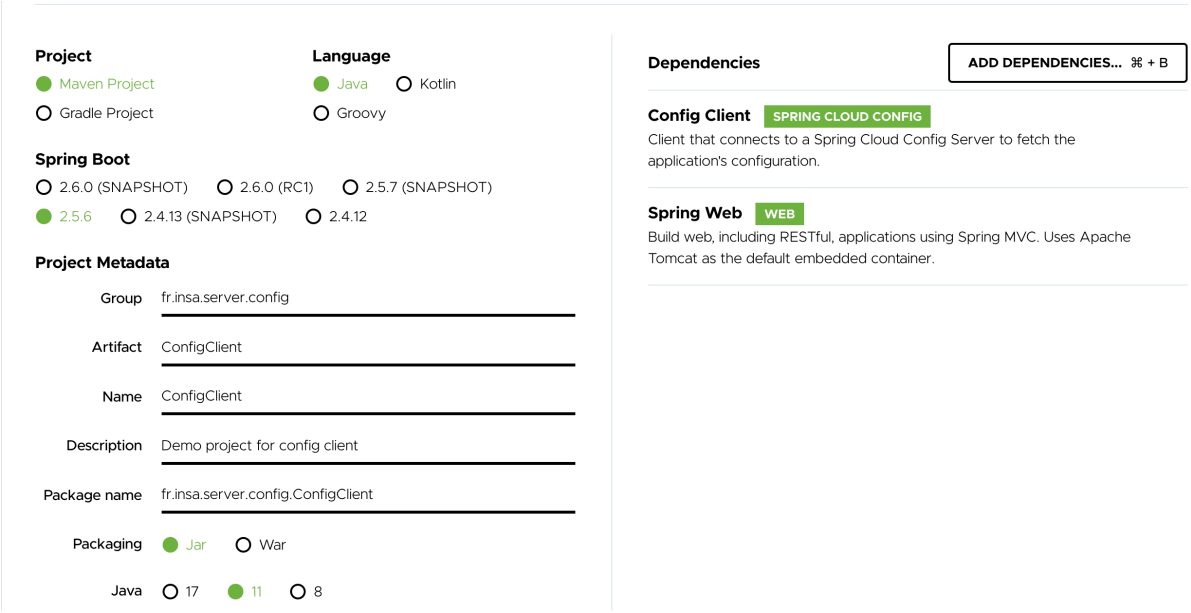
a. Compétences visées

Le but de cette étape est de vous permettre d'implémenter un client dont les paramètres de configuration sont externalisés via un microservice de configuration.

b. Client de service de configuration :

Créez un projet Spring Boot en ajoutant cette fois ci deux dépendances :

- Config client : pour que votre microservice puisse invoquer votre service de configuration
- Spring Web : pour permettre, entre autres, à votre service d'envoyer des requêtes Rest.



Importez votre projet dans votre IDE Eclipse.

On va indiquer qu'il doit solliciter le service de configuration. Rappelez vous que le port qu'on a défini pour le service de configuration était *8888*. Aussi, les fichiers de configuration que vous avez précédemment créés dans le dépôt Git était dédiés à un service appelé *client-service*. Donc votre client doit avoir ce nom.

Ajoutez dans le fichier *application.properties* le nom *client-service* à votre microservice. Aussi, vous devez indiquer l'URL du microservice de configuration en utilisant le suivant :

```
spring.config.import=optional:configserver:${SPRING_CLOUD_CONFIG_URI:http://localhost:8888}
```

Votre fichier *application.properties* devrait donc être comme suit :

application.properties

```
spring.application.name=client-service
spring.config.import=optional:configserver:${SPRING_CLOUD_CONFIG_URI:http://localhost:8888}
```

Remarque : Par défaut, la branche Git considérée est *master*. Si vous voulez dédier une autre branche à votre client dans laquelle vous allez mettre les fichiers de configuration, il suffit d'ajouter dans le fichier *application.properties* le code suivant :

```
spring.cloud.config.label=nom_de_votre_branche
```

Maintenant tout est prêt pour que vous puissiez développer un service Rest pour récupérer les paramètres de configuration, qui sont rappelez-vous :

```
db.connection=
db.host=
db.port=
server.port=
```

Créez une classe, nommée ici ***ClientConfigResource,*** qui a 4 paramètres qui correspondent aux paramètres de configuration précédents, et générez-y les getters et les setters :

```java
package fr.insa.ms.server.config.ConfigClient.resources;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class ClientConfigResource {

    private String serverPort;
    private String typeConnectionDeDB;
    private String dbHost;
    private String dbPort;


    public String getServerPort() {
        return serverPort;
    }
```

Maintenant il va falloir relier les paramètres définis dans la classe java avec les paramètres du fichier de configuration présent dans votre dépôt Git. Par exemple, le paramètre *typeConnectiondeDB* de votre classe correspond au paramètre *db.connection*. Pour cela, vous allez utiliser l'annotation `@Value("${nom_paramètre}")` comme suit :

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ClientConfigResource {

    @Value("${server.port}")
    private String serverPort;

    @Value("${db.connection}")
    private String typeConnectionDeDB;

    @Value("${db.host}")
    private String dbHost;

    @Value("${db.port}")
    private String dbPort;
```

L'annotation `@Value("${server.port}")` associée au paramètre *serverPort*, va permettre de scanner le fichier de configuration (disponible dans Git), et d'associer la valeur du paramètre de configuration *server.port* au paramètre *serverPort* que vous avez défini dans votre classe Java.

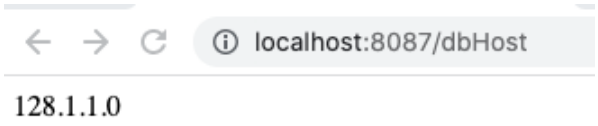
Pour récapituler, au lancement de votre microservice client, le fichier *application.properties* est d'abord lu. Comme l'URI du service de configuration est indiquée, ce service sera appelé. Ce service utilise les fichiers de configuration disponible dans un dépôt Git. Via l'annotation `@Value`, chaque paramètre concerné par cette annotation sera scanné et affecté au paramètre correspondant dans la classe java.

Maintenant, comment vous allez vérifier les valeurs de vos paramètres ? Vous allez tout simplement faire des appels Rest.

Par exemple, pour récupérer l'adresse du serveur de la base de données, allez dans le getter de dbHost et ajoutez-y l'annotation `@GetMapping("/dbHost")` :

```java
@GetMapping("/dbHost")
public String getDbHost() {
    return dbHost;
}
```

Lancez votre application spring et testez la méthode getDBHost. Rappelez vous que dans le fichier de configuration externalisé client-service.properties, vous avez mis le port 8087 (server.port=8087). Donc le client va être déployé en utilisant ce port.



Jusqu'ici, votre client a un profile par défaut (aucun profile n'est spécifié).

Imaginant que l'on veut utiliser le profile dev (les paramètres dédiés à l'environnement de développement). Il suffit tout simplement d'ajouter au fichier *application.properties* de votre client

```
spring.profiles.active=dev
```

Votre fichier *application.properties* doit être comme suit :

```
application.properties
spring.application.name=client-service
spring.profiles.active=dev
spring.config.import=optional:configserver:${SPRING_CLOUD_CONFIG_URI:http://localhost:8888}
```

Testez votre microservice pour voir quels paramètres de quel profile sont considérés. Attention, dans le fichier de configuration dédié au profil dev (i.e., *client-service-dev*), le port est 8088.

Exercice :

Vous allez essayer d'externaliser les paramètres de configuration du microservice *StudentInfoService* que vous avez précédemment créé. Concrètement, ce service devrait être connecté à une BDD relationnelle pour récupérer des étudiants. On ne vous demande pas de créer une BDD, mais de simuler la récupération des paramètres de connexion d'une BDD, puis de les afficher dans la console avant l'instanciation des étudiants.

Les paramètres à considérer sont :

db.uri : l'adresse du serveur de BDD
db.name : nom de la BDD
db.login : login pour se connecter à la BDD
db.pwd : mot de passe pour se connecter à la BDD

Le code attendu doit ressembler à ce qui est donné dans le code ci-dessous. Avant de retourner l'étudiant recherché, vous devez afficher les paramètres de la BDD (en affichant par exemple son nom et son URI).

```
@GetMapping("/{idStudent}")
public StudentInfos getInfoStudent(@PathVariable("idStudent") int id) {
    List<StudentInfos> etudiantInfos = Arrays.asList(
                                    new StudentInfos(0,"Godart","Noemie","12/12/1992"),
                                    new StudentInfos(0,"Perrin","Ania","10/02/1993"),
                                    new StudentInfos(0,"Azi","Sana","22/05/1992"),
                                    new StudentInfos(0,"Yala","Nelia","12/06/1994")
                                    );
    System.out.println("Connection à la BDD : "+this.getDbName()+" à l'URI :"+this.getDbURI());

    return etudiantInfos.get(id);

    }
```

Aussi, externalisez le numéro de port (qui est 8081). N'oubliez pas de le supprimer du fichier application.properties.

Remarque :
N'oubliez pas d'ajouter les dépendances nécessaires au fichier pom.xml. Il s'agit de toutes les parties liées à Spring Cloud pour que votre microservice puisse être client du serveur de

configuration. Le plus simple pour vous c'est de reprendre le pom.xml du client que vous venez de créer et de copier les dépendances et tous les éléments liés à Spring Cloud dans le fichier *pom.xml* du projet StudentInfoService.