Introduction à Spring Cloud

1 Introduction

Le projet Spring Cloud [1] regroupe de nombreux sous-projets destinés à faciliter l'utilisation et le déploiement d'applications Java/Spring dans des environnements Clouds. On trouve notamment

- **Spring Cloud Gateway** : construction d'une API-Gateway (passerelle) vers des microservices.
- **Spring Cloud Config** : mise en place d'un système réparti de configuration des microservices.
- **Spring Cloud Function** : facilite la mise en place d'opération métiers en tant que fonction.
- ... et beaucoup d'autres projets.

Nous allons simplement effleurer le sujet en étudiant la partie passerelle vers des microservices ainsi que le serveur de configuration.

2 Spring Cloud Gateway

2.1 Débutons

⏩ **Travail à faire :**

- Commencez par suivre cette introduction [a] en partant de zéro (via **Spring Initializr**). N'oubliez pas d'ajouter la dépendance vers `spring-boot-devtools` afin que votre application redémarre automatiquement.
- Téléchargez ces deux APIs [b], lancez-les et testez-les (`http://localhost:9080/swagger-ui/index.html` et `http://localhost:9090/swagger-ui/index.html`).
- Ajoutez à votre application **Spring Cloud Gateway** les routes qui permettent d'atteindre ces deux APIs.

a. https://spring.io/guides/gs/gateway
b. apis.zip

2.2 Les prédicats

Les prédicats permettent de sélectionner la requête qui va être routée. Il existe donc plusieurs prédicats qui correspondent au chemin, au serveur, aux paramètres, aux méthodes HTTP, etc. La méthode `path()` déjà vue est un prédicat.

1. https://spring.io/projects/spring-cloud

Travail à faire :

- Ouvrez dans un onglet cette documentation [a].
- Avec les prédicats `path("...").and().query(name,value)` créez une route pour `/api/hello/john?majuscule=oui` afin d'obtenir `Hello JOHN`. Vous devrez vous servir du filtre `setPath` afin de changer le chemin.
- Avec le prédicat `host(hostName)` renvoyez les routes `http://bizarre.fr/**` vers `http://httpbin.org:80/get` (utilisez `setPath`). Vous devrez utiliser la ligne ci-dessous pour transmettre le nom de l'hôte utilisé :

```
curl --header 'Host: bizarre.fr' http://localhost:8080/api/hello
```

- Avec le prédicat `method(methodName)`, renvoyez les requêtes `DELETE` vers `http://httpbin.org/delete`.
- Avec le prédicat `before(time)`, renvoyez les requêtes `/time` vers `http://httpbin.org/image/png` si la date est inférieure à cinq minutes après le démarrage du serveur (utilisez la mathode static `new()` de la classe `ZonedDateTime` ainsi que `plusMinutes(minutes)`).
- Avec le prédicat `header(headerName)`, traitez un cas particulier (à choisir).
- Si la forme "Fluent API" vous gêne, vous pouvez toujours définir des méthodes pour vos prédicats :

Définir mon prédicat comme une méthode

```
BooleanSpec myPredicate(PredicateSpec predicate) {
    return predicate.path("/myPath");
}

// utilisation:
.route(p -> myPredicate(p)//
    .uri("http://httpbin.org:80"))
```

a. https://docs.spring.io/spring-cloud-gateway/reference/spring-cloud-gateway-server-webflux/request-predicates-factories.html

2.3 Les filtres

Les filtres permettent de modifier la requête qui va être routée. Nous avons déjà utilisé `setPath()` qui redéfinit le chemin ainsi que `addRequestHeader` qui ajoute un *header*.

Travail à faire :

- Ouvrez dans un onglet cette documentation [a].
- Avec le filtre `rewritePath` faites en sorte que les routes `/api/films**` donnent le même résultat que `api/movies`.
- Avec le filtre `addRequestParameter` faites en sorte que `/param` renvoie vers `http://httpbin.org:80/get?myParam=myValue` (il faut combiner deux filtres).
- Nous pouvons également modifier les données avant ou après routage. Utilisez le filtre `modifyRequestBody` sur la route `/modif` renvoyée vers `http://httpbin.org:80/post` pour passer en majuscule les données de la requête (vous aurez besoin de la méthode ci-dessous).

```
record Message(String message) { }

Mono<Message> toUpper(ServerWebExchange request, String
    body) {
   return Mono.just(new Message(body.toUpperCase()));
}
```

Exemple à tester

```
curl -s -d 'hello' "http://localhost:8080/modif"
```

- Nous allons modifier les données après routage (utile pour supprimer par exemple des informations sensibles). Utilisez le filtre `modifyResponseBody` sur la route `/myFilm` renvoyée vers le film n°1 mais dans lequel la description sera supprimée (vous aurez besoin du `record` ci-dessous).

```
record MovieWithoutDescription(int id, String name, int
    year) { }
```

- Si la forme "Fluent API" vous gêne, vous pouvez toujours définir des méthodes pour vos filtres :

Définir mon filtre comme une méthode

```
UriSpec myFilter(GatewayFilterSpec filter) {
   return filter.metadata("myMeteData", "myMetaDataValue")
       ;
}

// utilisation:
.route(p -> myPredicate(p)//
    .filters(this::myFilter)//
    .uri("http://httpbin.org:80"))
```

a. https://docs.spring.io/spring-cloud-gateway/reference/spring-cloud-gateway-server-webflux/gatewayfilter-factories.html

2.4 Limitation du débit

⏩ Travail à faire :

- Pour étudier ce mécanisme, nous aurons besoin d'une instance de **Redis** fonctionnelle. Vous devez donc (re-)faire ces manipulations [a] afin d'activer **Redis** dans un conteneur docker.
- N'oubliez pas de récupérer la configuration de Spring boot ci-dessous :

```
# config du serveur Redis
spring.data.redis.host=<adresse IP de redis-vm>
spring.data.redis.port=6379
spring.data.redis.password=mypass
spring.data.redis.database=0
spring.data.redis.timeout=60000
```

- Vous pouvez maintenant suivre ce tutoriel [b] en enrichissant votre projet.

a. tp-reactive.html#install-redis
b. https://www.baeldung.com/spring-cloud-gateway-rate-limit-by-client-ip

3 Spring Cloud Config

Dans des environnements fortement répartis, comme les microservices, les informations de configuration (par exemple la référence à la base de données) sont disséminées dans chaque application (fichier `application.properties`). Le projet **Spring Cloud Config** [2] se propose de les centraliser et de les distribuer aux différentes applications. Nous allons donc mettre en place un serveur de configuration, et les applications classiques deviennent des clients de configuration.

⏩ Travail à faire :

- Vous allez suivre un petit tutoriel pour découvrir cette architecture, mais notez les trois points suivants :
  - ▷ N'hésitez pas à utiliser Spring Initializr [a] pour créer vos projets,
  - ▷ Utilisez l'etulab de l'université pour stocker vos fichiers de configuration (voir le sujet),
  - ▷ Vous pouvez sauter la partie chiffrement.
- Ok, allons-y avec ce tutoriel [b].

a. https://start.spring.io/
b. https://www.baeldung.com/spring-cloud-configuration

4 Pour aller plus loin sur Spring Cloud Gateway

Vous pouvez traiter les premières sections du tutoriel sur les filtres [3].

2. https://spring.io/projects/spring-cloud-config
3. https://www.baeldung.com/spring-cloud-custom-gateway-filters