

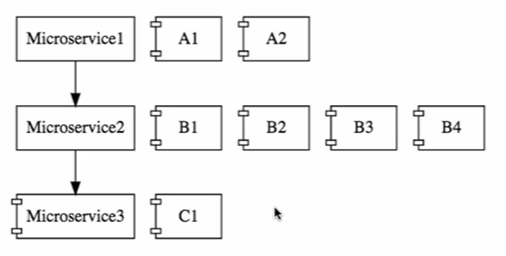
# Introduction:

Instead of creating one big microservice, create small services:





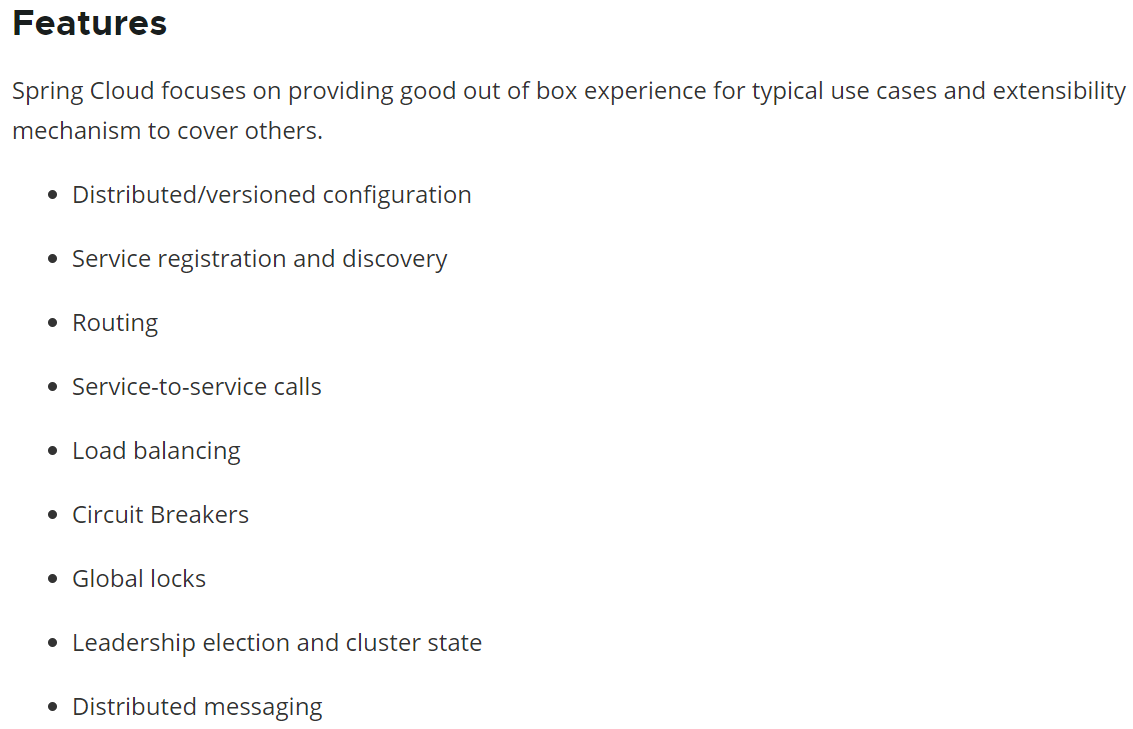
Below are the multiple instances of microservices-

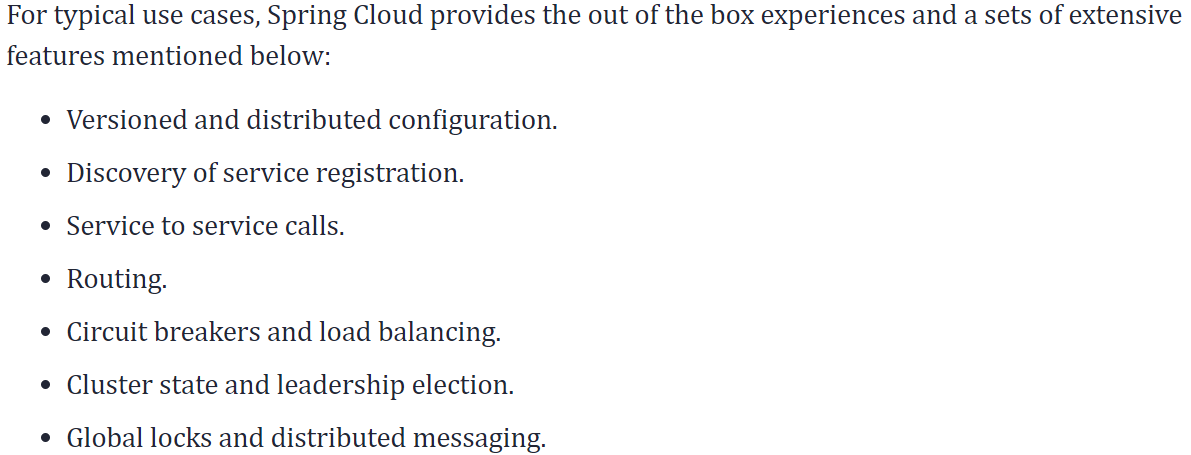


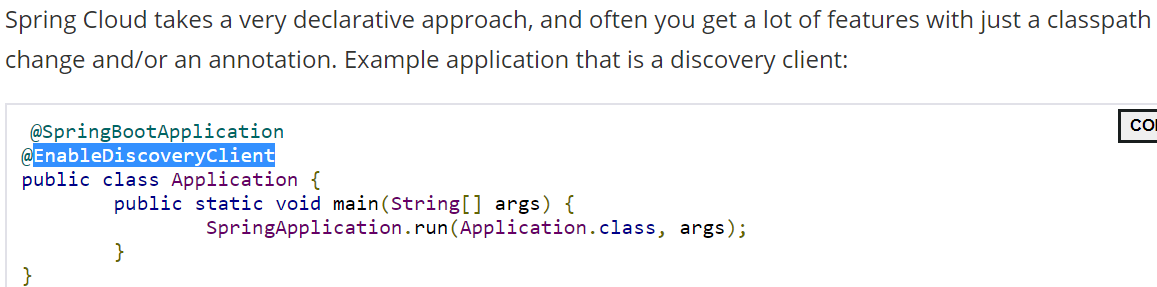
**Cloud Enabled means**- you can increase or decrease instances of microservice by using spring cloud very less configuration.

# Spring Cloud:

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, including the developer’s own laptop, bare metal data centres, and managed platforms such as Cloud Foundry one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment.







Spring Cloud Netflix: It is most importance project of spring cloud.

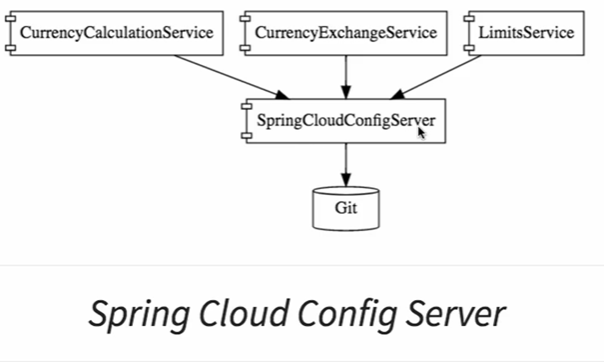
# Spring Cloud Component -:

Below is the list of Challenges that we solve by Spring cloud only like:

## Configuration Management:

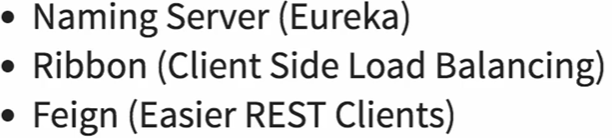
it is done by “Spring Cloud Config Server”.  
Multiple services + multiple Environment + multiple instances for those multiple services.  
means there are the lots of configuration.

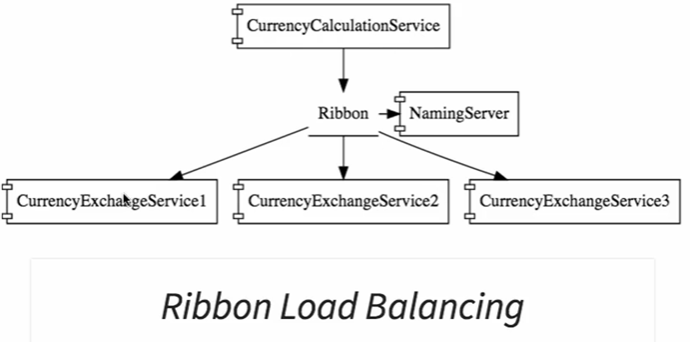
Spring Cloud Config Server provide an approach where you can store all the configuration for all the different environment of the all the microservices in a Git Repository (one place).



## Dynamic Scale up and scale down:

Dynamic Scaling means based on the load you can up and down dynamically (scaling).





1. All microservice can register with the Naming Server using Eureka
2. Service Discovery
3. We will use Ribbon for client side load balancing.
4. We will use Feign in the CurrencyCalculationService as a mechanism to write Simple Restfull Client.

## Visibility Monitoring



## Fault Tolerance:

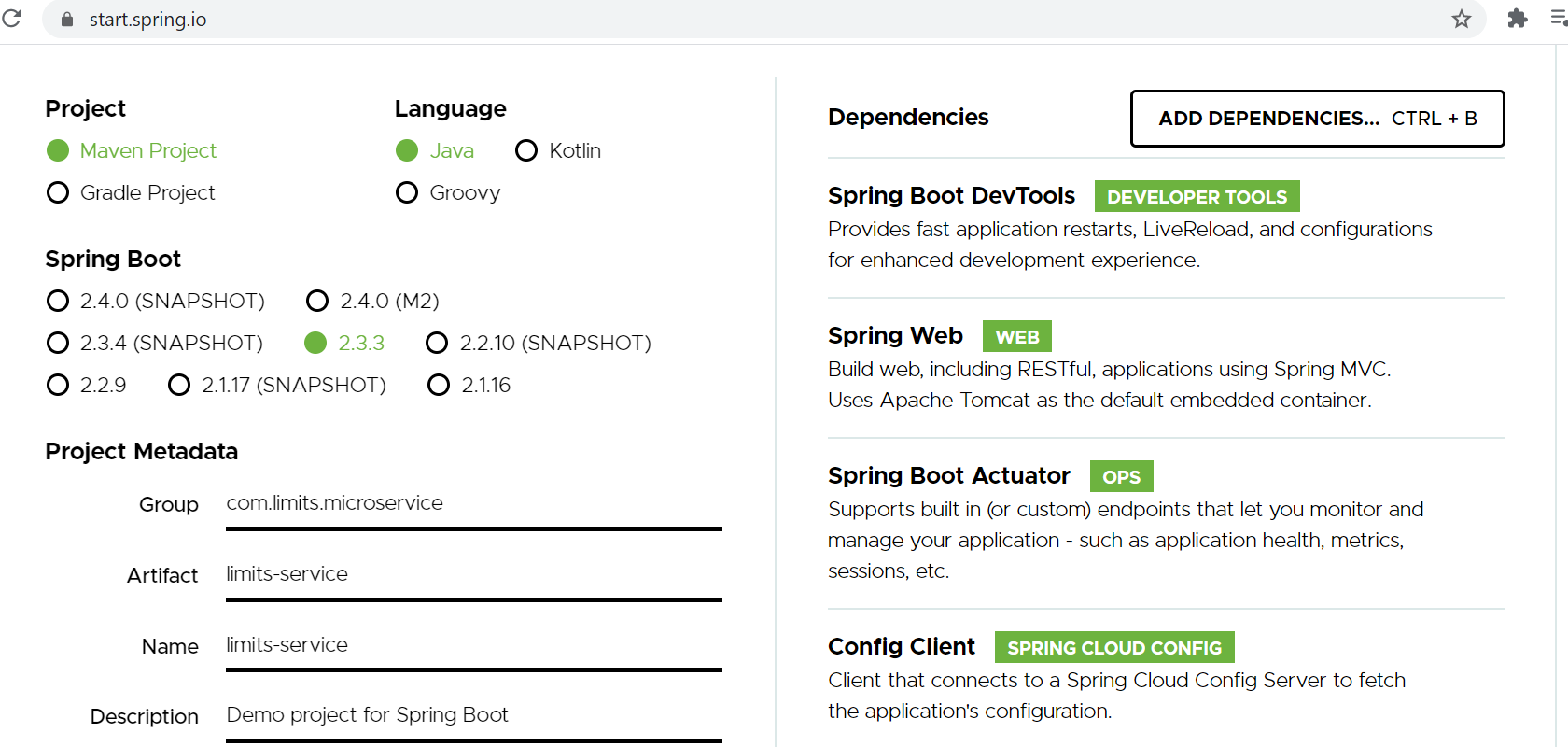
We will use Hystrix for fault tolerance-> If the service is done then **Hystrix** will configure default response.

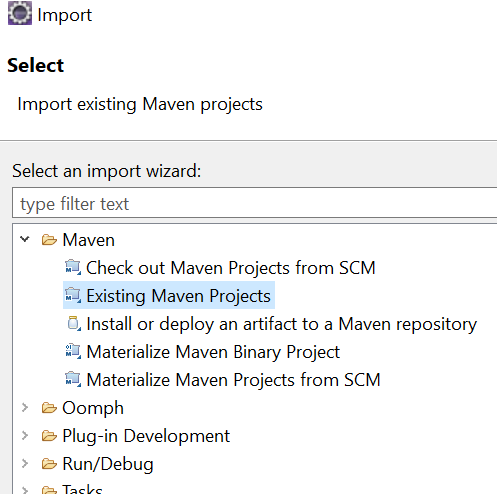
# Advantages of Microservices:

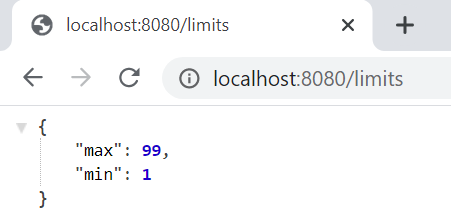
1. It is enabled to adapt new technologies and processes very easily.
2. Dynamic Scaling means based on the load you can up and down dynamically (scaling).
3. Faster Release cycles.

# Project-#1:

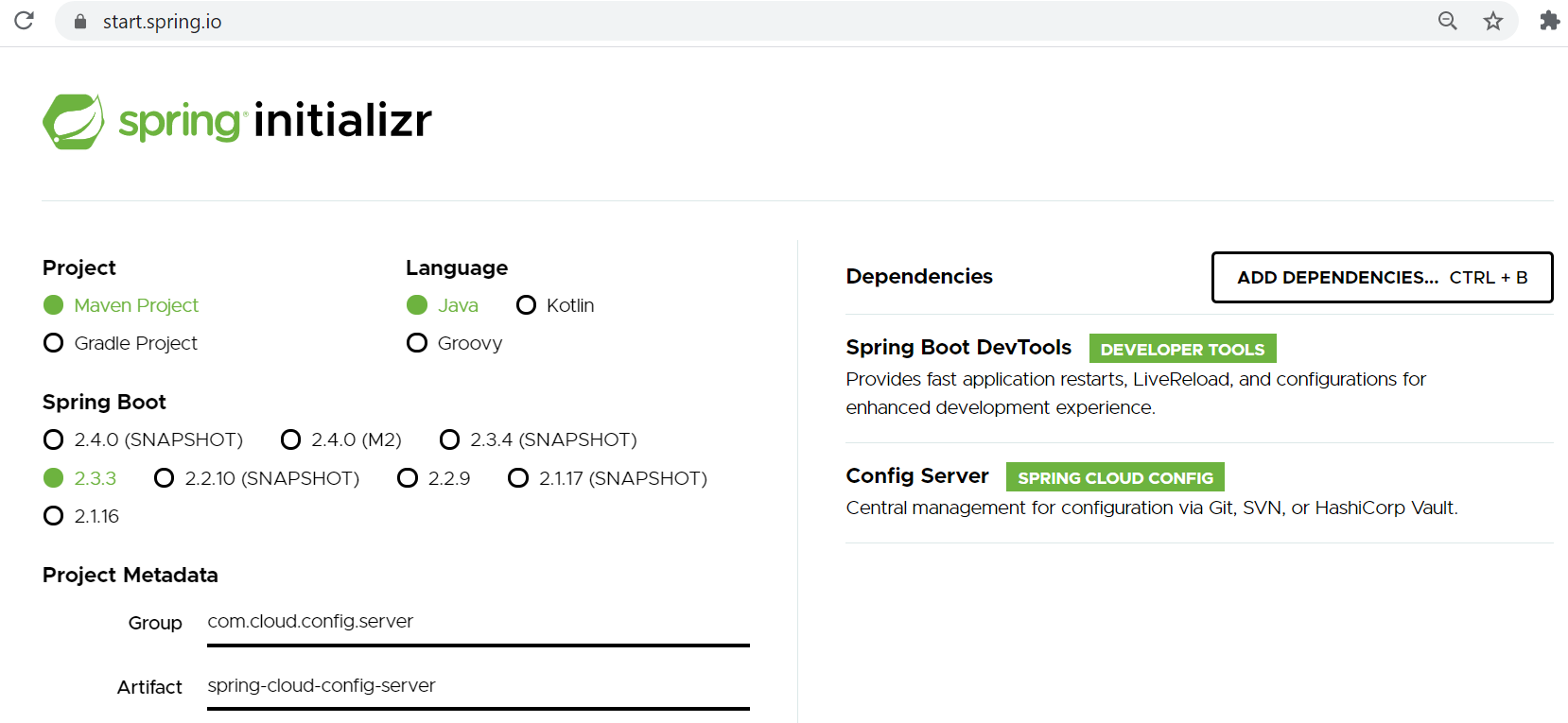
## Microservice-#1- > Limit-Services:



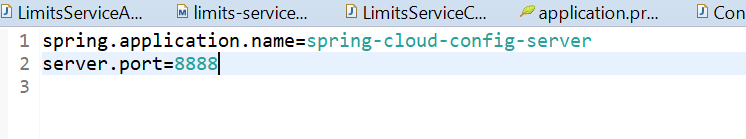


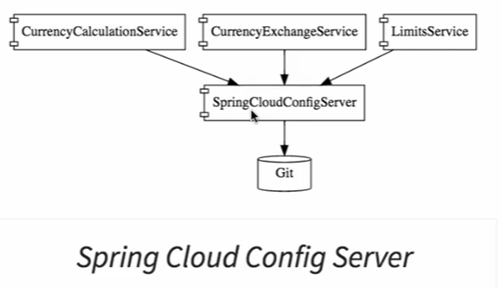


## Microservice-#2 - > Spring-Cloud-Config-Server

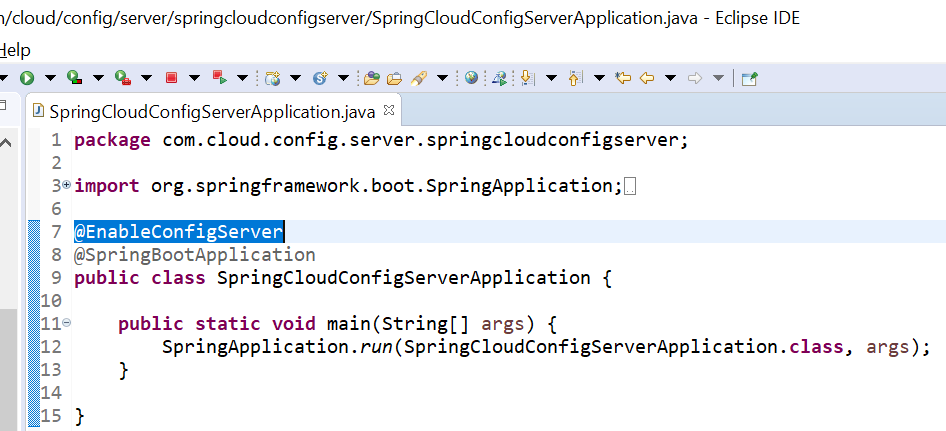


Import in eclipse and change properties files:



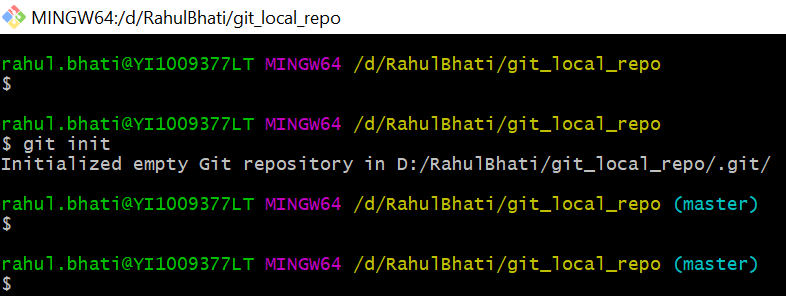


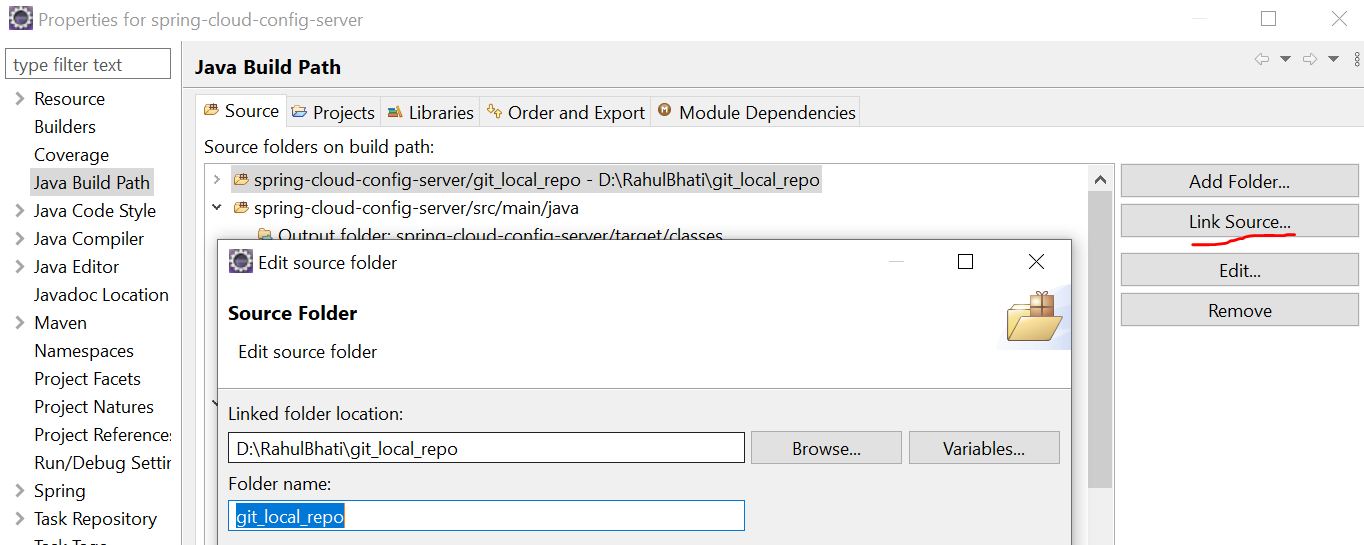
Enable Config Server -> @EnableConfigServer

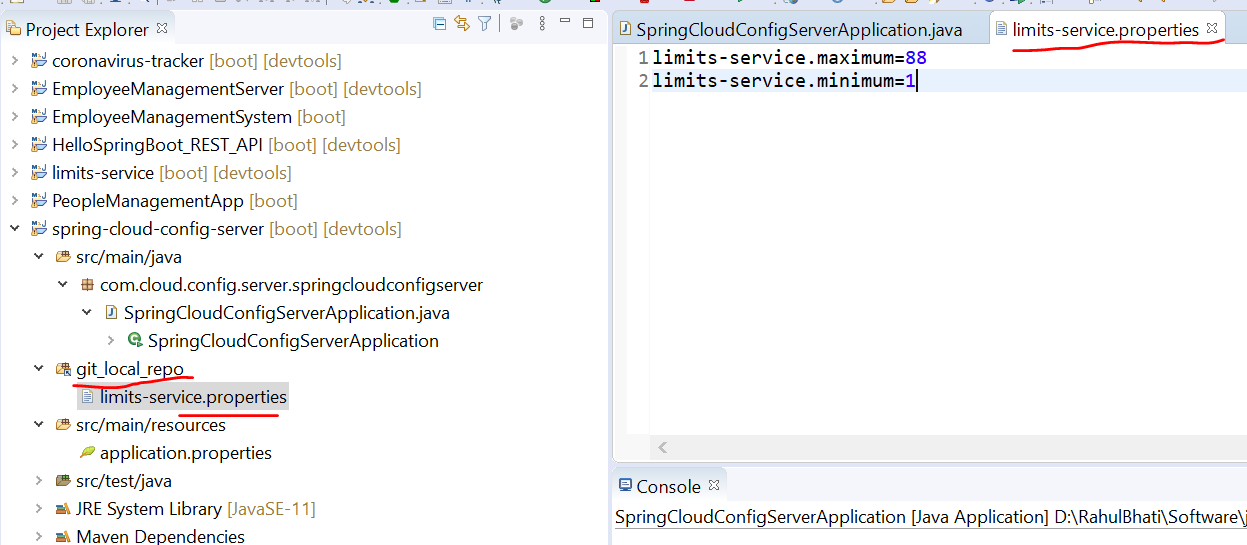


First we will read configuration file from git local repository.

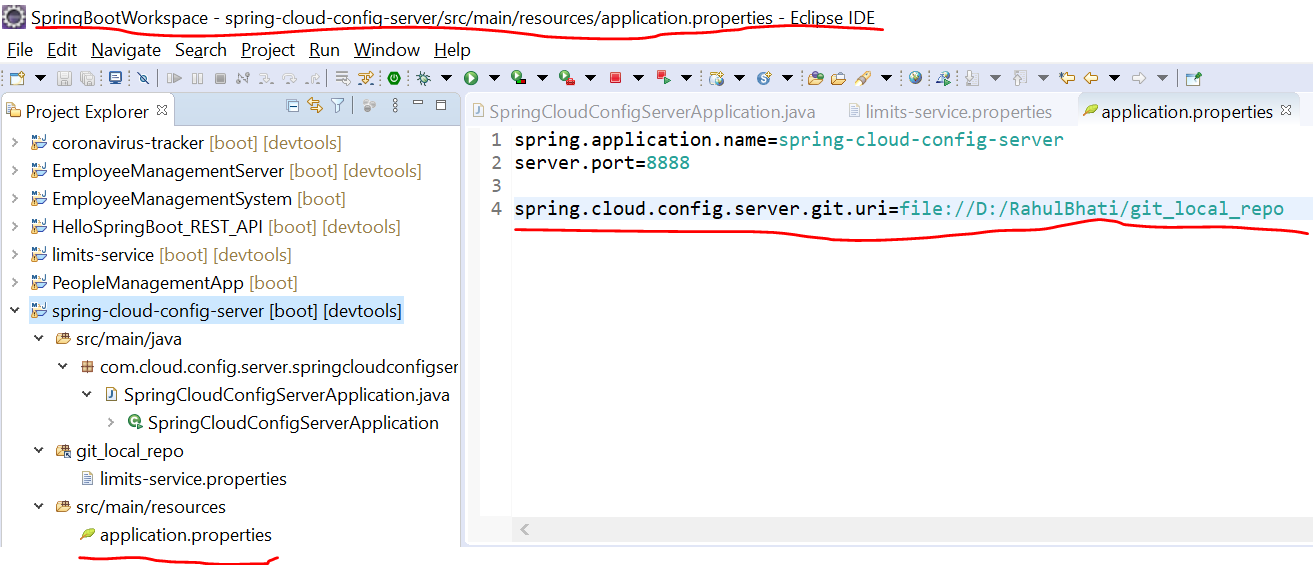
We will Link source git in “spring-cloud-config-server”:

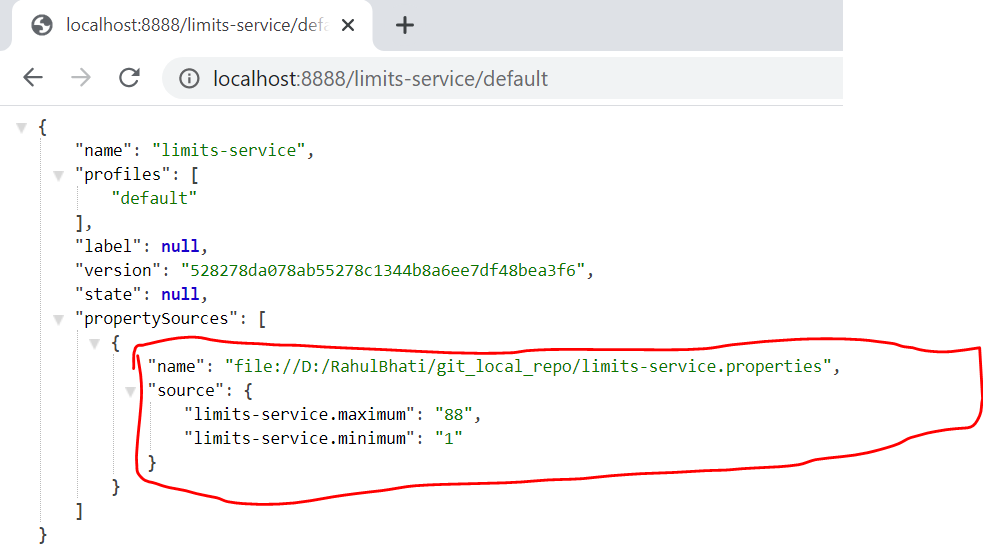




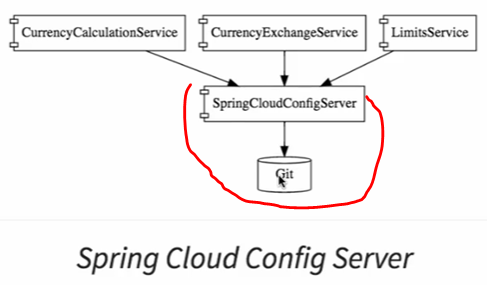


**Note: Repo properties file name should be your microservice name**.



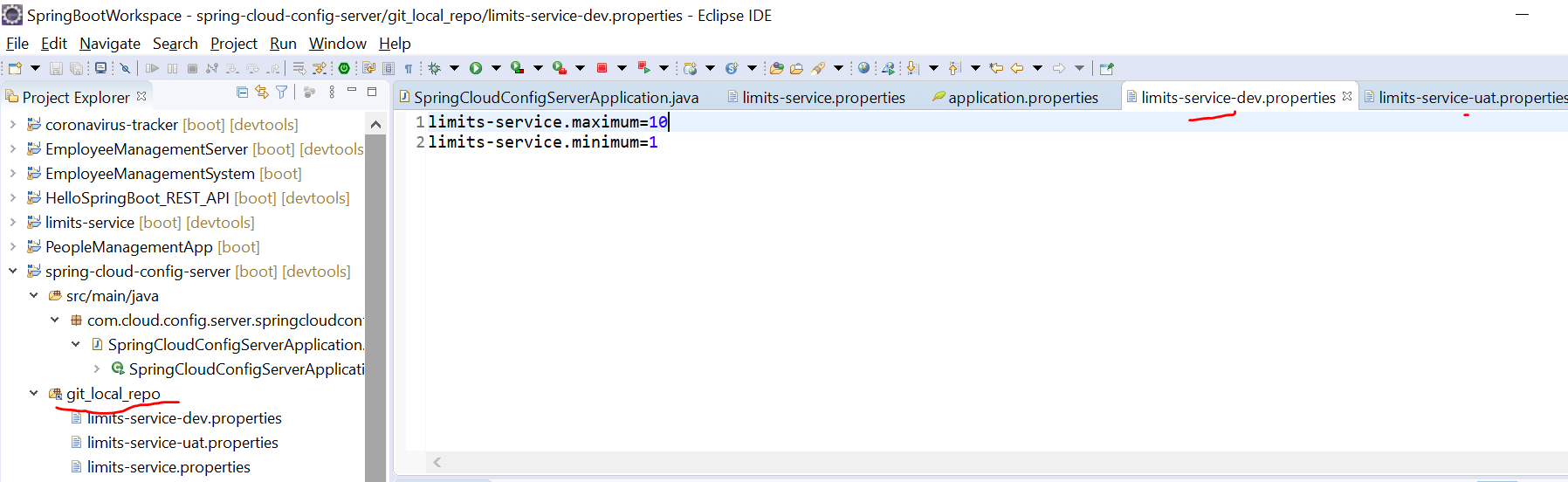


We have established git and SpringCloudConfigServer:

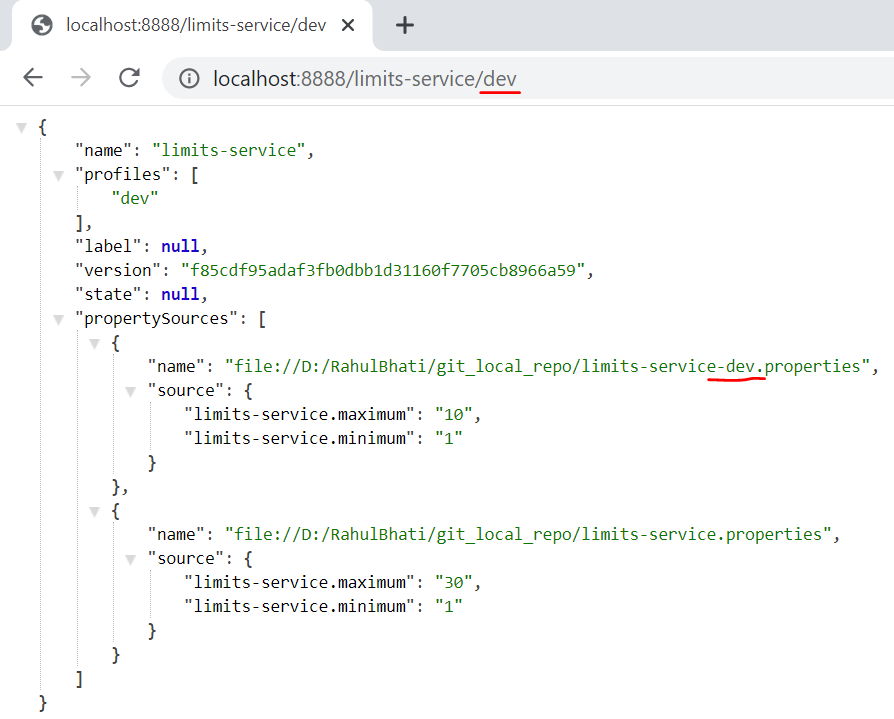


Now we will configure different environment for LimitService.

We will create dev and uat properties for envi.

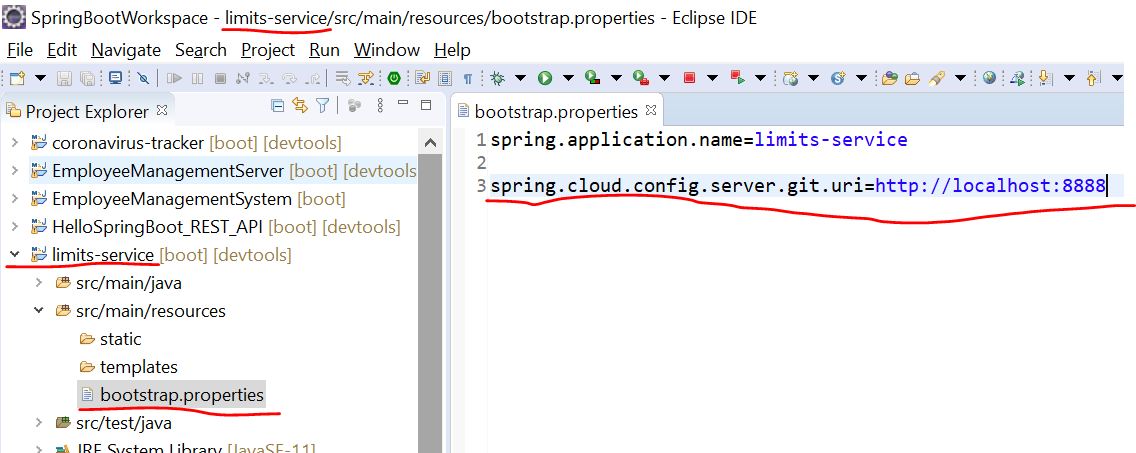


You can see properties values using URL:

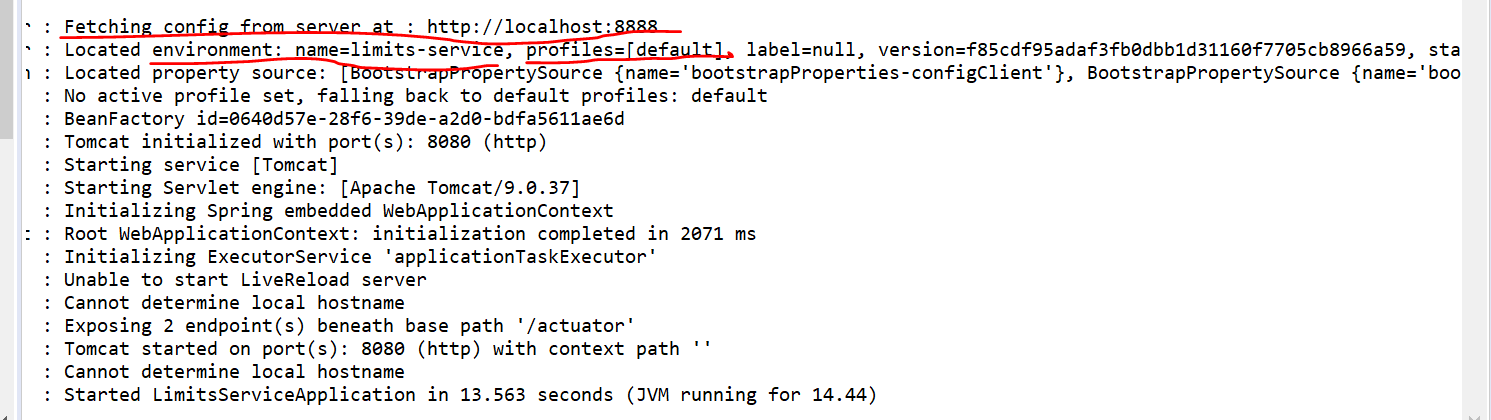


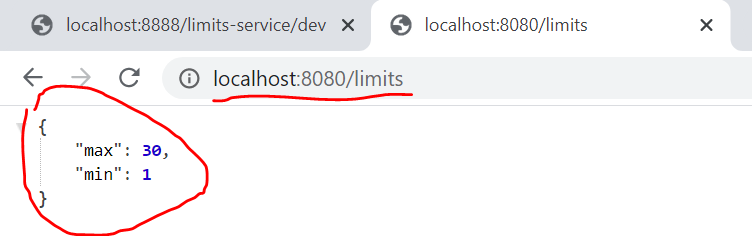
Now We are going to pick configured properties data “Limits-Service”(limit microservice) from “Spring-Cloud-Config-Server”.

Noted- when we need to pick data then we need to changes properties file (Limits-Service) to bootstrap.properties. Like:



When you run the limits then max and min data will be picked from spring cloud config server->git local repo… here we have not set any “Profile (environment file)” then it will pick default profile.





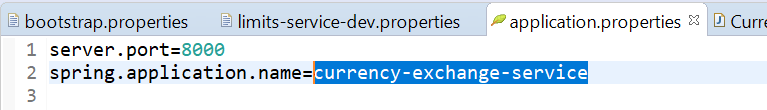
It’s reading default profile.

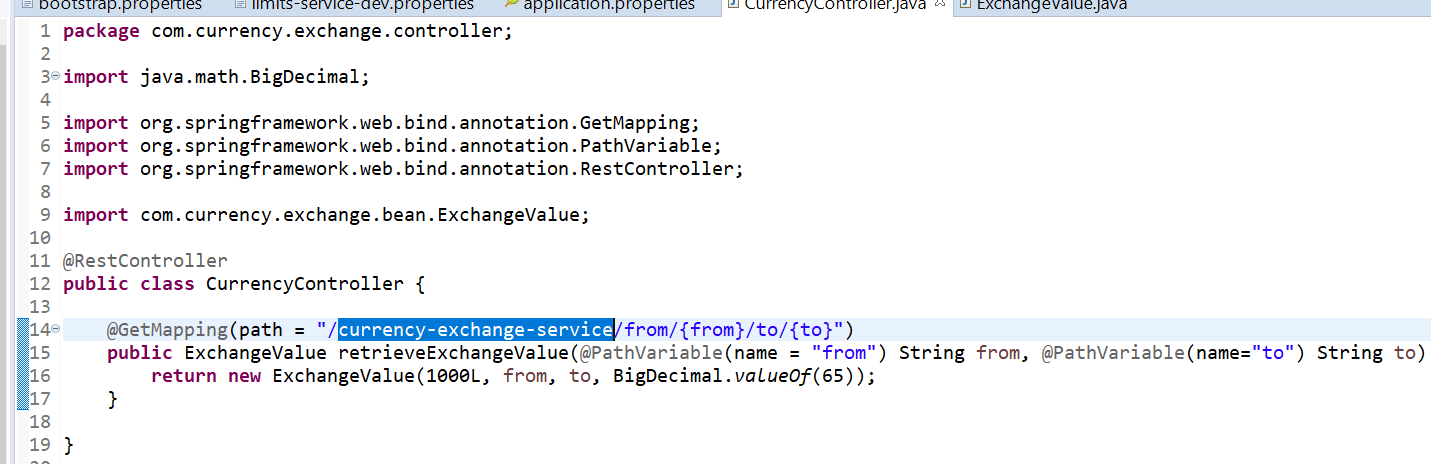
**Configure Profile for Limits Server:**

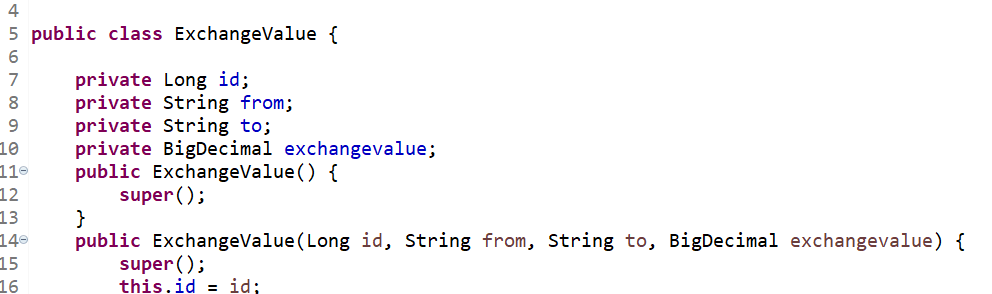


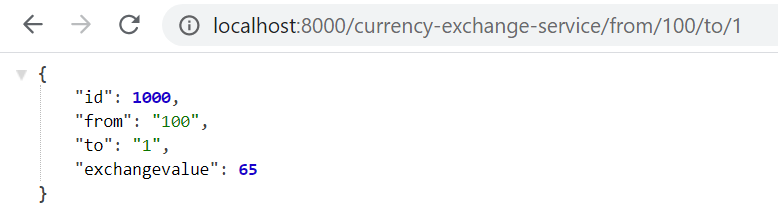
**Setting Currency Exchange Microservice:**

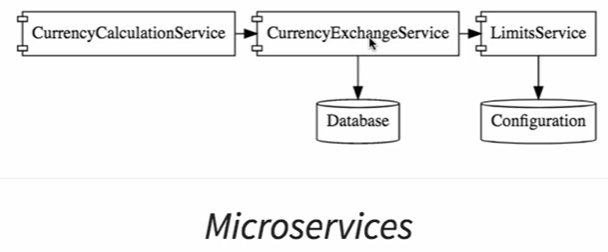
## Microservice-#3 - > Currency-Exchange-Service



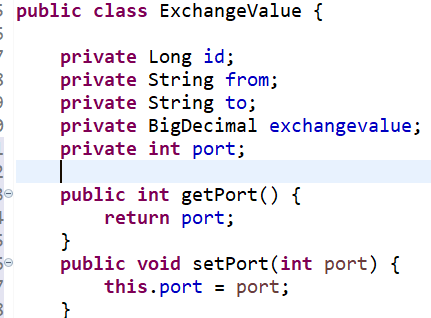


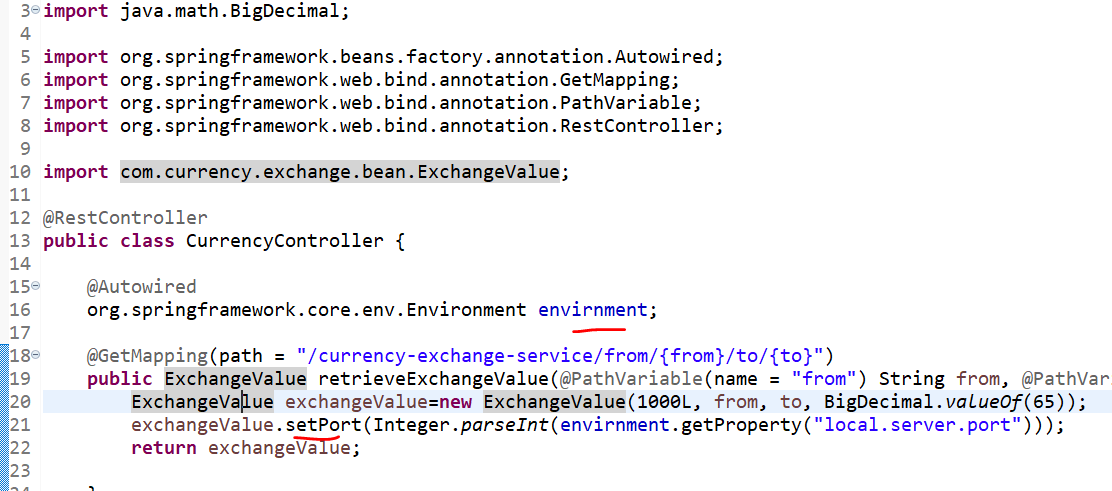


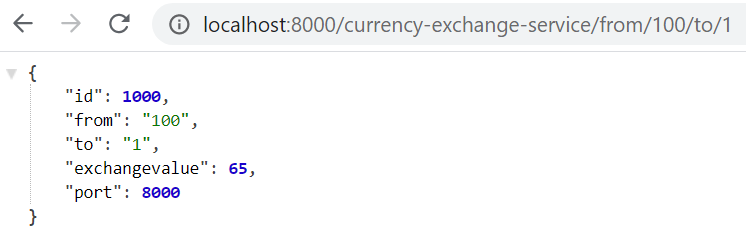




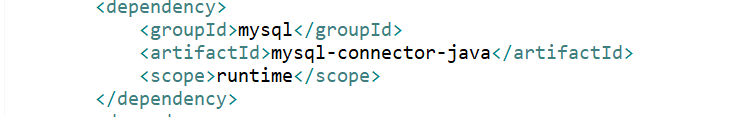
### Now we will dynamically setting Port in response:



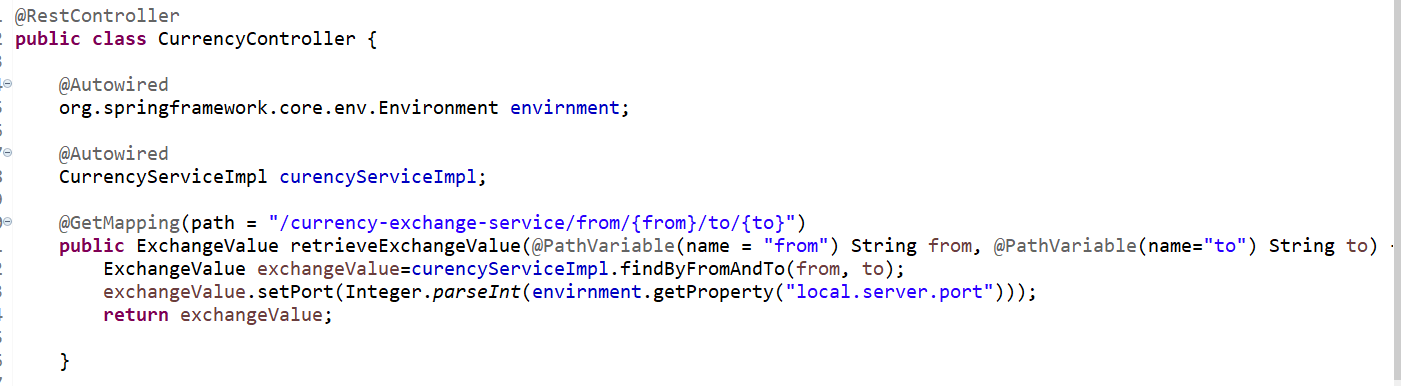


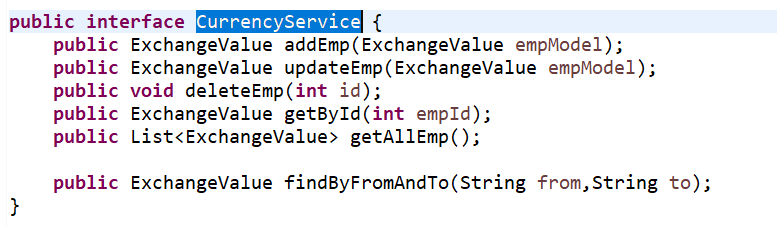


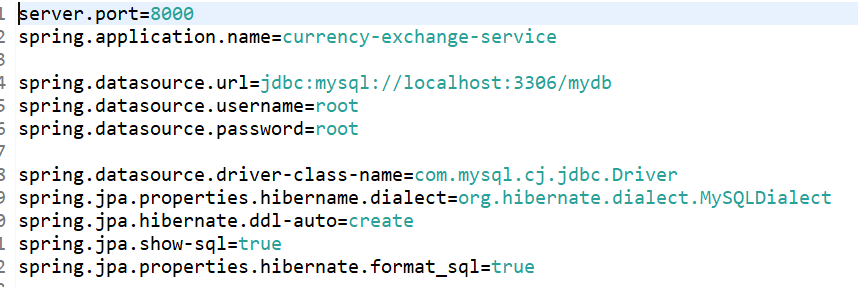
### Configure JPA and initialized data:

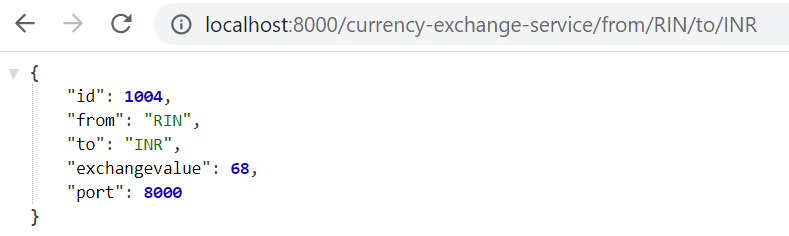




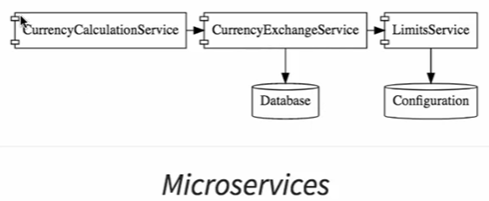




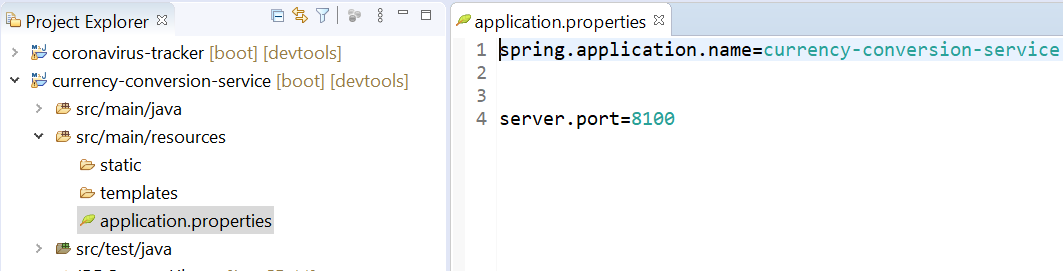


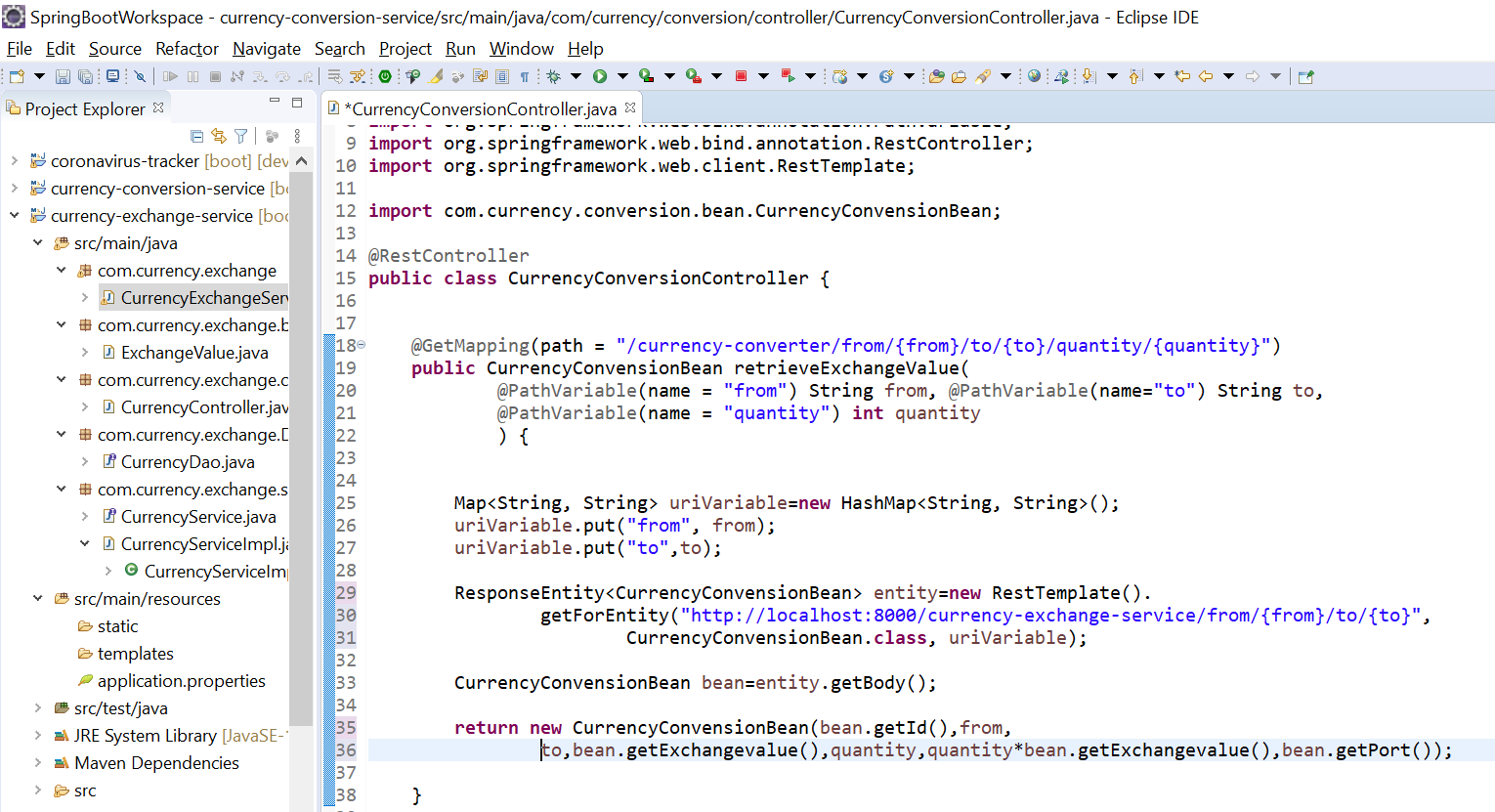


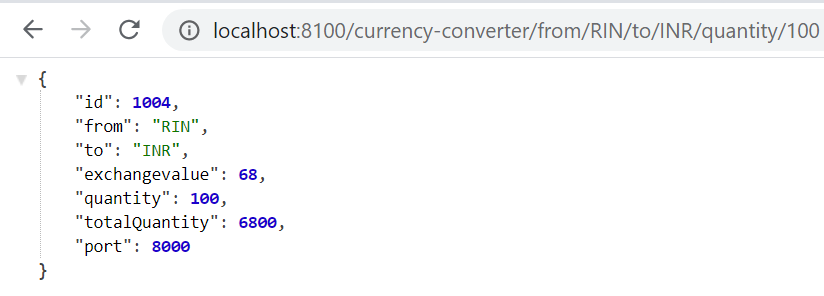
## Microservice-#3 - > Currency-Convension-Service:



Now we will call “CurrencyCalculationService from CurrencyExchangeService.







# Feign:

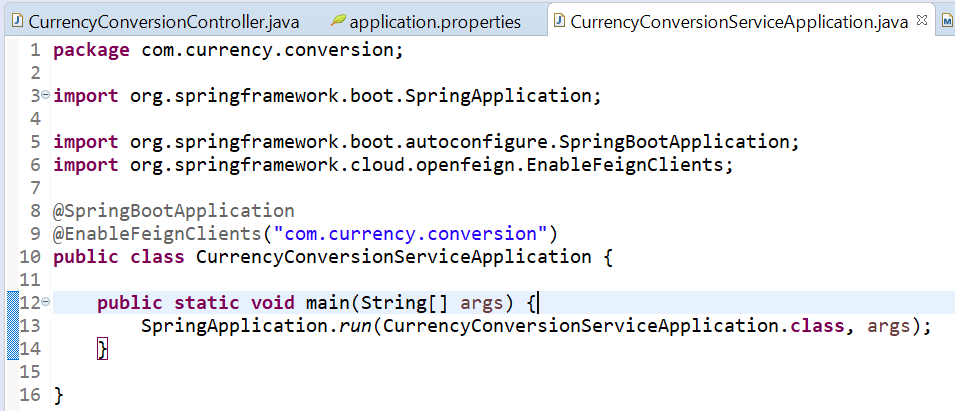
There are lots of ways to call other services. So, to call other service we should use easy way to solve this.

So, for this one of the service is Feign. Feign is very easy to call other microservices (restful services).

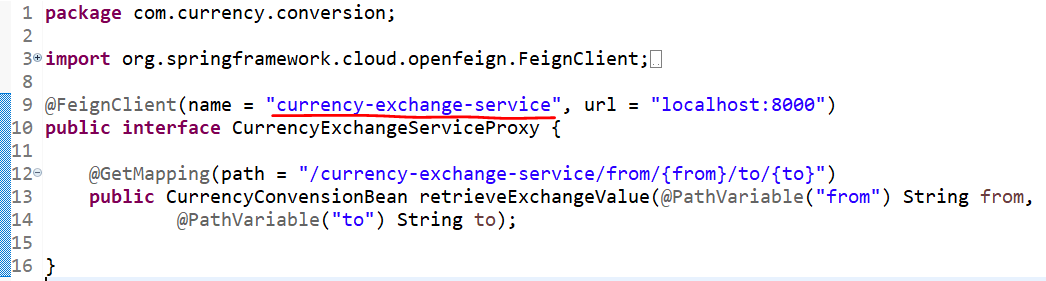
It provides integration with **Ribbon** which is the client-side load balancing framework.



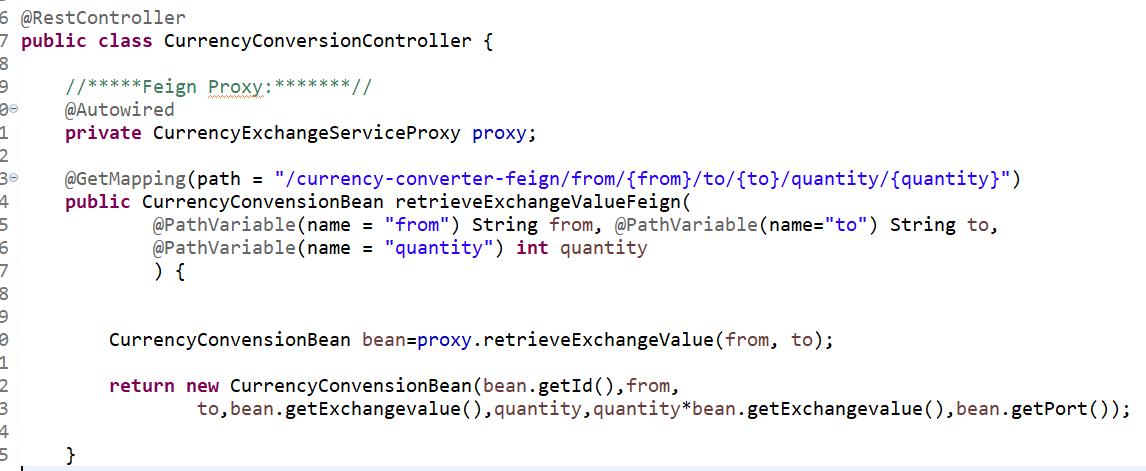
To enable it:



Now we will create feign proxy, to call other microservice.



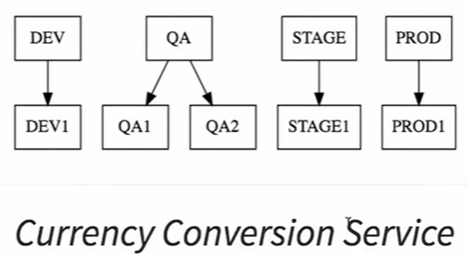
Way to call from controller:

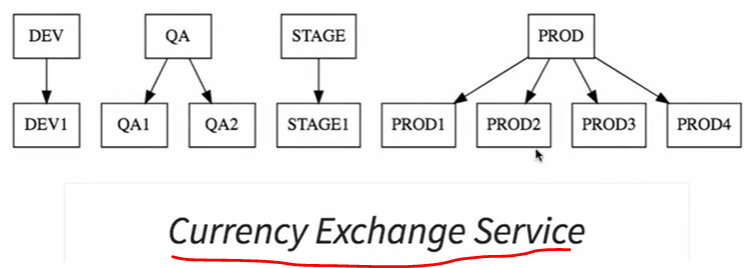




Now we will take for Environment related:

Suppose we have below things:



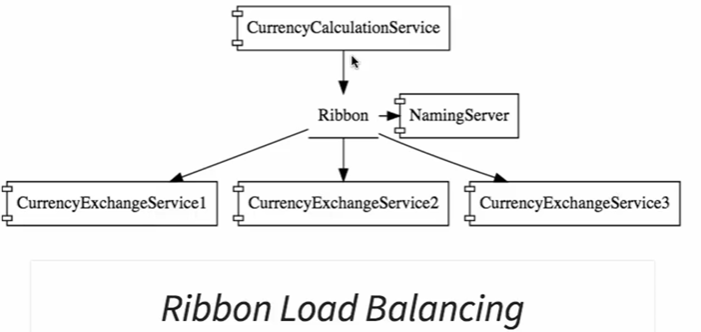


We want to distribute load between different instances(prod1,prod2,prod3,prod4)

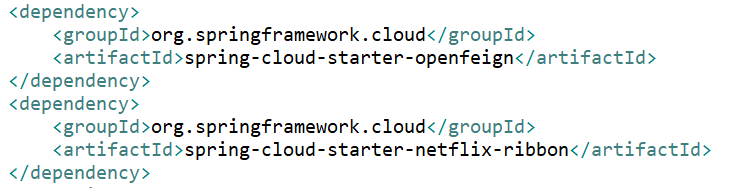
What is why “Ribbon” comes for this.. load balancing.

# Ribbon:

For load balancing we use Ribbon. In above feign we are calling currency exchange service from currency conversion🡪 now we will add Ribbon to call currencyexchangesservice instances based on the load.

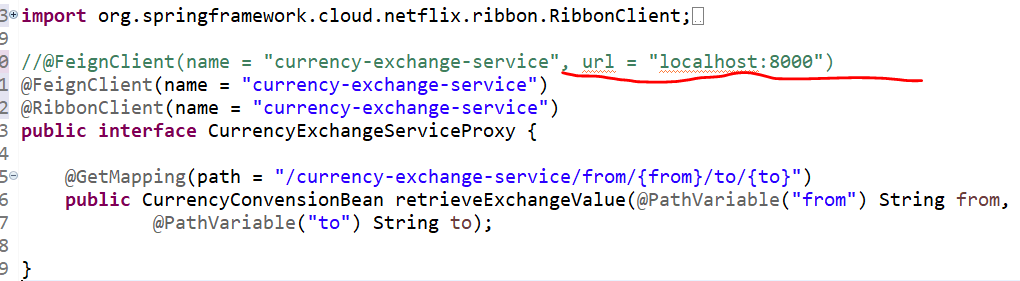


Pom.xml:

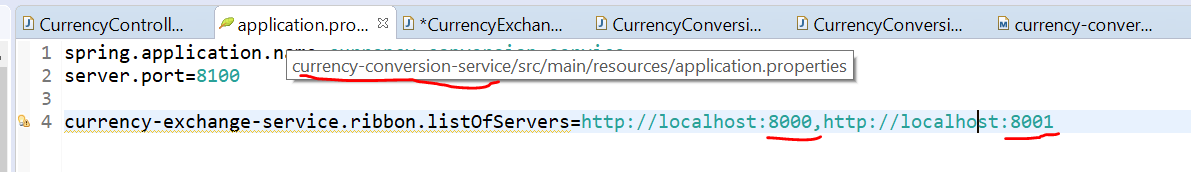


After add ribbon we need to Enable Ribbon in Fegin proxy:

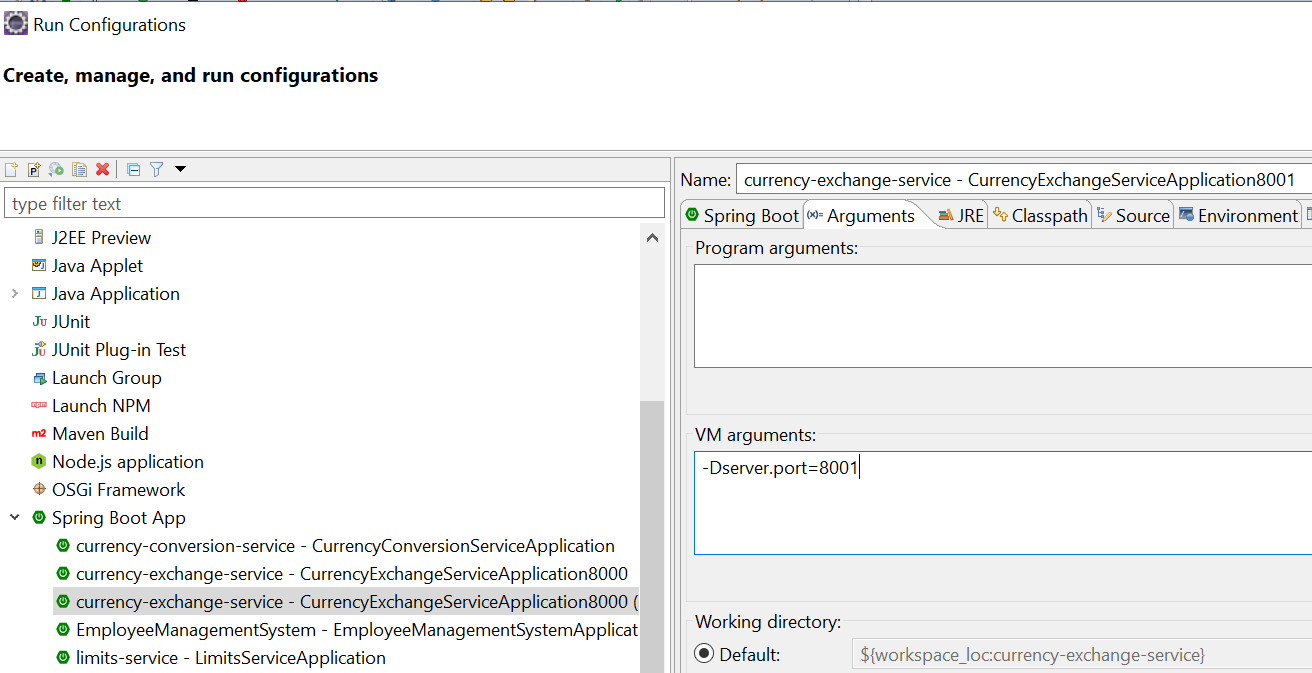
And for Feign Client we don’t need to declare URL, because we will confirm in different way, means we want to distribute load between multiple instances. What we will configure in application.properties file.



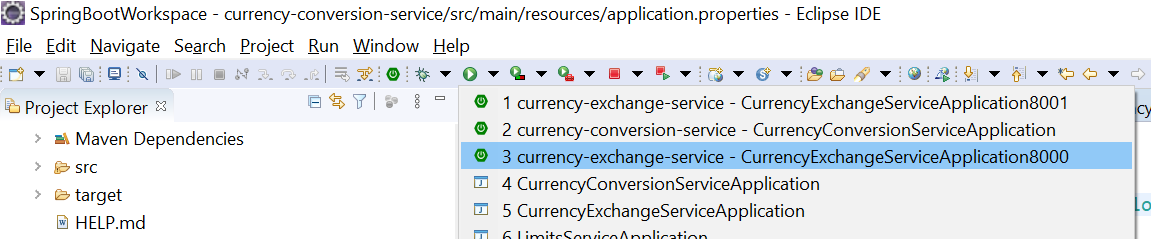
We will distribute services in two instance like:

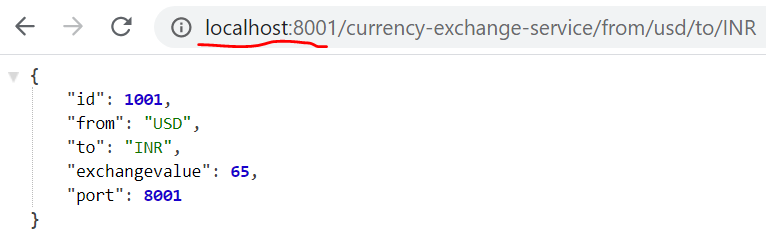


We will start exchanger server instances 8000 ad 8001🡪like:

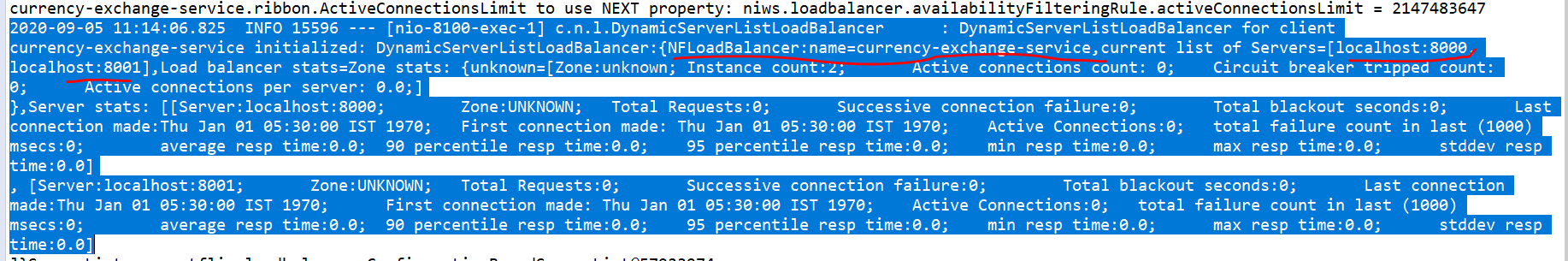


Now we are running 2 instances:





One you run the Conversion service you can see load balancing instances:



You can see load balance instances in below port : refresh and see data





Means🡪 ribbon now distributing load between two active service instances. Whatever call make using ribbon through the conversion service are being distributed between instances.

Note: Suppose we want to add one more instance of exchange service(8002) and we need to edit properties files.

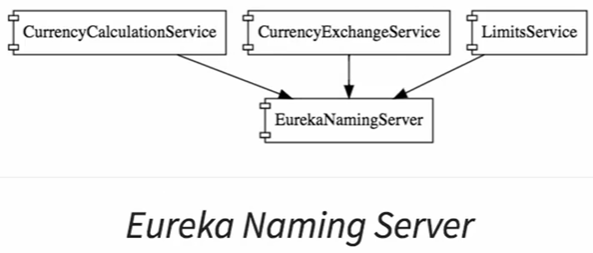
So, I need to change my configuration file whenever I want to add instances and that is not really good.

So what we want, based on the load, I can increase or decrease number of instances of the services.

Means dynamically increase and decrease instances.

If I keep changing the CurrenyCalculationService based on the how many exchange services. It become are very “very difficult haddock”.

That’s why “Name Server or Naming Server” comes.



If currenyExchangeSerivice running on URL XYZ. Don’t want to hard code on currencyCalculationService, because there may be currencyExchangeService instance is coming up and going down thinks like that.

Other then that we have a “Naming Server” 🡪 All microservice instances are registered with the NamingServer.whenever microservice instance comes in it will register into NamingServer or EurekaNamingServer🡪 this is called Service Registration.

And whenever services what’s to talk to other service like:

CurrenceyCalculationService-🡪 ……………. 🡪 CurrencyExchangeService

Suppose CurrenceyCalculationService what’s to talk with CurrencyExchangeService then it will talk to NameServer and ask to Nameserver that what are the instance of CurrencyExchangeServer that are the currently running. 🡪 This is called Service Discovery.

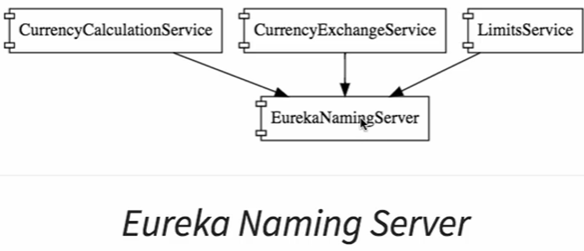
Two points

1. Registration 🡪 To 🡪 NamingServer (EurekaNamingServer)
2. Service Discovery 🡪 once service what to know other service currently running instance.

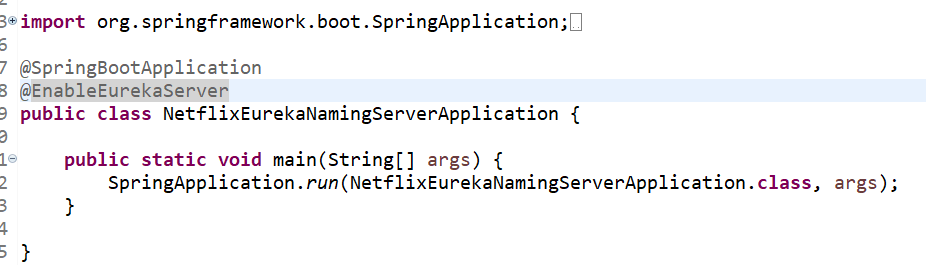
# Eureka Naming Server:

Three things we will do:

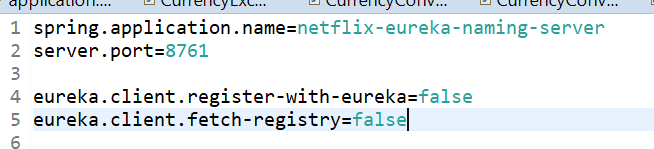
1. Create Components for Eureka naming server.
2. Update CurrencyCalculationService to connect Eureka naming server.
3. Update CurrencyExchangeService to connect Eureka naming server.



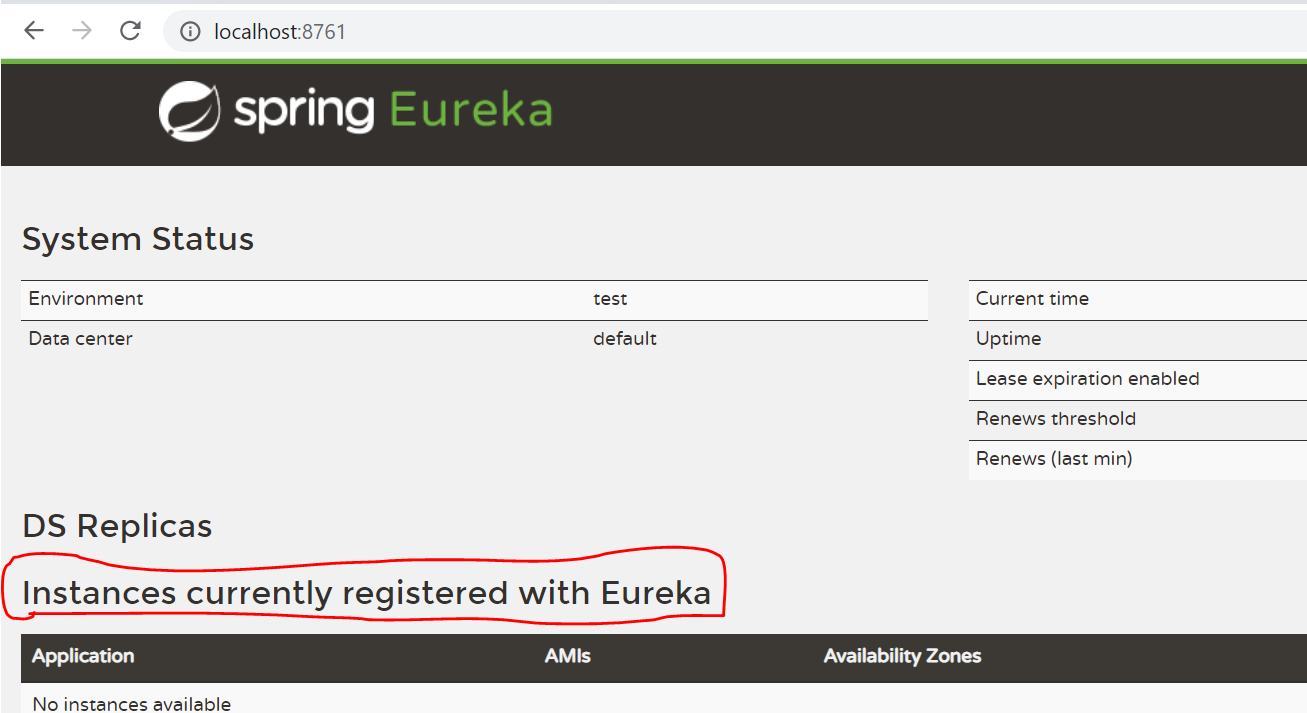
Enable Eureka server:



Property file:

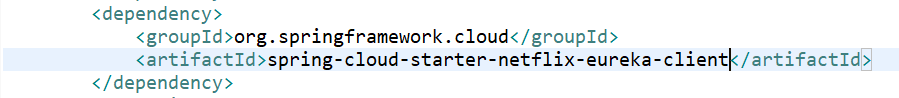


Run and see the Eureka Server Home page:

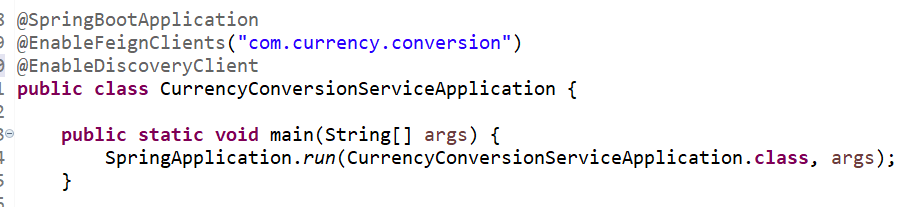


Now we will connect CurrenyExchange and CurrenyConvresion Service:

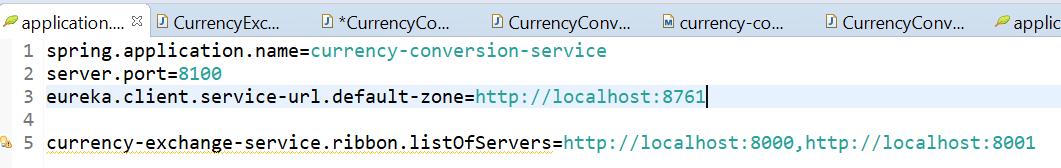
Conversion Service Pom.xml:



Enable EurekaClient:

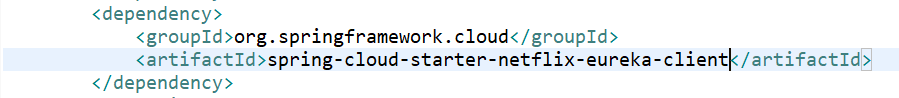


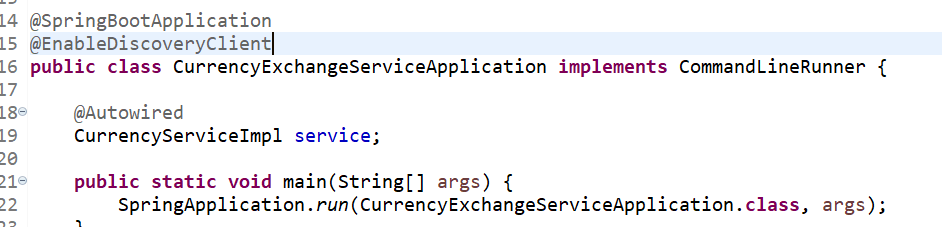
Register our service with eureka server in properties file:

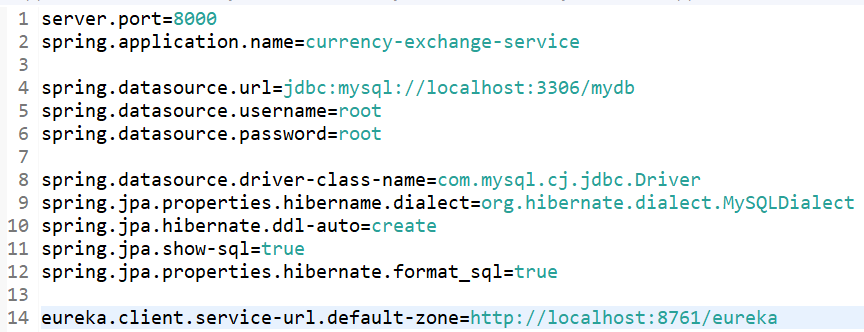


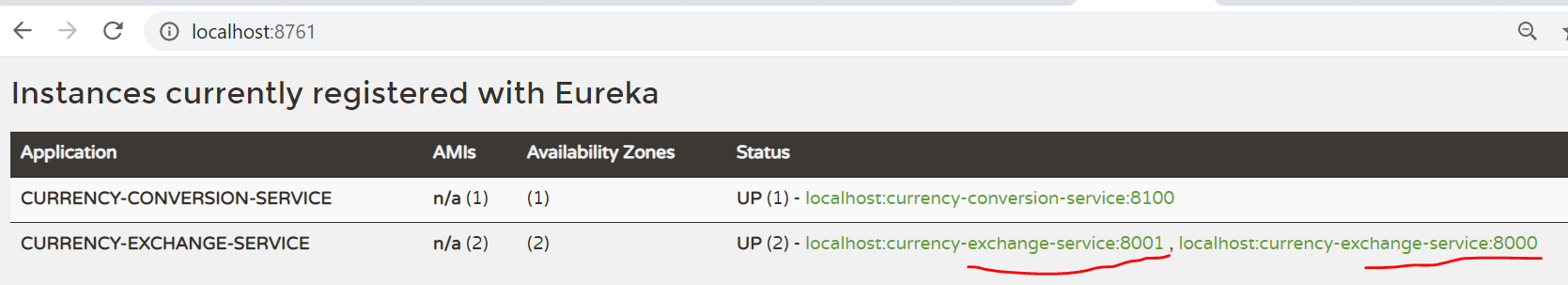


**Now we will configure Exchange Service:**



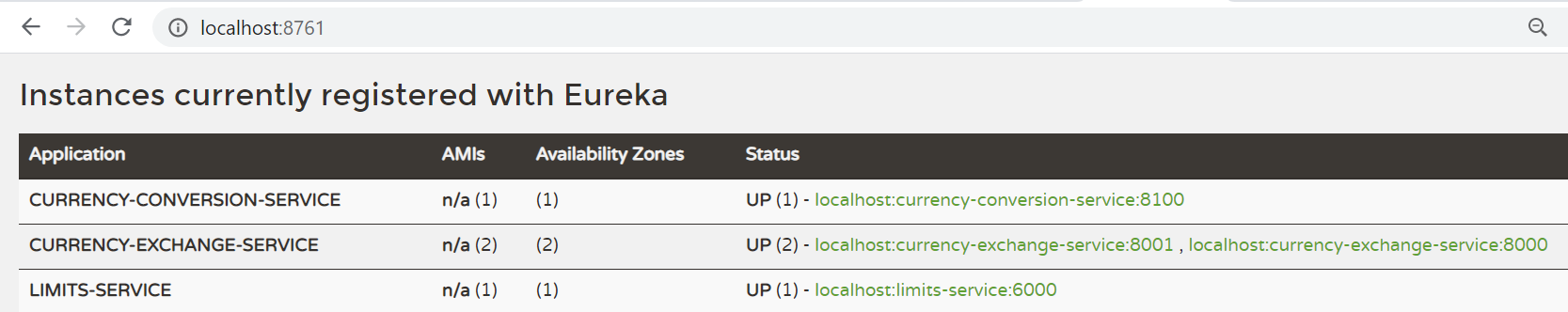


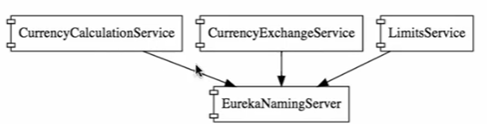




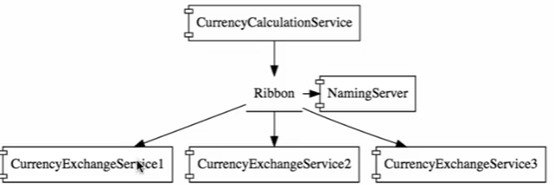
Now one instance of conversion service and 2 instance of exchange service are registered with eureka naming server.

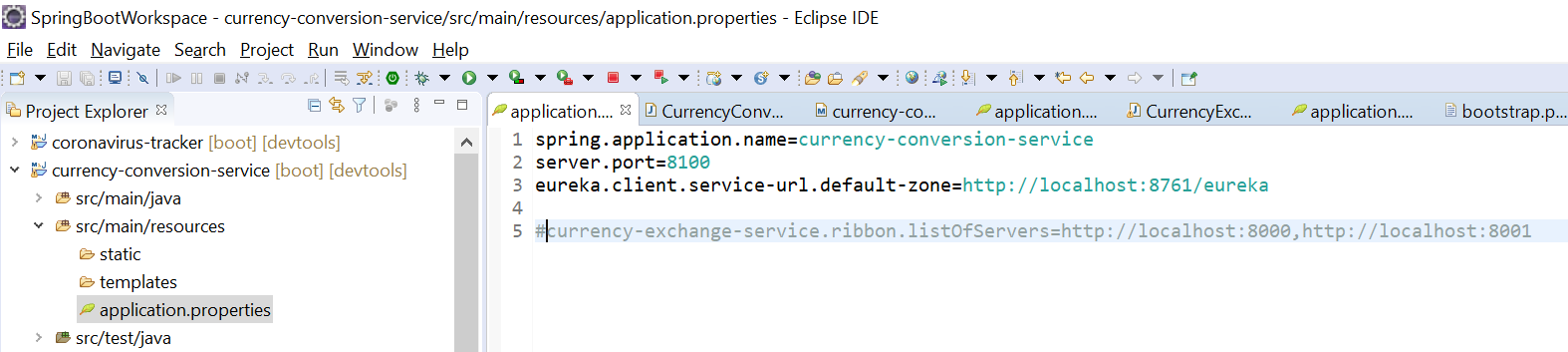
Now do the same for limit service:





Now what we want to do, to enable Ribbon to talk to Eureka naming server.

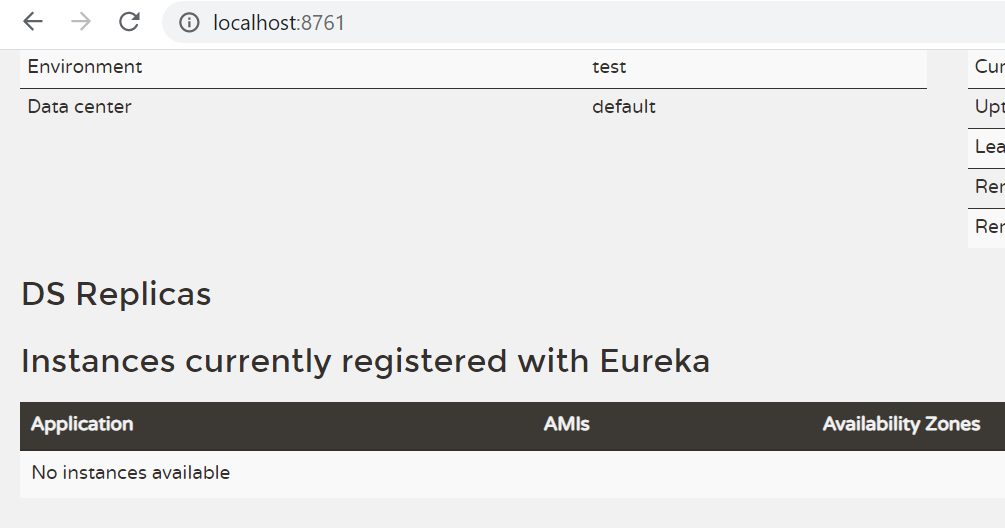




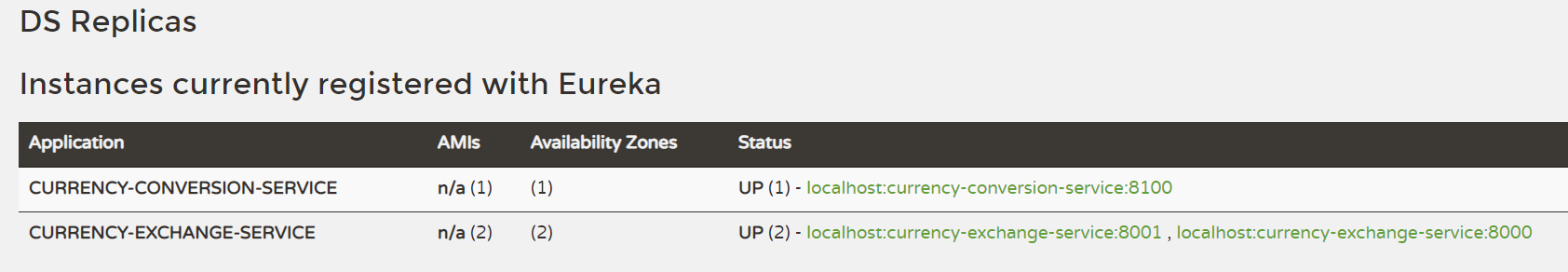
In this only thing we are doing is to removing hardcoded listOfServices removing.

And try up eureka with all the services and check…

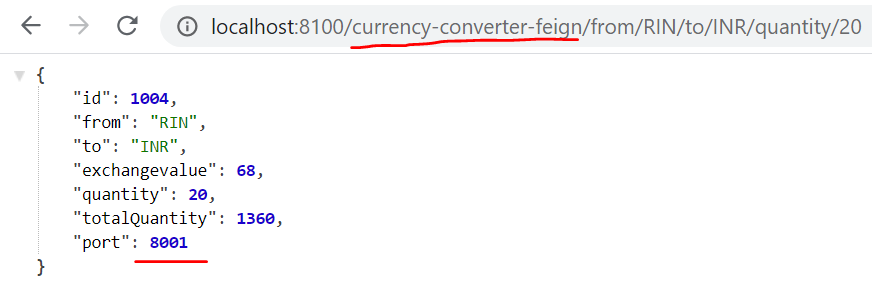
None of the services are running in Eureka:

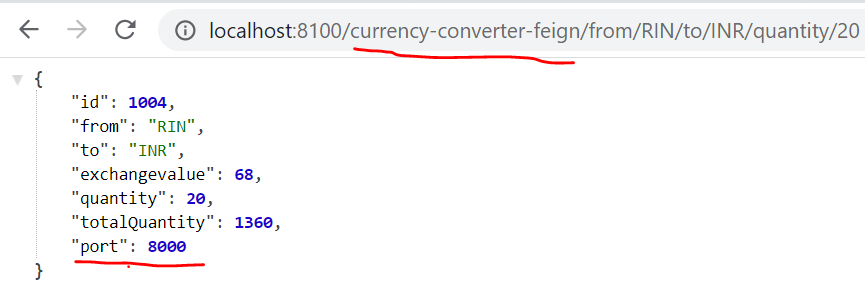


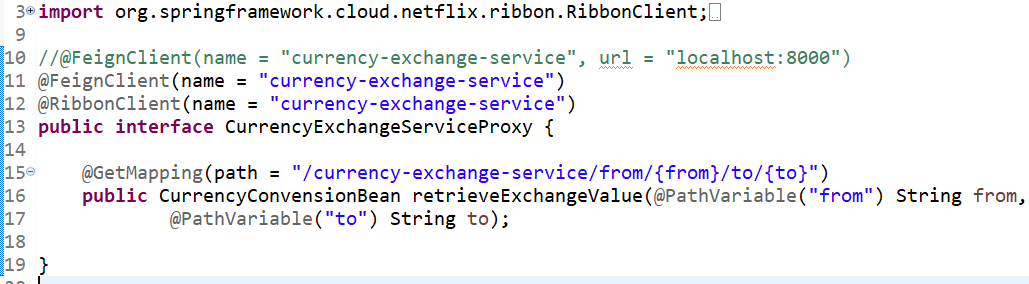
Now one by one up all the instances of the eureka:



Now we will check all instances and services are running :

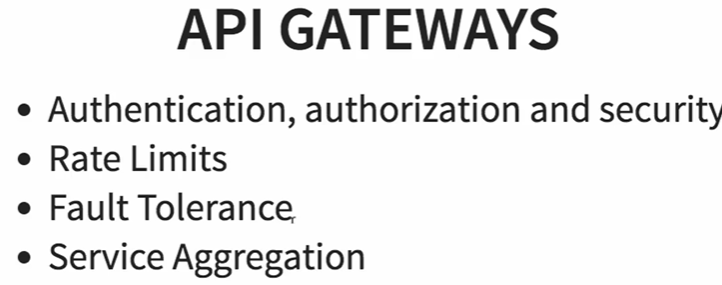






# API Gateways:

Common features of the all services marge:



This API’s provide common features that are common for all the services like, authentication, rate limit, fault tolerance(if data null then send default value),service aggregation.

API gateways are also good for debugging and analytics .

## Zuul API Gateways:

Netflix provide implementation is called Zuul.

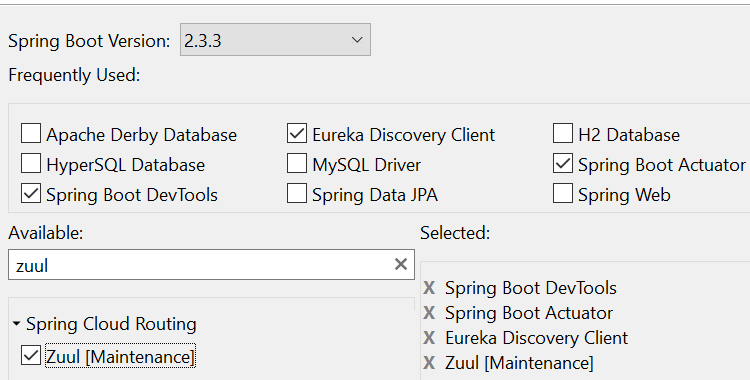
There are three steps to setting up Zuul API Gateway Server.:

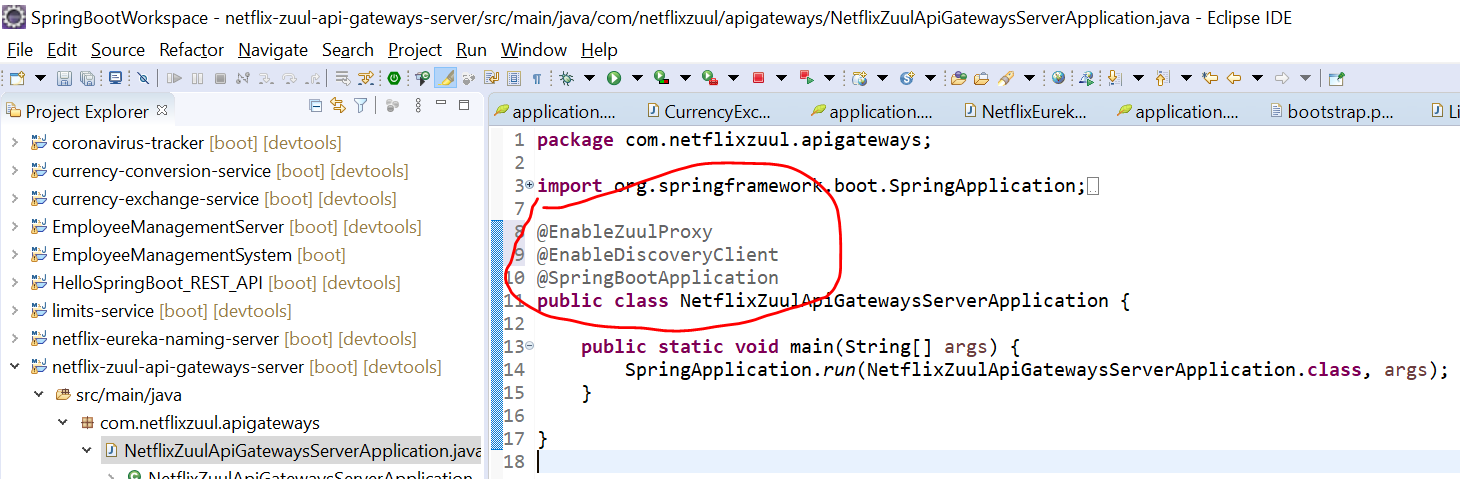
1. Create Components for it.
2. What it do when it intercept with request, means when we call service 1 to service-2 then what we need to do.

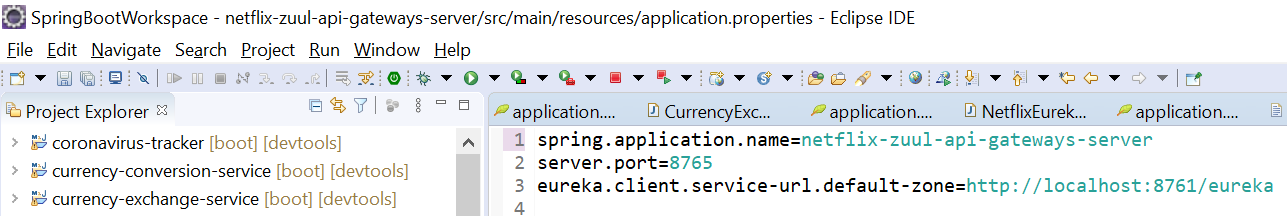


1. All the api should be configure to the Zuul.

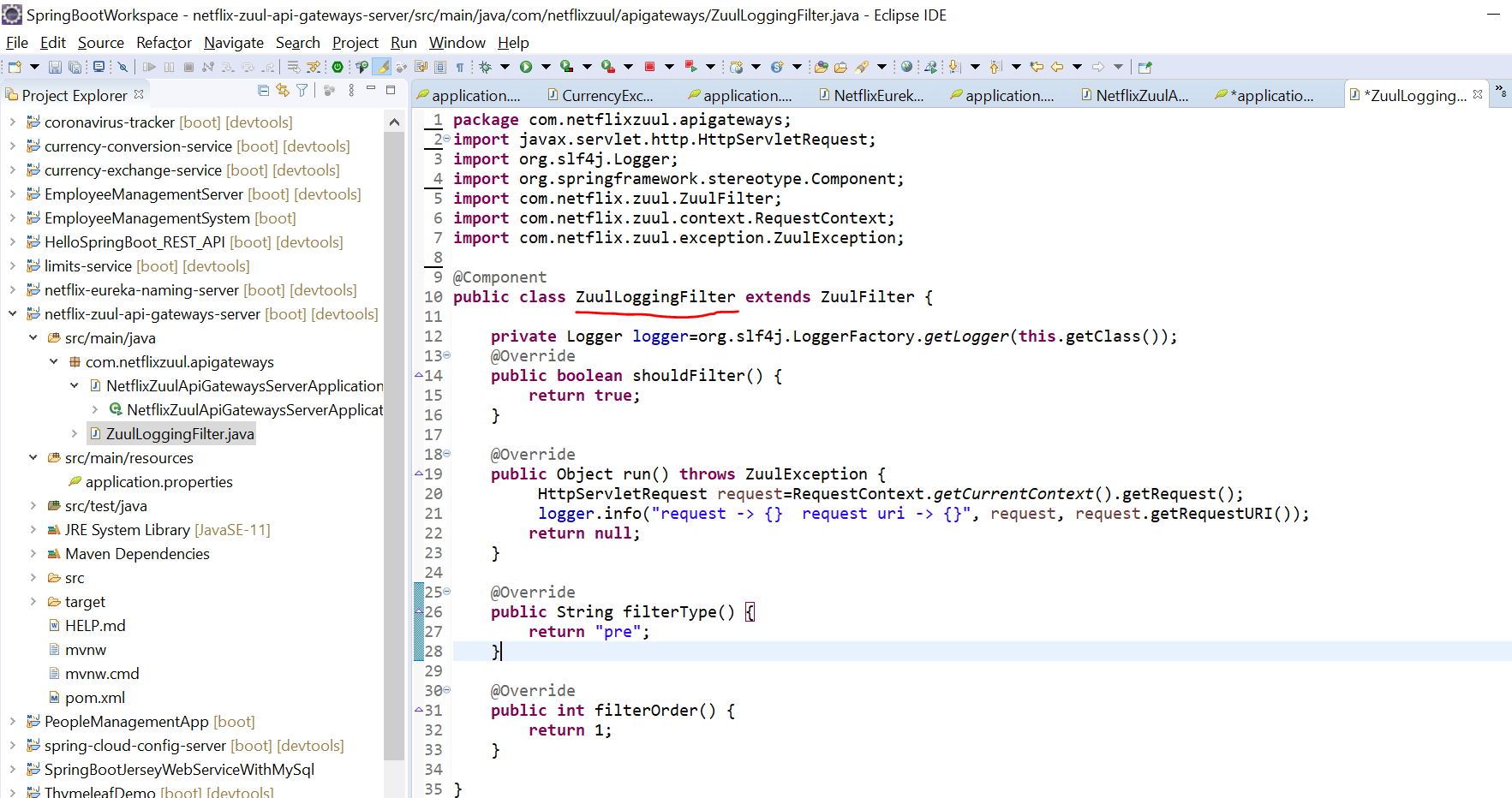
### Create Zuul Server:





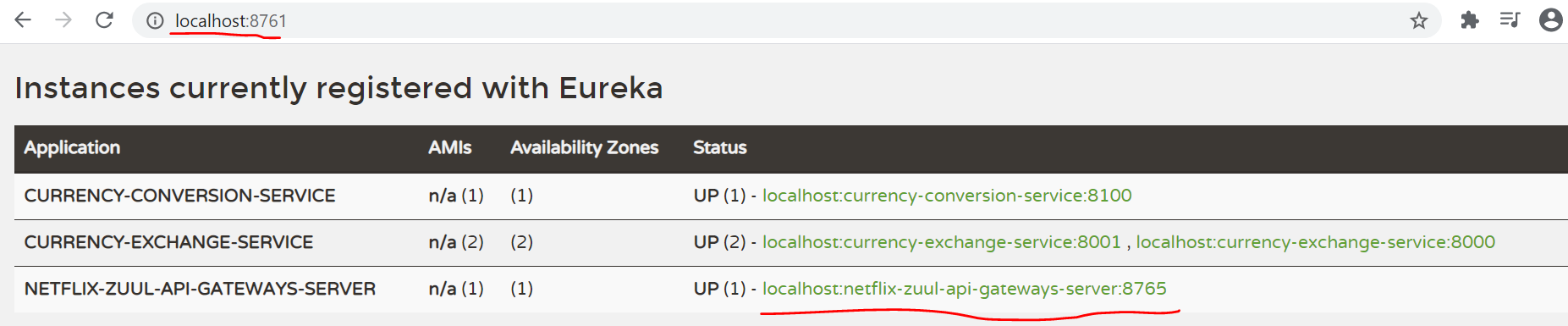


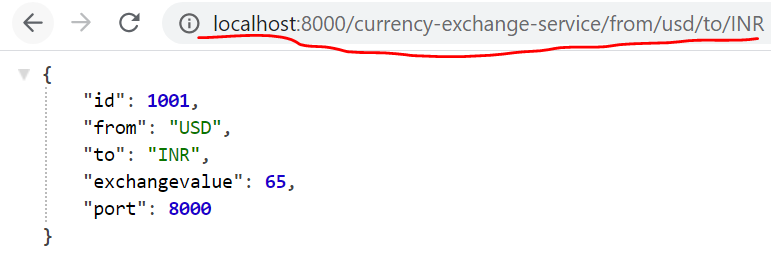
### Implement Zuul Logging Filter:



### Executing a request through Zuul API Gateway:

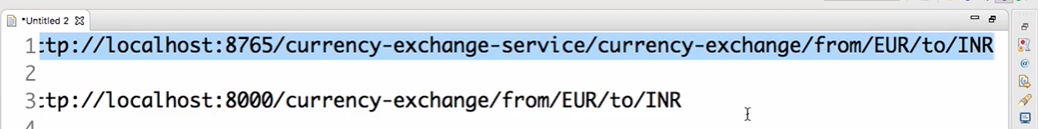
Run Netflix server and see the result.





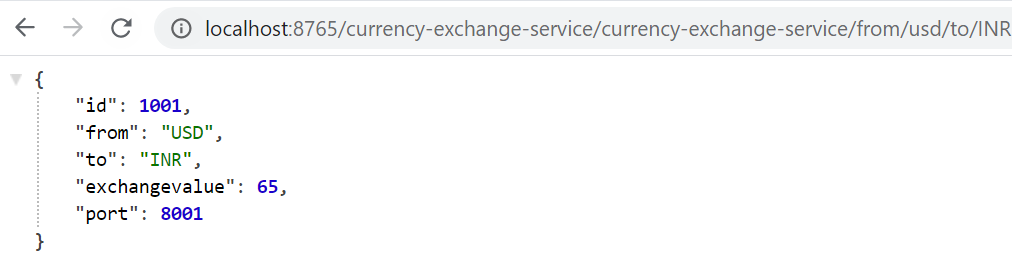
**Now we are going to execute “currency-exchange-server” using Zuul AP Gateway:**



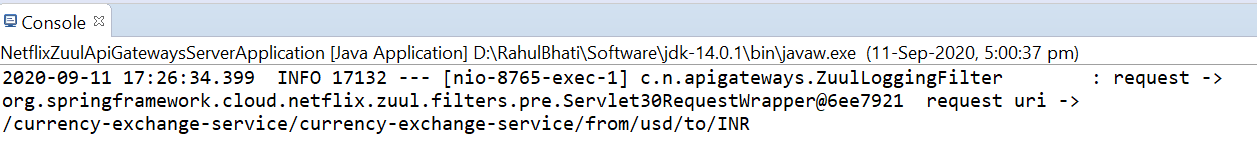


**Means:**

<http://localhost:8765/currency-exchange-service/currency-exchange-service/from/usd/to/INR>

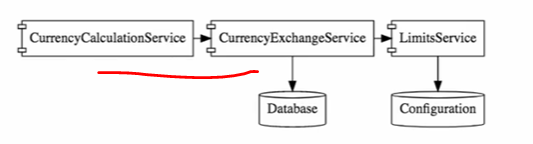


**Console:**

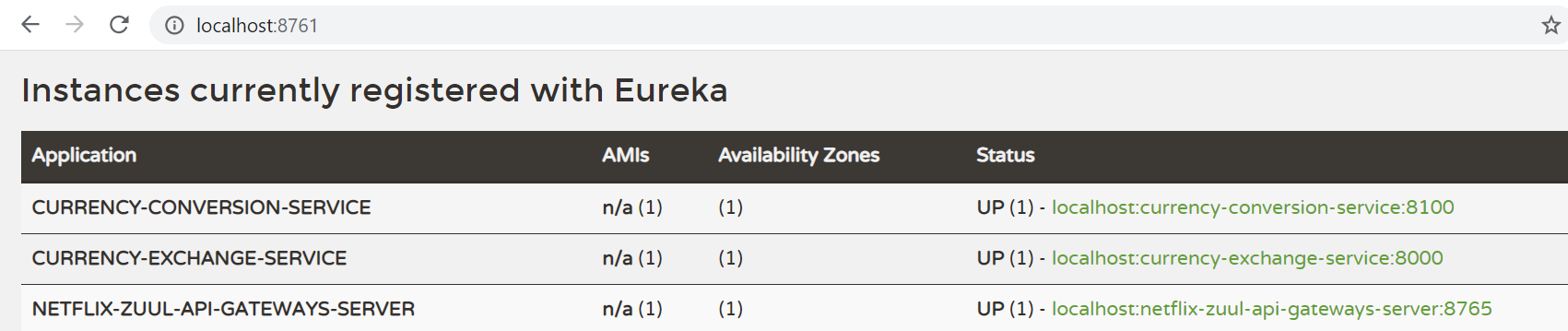


### Setting up Zuul API Gateway between microservices innovations:

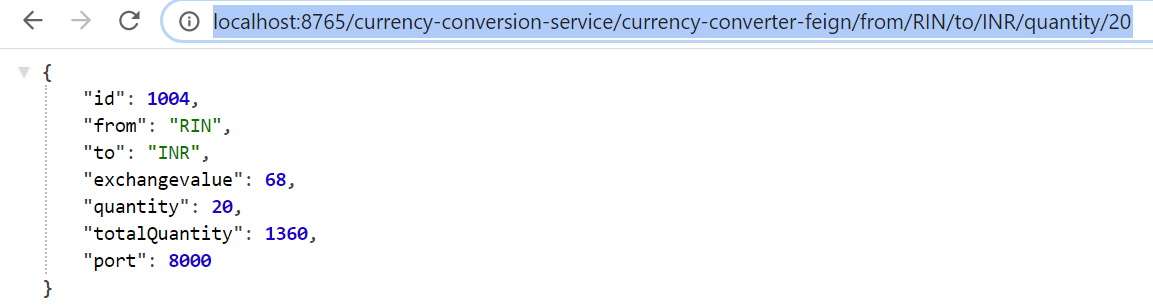
Between “currencyCalculationservice and currencyExchangeService” we will use Zuul Gateways.







<http://localhost:8765/currency-conversion-service/currency-converter-feign/from/RIN/to/INR/quantity/20>



# Distributed Tracing:

When multiple services are running and we want to debug it, then where we need to check if services are not running, means in which services we need to debug.

For this “Distributed Tracing comes into picture”.

We need centralized system where we can check all debugging details.

There are many “Distributed Tracing” options are there but we will discuss “Spring Cloud Sleuth” using Zipkin.

Noted : Zipkin is distributed Tracing System.

Important this is to assign a unique “Id” to the all the microservices to a request.

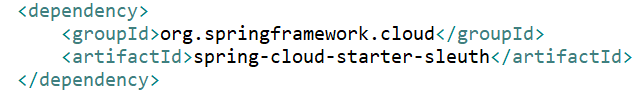
## Implementing Spring Cloud Sleuth:

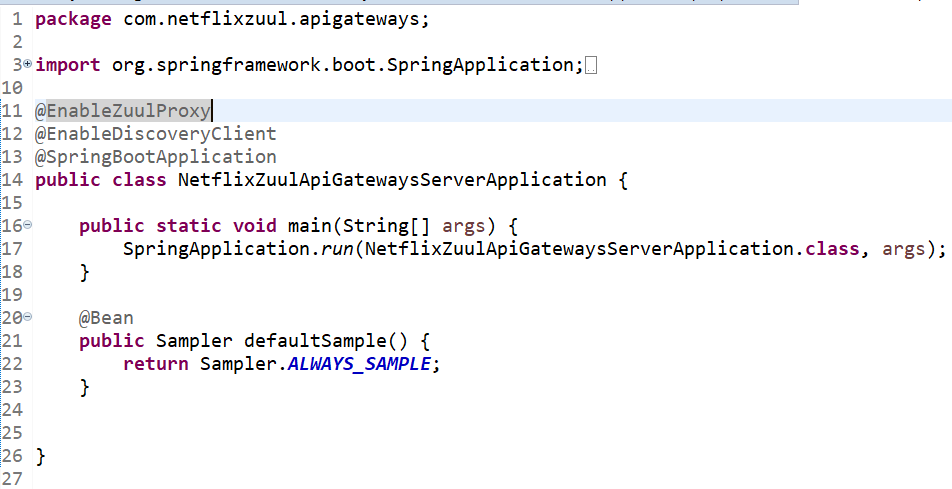
This is assign a unique Id to the microservices.

First step is decide where we all need to do spring cloud sleuth:

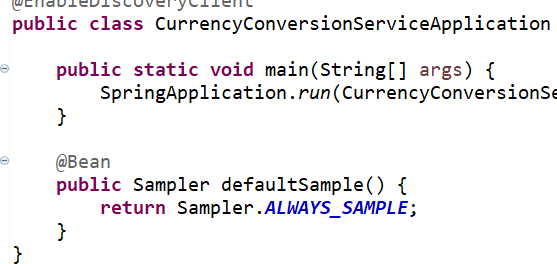
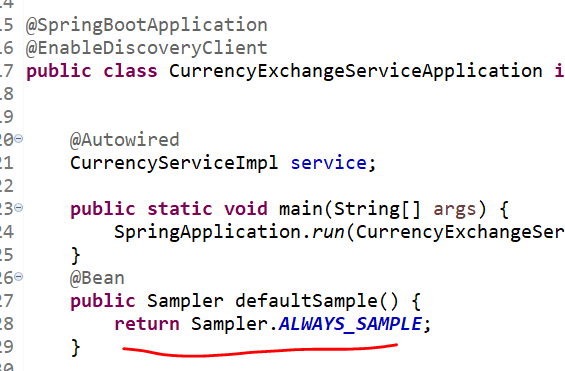
Here we will configure in – conversion, exchange and api-gateway microservices.

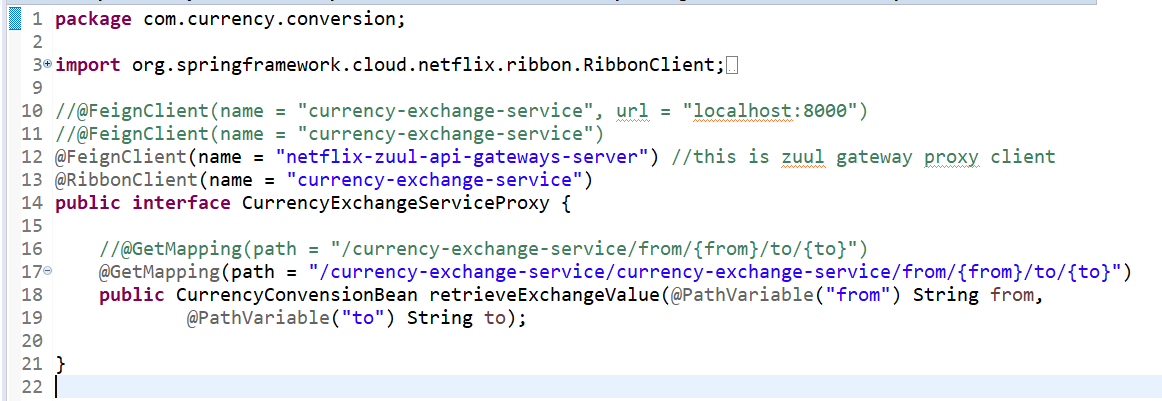
There are 2 steps- adding dependency in pom and what trace all the request.

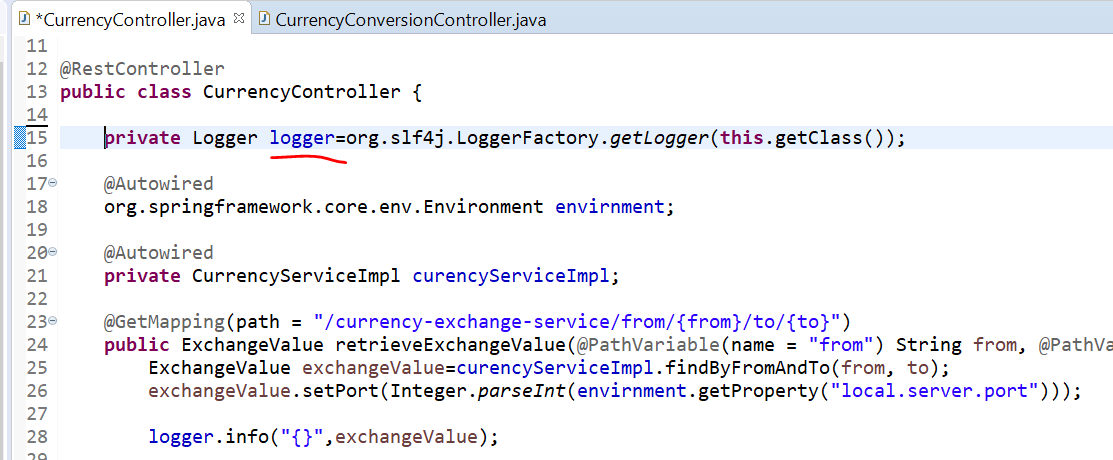


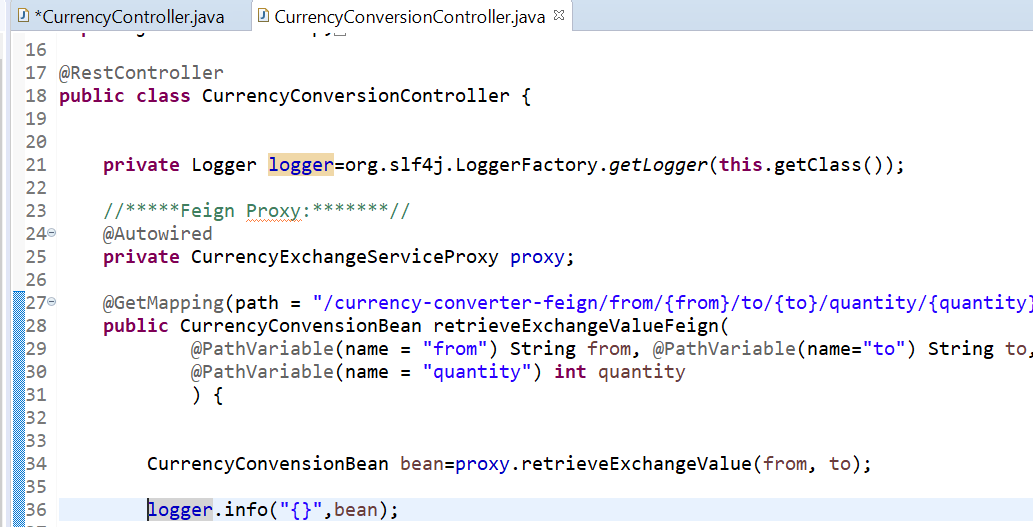


Same Setting for conversion and exchange service:



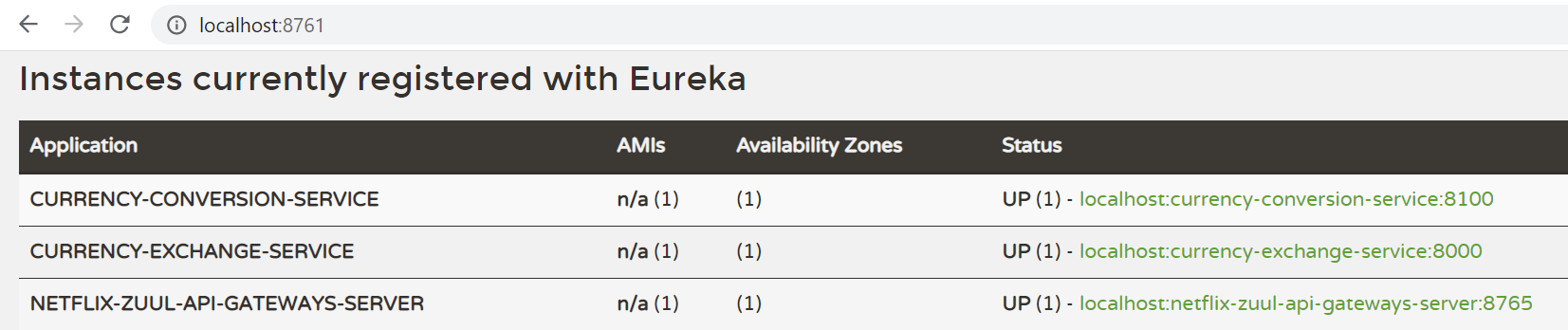




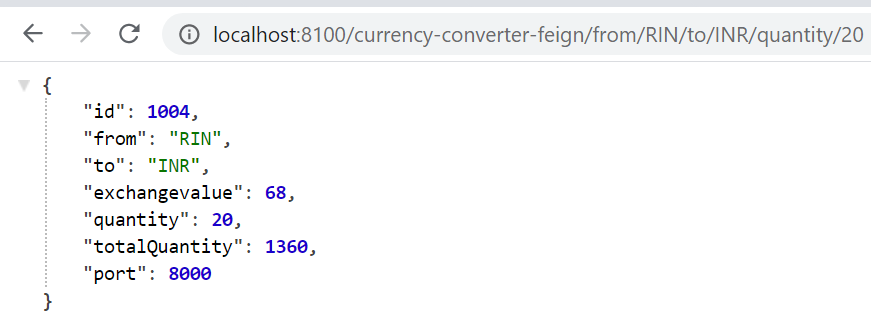


We will start application in below manner:

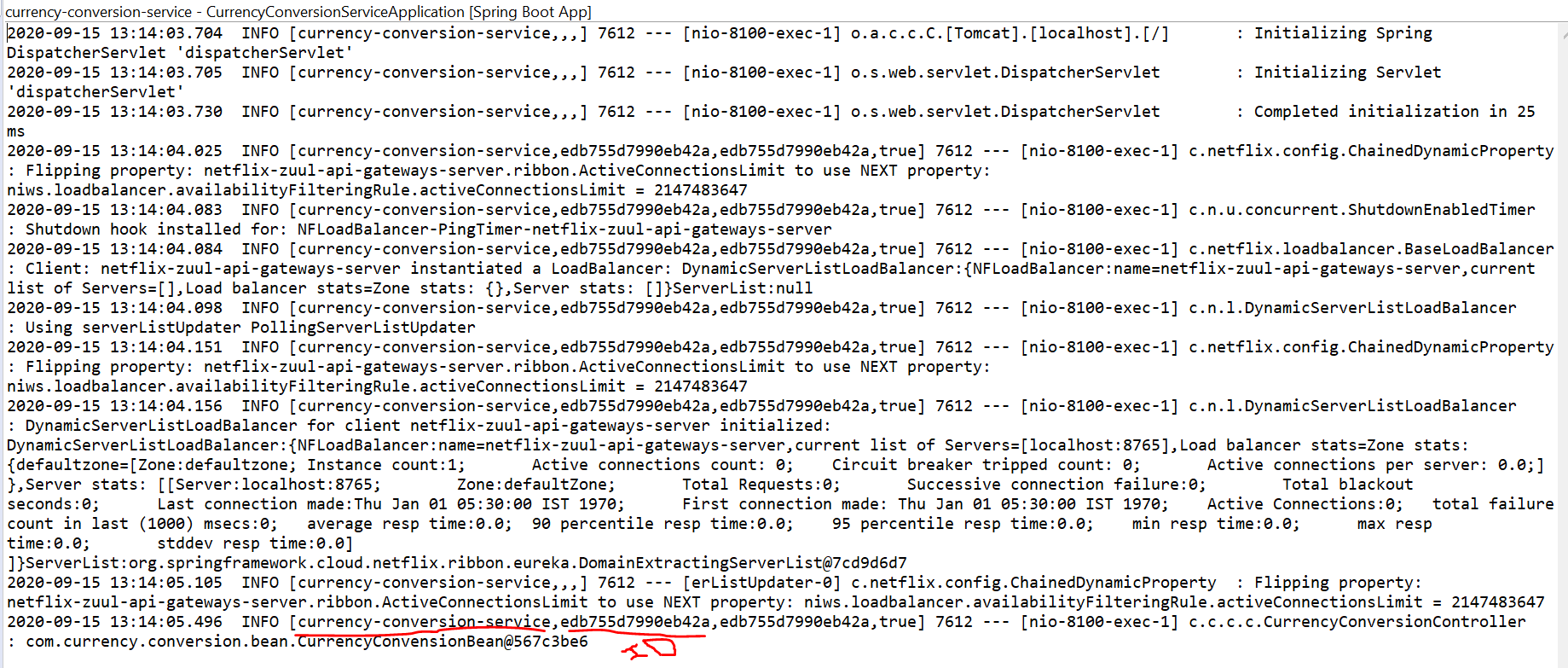
netflix-eureka-naming-server, exchange microservice, conversion microservice and netflix-zuul-api-gateways-server



Hit the conversion service url:

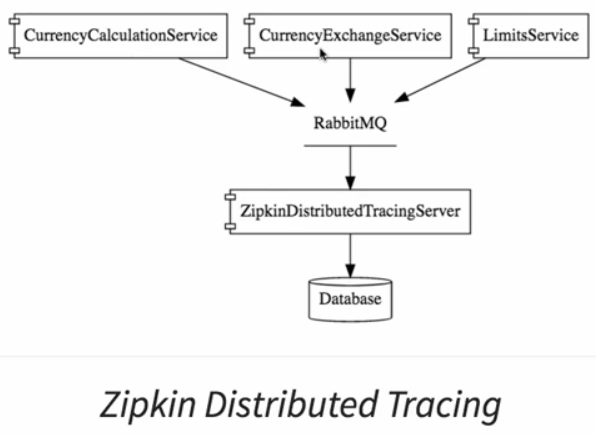


Once you start then you will see the “ID” will be generated in conversion service.



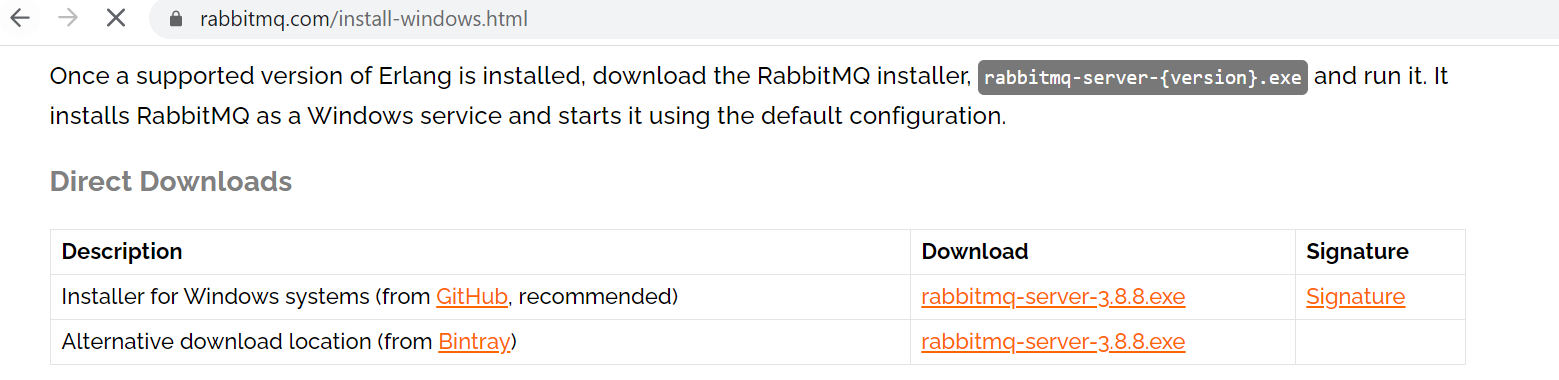
## Introduction to distributed tracing with Zipkin:

Centralized log – we will configure RabbitMQ, ZipkinDistributedTracingServer will listening the RabbitMQ for all the logs:



# Installing Rabbit MQ:

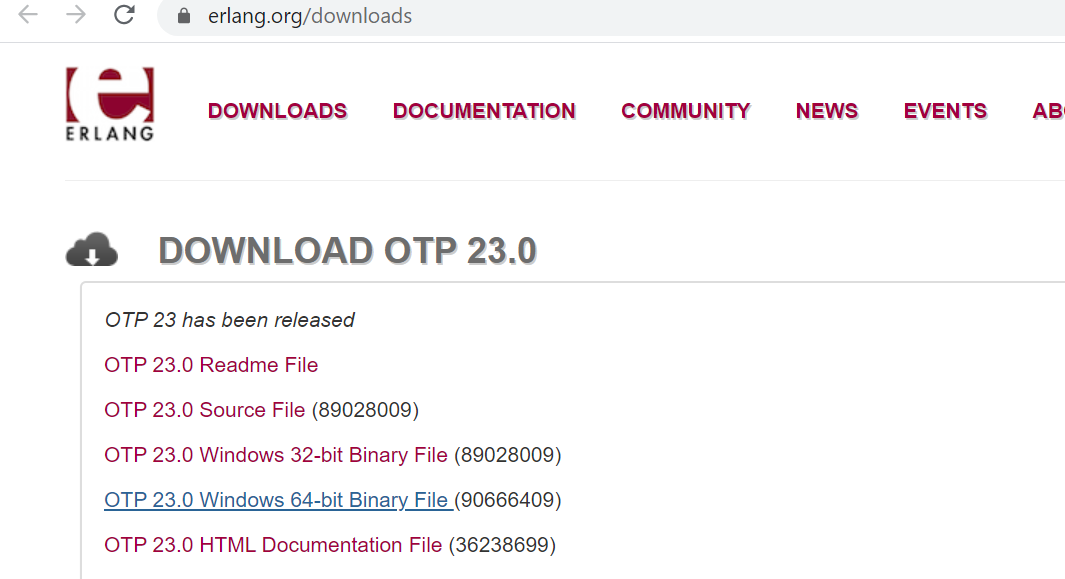
Download URL : <https://www.rabbitmq.com/>



We need to install Erlang before install the RabbitMQ.

## Install Erlang:

<https://www.erlang.org/downloads>

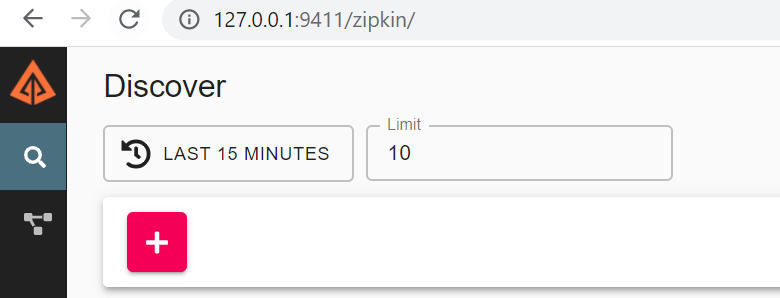


One it done we need to setup Zipkin by quickstart zipkin:

<https://repo1.maven.org/maven2/io/zipkin/zipkin-server/2.21.7/zipkin-server-2.21.7-exec.jar>

1. Download Jar file.
2. Launch the jar by using cmd command(java -jar zipkin-server.jar)





Skipping remaing Zipkins……………

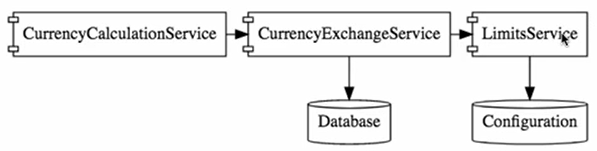
# Fault Tolerance with Hystrix:

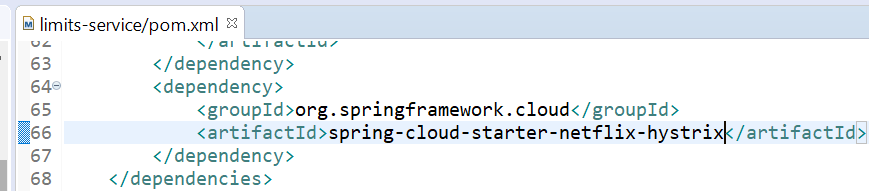
Fault tolerance means one or many microservices are down due to other services down.

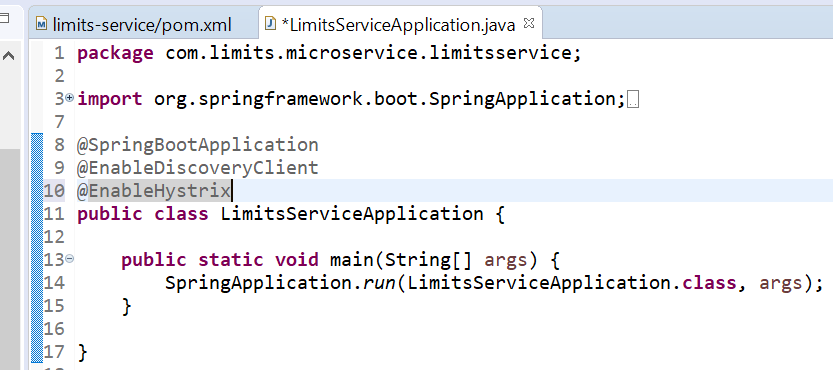
Means : below currency calculation service depend or currency exchange service and it’s depend on Limit service, Suppose limit service is down then ? both service(conversion and exchange) will be down.

This is the fault tolerance, to resolve this, Hystrix use for fault tolerance.

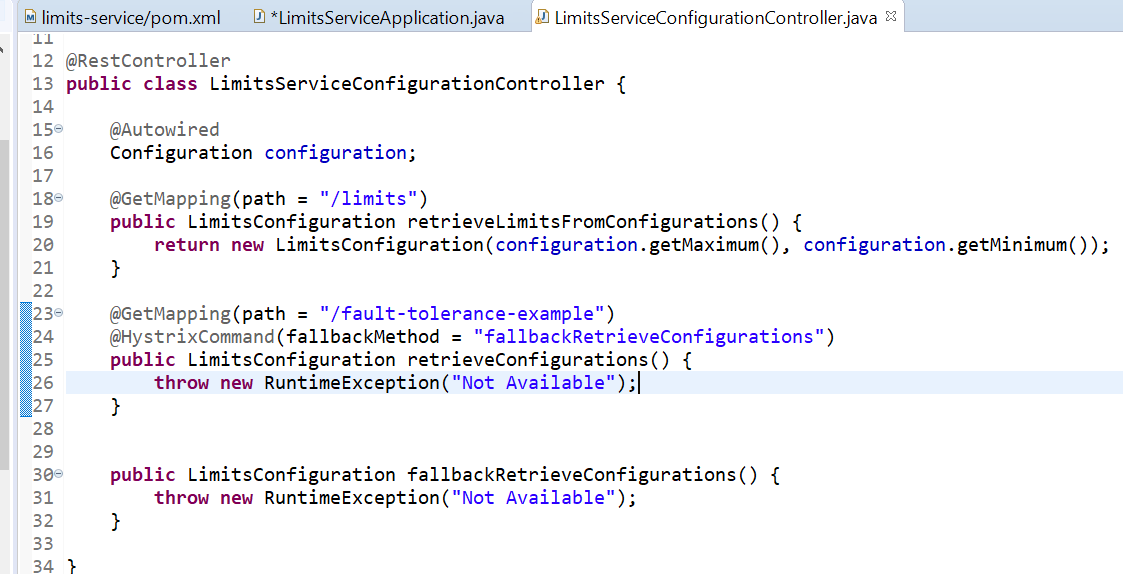
We need to return some default value if microservice is down.







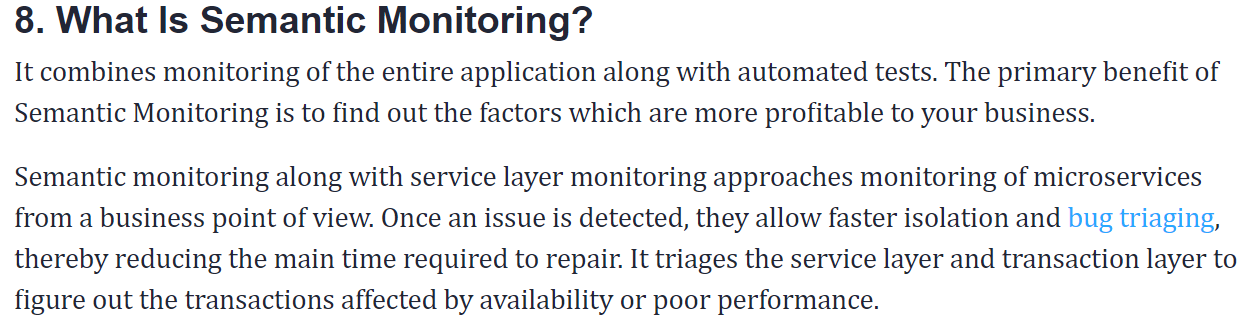
Now declare the Fallback in controller: means if this method is not work then what fallback we need to do:

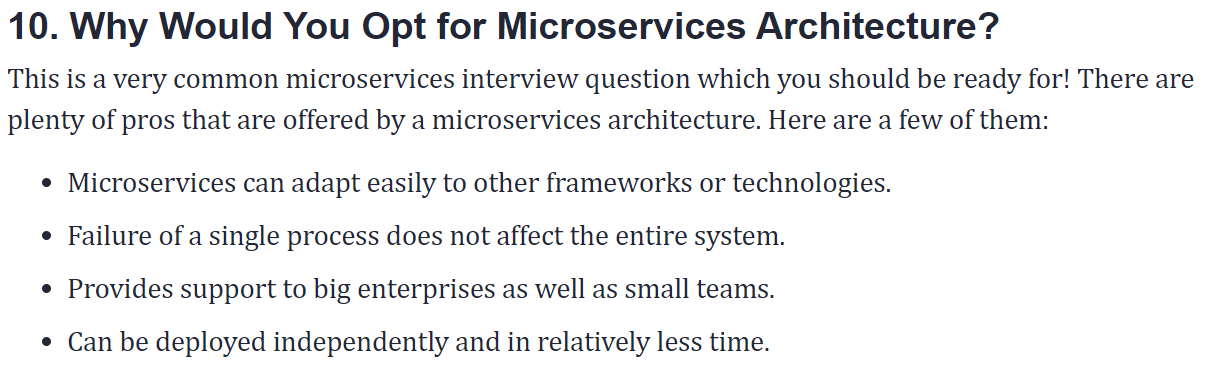


# Interview:

## **Semantic monitoring in Microservices architecture?**

Semantic monitoring combines automated tests with monitoring of the application. It allows you to find out reasons why your business is not getting more profits.





**What is Monolithic Architecture?**

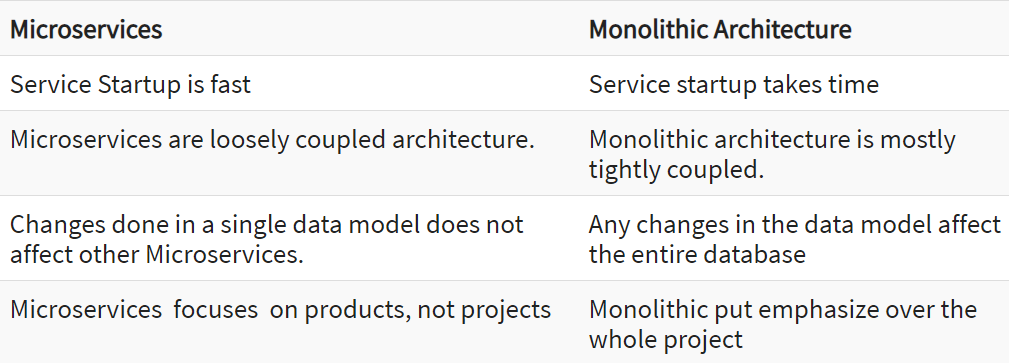
Monolithic architecture is like a big container in which all the software components of an application are clubbed inside a single package.

**What are the advantages of microservices?**

Here, are some significant advantages of using Microservices:

* Technology diversity, e., Microservices can mix easily with other frameworks, libraries,  and databases
* Fault isolation, e., a process failure should not bring the whole system down.
* Greater support for smaller and parallel team
* Independent deployment
* Deployment time reduce

**What are main differences between Microservices and Monolithic Architecture?**



## **What is a CDC?**

CDC is Consumer-Driven Contract. It is a pattern for developing Microservices so that external systems can use them.