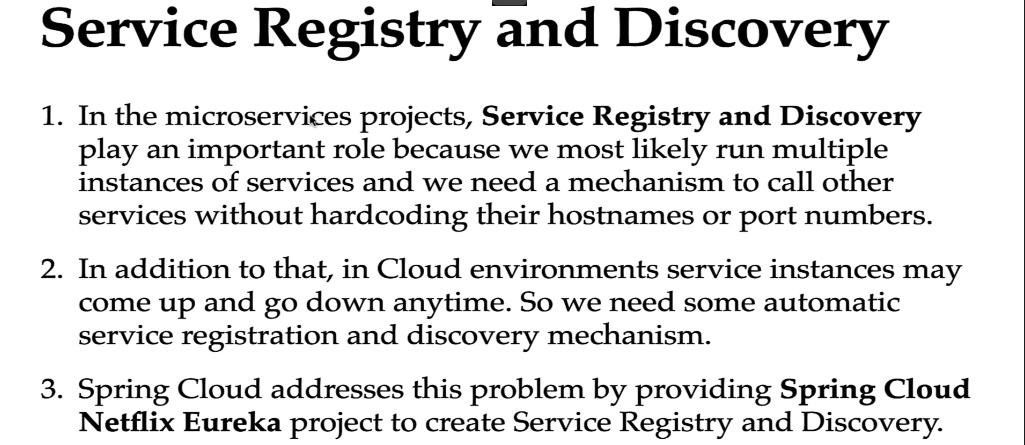
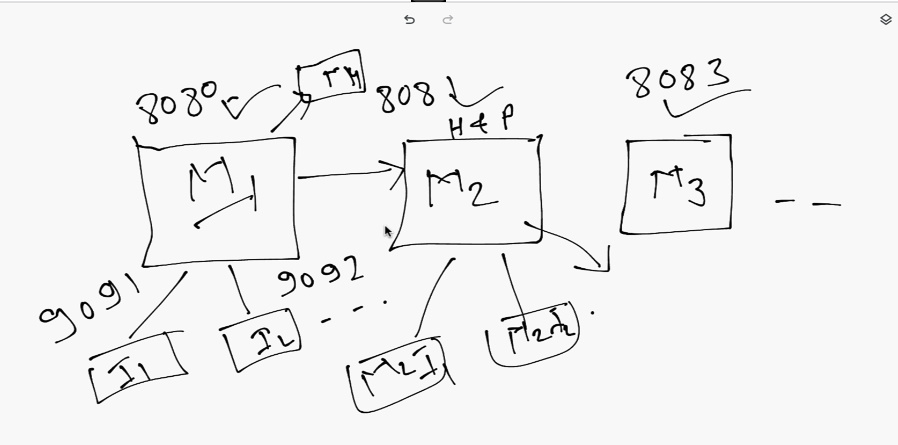
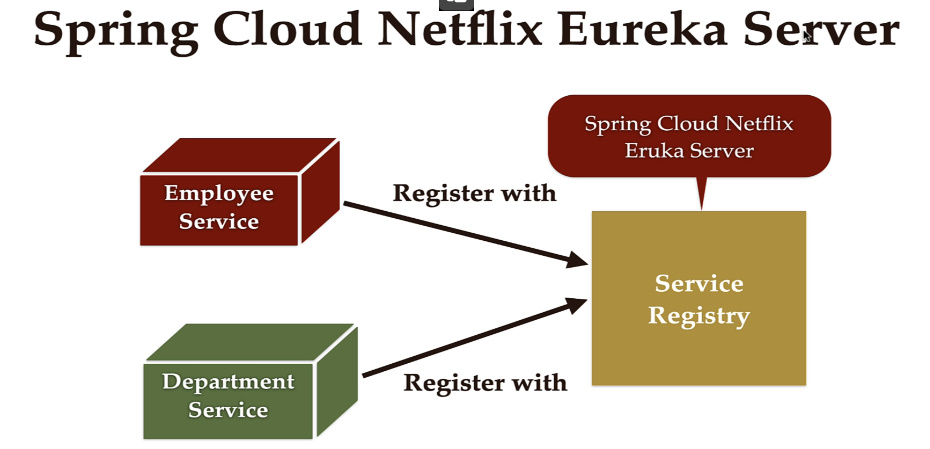
**Spring Cloud Netflix Eureka Server**

* Este folosit pentru a crea Service Registry
* 
* De ex, fie ca avem 3 microservices:



* Fiecare din ele ruleaza pe un port, si instantele lor lucreaza pe alt port
* Microservice 1 trebuie cumva sa stie toate porturile la Microservice2 si la instantele sale deci, si asta e problematic, asa cum noi instante pot aparea sau altele pot fi inchise. Ar fi mai simplu sa ne referim la un MicroService cu un ID.
* Asta si va face Service Registry
* Microservice1 nu va trebui sa stie porturile si toate astea, ci va putea comunica printr-un ID mult mai usor.
* Spring Cloud Netflix Eureka Server cam asa arata schematic:



Employee si Department Service vor fi **Eureka Clients**

**Creating**

1. **Cream Service Registry microservice**

* Service Registry va fi un project separat, un microservice
* El va avea ca dependenta doar Eurika Server

1. **Facem microservice ca Eureka Server**

Doar folosim anotatia

@EnableEurekaServer

Ea va face ca el sa fie Service registry

1. **Dezactivam Eureka Client**

* In mod default, eureka server este si eureka client, dar noi nu avem nevoie ca Service registry sa fie si client, ci doar server
* Pentru asta, adaugam niste proprietati:
* spring.application.name=SERVICE-REGISTRY  
  server.port=8761  
  eureka.client.fetch-registry=false  
  eureka.client.register-with-eureka=false

spring.application.name poate lua orice valoare

1. **Rulam Aplicatia**

Acum, daca vom intra pe

localhost:8761, vom vedea o interfata grafica la Eureka server despre microservices

1. **Adaugam microserviciile clients, gen DepartmentService**

* pentru asta, in microservicii adaugam o dependenta eureka client deja
* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
   <version>4.0.3</version>  
  </dependency>

1. **Adaugam in client o properietate ce il va lega de eureka server microservice**

eureka.client.service-url.defaultZone = http://localhost:8761/eureka/

Punem host:port la eureka server, nu la microsrrvie in care scriem asta!

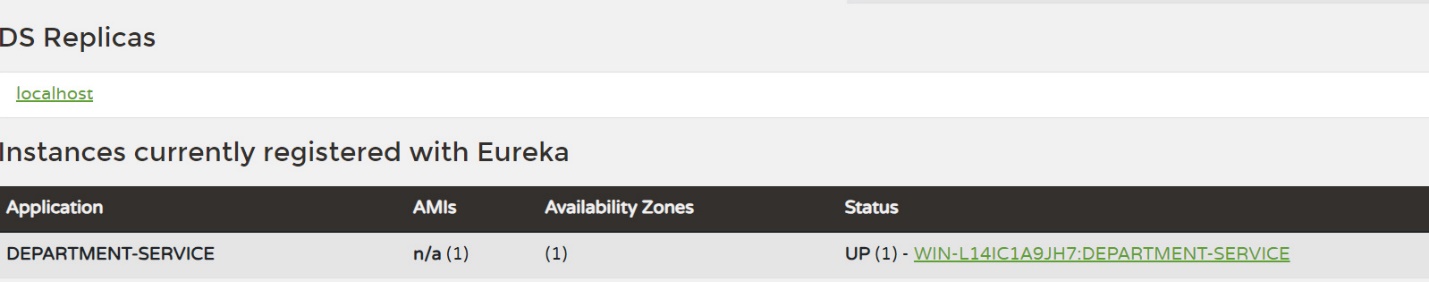
1. **Dam un nume la microservice Department, si acest nume va fi un id la microservice pe care Eureka il va folosi**

spring.application.name=DEPARTMENT-SERVICE

Anume acest id, sau nume, va fi folosit si afisat de Eureka pentru acest microservice.

Atentie! **~~@EnableEurekaClient~~** nu mai e nevoie de a fi pusa in start class din microservice!

Acum, il vedem in eureka server



**Running multiples instances of a microservice**

* S-ar putea sa avem nevoie sa rulam mai multe instante ale unui microservice, ca de ex la DepartmentService
* Avantajele la a crea mai multe imstance la un service sunt:

1. Viteza mai mare de raspuns, asa cum avem mai multe microservice identice, deci mai multe requesturi identice pot fi luate
2. Putem rula acelasi microservice pe difertie domains
3. Daca un microservice e down, se vor face calluri la cele disponibile cu acelasi nume, si asta e bine.
4. Putem rula acelasi microservice cu scopes diferite, gen unul pentru production, altul pentru test
5. Daca modificam un microservice, ca el sa nu fie inchis, cream o noua instanta si il lasam deschis pentru clienti cat lucram la el

* Pasi de urmat:

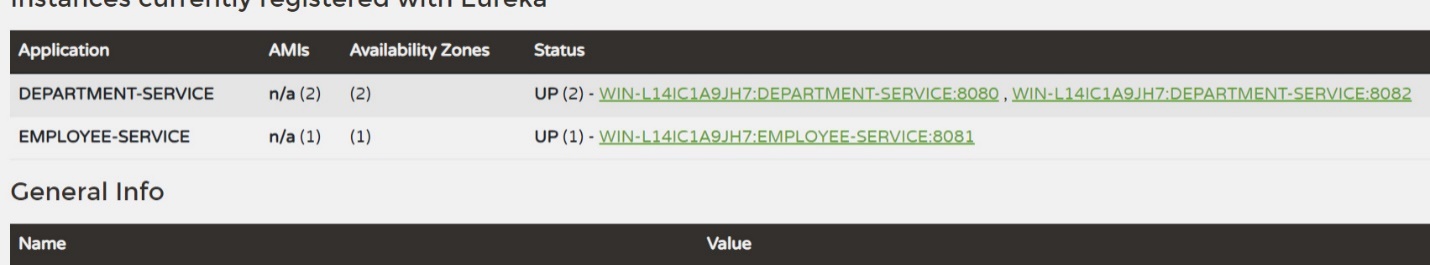
1. Cream package la microservice
2. Deschidem terminal de la intellij, si folosim:

**cd department-service/target**

**java -jar packaje.jar --server.port=8082**

Punem ce port vrem, dar sa fie unul liber.

1. Eurika client, adica microservice acum rulat, ca a 2 instanta, e inregistrat automat in Eurika Server



**Load Balancing with Eureka**

- O facem prin intermediu la **Spring Cloud LoadBalancer**

- EmployeeService foloseste Spring Cloud Feign pentru a face un request la DepartmentService, dar acum avem mai multe instante de DepartmentService, si trebuie cumva sa facem ca EmployeeService sa apeleze una din ele care e disponibila ori oricare daca ambele sunt

- Spring Cloud LoadBalancer se ocupa de asta

- Nu necesita dependenta, caci Eureka deja o contine

- Tot ce trebuie sa facem este sa modificam url din Spring Cloud Feign client in Numele la service pus de noi si gata, Spring Cloud LoadBalancer se va ocupa de tot

- Deci, doar punem name = “” si gata, Spring Cloud LoadBalancer se va ocupa de a obtine host si ports de la Eureka, va apela microservice instance disponibil etc.

@FeignClient(name = "DEPARTMENT-SERVICE")  
public interface DepartmentFeignClient {  
 @GetMapping("/api/departments/{department-code}")  
 DepartmentDto getDepartmentByDepartmentCode(@PathVariable("department-code") String departmentCode);  
}