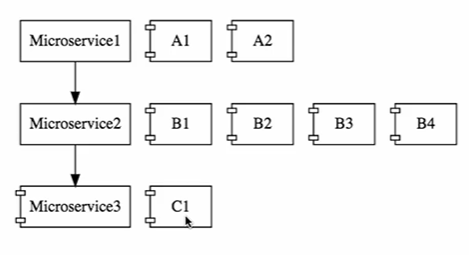
## Microservices with spring cloud

## What is microservices?

* REST
* Small well chosen deployable units
* Cloud enabled

Instead of having single monolithic architecture we divide them into small microservices which will be independent of each other.

So that if there are 4 microservices instances of each microservice can be increased or decreased based on load for each microservice.



## Challenges with microservices

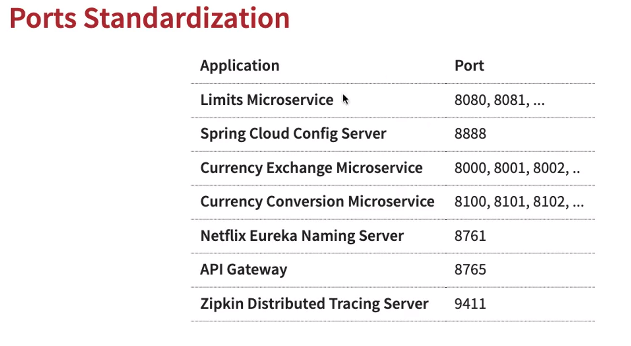
1. Bounded context
   1. Deciding right boundary
2. Configuration management
   1. If there are hundreds of microservices we have manage environment of all of them
3. Dynamic scale up and scale down
4. Visibility
   1. Monitoring all microservices so that if bugs comes we should able detect which one is down
5. Pack of cards
   1. 
      1. If multiple microservices are dependent on specific microservice need set fault tolerance because if it goes down it will affect others

## Introduction to spring cloud

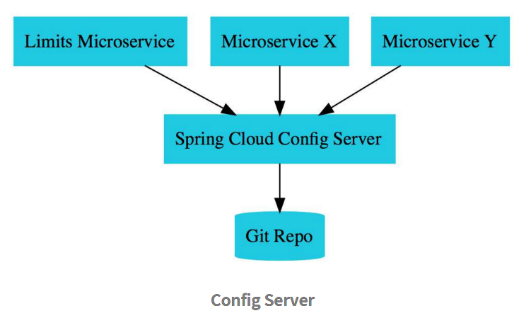
* Spring cloud Netflix

## Advantages

1. New technology and process adaption
   1. 
      1. Each of them build in different technology like java, python,………. And they can easily communicate with each other
2. Dynamic scaling
3. Faster release cycles
   1. As we develop app into small microservices instead of whole monolithic app giving any small updates is very easy



---------------------------------------------------------------------**STEP 01**---------------------------------------------------------------------



## Setup

Dependency

1. Spring web
2. Actuator => Monitor and manage app
3. Devtools
4. Config client

In application.properties add

Spring.config.import = optional:configserver:http://localhost:8888

138

Create controller LimitsController and returning hardcoded limits (line 13)

139 Setup limit service

Get limits from application.properties

* Add : limits-service.minimum=2
* Add : limits-service.maximum=999

Access these in controller

* Create class called Configuration and annotate with @ConfigurationProperties(“limits-service”) and access minimum and maximum

140 Setup spring cloud config server

Dependency

* + Config server
  + Devtools
* Gives port as server.port=8888

141 Creating git repository

143 Connect spring cloud config server to local repository

=> Add in application.properties

spring.cloud.config.server.git.uri=file:///D:/Spring\_Udemy/git-config-server

* Add in main annotation *@EnableConfigServer*
* Check on localhost:8888/limits-service/default
  + limits-service -> name of file in git path

144 run localhost:8080/limits => we will see content from local git file

Note:

* In Application.properties of limitsService the spring.application.name must be same as file name in local git location

What happens in background?



* + As soon as localhost:8080/limits is ran it goes to config server i.e. localhost:8888 as we have spring.config.import=optional:configserver:http://localhost:8888 in application.properties
  + Then it takes limits-service as we mentioned as spring.application.name and profile as default
    - Which gives in together localhost:8888/limits-service/default as url and redirected to it and gets the required data and it in limitsService

145 => create two more file at git repo called limits-service-dev.properties and limits-service-qa.properties

=> now we would like to access these

=> if we try [**http://localhost:8888/limits-service/dev**](http://localhost:8888/limits-service/dev)

Gives

{

"name": "limits-service",

"profiles": [

"dev"

],

"label": null,

"version": "aec2411d0f075b87e5baca730955f378b19bbd34",

"state": null,

"propertySources": [

{

"name": "file:///D:/Spring\_Udemy/git-config-server/limits-service-dev.properties",

"source": {

"limits-service.minimum": "2",

"limits-service.maximum": "998"

}

},

{

"name": "file:///D:/Spring\_Udemy/git-config-server/limits-service.properties",

"source": {

"limits-service.minimum": "1",

"limits-service.maximum": "999"

}

}

]

}

limits-service-dev.properties

limits-service.properties

* Similarly one can try with http://localhost:8888/limits-service/qa
* Picking only qa or dev
  + Spring.cloud.config.profile=dev
  + Try with http://localhost:8080/limits

\*\* Play with above two services created as these will be deleted in further steps

----------------------------------------------------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------**STEP 02**---------------------------------------------------------------------

\* Currency\_Conversion\_Service 🡪 Currency\_Exchange\_Service 🡪 database

149 Creating currency exchange service

Dependency

* Spring web
* Devtools
* Actuator
* Config client

Add these application.properties

server.port=8000

spring.config.import=optional:configserver:http://localhost:8888

150

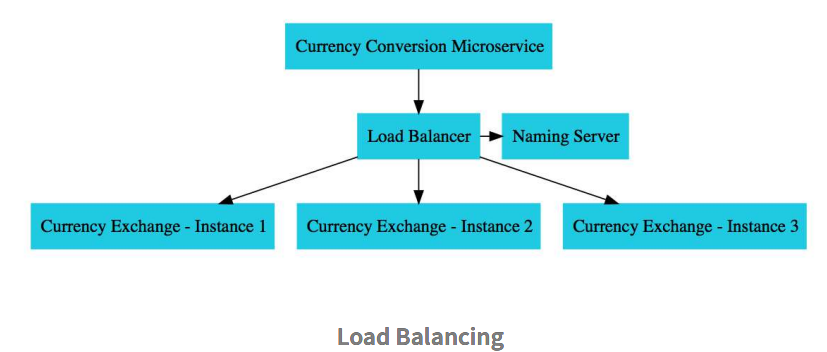
URL

http://localhost:8000/currency-exchange/from/USD/to/INR

Response Structure

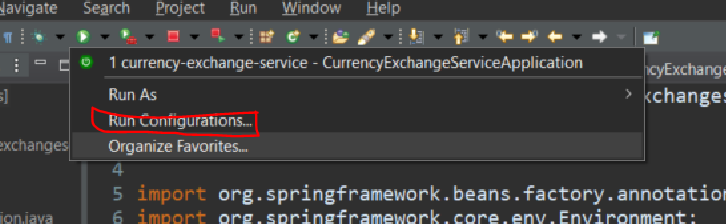
1. {
2. "id":10001,
3. "from":"USD",
4. "to":"INR",
5. "conversionMultiple":65.00,
6. "environment":"8000 instance-id"
7. }

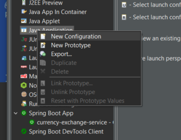
152

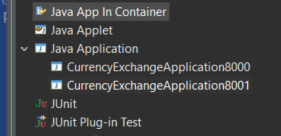


* + To know which instance currency conversion running on add field environment in CurrencyExchange class
  + *@RestController*
  + public class CurrencyExchangeController {
  + *@Autowired*
  + private Environment environment;
  + *@GetMapping*("/currency-exchange/from/{from}/to/{to}")
  + public CurrencyExchange retrieveExchangeValue(*@PathVariable* String from, *@PathVariable* String to) {
  + String port = environment.getProperty("local.server.port");
  + CurrencyExchange currencyExchange = new CurrencyExchange(1000L, from, to, BigDecimal.*valueOf*(50));
  + currencyExchange.setEnvironment(port);
  + return currencyExchange;
  + }
  + }

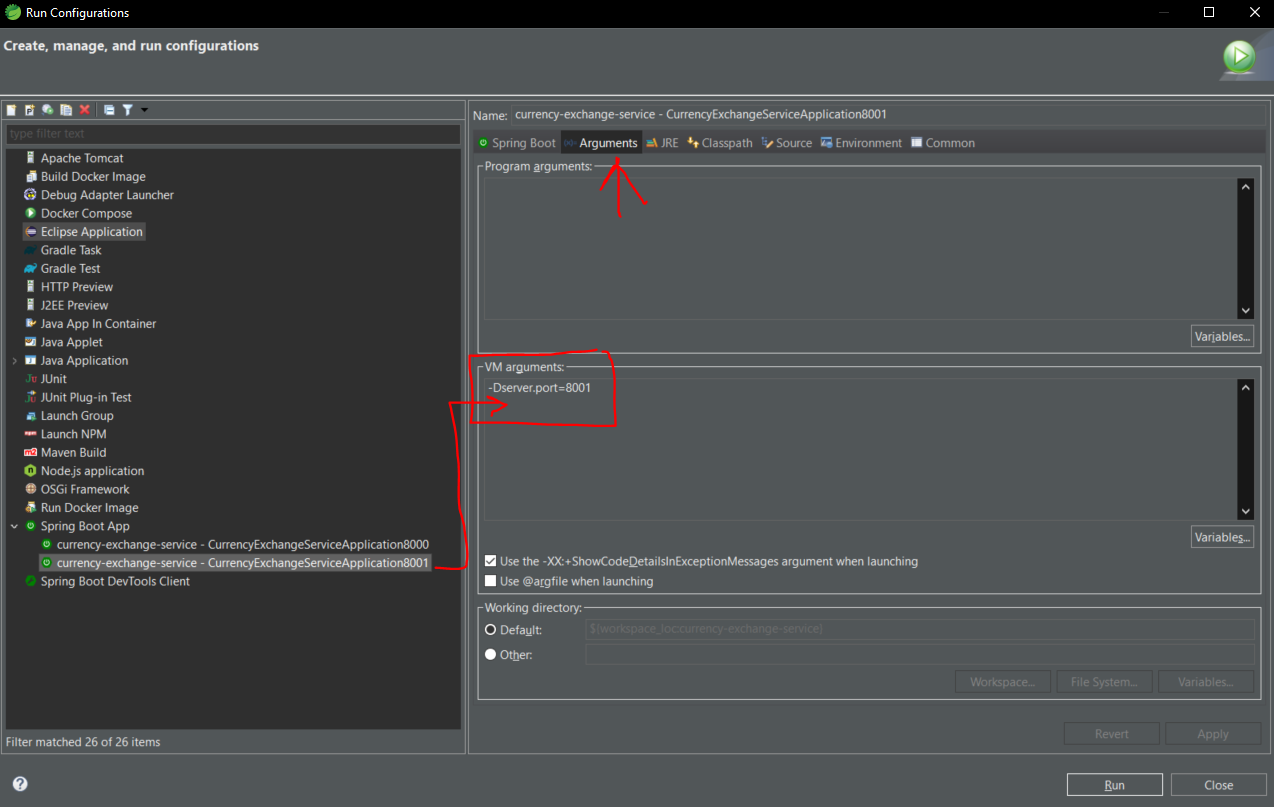
## How can we have 2 instances running?

* 



 (Instead of Java application use Spring boot)

Now go in 8001



153

JPA and H2 db

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

Configure db

spring.jpa.show-sql=true

spring.datasource.url=jdbc:h2:mem:testdb

spring.h2.console.enabled=true

To access database

* Go to localhost:8000/h2-console
  + Make sure to have same JDBC url and click Connect

To insert data in table evrytime when it get started bcoz in h2 is inmemory db

* Create text file have query into it and save at
  + Src/main/resources
  + File name data.sql

Add spring.jpa.defer-datasource-initialization=true

So that queries will be fired once table is created bcoz in spring it happen reverse queries fired first and then tables created (strange)

157

Currency conversion service

#### URL

http://localhost:8100/currency-conversion/from/USD/to/INR/quantity/10

#### Response Structure

1. {
2. "id": 10001,
3. "from": "USD",
4. "to": "INR",
5. "conversionMultiple": 65.00,
6. "quantity": 10,
7. "totalCalculatedAmount": 650.00,
8. "environment": "8000 instance-id"
9. }

160

## Calling Currency\_Exchange form Currency\_Conversion

* + RestTemplate
    - Tedious work can be minimized with spring cloud

161

## To avoid above boiler code make use of “**Feign**”

## Dependency

<dependency>

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

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

## Interface

package com.example.microservices.currencyconversionservice;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

*@FeignClient*(name="currency-exchange", url="localhost:8000")

public interface CurrencyExchangeProxy {

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

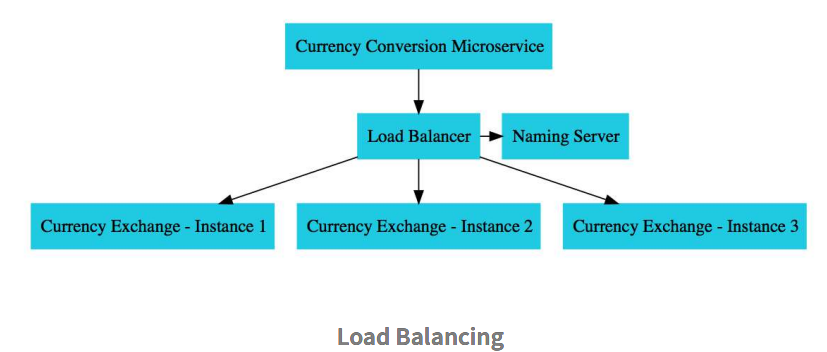
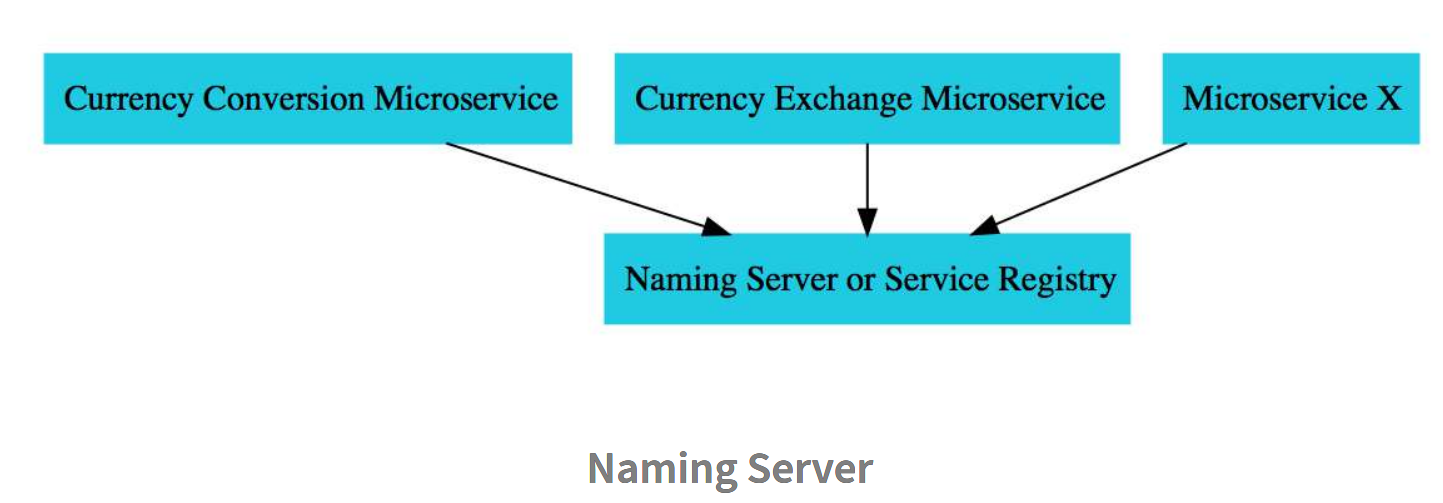
public CurrencyConversion retrieveExchangeValue(*@PathVariable* String from, *@PathVariable* String to);

}

## Annotation at main *@EnableFeignClients*

## What is proxy?

162



## Dynamically launch instances and distribute load between them

* + All instances of microservices must register with Service registry/Naming server
  + If one microservice want connect with other microservice instead of directly connecting with it, it will take address from Naming server and then connect…

## Create naming-server service

Dependency

* Devtools
* Actuator
* Eureka server

## add *@EnableEurekaServer* over main

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

163

Registering microservices with naming server (eureka)

Add dependency

<dependency>

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

<artifactId>spring-cloud-netflix-eureka-client</artifactId>

</dependency>

168

## Load balancing with Eureka, Feign and spring cloud balancer

//@FeignClient(name="currency-exchange", url="localhost:8000")

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

By making above change feign automatically manages loadBalancing

170

## Where will you execute common features like authentication, logging….

🡪 Cloud api gateway --- spring cloud gateway

Dependencies

1. Eureka discovery client
2. Actuator
3. Gateway (spring cloud routing)
4. Devtools

171

Initial

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

- http://localhost:8765/CURRENCY-CONVERSION/currency-conversion/from/USD/to/INR/quantity/10

- http://localhost:8765/CURRENCY-CONVERSION/currency-conversion-feign/from/USD/to/INR/quantity/10

Lower Case

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

- http://localhost:8765/currency-conversion/currency-conversion/from/USD/to/INR/quantity/10

- http://localhost:8765/currency-conversion/currency-conversion-feign/from/USD/to/INR/quantity/10

Custom Routes

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

- http://localhost:8765/currency-conversion/from/USD/to/INR/quantity/10

- http://localhost:8765/currency-conversion-feign/from/USD/to/INR/quantity/10

- <http://localhost:8765/currency-conversion-new/from/USD/to/INR/quantity/10>

172

Now we in order to call 8000 and other through gateway

Add

spring.cloud.gateway.discovery.locator.enabled=true