Spring cloud

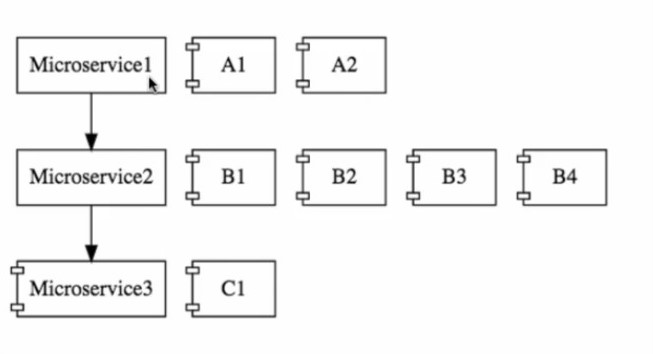
1. Spring cloud config server and bus
2. Load balancing with ribbon and feign
3. Implement naming server with Euerka
4. Implementing API gate way with zuul
5. Distributed Tracking with zipkin
6. Fault tolerance with Hystrix

Microservice: Small anonymous services that work together.

Or

Rest and small well-chosen deployable units and cloud enabled.





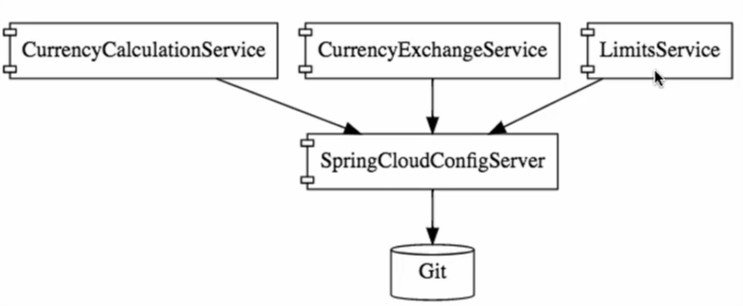
Challenges in MS:

1. Bounded Context

How to identify the boundary of Micro Service i.e. what to do with these micro services.

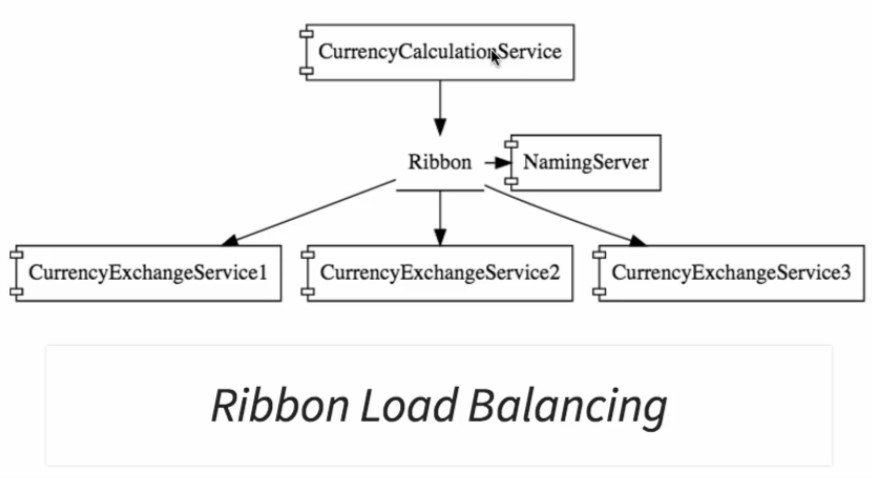
1. Configuration Management

Each micro service has more than one instance.



1. Dynamic Scale up and scale down

The load in different micro service.



Naming Server (Eureka)

Ribbon (Client-side load balancing)

Feign (Easier rest client)

1. Visibility

How to identify bugs and monitoring the micro services

Zipkin Distributed Tracing

Netflix API gate way

1. Pack of card

Fault tolerance using Hystrix

Advantage of Micro services

Communicate with other language

Dynamic scaling i.e load balancing

Faster release build cycle.

Twelve Factors

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

Rule : A1B2C3D3P2L

A : Admin

B: Backing System, Build Release Run

C: Codebase, Configuration, Concurrency

D: Dependencies, Disposability, Dev/Prod env

P: Port binding, Process

L: Logging

|  |  |
| --- | --- |
| Factor | Details |
| Code Base | 1. Recommended version control system is Git. 2. Tool can find out if the code base is using Git. 3. If tool determines that git is not used, developer has to work with BE team to migrate. |
| Dependencies | 1. All 3rd party dependencies should have been approved by the enterprise before they are put in to cloud. 2. Tools create effective build file from the pom.xml or build.gradle and then inspects the dependencies in resultant file. 3. Tool then provide a report of all dependencies with status falling into one of the following categories   Approved  Deprecated  Unverified   1. Developers needs to fix the deprecated/unverified dependencies. |
| Configuration | 1. All the configuration should be available in application.properties 2. Tool merges all property file into application.properties 3. It also identifies N1 variable and create manifest.yml 4. Developers need to work with BE team to define the N1 variable in udeploy. |
| Backing Service | 1. Configure all services used by application as backing services in manifest.yml 2. Tool will create necessary java code for backing services and also configuration in manifest.yml and services.yml 3. Developers need to work with BE team to define values for services configuration like user name, password etc. |
| Build, Release, Run | 1. Ensure the application uses CI/CD pipeline. 2. Tool determine if the application is using Jenkins 3. Developer has to setup the pipeline by working with BE team |
| Process | 1. Ensure all classes used in sessions are Serializable 2. Application will list out all the classes that would require Serialization. 3. Developer has to make the identified classes Serializable. |
| Port Binding | 1. Application has to use port 80 and 443 for non secure and secure communications respectively. 2. Tool will not be able to determine. Developers has to work with cloud team. |
| Concurrency | 1. Have each process do one task well 2. Developers need to work with architect if there is need to divide the app into sub companies. |
| Disposability | Spring Application   1. Migrate Spring application to Spring boot application. 2. Tool will create application to spring boot application along with java configuration 3. Developers need to fix any TODO items generated by the tool.   Spring XML configuration   1. Spring xml configuration is migrated to java config classes 2. Read beans from xml file config, Identify relationships between beans 3. Create java class with bean definitions using java application.   Web.xml configuration   1. Web.xml configuration is migrated to web app Initializer and web.xml 2. Create spring-based configuration classes for web.xml 3. Convert web.xml elements to respective java config, Keep only jsp config. 4. Security config and resource ref etc in web.xml 5. Create entry point class for spring boot |
| Dev/Prod Parity | 1. Keep various environments same 2. Developers need to identify and fix any disparities in various environments. |
| Logs | 1. Logs should be written to console only 2. Tool will create new log4j configuration file with console appender 3. Developers need to verify the log4j config generated by the tool |
| Admin process | 1. Run admin process separate from long running processes 2. Developer has to identify and fix accordingly. |

Ports

|  |  |
| --- | --- |
| Application | Port |
| Limit Service | 8080, 8081, … |
| Spring Cloud Config Server | 8888 |
| Currency Exchange Service | 8000, 8001, 8002, … |
| Currency Conversion Service | 8100, 8101, 8102, … |
| Netflix Eureka Naming Server | 8761 |
| Netflix Zulu API Gateway Server | 8765 |
| Zipkin Distributed Tracking Server | 9411 |

URLS:

|  |  |
| --- | --- |
| Limits service | Get: <http://localhost:8080/limits> |
| Spring cloud config server | Get: <http://localhost:8888/limits-service/default>   <http://localhost:8888/limits-service/dev>   <http://localhost:8888/limits-service/qa> |
| Currency exchange service | Get: <http://localhost:8000/currency-exchange/from/USD/to/INR> |
| Currency converter service | Get: <http://localhost:8100/currency-convert/from/USD/to/INR/quantity/30>  Get: <http://localhost:8100/currency-convert-feign/from/AUD/to/INR/quantity/30> |
| Netflix Eureka naming Server | Get: <http://localhost:8761/> |
| Netflix Zuul Api Gate Way | Get: <http://localhost:8765/currency-exchange-service/currency-exchange/from/USD/to/INR>  Get: <http://localhost:8765/currency-conversion-service/currency-convert-feign/from/AUD/to/INR/quantity/30>  Get: <http://localhost:8100/currency-convert-feign/from/AUD/to/INR/quantity/30> |
| Zipkin URL | <http://localhost:9411/zipkin/> |

Project Work 1:

1. limits-service
2. spring-cloud-config-server
3. git\_config\_local

Step 01:

Create limits-service project.

pom.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.udemy.ms28min</groupId>

<artifactId>limits-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>limits-service</name>

<description>Limit service application</description>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.4.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring.cloud.version>Finchley.SR2</spring.cloud.version>

</properties>

<dependencies>

<!-- Cloud -->

<dependency>

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

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

</dependency>

<!-- Basic -->

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<!-- @ConfigurationProperties("limits-service")

To create read data from properties file

This is optional to use this dependency

-->

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring.cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

application.properties file

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=limits-service

server.port=8080

limits-service.maximum=9999

limits-service.minimum=99

Read the data from properties file

@Component

@ConfigurationProperties("limits-service")

**public** **class** Configuration {

**private** **int** maximum;

**private** **int** minimum;

@RestController

**public** **class** LimitsConfigurationController {

@Autowired

**private** Configuration configuration;

@GetMapping("/limits")

**public** LimitsConfiguration retriveLimitsFromConfiguration() {

**return** **new** LimitsConfiguration(configuration.getMaximum(), configuration.getMinimum());

}

}

Step 02:

Create spring-cloud-config-server

<dependencies>

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

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

server.port=8888

creating local git repository

D:\dev\_env\dev\_workspace\udemy\git\_config\_local

Now add git\_config\_local link to spring-cloud-config-server

Right click on spring-cloud-config-server -> build path -> link source

Create limits-service.properties file in git\_config\_local folder

limits-service.minimum=88

limits-service.maximum=8888

Then commit your changes to local i.e. git add . then git commit -m “message”

Now start the spring-cloud-config-server server

http://localhost:8888/{properties\_file\_name}/{default \_enviroment}

i.e. <http://localhost:8888/limits-service/default>

We will get no response as because we need to enable config server.

@EnableConfigServer

@SpringBootApplication

**public** **class** SpringCloudConfigServerApplication {

//OP:

{

name: "limits-service",

profiles: [

"default"

],

label: null,

version: "4a3f32d8e6701fcaf1c1ba39350f67fa01261d96",

state: null,

propertySources: [

{

name: "file://D:/dev\_env/dev\_workspace/udemy/git\_config\_local/limits-service.properties",

source: {

limits-service.minimum: "88",

limits-service.maximum: "8888"

}

}

]

}

Step 3

Connect spring-cloud-config-server to (multiple) local git config repository

Create two properties file

limits-service-dev.properties

limits-service.minimum=11

limits-service.maximum=1111

limits-service-qa.properties

limits-service.minimum=22

limits-service.maximum=2222

commit your changes to local repository.

<http://localhost:8888/limits-service/dev>

<http://localhost:8888/limits-service/qa>

propertySources: [

{

name: "file://D:/dev\_env/dev\_workspace/udemy/git\_config\_local/limits-service-qa.properties",

source: {

limits-service.minimum: "22",

limits-service.maximum: "2222"

}

},

{

name: "file://D:/dev\_env/dev\_workspace/udemy/git\_config\_local/limits-service.properties",

source: {

limits-service.minimum: "88",

limits-service.maximum: "8888"

}

}

]

Step 4

Let limits-service pick up the values from spring-cloud-config-server

application.properties file of limits-service

### Project Work 1 Step 1 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=limits-service

server.port=8080

## Comment for Project Work 1 step 4 ##

# read these value form git local config file #

#limits-service.maximum=9999

#limits-service.minimum=99

### Project Work 1 Step 4 ###

spring.cloud.config.uri=http://localhost:8888

Note : spring.application.name should be same with properties file name present in local git config repository.

//fetch from log

Fetching config from server at : http://localhost:8888

Located environment: name=limits-service, profiles=[default], label=null, version=4a3f32d8e6701fcaf1c1ba39350f67fa01261d96, state=null

Located property source: CompositePropertySource {name='configService', propertySources=[MapPropertySource {name='configClient'}, MapPropertySource {name='file://D:/dev\_env/dev\_workspace/udemy/git\_config\_local/limits-service.properties'}]}

No active profile set, falling back to default profiles: default

For dev/qa environment add spring profile name in application.properties file

spring.profiles.active=qa

But it will not pick up the value. So that we need to rename application.properties file to bootstrap.properties file in limits-service application.

Step 5

Understand Spring cloud bus

Suppose there are 3 instance of limit service running

I change the value in qa (D:\dev\_env\dev\_workspace\udemy\git\_config\_local) env limits-service.minimum=22 to limits-service.minimum=2

After that I commit my changes to git local

<http://localhost:8082/limits>

Still we are getting the old value after committing the changes

To reflect the change we need send a post refresh request

Open postman fire a post request

POST : http://localhost:8082/actuator/refresh

{

"timestamp": "2018-12-10T18:01:36.169+0000",

"status": 404,

"error": "Not Found",

"message": "No message available",

"path": "/actuator/refresh"

}

We need to turn of security of actuator

Add below line in bootstrap.proerties file

management.endpoints.web.exposure.include=\*

Now change the value in qa (D:\dev\_env\dev\_workspace\udemy\git\_config\_local) env limits-service.minimum=2 to limits-service.minimum=22

After that I commit my changes to git local

<http://localhost:8082/limits>

still old value is there now hit post request

POST : http://localhost:8082/actuator/refresh

Now <http://localhost:8082/limits> give the updated value

But problem is <http://localhost:8080/limits> and <http://localhost:8081/limits> has old value

so we need to send post actuator/refresh for all other instance

So we need spring cloud bus to handle this issue

Implement spring cloud bus

<!-- project work 1 step 5 add spring cloud bus dependencies start -->

<dependency>

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

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

<!-- project work 1 step 5 add spring cloud bus dependencies end -->

Add in both pom file limit service and spring-cloud-config-server

### Project Work 1 Step 5###

management.security.enabled=false

Add this in bootstrap.properties file

Start both the server and Now change the value in qa (D:\dev\_env\dev\_workspace\udemy\git\_config\_local) env limits-service.minimum=22 to limits-service.minimum=222

Commit your changes

Now

POST : <http://localhost:8080/actuator/bus-refresh> in post man

Now changes will reflect in all the instance

Step 6

Fault tolerance using hystrix

limit-service pom.xml

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

@SpringBootApplication

@EnableHystrix

**public** **class** LimitsServiceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(LimitsServiceApplication.**class**, args);

}

}

@GetMapping("/limits-fault")

//project work 1 step 6

// if any exception throw here then it will go to fallback method

@HystrixCommand(fallbackMethod="fallbackRetriveConfiguration")

**public** LimitsConfiguration retriveLimitsFromConfigurationFaultTolerance() {

**throw** **new** RuntimeException("own exception");

}

**public** LimitsConfiguration fallbackRetriveConfiguration() {

**return** **new** LimitsConfiguration(5, 5555);

}

http://localhost:8080/limits-fault

Project Work 2:

1. currency-exchange-service
2. currency-conversion-service
3. netflix-eureka-naming-server
4. netflix-zuul-api-gateway-server

Step 1:

Set up currency-exchange-service

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version> Finchley.SR2<version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

### Project Work 2 Step 1 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=currency-exchange-service

server.port=8000

Overwrite existing post

Start application run argument -> vm argument ->

-Dserver.port=8001

Now it will overwrite 8000 port with 8001

Start with multiple port

goto run confoguration -> right click on existing instance(from left side) -> duplicate

new istance -> rename -> arguments -> add -Dserver.port=8001 ->apply-> run

Step 2

Connect with h2 data base and store exchange value details in db

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<!-- Spring Data rest Starter -->

<dependency>

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

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

</dependency>

@Entity

@Table(name="exchange\_value")

**public** **class** ExchangeValue {

@Id

**private** **long** id;

@Column(name="currency\_from")

**private** String from;

@Column(name="currency\_to")

**private** String to;

**private** BigDecimal conversionMultiple;

**private** **int** port;

Add data.sql to resource folder

insert into exchange\_value(id, currency\_from, currency\_to, conversion\_multiple, port)

values(10001, 'USD', 'INR', 69, 0);

insert into exchange\_value(id, currency\_from, currency\_to, conversion\_multiple, port)

values(10002, 'EUR', 'INR', 75, 0);

insert into exchange\_value(id, currency\_from, currency\_to, conversion\_multiple, port)

values(10003, 'AUD', 'INR', 25, 0);

application.properties file

### Project Work 2 Step 2 ###

## connect with data base ##

spring.jpa.show-sql=true

spring.h2.console.enabled=true

See the data in h2-console

<http://localhost:8000/h2-console>

 jdbc:h2:mem:testdb jdbc name -> connect

**public** **interface** ExchangeValueRepository **extends** JpaRepository<ExchangeValue, Long>{

**public** ExchangeValue findByFromAndTo(String from, String to);

}

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

**public** ExchangeValue retriveExchangeValue(@PathVariable String from, @PathVariable String to) {

ExchangeValue exchangeValue = currencyExchangeValueRepository.findByFromAndTo(from, to);

exchangeValue.setPort(`));

**return** exchangeValue;

}

Step 3

Set up currency conversion service and invoke currency exchange service

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

### Project Work 2 Step 3 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=currency-conversion-service

server.port=8100

@RestController

**public** **class** CurrencyConversionController {

@GetMapping("/currency-convert/from/{from}/to/{to}/quantity/{quantity}")

**public** CurrencyConversionBean getConversionValue(@PathVariable String from, @PathVariable String to,

@PathVariable String quantity) {

String url = "http://localhost:8000/currency-exchange/from/{from}/to/{to}";

Map<String, String> uriVariables = **new** HashMap<>();

uriVariables.put("from", from);

uriVariables.put("to", to);

ResponseEntity<CurrencyConversionBean> responseEntity = **new** RestTemplate().getForEntity(url,

CurrencyConversionBean.**class**, uriVariables);

CurrencyConversionBean responseBody = responseEntity.getBody();

BigDecimal quantityNo = **new** BigDecimal(quantity);

CurrencyConversionBean currencyConversionBean = **new** CurrencyConversionBean(responseBody.getId(), from, to,

responseBody.getConversionMultiple(), quantityNo,

quantityNo.multiply(responseBody.getConversionMultiple()));

currencyConversionBean.setPort(responseBody.getPort());

**return** currencyConversionBean;

}

}

Step 4:

Use Feign REST client for rest service invocation

Currency-conversion-service

<!-- Feign dependencies -->

<dependency>

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

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

</dependency>

@SpringBootApplication

@EnableFeignClients("com.udemy.ms28min")

**public** **class** CurrencyConversionServiceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CurrencyConversionServiceApplication.**class**, args);

}

}

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

**public** **interface** CurrencyExchangeServiceProxy {

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

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

}

@GetMapping("/currency-convert-feign/from/{from}/to/{to}/quantity/{quantity}")

**public** CurrencyConversionBean getConversionValueFeign(@PathVariable String from, @PathVariable String to,

@PathVariable String quantity) {

CurrencyConversionBean bean = proxy.retriveExchangeValue(from, to);

BigDecimal quantityNo = **new** BigDecimal(quantity);

**return** **new** CurrencyConversionBean(bean.getId(), from, to, bean.getConversionMultiple(),

quantityNo, quantityNo.multiply(bean.getConversionMultiple()), bean.getPort());

}

Step 5

Setting up and running client side load balancing with ribbon

<!-- load balancing dependencies Project Work 2 Step 5 start-->

<dependency>

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

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

</dependency>

<!-- load balancing dependencies Project Work 2 Step 5 end-->

Add ribbon client to proxy service

/\*

\* comment for @RibbonClient as because it will load and balance multiple instance service

\*/

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

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

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

**public** **interface** CurrencyExchangeServiceProxy {

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

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

}

### Project Work 2 Step 5 ###

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

{currency-exchange-service i.e which service name to use}.ribbon.listOfServers

Step 6

Setting up Eureka naming server i.e netflix-eureka-naming-server

netflix-eureka-naming-server

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

@SpringBootApplication

@EnableEurekaServer

**public** **class** NetflixEurekaNamingServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(NetflixEurekaNamingServerApplication.**class**, args);

}

}

### Project Work 2 Step 6 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

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

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

<http://localhost:8761/>

Step 7

Connect currency conversion service and currency exchange service application to Netflix eureka naming server

Currency conversion service

<!-- eureka naming server client dependencies Project Work 2 Step 7 start-->

<dependency>

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

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

</dependency>

<!-- eureka naming server client dependencies Project Work 2 Step 7 start-->

@SpringBootApplication

@EnableFeignClients("com.udemy.ms28min")

//Project work 2 Step 7

@EnableDiscoveryClient

**public** **class** CurrencyConversionServiceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CurrencyConversionServiceApplication.**class**, args);

}

}

### Project Work 2 Step 3 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=currency-conversion-service

server.port=8100

### Project Work 2 Step 5 ###

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

### Project Work 2 Step 7 ###

eureka.client.service-url.default-zone=http://localhost:8761/eureka

Currency exchange service

<!-- eureka naming server client dependencies Project Work 2 Step 7 start-->

<dependency>

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

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

</dependency>

<!-- eureka naming server client dependencies Project Work 2 Step 7 start-->

@SpringBootApplication

//Project work 2 Step 7

@EnableDiscoveryClient

**public** **class** CurrencyExchangeServiceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CurrencyExchangeServiceApplication.**class**, args);

}

}

### Project Work 2 Step 1 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=currency-exchange-service

server.port=8000

### Project Work 2 Step 2 ###

## connect with data base ##

spring.jpa.show-sql=true

spring.h2.console.enabled=true

### Project Work 2 Step 7 ###

eureka.client.service-url.default-zone=http://localhost:8761/eureka

Step 8

Distributed call using ribbon and eureka naming server

Currency conversion service

### Project Work 2 Step 5 ###

### Project Work 2 Step 8 as we are going to fetch from eureka naming server ###

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

<http://localhost:8100/currency-convert-feign/from/AUD/to/INR/quantity/30>

in response port will change automatically

Step 9

Setting and implement Zuul API Gateway

* Authentication, authorization, security
* Rate Limits
* Fault Tolerance
* Service Aggregration

netflix-zuul-api-gateway-server

<dependency>

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

<artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

<dependency>

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

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

</dependency>

@SpringBootApplication

//Project Work 2 Step 9

@EnableZuulProxy

@EnableDiscoveryClient

**public** **class** NetflixZuulApiGatewayServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(NetflixZuulApiGatewayServerApplication.**class**, args);

}

}

### Project Work 2 Step 9 ###

#Enable all endpoints for actuator

#management.endpoints.web.exposure.include=\*

spring.application.name=netflix-zuul-api-gateway-server

server.port=8765

eureka.client.service-url.default-zone=http://localhost:8761/eureka

Step 10

Implementing Zuul logging filter

netflix-zuul-api-gateway-server

**package** com.udemy.ms28min.filter;

**import** javax.servlet.http.HttpServletRequest;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.stereotype.Component;

**import** com.netflix.zuul.ZuulFilter;

**import** com.netflix.zuul.context.RequestContext;

**import** com.netflix.zuul.exception.ZuulException;

@Component

**public** **class** ZuulLoggingFilter **extends** ZuulFilter{

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(ZuulLoggingFilter.**class**);

@Override

**public** **boolean** shouldFilter() {

//to execute the filter or not

**return** **true**;

}

@Override

**public** Object run() **throws** ZuulException {

HttpServletRequest request = RequestContext.*getCurrentContext*().getRequest();

***LOGGER***.info("request {} -> request uri {}", request, request.getRequestURI());

**return** **null**;

}

@Override

**public** String filterType() {

**return** "pre";

}

@Override

**public** **int** filterOrder() {

//order of this filter. if multiple filters are present

**return** 1;

}

}

Now run currency conversion service through Zuul API gate way server.

<http://localhost:8100/currency-convert-feign/from/AUD/to/INR/quantity/30>

[http://localhost:8765/{application-name}/{uri}](http://localhost:8765/%7bapplication-name%7d/%7buri%7d)

<http://localhost:8765/currency-conversion-service/currency-convert-feign/from/AUD/to/INR/quantity/30>

INFO 18220 --- [nio-8765-exec-6] c.u.ms28min.filter.ZuulLoggingFilter : request org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@dbd729e -> request uri /currency-conversion-service/currency-convert-feign/from/AUD/to/INR/quantity/30

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

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

It will print

INFO 18220 --- [nio-8765-exec-1] c.u.ms28min.filter.ZuulLoggingFilter : request org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@749aae26 -> request uri /currency-exchange-service/currency-exchange/from/USD/to/INR

Step 11

Routed Zuul API gate way between microservices instead of direct call

package com.udemy.ms28min.service;

currency-conversion-service

package com.udemy.ms28min.service;

import org.springframework.cloud.netflix.ribbon.RibbonClient;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.udemy.ms28min.bean.CurrencyConversionBean;

/\*

\* comment for @RibbonClient as because it will load and balance multiple instance service

\*/

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

/\*

\* comment for routing through Zuul api gate way.

\* Instead of hard coded pick the service name from zuul api service

\* project work 2 step 11

\*/

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

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

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

public interface CurrencyExchangeServiceProxy {

//appened as because we are picking the application name from naming server. project work 2 step 11

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

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

public CurrencyConversionBean retriveExchangeValue(@PathVariable String from, @PathVariable String to);

}

<http://localhost:8100/currency-convert-feign/from/AUD/to/INR/quantity/30>

Now call routed through Zuul api gate way

U can fetch log from api gate way console

2018-12-10 20:25:10.192 INFO 18220 --- [nio-8765-exec-9] c.u.ms28min.filter.ZuulLoggingFilter : request org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@77f8782d -> request uri /currency-exchange-service/currency-exchange/from/AUD/to/INR

Step 12

Distributed tracking with Zipkin

Implement Spring cloud Sleuth

It will add a unique id , so we can place it in multiple component

<!-- spring cloud sleuth dependencies Project Work 2 Step 12 start-->

<dependency>

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

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

</dependency>

<!-- spring cloud sleuth dependencies Project Work 2 Step 12 end-->

Add this entry in pom.xml file for currency-conversion-service, currency-exchange-service, netfliz-zuul-api-gateway-server

@Bean

public Sampler defaultSampler() {

return Sampler.ALWAYS\_SAMPLE;

}

Add this entry in starter class for currency-conversion-service, currency-exchange-service, netfliz-zuul-api-gateway-server

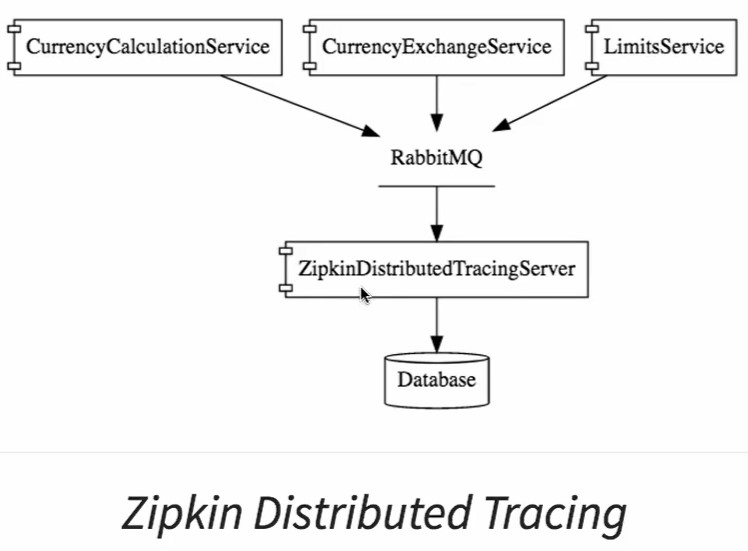
Run application in right way

Naming server -> zuup api srever -> exchange -> conversion

2018-12-10 21:41:34.442 INFO [currency-conversion-service,d0b824be421695c0,d0b824be421695c0,true] 12240 --- [nio-8100-exec-3] c.u.m.c.CurrencyConversionController : Bean : CurrencyConversionBean [id=10001, from=USD, to=INR, conversionMultiple=69.00, quantity=null, totalCalculatedAmount=null, port=8000]

Spring cloud sleuth create and assign Id request and log for that id

Centralized logging



Install Rabbit MQ

Before that we need to install erlang

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

<https://www.rabbitmq.com/install-windows.html>

Install zipkin distributed tracing server

Download zipkin foe windows

<https://zipkin.io/pages/quickstart>

java -> latest release

Copy the jar to any specific folder

Goto folder directory open command prompt

D:\dev\_env\dev\_workspace\udemy\005. udemy microservice with cloud in28min>java -jar zipkin-server-2.11.8-exec.jar

<http://localhost:9411/zipkin/>

Connect rabbit mq server with zipkin

D:\dev\_env\dev\_workspace\udemy\005. udemy microservice with cloud in28min >set RABBIT\_URI=amqp://localhost

D:\dev\_env\dev\_workspace\udemy\005. udemy microservice with cloud in28min>java -jar zipkin-server-2.11.8-exec.jar

Start rabbit mq server

<!-- spring cloud sleuth dependencies Project Work 2 Step 12 start-->

<dependency>

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

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

<!-- send message to mq so this dependencies required -->

<dependency>

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

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

<!-- spring cloud sleuth dependencies Project Work 2 Step 12 end-->

Add in all 3 application, conversion, exchange, api gate way

If u are using spring boot version 2.1 > version then add below dependencies instead of above one

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-rabbit</artifactId>

</dependency>

Step 13

Fault tolerance with hystrix

<!-- project work 2 step 13 add fault tolerance dependencies start -->

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

<!-- project work 2 step 13 add fault tolerance dependencies end -->

package com.udemy.ms28min;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.hystrix.EnableHystrix;

import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;

import org.springframework.cloud.openfeign.EnableFeignClients;

import org.springframework.context.annotation.Bean;

import brave.sampler.Sampler;

@SpringBootApplication

@EnableFeignClients("com.udemy.ms28min")

//Project work 2 Step 7

@EnableDiscoveryClient

//Project work 2 Step 13

@EnableHystrix

@EnableHystrixDashboard

@EnableCircuitBreaker

public class CurrencyConversionServiceApplication {

public static void main(String[] args) {

SpringApplication.run(CurrencyConversionServiceApplication.class, args);

}

@Bean

public Sampler defaultSampler() {

return Sampler.ALWAYS\_SAMPLE;

}

}

@GetMapping("/currency-convert-feign/from/{from}/to/{to}/quantity/{quantity}")

@HystrixCommand(fallbackMethod = "fallbackRetriveConfiguration")

**public** CurrencyConversionBean getConversionValueFeign(@PathVariable String from, @PathVariable String to,

@PathVariable String quantity) {

CurrencyConversionBean bean = proxy.retriveExchangeValue(from, to);

***LOGGER***.info("Bean : {}", bean);

BigDecimal quantityNo = **new** BigDecimal(quantity);

**return** **new** CurrencyConversionBean(bean.getId(), from, to, bean.getConversionMultiple(), quantityNo,

quantityNo.multiply(bean.getConversionMultiple()), bean.getPort());

}

**public** CurrencyConversionBean fallbackRetriveConfiguration(String from, String to, String quantity) {

**int** port = Integer.*parseInt*(environment.getProperty("local.server.port"));

BigDecimal quantityNo = **new** BigDecimal(quantity);

**return** **new** CurrencyConversionBean(0l, from, to, BigDecimal.*valueOf*(0), quantityNo, BigDecimal.*valueOf*(0), port);

}

Now Stop currency-exchange-service service

So currency-conversion-service method throw exception -> fallback method call happened and it will return default vale

<http://localhost:8100/hystrix> (work in edge)

<http://localhost:8100/hystrix.stream>

<http://localhost:8100/turbine/turbine.stream>