Guide d'utilisation des Microservices Commande et Produits

Groupe : 5IIR 4

Binôme: *AHESBI Badr – BENOMAR Souhail*

1. Aperçu

Ce projet implémente une architecture basée sur des microservices pour la gestion des commandes et des produits. Chaque microservice est autonome et communique via des appels HTTP grâce à **Spring Cloud OpenFeign**.

2. Microservices utilisés

2.1. Microservice Commandes

- **Description :** Gère la création, la récupération, la mise à jour et la suppression des commandes.
- **Port :** 9001
- **Endpoints principaux :**
  - GET /commandes : Récupère toutes les commandes.
  - POST /commandes : Crée une nouvelle commande.
  - GET /commandes/{id} : Récupère une commande spécifique.
  - GET /commandes/test-resilience/{id} : Test de la résilience face aux pannes.

2.2. Microservice Produits

- **Description :** Fournit les informations sur les produits et simule des délais pour tester la résilience.
- **Port :** 9002
- **Endpoints principaux :**
  - GET /produits : Récupère la liste des produits.
  - GET /produits/{id} : Vérifie l'existence d'un produit.

2.3. Microservice Config Server

- **Description :** Fournit la configuration centralisée des microservices via un dépôt Git.
- **Configuration Git:**
  - Le Config Server récupère les fichiers de configuration depuis un dépôt Git distant.

- spring.cloud.config.server.git.uri=***https://github.com/Zuriif/microservice-config.git***

2.4. Microservice API Gateway

- **Description :** Fournit un point d'entrée unique pour les clients et permet le routage des requêtes vers les microservices appropriés.
- **Port :** 8080
- **Endpoints principaux :**
  - GET /8080/commandes/** : Redirige vers le microservice commandes.
  - GET /8080/produits/** : Redirige vers le microservice produits.

2.5. Eureka Server

- **Description :** Service de découverte permettant aux microservices de s'enregistrer et de découvrir dynamiquement d'autres services.
- **Port :** 8761
- **Endpoints principaux :**
  - http://localhost:8761/ : Interface web pour visualiser les services enregistrés.

3. Fonctionnalités de résilience

Nous utilisons **Resilience4j** pour améliorer la tolérance aux pannes des microservices.

- **TimeLimiter :** Interrompt les appels dépassant une durée maximale (configurée à 2s).
- **CircuitBreaker :** Coupe les appels après un nombre d'échecs consécutifs.
- **Retry :** Réessaie les appels en cas d'échec temporaire.

Configuration :

```
@Configuration
public class ResilienceConfig { @Bean
  public TimeLimiter produitServiceTimeLimiter() { return
    TimeLimiter.of(TimeLimiterConfig.custom()
      .timeoutDuration(Duration.ofSeconds(2))
      .build());
  }
}
```

4. Instructions pour l'exécution

1. ***Démarrer le microservice config server :***
2. ***Démarrer le microservice Eureka Server :***
3. ***Démarrer le microservice API Gateway :***
4. ***Démarrer le microservice produits :***
5. ***Démarrer le microservice commandes :***
6. ***Tester les endpoints via Postman***
    - Vérification du produit : GET http://localhost:9002/produits/1
    - Test de résilience : GET http://localhost:8080/commandes/test-resilience/1

5. Monitoring avec Actuator

L'état des microservices peut être surveillé via Actuator :

- http://localhost:9002/actuator/health : Vérification de l'état du microservice produits.
- http://localhost:9001/actuator/refresh : utilisée pour **recharger dynamiquement la configuration** (à chaud) du microservice produits sans nécessiter de redémarrage. Cette commande est particulièrement utile dans le cadre de l'utilisation de **Spring Cloud Config**, où les paramètres de configuration sont centralisés et peuvent être modifiés à la volée.