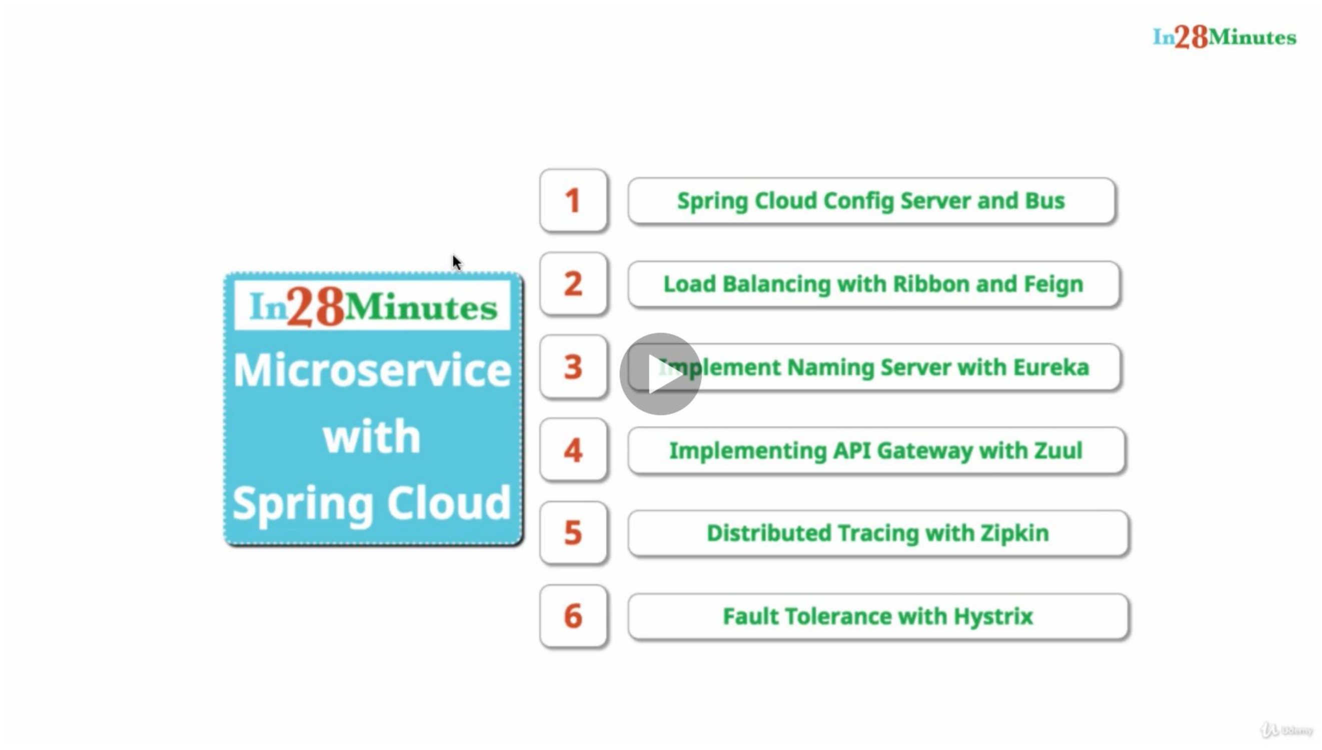
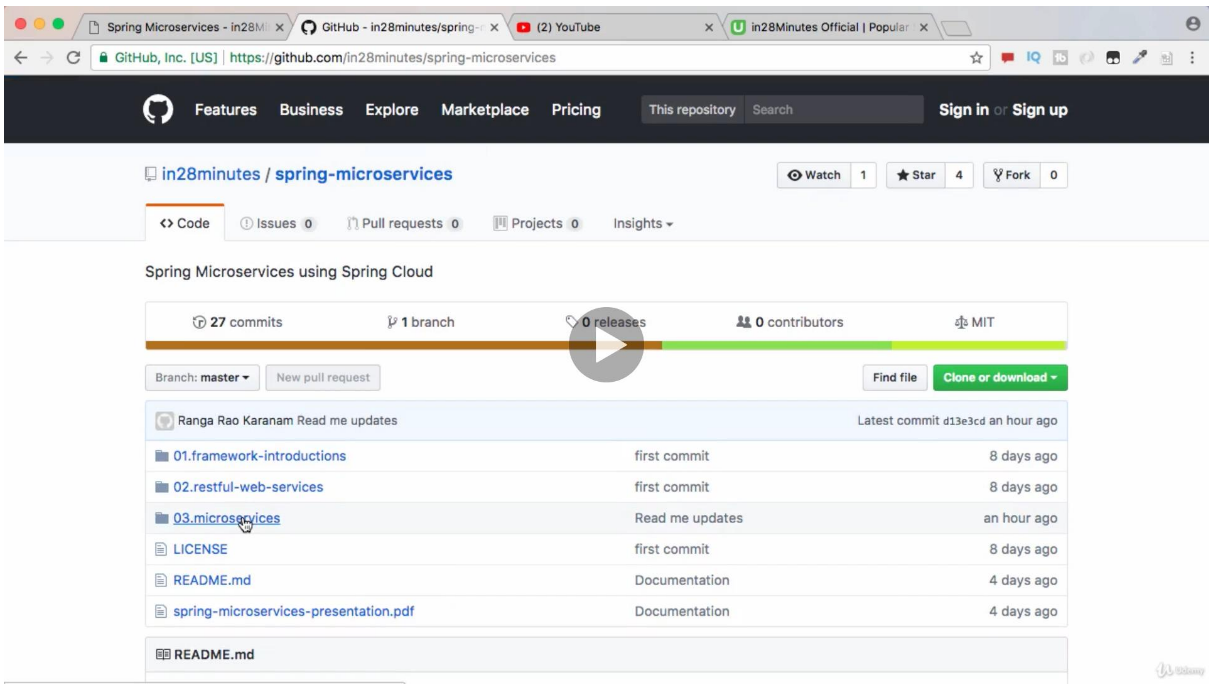
Microservice introduction



In28Minutes

In28Minutes Microservice with Spring Cloud

1. Spring Cloud Config Server and Bus
2. Load Balancing with Ribbon and Feign
3. Implement Naming Server with Eureka
4. Implementing API Gateway with Zuul
5. Distributed Tracing with Zipkin
6. Fault Tolerance with Hystrix



https://github.com/in28minutes/spring-microservices

in28minutes / spring-microservices

Spring Microservices using Spring Cloud

27 commits | 1 branch | 0 releases | 0 contributors | MIT

Ranga Rao Karanam Read me updates — Latest commit d13e3cd an hour ago

01.framework-introductions	first commit	8 days ago
02.restful-web-services	first commit	8 days ago
03.microservices	Read me updates	an hour ago
LICENSE	first commit	8 days ago
README.md	Documentation	4 days ago
spring-microservices-presentation.pdf	Documentation	4 days ago

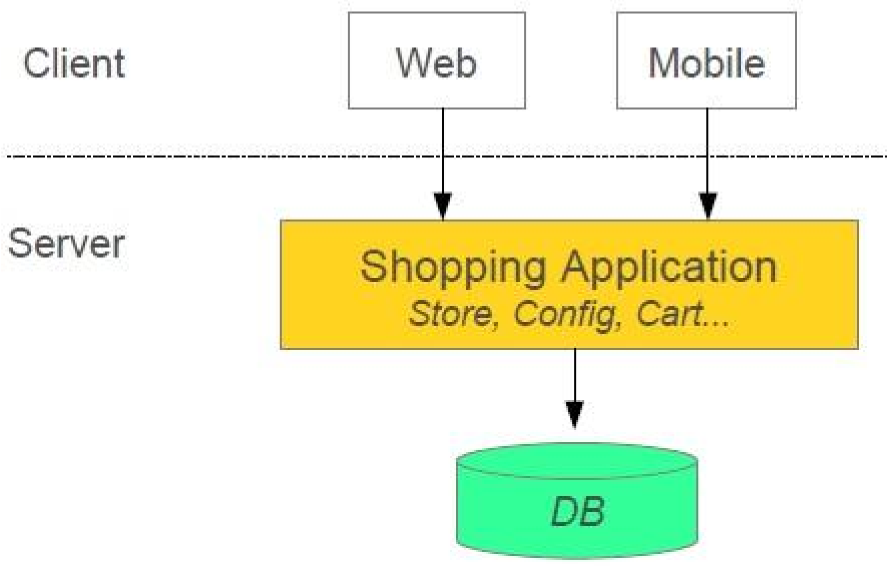
README.md

Microservice ：- Small autonomous service   that work to gather

What is Microservices Architecture?

Microservices architecture allows avoiding monolith application for the large system. It provides loose coupling between collaborating processes which running independently in different environments with tight cohesion.

For example imagine an online shop with separate microservices for user-accounts, product-catalog order-processing and shopping carts. So these components are inevitably important for such a large online shopping portal. For online shopping system



Challenges   for microservice:-

- **Bounded context**
- **Configuration management**
- **Dynamic scaleUp and scaleDown**
- **Visibility**
- **Pack OF cards**

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, one-time tokens, global locks, leadership election, distributed sessions, cluster state).

quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Features :-

- Distributed/versioned configuration
- Service registration and discovery
- Routing
- Service-to-service calls
- Load balancing
- Circuit Breakers
- Global locks
- Leadership election and cluster state
- Distributed messaging

Spring cloud provides the umbrella of the projects spring cloud provides the umbrella of the projects

Exe:

Spring cloud config

Spring cloud Netflix

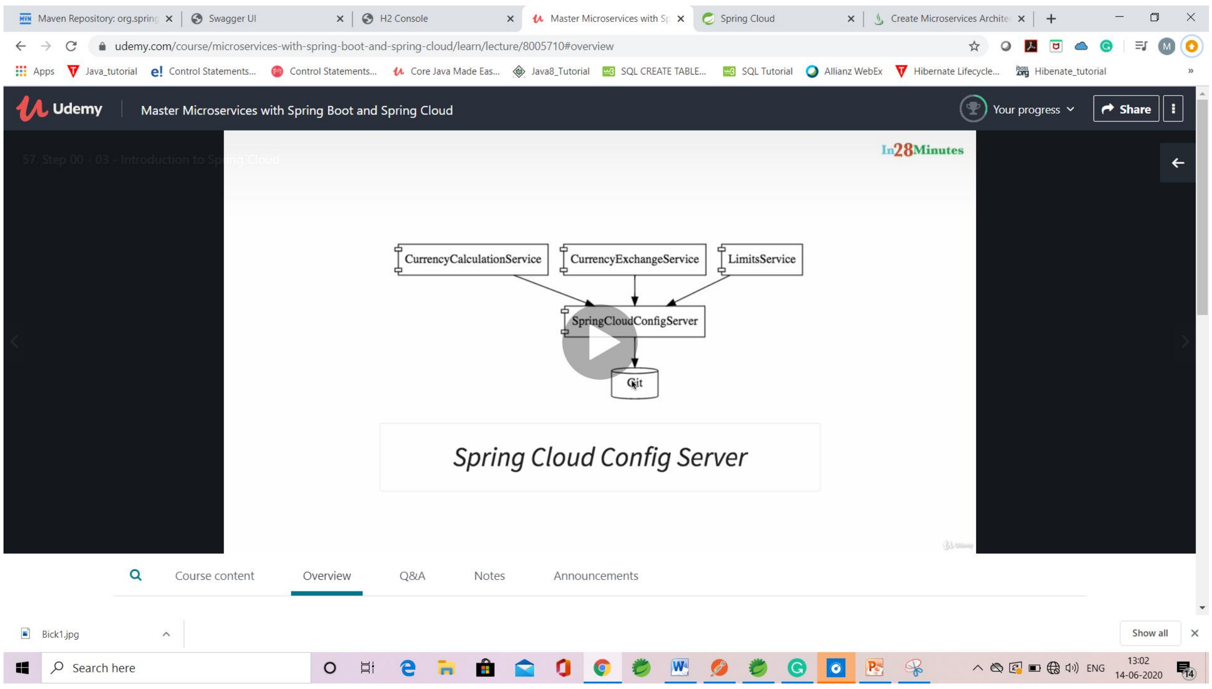
Spring cloud bus (Enable the micro service)…..etc

Challenges discuss here :-

- **Configuration management :-**
  There is multiple microservice and multiple instances, these means there is lot of configurations,

Spring cloud configuration server , provides the all the microservice configuration at one place ,GIT repository

microservice1
microservice2
microservice3



Dynamic scale up and scale down :-

Naming server (eureka)

Ribbon (client side load balancing)

There is the currency calculation service and multiple current exchanges services,

Its possible any point of time, new instance added OR exist, this microservice checks,

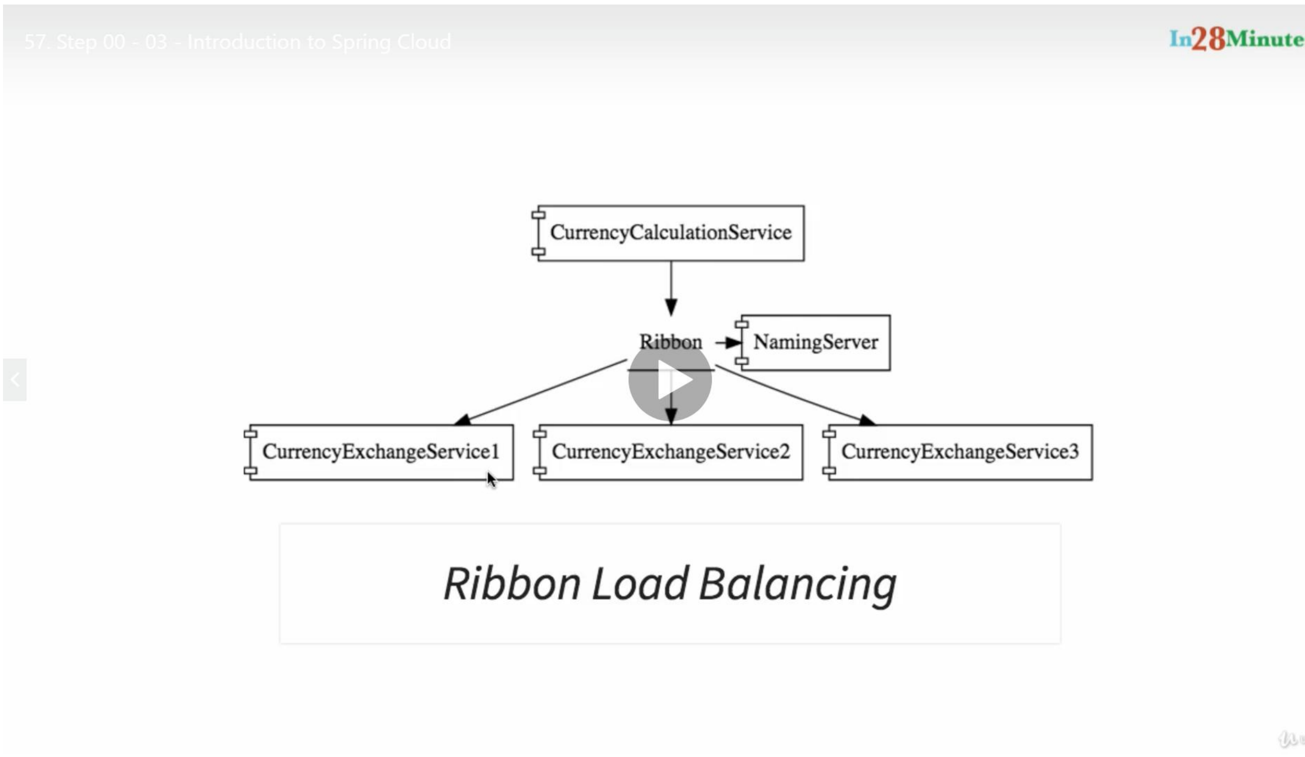
What are the available, to share among them ,

The solution which is Naming server(**Eureka server**)

The naming server two important feauters

- Service registery
- Service discovery

  Note :- this microservice asking to naming server, hey server give me to the current instance server to, available instance ,
  The naming server provide to the those URL's



Visibility And Monitoring

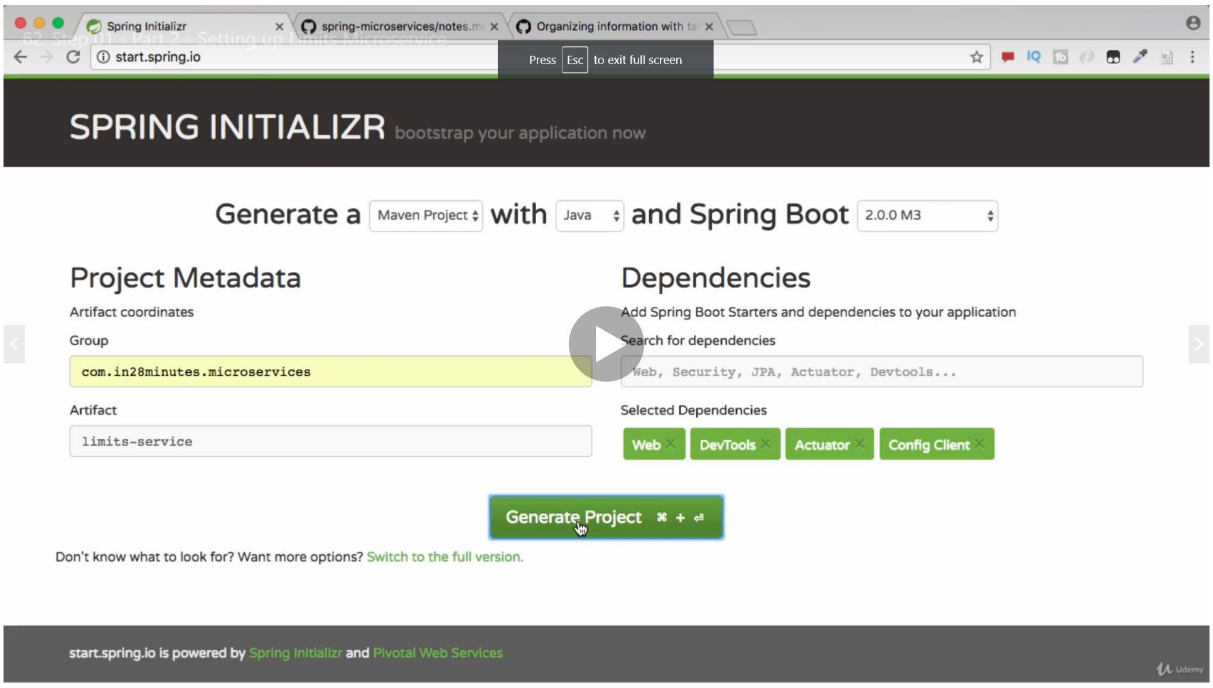
- Zipkin Distributed tracing
- Netflix API gateway

  You would use the id request , across multiple component,

We would use the Zipkin Distributed tracing to trace request across multiple request

Note : if service is down Hystrix is helps to configure default response

Project configuration simple Microservice project


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
limits-service

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web DevTools Actuator Config Client

Generate Project

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

Picks the values from the property's

Propertys file :-

```
limits-service.minimum=9
limits-service.maxmum=99
```

configuration class :-

```
@Component
@ConfigurationProperties("limits-service")
public class Configuration {

	private int minimum;
	private int maxmum;
// setters and getters
```

Controller class :-

```
@RestController
public class LimitConfigurationController {

	@Autowired
	private Configuration Configuration;

	@GetMapping("/limits")
	public LimitConfiguration retriveLimitFromConfiguration()
{
```

```
            return new
LimitConfiguration(Configuration.getMinimum(),Configuration.ge
tMaxmum());
       }
```

// add spring configuration file in application

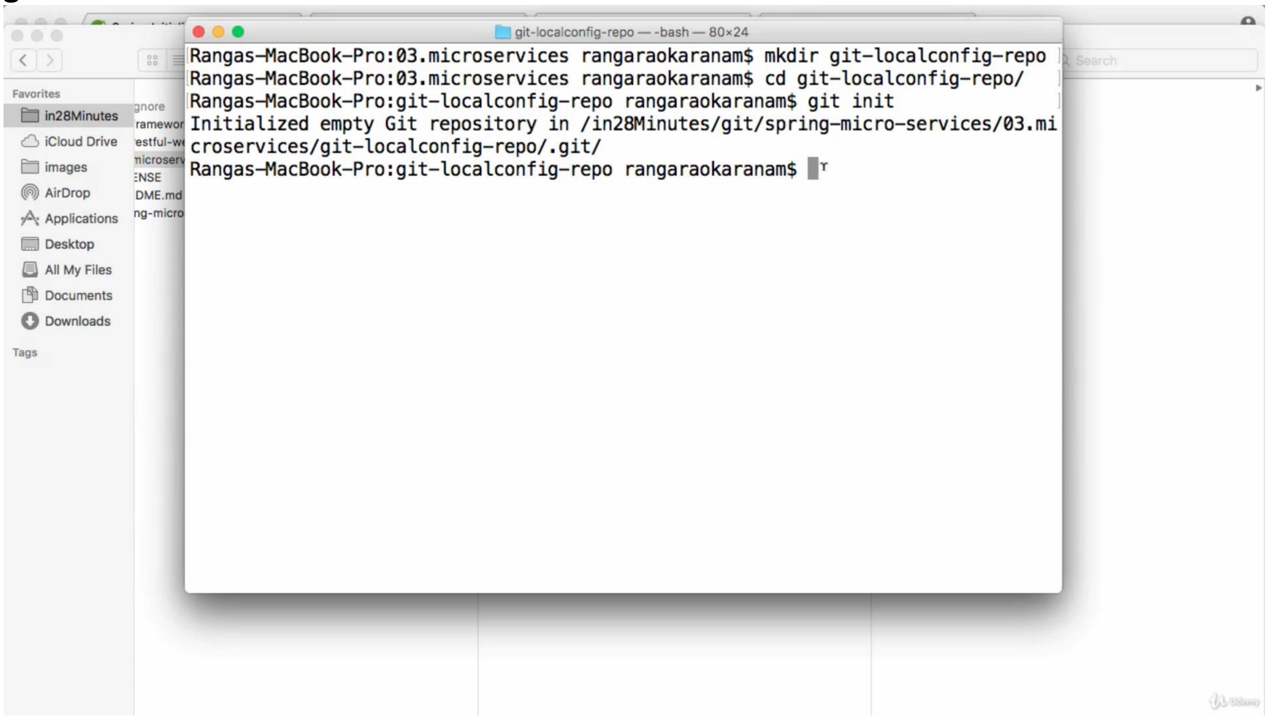
- Create the same as git directory
- Add the properties in the folder
- Add some of the value in the directory

Application.properties :-

spring.application.name=spring-cloud-config-server
server.port=8888

spring.cloud.config.server.git=file://C:/Users/User/test

git commands:-



```
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ mkdir git-localconfig-repo
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ cd git-localconfig-repo/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git init
Initialized empty Git repository in /in28Minutes/git/spring-micro-services/03.mi
croservices/git-localconfig-repo/.git/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$
```

Spring cloud config:-

Application.properties file

```
spring.application.name=spring-cloud-config-server
server.port=8888

spring.cloud.config.server.git.uri=file://C:/Users/User/git-config-repo
```

- **create folder, put the properties retrieve from there only**
- **create under the folder file , put properties , this file will retrieve**
-
- `limits-service.minimum=99`
- `limits-service.maxmum=9999`

Note : - pick the files from config server
Changes application.properties file instead of bootstrap.properties file and

```
spring.application.name=limits-service  // getting class name
   spring.cloud.config.uri =http://localhost:8888 // pick from her  config class
```

NOTE :-

Debugging problems with Spring Cloud Config Server

Debugging microservices problems can be difficult as there are multiple components involved. ***Step by Step instructions*** are provided in the ***troubleshooting guide*** to help you troubleshoot frequently occurring problems. Using ***Chrome Browser*** is recommended.

https://github.com/in28minutes/in28minutes-initiatives/tree/master/The-in28Minutes-TroubleshootingGuide-And-FAQ#debugging-problems-with-spring-cloud-config-server

Spring active profile :- if you want to retrieve specific type of evn profile just use the spring active profile its mention in properties file

```
spring.profiles.active=dev
```

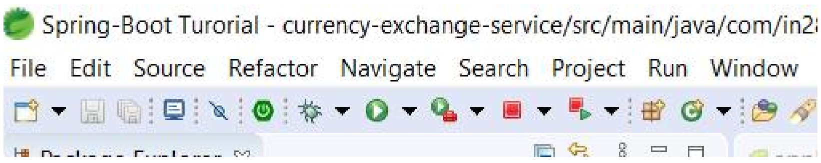
NOTE :-

***SNAPSHOT, M1, M2, M3, M4** releases are typically **WORK IN PROGRESS**. The Spring team is still working on them. We recommend **NOT** using them.*

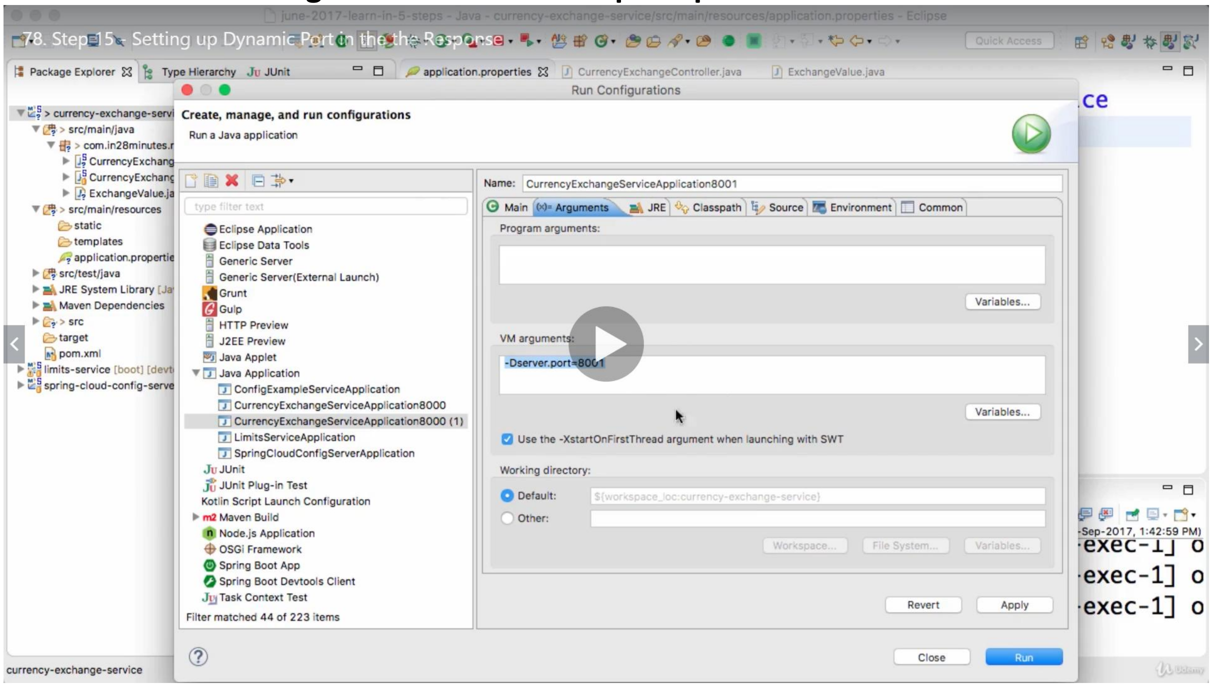
RECOMMENDED VERSION: *2.1.13*

Question :- how to run the application with different porst

Ans :- we will set the port externally set the port in , put in the configuration


Spring-Boot Turorial - currency-exchange-service/src/main/java/com/in2

And make it the project duplicate ,
And Write the VM-argment box -Dserver.port=portnum


78. Step 15 - Setting up Dynamic Port in the the Response

Run Configurations

Create, manage, and run configurations

Run a Java application

Name: CurrencyExchangeServiceApplication8001

VM arguments: -Dserver.port=8001

Use the -XstartOnFirstThread argument when launching with SWT

Working directory: Default: ${workspace_loc:currency-exchange-service}

@Environment :- this class is used to get the properties details

RestTemplate();

Rest Template is used to create applications that consume RESTful Web Services. You can use the **exchange()** method to consume the web services for all HTTP methods

Use Spring Cloud - Greenwich.RC2 and Spring Boot - 2.1.3.RELEASE

We recommend using **2.1.3.RELEASE** version of Spring Boot **from here on in the course**.

Use Spring Cloud - **Greenwich.RC2** and Spring Boot - **2.1.3.RELEASE**.

Make these changes in **all your pom.xml files**. **Save and Restart**. You are all set.

Change 1

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.3.RELEASE</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
```

Change 2

```
<spring-cloud.version>Greenwich.RC2</spring-cloud.version>
```

Feign :-

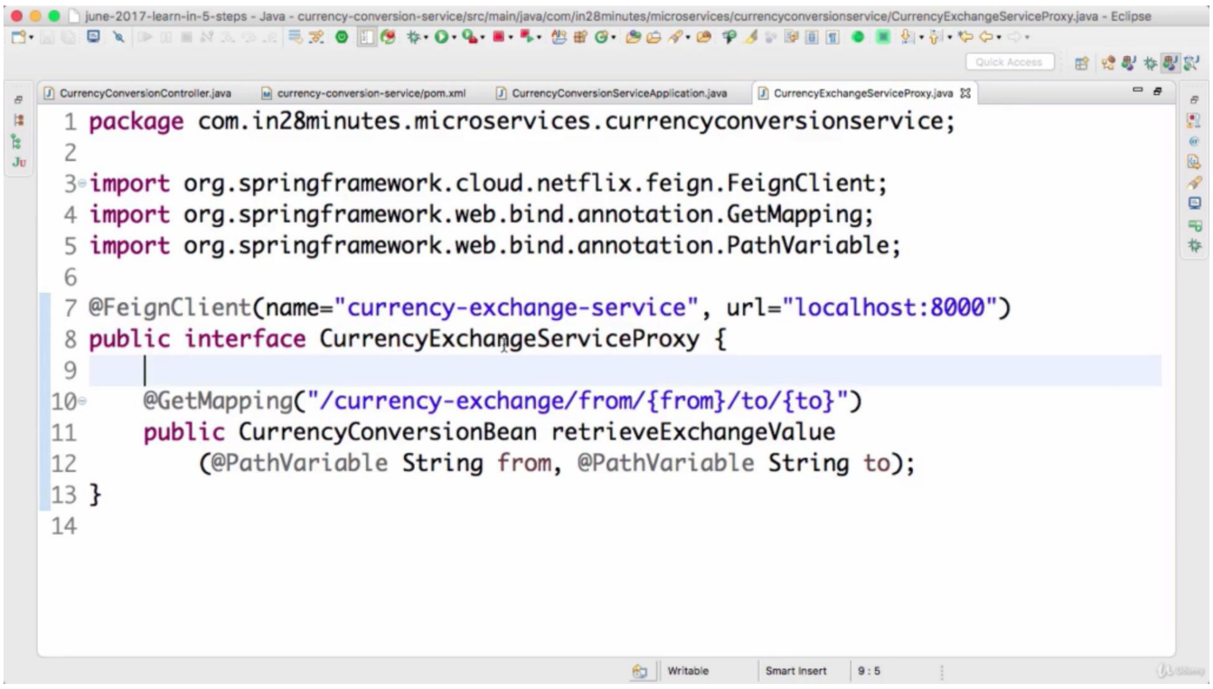
Feign is invoking another microservice, feign is one of the spring frame work dependence, which is used to invoking another microservice,

Add the feign dependency in pom.xml file

And add the @EnableFeignClients("package name")

You can declare this Annotation main class under the @SpringBootApplication

Feign client :-

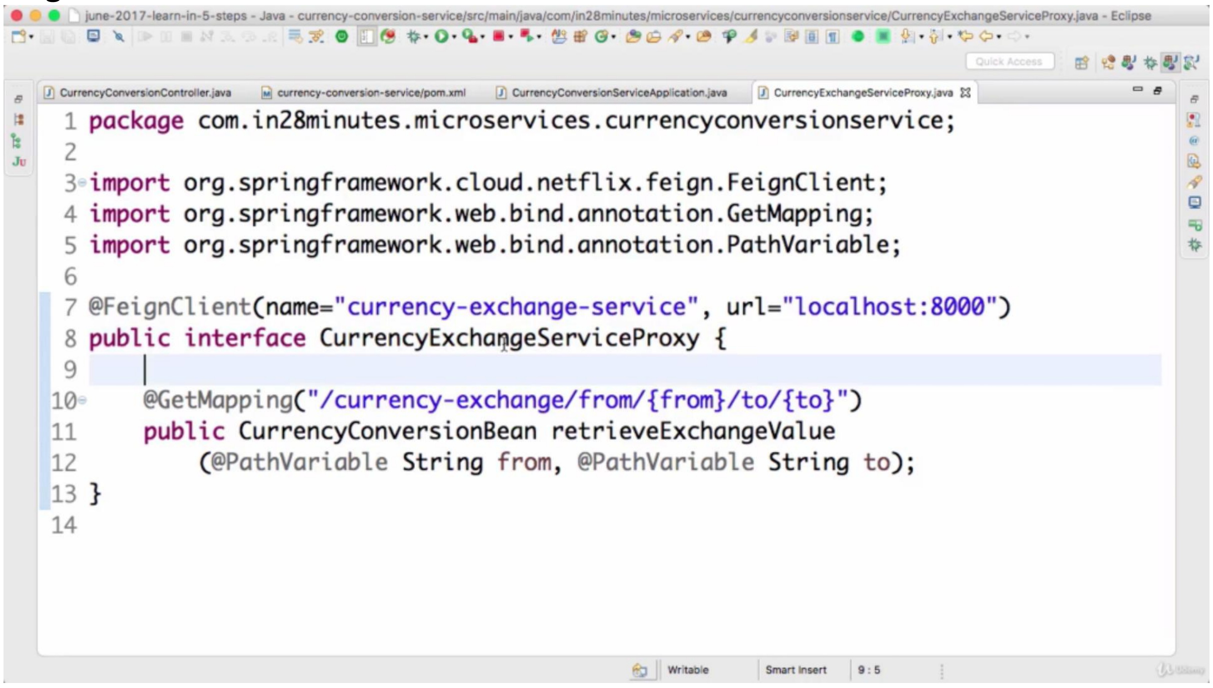


```java
package com.in28minutes.microservices.currencyconversionservice;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="currency-exchange-service", url="localhost:8000")
public interface CurrencyExchangeServiceProxy {
    
    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public CurrencyConversionBean retrieveExchangeValue
        (@PathVariable String from, @PathVariable String to);
}
```

Feign client :-



```java
package com.in28minutes.microservices.currencyconversionservice;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="currency-exchange-service", url="localhost:8000")
public interface CurrencyExchangeServiceProxy {
    
    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public CurrencyConversionBean retrieveExchangeValue
        (@PathVariable String from, @PathVariable String to);
}
```

Ribbon :- ribbon is one the service it will use to share the multiple service, it will help us ,to connect many microservice application

- **Add the ribbon dependency**

Feign client class :-

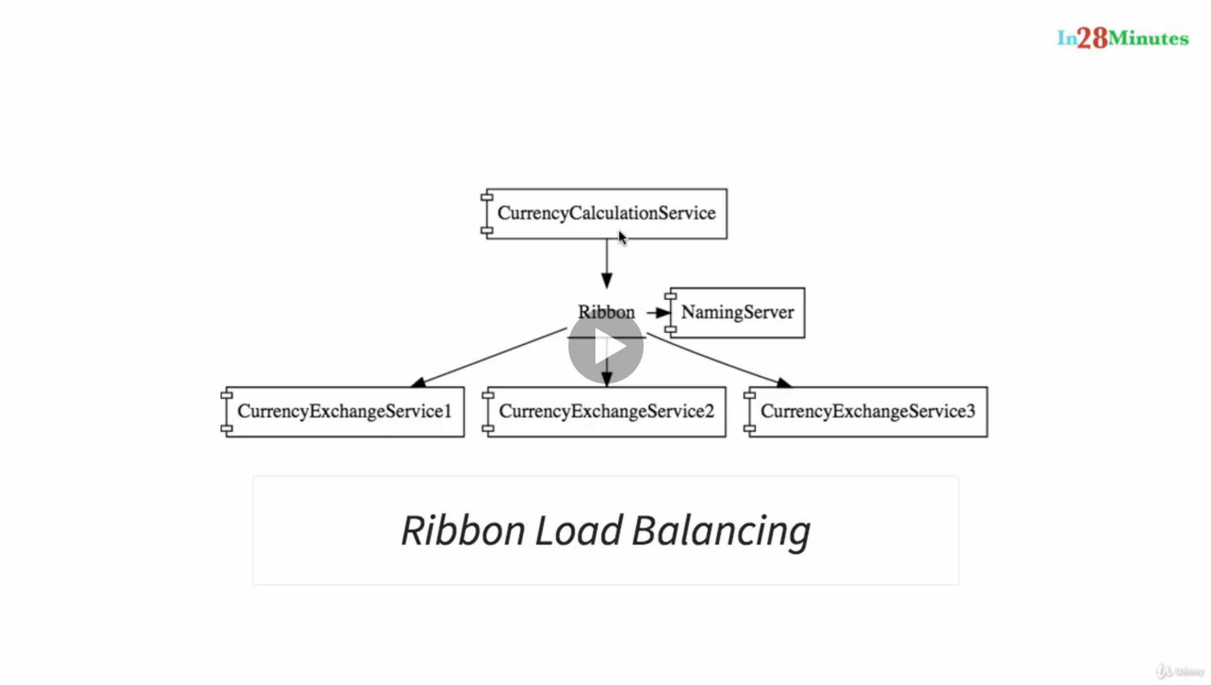
```
//@FeignClient(name="currency-exchange-service",
url="localhost:8000")
//@FeignClient(name="currency-exchange-service")
@FeignClient(name="netflix-zuul-api-gateway-server")
@RibbonClient(name="currency-exchange-service")
public interface CurrencyExchangeServiceProxy {
     //@GetMapping("/currency-exchange/from/{

     @GetMapping("/currency-exchange/from/{from}/to/{to}")
     public CurrencyConversionBean retrieveExchangeValue0
     (     @PathVariable String from, @PathVariable String
to);

     }
```

Properties file:-

```
spring.application.name=currency-convertion-service
server.port=8100
   currency-convertion-service.ribbon.listOfServers=http://localhost:8000,
   http://localhost:8100
```



Debugging problems with Feign and Ribbon

Debugging microservices problems can be difficult as there are multiple components involved.

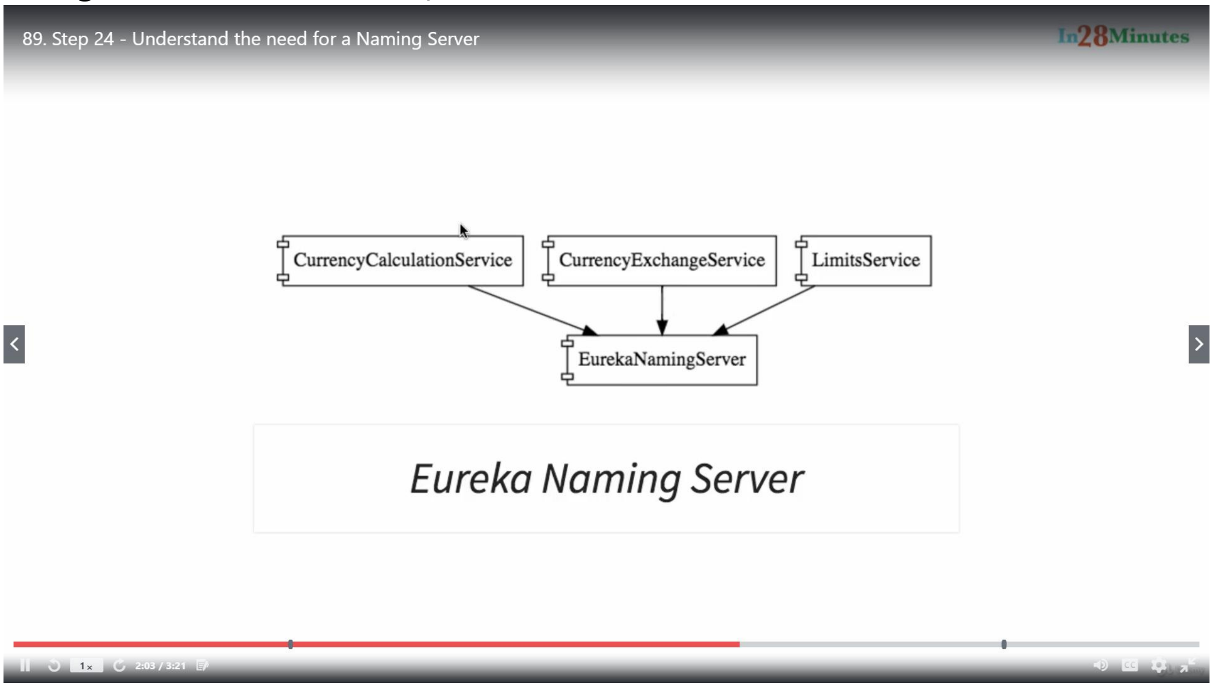
Step by Step instructions are provided in the ***troubleshooting guide*** to help you troubleshoot frequently occurring problems.

Using ***Chrome Browser*** is recommended.

https://github.com/in28minutes/in28minutes-initiatives/tree/master/The-in28Minutes-TroubleshootingGuide-And-FAQ#debugging-problems-with-feign-and-ribbon

Naming Server Eureka :-

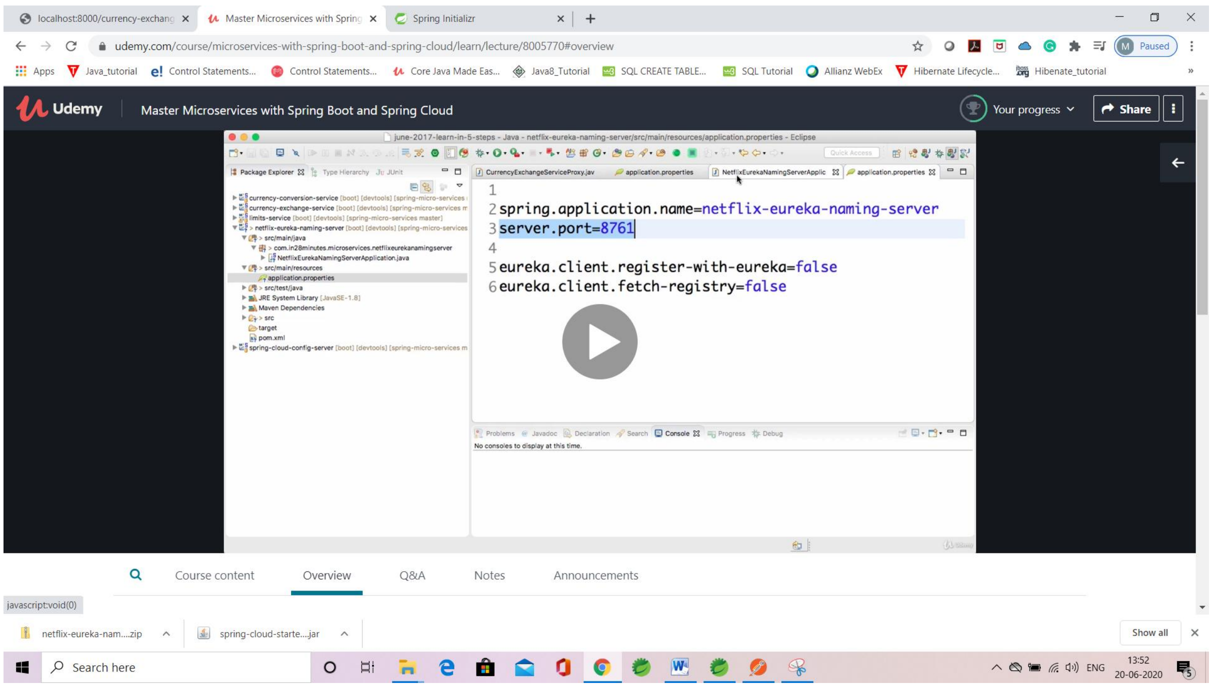
Eureka is the one of the microservice all the instance or all the service would be register the eureka server,



(Or)

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the **Eureka server** and **Eureka server** knows all the client applications running on each port and IP address.

Eureka server configuration properties file :-



```

spring.application.name=netflix-eureka-naming-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

API Gateways

An **API gateway** is the core of an **API** management solution. It acts as the single entryway into a system allowing multiple **APIs** or microservices to act cohesively and provide a uniform experience to the user.

- **Authentication and Authorization and Security**
- **Rate Limits**
- **Fault tolerance**
- **Service Aggregation**

Note :- Netflix has provide API Gateway called as ZUUL there are setting up zuul

- **1 create the component for the zuul**
- **2 what should zuul api does it**
- **3 write request right api request api**

Note :- write the component class

Class ZuulLoggingFilter extends ZuulFilter{

This class has provide four unImplimented abstract methods

- **shouldFilter() :- you can decide if the filter is execute or not , put true or false**

- **run() :- here we will put complete logic in run method**

- **filterType() :- when it will execute the filter before execute or after execute or Only it will display the erro.**

Exe : it its excute before

filterType(){

Return "pre"
}

- **filterOrder() :- if there are multiple filters, you can say priority of order**

}

Note :- we can call every microservice through api gatway zuul

Debugging issue :- in microservice there are multiple service invoked to its difficult to identify the issue microservice, there is solution to identify issue spring cloud sleuth what we call is the distributed tracing system all this service we would put MQ rabbit MQ we would send to zipkin server and find what happened specific request

Question :- what does it spring cloud sleuth ?
Ans :- it would add unique id request for specific microservice and zupking is distributed tracing system

NOTE :- if you want to execute the service through api gatway, you can call first apigatway url/service request rul

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

Sleuth dependency :- in microservice its difficult to find the issue one of the way to achieve this issue using Sleuth and create the simple bean in main application .

```
@Bean
Public  AlwaysSamler defaultsampler(){

return  new AlwaysSampler()
}
```

Updates to Step 39 - Running Zipkin on Windows

In the next step, we set up our Zipkin Server by downloading a jar.

If you get a 404 while downloading the jar, use the curl command to download :

```
curl -sSL https://zipkin.io/quickstart.sh | bash -s
java -jar zipkin.jar
```

For more information, please go through the below link:

https://zipkin.io/pages/quickstart

ONLY FOR WINDOWS USERS

If you are on Windows, this is important for you:

After you watch the next video, You can use the below commands to run Zipkin Server.

```
set RABBIT_URI=amqp://localhost
java -jar zipkin-server-2.7.0-exec.jar
```

Updates to Step 40 : Use spring-cloud-starter-zipkin and spring-rabbit

The dependencies are ever-changing with Spring Cloud and Spring Boot.

If you are using Spring Boot Release >= 2.1.*, you would need to use `spring-cloud-starter-zipkin` and `spring-rabbit` instead of spring-cloud-sleuth-zipkin and spring-cloud-starter-bus-amqp.

You would need to make this change in THREE pom.xmls - in currency-conversion-service, currency-exchange-service, and zuul-api-gateway projects

New Dependencies

```xml
        <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-zipkin</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.amqp</groupId>
                <artifactId>spring-rabbit</artifactId>
        </dependency>
```

OLD Dependencies to be Replaced

```xml
        <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-sleuth-zipkin</artifactId>
        </dependency>

        <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-bus-amqp</artifactId>
        </dependency>
```

Debugging Problems with Zipkin

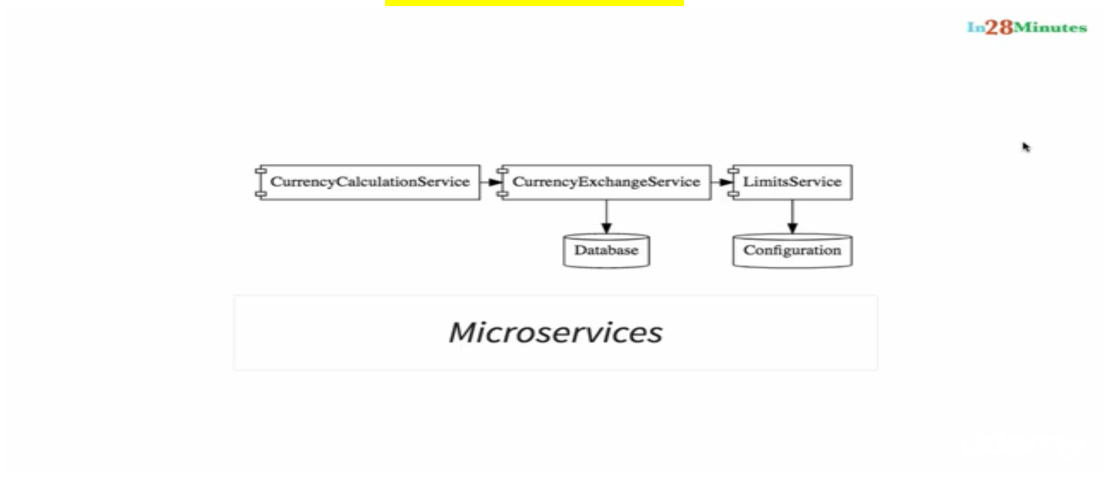
Debugging microservices problems can be difficult as there are multiple components involved.

Step by Step instructions are provided in the ***troubleshooting guide*** to help you troubleshoot frequently occurring problems.

Using ***Chrome Browser*** is recommended.

https://github.com/in28minutes/in28minutes-initiatives/tree/master/The-in28Minutes-TroubleshootingGuide-And-FAQ#debugging-problems-with-zipkin

Fault tolerance



For example :- if any one microservice is goes to down it will effect to some other services also its not good thing, there is hystricx it has been came in to the pictures

Hystricx frame work provide to fault tolerance microservice, if any one microservice is gos down it pick and provide some of the responseable message,

We need to add the hystricx dependency and add the @EnableHystricx annotation in main method()

Question :- What is the hystricx **in microservice ?**

Ans :- The Hystrix framework library helps to control the interaction between services by providing fault tolerance and latency tolerance. It improves overall resilience of the system by isolating the failing services and stopping the cascading effect of failures.