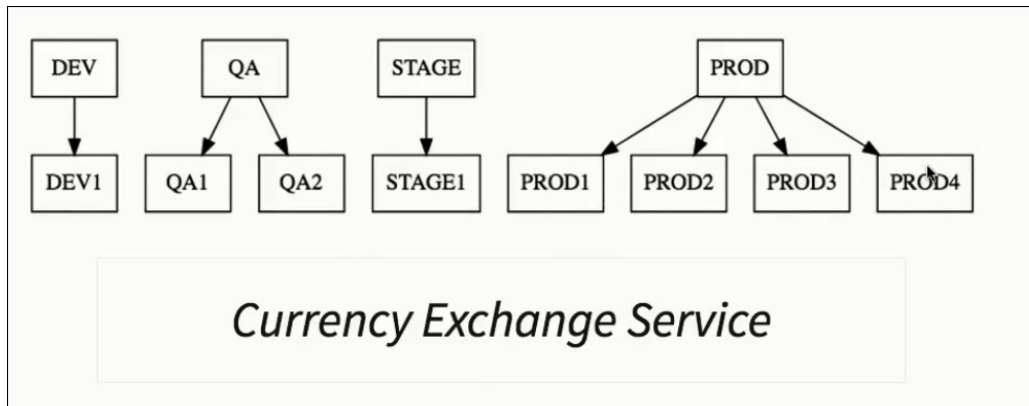
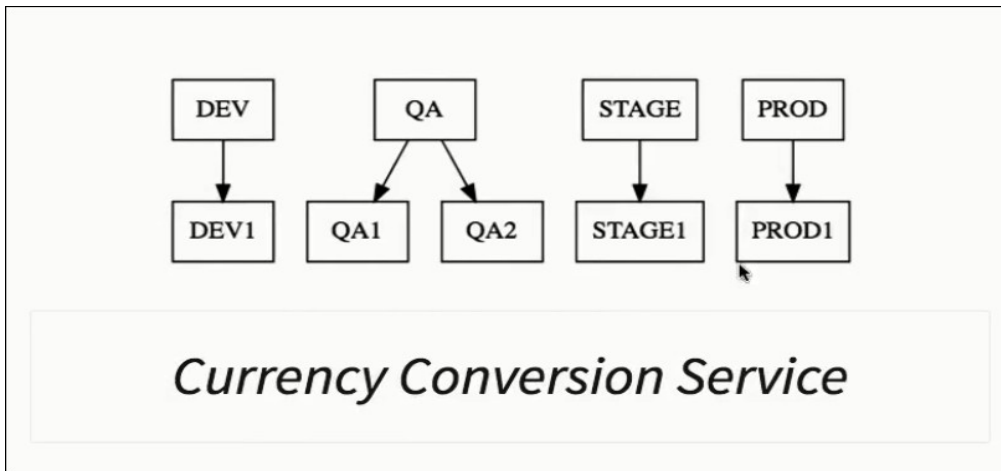
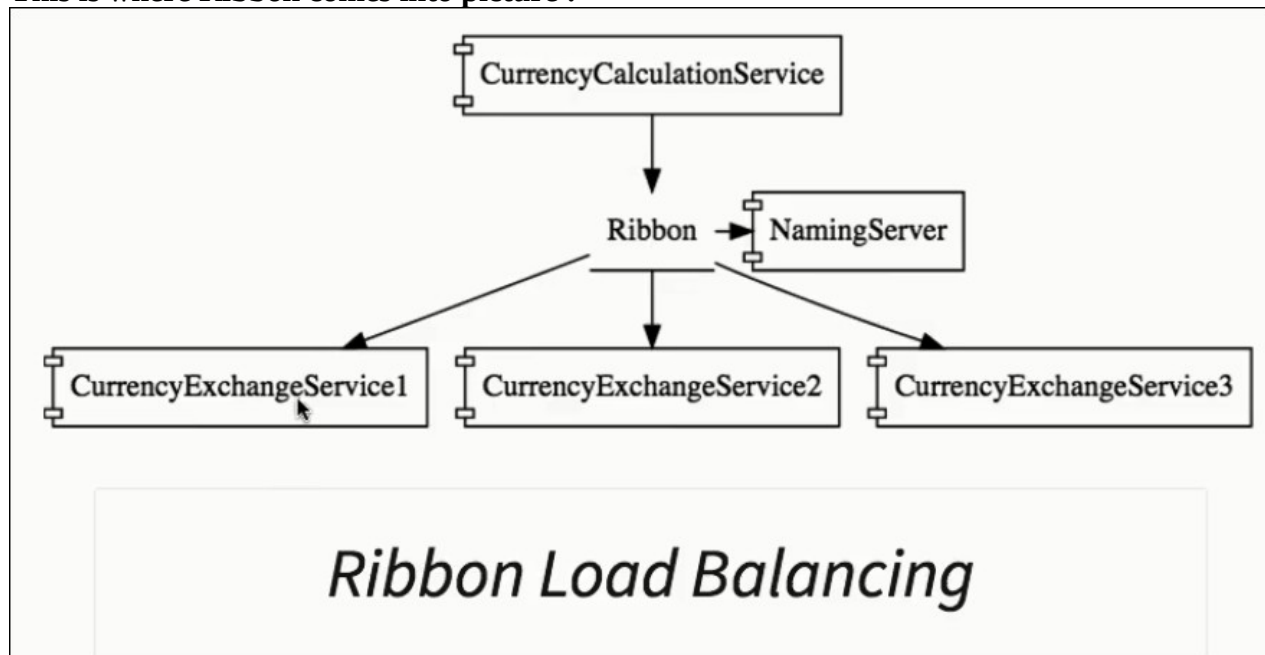


## Lecture 79,80: Client Side Load Balancing with Ribbon

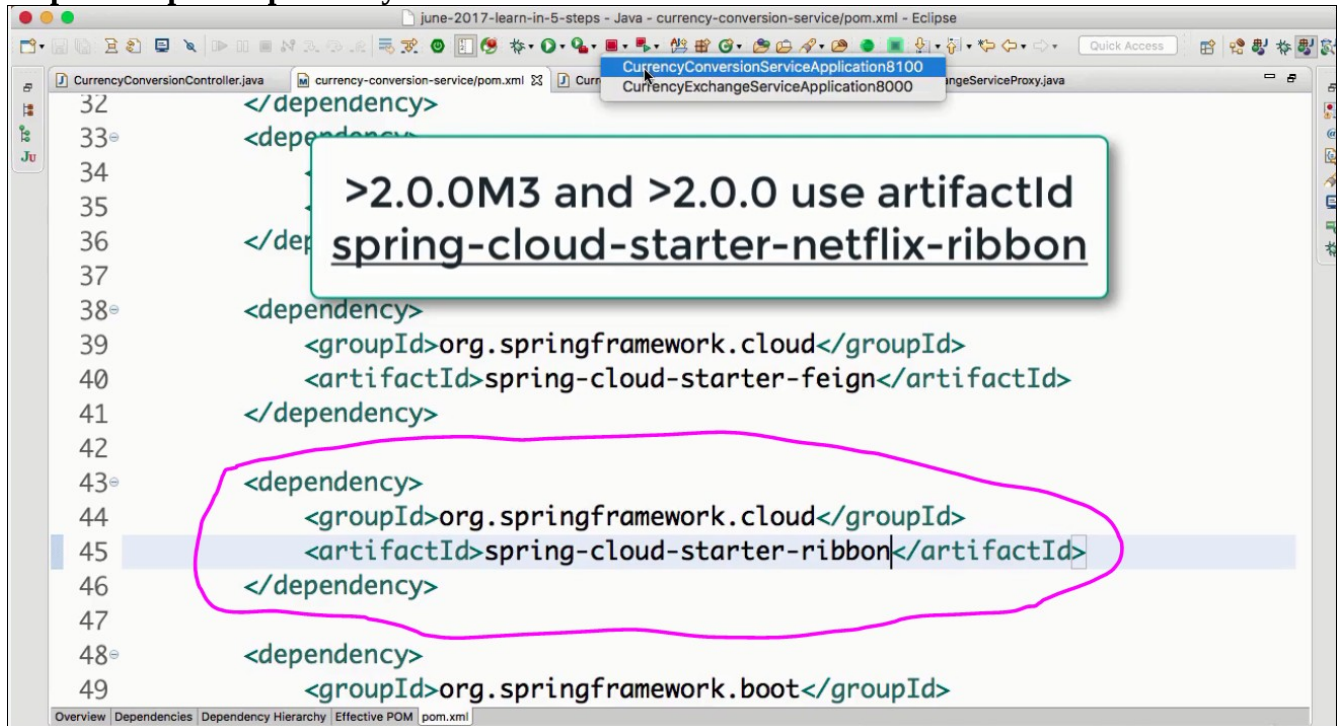
With previous approach , only CurrencyConversionService could only talk to any one instance of CurrencyExchangeService. With Ribbon implementation , we'll try to talk to any instance of CES.



This is where Ribbon comes into picture :

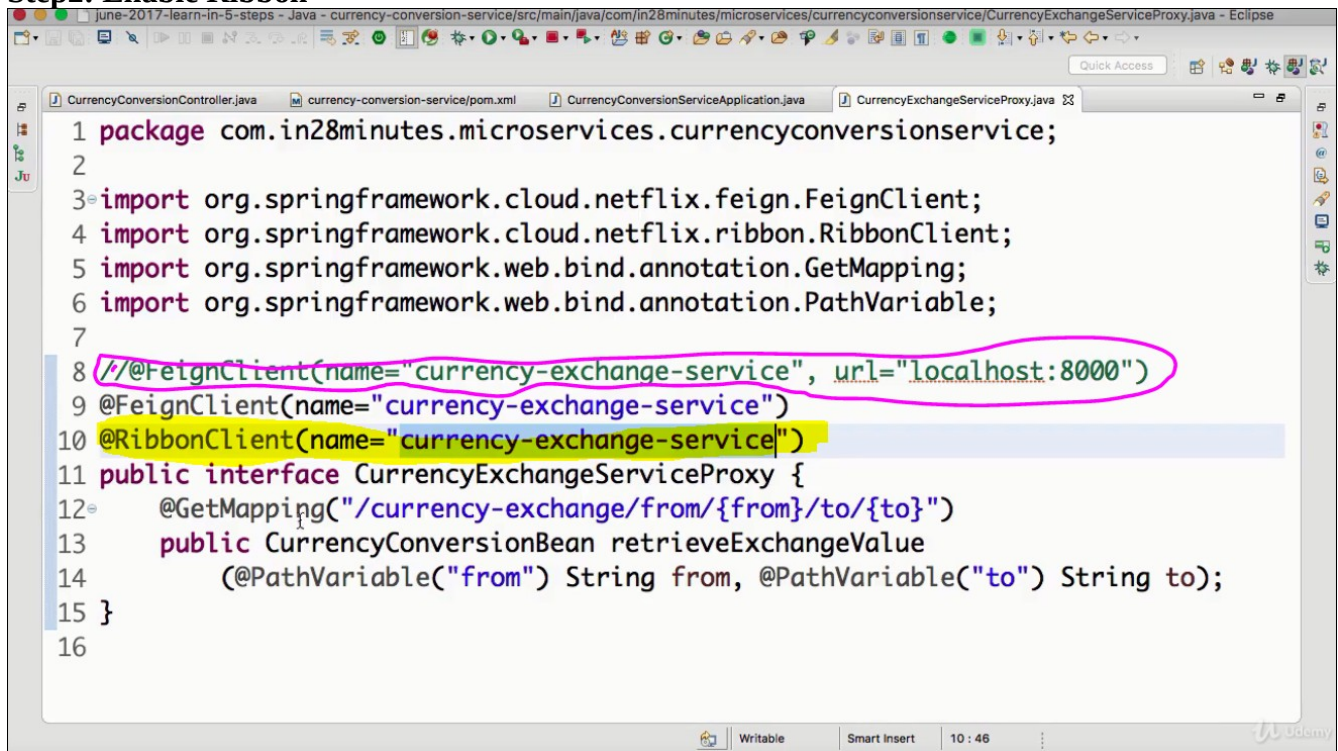


## Step1: Add pom dependency



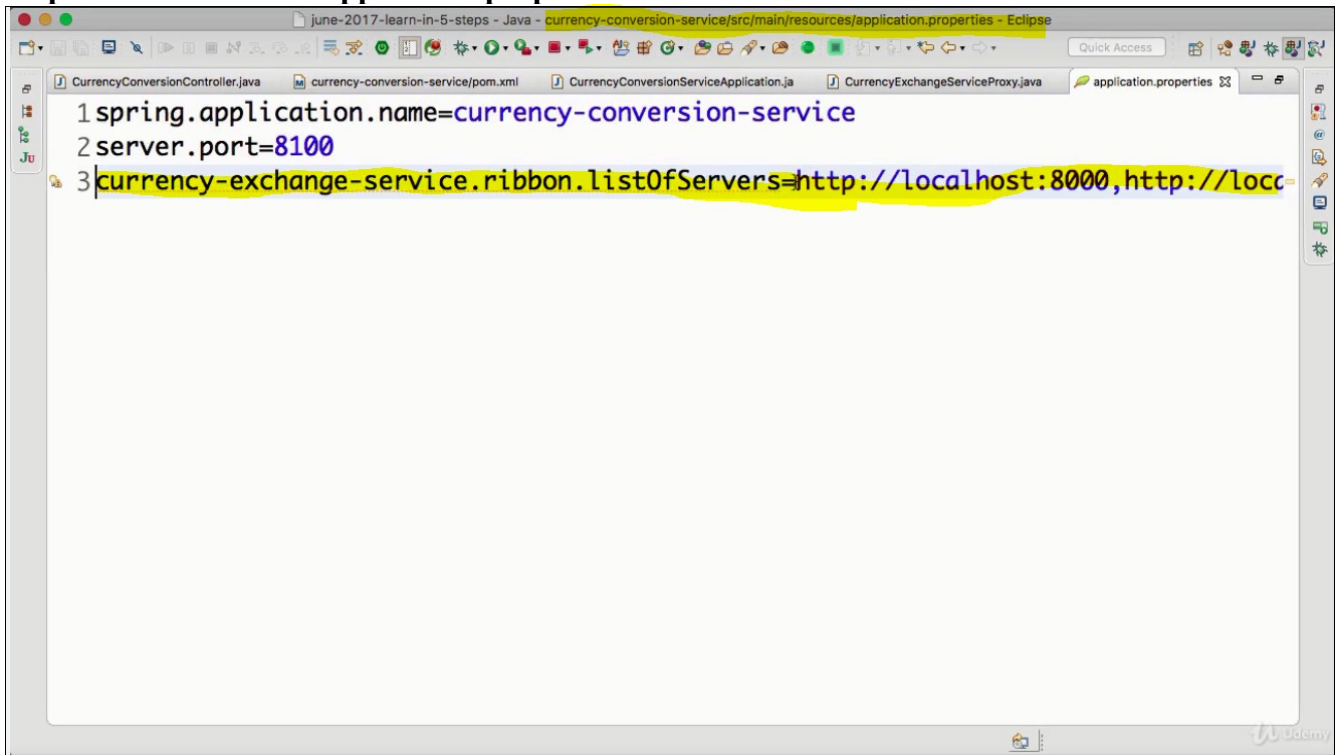
```
32 </dependency>
33 <dependency>
34     <groupId>org.springframework.cloud</groupId>
35     <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
36 </dependency>
37
38 <dependency>
39     <groupId>org.springframework.cloud</groupId>
40     <artifactId>spring-cloud-starter-feign</artifactId>
41 </dependency>
42
43 <dependency>
44     <groupId>org.springframework.cloud</groupId>
45     <artifactId>spring-cloud-starter-ribbon</artifactId>
46 </dependency>
47
48 <dependency>
49     <groupId>org.springframework.boot</groupId>
```

## Step2: Enable Ribbon



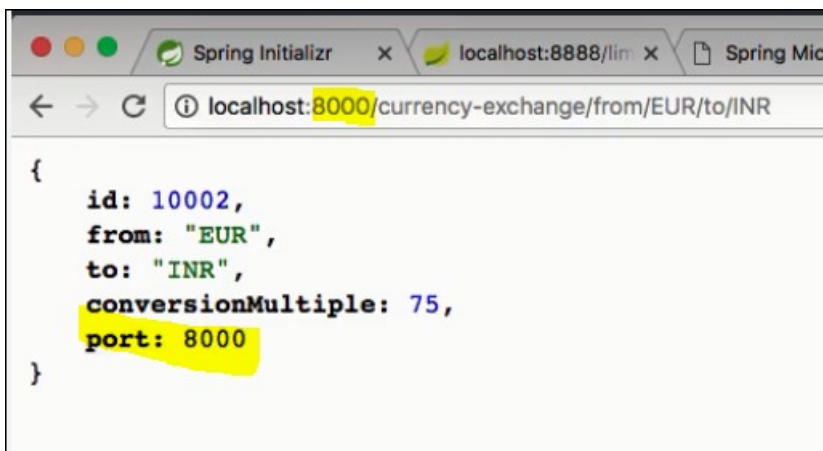
```
1 package com.in28minutes.microservices.currencyconversionservice;
2
3 import org.springframework.cloud.netflix.feign.FeignClient;
4 import org.springframework.cloud.netflix.ribbon.RibbonClient;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 // @FeignClient(name="currency-exchange-service", url="localhost:8000")
9 @FeignClient(name="currency-exchange-service")
10 @RibbonClient(name="currency-exchange-service")
11 public interface CurrencyExchangeServiceProxy {
12     @GetMapping("/currency-exchange/from/{from}/to/{to}")
13     public CurrencyConversionBean retrieveExchangeValue
14         (@PathVariable("from") String from, @PathVariable("to") String to);
15 }
16
```

### Step3: Add URL in the application.properties

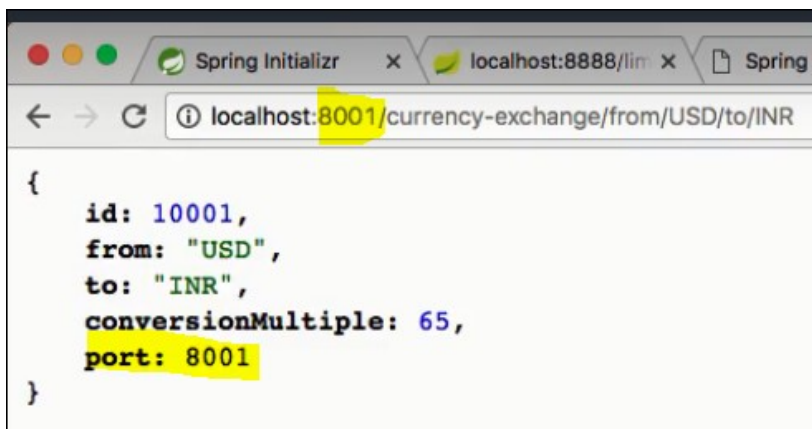
A screenshot of the Eclipse IDE interface. The 'application.properties' file is open in the editor. It contains three lines of configuration: 1. 'spring.application.name=currency-conversion-service', 2. 'server.port=8100', and 3. 'currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001'. The third line is highlighted in yellow. The IDE's toolbar and project explorer are visible at the top and left respectively.

```
1spring.application.name=currency-conversion-service
2server.port=8100
3currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001
```

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

A screenshot of a web browser window. The address bar shows 'localhost:8000/currency-exchange/from/EUR/to/INR'. The page displays a JSON response with the following fields: 'id: 10002', 'from: "EUR"', 'to: "INR"', 'conversionMultiple: 75', and 'port: 8000'. The 'port: 8000' field is highlighted in yellow.

```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  port: 8000
}
```

A screenshot of a web browser window. The address bar shows 'localhost:8001/currency-exchange/from/USD/to/INR'. The page displays a JSON response with the following fields: 'id: 10001', 'from: "USD"', 'to: "INR"', 'conversionMultiple: 65', and 'port: 8001'. The 'port: 8001' field is highlighted in yellow.

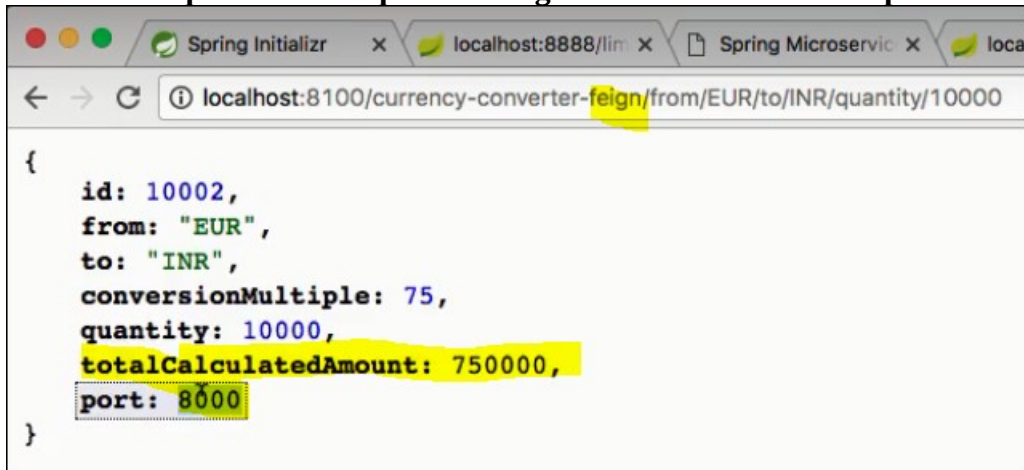
```
{
  id: 10001,
  from: "USD",
  to: "INR",
  conversionMultiple: 65,
  port: 8001
}
```

### Step5: Invoke currency concersion service & see the results:



```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  port: 8001
}
```

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



```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  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 be able to manage

Ans: No. CES property needs to be changed

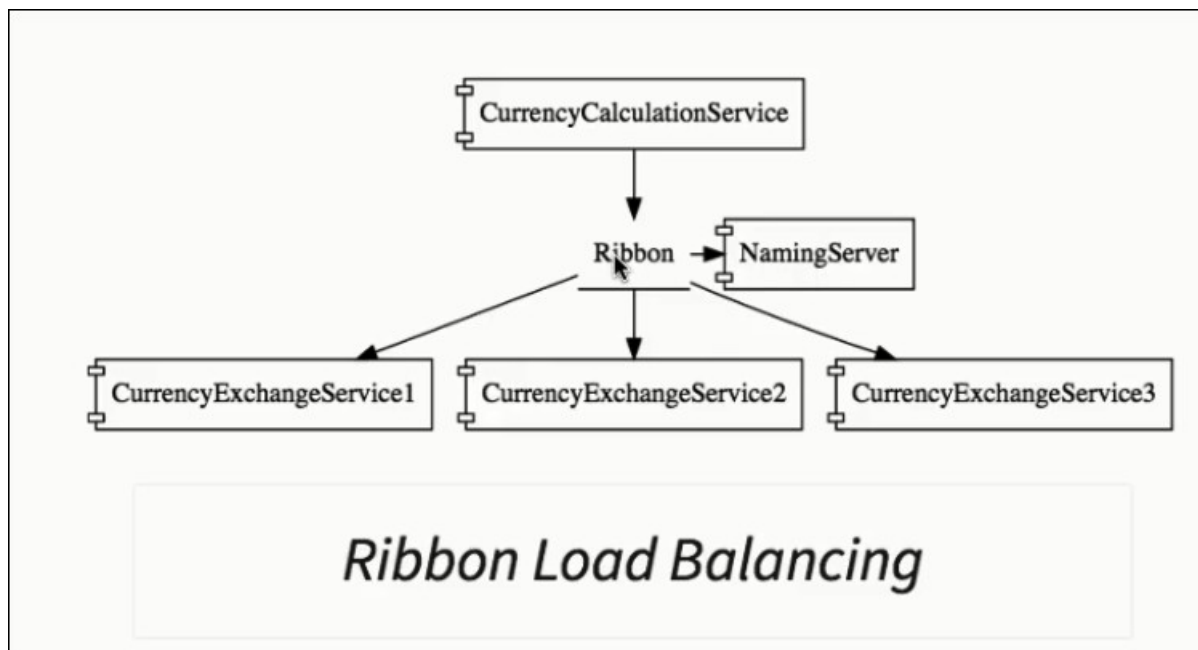
Q: What will happen when one of the instance of CES service is added? Will ribbon be able to manage

Ans: No. CES property needs to be 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

The screenshot shows the Spring Initializr web application at [start.spring.io](https://start.spring.io). The page is titled "SPRING INITIALIZR bootstrap your application now".

**Generate a** **Maven Project** **with** **Java** **and Spring Boot** **2.0.0 M3**

**Project Metadata**

Artifact coordinates

Group: **com.in28minutes.microservices**

Artifact: **netflix-eureka-naming-server**

**Dependencies**

Add Spring Boot Starters and dependencies to your application

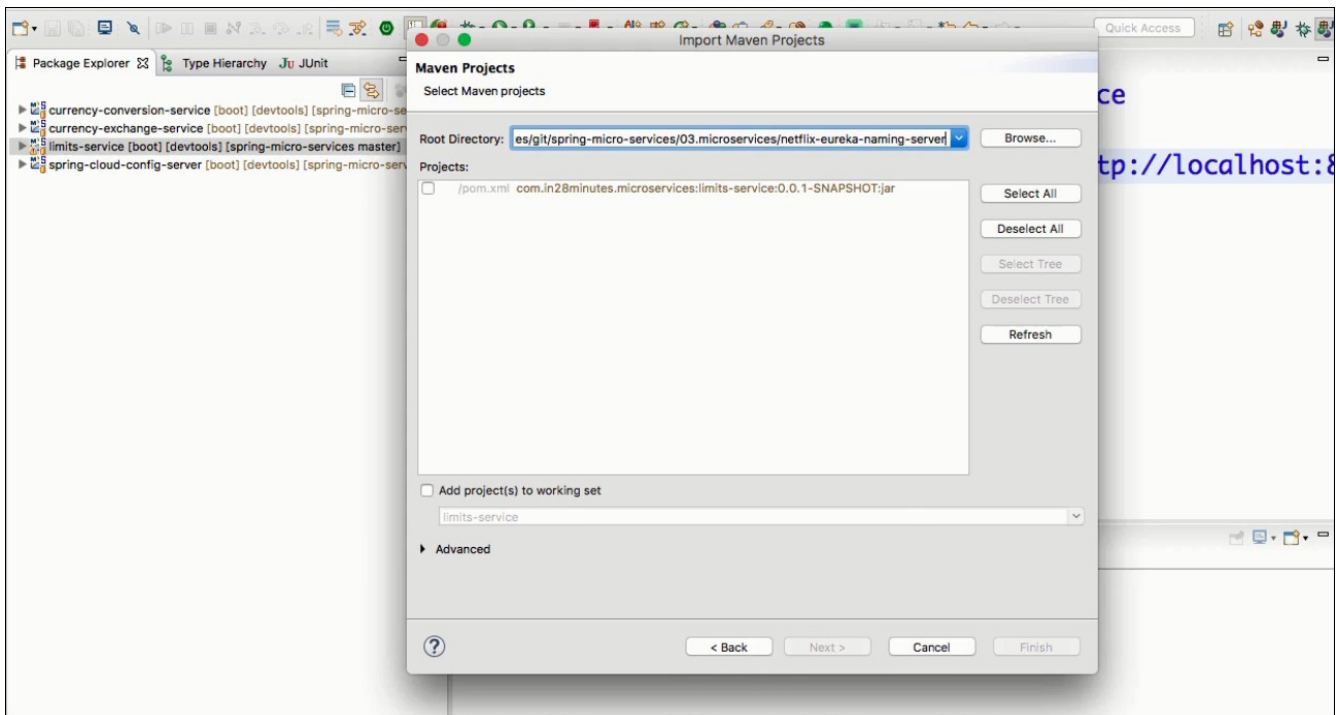
Search for dependencies: Web, Security, JPA, Actuator, DevTools...

Selected Dependencies: **Eureka Server**, **Config Client**, **Actuator**, **DevTools**

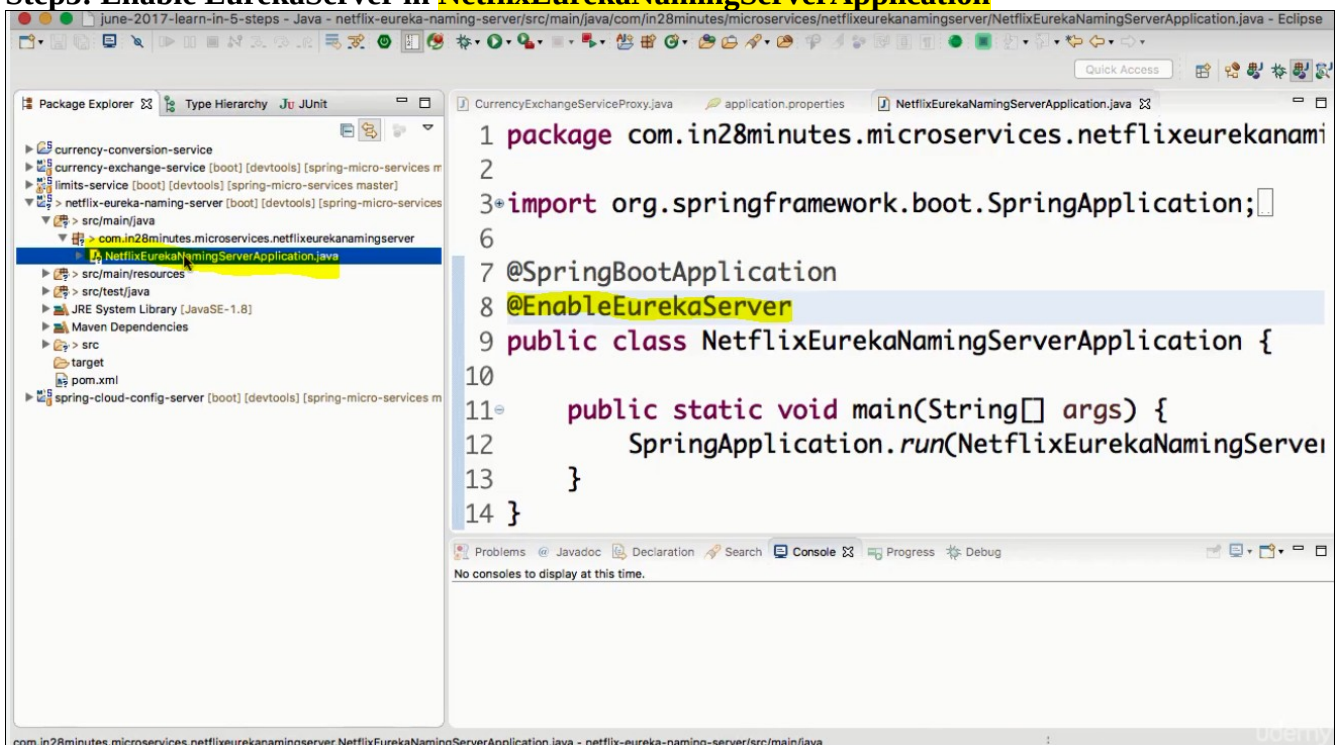
**Generate Project**

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

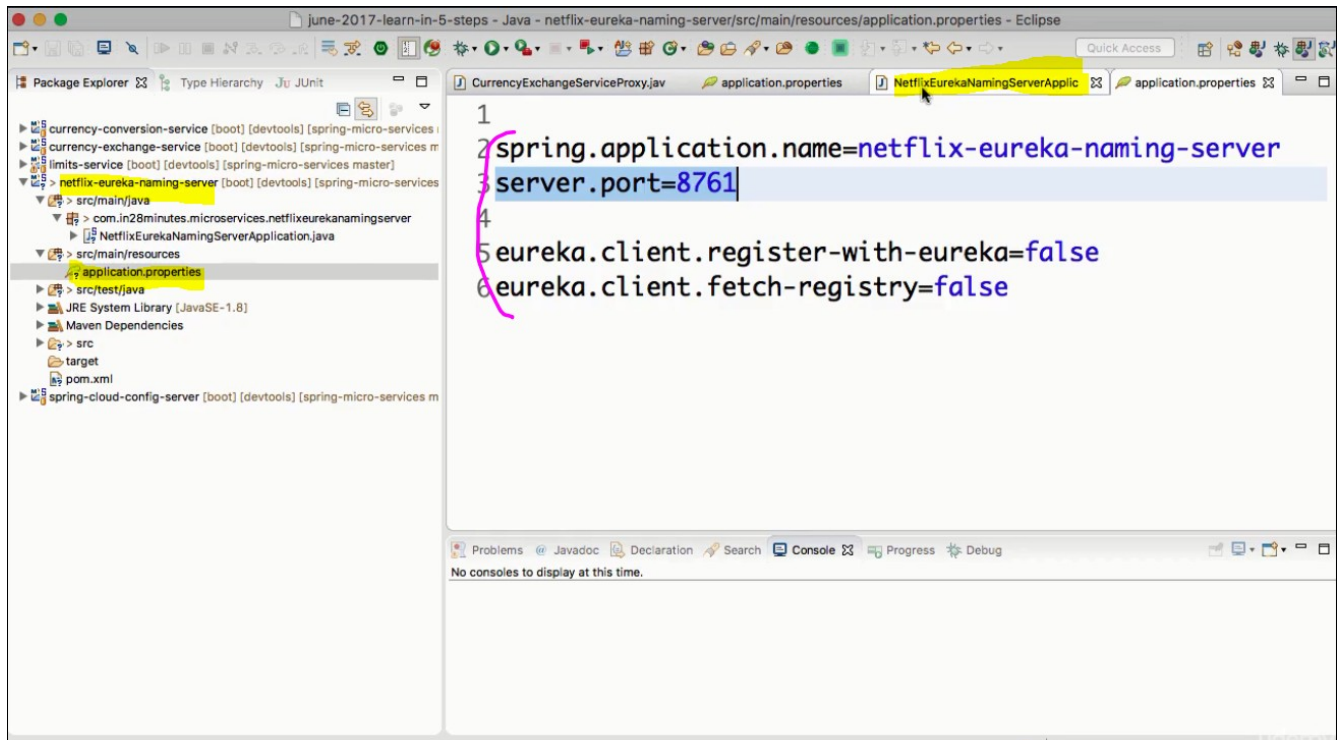
### Step2: Download and import it to workspace as maven project



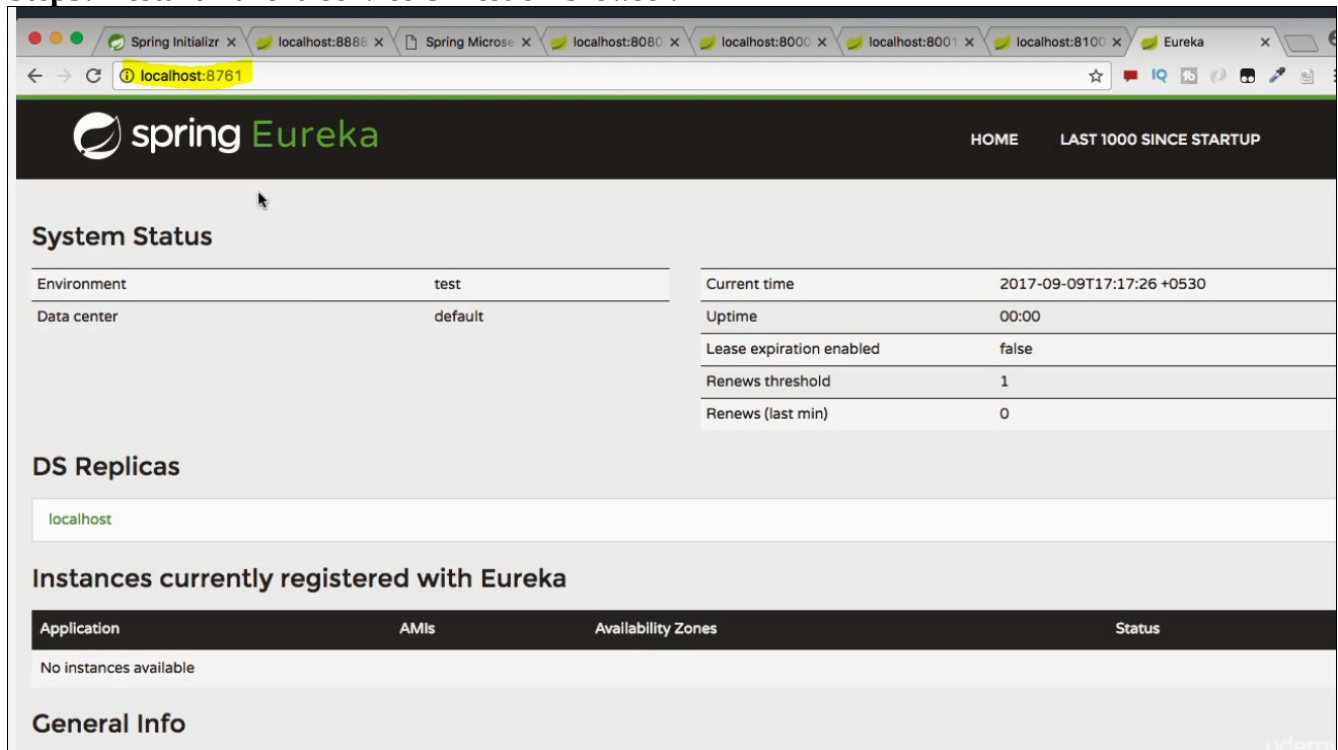
### Step3: Enable EurekaServer in NetflixEurekaNamingServerApplication



### Step4:



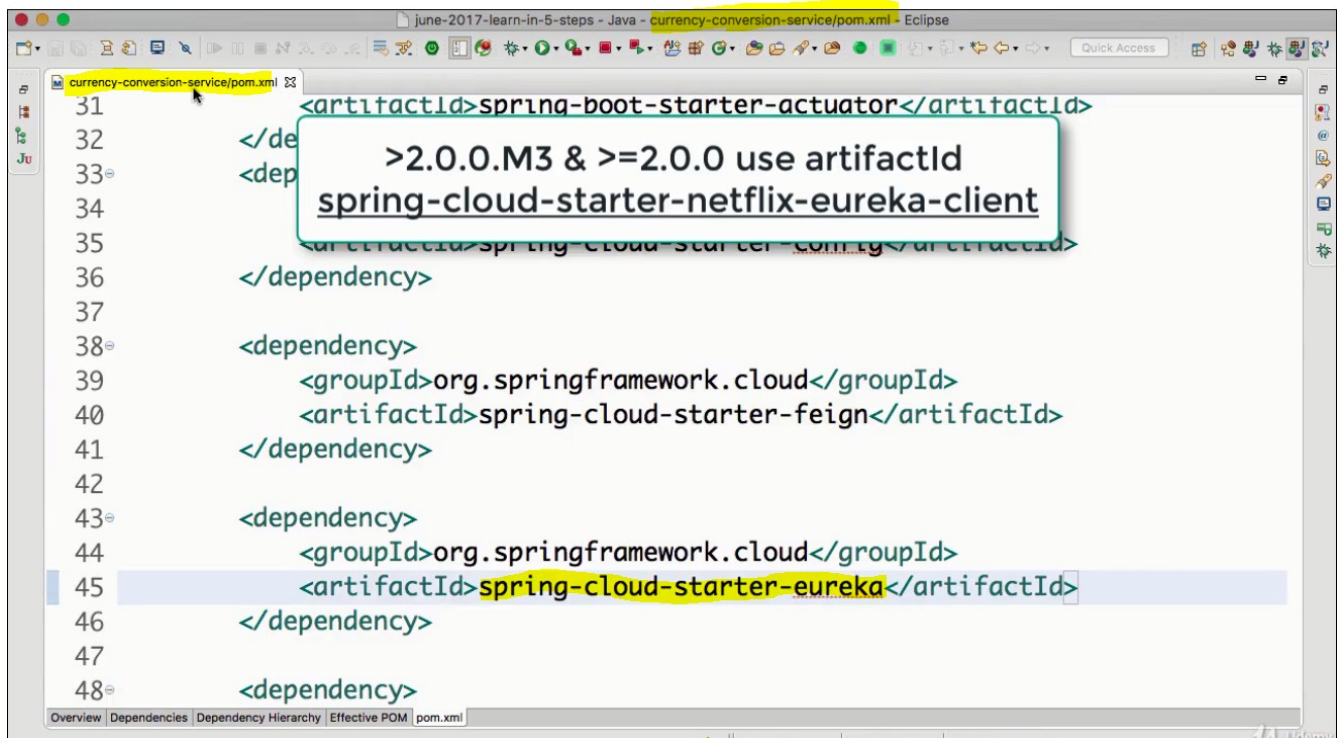
## Step5: Restart Eureka service & Test on browser.





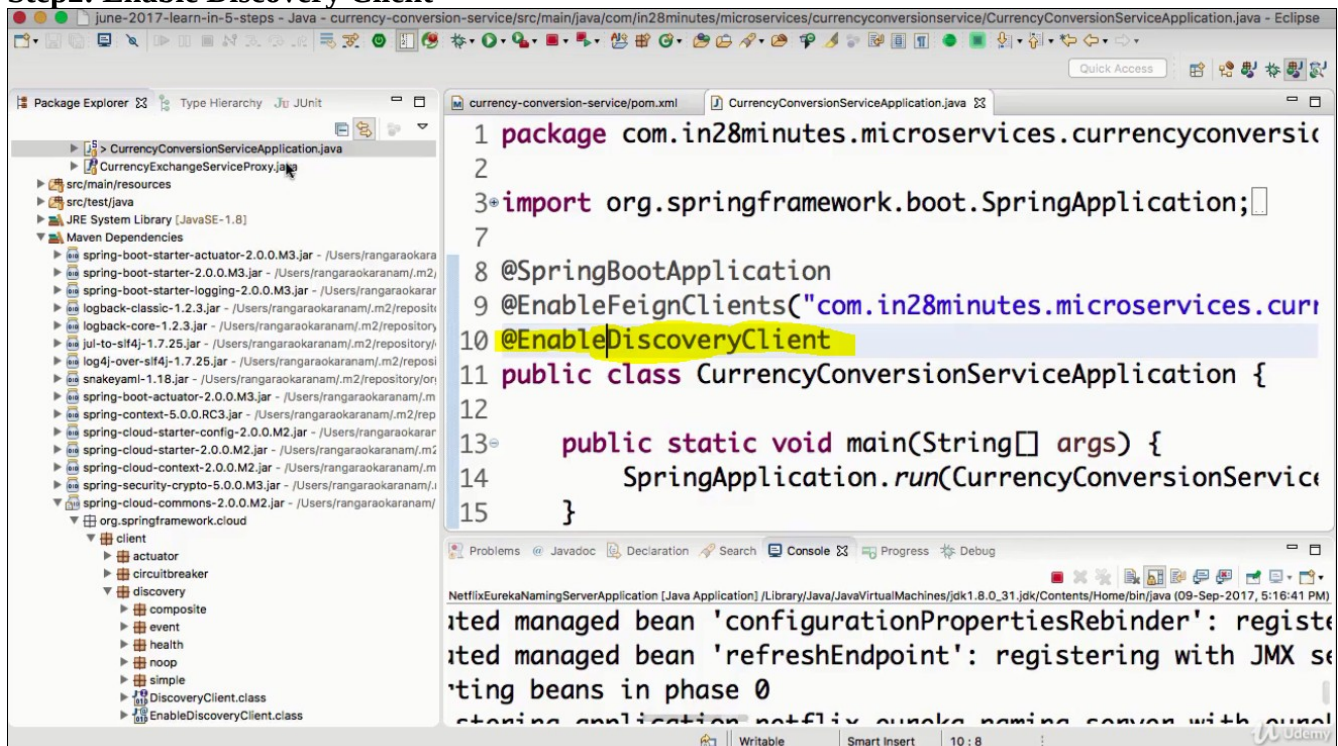
## Lecture 83: Connect Currency Conversion Microservice to Eureka

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



```
31 <artifactId>spring-boot-starter-actuator</artifactId>
32 </dependency>
33 <dependency>
34 <groupId>org.springframework.cloud</groupId>
35 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
36 </dependency>
37
38 <dependency>
39 <groupId>org.springframework.cloud</groupId>
40 <artifactId>spring-cloud-starter-feign</artifactId>
41 </dependency>
42
43 <dependency>
44 <groupId>org.springframework.cloud</groupId>
45 <artifactId>spring-cloud-starter-eureka</artifactId>
46 </dependency>
47
48 <dependency>
```

### Step2: Enable Discovery Client



```
1 package com.in28minutes.microservices.currencyconversion;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7
8 @SpringBootApplication
9 @EnableFeignClients("com.in28minutes.microservices.currencyconversion")
10 @EnableDiscoveryClient
11 public class CurrencyConversionServiceApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(CurrencyConversionServiceApplication.class, args);
15     }
16 }
```



### Step3: Add Eureka Service URL to application.properties

```
1 spring.application.name=currency-conversion-service
2 server.port=8100
3 eureka.client.service-url.default-zone=http://localhost:8761/eureka
4 currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001,http://localhost:8002
```

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

The screenshot shows the Spring Eureka web interface in a browser. The address bar shows 'localhost:8761'. The page has a dark header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into sections: 'System Status', 'DS Replicas', 'Instances currently registered with Eureka', and 'General Info'.

**System Status**

Environment	test	Current time	2017-09-09T17:29:41 +0530
Data center	default	Uptime	00:03
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	4

**DS Replicas**

localhost

**Instances currently registered with Eureka**

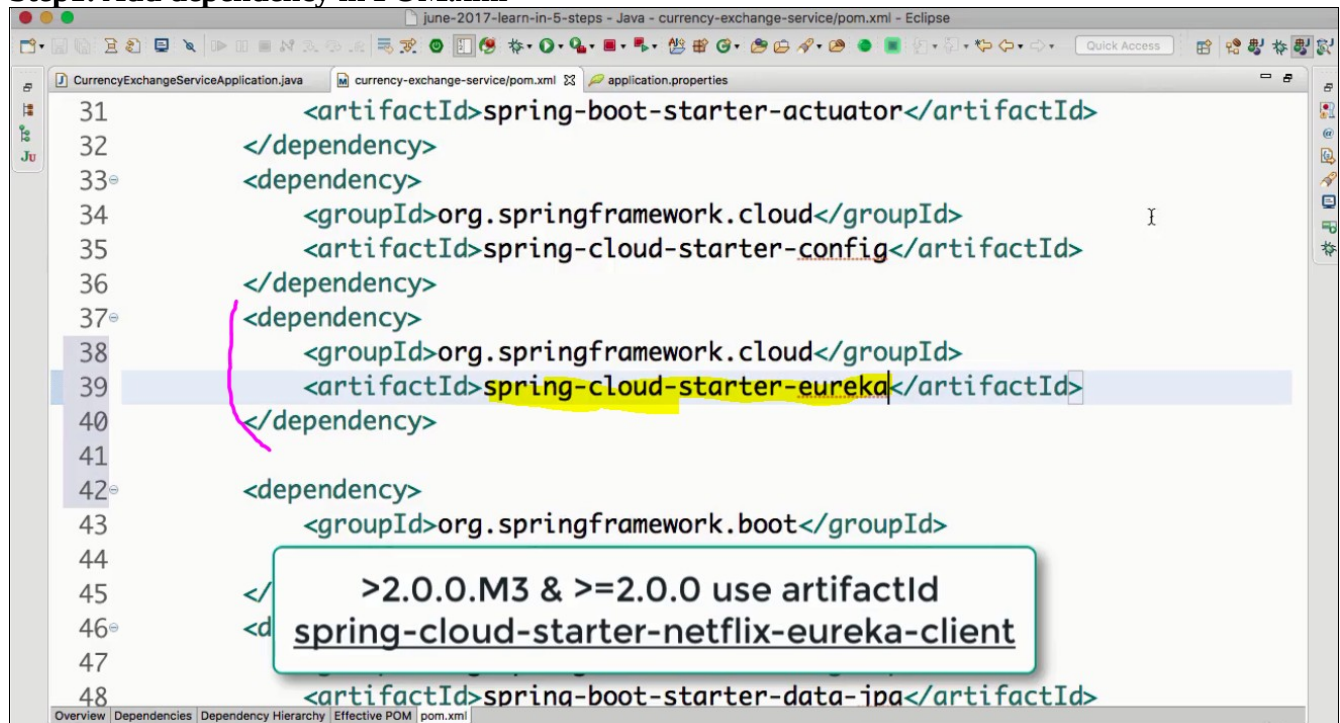
Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION-SERVICE	n/a (1)	(1)	UP (1) - 10.101.224.72:currency-conversion-service:8100

**General Info**

Similarly launch another service instance of CCS and check here .

## Lecture 84: Connect Exchange Microservice to Eureka

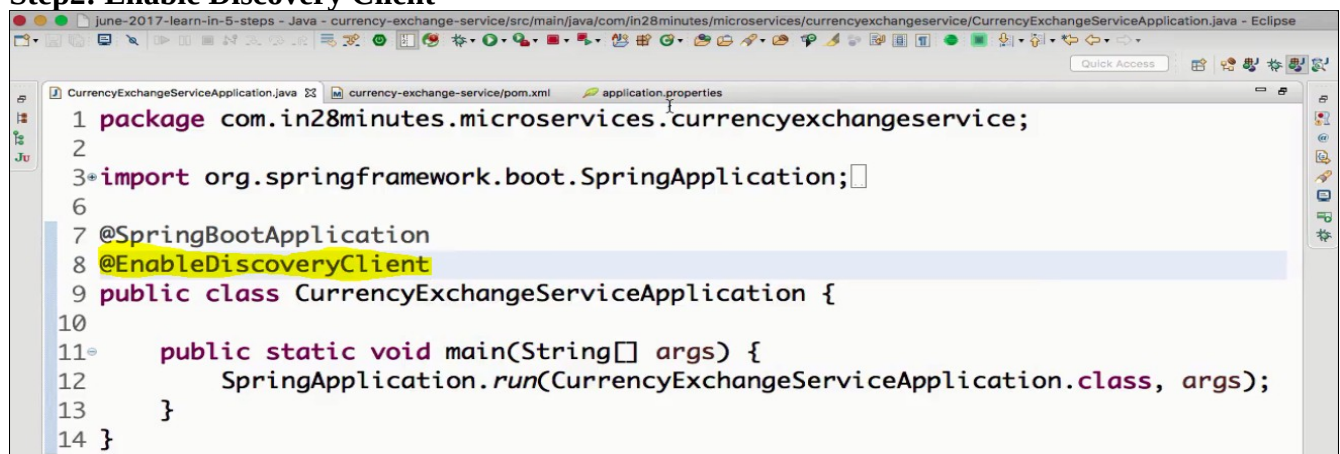
### Step1: Add dependency in POM.xml



```
31     <artifactId>spring-boot-starter-actuator</artifactId>
32 </dependency>
33 <dependency>
34     <groupId>org.springframework.cloud</groupId>
35     <artifactId>spring-cloud-starter-config</artifactId>
36 </dependency>
37 <dependency>
38     <groupId>org.springframework.cloud</groupId>
39     <artifactId>spring-cloud-starter-eureka</artifactId>
40 </dependency>
41
42 <dependency>
43     <groupId>org.springframework.boot</groupId>
44     <artifactId>spring-boot-starter-data-jpa</artifactId>
45 </dependency>
46 <dependency>
47     <groupId>org.springframework.cloud</groupId>
48     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
49 </dependency>
50 <dependency>
51     <groupId>org.springframework.boot</groupId>
52     <artifactId>spring-boot-starter-data-jpa</artifactId>
53 </dependency>
```

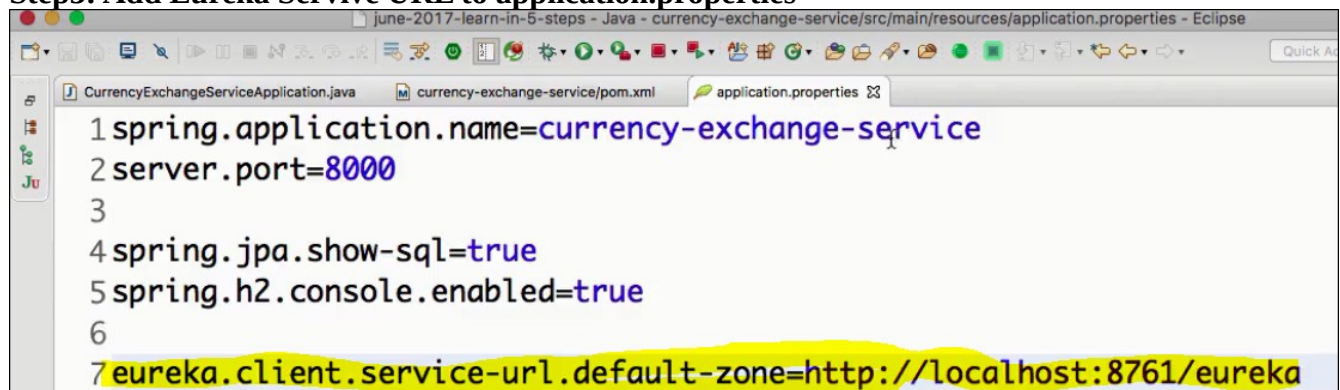
>2.0.0.M3 & >=2.0.0 use artifactId  
spring-cloud-starter-netflix-eureka-client

### Step2: Enable Discovery Client



```
1 package com.in28minutes.microservices.currencyexchangeservice;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableDiscoveryClient
9 public class CurrencyExchangeServiceApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(CurrencyExchangeServiceApplication.class, args);
13     }
14 }
```

### Step3: Add Eureka Service URL to application.properties



```
1 spring.application.name=currency-exchange-service
2 server.port=8000
3
4 spring.jpa.show-sql=true
5 spring.h2.console.enabled=true
6
7 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.**

The screenshot shows the Eureka service console in a web browser. The address bar shows 'localhost:8761'. The main heading is 'Instances currently registered with Eureka'. Below this is a table with columns: Application, AMIs, Availability Zones, and Status. The table lists two instances: 'CURRENCY-CONVERSION-SERVICE' and 'CURRENCY-EXCHANGE-SERVICE'. The 'CURRENCY-EXCHANGE-SERVICE' row is highlighted in yellow. Below the table is a 'General Info' section with a table of system metrics. At the bottom, there is an 'Instance Info' section.

Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION-SERVICE	n/a (1)	(1)	UP (1) - 10.101.224.72:currency-conversion-service:8100
CURRENCY-EXCHANGE-SERVICE	n/a (2)	(2)	UP (2) - 10.101.224.72:currency-exchange-service:8001, 10.101.224.72:currency-exchange-service:8000

Name	Value
total-avail-memory	267mb
environment	test
num-of-cpus	4
current-memory-usage	141mb (52%)
server-uptime	00:07
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/
available-replicas	

**Instance Info**  
10.101.224.72:8001/info

## Lecture 85: Distributing calls using Ribbon & Eureka

**Step1: Simple: comment the below line**

The screenshot shows the Eclipse IDE with the 'application.properties' file open. The file contains four lines of configuration. The fourth line, which sets the ribbon list of servers for the currency-exchange-service, is highlighted in yellow and has a comment symbol at the beginning, indicating it is commented out.

```
1 spring.application.name=currency-conversion-service
2 server.port=8100
3 eureka.client.service-url.default-zone=http://localhost:8761/eureka
4 #currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001
```

**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**

The screenshot shows the Spring Eureka dashboard. The top navigation bar includes the Spring Eureka logo and links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into two sections: System Status and DS Replicas.

**System Status**

Environment	test	Current time	2017-09-09T17:43:31 +0530
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

**DS Replicas**

localhost

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION-SERVICE	n/a (1)	(1)	UP (1) - 10.101.224.72:currency-conversion-service:8100
CURRENCY-EXCHANGE-SERVICE	n/a (1)	(1)	UP (1) - 10.101.224.72:currency-exchange-service:8000

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

Step5: Test the CES & CCS m/s .

CES----->OK

The screenshot shows a web browser with the URL `localhost:8000/currency-exchange/from/EUR/to/INR`. The response is a JSON object:

```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  port: 8000
}
```

CCS -----> OK

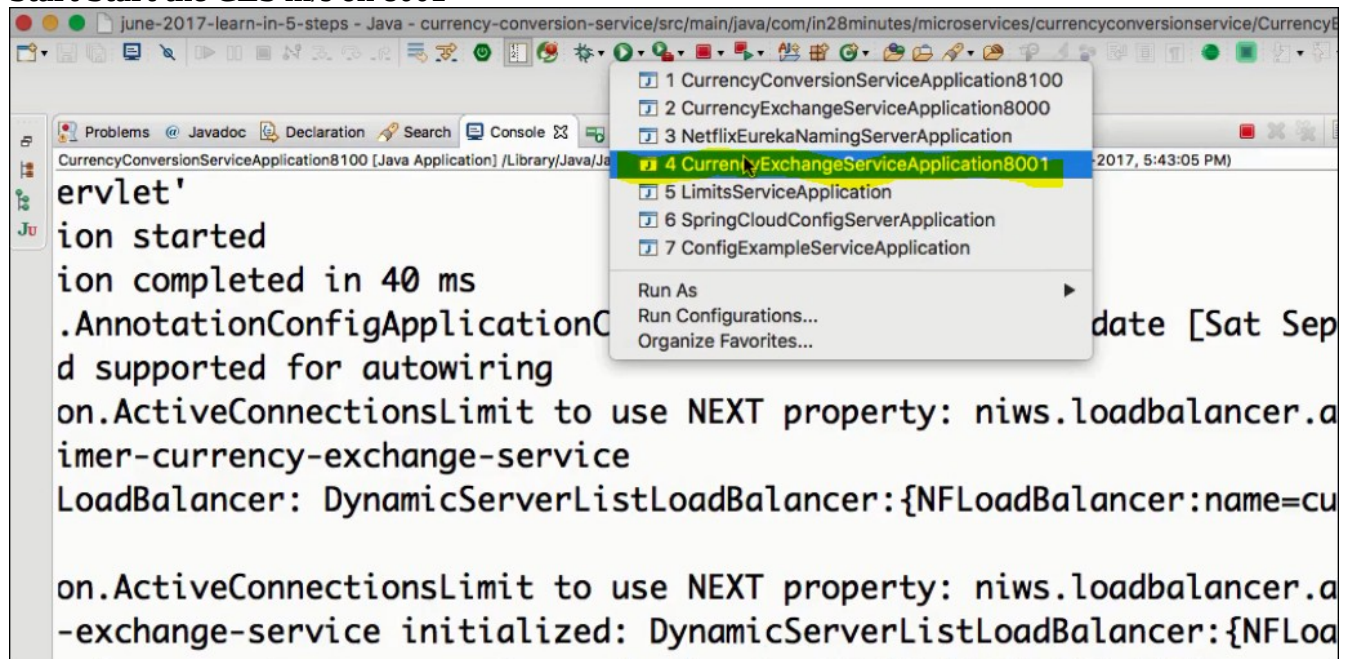
The screenshot shows a web browser with the URL `localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000`. The response is a JSON object:

```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  port: 8001
}
```

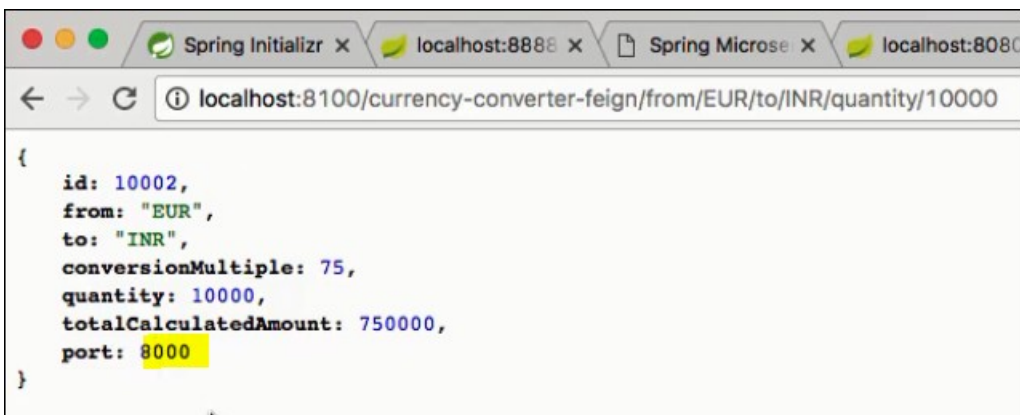
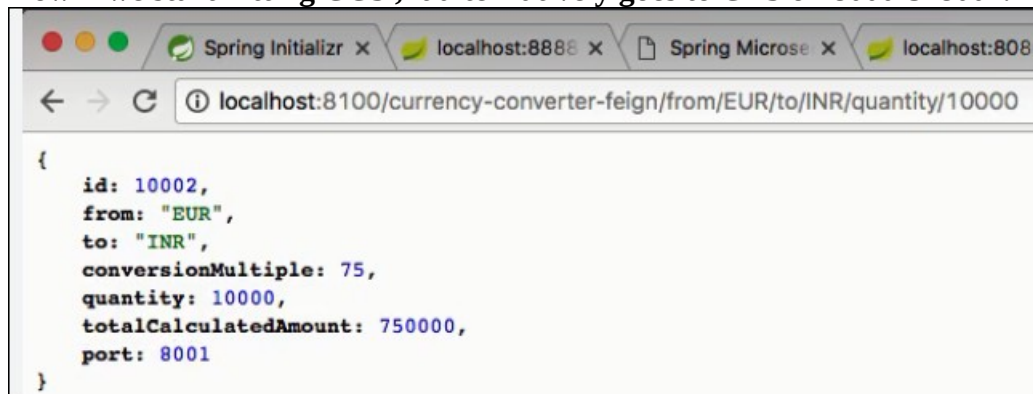


## Step5: Scalup test

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.