TP n°4 en Systèmes distribues

Objectifs : Mise en œuvre de l'architecture 3-tiers (Technologies Web Services) Microservices-SpringBoot-SpringCloud-JPA-REST-Eureka

L'architecture microservices représente un moyen de concevoir des applications comme ensemble de services indépendamment déployables. Ces services doivent de préférence être organisés autour des compétences métier, de déploiement automatique, d'extrémités intelligentes et de contrôle décentralisé des technologies et des données.
Spring Cloud fournit des outils pour les développeurs pour construire rapidement et facilement des patrons communs de systèmes répartis (tel que des services de configuration, de découverte ou de routage intelligent). Spring Boot permet de son côté de construire des applications Spring rapidement aussi rapidement que possible, en minimisant au maximum le temps de configuration, d'habitude pénible, des applications Spring.

Dans une architecture microservices, plusieurs services s'exécutent en même temps, sur des processus différents, avec chacun sa propre configuration et ses propres paramètres. Spring Cloud Config fournit un support côté serveur et côté client pour externaliser les configurations dans un système distribué. Grâce au service de configuration, il est possible d'avoir un endroit centralisé pour gérer les propriétés de chacun de ces services.

Eureka, registre de services au cœur de l'architecture:

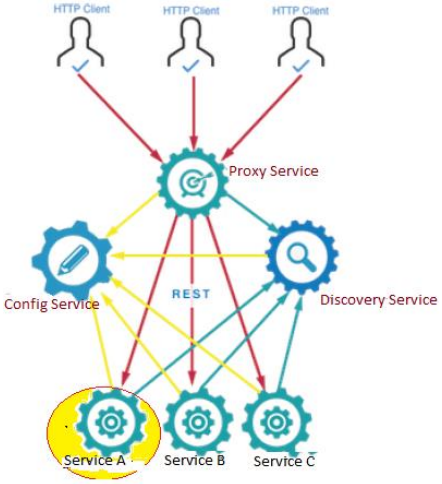
- Eureka est une application permettant la localisation d'instances de services.
- Elle se caractérise par une partie serveur et une partie cliente.
- La communication entre les parties se fait via les API Web exposées par le composant serveur.
- Ces services doivent être créés en tant que clients Eureka, ayant pour objectif de se connecter et s'enregistrer sur un serveur Eureka.
- De ce fait, les clients vont pouvoir s'enregistrer auprès du serveur et périodiquement donner des signes de vie.
- Le service Eureka (composant serveur) va pouvoir conserver les informations de localisation desdits clients afin de les mettre à disposition des autres services (service registry).

Exercice 1 : L'objectif de ce TP est de montrer comment créer plusieurs services indépendamment déployables qui communiquent entre eux, en utilisant les facilités offertes par Spring Cloud et Spring Boot.

Nous allons donc créer les quatre microservices ci-dessous :

- departement-Service (Service A) : Service principal, qui offre une API REST pour lister une liste des departements.
- *employe-Service* (Service B) : Service principal, qui offre une API REST pour gerer les employes des departements.
- *Config Service* : Service de configuration, dont le rôle est de centraliser les fichiers de configuration des différents microservices dans un endroit unique.
- *proxy Service* : Passerelle se chargeant du routage d'une requête vers l'une des instances d'un service, de manière à gérer automatiquement la distribution de charge.
- *Discovery Service*: Service permettant l'enregistrement des instances de services en vue d'être découvertes par d'autres services.

Architecture de microservices



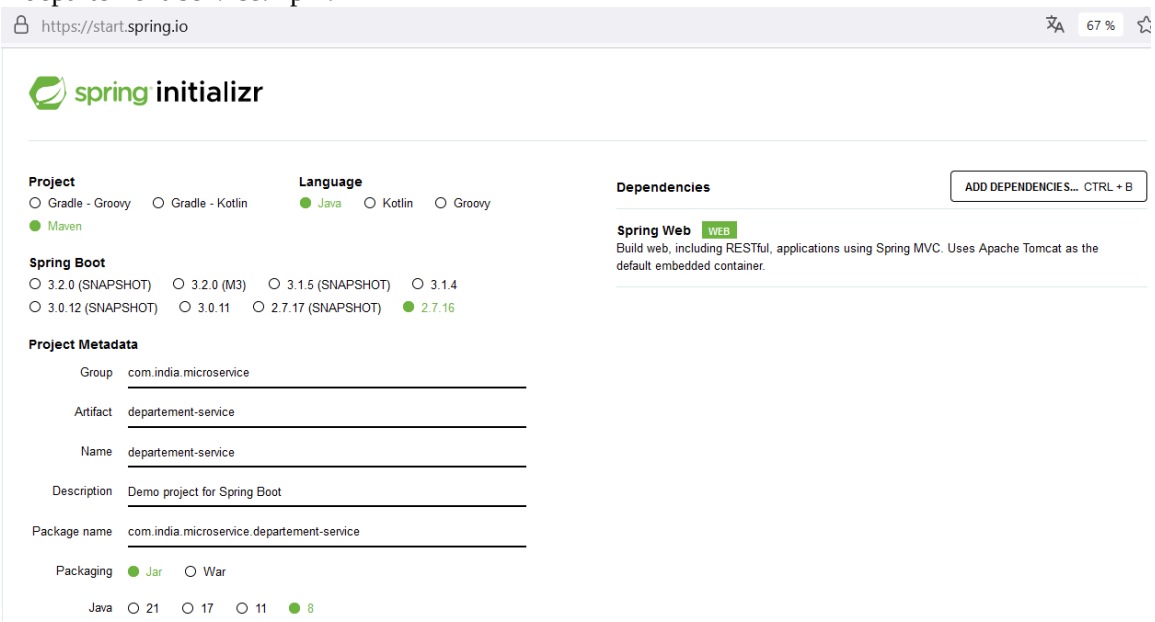
Imaginons un scénario simple de boutique en ligne, dans lequel les clients peuvent passer des commandes. Nous pouvons déjà identifier certains services dont nous aurons besoin - un service client et un service de commande.
Nous allons procéder étape par étape pour construire chacun d'eux.

Remarque : pour les besoins de cette démo, nous nous en tiendrons aux projets Java utilisant Maven et Spring Boot 2.2.0. Nous utiliserons également le groupe / package com.india.microservice pour tous les projets.

1. Le Service Client

Rendons-nous **sur start.spring.io pour** créer notre projet de service client. C'est généralement le meilleur point de départ pour toute nouvelle application Spring Boot.

Créer un projet avec :
Group : **com.india.microservice**
Artefact : **departement-service**
Name : **departement-service**,
en spécifiant uniquement ***Spring Web*** *comme dépendance*, puis appuyer sur Générer pour créer le fichier « departement-service.zip ».



Une fois téléchargé et extrait, nous créerons une classe Departement à utiliser comme objet de domaine:

```
package com.india.microservice.departementservice;
public class Departement {
   private final int id;
   private final String nom;

   public Departement(final int id,  final String nom) {
      this.id = id;
      this.nom = nom;
   }
   public int getId() {
      return id;
   }
   public String getNom() {
      return nom;
   }
}
```

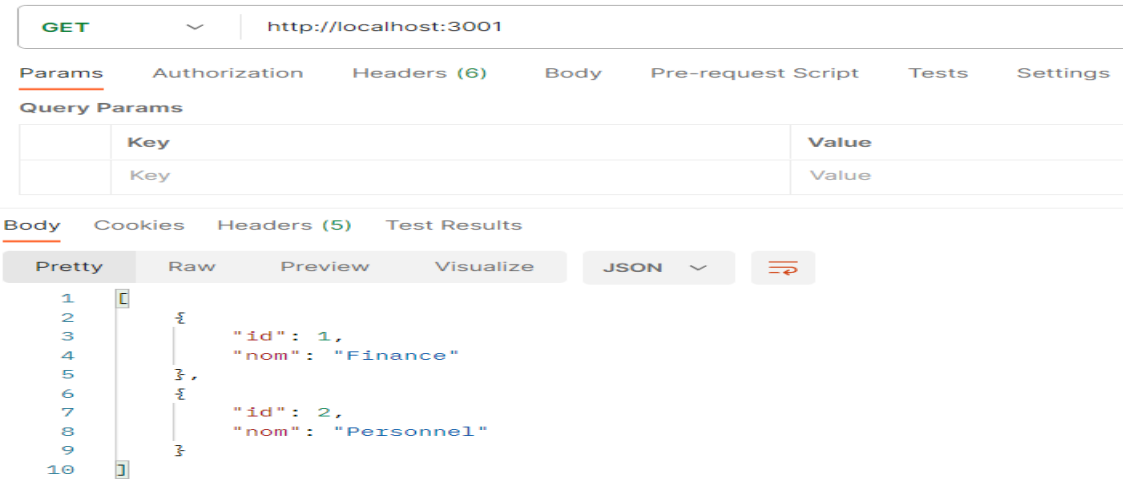
Ensuite, nous ajouterons une classe DepartementController qui expose quelques points de terminaison pour nous permettre d'afficher tous les departements :

```
package com.india.microservice.departementservice;

import java.util.Arrays ;
import java.util.List;
import org.springframework.web.bind.annotation.GetMapping ;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class DepartementController {
   private List<Departement> departements = Arrays.asList(
         new Departement(1, "Finance"),
         new Departement(2, "Personnel"));

   @GetMapping
   public List<Departement> getAllDepartements() {
      return departements;
   }
   @GetMapping("/{id}")
   public Departement getDepartementById(@PathVariable int id) {
      return departements.stream()
                  .filter(Departement -> Departement.getId() == id)
                  .findFirst()
                  .orElseThrow(IllegalArgumentException::new);
   }
}
```

Les données sont représentées ici comme une propriété dans la classe du contrôleur.
Enfin, configurons le service departement pour lui donner le nom **departement-service** et un port par défaut **3001**. Ajouter ce qui suit au fichier src / main / resources / application.properties:

```
spring.application.name=departement-service
server.port=3001
```

Vérifions que tout fonctionne en créant et en exécutant le projet à l'aide de l'application **Run As / Spring Boot**:
Visiter http://localhost:3001 devrait afficher une liste des departements représentés dans JSON.
Ouvrir http://localhost:3001/1 pour voir « Finance » ou http://localhost:3001/2 pour voir « Personnel » en tant qu'éléments de données individuels.



2. Le service employe

La configuration du service employe est pratiquement identique à celle du service departement. Créer un projet avec Artifact="**employe-service**" en utilisant start.spring.io et les mêmes dépendances qu'auparavant. Ensuite, nous pouvons ajouter un objet de domaine Employe:

```
package com.india.microservice.employeservice;

public class Employe {
    private final int id;
    private final int departementId;
    private final String nom;

    public Employe(final int id, final int departementId, final String nom) {
        this.id = id;
        this.departementId = departementId;
        this.nom = nom;
    }

    public int getId() {
        return id;
    }
    public int getDepartementId() {
        return departementId;
    }
    public String getNom() {
        return nom;
    }
}
```

Créer ensuite la classe **EmployeController** pour gérer les employes :

```
package com.india.github.microservice.employeservice;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.util.Arrays;
import java.util.List;

@RestController
public class EmployeController {
    private final List<Employe> employes = Arrays.asList(
            new Employe(1, 1, "Mark"),
            new Employe(2, 1, "Alex"),
            new Employe(3, 2, "James"),
            new Employe(4, 1, "King"),
            new Employe(5, 2, "Alex"));

    @GetMapping
    public List<Employe> getAllEmployes() {
        return employes;
    }

    @GetMapping("/{id}")
    public Employe getEmployeById(@PathVariable int id) {
        return employes.stream()
                    .filter(Employe -> Employe.getId() == id)
                    .findFirst()
                    .orElseThrow(IllegalArgumentException::new);
    }
}
```

Et mettre à jour le fichier application.properties:

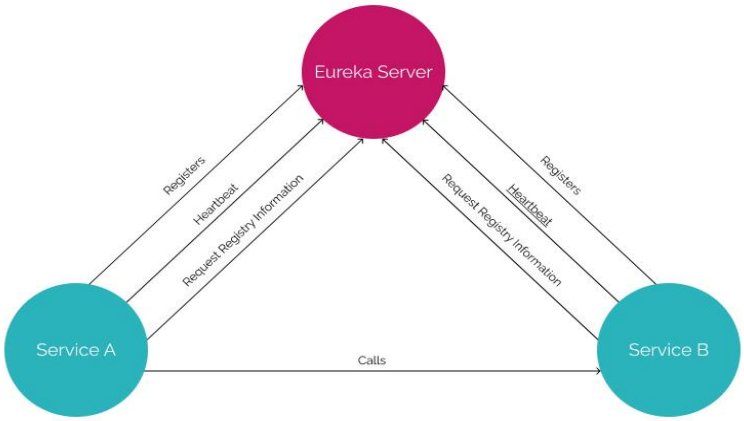
```
spring.application.name=employe-service
server.port=3002
```

Enfin, exécutons le service employe Run As / Spring Boot App) et vérifions que tout fonctionne.

3. Le service de découverte (discovery-service)

Une architecture de microservice peut être incroyablement dynamique. Les services n'ont pas nécessairement d'adresses fixes, connues à l'avance. Ils peuvent être déplacés sur différents ports, machines et même différents centres de données entièrement. Le plus souvent, il y aura de nombreuses instances d'un service donné - un nombre qui est rarement constant car de nouvelles instances sont souvent introduites pour répondre à la demande et sont supprimées lorsque la demande diminue. Ils ont également besoin de découvrir d'autres services dans employe pour communiquer avec eux.

Eureka de Netflix est un outil de découverte de services, conçu pour résoudre ce problème. Lorsqu'un service démarre, il s'enregistre auprès d'Eureka, en précisant son nom, son adresse et d'autres informations pertinentes. Il envoie régulièrement des messages de pulsation à Eureka pour lui indiquer qu'il est toujours en vie et capable de traiter les demandes. Si ce service s'arrête pour une raison quelconque, Eureka le désenregistrera après un délai d'expiration configuré. Les services peuvent également demander des informations de registre à Eureka afin de découvrir d'autres services.



Service decouverte

Créer un nouveau projet à l'aide de start.spring.io avec un artefact="**discovery-service**". Sélectionner **Eureka Server** comme sa seule dépendance et appuyer sur Générer. Ouvrir la classe DiscoveryServiceApplication et ajouter l'annotation @EnableEurekaServer, pour mettre en place un registre de services Eureka:

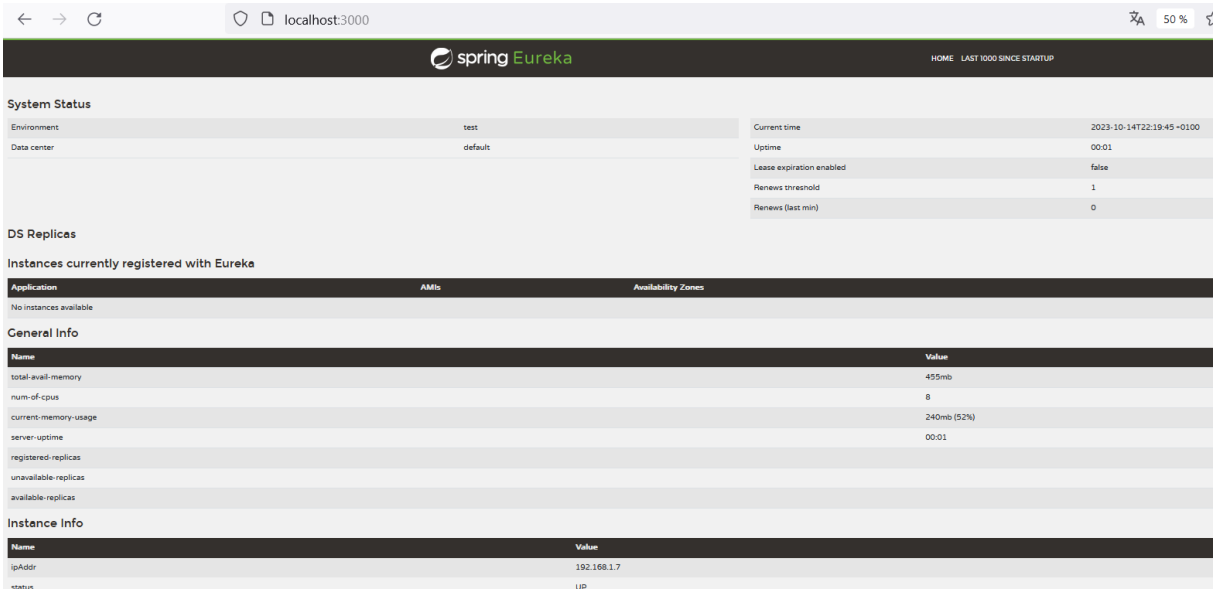
```
package com.india.microservice.discoveryservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServiceApplication.class, args);
```

```
    }
}
```

Par défaut, un serveur Eureka communique avec des pairs pour partager leurs informations de registre afin de fournir une haute disponibilité, mais comme nous allons simplement exécuter une seule instance ici, désactivons cette fonctionnalité et configurons notre service pour ne pas s'enregistrer avec un pair et ne pas chercher un registre de pairs. Nous lui donnerons également un nom et un port par défaut de **3000** (la valeur par défaut pour Eureka est 8761). Dans application.properties, ajouter ce qui suit:

```
spring.application.name=discovery-service
server.port=3000
eureka.client.registerWithEureka=false
eureka.client.fetchRegistry=false
eureka.instance.hostname=localhost
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/
```

Construire et exécuter le service (Exécuter en tant que / Spring Boot App) et confirmer qu'il fonctionne en visitant http://localhost:3000. Vous devriez voir un tableau de bord Eureka qui affiche des informations sur l'instance en cours d'exécution :



Inscription au service de découverte

Maintenant que notre service de découverte est opérationnel, nos services de domaine doivent communiquer avec lui pour s'enregistrer et recevoir les mises à jour du registre. Pour ce faire, nous devons ajouter la dépendance **Eureka Discovery Client** à nos projets

```
<dependency>
	<groupId>org.springframework.cloud</groupId>
	<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
	<version>3.1.4</version>
</dependency>
```

Maintenant, annotons nos classes DepartementServiceApplication et EmployeServiceApplication avec l'annotation @EnableEurekaClient:

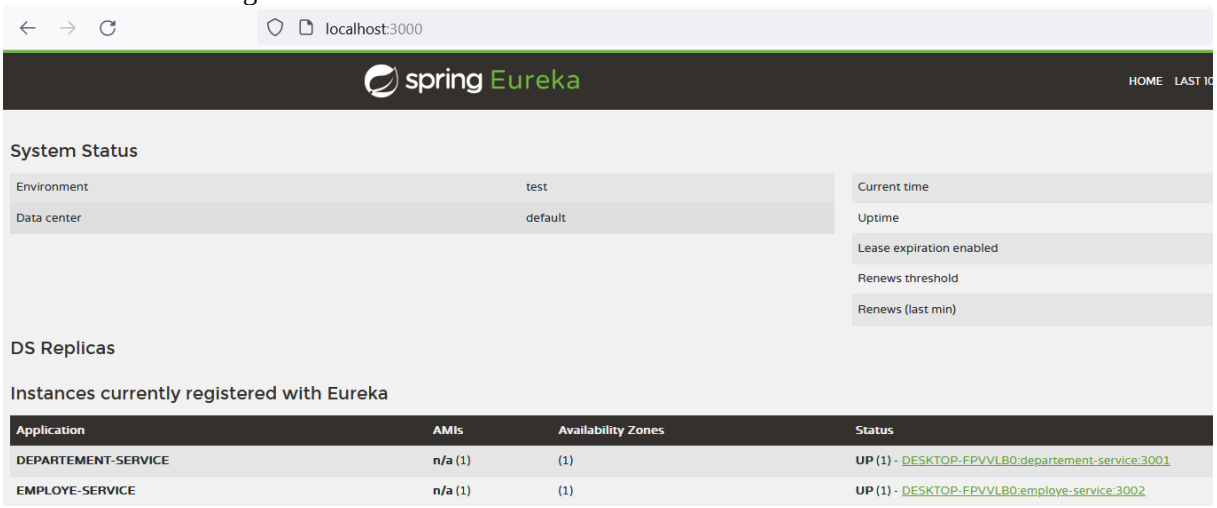
```
package com.india.microservice.departementservice;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class DepartementServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(DepartementServiceApplication.class, args);
   }
}
```

Enfin, indiquons au client Eureka où trouver notre service de découverte. Dans application.properties de chaque service, ajouter la ligne suivante:

```
eureka.client.serviceUrl.defaultZone=http://localhost:3000/eureka/
```

Lancer votre service de découverte, suivi des applications de service departement et de service de employe, puis ouvrir le tableau de bord Eureka du service de découverte - vous devriez voir que les deux services ont été enregistrés.

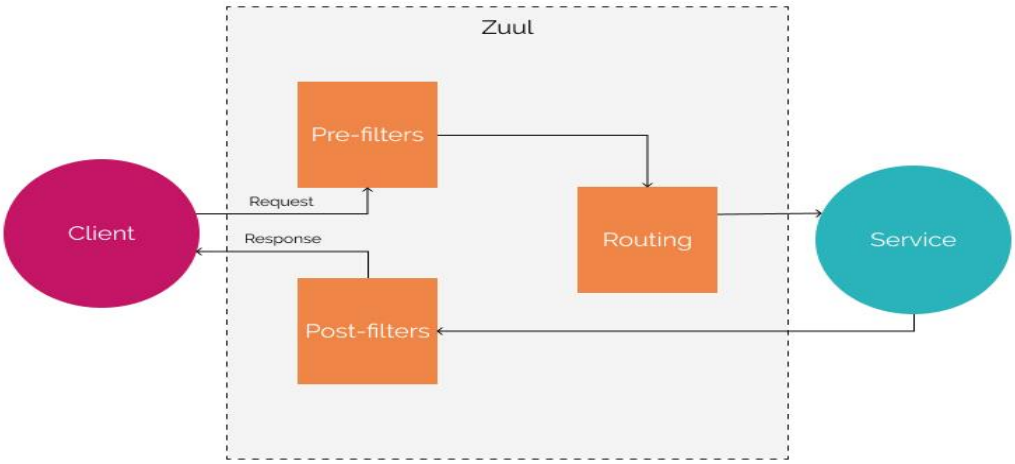


4. Routage et équilibrage de charge côté serveur avec Zuul

Dans une architecture de microservices, il peut y avoir des dizaines, des centaines voire des milliers de services. Beaucoup sont privés et internes, mais certains doivent être exposés au monde extérieur. Nous avons besoin d'un point d'entrée unique dans le système pour nous permettre de câbler et d'exposer certains services au monde extérieur.

Le Zuul de Netflix est un routeur basé sur JVM et un équilibreur de charge côté serveur. En mappant les itinéraires vers les services via sa configuration, Zuul peut s'intégrer à Eureka pour découvrir les emplacements de service afin d'équilibrer la charge et les requêtes proxy.

Zuul prend également en charge les filtres qui permettent aux développeurs d'intercepter les demandes avant qu'elles ne soient envoyées aux services (pré-filtres) et les réponses avant d'être renvoyées aux departements (post-filtres). Cela permet aux développeurs d'implémenter des fonctionnalités communes à tous les services, exécutées avant ou après le traitement des demandes.



Service de passerelle (Gateway Service**)**

De retour à start.spring.io, créer un nouveau projet avec Artifact="**gateway-service**", et Zuul et Eureka Discovery Client comme dépendances. Une fois générée, ouvrir la classe GatewayServiceApplication et ajouter les annotations @EnableZuulProxy et @EnableEurekaClient.

```
package com.india.microservice.gatewayservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
@SpringBootApplication
@EnableZuulProxy
@EnableEurekaClient
public class GatewayServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(GatewayServiceApplication.class, args);
   }
}
```

```
<?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>2.2.2.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.example</groupId>
        <artifactId>gateway-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>gateway-service</name>
        <description>Demo project for Spring Boot</description>

        <properties>
                <java.version>1.8</java.version>
                <spring-cloud.version>Hoxton.SR1</spring-cloud.version>
        </properties>
```

```
	<dependencies>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-zuul</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
			<exclusions>
				<exclusion>
					<groupId>org.junit.vintage</groupId>
					<artifactId>junit-vintage-engine</artifactId>
				</exclusion>
			</exclusions>
		</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>
```

Ajouter maintenant ce qui suit à application.properties:

```
spring.application.name=gateway-service
server.port=8089
eureka.client.serviceUrl.defaultZone=http://localhost:3000/eureka/
```

```
zuul.routes.departements.path=/departements/**
zuul.routes.departements.serviceId=departement-service

zuul.routes.employes.path=/employes/**
zuul.routes.employes.serviceId=employe-service
```

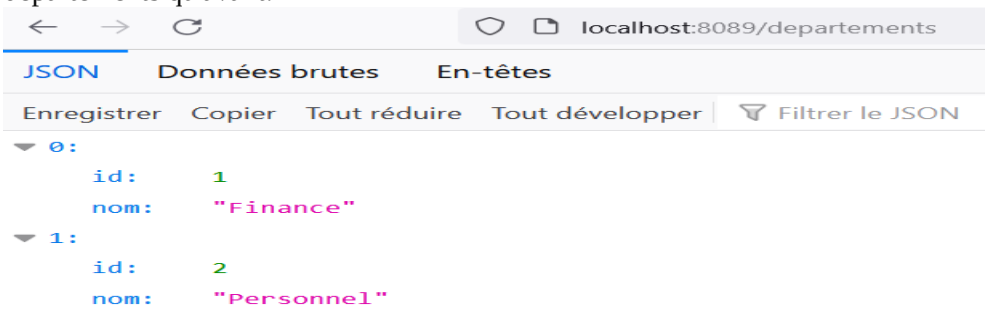
Nous avons donné au projet un nom de service de passerelle, l'avons configuré pour qu'il s'exécute sur le port 8089 (qui est la valeur par défaut pour HTTP) et lui avons dit de trouver notre service de découverte sur le port 3000.

Nous avons également ajouté des mappages d'itinéraire afin que les demandes adressées à / departements / ** et / employes / ** seront transmises aux services nommés respectivement service departement et service employe.

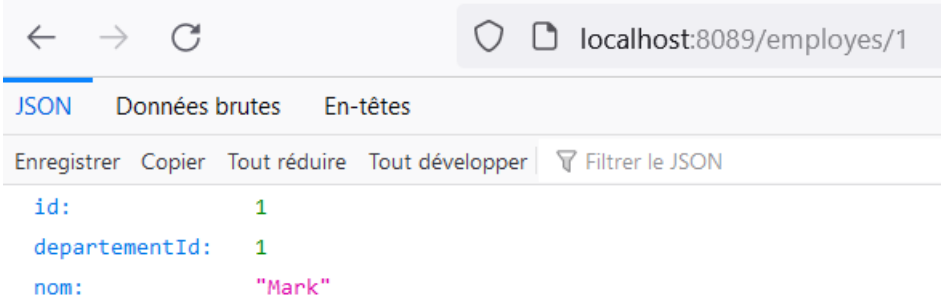
```
zuul.routes.your-service-name.path=/votre-service-path/**
zuul.routes.your-service-name.serviceId=votre-nom-de-service
```

Zuul utilise le registre de service fourni par notre service de découverte pour localiser chaque service cible.
Une fois que tout est en cours d'exécution, attender quelques minutes que le service de découverte reçoive les connexions de chaque service et propage son registre vers chaque application.

Visiter http://localhost:8089/departements. Vous devriez voir la même représentation JSON des departements qu'avant.



Essayer de cliquer sur http://localhost:8089/employes pour voir également les employes.



Exercice 2 : Refaire l'exercice 1 en utilisant une base de donnees H2.
Ajouter a chaque classe controller Departement et Employe, les methodes @GetMapping (), @DeleteMapping (), @PostMapping(), et @PutMapping()