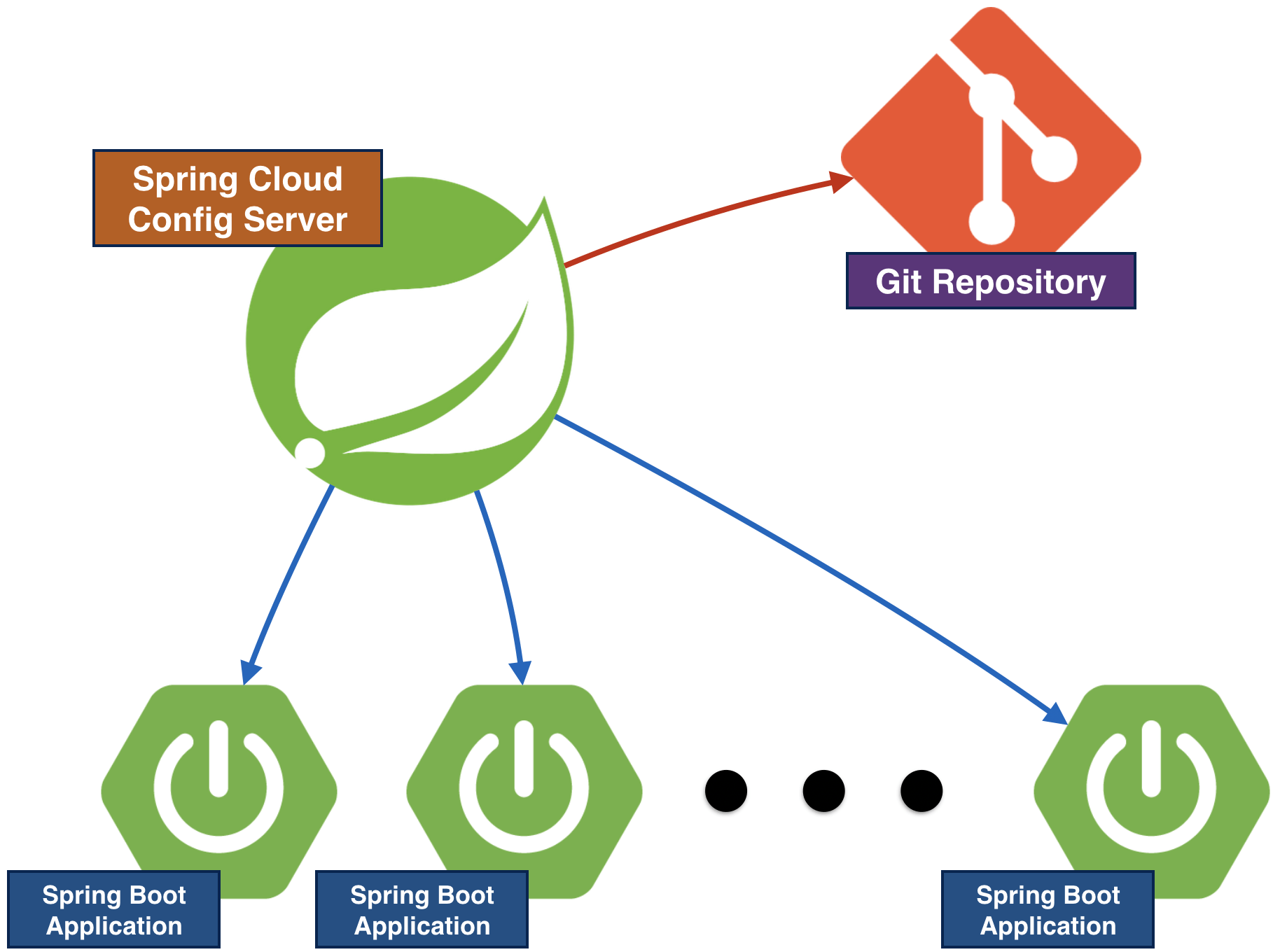
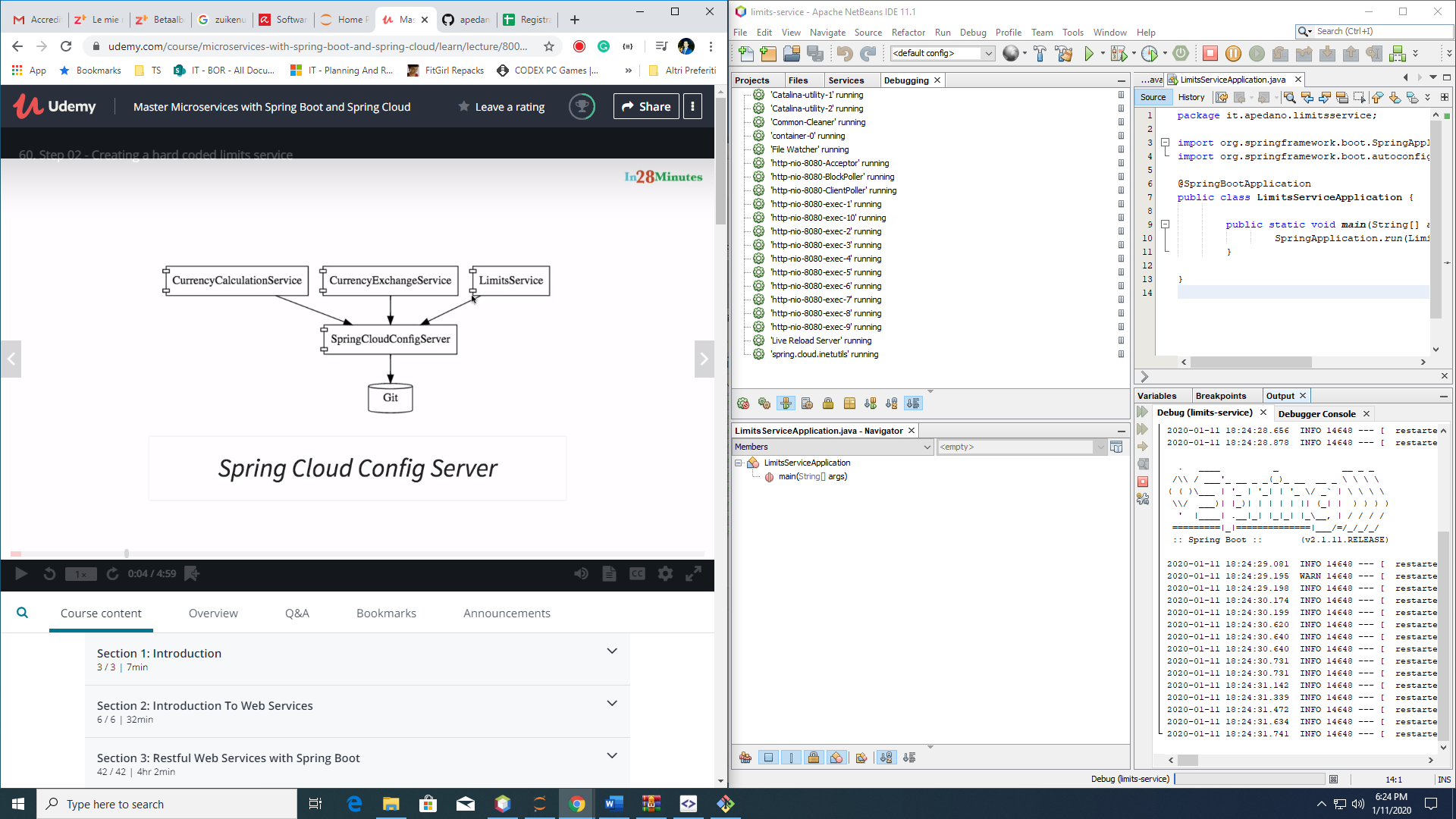
# Microservices architecture – Spring Cloud

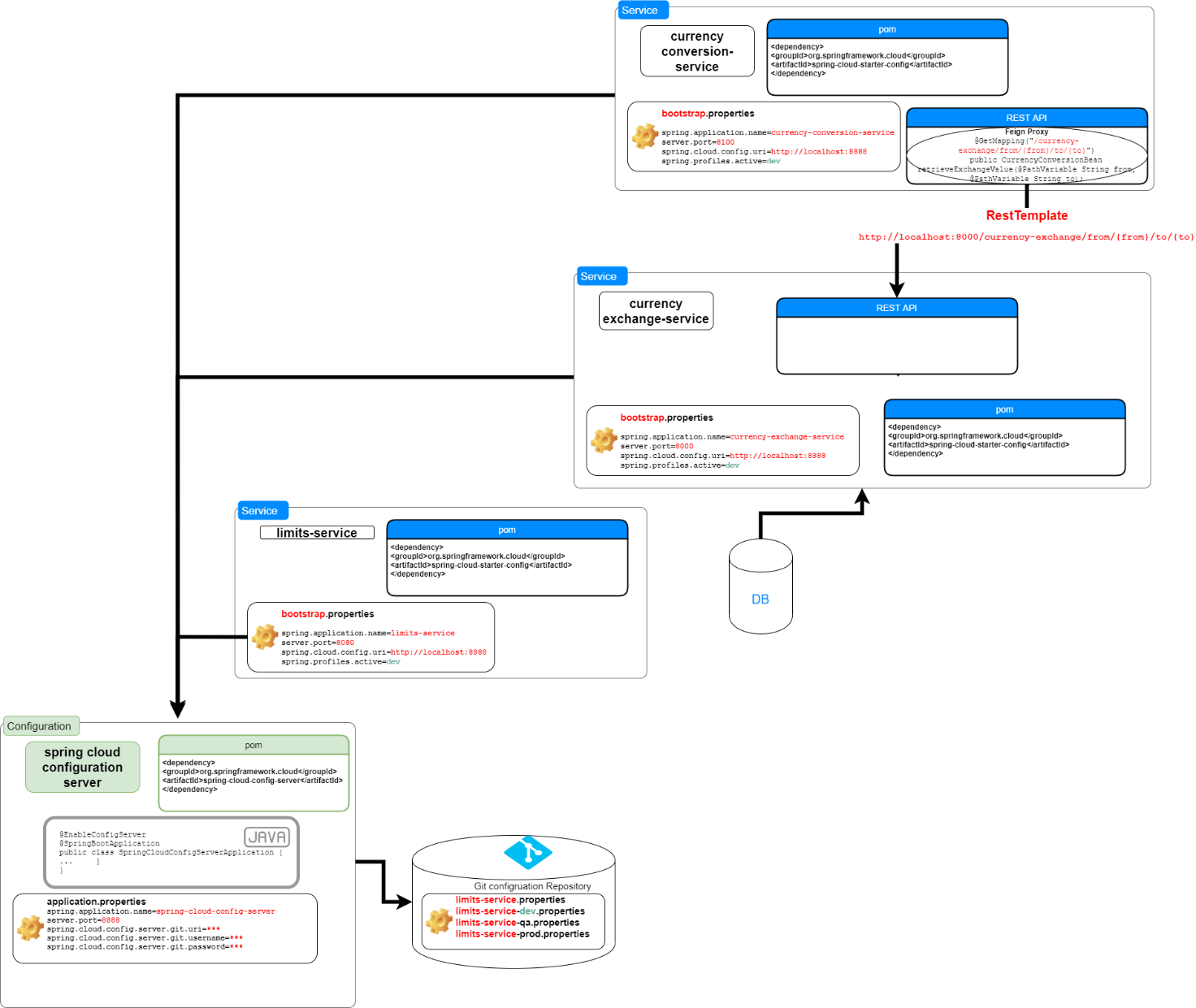
## Centralized application configuration – Spring Cloud Config Server

 Server responsible to host all microservices configuration in one centralized repository (Git). Each microservice has its own configuration (database, business logic config or external integration configuration) also depending on the running environments (each environment could run multiple instances of the same microservices):

* A **Development environment** is where you configure, customize, and use source control to build an image of the Waveset application to be promoted to another environment. You also write an upgrade procedure in this environment that you follow in each target environment.
* A **Test environment** is where you test your upgrade procedure against controlled data and perform controlled testing of the resulting Waveset application.
* A **QA environment** is where you test your upgrade procedure against data, hardware, and software that closely simulate the Production environment and where you allow intended users to test the resulting Waveset application.
* A **Production environment** is where the Waveset application is actually available for business use.

The server is able to provide all possible configurations depending on the running environment.





## Feign

Feign is used to implement communication between microservices is based on proxies on the caller, to be able to interact with a specific call to the target api

## Ribbon

When a microservice requires communication with other microservices, it shouldn’t be aware of the number of instances and the specific addresses of the called microservice, this makes the caller independent from the called microservice and also adds flexibility if we want to scale the number of the called microservice.

Ribbon helps to do so:

### Ribbon acts as a load balancer, hiding the real instances of the currency exchange ms to the currency calculation service. How to configure Ribbon

#### Maven depencency on the caller

<dependency>

<groupId>**org.springframework.cloud**</groupId>

<artifactId>**spring-cloud-starter-netflix-ribbon**</artifactId>

<version>**2.2.1.RELEASE**</version>

</dependency>

#### Add Ribbon client to Feign Proxy

Set the client with the name of the target service

@FeignClient**(**name**=**"currency-exchange-service"**)**

@RibbonClient**(**name**=**"currency-exchange-service"**)**

public interface CurrencyExchangeFeignProxy **{**

@GetMapping**(**"/currency-exchange/from/{from}/to/{to}"**)**

public CurrencyConversionBean retrieveExchangeValue**(**@PathVariable String from**,** @PathVariable String to**);**

**}**

#### Add properties

In the application.properties file. We need to specify the running instances of the called microserivce

#======== RIBBON ============

#format: <name\_of\_the\_service>.ribbon.listOfServers

currency-exchange-service.ribbon.listOfServers**=**http://localhost:8000,http://localhost:8001

#### Call the microservice

Supposing we have the running instances of the microservices, when we call those through Ribbon, the following happens:

2020-01-25 18:52:37.226 INFO 8928 --- [nio-8100-exec-1] c.netflix.config.ChainedDynamicProperty : Flipping property: currency-exchange-service.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647

2020-01-25 18:52:37.247 INFO 8928 --- [nio-8100-exec-1] c.netflix.loadbalancer.BaseLoadBalancer : Client: currency-exchange-service instantiated a LoadBalancer: DynamicServerListLoadBalancer:{NFLoadBalancer:name=currency-exchange-service,current list of Servers=[],Load balancer stats=Zone stats: {},Server stats: []}ServerList:null

2020-01-25 18:52:37.264 INFO 8928 --- [nio-8100-exec-1] c.n.l.DynamicServerListLoadBalancer : DynamicServerListLoadBalancer for client currency-exchange-service initialized: DynamicServerListLoadBalancer:{NFLoadBalancer:name=currency-exchange-service,current list of Servers=[**localhost:8000, localhost:8001**],Load balancer stats=Zone stats: {unknown=[Zone:unknown; Instance count:2; Active connections count: 0; Circuit breaker tripped count: 0; Active connections per server: 0.0;]

},Server stats: [[Server:localhost:8000; Zone:UNKNOWN; Total Requests:0; Successive connection failure:0; Total blackout seconds:0; Last connection made:Thu Jan 01 01:00:00 CET 1970; First connection made: Thu Jan 01 01:00:00 CET 1970; Active Connections:0; total failure count in last (1000) msecs:0; average resp time:0.0; 90 percentile resp time:0.0; 95 percentile resp time:0.0; min resp time:0.0; max resp time:0.0; stddev resp time:0.0]

, [Server:localhost:8001; Zone:UNKNOWN; Total Requests:0; Successive connection failure:0; Total blackout seconds:0; Last connection made:Thu Jan 01 01:00:00 CET 1970; First connection made: Thu Jan 01 01:00:00 CET 1970; Active Connections:0; total failure count in last (1000) msecs:0; average resp time:0.0; 90 percentile resp time:0.0; 95 percentile resp time:0.0; min resp time:0.0; max resp time:0.0; stddev resp time:0.0]

]}ServerList:com.netflix.loadbalancer.ConfigurationBasedServerList@3f0361b7

We inserted the port of the called microservices in the response.

**{**

"id"**:** 10001**,**

"from"**:** "USD"**,**

"to"**:** "INR"**,**

"conversionMultiple"**:** 65.00**,**

"quantity"**:** 123456**,**

"totalCalculatedAmount"**:** 8024640.00**,**

"port"**:** 8000

**}**

If we send the request multiple time we will notice the port changing because of the load balancer.

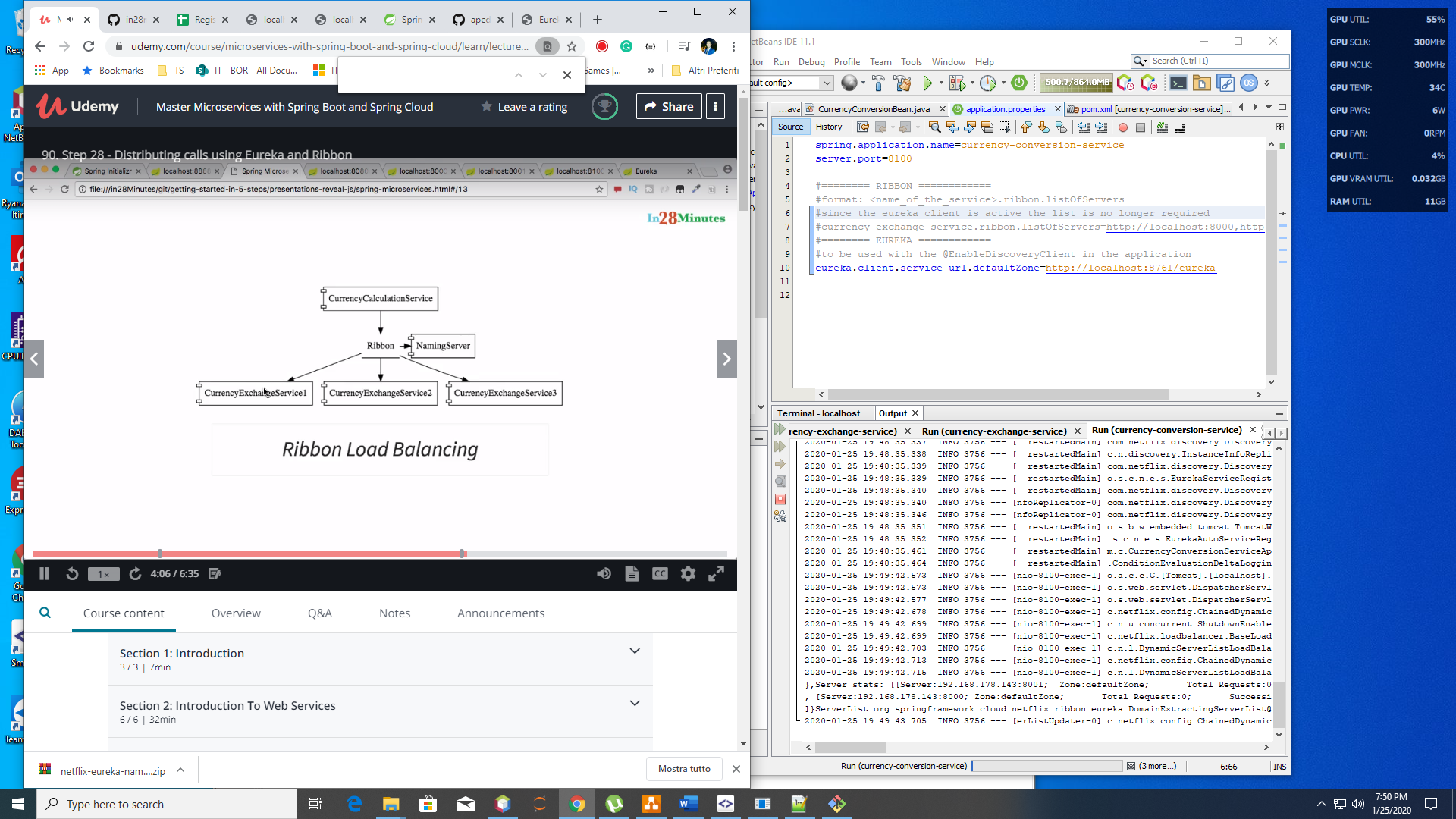
### Naming server

The issue with this implementation is that the list of servers is statically configured in the ribbon client properties file. If we need to scale up of down the called ms, we need to dynamically inform the ribbon client of the server list.

## Eureka naming server (NETFLIX-EUREKA-NAMING-SERVER)

Whenever a new instance of one microservices starts, it has to register to the naming server (**SERIVCE REGISTRATION**).

Whenever a service wants to talk to a service, then, it has to call Eureka to be informed about the currently running instances of that specific microservice (**SERVICE DISCOVERY**).



## API Gateways

Used to implement **common features** like

* authentication, authorization, security
* loggin
* rate limits
* fault tolerance
* service aggregation (suppose you need to call several different microservices to achieve a higher level functionality, for this you’d need an aggregation technic to show the set of ms as a high level functionality to the external environment)

All calls to our microservice ecosystem will go through the API gateways, implementing the common features.

## Netflix implementation of an API gateway – ZUUL

Three steps are required to set-up a Zuul gateway

1. Create a specific component as ZUUL getaway server
2. Decide what it should do when the server intercept a call
3. Make sure that the right request goes through the API

Once the Zuul gateway service is up and running, In order to use it you have to call the port where the server is running with the following schema

http://{zuul-host}:{zuul-port}/{service-name}/{service-uri}

example

http://localhost:8765/currency-exchange-service/currency-exchange/from/USD/to/INR

If you try the call the Zuul logging will log

2020-01-26 16:38:55.974 INFO 15984 --- [nio-8765-exec-9] a.m.n.ZuulLogginFilter : Request received -> org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@7717bb07

2020-01-26 16:38:55.974 INFO 15984 --- [nio-8765-exec-9] a.m.n.ZuulLogginFilter : Request URI -> /currency-exchange-service/%20from/USD/to/INR

2020-01-26 16:39:52.314 INFO 15984 --- [nio-8765-exec-2] a.m.n.ZuulLogginFilter : Request received -> org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@64a4131c

2020-01-26 16:39:52.314 INFO 15984 --- [nio-8765-exec-2] a.m.n.ZuulLogginFilter : Request URI -> /currency-exchange-service/currency-exchange/from/USD/to/INR

### Use the gateway between microservice calls

For instance, we want to run the gateway between a call going from the currency conversion to the currency exchange service.

We need to use the proxy in the caller

**@FeignClient(name="{zuul-server-service-name}")**

That is

**@FeignClient(name="netflix-zuul-api-gateway-server")**

In the proxy method you also need to add the service name of the target service, otherwise zuul would not know where to forward the call to.

**@GetMapping("{target-service-name}/{call-uri}")**

That is

**@GetMapping("/currency-exchange-service/currency-exchange/from/{from}/to/{to}")**

So, if we call

<http://localhost:8100/currency-converter-feign/from/USD/to/INR/quantity/123456>

we will get in the zuul output (even though the call is not directed to 8765

2020-01-26 16:52:27.365 INFO 15984 --- [erListUpdater-1] c.netflix.config.ChainedDynamicProperty : Flipping property: currency-exchange-service.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647

2020-01-26 16:53:53.425 INFO 15984 --- [nio-8765-exec-3] a.m.n.ZuulLogginFilter : Request received -> org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@7497ee72

2020-01-26 16:53:53.425 INFO 15984 --- [nio-8765-exec-3] a.m.n.ZuulLogginFilter : Request URI -> /currency-exchange-service/currency-exchange/from/USD/to/INR **(the url from the proxy)**

#### Execute the gateway from the first call

We need to point the call to the gateway

<http://localhost:8765/currency-conversion-service/currency-converter-feign/from/USD/to/INR/quantity/123456>

2020-01-26 16:59:58.455 INFO 15984 --- [io-8765-exec-10] a.m.n.ZuulLogginFilter : Request URI -> /currency-conversion-service/currency-converter-feign/from/USD/to/INR/quantity/123456

2020-01-26 16:59:58.462 INFO 15984 --- [nio-8765-exec-1] a.m.n.ZuulLogginFilter : Request URI -> /currency-exchange-service/currency-exchange/from/USD/to/INR

## Distributed tracing

If something goes wrong during a call execution going through multiple microservices, we need a single point to look into, in order to get all required info, the complete chain of what happened to a specific call.

Spring cloud Sleuth will assign a unique id to a request so that I can trace the request through the ms. This is a **distributed tracing system**. Wi will log all the calls to a queue (Rabbit MQ) and we will send out to a Zipking server, where the messages will be consolidated to look through all the steps of the request handling.