

TP N°3 :

Spring Cloud Config :

**Externalisation et Centralisation de la configuration des
micro-services : Use Case avec Github**

1. Prérequis

- IntelliJ ultimate
- Maven 3.x ;
- Spring Boot 3.x
- JDK 17 ;
- Connection à Internet pour permettre à Maven de télécharger les dépendances nécessaires (Spring Boot 3.x , ...).
- POSTMAN ou un autre outil pour tester les méthodes POST, PUT et DELETE.

2. Objectifs

1. Externalisation et centralisation de la configuration :

✓ Mise en œuvre d'un serveur de configuration basé sur Spring Cloud Config :

[@EnableConfigServer](#)

✓ Paramétrage d'un Repo local synchronisé avec Github

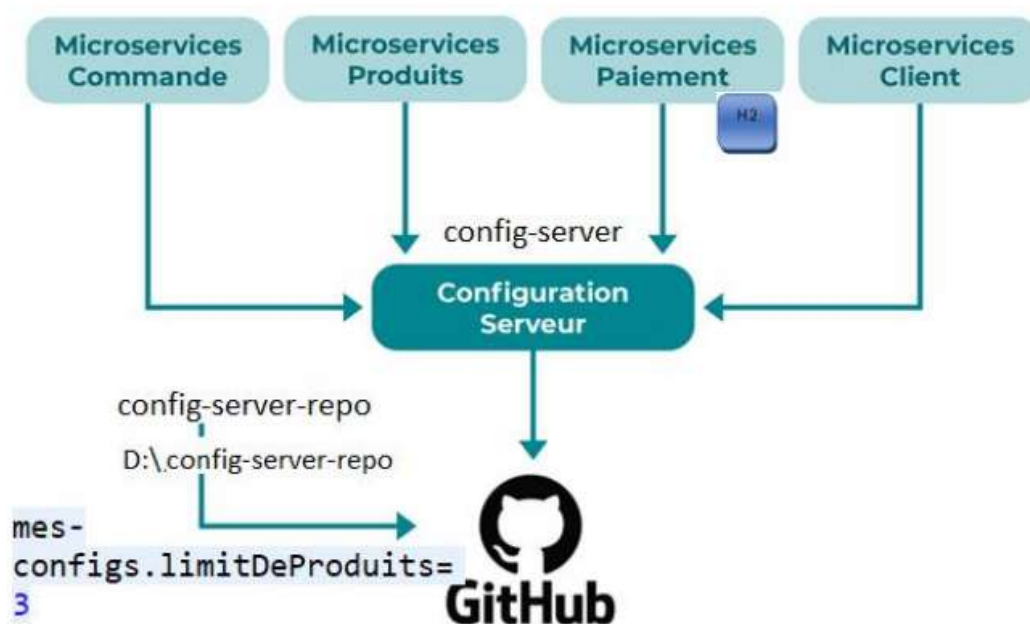
✓ Communication entre le serveur de configuration et Github.

✓ Développer le microservice « Produit »: [@EnableConfigurationProperties](#) et

[@ConfigurationProperties](#)

✓ Lier le MS-Produits avec le serveur de configuration

3. Architecture de mise en œuvre



4. Démarche de développement de l'application :

1. Use case : Le micro-service « Produits » permet de réaliser les opération CRUD et permet en outre d'afficher la liste des produits avec une taille limite configurée au niveau du fichier de configuration : « mes-configs.limitDeProduits= 3 »
2. On va se concentrer sur le micro-service « Produits », et on pourra généraliser la démarche de développement aux autres micro-services.
3. Création du Repository local et le synchroniser avec Github : « config-server-repo »
4. Création du projet Spring Cloud « Server Config » et paramétrage avec Github
5. Adapter le micro-service « Produits » à communiquer avec le Server Config
6. Dérouler les tests de bon fonctionnement

5. Développement du micro-service « Produit »

- a. Création du projet Maven : **microservice-produits**
- b. Classe principale Spring Boot : **com.mproduits.MproduitsApplication**
- c. Créer les sous packages correspondants à l'architecture micro-service :

→ On peut tester le bon fonctionnement unitaire du microservice « Produit » avec la configuration interne du fichier « **application.properties** » :

```
spring.application.name=microservice-produits
#Configurations H2
spring.jpa.show-sql=true
spring.h2.console.enabled=true
#defini l'encodage pour data.sql
spring.datasource.sql-script-encoding=UTF-8
#Les configurations externalisés
mes-configs.limitDeProduits= 5
#Configuration Actuator(optionnel pour le moment)
management.endpoints.web.exposure.include=refresh
```

→ Par la suite et pour les besoins de l'externalisation de la configuration on va splitter le fichier

« application.properties » comme suit :

1. Renommer « **application.properties** » en « **bootstrap.properties** » et le garder dans le projet du microservice-produits:

```
spring.application.name=microservice-produits

#URL de Spring Cloud Config Server

spring.cloud.config.uri=http://localhost:9101

#Configuration Actuator

management.endpoints.web.exposure.include=refresh
```

2. Créer le fichier « microservice-produits.properties » au niveau du repository local pour pouvoir le synchroniser avec Github par la suite « D: \config-server-repo »

```
#Configurations H2

spring.jpa.show-sql=true

spring.h2.console.enabled=true

#defini l'encodage pour data.sql

spring.datasource.sql-script-encoding=UTF8

#Les configurations exetrenalisés

mes-configs.limitDeProduits= 5
```

3. Le fichier pom.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>3.2.0</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.mproduits</groupId>
  <artifactId>microservice-produits</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>microservice-produits</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2023.0.0</spring-cloud.version>
  </properties>
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </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>
    </plugin>
  </plugins>
</build>
</project>
```

```
package com.mproduits;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableConfigurationProperties
@EnableDiscoveryClient

public class MproduitsApplication {

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

}
```

```
package com.mproduits.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Product {

    @Id
    @GeneratedValue
    private int id;
    private String titre;
    private String description;
    private String image;
    private Double prix;

    public Product() {
    }
}
```

```
    public Product(int id, String titre, String description, String image, Double prix) {
        this.id = id;
        this.titre = titre;
        this.description = description;
        this.image = image;
        this.prix = prix;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getTitre() { return titre; }
    public void setTitre(String titre) { this.titre = titre; }
    public String getDescription() { return description; }
```

```

public void setDescription(String description) { this.description = description; }

public String getImage() { return image; }

public void setImage(String image) { this.image = image; }

public Double getPrix() { return prix; }

public void setPrix(Double prix) { this.prix = prix; }

@Override

public String toString() {

return "Product{" + "id=" + id + ", titre=" + titre + "\" + ", description=" + description + "\" +
",
image=" + image + "\" + ", prix=" + prix + "}"; }

}

```

```

package com.mproduits.dao;

import com.mproduits.model.Product;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

//@Repository est une annotation Spring pour indiquer que la classe est un bean,
//et que son rôle est de communiquer avec une source de données (en l'occurrence la base
de données).

//@Repository est une spécialisation de @Component.

//Tout comme @Component, elle permet de déclarer auprès de Spring qu'une classe est un
bean à exploiter.

```

@Repository

```

public interface ProductDao extends JpaRepository<Product, Integer>{

}

package com.mproduits.configurations;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.cloud.context.config.annotation.RefreshScope;

import org.springframework.stereotype.Component;

```

@Component

@ConfigurationProperties("mes-configs")

@RefreshScope

```
public class ApplicationPropertiesConfiguration {  
    // correspond à la propriété « mes-configs.limitDeProduits » dans le fichier de configuration  
    //du MS  
  
    private int limitDeProduits;  
  
    public int getLimitDeProduits() {  
        return limitDeProduits;  
    }  
  
    public void setLimitDeProduits(int limitDeProduits) {  
        this.limitDeProduits = limitDeProduits;  
    }  
}
```

```
package com.mproduits.web.controller;  
  
import com.mproduits.configurations.ApplicationPropertiesConfiguration;  
import com.mproduits.dao.ProductDao;  
import com.mproduits.model.Product;  
import com.mproduits.web.exceptions.ProductNotFoundException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.actuate.health.Health;  
import org.springframework.boot.actuate.health.HealthIndicator;  
import org.springframework.cloud.context.config.annotation.RefreshScope;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RestController;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
public class ProductController implements HealthIndicator {  
    @Autowired
```

```
ProductDao productDao;

@Autowired

ApplicationPropertiesConfiguration appProperties;

// Affiche la liste de tous les produits disponibles

@GetMapping(value = "/Produits")

public List<Product> listeDesProduits() {

System.out.println(" ***** ProductController listeDesProduits() ");

List<Product> products = productDao.findAll();

if (products.isEmpty())

throw new ProductNotFoundException("Aucun produit n'est disponible à

la vente");

List<Product> listeLimitee = products.subList(0,

appProperties.getLimitDeProduits());

return listeLimitee;

}

// Récupérer un produit par son id

@GetMapping(value = "/Produits/{id}")

public Optional<Product> recupererUnProduit(@PathVariable int id) {

System.out.println(" ***** ProductController

recupererUnProduit(@PathVariable int id) ");

Optional<Product> product = productDao.findById(id);

if (!product.isPresent())

throw new ProductNotFoundException("Le produit correspondant à l'id "

+ id + " n'existe pas");

return product;

}

@Override

public Health health() {

System.out.println("***** Actuator : ProductController health() ");
```

```
List<Product> products = productDao.findAll();  
if (products.isEmpty()) {  
    return Health.down().build();  
}  
return Health.up().build();  
}  
}
```

6. Configuration du Repository local

a. Création d'un dossier local pour manipuler les différents fichiers de configuration de tous les micro-services de l'application :

b. Remarquer que ce dossier :

- Contient les fichiers de tous les microservices
- Est un repository local de Github
- Projet Eclipse pour faciliter la synchronisation avec Github
- Le nom du fichier « microservice-produit.properties » doit correspondre exactement au nom donné au Microservice-produits dans bootstrap.properties
- Grâce à cette correspondance de noms, le serveur de configuration fera le lien entre ce fichier et le microservice correspondant.

c. « microservice-produits.properties » :

- Pointer vers le dossier créé : cd config-server-repo/
- Initialiser un nouveau dépôt GIT local que nous allons « pusher » plus tard : git init ou bien via Eclipse :
- Ajoutez les fichiers du dossier au GIT :git add
- Git commit puis git push du fichier

h. Créer le repo Github «mcommerce-config-repo »

i. Au niveau du Password, il faut mettre le token GIT au lieu du mot de passe. Le token peut être récupéré depuis : <https://github.com/settings/tokens>

j. Vérifier que le push s'est bien déroulé :

Branch : master

7. Création de Spring Server Config

a. Starter : SPRING CLOUD CONFIG

```
<?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>2.7.16</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.mcommerce</groupId>
<artifactId>config.server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>config.server</name>
<description>Demo project for Spring Boot</description>
<properties>
<java.version>1.8</java.version>
<spring-cloud.version>2021.0.8</spring-cloud.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
```

```
</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>
</plugin>
</plugins>
</build>
</project>
```

```
spring.cloud.compatibility-verifier.enabled=false
spring.application.name=config-server
server.port=9101
spring.cloud.config.server.git.uri=https://github.com/xxxxx/mcommerce
-config-repo.git
spring.cloud.config.server.git.default-label=master
#Log level configuration
```

```
logging.level.root=INFO
```

```
logging.level.com.mcommerce.config.server=INFO
```

```
logging.level.org.springframework.boot.web.embedded.tomcat=INFO
```

- Convention : Tous les Edge Microservices seront sur des ports commençant par 91.

- Celui de config-server est 9101.

b. Ajouter l'annotation `@EnableConfigServer` pour indiquer que ce microservice comme étant un serveur de configuration :

c. `http://localhost:9101/microservice-produits/default`

d. Si besoin, Changer la branche git par défaut de main à master

e. `http://localhost:9101/microservice-produits/default/master`

f. Explication :

a. Le serveur de configuration Spring Cloud Config est allé chercher le fichier de configuration dans le GitHub et expose une API qui répond à l'URL `"/nom-dumicroservice/default/branche"`.

b. Il fournit ensuite sous format JSON toutes les configurations présentes dans le fichier.

8. Lier le Microservice-produit au serveur de configuration Spring Cloud Config:

a. Pour rappel, on a le dépôt Git relié à notre serveur Spring Cloud Config.

b. On va demander au Microservice-produits de récupérer le contenu de ces fichiers de configuration depuis le serveur de configuration externalisé et non pas en interne.

c. Appliquer la procédure de Renommage «application.properties» en «bootstrap.properties» décrite en haut

d. Démarrage Microservice-produit :

→ Remarquez : Fetching config from server at : `http://localhost:9101`

e. `http://localhost:9001/Produits` :

On obtient une liste de 3 produits uniquement et qui correspond à la configuration du microservice