**Spring CLOUD**

When we implement micro services architectures we can create any number of micro services by using the modules provided by Spring Boot.For example if we are working on a hospital management software we can create a patient registration micro service a patient clinical micro service patient claim management and patient bed management micro services. But there will be several non-functional requirements for these micro services starting with service registration and discovery. That is each macro service will have to register itself with a centralized server and the other micro services will be able to discover that particular micro service and communicate with it dynamically without that each macro service will be tightly coupled to another micro service. So that is number one non-functional requirement.

Load balancing as the Load to our micro services increase will have to have multiple instances of the same micro service running on different servers and the load should be balanced.

Thirdly if something goes wrong in one of these micro services then entire systems should not come down or collapse. So these Microservice results should be fault tolerant and they should handle the faults gracefully. Easy integration they should be able to communicate with each other easily through restful client API. As for the API in easy steps.

Next will also how cross cutting concerns which are common across these macro services like security could be authentication authorization logging etc. which are common requirements across these micro services. So instead of repeating those cross cutting concerns across these micro services we should have one place where we can address them.

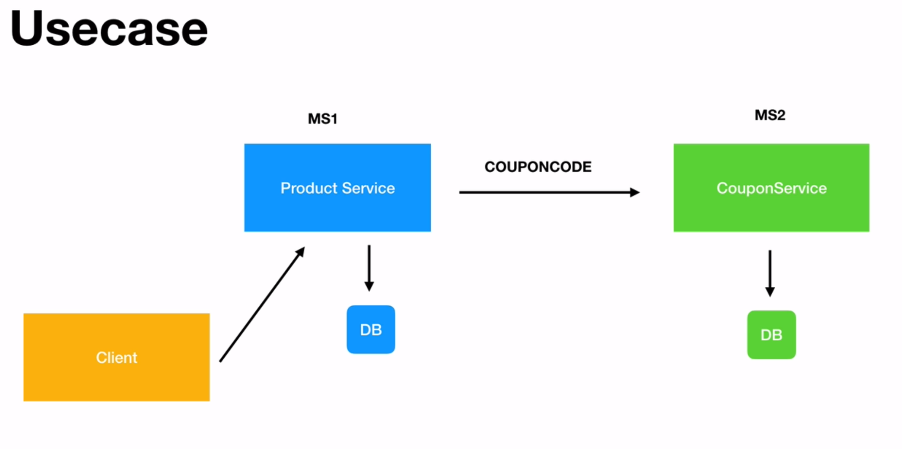
Last but not the least distributed tracing as the requests go from one micro service to another for example clinical to registration or registration too clinical. Too bad management. We should be able to trace how the requests are going and how the responses are coming back. When something goes wrong we can exactly pinpoint where it went wrong. Using distributed tracing so all these are not available in Spring Boot.

That is where ***spring cloud*** comes in spring cloud is a collection of open source components that help us implementing all these and more.

**Ports**

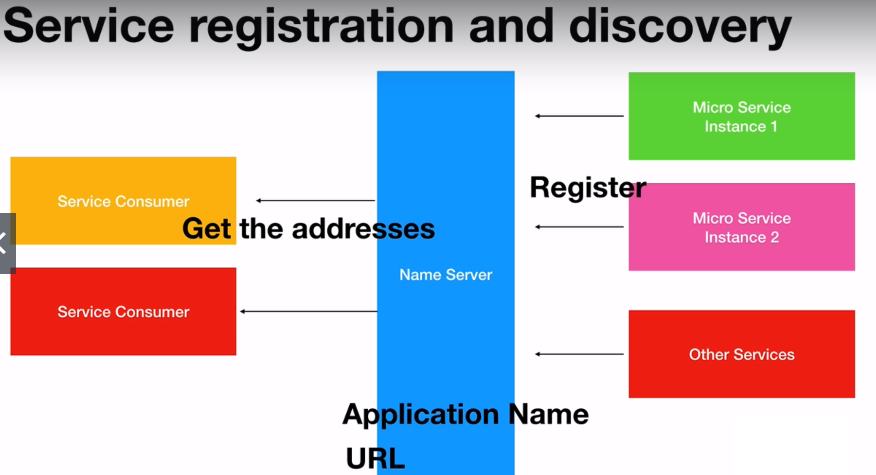
|  |  |
| --- | --- |
| ***Microservice Component*** | ***Ports Used*** |
| ***Coupon Service*** | ***8080 & 881*** |
| ***Product Service*** | ***9090*** |
| ***Eureka Server*** | ***8761*** |
| ***Zuul API Gateway Server*** | ***8765*** |
| ***Config Server (Spring Cloud Config)*** | ***8888*** |
| ***Zipkin Distributed Tracing Server*** | ***9411*** |

**Use case –**

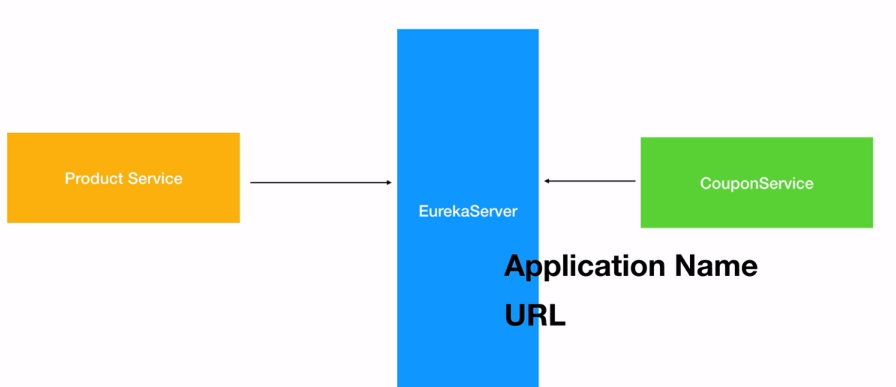


**Eureka**

When we have multiple Microservices running or multiple instances of the same Microservices running. There will be multiple Microservices that require or communicate with these micro services through restful calls and to do that the consumers will have to know the URL details the details the port number etc. in order to communicate with the appropriate micro service. This will be very hard to maintain because there could be multiple instances of the same Microservices is running one of the instances could be down at a given point in time. So, this dynamism is hard to maintain. ***That is where spring cloud offers naming services or naming Server called Eureka.***

******

Micro services will register themselves as soon as they start or come up they will register themselves with this naming server through an application name or application ID which is a unique ID for each application and also that URL that is required to communicate with that. Server will automatically be fetch by this name server and it will store all that information once all that information is stored in Eureka Code naming Server the service consumers when they come up can communicate with the naming server fetch those details based on just the application name as long as the consumers know the application ID the unique application ID they can fetch the URL port number. All that communication details will be maintained by the Eureka server so Eureka server will decouple their micro services that are running and they can communicate with any other micro services easily through registration and discovery without maintaining any URL etc. That are required otherwise.



**Development Process**

**Part 1**

1. **Create a Spring Starter project (Eureka Server)**
2. **Add dependencies on pom.xml**
3. **Add annotation @EnableEurekaServer in the main application file.**
4. **Define the port no. and turning off defaults in *application.properties* file**
5. **Run the localhost on browser**

**Part 2**

1. **Create a client service for the Eureka Server**
2. **Add dependencies on pom.xml**
3. **Add annotation @EnableEurekaClient in the main application file.**
4. **Define application name, url of the Eureka server in the *application.properties* file.**

**Part 1**

1. **Creating a Spring Starter project**
2. **Adding dependencies on pom.xml**

****

****

1. **Adding annotation @EnableEurekaServer in the main application file**

****

1. **Defining the port no. and turning off defaults in *application.properties* file**

****

1. **Run it as Spring Boot App or as application after right click on main driver app or whole project.**

<http://localhost:8761/>

**Part 2**

1. **Creating Client Service for Eureka Server (Controllers, Model or Entity, Repo – not adding full code)**
2. **Adding dependencies on pom.xml**

****

****

1. **Adding annotation @EnableEurekaClient in the main application file**

****

1. **Defining application name, url of the Eureka server in the *application.properties* file.**

****

**Rest Clients Using Feign**

Two micro services coupon service and product service for the product service. To work it needs to call the coupon service and get a coupon so that the discount can be applied to Product service will become a restful client to the coupon service.

Spring mvc offers the rest template the spring web has something called rest template that can be used. To make those restful calls but using spring rest template will how to do a lot of coding.

On the other hand spring cloud offers a Feign claimed project using this feign client we can declaratively create restful clients. It's very easy to create restful clients using feign clients and it integrates pretty well with Eureka and other projects in spring cloud.

**Development process**

1. **Go to the created product service and add** **dependency to the pom.xml called open feign**
2. **Enable the support for the open feign by going to the root class at enable feign clients annotation will be added that will add the feign client support for our project.**
3. **Create a coupon client. (interface, not an class)**
4. **Provide methods to it.**
5. **Bind any variables using path variable**
6. **On the client interface you will market with @ find feign client with application id or name (with which it is registered on Eureka Server)**
7. **Invoke it in the respective rest controller.**
8. **Adding dependency to the pom.xml called open feign**

****

****

1. **Enabling the support for the open feign by going to the root class at enable feign client’s annotation will be added that will add the feign client support for our project.**

****

1. **Creating a coupon client (interface)**
2. **Adding methods**
3. **Binding variables using Path Variable**
4. **The client interface you will market with @ find feign client with application id or name (with which it is registered on Eureka Server)**

****

**Coupon added as entity**

****

1. **Invoking it in the respective rest controller**

****

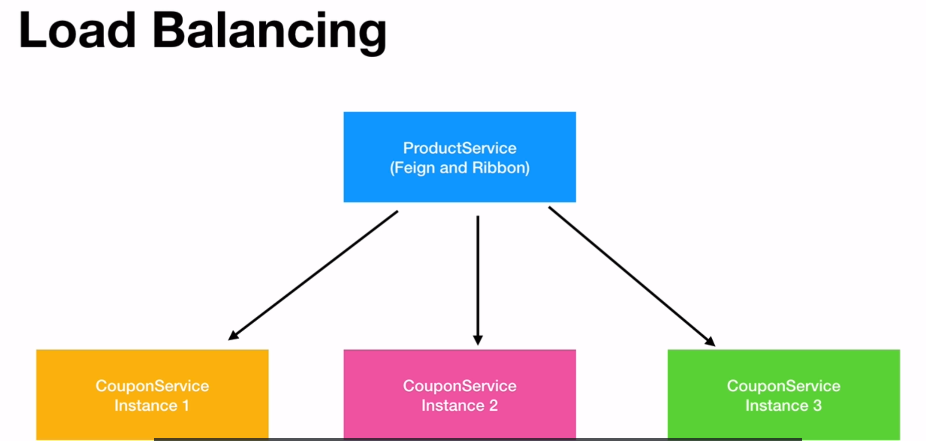
**Product.java (where CouponCode is being added)**

****

****

**Client-Side Load Balancing using Ribbon**

As the number of requests to our micro services increase we will be scaling our application across different servers. That is we will apply the same micro service and there will be multiple instances of the same micro service running on different servers so that the client can make calls to one of these instances as required. So instead of all the clients calling into the same instance the load should be distributed across these multiple instances of Micro service running here we have a product service which can make the first call to the coupon service instance running on the first server then the next request will go to the second server and so on to do this client side. Load balancing we use a ribbon from Spring cloud Feign client which you have already used beautifully integrates with ribbon and the Load will be balanced from the client side itself. Later on you will see how to do some outside load balancing using zuul and ribbon. For now this is client side. Load balancing the client itself will decide to which instance the request should go to.



Development Process

1. Add dependency in the pom.xml
2. Go to restful client and add this annotation @RibbonClient provided the application ID with which the application has registered on eureka

Again, ribbon and faith will work together along with Eureka to balance the Load on the client site that is on the client side. The decision will be made to which instance of the micro service the request should be sent and then it will be applying the coupon service on multiple servers it's super simple will run the same service on multiple ports of our machine so that there will be two instances of Tomcat two instances of coupon service and they will make calls from the product service and those calls will be going to different instances instead of going to the same instance of coupon service.