INSA de Toulouse
guermouc@insa-toulouse.fr

Externalisation des configurations

Après la création d'un microservice simple, vous avez travaillé progressivement sur :

- Création d'une collaboration de microsercices
- Mise en place d'un serveur de découverte
- Instanciation multiple et répartition de charge

Pour le moment, votre architecture est constituée d'un service de découverte et de 3 microsercices. Cette architecture va évoluer pour y intégrer un service de configuration.

Question : C'est quoi l'intérêt de définir un service de configuration ? Si vous ne savez pas répondre, revoyez le cours avant de continuer.

a. Compétences visées

Le but de cette étape est de vous permettre d'externaliser la gestion des configurations via un microservice de configuration.

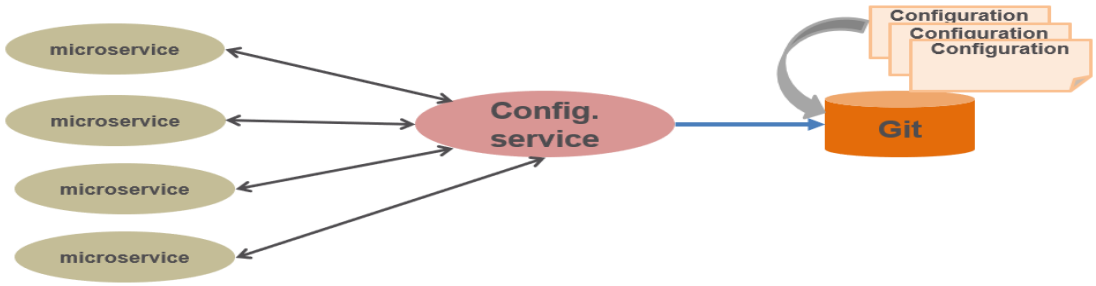
b. Microservice de configuration:

Vous allez maintenant reprendre votre architecture pour la rendre conforme aux règles de bonnes pratiques. A l'issue de ce tutoriel, votre architecture sera constituée de :

- 3 microservices
- 1 microservice de découverte
- 1 microservice de configuration

Les microservices peuvent avoir des paramètres de configuration (par exemple le numéro de port, les paramètres de connexion des BDDs, …etc). Jusqu'à maintenant, vous aviez utilisé le fichier *application.properties* pour y indiquer par exemple le port d'écoute. Avec cette méthode, à chaque fois que vous voulez mettre à jour les paramètres de configuration, vous devez arrêter votre microservice, modifier les paramètres, le recompiler, et le redéployer. A l'échelle d'une application, cela peut présenter d'énormes inconvénients.
Pour éviter cela, la solution consiste à externaliser les paramètres de configuration via un microservice de configuration.

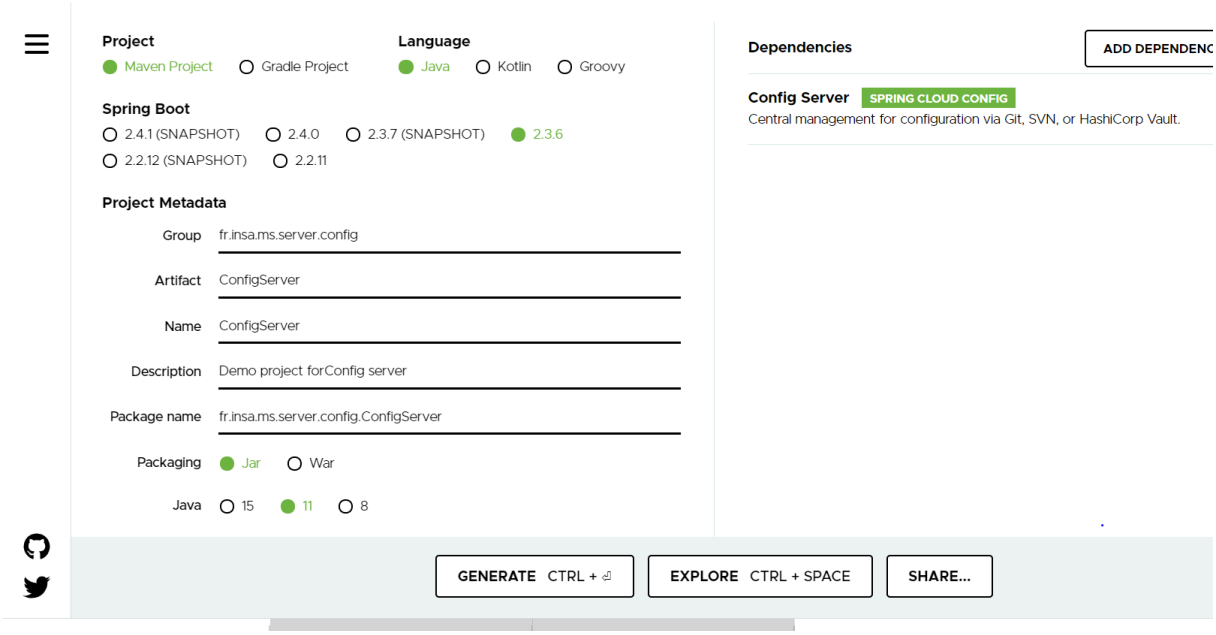
Le schéma suivant résume une architecture qui intègre un service de configuration. Les fichiers de configurations sont placés dans un dépôt Git. Les paramètres de configuration sont fournis aux différents microservices via le microservice de configuration.



INSA de Toulouse
guermouc@insa-toulouse.fr

c. Microservice de configuration:

Comme vous avez fait précédemment, créez un projet Spring Boot en ajoutant cette fois la dépendance Config Server de Spring Cloud. Comme vous pouvez le lire dans la description de la dépendance, elle permet de gérer des configurations en se basant sur par exemple Git :



Importez votre projet dans votre IDE Eclipse. Vous remarquerez dans le fichier pom.xml de votre projet la présence de la dépendance *config-server* de *spring Cloud* qui va vous permettre de définir votre service de configuration :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>fr.insa.ms.server.config</groupId>
    <artifactId>ConfigServer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>ConfigServer</name>
    <description>Demo for config serverfor Spring Boot</description>

    <properties>
        <java.version>11</java.version>
        <spring-cloud.version>Hoxton.SR9</spring-cloud.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
        </dependency>
```

Retournez à l'application principale de Spring Boot et ajoutez l'annotation `@EnableConfigServer` pour que votre microservice joue le rôle de service de configuration :

```
package fr.insa.ms.server.config.ConfigServer;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }

}
```

Remarque : Rappelez-vous que les fichiers de configuration doivent être dans un dépôt Git. Vous aurez donc besoin d'un dépôt Git. Soit vous utilisez un dépôt que vous avez déjà, soit vous en créez un dédié à vos tests. Pour ceux qui ne savent pas comment en créer un dépôt Git et l'associer à Github, vous pouvez trouver deux tutoriels sur moodle sous la section *Maven et Git.*

Avec Spring Cloud, on peut créer des fichiers de configuration dédiés à différents environnements de développement. Cela permet d'avoir des paramètres de configuration en fonction de l'environnement (développement, test, production, …etc). Cela correspond à ce qui est appelé ***profile*** dans Spring Cloud.

Ainsi vous pouvez avoir par exemple :

- dev : Developement
- test : Test
- qa : Quality assurance
- prod : Production
- et vous avez le profil « default » : le profil par défaut est utilisé quand aucun profile n'est spécifié.

Dans ce qui suit, vous allez voir les deux profils : *default* et *dev*. Pour cela, créez deux fichiers **client-service.properties** et **client-service-dev.properties.**

Cela veut dire qu'ultérieurement, un microservice client appelé *client-service* va utiliser votre microservice de configuration. Le fichier client-service-**dev** est dédié au profil *dev* et le fichier client-service est dédié au profil *par défaut*.
Ici on a créé un dépôt dans lequel on a mis les deux fichiers de configuration avec des paramètres simulés comme suit :

client-service.properties :

```
db.connection=type de connection
```

```
db.host=128.1.1.0
db.port= 1200
server.port=8087
```

client-service-dev.properties :

```
db.connection=type de connection dev
db.host=128.2.2.0
db.port= 1400
server.port=8088
```

Dans le fichier de configuration de votre microservice *application.properties* de votre microservice, vous allez indiquer le port (ici 8888) ainsi que l'URI du Git dans lequel se trouve vos fichiers de configuration :

```
server.port=8888
spring.cloud.config.server.git.uri=https://github.com/nggei/Config.git
```

Enregistrez tout et lancez votre projet (application spring boot).
L'URL du service de configuration est comme suit :

<adresse-serveur>/<nom-fichier>/<profile> tel que :

- adresse-serveur : correspond à l'adresse du serveur de déploiement. Ici ce sera localhost :8888
- nom-fichier : correspond au nom du fichier de configuration dont le nom doit correspondre au nom du service client. Ici client-service
- profile : correspond au profile utilisé. Dans notre cas, on peut avoir deux profiles soit *dev* soit *default*.

Testez le profile default :

localhost:8888/client-service/default


```
// 20211108133409
// http://localhost:8888/client-service/default

{
  "name": "client-service",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "0968501292b7daee877546ebb106460775532a57",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/nggei/Config.git/client-service.properties",
      "source": {
        "db.connection": "type de connection",
        "db.host": "128.1.1.0",
        "db.port": "1200",
        "server.port": "8087"
      }
    }
  ]
}
```

Puis testez le profile dev :


```
{
  "name": "client-service",
  "profiles": [
    "dev"
  ],
  "label": null,
  "version": "aa2e51f6768d6e1b91fb7304fa97197d257f3583",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/GnawalIL/Config.git/client-service-dev.properties",
      "source": {
        "db.connection": "type de connection dev",
        "db.host": "128.2.2.0",
        "db.port": "1400",
        "server.port": "8088"
      }
    },
    {
      "name": "https://github.com/GnawalIL/Config.git/client-service.properties",
      "source": {
        "db.connection": "type de connection",
        "db.host": "128.1.1.0",
        "db.port": "1200",
        "server.port": "8087"
      }
    }
  ]
}
```


En externalisant les paramètres de configuration, vous pouvez les modifier sans pour autant arrêter et redémarrer vos microservices.
Votre service de configuration est prêt. La prochaine étape consiste à créer un service client qui consomme le service de configuration. Autrement dit, un service vous permettant de solliciter ce service de configuration pour lire les paramètres de configuration.

Remarque : Le service de configuration utilise la branche master de Git. Vous avez la possibilité de dédier une branche par client. Dans la suite, vous allez voir comment le faire.