INSA de Toulouse
guermouc@insa-toulouse.fr

Découverte de microservices en utilisant Spring Cloud

Précédemment, vous avez fait des appels de vos microservices en dur dans le code source. Si par exemple l'adresse de déploiement de votre service change, il faut modifier le code pour considérer la nouvelle adresse. Pour remédier à cette limite, vous devriez rendre vos microservices *découvrables* et utilisables par les clients indépendamment de leur adresse de déploiement. C'est ce que vous allez voir maintenant.

a. Compétences visées

Le but est de développer des microservices découvrables en utilisant *Spring Cloud*.

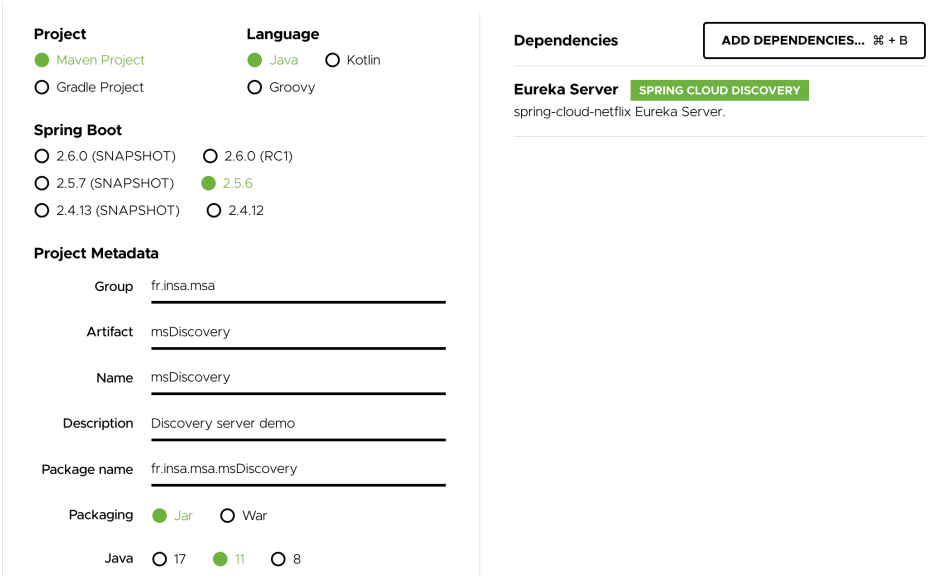
b. Spring Cloud

Spring Cloud offre tous les supports pour la gestion des microservices. Ceci permet de faire face à la volatilité de déploiement des microservices. Plusieurs API existent (Zookeper, Consul, Hysterix, Netflix …etc)

Ici vous allez utiliser l'API développée par la société de streaming vidéo ***Netflix OSS*** (Open Source Software). Parmi ce que cette API offre, nous nous intéressons à ***Eureka*** qui offre un service d'enregistrement (*service Registry*). Ce dernier est un annuaire de services, qui permet de mettre en place un serveur de découverte. Ainsi, les microservices qu'on souhaiterait rendre « découvrables », s'inscrivent sur le serveur, pour être vus par les autres.

c. Création du projet avec Eureka :

Comme vous avez fait précédemment, créez un projet Spring boot en ajoutant cette fois la dépendance *Eureka Server* :



Importez votre projet dans votre IDE.

Dans le fichier pom.xml de votre projet, vous remarquerez la dépendance *Spring-cloud* nécessaire pour ce que vous allez faire.

```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>2.5.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>fr.insa.msa</groupId>
    <artifactId>msDiscovery</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>msDiscovery</name>
    <description>Discovery server demo</description>
    <properties>
        <java.version>11</java.version>
        <spring-cloud.version>2020.0.4</spring-cloud.version>
    </properties>
    <dependencies>
        <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>
```

Retournez à l'application principale de Spring Boot et ajoutez l'annotation `@EnableEurekaServer` nécessaire pour mettre en place le serveur Eureka (votre microservice devient un serveur de découverte) :

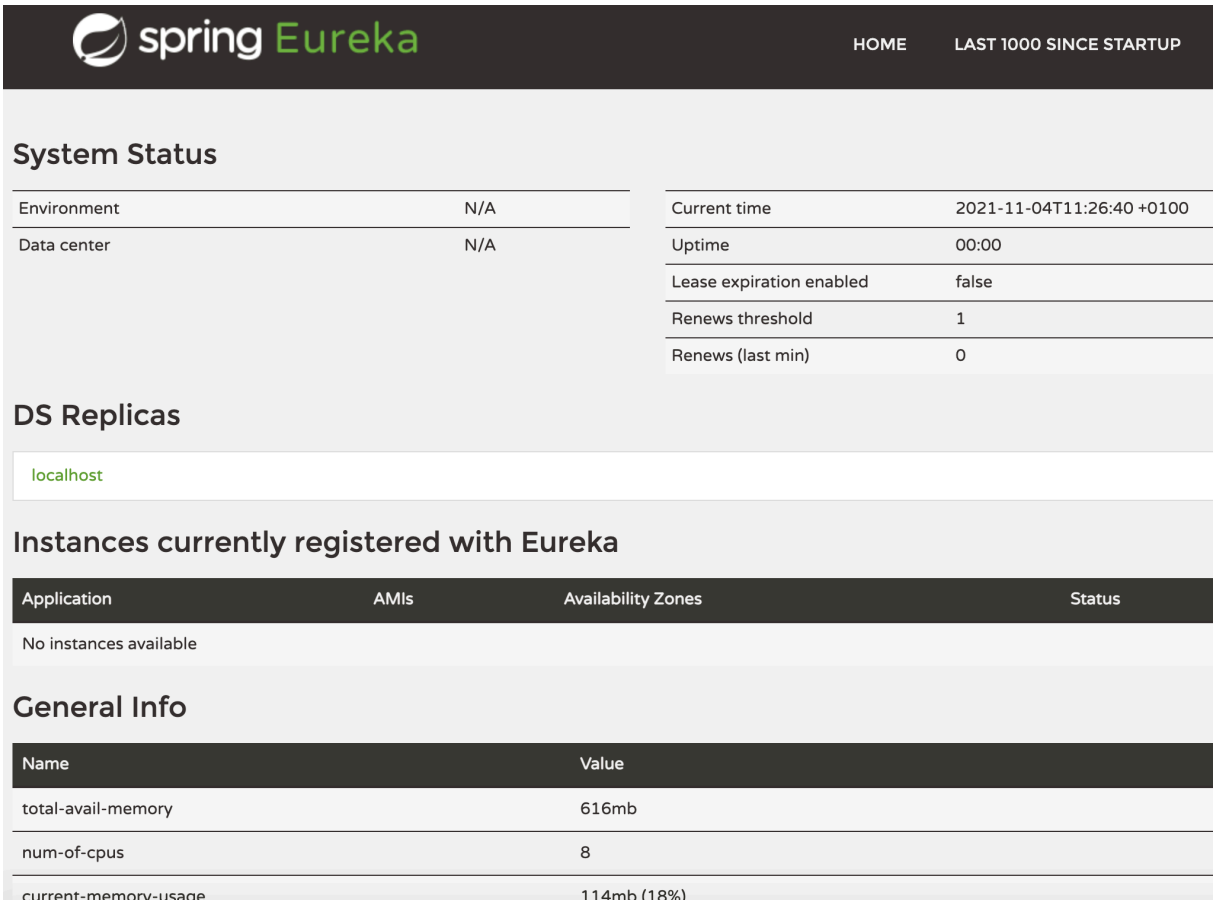
```java
package fr.insa.msa.msDiscovery;

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

@SpringBootApplication
@EnableEurekaServer
public class MsDiscoveryApplication {

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

}
```

Rappelez-vous que *Eureka Service Registry* permet aux microservices de s'enregistrer afin qu'ils soient découvrables. Le *service registry* en soit, n'a pas à s'enregistrer. Cependant, par défaut, au lancement de Spring Cloud, une instance de Eureka Service Registry va essayer de s'enregistrer. Pour empêcher cela et qu'il joue seulement le rôle de serveur, ajoutez les deux dernières instructions :

```
server.port=8761
eureka.client.register-with-eureka=false
```

```
eureka.client.fetch-registry=false
```

Lancez votre application, puis ouvrez Spring Eureka en allant sur : `localhost:8761`



Votre serveur est prêt à être utilisé pour découvrir et enregistrer les miscroservices !
Vous allez publier vos microservices précédemment développés.
Vous allez commencer par le microservice StudentInfoService. Vous allez avoir besoin d'ajouter des dépendances nécessaires pour pouvoir le publier en utilisant Spring Cloud.

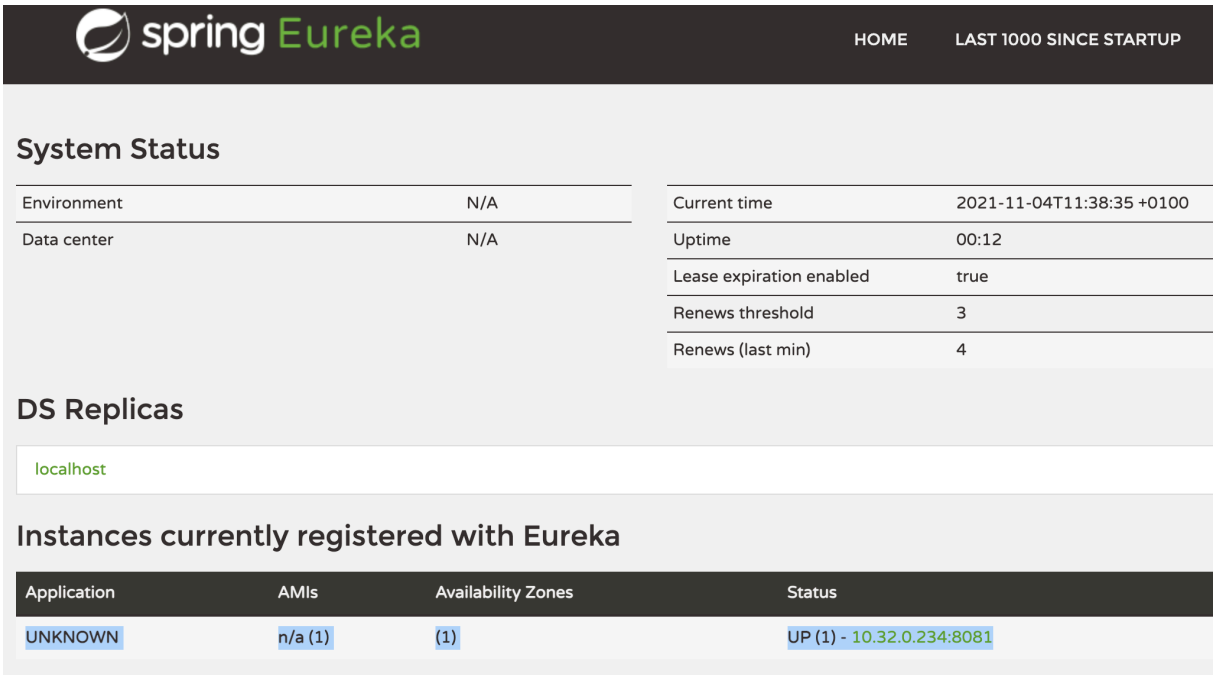
Allez dans le fichier pom.xml de votre microservice et copiez-y les dépendances liées à Spring Cloud que vous pouvez récupérer du fichier pom.xml du projet Spring Cloud que vous venez de créer (voir le commentaire « Nouvel Ajout » dans la figure suivante) :

```
<?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>2.5.6</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>fr.insa.mas</groupId>
        <artifactId>studentInfoService</artifactId>
        <version>0.0.1-SNAPSHOT</version>
```

```
        <name>studentInfoService</name>
        <description>Example of MS</description>
        <properties>
                <java.version>11</java.version>

                <!-- Nouvel ajout -->
                <spring-cloud.version>2020.0.4</spring-cloud.version>

        </properties>
        <dependencies>

                <!-- Nouvel ajout -->
            <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-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>

                <dependency>
                                <groupId>org.springframework.cloud</groupId>
                                <artifactId>spring-cloud-dependencies</artifactId>
                                <version>${spring-cloud.version}</version>
                                <type>pom</type>
                                <scope>import</scope>
                </dependency>
        </dependencies>

        <!-- Nouvel ajout -->
        <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>
```

Lancez l'application du microservice. Actualisez votre serveur. Vous devriez voir que votre microservice a été publié mais sous un nom inconnu (*Unknown*).

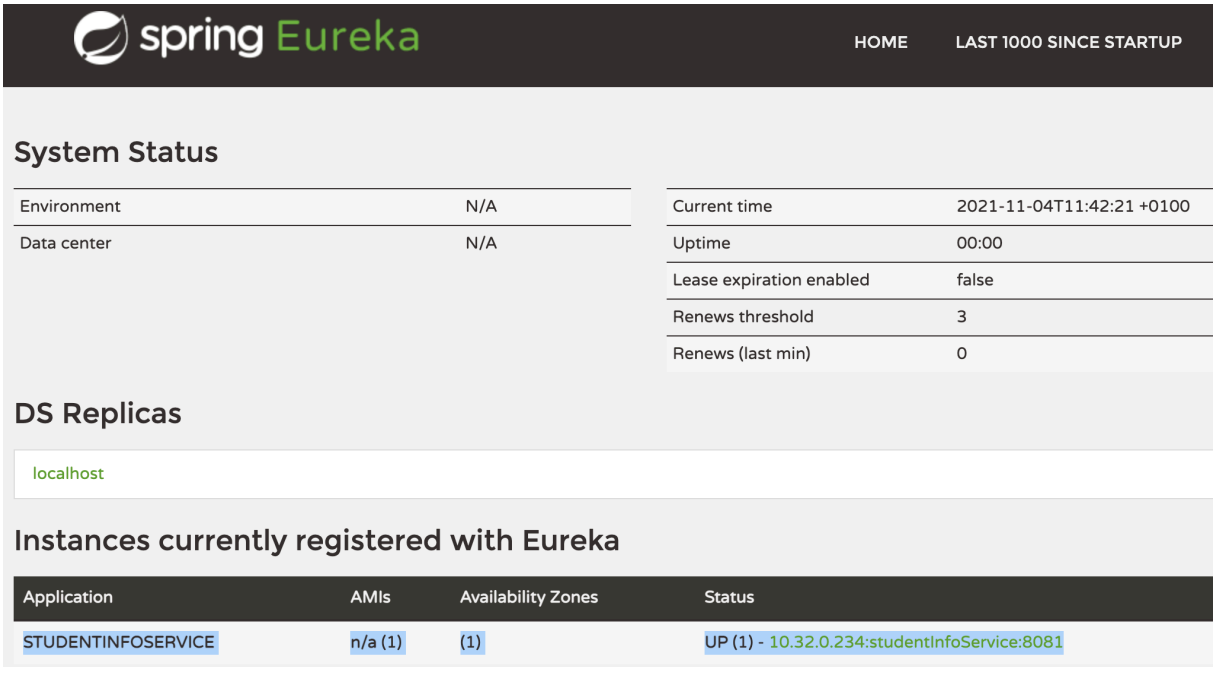
INSA de Toulouse
guermouc@insa-toulouse.fr



Pour que votre microservice ait un nom, allez dans son fichier *application.properties* et ajoutez la ligne suivante :

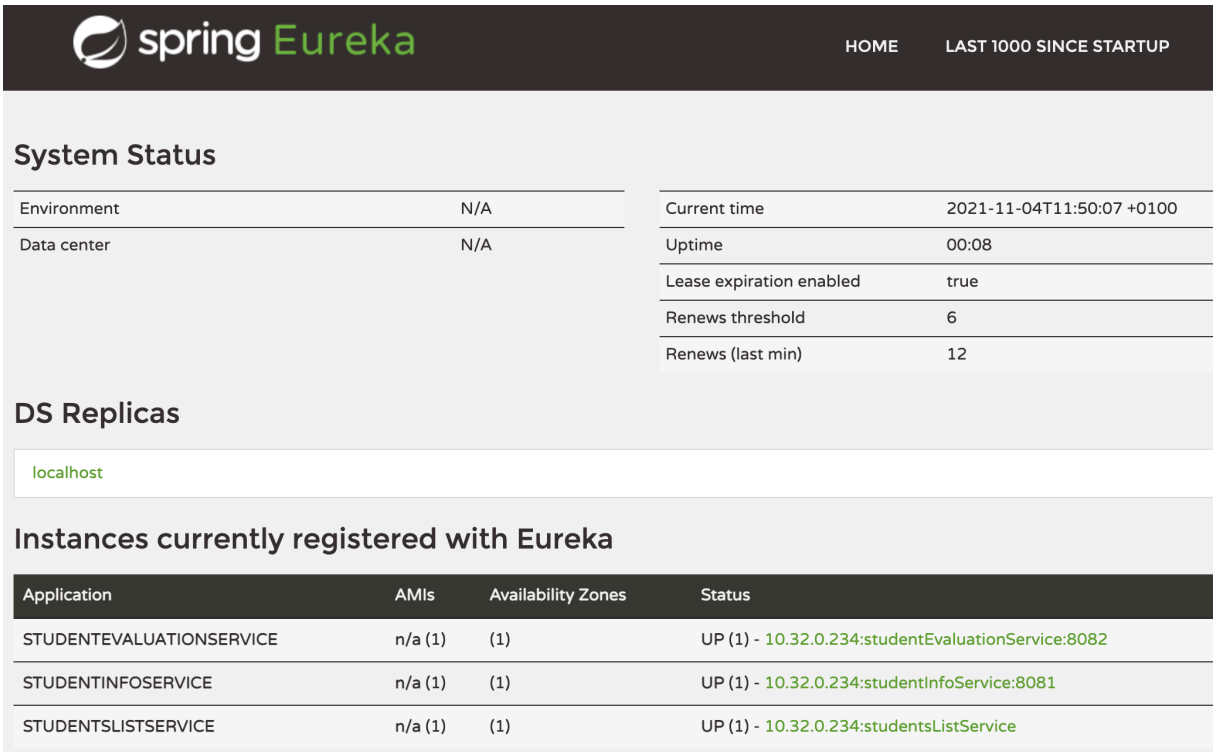
```
spring.application.name=studentInfoService
```

Actualiser votre serveur. Vous devriez voir votre microservice avec un nom.



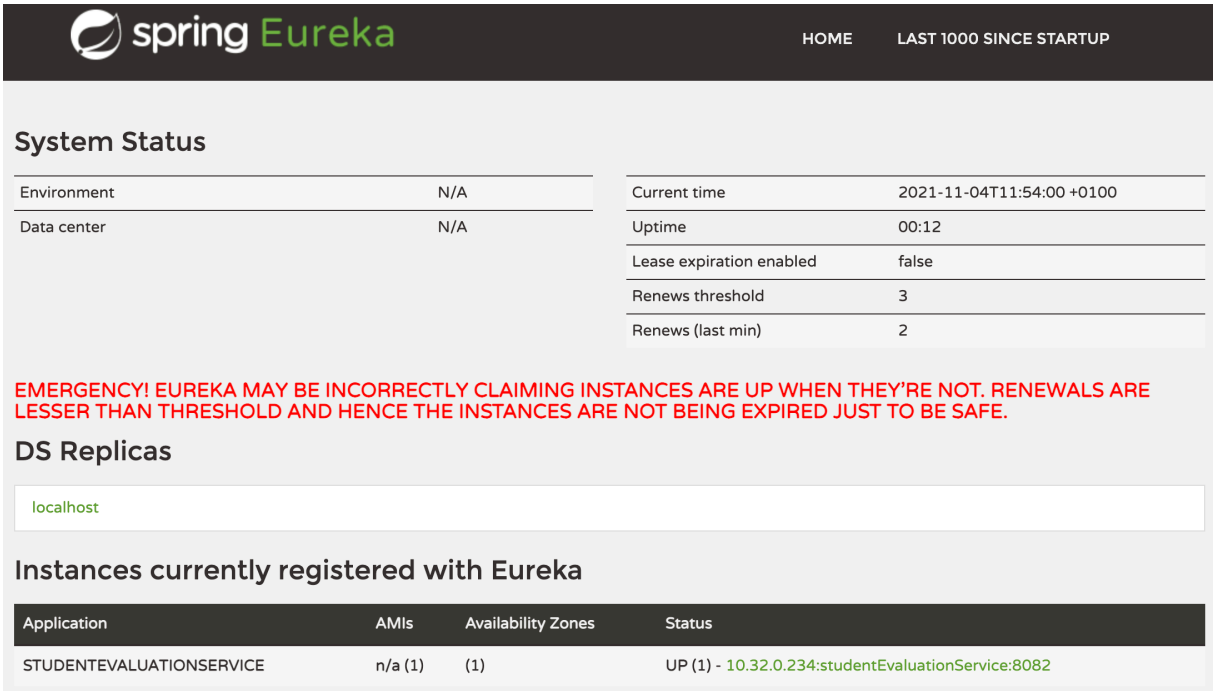
Faites pareil pour les deux autres services. Actualisez Eureka :

INSA de Toulouse
guermouc@insa-toulouse.fr



Vos microservices sont maintenant enregistrés et peuvent être découverts par d'autres services clients.

Remarque :
Si à un moment, après avoir déployé et arrêté des services, vous voyez sur Eureka le message suivant, c'est juste que le serveur vous allerte qu'il y a moins de services déployés. En déployant vos services, le message va disparaitre après quelques temps en actualisant.



d. Appel de microservices :

Ce que vous allez faire maintenant c'est d'appeler vos services en passant par Eureka. Cela va permettre de dissocier vos appels de l'adresse de déploiement de vos microservices. Vous allez utiliser leur nom, tel qu'il est publié sur Eureka.

Rappelez-vous, le service qui se charge d'appeler les services est celui de *StudentsListService*. Vous allez apporter des modifications pour pouvoir appeler les services non pas directement, mais en sollicitant le serveur Eureka.

Dans un premier temps, vous allez d'abord apporter des modifications nécessaires. Reprenez le code source de votre microservice *StudentsListService*. Précédemment, pour appeler les microservices, vous avez utilisé *RestTemplate*. A chaque appel du microservice, une instance de RestTemplate est créée.

Vous allez faire en sorte de créer une seule instance de *RestTemplate* au lancement de l'application du microservice et de l'utiliser à chaque fois que le microservice est appelé. Pour ce faire, vous allez l'ajouter à la classe principale (que vous lancez une seule fois) en tant que bean (annotation `@Bean`). Pour ce faire, ajoutez le code souligné dans la figure suivante à l'application principale du microservice *StudentsListService* :

```
package fr.insa.mas.studentListService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class StudentListServiceApplication {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

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

}
```

Maintenant, vous allez l'associer à votre microservice. Pour cela, vous allez utiliser l'annotation `@Autowired`

Dans la classe de votre microservice, déclarez un objet privé de type RestTemplate précédé de l'annotation `@Autowired`. Ce qui va se passer c'est que lorsque vous lancez l'application Spring Boot, une instance de RestTemplate va être créée (@Bean). Puis, cette instance sera passée au microservice à chaque fois qu'il en a besoin (@Autowired).

```java
@RestController
@RequestMapping("/students")
public class StudentListResource {

    @Autowired
    private RestTemplate restTemplate;
```

En ajoutant ce code, vous n'avez plus besoin d'instancier un objet RestTemplate dans le code de votre microservice. Enlevez donc la ligne d'instanciation d'un objet RestTemplate dans le code du microservice.

Testez votre microservice pour vérifier qu'il fonctionne toujours.

Maintenant vous allez reprendre l'appel de vos microservices en utilisant juste leur nom tel qu'il est publié sur le serveur Eureka.

Dans la classe principale de votre microservice *StudentListService*, ajoutez l'annotation `@LoadBalanced` pour indiquer que lorsque le microservice reçoit une requête, celle-ci ne va pas être envoyée directement au microservice mais va être redirigée vers Eureka.

```java
package fr.insa.mas.studentListService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class StudentListServiceApplication {

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

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

}
```

Sachant que Eureka identifie chaque microservice avec son nom, les appels doivent considérer les noms des microservices. Retournez au microservice *studentListResource* et modifiez l'URL des appels des deux microservices studentInforService et studentEvalService comme suit :

http://localhost:8081/student/ ➔ http://studentInfoService/student/

http://localhost:8082/evaluation/ ➔
http://studentEvaluationService/evaluation

Ainsi, l'adresse de déploiement de vos microservices devient transparente au client lors des appels. Eureka se charge de rediriger les requêtes vers les microservices en utilisant leur vraie adresse de déploiement.

INSA de Toulouse
guermouc@insa-toulouse.fr

Testez !

Remarque : Pour tester vos microservices via un navigateur (ou client Rest type Postman), il faut utiliser les adresses complètes de vos microservices. On peut utiliser les nom des services indépendamment de leur adresse de déploiement au sein des services clients qui eux même s'enregistrent au serveur de découverte.