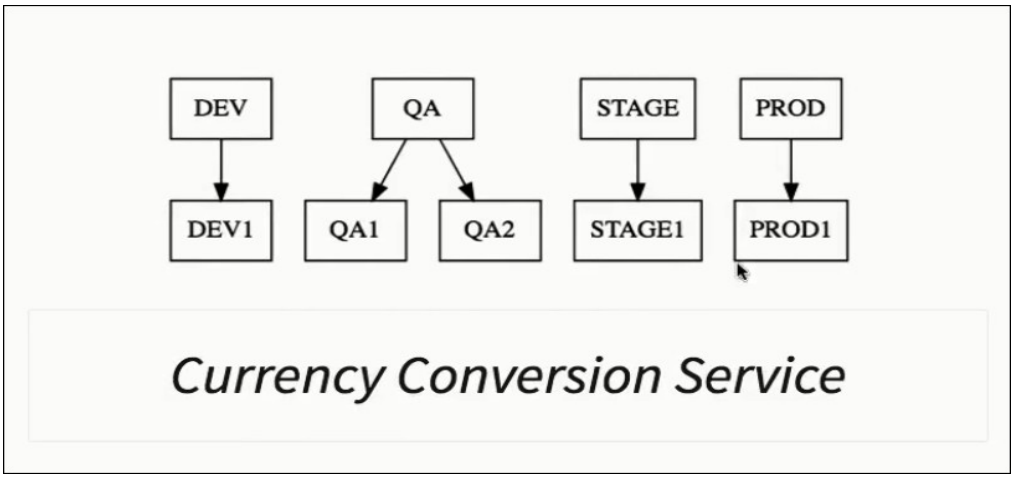
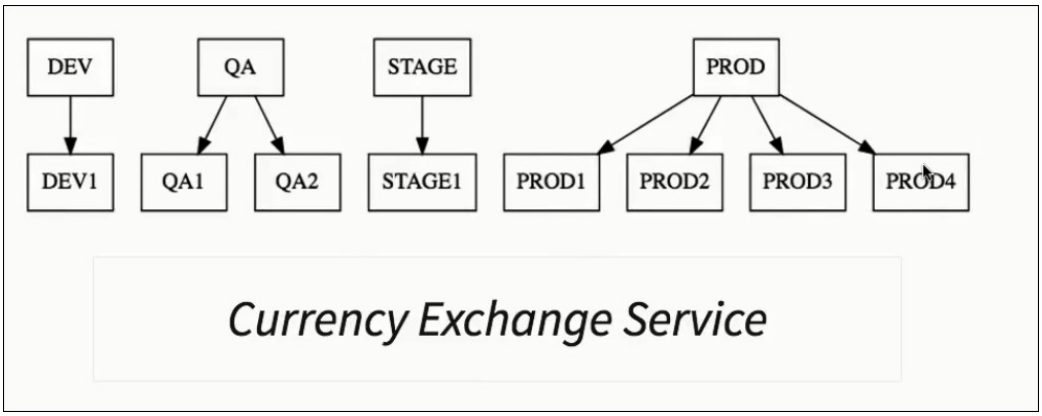
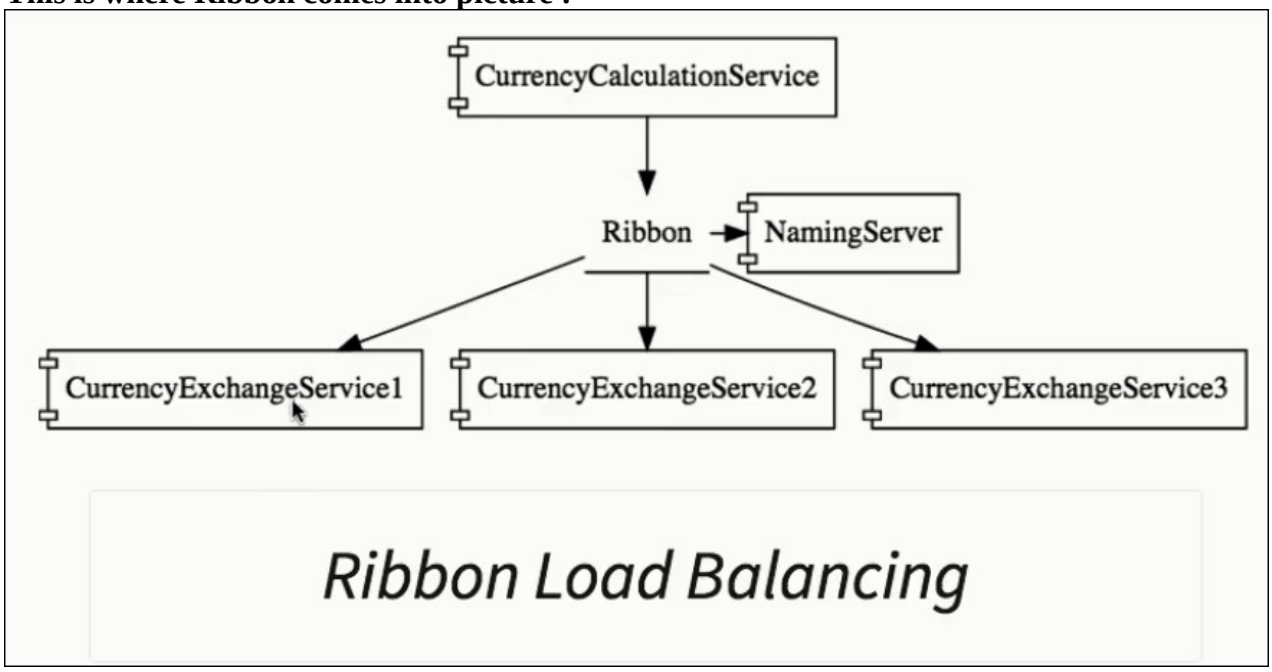
Lecture 79,80: Client Side Load Balancing with Ribbon

With previous approach , only CurrencyConversionService could only talk to any one instance of CurrencyExchangeService. With Ribbon implementation , we'll try to talk to any instance of CES.





This is where Ribbon comes into picture :



Step1: Add pom dependency

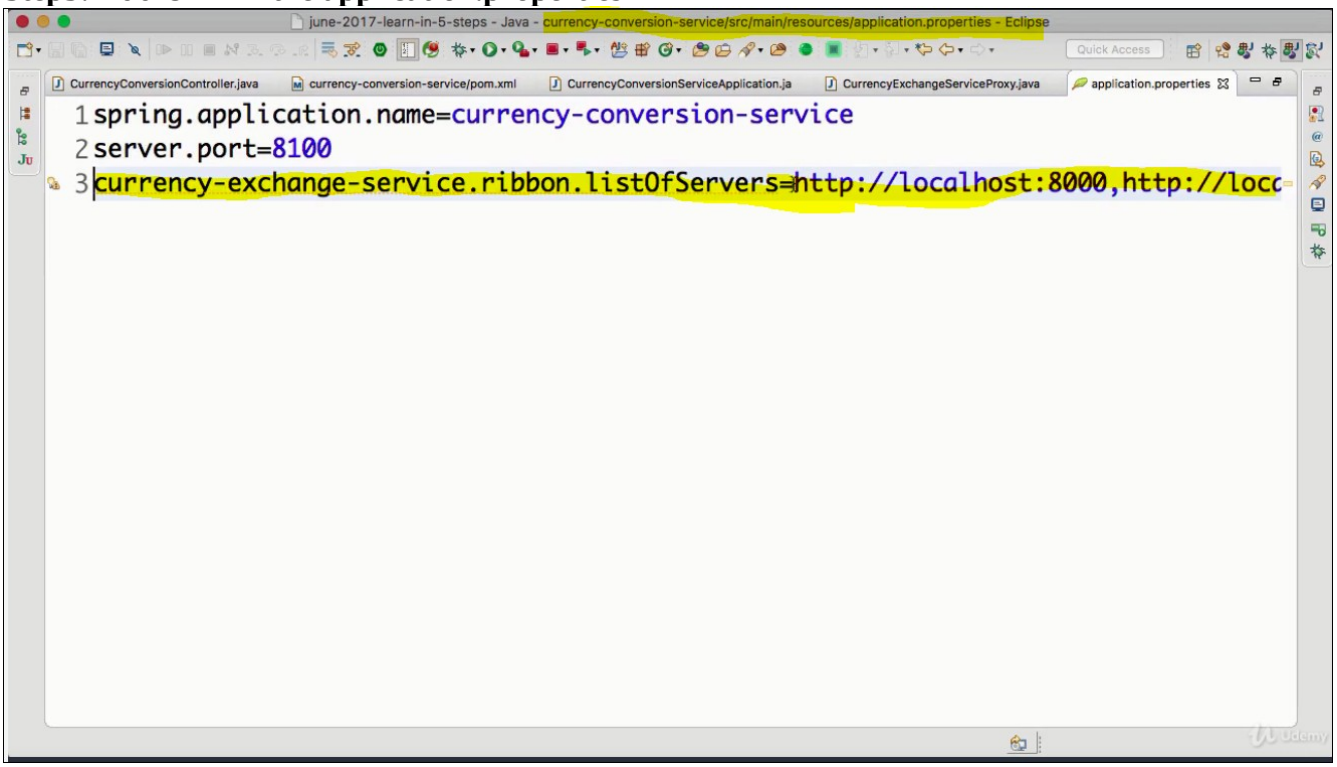
```
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
```

>2.0.0M3 and >2.0.0 use artifactId
spring-cloud-starter-netflix-ribbon

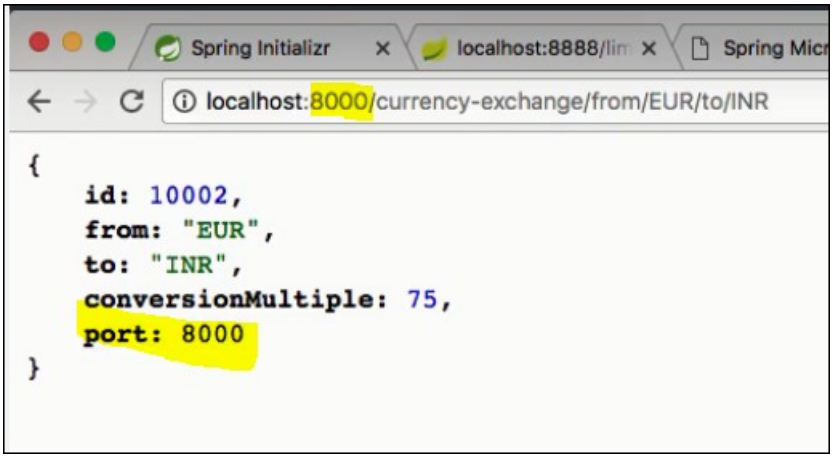
Step2: Enable Ribbon

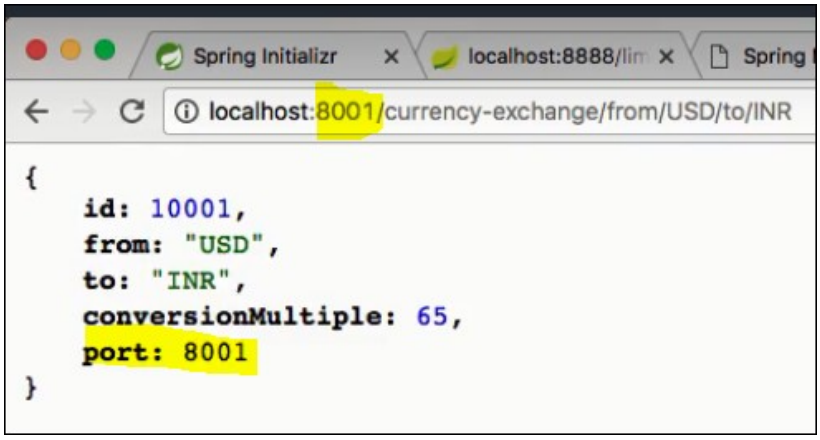
```
package com.in28minutes.microservices.currencyconversionservice;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

//@FeignClient(name="currency-exchange-service", url="localhost:8000")
@FeignClient(name="currency-exchange-service")
@RibbonClient(name="currency-exchange-service")
public interface CurrencyExchangeServiceProxy {
    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public CurrencyConversionBean retrieveExchangeValue
        (@PathVariable("from") String from, @PathVariable("to") String to);
}
```

Step3: Add URL in the application.properties

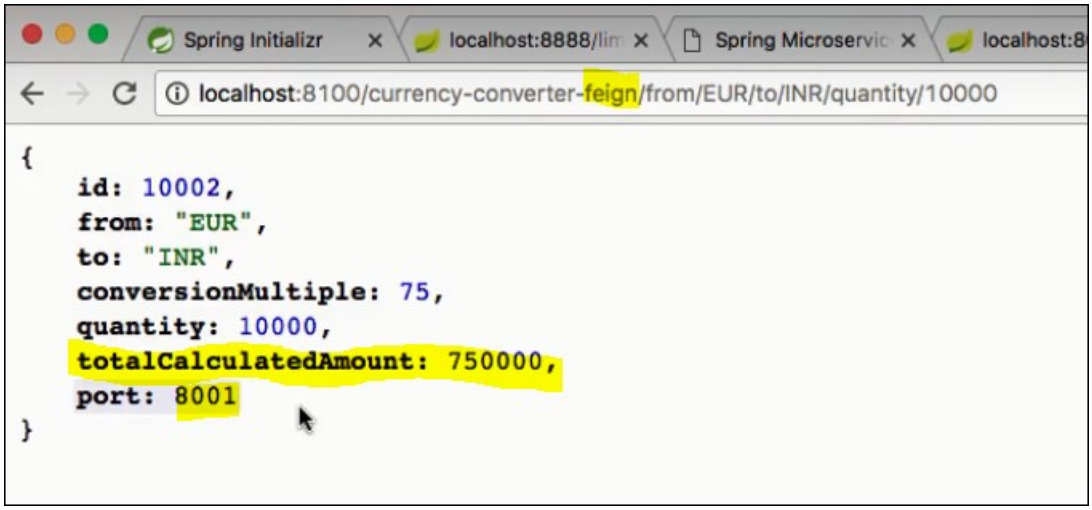


Step4: Launch Currency Exchange Service on 8000 & 8001 port & test that they are available.





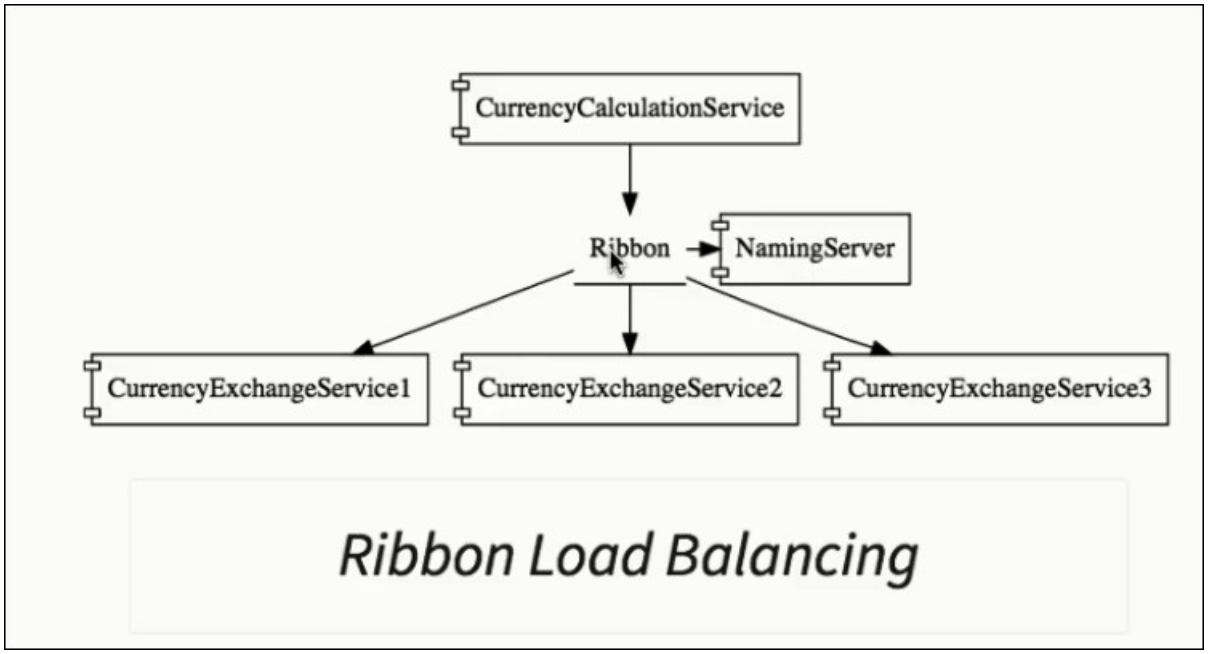
Step5: Invoke currency concersion service & see the results:



Give a 2nd request & the request is delegated to CES instance on port 8000 .

Lecture 81: Understand the need of NAMING SERVER

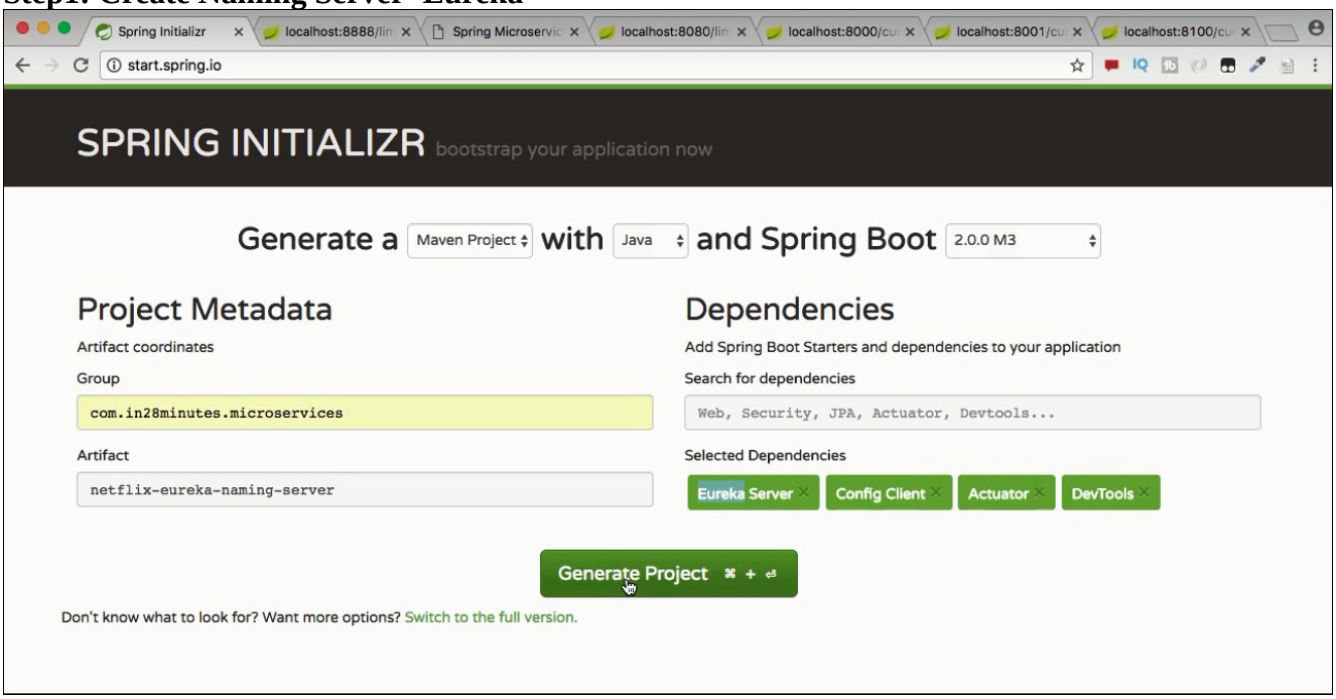
Q: What will happen when one of the instance of CES service is down?Will ribbon able to manage
Ans: No.CES property needs to be changed
Q: What will happen when one of the instance of CES service is added?Will ribbon able to manage
Ans: No.CES property needs to ne changed
This is where NAMING SERVER (EUREKA) comes into picture. When ever a service instance is added or removed , it should be registered or revoked from the NAMING SERVER. This is called SERVICE REGISTRATION
Similarly all the service communicating to another service needs to check with NAMING SERVER on the availability of the target service. This is called SERVICE DISCOVERY.



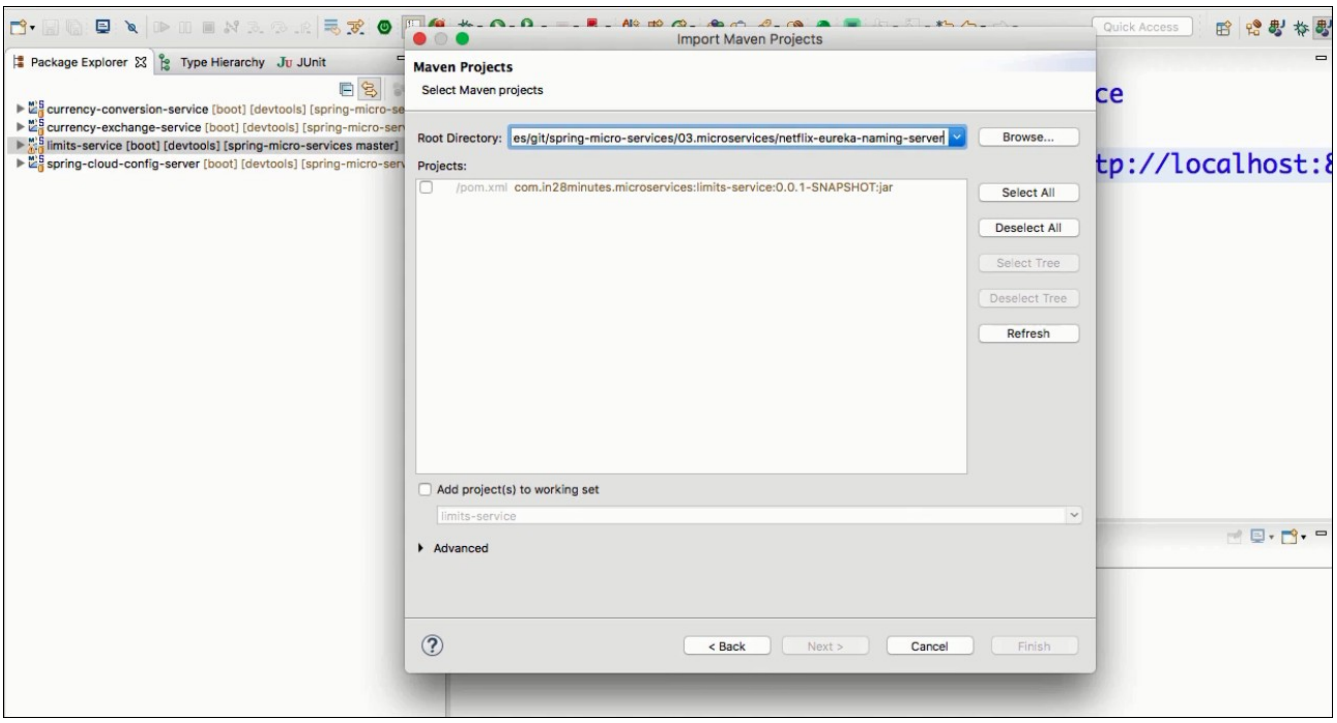
Lecture 82: Setting up Naming Server

- **Create Naming Server- Eureka**
- **Register Currency Exchange Service to Eureka**
- **Register Currency Conversion Service to Eureka**
- **Configure Ribbon to talk to Eureka.**

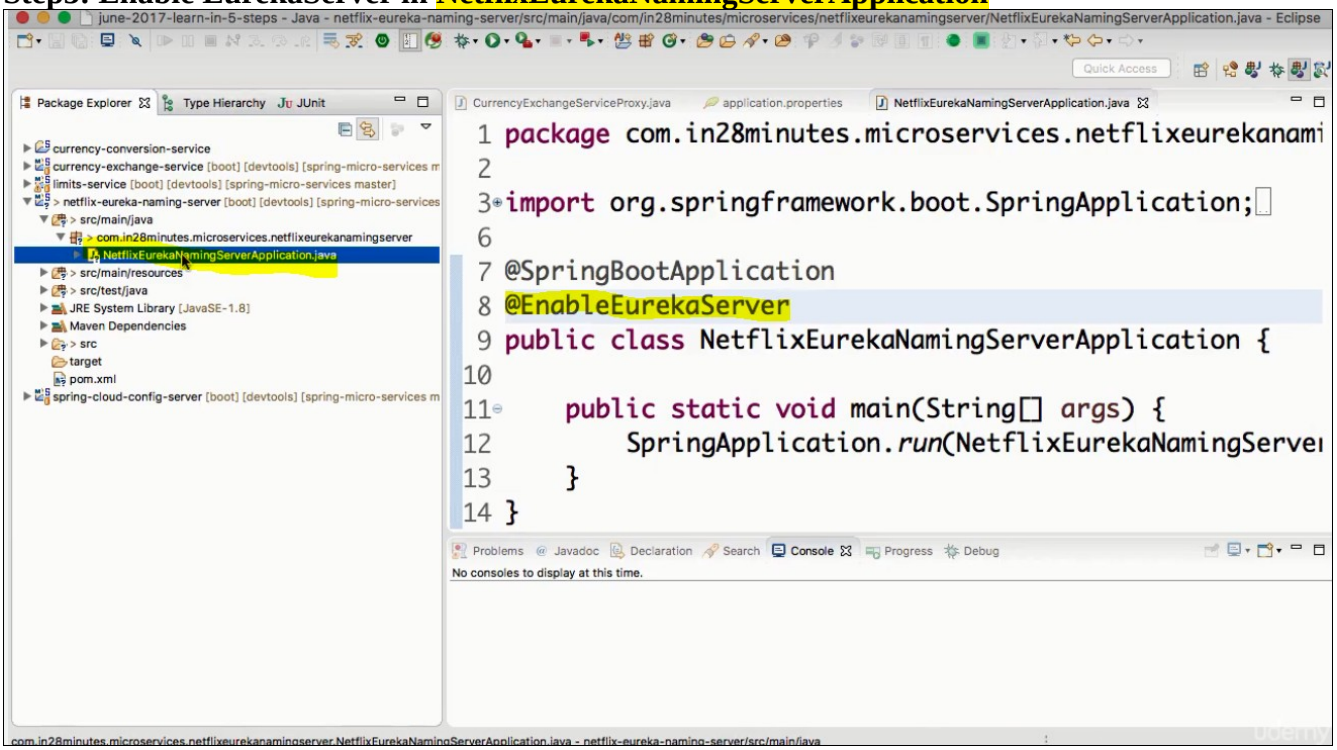
Step1: Create Naming Server- Eureka



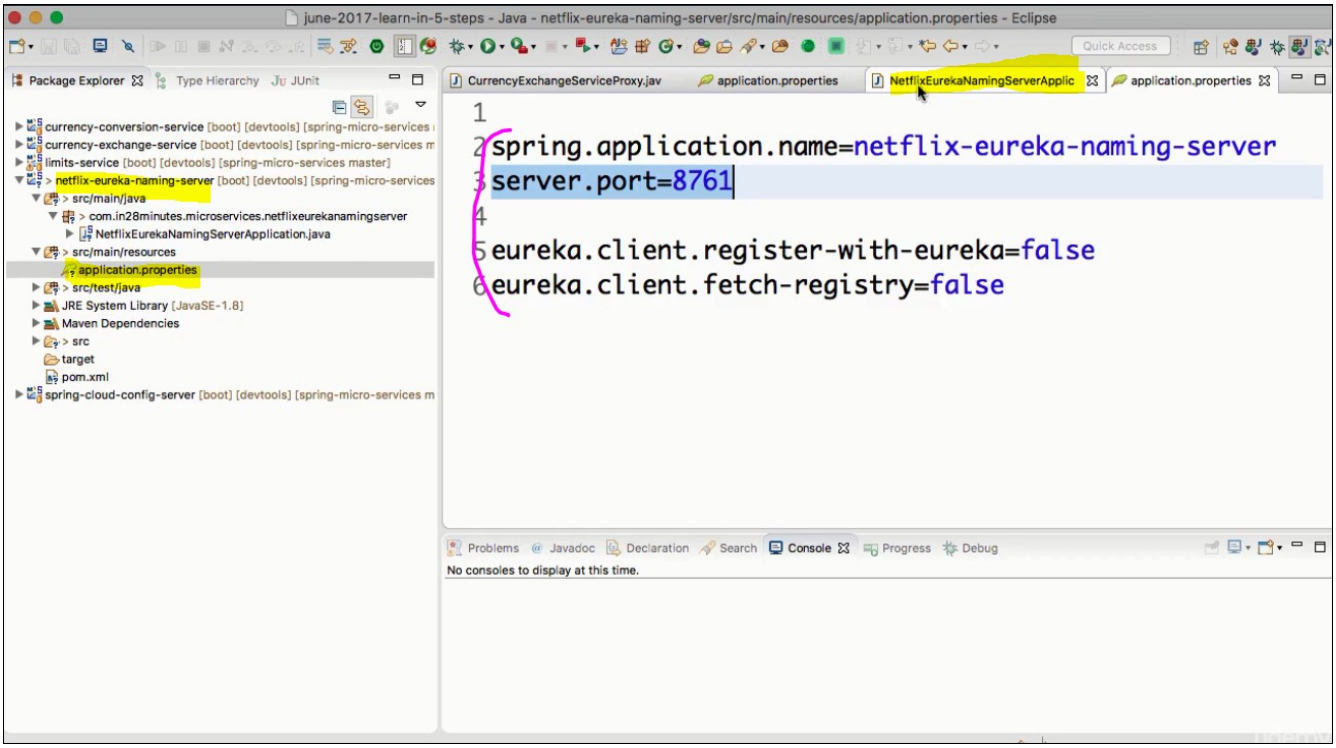
Step2: Download and import it to workspace as maven project



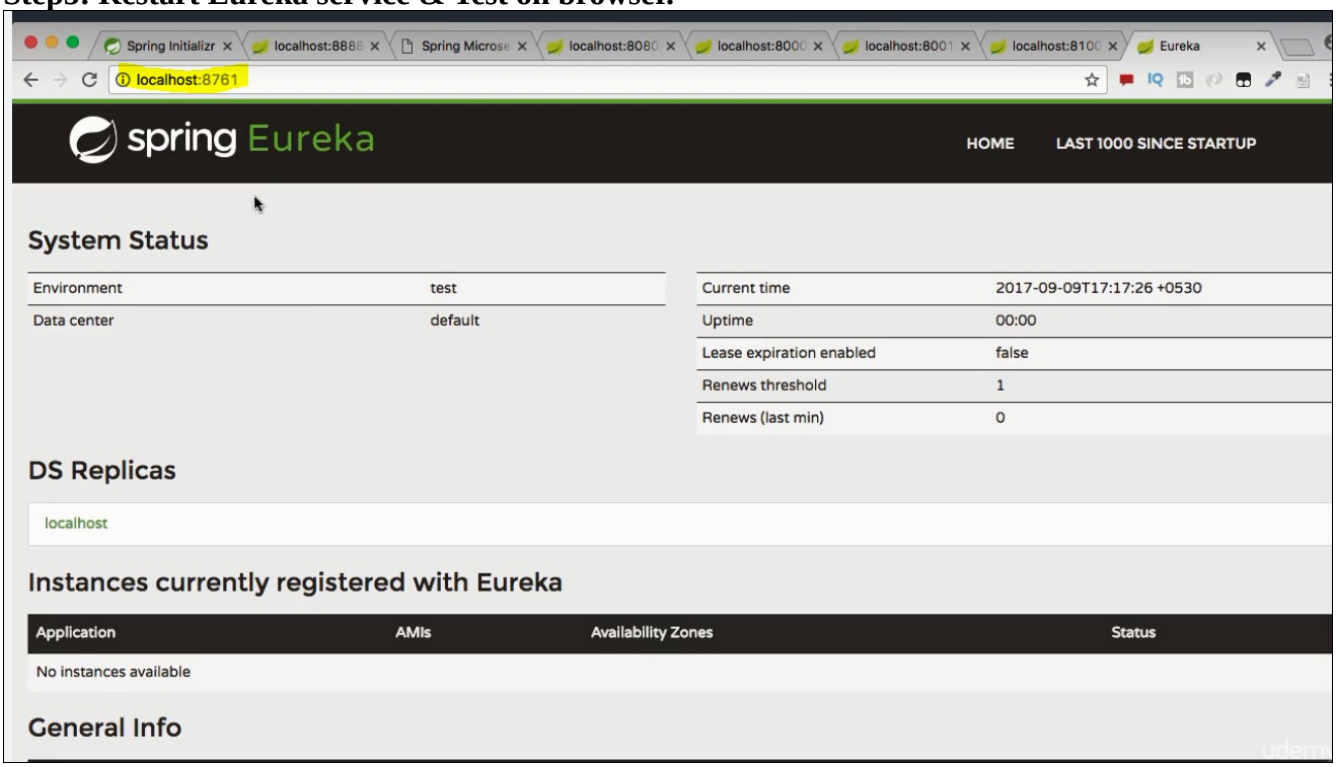
Step3: Enable EurekaServer in NetflixEurekaNamingServerApplication



Step4:

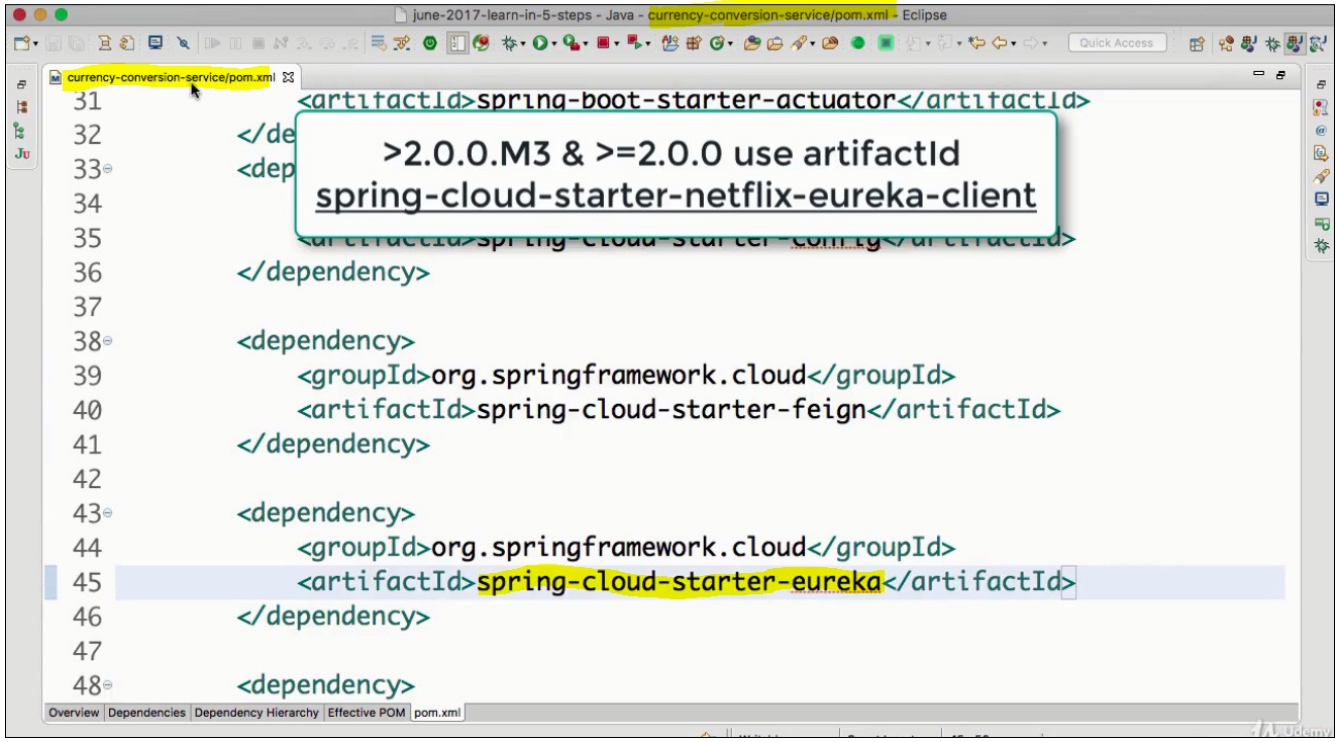


Step5: Restart Eureka service & Test on browser.

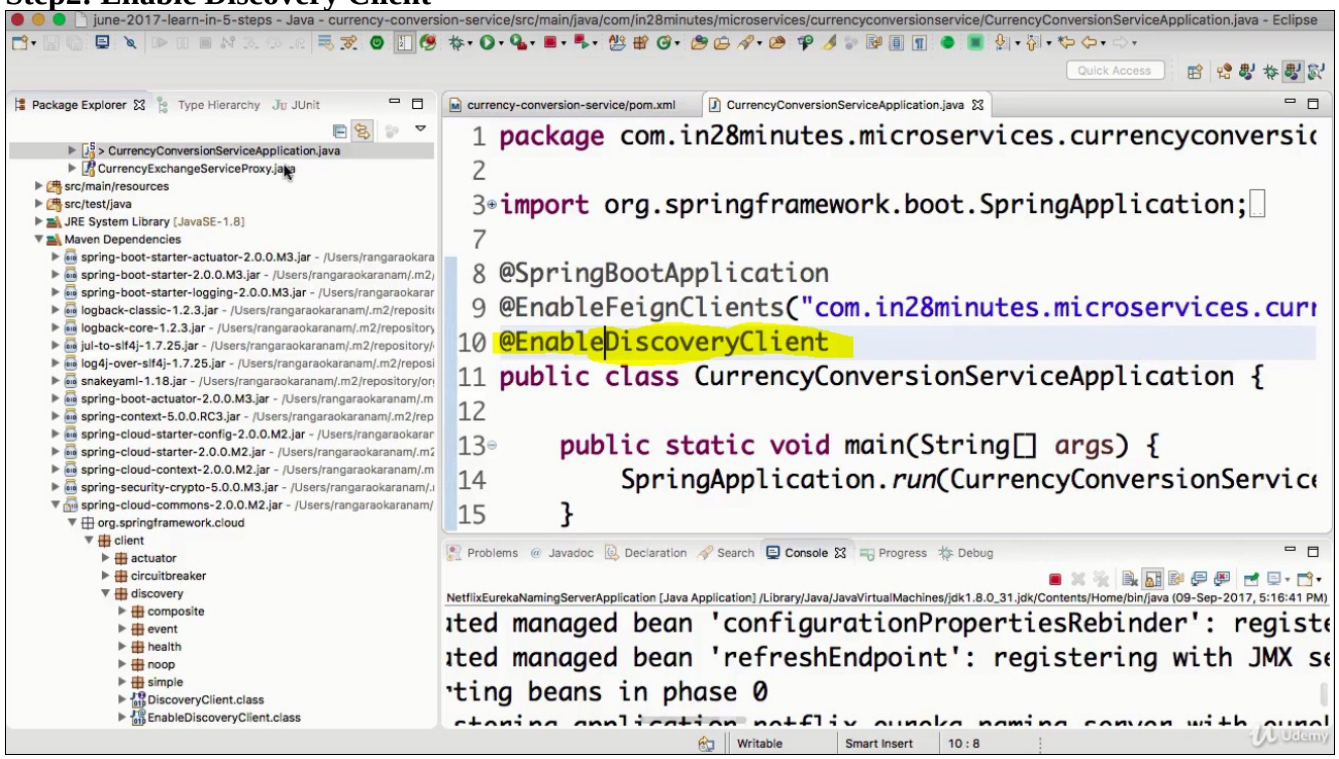


Lecture 83: Connect Currency Conversion Microservice to Eureka

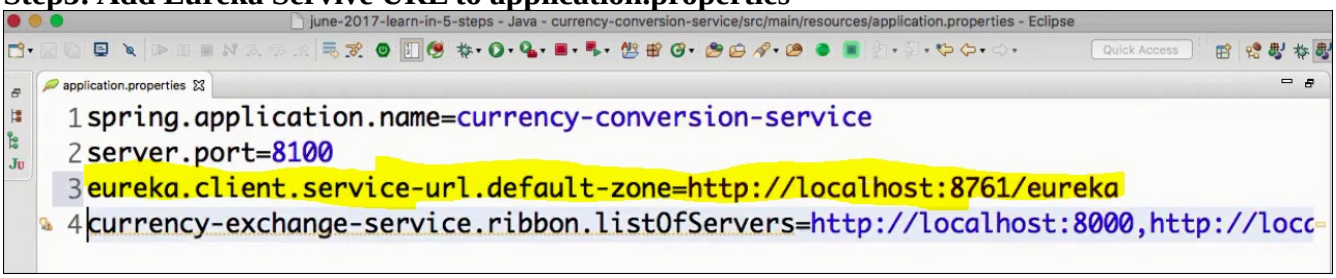
Step1: Add dependency in pom.xml of CCS service.



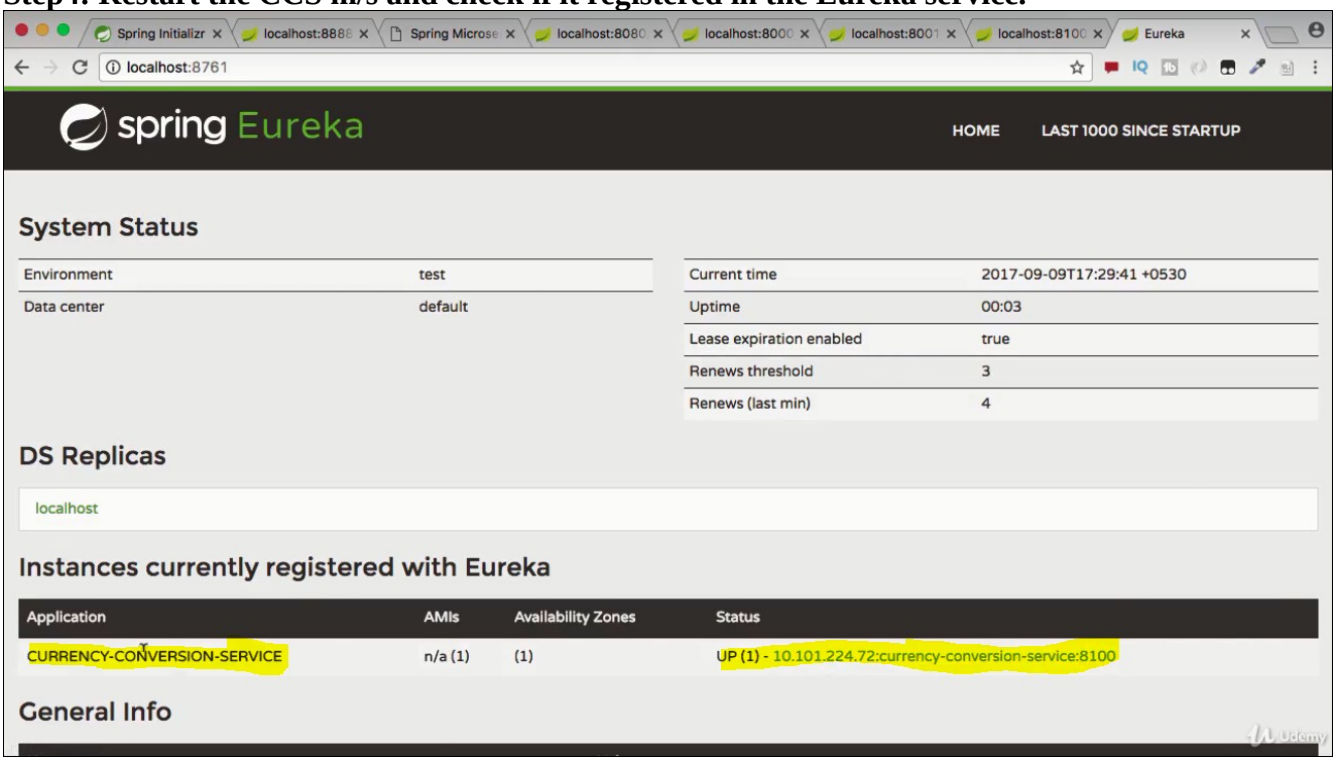
Step2: Enable Discovery Client



Step3: Add Eureka Servive URL to application.properties



Step4: Restart the CCS m/s and check if it registered in the Eureka service.



Similarly launch another service instance of CCS and check here .

Lecture 84: Connect Exchange Microservice to Eureka

Step1: Add dependency in POM.xml

```
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
```

>2.0.0.M3 & >=2.0.0 use artifactId
spring-cloud-starter-netflix-eureka-client

```
        <artifactId>spring-boot-starter-data-jpa</artifactId>
```

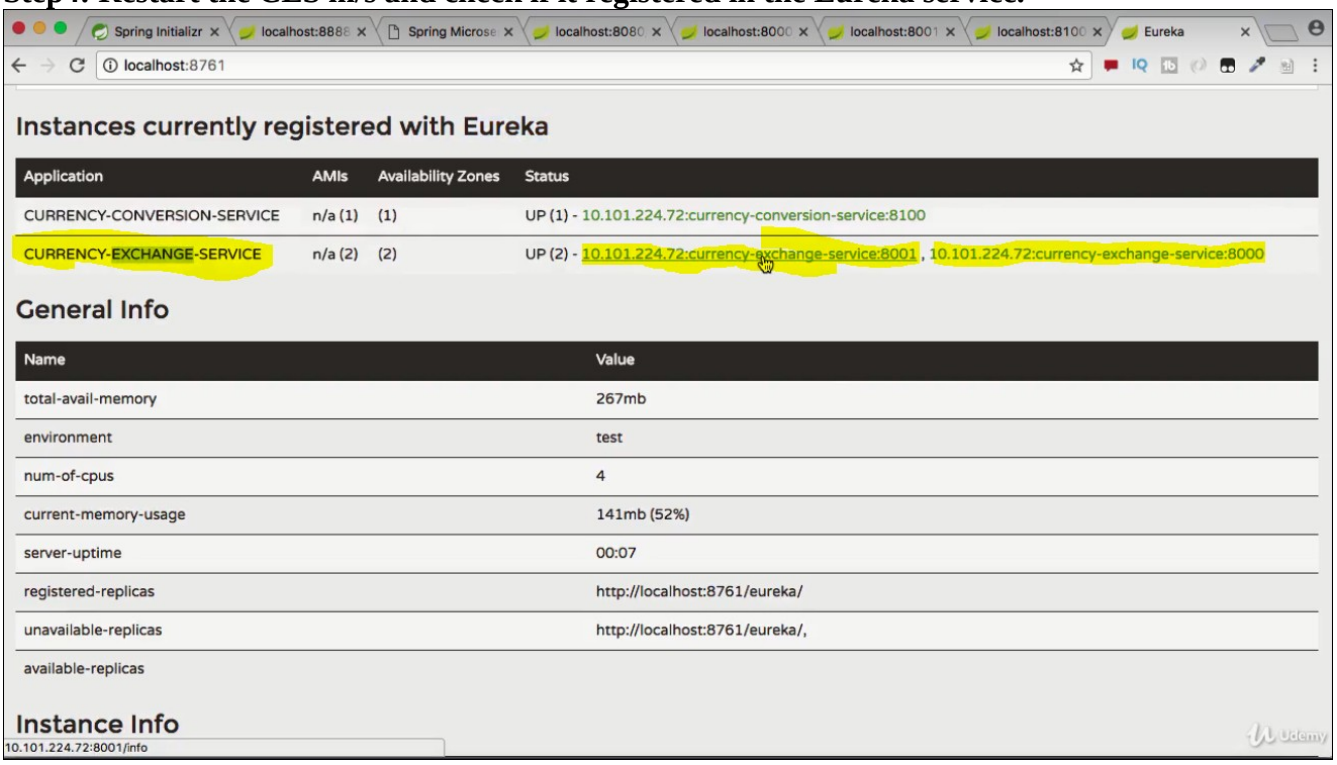
Step2: Enable Discovery Client

```
package com.in28minutes.microservices.currencyexchangeservice;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableDiscoveryClient
public class CurrencyExchangeServiceApplication {

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

Step3: Add Eureka Servive URL to application.properties

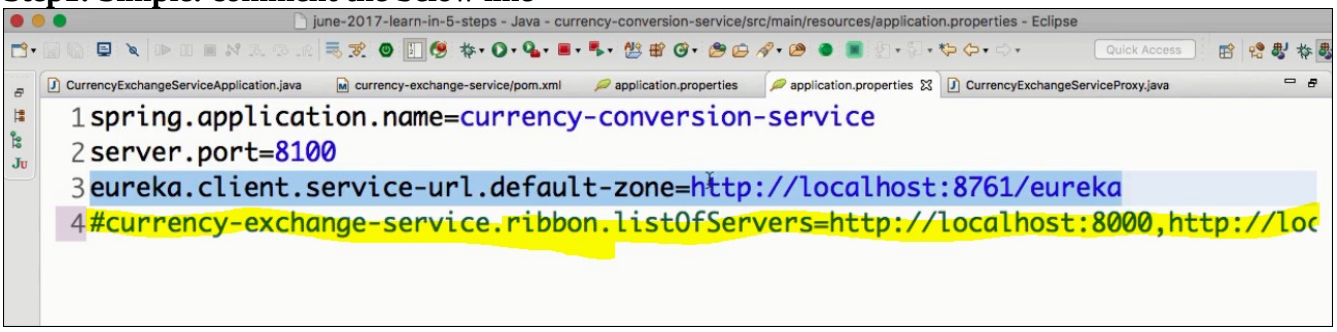
```
spring.application.name=currency-exchange-service
server.port=8000

spring.jpa.show-sql=true
spring.h2.console.enabled=true

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

Step4: Restart the CES m/s and check if it registered in the Eureka service.



Lecture 85: Distributing calls using Ribbon & Eureka

Step1: Simple: comment the below line

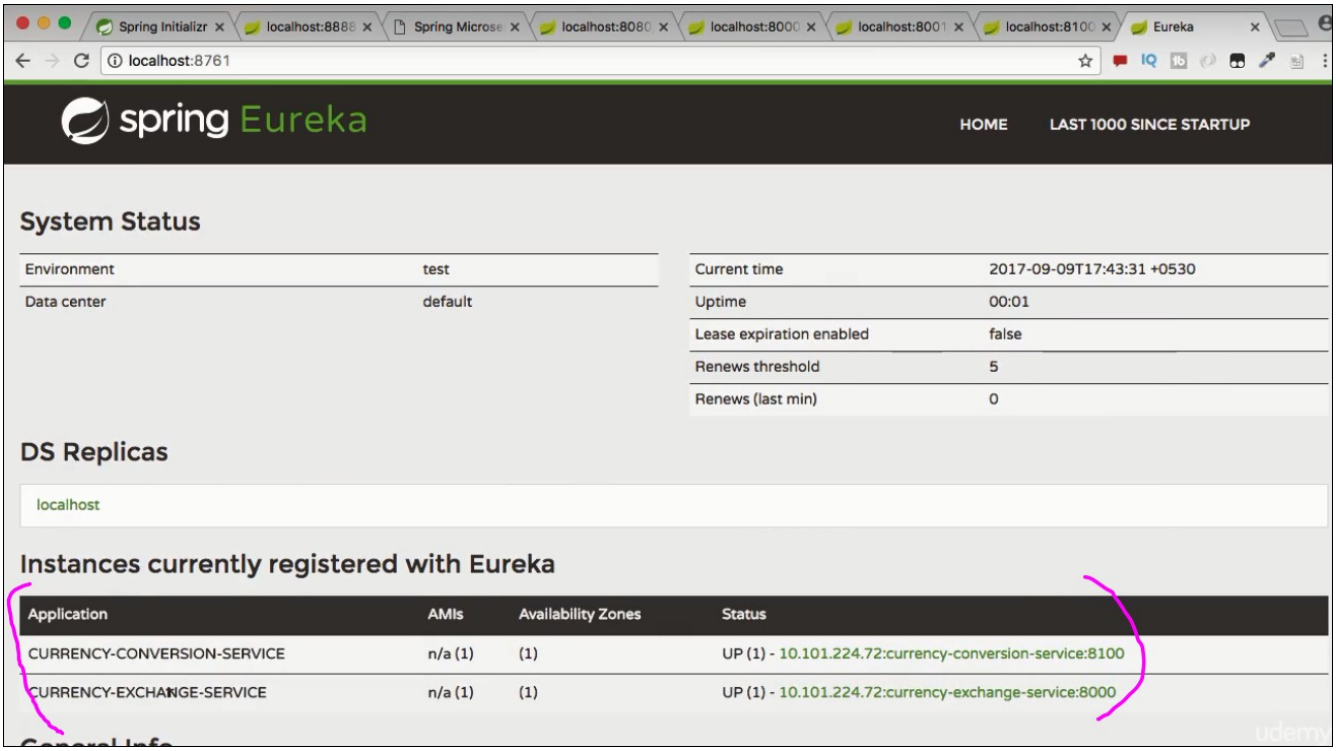


Step2: Kill all the service running

Step3: Start up the application in below sequence :

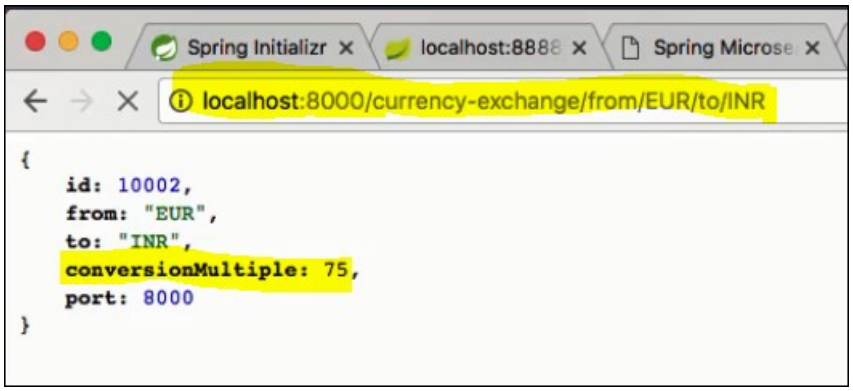
1. **Start the Eureka Naming Server on 8761**
2. **Start the CES m/s on 8000**
3. **Start the CSS m/s on 8100**

Step4: Verify the same on Eureka console

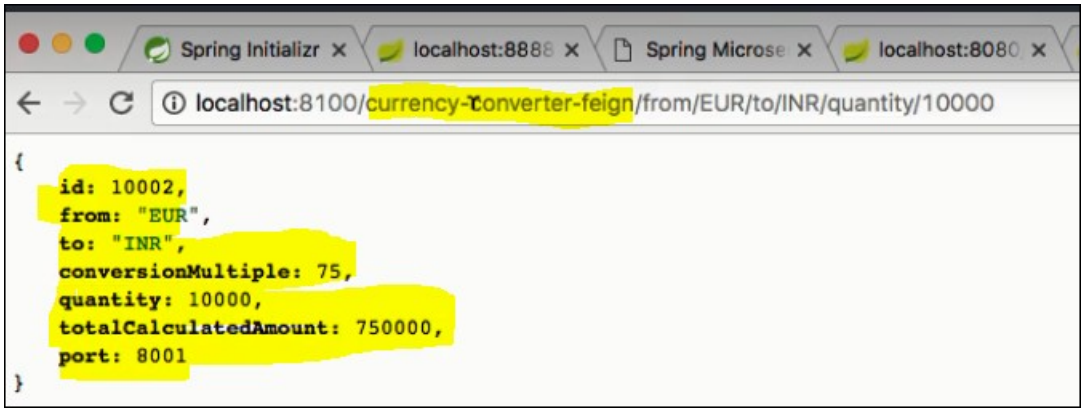


ALLOW 1-2 min for the Eureka to come up fully.

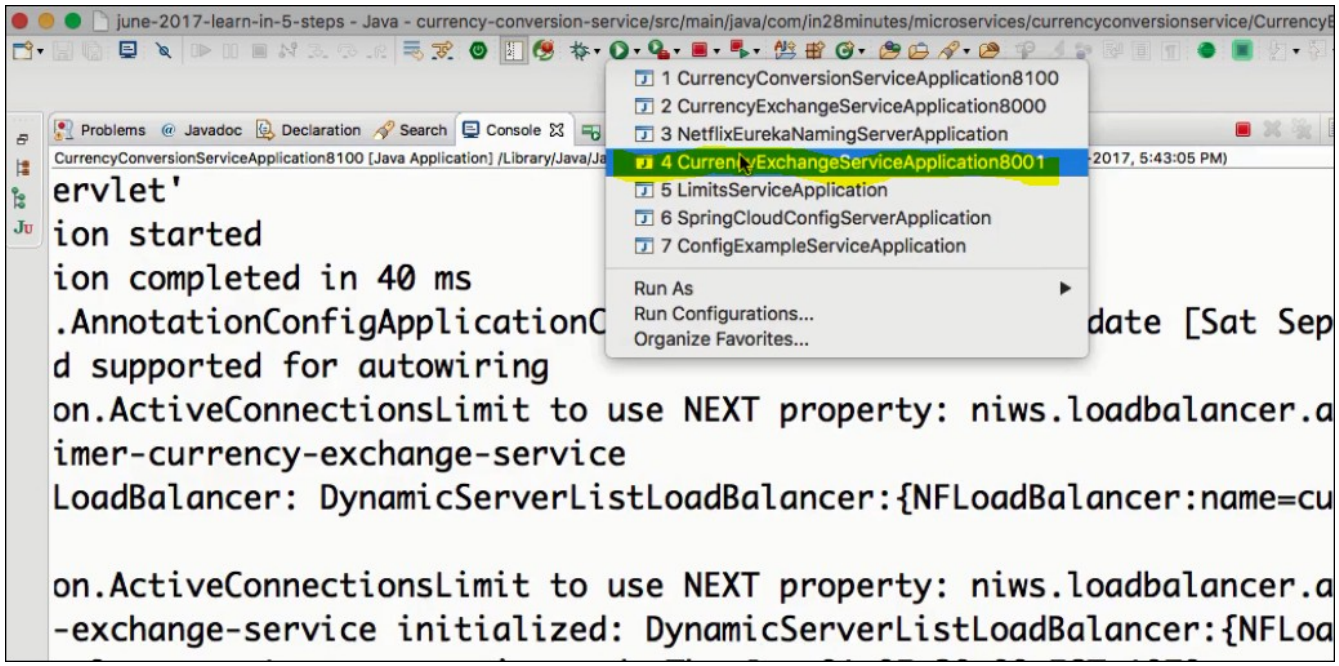
Step5: Test the CES & CCS m/s .
CES---------->OK



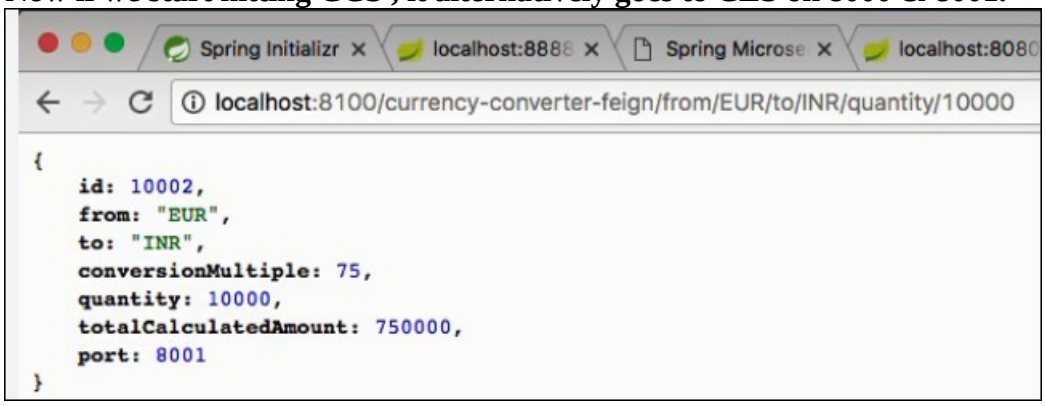
CCS ----------> OK



Step5: Scalup test
Start Start the CES m/s on 8001



Now if we start hitting CCS , it alternatively goes to CES on 8000 & 8001.



Step6:/Scaledown test
Now kill instance of CES on 8000 & invoke CSS , it will got to CES on 8001 automatically.