

---

## **TP N°3:**

### **Spring Cloud Config : Externalisation et Centralisation de la configuration des microservices : Use Case avec Github**

---

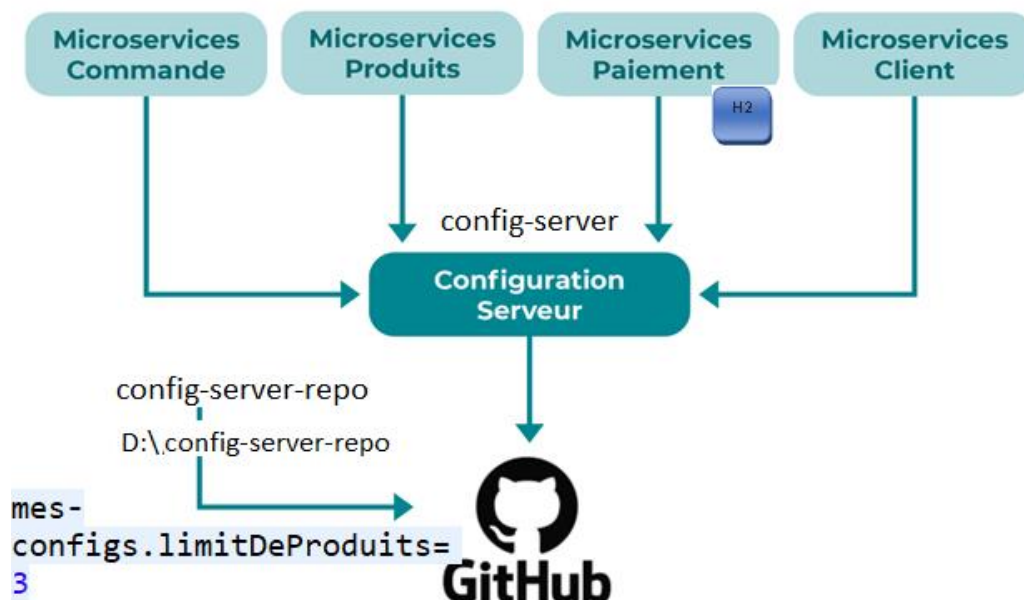
## 1. Prérequis

- Eclipse Mars (ou autre 202X...) avec le plugin Maven 3.x ;
- JDK 1.8;
- Connection à Internet pour permettre à Maven de télécharger les dépendances nécessaires (Spring Boot 2.17, ...).
- 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

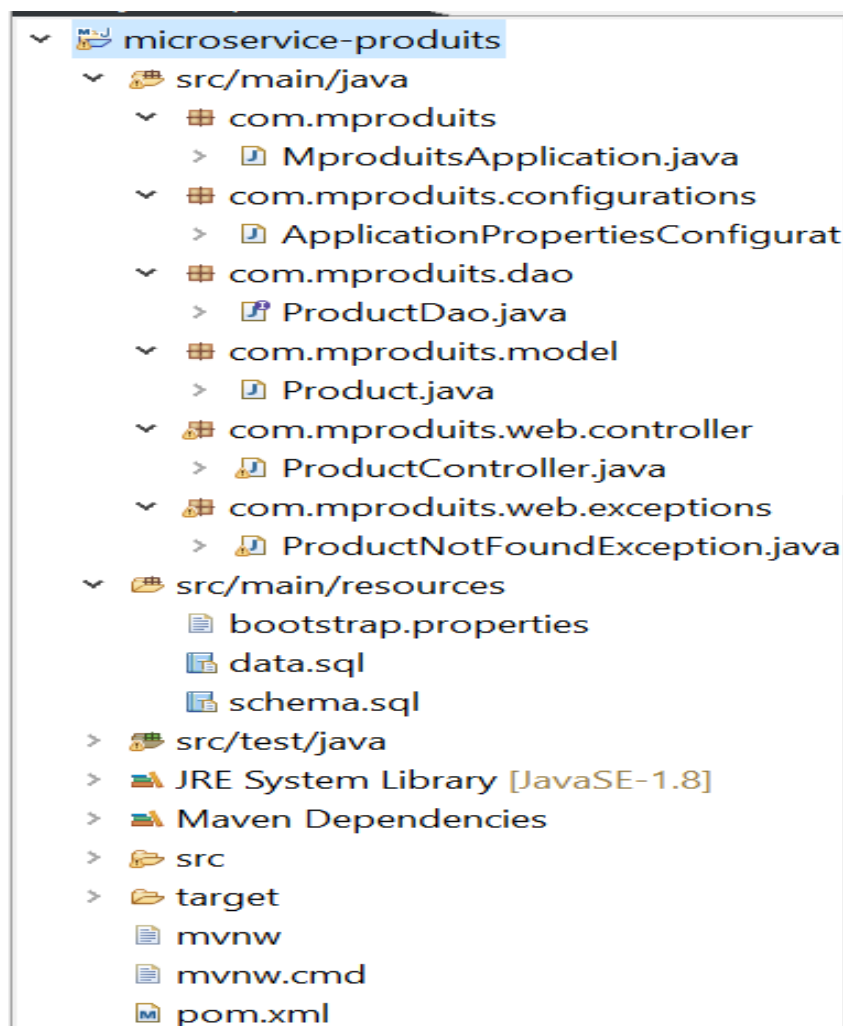


#### 4. Démarche de développement de l'application :

1. Use case : Le microservice « 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 microservice « Produits », et on pourra généraliser la démarche de développement aux autres microservices.
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 microservice « Produits » à communiquer avec le Server Config
6. Dérouler les tests de bon fonctionnement

#### 5. Développement du microservice « 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 microservice :



- ➔ 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 exetrenalisés
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=UTF-8
#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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mproduits</groupId>
  <artifactId>mproduits</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```

    <name>mproduits</name>
    <description>Microservice de gestion des
produits</description>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.16</version>
        <relativePath/>
    </parent>
    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <spring-cloud.version>2021.0.8</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-
jpa</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.cloud</groupId>
            <artifactId>spring-cloud-starter-
config</artifactId>
        </dependency>
        <!-- Add the following dependency to avoid : No spring.config.import
property has been defined -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-bootstrap</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-

```

```

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>
<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
    </repository>
</repositories>
</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() {

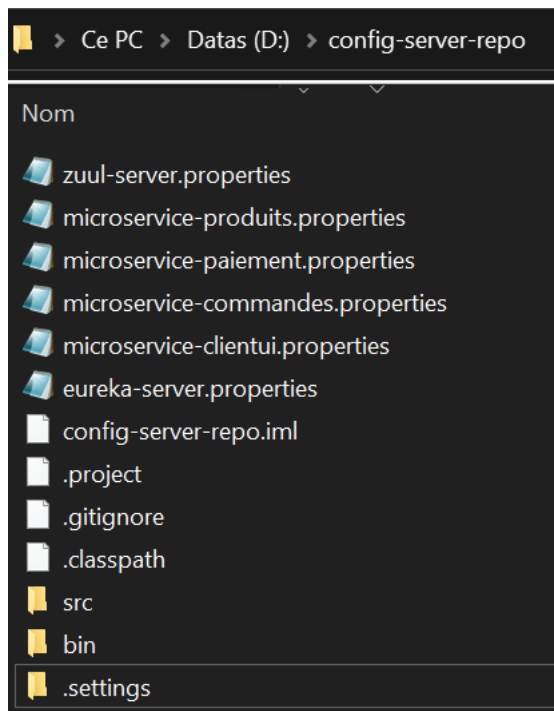
        System.out.println("***** Actuator : ProductController health() ");

        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 microservices de l'application :



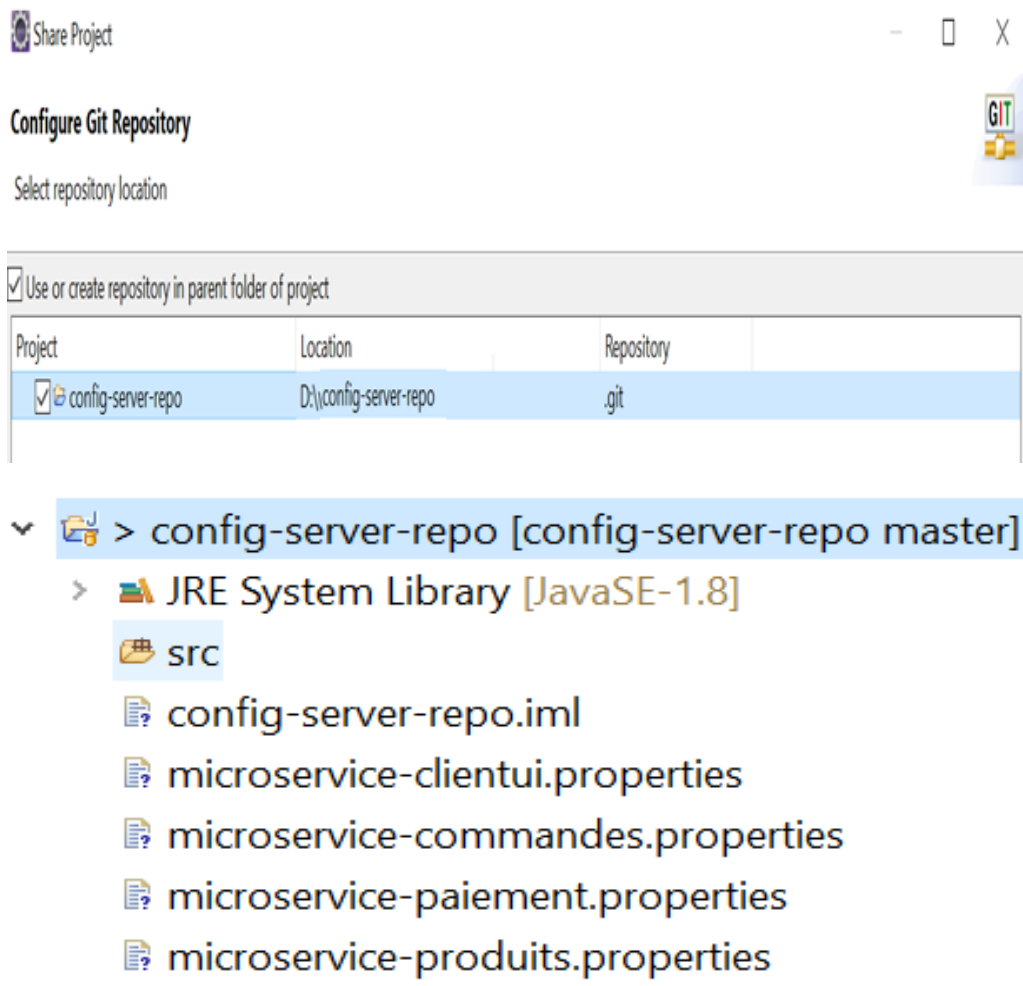
- b. Remarquer que ce dossier :
- Contient les fichiers de tous les microservices
  - Est un repository local de Github
  - Projet Eclipse pour faciliter la synchronisation avec Github
  - Le nom du fichier « **microservice-produit.properties** » doit correspondre exactement au nom donné au Microservice-produits 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 » :

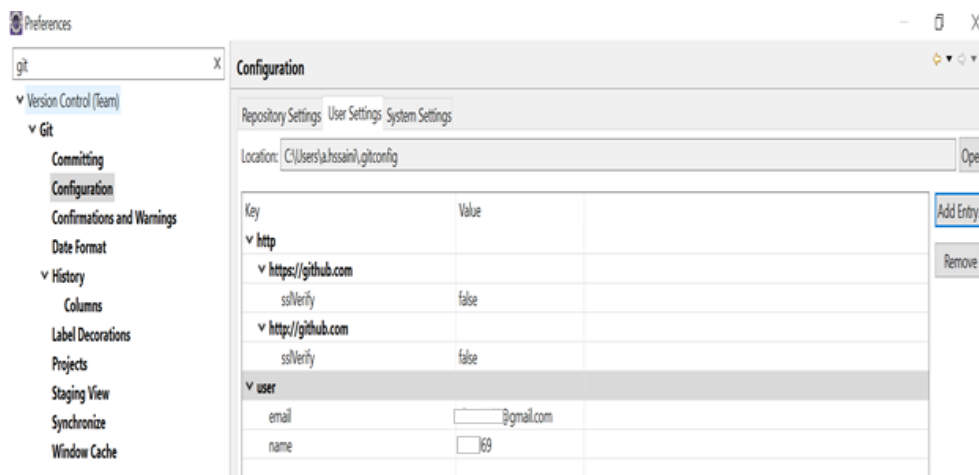
```
1 server.port=9001
2
3 #Configurations H2
4 spring.jpa.show-sql=true
5 spring.h2.console.enabled=true
6
7
8 spring.datasource.url=jdbc:h2:mem:testdb
9 spring.datasource.driverClassName=org.h2.Driver
10 spring.datasource.username=sa
11 spring.datasource.password=
12 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
13 spring.jpa.hibernate.ddl-auto=none
14
15 #Nos configurations
16 mes-configs.limitDeProduits= 4
```

d. Pointer vers le dossier créé : `cd config-server-repo/`

e. Initialiser un nouveau dépôt GIT local que nous allons « pusher » plus tard : `git init` ou bien via Eclipse :



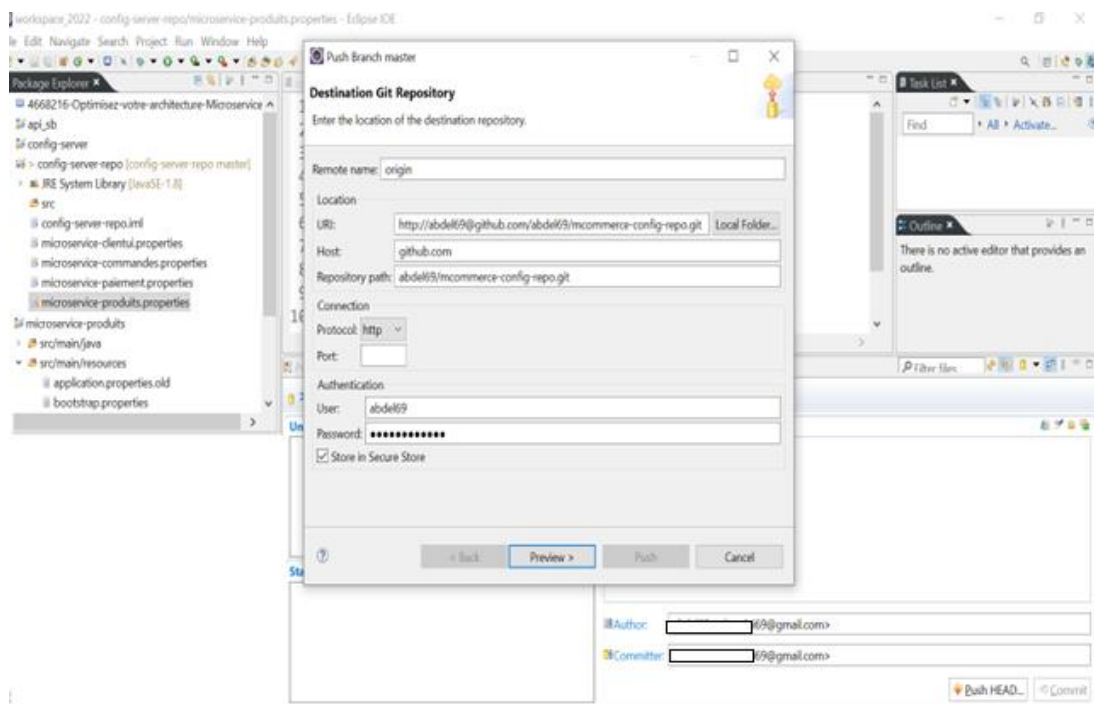
- f. Ajoutez les fichiers du dossier au GIT : **git add**

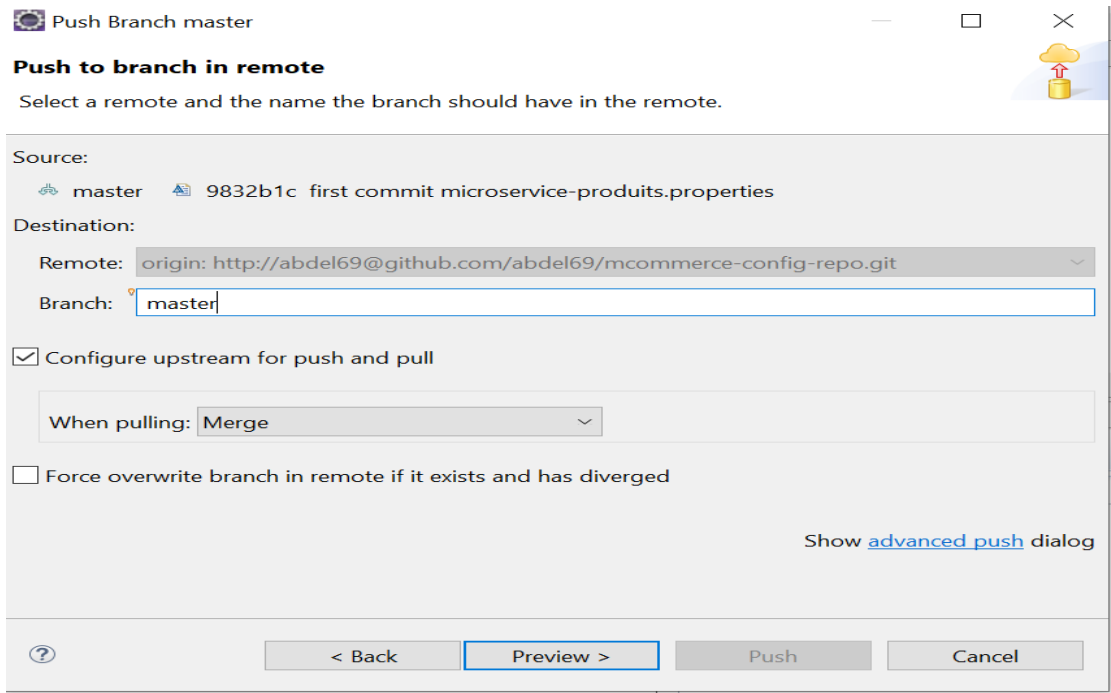


- g. **Git commit** puis **git push** du fichier

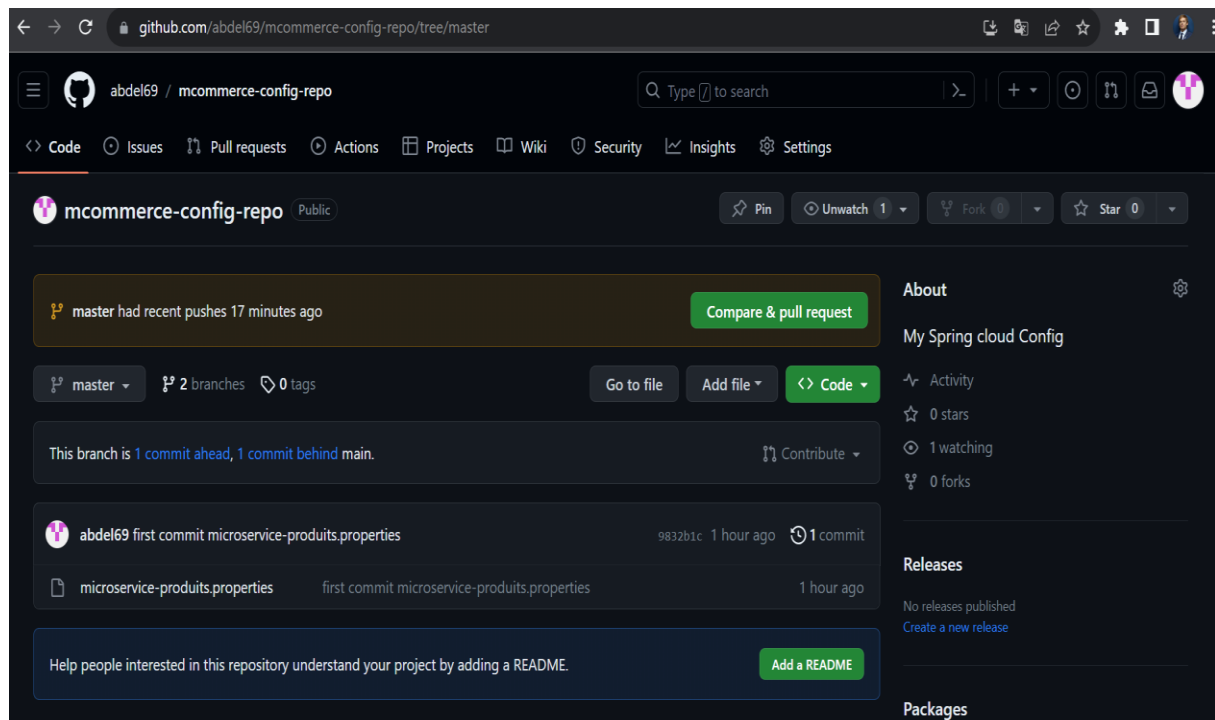
- h. Créer le repo Github «mcommerce-config-repo »

- 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>

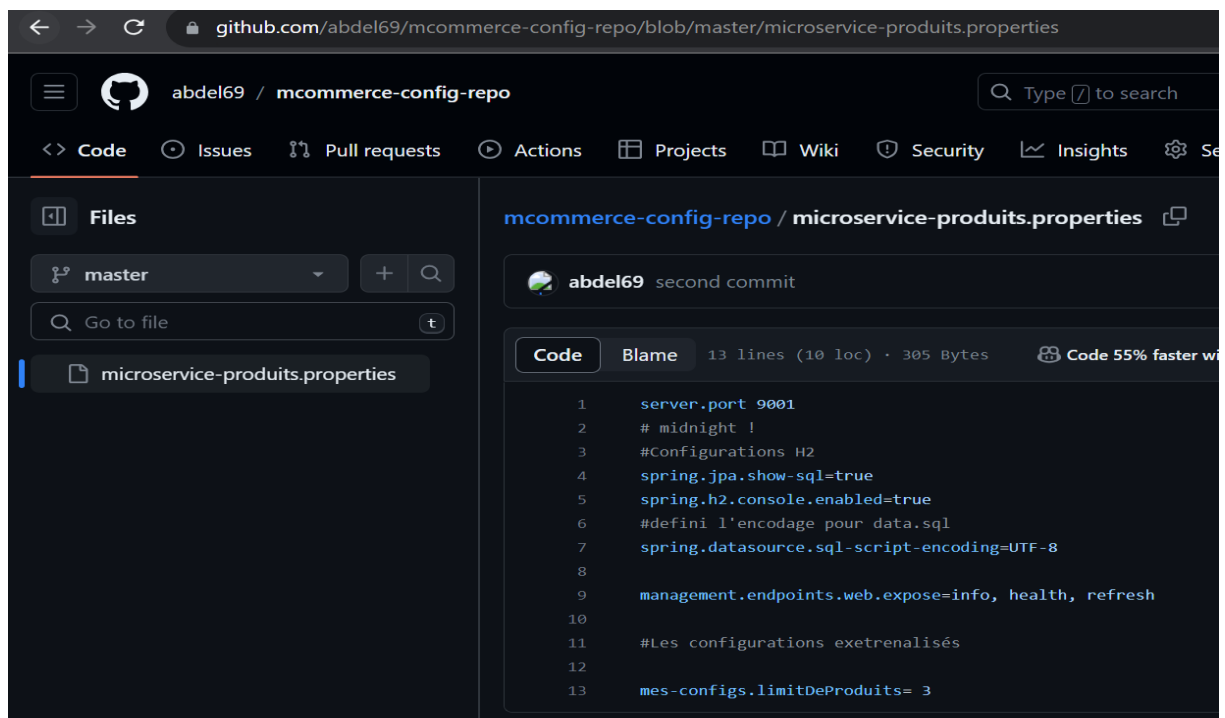




j. Vérifier que le push s'est bien déroulé :



Branch : **master**



## 7. Création de Spring Server Config

### a. Starter : SPRING CLOUD CONFIG

The screenshot shows the Spring Boot project configuration form. The 'Project' section includes 'Gradle - Groovy', 'Gradle - Kotlin', 'Java' (selected), 'Kotlin', 'Groovy', and 'Maven'. The 'Spring Boot' section shows version '2.7.16' selected. The 'Project Metadata' section includes fields for Group, Artifact, Name, Description, and Package name. The 'Dependencies' section shows 'Config Server' with 'SPRING CLOUD CONFIG' selected.

**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

☒ Maven

**Spring Boot**

☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (M3) ☐ 3.1.5 (SNAPSHOT) ☐ 3.1.4

☐ 3.0.12 (SNAPSHOT) ☐ 3.0.11 ☐ 2.7.17 (SNAPSHOT) ☒ 2.7.16

**Project Metadata**

Group

Artifact

Name

Description

Package name

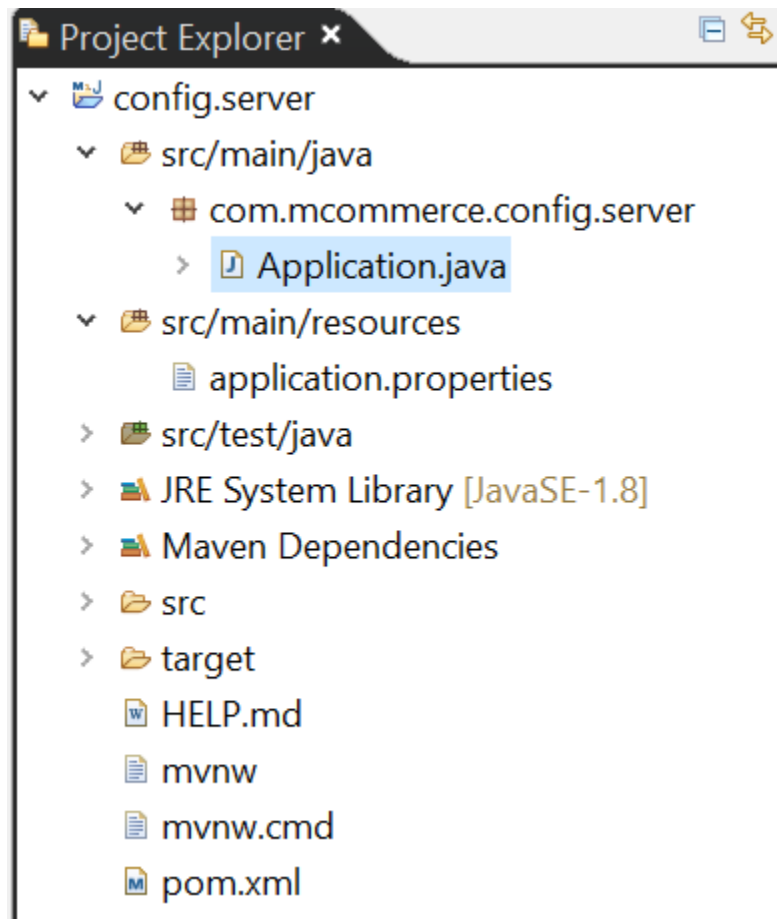
Packaging ☒ Jar ☐ War

Java ☐ 21 ☐ 17 ☐ 11 ☒ 8

**Dependencies**

**Config Server** ☒ **SPRING CLOUD CONFIG**

Central management for configuration via Git, SVN, or HashiCorp Vault.



```
<?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
    http://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

```

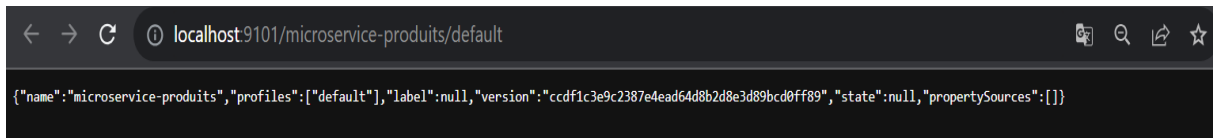
- Convention : Tous les Edge Microservices seront sur des ports commençant par 91.
  - Celui de *config-server* est 9101.
- b. Ajouter l'annotation `@EnableConfigServer` pour indiquer que ce microservice comme étant un serveur de configuration :

```

1 package com.mcommerce.config.server;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @EnableConfigServer
8 @SpringBootApplication
9 public class Application {
10
11     public static void main(String[] args) {
12         SpringApplication.run(Application.class, args);
13     }
14
15 }

```

c. <http://localhost:9101/microservice-produits/default>

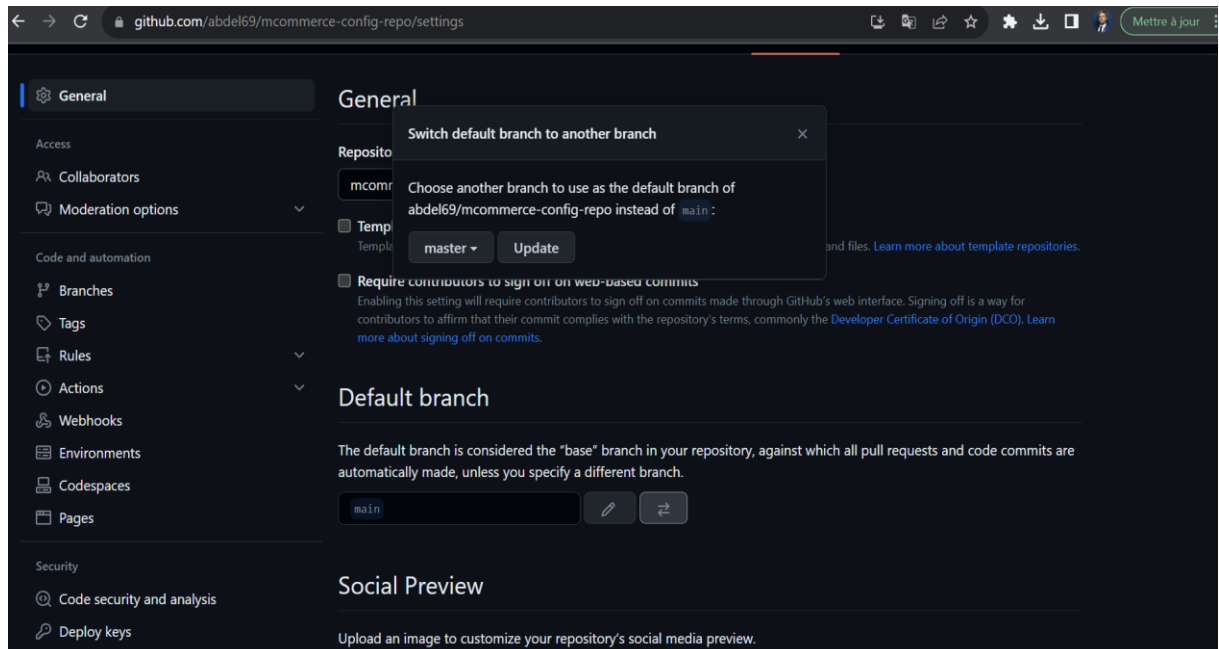


```

{"name":"microservice-produits","profiles":["default"],"label":null,"version":"ccdf1c3e9c2387e4ead64d8b2d8e3d89bcd0ff89","state":null,"propertySources":[]}

```

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



github.com/abdel69/mcommerce-config-repo/settings

General

Access

- Collaborators
- Moderation options

Code and automation

- Branches
- Tags
- Rules
- Actions
- Webhooks
- Environments
- Codespaces
- Pages

Security

- Code security and analysis
- Deploy keys

Repository

mcommerce-config-repo

Template repository

Require contributors to sign off on web-based commits

Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). [Learn more about signing off on commits.](#)

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

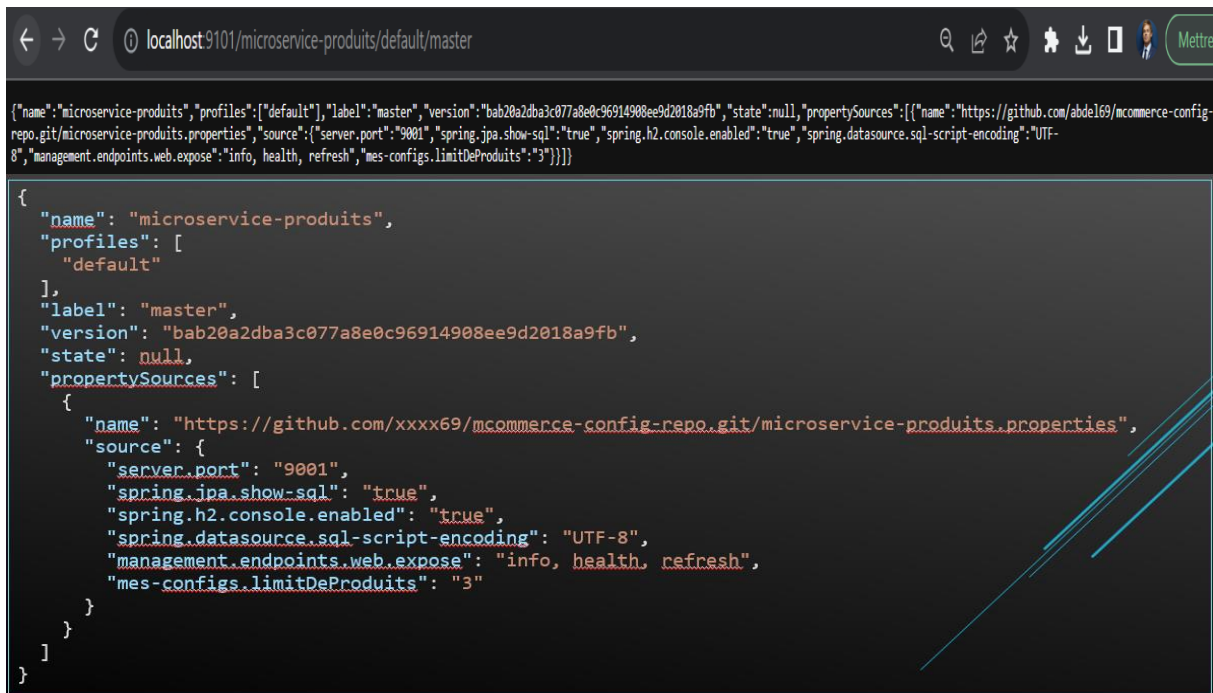
main

Social Preview

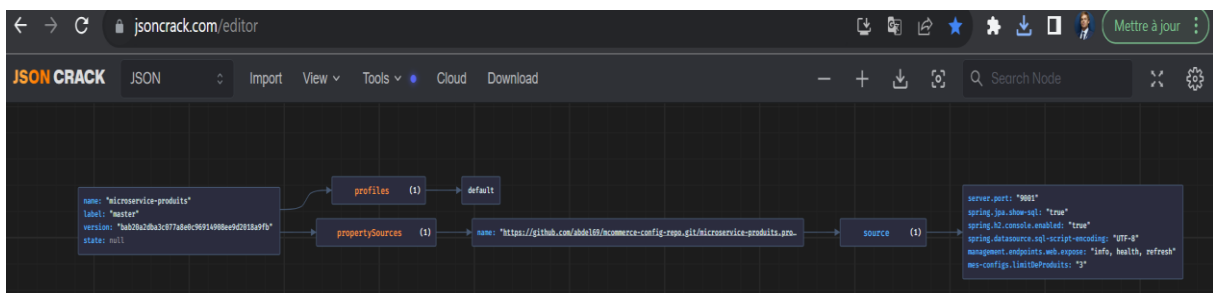
Upload an image to customize your repository's social media preview.



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



```
{
  "name": "microservice-produits",
  "profiles": [
    "default"
  ],
  "label": "master",
  "version": "bab20a2dba3c077a8e0c96914908ee9d2018a9fb",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/xxxx69/mcommerce-config-repo.git/microservice-produits.properties",
      "source": {
        "server.port": "9001",
        "spring.jpa.show-sql": "true",
        "spring.h2.console.enabled": "true",
        "spring.datasource.sql-script-encoding": "UTF-8",
        "management.endpoints.web.expose": "info, health, refresh",
        "mes-configs.limitDeProduits": "3"
      }
    }
  ]
}
```



f. Explication :

- 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-du-microservice/default/branche".
- 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:](#)

- Pour rappel, on a le dépôt Git relié à notre serveur Spring Cloud Config.
- 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.
- Appliquer la procédure de Renommage «[application.properties](#)» en «[bootstrap.properties](#)» décrite en haut
- Démarrage Microservice-produit :

➔ Remarquez : Fetching config from server at : <http://localhost:9101>

```
Fetching config from server at : http://localhost:9101L
Located environment: name=microservice-produits, BootstrapPropertySource {name='bootstrapProperties-
https://github.com/xxx69/mcommerce-config-repo.git/microservice-
produits.properties'}}com.mproduits.MproduitsApplication
Spring Data JPA repositories. Found 1 JPA repository interfaces.Tomcat initialized with port(s): 9001 (http)Starting service
[Tomcat]Starting Servlet engine: [Apache Tomcat/9.0.80]
Initializing Spring embedded WebApplicationContextH2 console available at '/h2-console'.
Database available at 'jdbc:h2:mem:9fc696fd-ad51-439a-9c56-4564786a80e6'
H2DialectHibernate: drop table if exists product CASCADE Hibernate: drop sequence if exists create sequence
hibernate_sequence start with 1 increment by 1
Hibernate: create table product (id integer not null, description varchar(255), image varchar(255), prix double, titre
varchar(255), primary key (id)).
Tomcat started on port(s): 9001 (http) with context path "/com.mproduits.MproduitsApplication" : Started
MproduitsApplication in 7.046 seconds (JVM running for 7.32)
```

The screenshot shows the H2 console interface in a web browser. The address bar displays the URL: `localhost:9001/h2-console/login.do?sessionId=bfc1d5b057d338b91bc283cc6ac3ba10`. The console shows the connection string: `jdbc:h2:mem:9fc696fd-ad51-439a-9c56-4564786a80e6`. The left sidebar lists the database schema: `PRODUCT`, `INFORMATION_SCHEMA`, `Sequences`, `Users`, and `H2 2.1.214 (2022-06-13)`. The main area shows the SQL statement: `SELECT * FROM PRODUCT`. The result is displayed as a table with 8 rows and 3 columns: `ID`, `DESCRIPTION`, and `IMAGE`. The data is as follows:

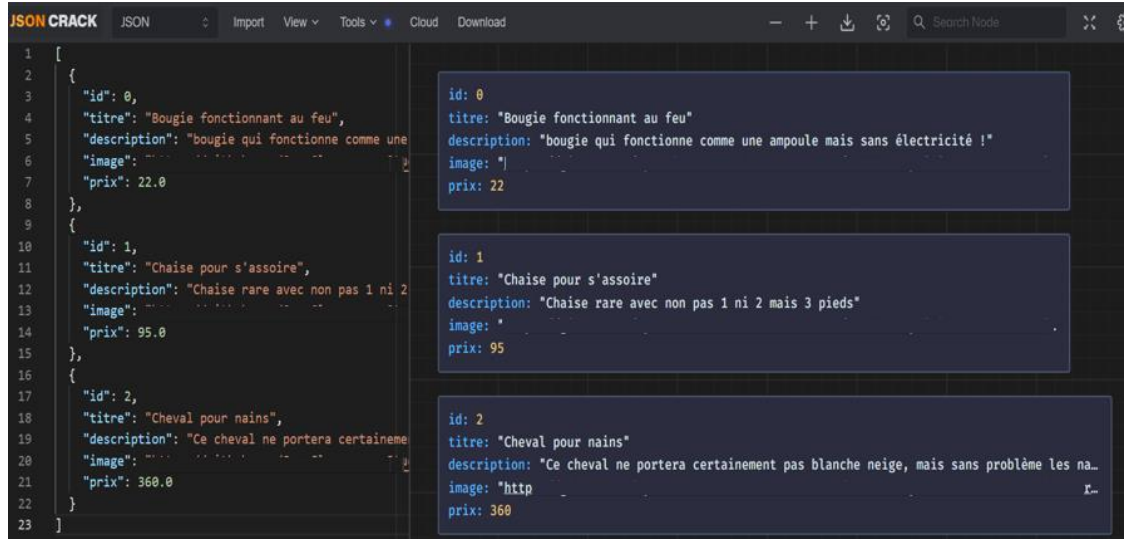
ID	DESCRIPTION	IMAGE
0	bougie qui fonctionne comme une ampoule mais sans électricité !	/blob/master/images/Bougie.PNG?raw=true
1	Chaise rare avec non pas 1 ni 2 mais 3 pieds	/blob/master/images/Chaise.PNG?raw=true
2	Ce cheval ne portera certainement pas blanche neige, mais sans problème les nains/blob/master/images/Cheval.PNG?raw=true	/blob/master/images/Cheval.PNG?raw=true
3	Ne passe pas au four	/blob/master/images/coq.PNG?raw=true
4	Vous donne droit à l'équivalent de 3/0 voeux	/blob/master/images/lampe.PNG?raw=true
5	Donne l'heure, les minutes et même les secondes. Ne fait pas de café	/blob/master/images/Horloge.PNG?raw=true
6	Pour réaliser vos opérations chirurgicales sur votre Hamster!	/blob/master/images/table%20d'op%C3%A9ration.PNG?
7	Risque de choc électrique	/blob/master/images/Vase.PNG?raw=true

(8 rows, 0 ms)

An `Edit` button is located at the bottom left of the result area.

e. <http://localhost:9001/Produits> :

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



```
server.port 9001
# midnight commit!
#change git branch from master To main : didn't work #

#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

management.endpoints.web.expose=info, health, refresh

#Les configurations exetrenalisés
mes-configs.limitDeProduits= 3
```