

## Lecture87,88,89,90: Implementing API GATEWAY

Why is API Gateway needed :

# API GATEWAYS

- Authentication, authorization and security
- Rate Limits
- Fault Tolerance
- Service Aggregation

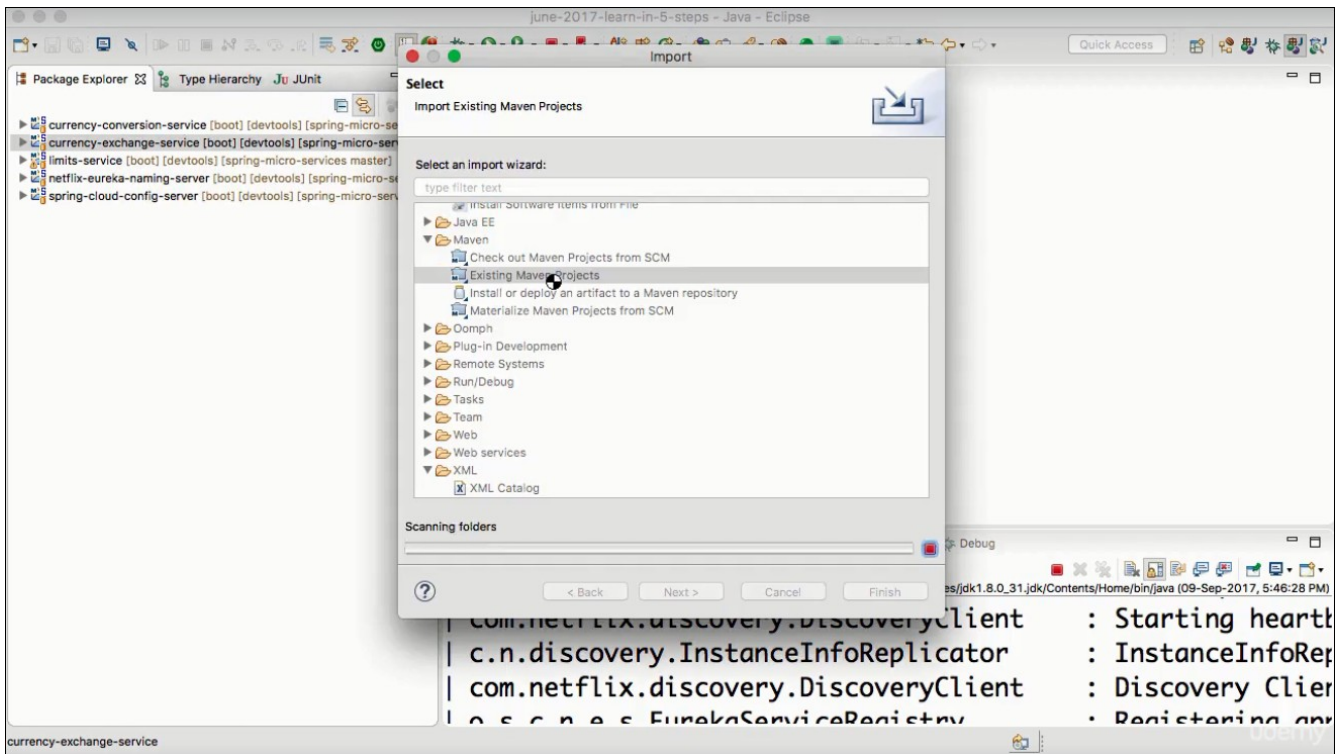
**\*\* API Gateway is the best place to perform a : Analytics, Debugg, Call tracing etc.**

**Action1: Setup Zuul API Gateway.**

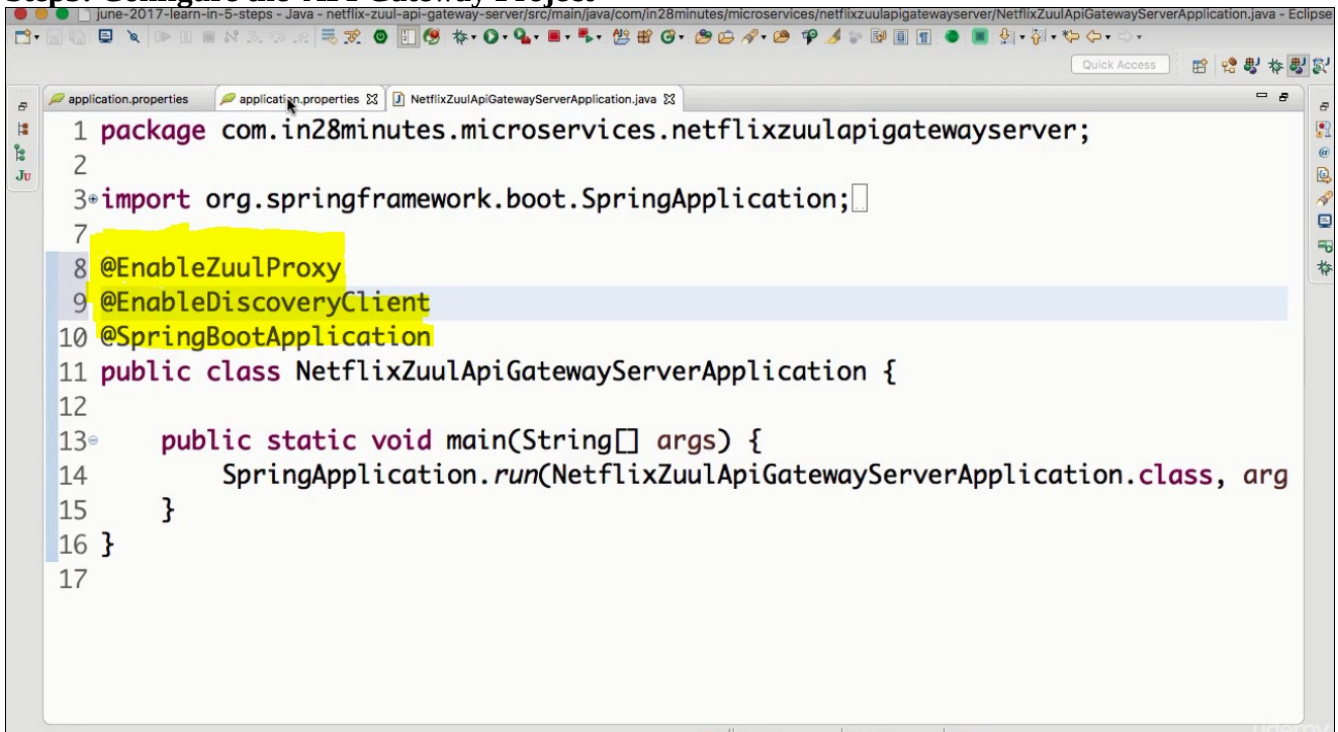
**Step1: Initialize & download**

The screenshot shows the Spring Initializr web application in a browser. The browser tabs include 'Spring Initializr', 'localhost:8888', 'Spring Microse', 'localhost:8080', 'localhost:8000', 'localhost:8001', 'localhost:8100', and 'Eureka'. The address bar shows 'start.spring.io'. The page has a dark header with 'SPRING INITIALIZR' and the tagline 'bootstrap your application now'. Below the header, there's a form to generate a project. At the top, it says 'Generate a' followed by a dropdown menu set to 'Maven Project', 'with' followed by a dropdown menu set to 'Java', and 'and Spring Boot' followed by a dropdown menu set to '2.0.0 M3'. The form is divided into two main sections: 'Project Metadata' and 'Dependencies'. Under 'Project Metadata', there's a field for 'Artifact coordinates' with the value 'com.in28minutes.microservices' and a field for 'Artifact' with the value 'netflix-zuul-api-gateway-server'. Under 'Dependencies', there's a search bar with the text 'Web, Security, JPA, Actuator, Devtools...' and a list of 'Selected Dependencies' which includes 'Zuul', 'Eureka Discovery', 'Actuator', and 'DevTools'. A large green button labeled 'Generate Project' is at the bottom of the form. Below the button, there's a link that says 'Don't know what to look for? Want more options? Switch to the full version.' The footer of the page says 'start.spring.io is powered by Spring Initializr and Pivotal Web Services'.

**Step2: Import to workspace as “Existing Maven Project”**



### Step3: Configure the API Gateway Project



```
1 spring.application.name=netflix-zuul-api-gateway-server
2 server.port=8765
3 eureka.client.service-url.default-zone=http://localhost:8761/eureka
4
```

## Lecture 89:

Action 2: Define what to do when API Gateway intercept the request

Ex: Logging

Step1: Add the Filter implementation for the Zuul.

```
1 package com.in28minutes.microservices.netflixzuulapigatewayserver;
2
3 import org.springframework.stereotype.Component;
4
5 import com.netflix.zuul.ZuulFilter;
6
7 @Component
8 public class ZuulLoggingFilter extends ZuulFilter{
9
10     @Override
11     public boolean shouldFilter() {
12         // TODO Auto-generated method stub
13         return false;
14     }
15
16     @Override
17     public Object run() {
18         // TODO Auto-generated method stub
19     }
20 }
```

boolean **shouldFilter()** → used to evaluate if the filter should be applied or not on this request.  
[return true]

Object **run()** → task to be performed.

String **filterType()** → **pre,post,error,all.** [ return “pre”]

int **filterOrder()** → set the priority order , when multiple filters are present.[ return 1]

```
16
17 @Override
18 public boolean shouldFilter() {
19     return true;
20 }
21
22 @Override
23 public Object run() {
24     HttpServletRequest request =
25         RequestContext.getCurrentContext().getRequest();
26     logger.info("request -> {} request uri -> {}",
27         request, request.getRequestURI());
28     return null;
29 }
30
31 @Override
32 public String filterType() {
33     return "pre";
34 }
```

### Action 3: Invoking Zull API Gateway

Step1: Make sure the following services are up n running in the order:

The screenshot shows the Eclipse IDE with the Package Explorer on the left, the Run/Debug dialog in the center, and the Console at the bottom. The Run/Debug dialog lists eight services to be started in a specific order, indicated by pink numbers 1 through 4. The console at the bottom shows the output of the application, including log messages and the command 'restartMain'.

**Service Startup Order:**

- 1 NetflixZuulApiGatewayServerApplication
- 2 CurrencyConversionServiceApplication8100
- 3 CurrencyExchangeServiceApplication8000
- 4 NetflixEurekaNamingServerApplication
- 5 CurrencyExchangeServiceApplication8001
- 6 LimitsServiceApplication
- 7 SpringCloudConfigServerApplication
- 8 ConfigExampleServiceApplication

**Console Output:**

```
2017-09-09 19:01:53.094 INFO 69104 --- [ restartedMain] o
2017-09-09 19:01:53.094 INFO 69104 --- [ restartedMain] .
2017-09-09 19:01:53.097 INFO 69104 --- [ restartedMain] n
```



## Step2: Ensure all the services are

The screenshot shows the Eureka web interface in a browser. The address bar is at localhost:8761. The interface is divided into several sections:

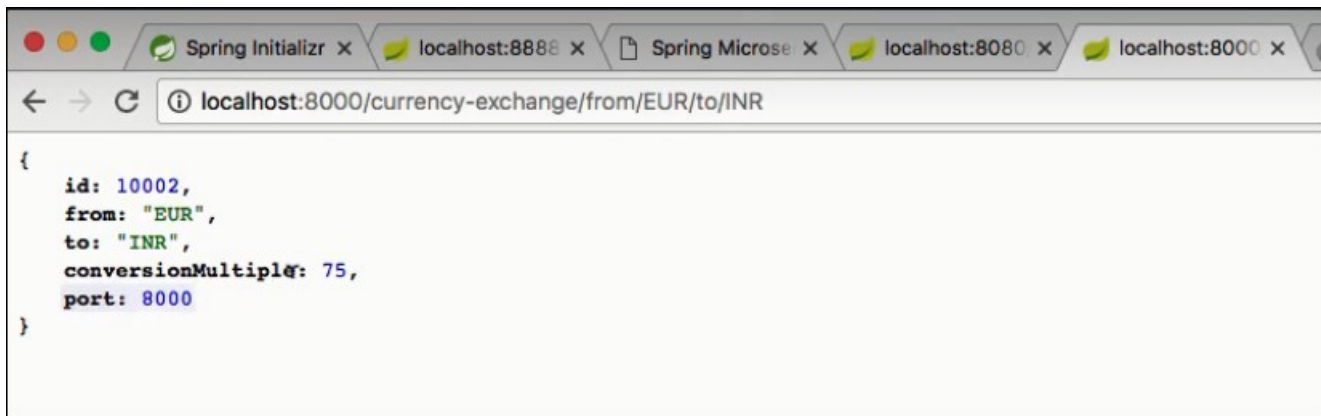
- Environment:** test
- Data center:** default
- Current time:** 2017-09-09T19:07:06 +0530
- Uptime:** 00:19
- Lease expiration enabled:** true
- Renews threshold:** 6
- Renews (last min):** 12
- DS Replicas:** localhost
- Instances currently registered with Eureka:**

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
NETFLIX-ZUUL-API-GATEWAY-SERVER	n/a (1)	(1)	UP (1) - 10.101.224.72:netflix-zuul-api-gateway-server:8765
- General Info:**

Name	Value
total-avail-memory	273mb
environment	test

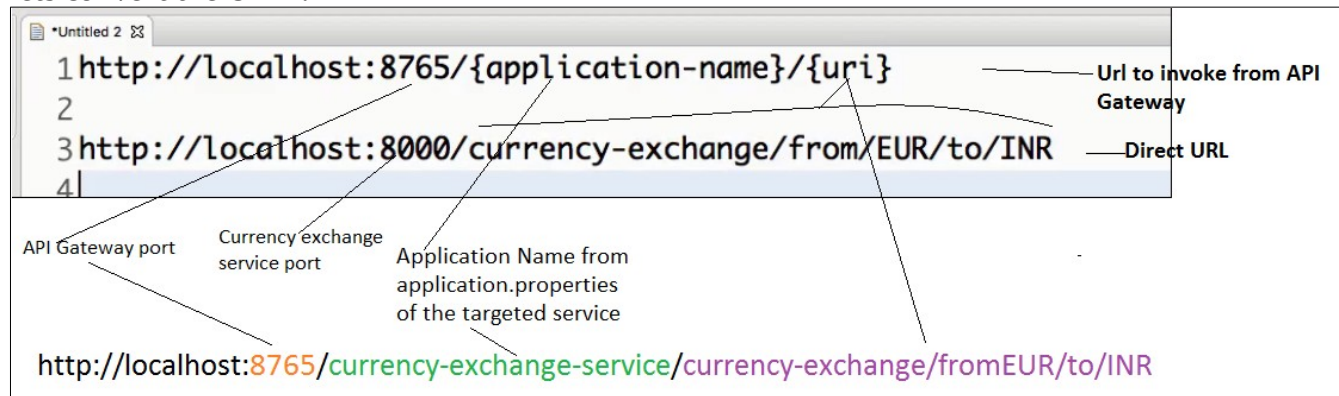
## Step3: Invoke the Currency Exchange Service

If we invoke it directly it will not go through API Gateway



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

lets convert the URL :



1 `http://localhost:8765/{application-name}/{uri}` — Url to invoke from API Gateway

2

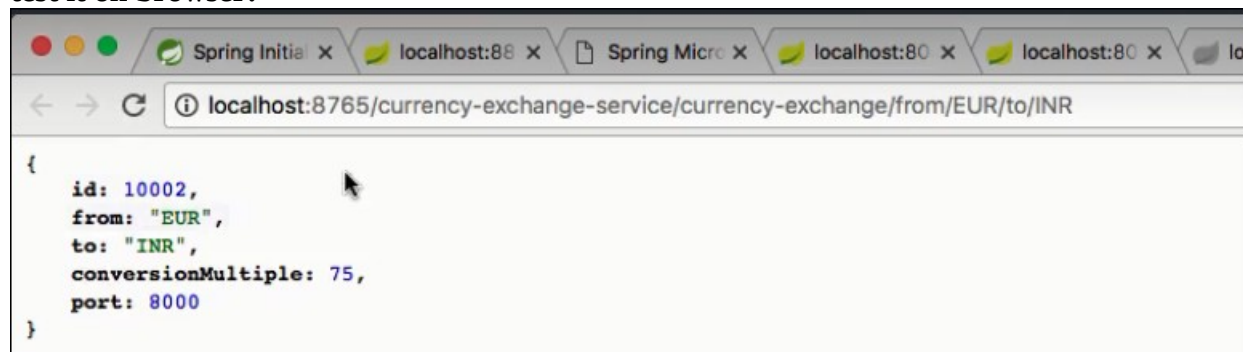
3 `http://localhost:8000/currency-exchange/from/EUR/to/INR` — Direct URL

4

API Gateway port      Currency exchange service port      Application Name from application.properties of the targeted service

`http://localhost:8765/currency-exchange-service/currency-exchange/fromEUR/to/INR`

test it on browser:



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

Check the logs to see the output:

```
request -> org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWr
request uri -> /currency-exchange-service/currency-exchange/from/EUR/to/INR
2017-09-09 19:11:35.740 INFO 69104 --- [nio-8765-exec-1] s.c.a.AnnotationConfig
```

**Action 4:** Lets update Feign client of CCS m/s to call the CES service via gateway rather than its direct URL. Update the feign client proxy as below.

```
1 package com.in28minutes.microservices.currencyconversion.service;
2
3 import org.springframework.cloud.netflix.feign.FeignClient;
4
5 // @FeignClient(name="currency-exchange-service", url="localhost:8000")
6 // @FeignClient(name="currency-exchange-service")
7 @FeignClient(name="netflix-zuul-api-gateway-server")
8 @RibbonClient(name="currency-exchange-service")
9 public interface CurrencyExchangeServiceProxy {
10     // @GetMapping("/currency-exchange/from/{from}/to/{to}")
11     // @GetMapping("/currency-exchange-service/currency-exchange/from/{from}/to/{to}")
12     public CurrencyConversionBean retrieveExchangeValue
13         (@PathVariable("from") String from, @PathVariable("to") String to);
14 }
15
16
17
18
```

Now we can call the CCS from its URL( the one that used feign client) & see it calls the CES via API gateway from its feign client.

```
{
  id: 10001,
  from: "USD",
  to: "INR",
  conversionMultiple: 65,
  quantity: 10,
  totalCalculatedAmount: 650,
  port: 8000
}
```

CCS → API GATEWAY → CES

lets make it pass through API GATEWAY even before caoming to CCS.

API GATEWAY → CCS → API GATEWAY → CES

API GATEWAY port

ApplicationName of CCS

URI of the CCS via Feign Client. This feign client has mapped URL to call CES via the API gateway