

# Eureka Naming Server

Dans les articles précédents nous avons abordé les microservices. Nous avons créé un microservice, nous avons modifié ce microservice pour qu'il consomme les données d'un autre microservice. Et nous avons vu comment faire monter en charge ce microservice et le rendre résilient et disponible.

Nous avons utilisé RIBBON pour distribuer la charge entre les deux instances de notre service de cotes. Mais nous avons saisi en dur l'adresse du service dans le service 'Paris', ce qui veut dire que si on déplace le microservice des cotes ou si nous devons ajouter de nouvelles instances du service il faudra modifier le code du service Paris et le re-déployer, ce qui n'est pas très « devops friendly ».

Heureusement les développeurs de Netflix ont pensé à tout, et ils nous proposent Eureka Naming Server pour résoudre ce problème.

Eureka est un service basé sur une API REST dont le rôle principal est trouver à la volée des instances de service et de répartir la charge entre les services. Eureka propose également un client Java - Eureka Client - qui permet de communiquer très facilement avec le serveur.

## Comment créer un serveur Eureka ?

Vous êtes maintenant habitué, avec Spring Boot c'est très simple. On va créer un nouveau projet sur <https://start.spring.io/> :

Choisir un nom de groupe, un nom pour l'artefact maven, n'oubliez pas d'inclure la dépendance "Eureka Server".

**SPRING INITIALIZR** bootstrap your application now

**Generate a** Maven Project **with** Java **and Spring Boot** 2.1.3

### Project Metadata

Artifact coordinates

Group

Artifact

### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

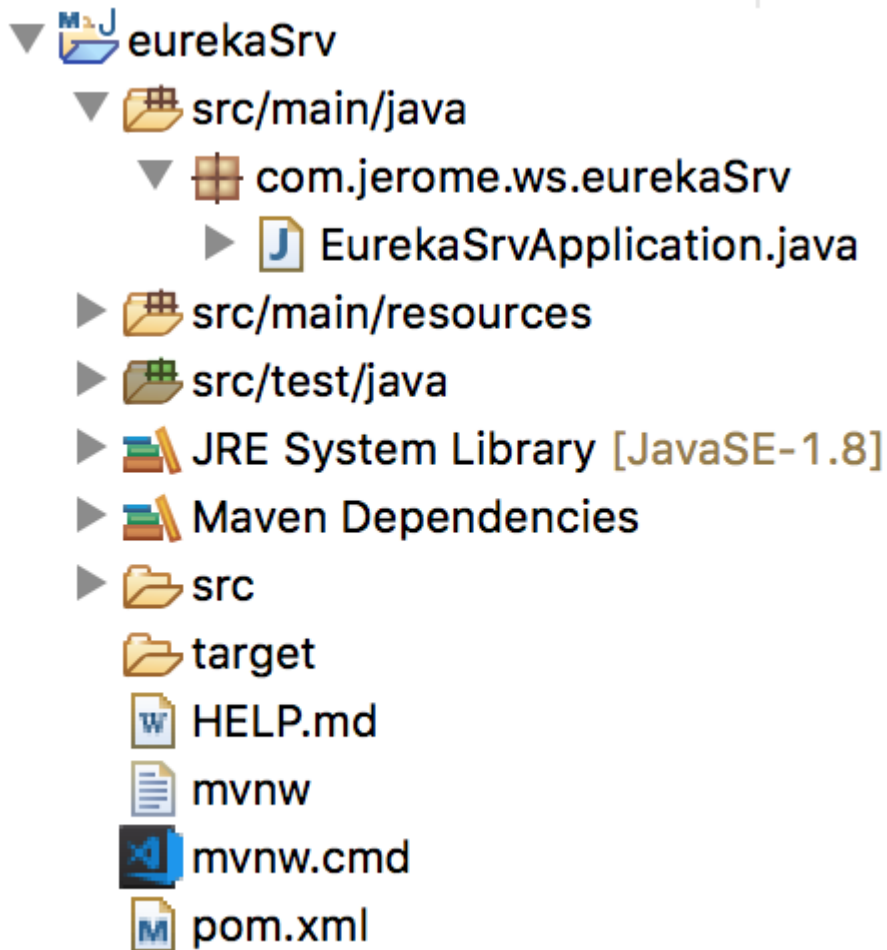
Selected Dependencies

Eureka Server

**Generate Project** % + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)

On télécharge le projet et on l'ouvre avec son IDE préféré :



Il faut activer le Eureka dans notre projet, avec une annotation `@EnableEurekaServer` dans la classe de lancement :

```
package com.jerome.ws.eurekasrv;

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

@SpringBootApplication
@EnableEurekaServer
public class EurekaSrvApplication {

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

}
```

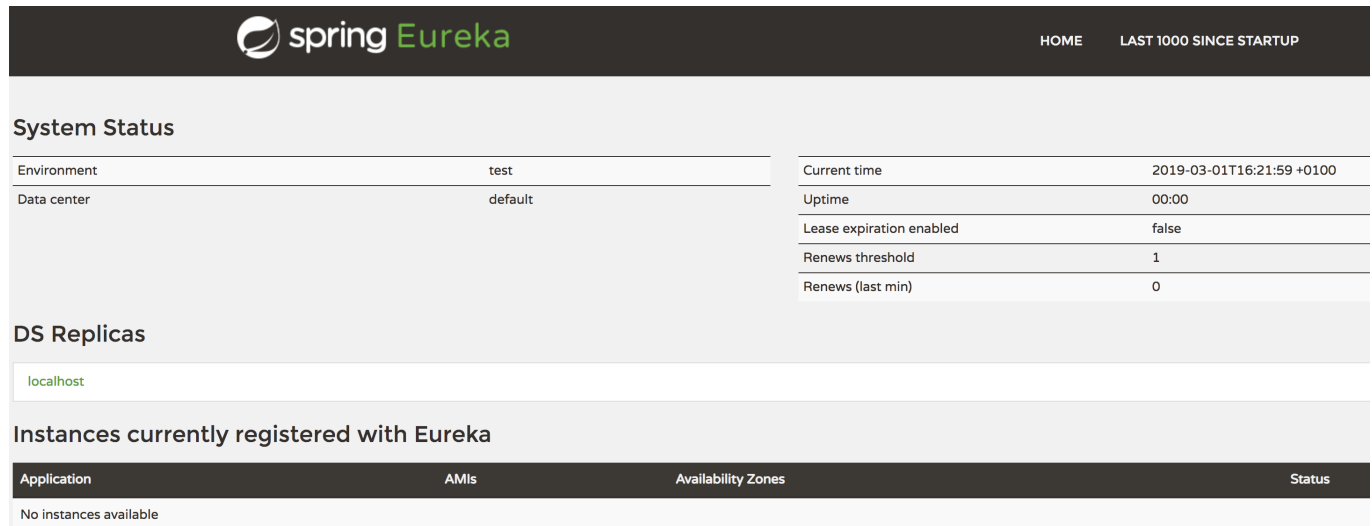
Ensuite il faut définir le nom de l'application et le port utilisé par le serveur dans **application.properties** :

```
spring.application.name=betting-eureka-naming-server
server.port=8761
```

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

On démarre le serveur comme nous avons fait pour les autres applications Spring Boot en lançant `EurekaSrApplication` en tant qu'application Java.

Une console web est disponible sur le port configuré ci dessus :



The screenshot shows the Spring Eureka web console. At the top, there's a header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. Below the header, the 'System Status' section is displayed, containing two tables. The left table shows 'Environment' as 'test' and 'Data center' as 'default'. The right table shows 'Current time' as '2019-03-01T16:21:59 +0100', 'Uptime' as '00:00', 'Lease expiration enabled' as 'false', 'Renews threshold' as '1', and 'Renews (last min)' as '0'. Below this, the 'DS Replicas' section shows 'localhost'. The 'Instances currently registered with Eureka' section has a table with columns 'Application', 'AMIs', 'Availability Zones', and 'Status', but it shows 'No instances available'.

## Eureka Client

Pour l'instant aucune instance de micro service n'est déclarée sur le serveur Eureka, nous allons modifier nos services pour qu'ils soient configurés par Eureka.

Dans deux services, il faudra ajouter une dépendance vers le client Eureka dans le fichier `pom.xml` :

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

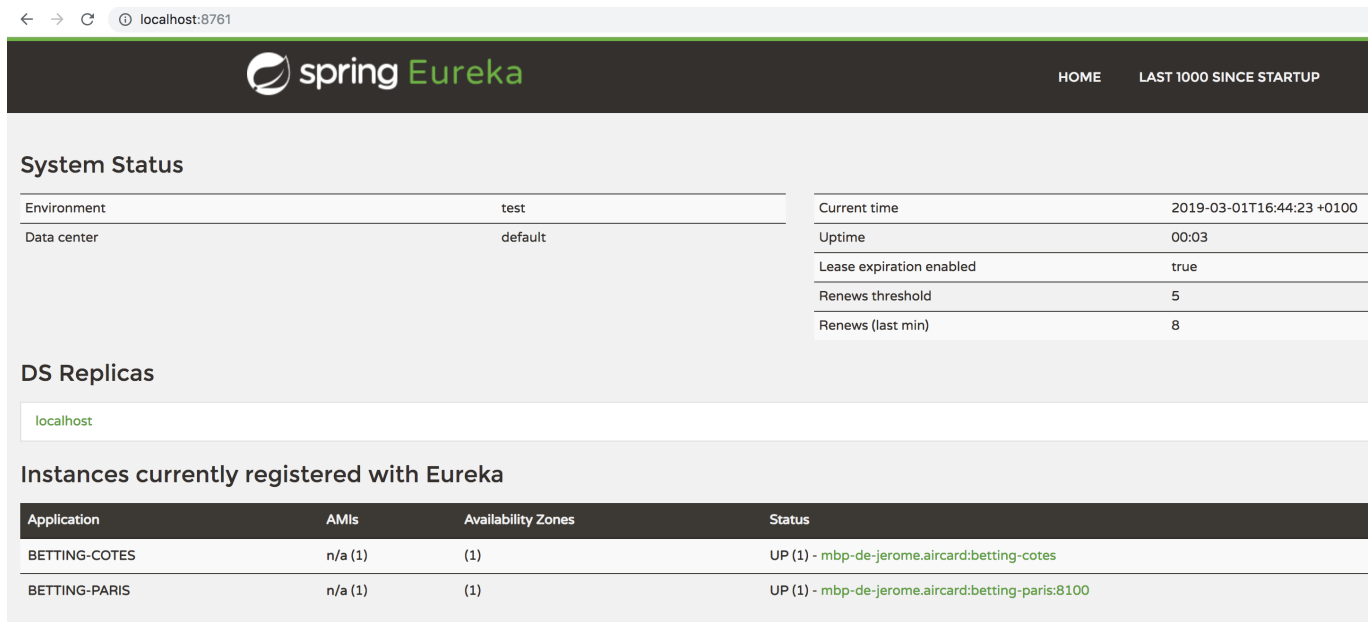
Puis on va définir dans les services à quelle adresse ils trouveront leur serveur Eureka, dans le fichier `application.properties` :

```
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

On démarre toutes les instances de nos microservices, pour rappel nous avons :

- Le service qui calcule les paris sur le port 8080
- Deux instances du service de gestion des cotes sur les ports 8100 et 8101

On retourne sur la console Eureka et on voit que les services sont désormais enregistrés :



The screenshot shows the Spring Eureka web interface. At the top, there's a navigation bar with the Spring Eureka logo and links for HOME and LAST 1000 SINCE STARTUP. Below this, the 'System Status' section displays two tables. The first table shows 'Environment' as 'test' and 'Data center' as 'default'. The second table shows 'Current time' as '2019-03-01T16:44:23 +0100', 'Uptime' as '00:03', 'Lease expiration enabled' as 'true', 'Renews threshold' as '5', and 'Renews (last min)' as '8'. Below the system status, there's a 'DS Replicas' section showing 'localhost'. Finally, the 'Instances currently registered with Eureka' section contains a table with two rows: 'BETTING-COTES' and 'BETTING-PARIS'. Both are in 'UP' status with one instance each, managed by 'mbp-de-jerome.aircard'.

Application	AMIs	Availability Zones	Status
BETTING-COTES	n/a (1)	(1)	UP (1) - mbp-de-jerome.aircard:betting-cotes
BETTING-PARIS	n/a (1)	(1)	UP (1) - mbp-de-jerome.aircard:betting-paris:8100

Dans le service des Paris nous avons toujours l'adresse des services de gestion des cotes qui sont en dur dans un fichier de configuration. Comment utiliser Eureka ? Il suffit de supprimer la configuration Ribbon et comme par magie notre service va utiliser Eureka pour récupérer l'adresse des services.

On supprime cette ligne dans la configuration du service des Paris :

```
cote-service.ribbon.listOfServers=localhost:8080,localhost:8081
```

Nous pouvons maintenant ajouter / supprimer des instances du service des 'Cotes' et le service des 'Paris' utilisera à chaque fois une instance différente du service des cotes. Si une instance s'arrête, la requête sera routée sur une autre instance.

Pour finir ce tutoriel sur les micro service je vous propose comme exercice d'ajouter plusieurs instances du service des Cotes et de vérifier leur inscription dans Eureka. Puis créer un nouveau service (par exemple un service qui retourne l'URL des photos du logo des équipes de foot) et d'ajouter un appel de ce service dans le service des paris tout en utilisant les mécanismes vu dans les articles précédents.

Nous avons vu sur cette petite série d'articles comment mettre en place une architecture micro service en utilisant Spring Boot et des outils comme Ribbon et Eureka. Il resterait à mettre en place un outil de gestion des logs commun à tous nos services et voir comment déployer les micro services sur AWS ou sous forme de containers Docker !