Micro Service Intra Communication

(For this we need to use 3 types of client comps)
a) Discovery Client b) LoadBalancerClient c) Feign client
(best)

Limitations of DiscoveryClient type ClientComp

(a) We get list of target MS instances and we need to choose instance manually.. But actually want one instance of target MS (Producer MS) which having less Load Factor (Load balancing is not possible)

(b) This Basic Client Comp or Client Type which very much Legacy.. i.e it is not industry standard..

(c) Collecting one instance from the list of instance is the responsibility of Consumer App that is manual process.. So other instances of target/producer MS may sit idle..

(d) we need to write all the logic in Client Comp manually (means we need to use RestTemplate explicitly to communicate with Target MS)

note:: To overcome these problems take the support of LoadBalancerClient type Client Comp.

LoadBalancerClient

Load Factor = current Load/Max Load

=> It is another Client type Comp or Client Comp.. which choose the less load factor instance though they are multiple instances for target/Provider MS i.e we never get list of instances though they are multiple instances for MS.. we always get one instance of MS which is having Less Load Factor
(Producer MS) (Producer MS)

=> LoadBalancerClient is an interface and implementation is given by Spring Cloud Netflix people In the form of RibbonLoadBalancerClient class which can be injected to Consumer App through Autowiring process

(In spring boot 3.x the the impl class name of the LoadBalancerClient (I) is org.springframework.cloud.loadbalancer.blocking.client.BlockingLoadBalancerClient)

=> The method choose(-) with instanceId called on the LoadBalancerClient obj will bring the less Load Factor instance of target/producer MS.

note:: To create multiple instances for any MS run that Ms App for multiple times with different Port numbers becoz two Servers or two Ms of same computer can not take same port number .. but possible across the multiple machines.
(Do not add "dev tools" in the support to this process)

Example App using LoadBalancerClient Type Client Comp (MS Intra communication using LoadBalancerClient)

step1) Develop Eureka Server App

same as previous App

step2) Develop Producer/Provider/Target MS
(This time provide random number as the instance id)

=> Since we are planning to take multiple instances of producer App by running the Producer App for multiple times it is better to give separate name for every instance generally it is serviceId:randomvalue using

```
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

App name as the ServiceId — System property giving one pseudo random number for every execution

(do not add dev tools)
=> create project having spring web, eurekaDiscoveryClient dependencies
=> add @EnableEurekaClient on the top main class (In spring boot 3.x it is @EnableDiscoveryClient)
=> add the following entries in application.properties

application.properties

```
#MS Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# provide application + random value as instance id
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

=> Develop RestController activing Producer MS

```java
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/billing/api")
public class BillingInfoController {
	@Value("${server.port}")
	private int port;
	@Value("${eureka.instance.instance-id}")
	private String instanceId;

	@GetMapping("/info")
	public ResponseEntity<String> fetchBillingDetails(){
	 return new ResponseEntity<String>("Final Bill Amt = Bill Amt - discount (Rs.5000) :: using instance::-->"+instanceId+" @port::"+port,HttpStatus.OK);
	}

}
```

step3) Develop the Consumer App with support of LoadBalancerClient

(do not add dev tools)
=> create project having spring web, eurekaDiscoveryClient dependencies (Ribbon comes automatically)

Micro Service Intra Communication =======

========

Limitations of DiscoveryClient type ClientComp

=======

chooose

(For this we need to use 3 types of client comps)

a) Discovery Client b) LoadBancerClient c) FeignClient

We

(a) We get list of target MS instances and we need to instance manually.. But actually want one instance of target MS(Producer Ms) which having less Load Factor (Load balacing is not possible)

is

comp

is

n

(b) This Baisc Client Comp or Client Type which very much Legacy .. i.e it is not indurstry standard..

S the

n

(c) Collecting one instance from the list of instance is pure responsibity of Cosumer App that to manual process.. So other instancces of target/produce Ms may sit idle..

(best)

(d) we need to write all the logics in Client Comp manually (means we need to use RestTemplate explicitly to communicate with Target MS)

note:: To overcome these problems take the support of Load BalancerClient type Client Comp.

LoadBalancerClient

==========

Load Factor = current Load/Max Load

=> It is another Client type Comp or Client Comp.. which choose the less load factor instance though they are multiple instances for Target /Procuder Ms i.e we never get List of instances though they are muliple instances for MS.. we always get one instance of MS which is having Less LoadFactor (Producer MS)

(Producer MS)

=>LoadBalnaceClient is an interface and implementation is given by Spring Clould Netflix people in the form of RibbonLoadBalanceClient class which can be injected to Cosumer App through Autowiring .. process

(upto 2.x)

n

=> The method choose(-) with instance Id called on the Load BalanceClient obj will bring the Less LoadFactor Instance of target/producer Ms.

(In spring boot 3.x the the impl class name of the LoadBalancerClient (I) is

org.springframework.cloud.loadbalancer .blocking.client.BlockingLoadBalancerClient)

note:: To create multiple instances for any MS run that Ms App for multiple times with different Port numbers becoz Two Servers or Two Ms of same computer can not take same port number .. but possible_across the multiple machines. (Do not add "dev tools" in the support to this process)

Example App using Load Balance Client Type Client Comp (MS intra communication using Load BalanceClient)

==========

step1) Develop Eureka Server App

same as previous App

step2) Develop Producer/Provider/Target MS

(This time provide random number as the instance Id)

=>Since we are planning take multiple instances of producer App by running the Producer App for multiple times it is better to give seperate name for every instance generally it is serviceId:<randomvalue> using

eureka.instance.instance-id=${spring.application.name}:${random.value}

App name as the ServiceId

NT System property

giving one psuedo random

number for every execution

(do not add dev tools)

=> create project having sprigng web, eureka DisoveryClient, dependencies

=> add @EnableEurekaClient on the top main class (In spring boot 3.x it is @EnableDiscoveryClient)

=> add the following entries in application.properties

application.properties

```
#Ms Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# provid applicaiton + radom vlaue as Instance Id
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

=> Develop Restcontroller actiging Producer MS

```
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/billing/api")
```

```
public class BillingInfoController {
@Value("${server.port}")
private int port;
@Value("${eureka.instance.instance-id}")
private String instanceid;
@GetMapping("/info")
public ResponseEntity<String> fetchBillingDetails(){
return new ResponseEntity<String>(" Final BillAmt= BillAmt- discount (Rs.5000) :: using instance::-->"+instanceid+" @port::"+port, HttpStatus.OK);
step3)
}
}
r
```

Devlop the Consumer App with support of Load BalanceClient

(do not add dev tools)

=> create project having sprigng web, eureka DisoveryClient dependencies (Ribbon comes automatically)

=> add @EnableEureka Client on the top main class (in spring boot 3.x use @EnableDiscoveryClient)

=> add the following entries in application.properties

```
#Ms Properties
#Port number
server.port=6600
#Service id
spring.application.name=Shopping-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

=> develop helper having Load BalanceClient comp Injection

```
package com.nt.client;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.Load BalancerClient; import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
@Component
public class BillingServiceConsumerClient {
@Autowired
private LoadBalancerClient client;
public String getBillingInfo() {
}
```

on

```
// Get Billing-Service Instance from eureka server based LoadFactor ServiceInstance
instance=client.choose("Billing-Service");
// get details from Serivce Instance
URI uri=instance.getUri();
//prepare provider MS related url to cosume method
String url=uri.toString()+"/billing/api/info";
//create RestTemplate class obj to cosume the provider service RestTemplate template=new RestTemplate();
// consume the provider service
ResponseEntity<String> response-template.getForEntity(url,String.class);
77 get response content from ResponseEntity object
String responseContent-response.getBody();
return responseContent;
```

=>When we add Eureka DiscoveryClient starter to the Project

we get the following client comps automatically through AutoConfiguration

a) DiscoveryClient object (basic client comp)

b) LoadBalancerClient object (advanced client comp)

Develoe the RestController

=>Develoe

```
@RestController
@RequestMapping("/shopping/api")
public class ShoppingSerivceOperationsController {
@Autowired
private BillingServiceConsumerClient client;
@GetMapping("/cart")
public ResponseEntity<String> doShopping(){
//use Client Comp
String resultMsg=client.getBillingInfo(); try {
}
Thread.sleep(20000);
catch(Exception e) {
}
}
}
e.printStackTrace();
return new ResponseEntity<String>("Shopping the items(shirt,trouser) ::::"+resultMsg,HttpStatus.OK);
```

ep4) run Apps in the following order

=>run Eureka server App

each time

=> Run Producer App multiple times but change port number (server.port value)

in application.properties each time (at least 2 times)

=> Run the Cosumer App

=>Go to Eureka server Consle modify the

Consumer App url

Instances currently registered with Eureka

Application

AMIS

Availability Zones

Status

BILLING-SERVICE

n/a (2) (2) SHOPPING-SERVICE n/a (1) (1)

←

NATARAZ- Java Consultant"

UP (2) - Billing-Service:f66872cd649a60e71268d87e3eb8962c,
Billing-Service:a3ab84ba38566cc089de2eb7adf0ab2d UP (1) - 192.168.1.115:Shopping-Service:7070

(First request to Consumer)

192.168.1.115:7070/shopping/api x

You are screen sharing

■ Stop Share

192.16

hg/api x

+

Eureka

x

← →

C

▲ Not secure

192.168.1.115:7070/shopping/api/cart

Shopping the items(shirt,trouser) :::: Final BillAmt=BillAmt- discount (Rs.5000) :: using
instance::-->Billing-Service:5e427657df61ce4fad98375d8c222d62 @port::9090

A Not secure

192.168.1.115:7070/shopping/api/cart

Second request to consumer

Shopping the items (shirt, trouser) :::: Final BillAmt= BillAmt- discount (Rs.5000):: using
instance::-->Billing-Service:3e9ef0ef588518c3827ff66e74d8bbd9 @port::9091

Important maven goals to use from eclipse IDE

========

=========

============

To prepare jar or war file representing the current application

right click on the project --->

===

from

two different

browser s/ws

or windows or tabs

run as ---> maven build.. ---> goals :: package

note:: gives the war/jar file in the target folder of the maven project

To prepare jar or war file representing the current application and to keep that file maven local repository

right click on the project --->

run as ---> maven build.. ---> goals :: install

note:: gives the war/jar file in the target folder of the maven project and keeps c:\windows\user\.m2\repository\..

To clean the content from the target folder

right click on the project --->

run as ---> maven build..> goals::clean

To create war/jar file by cleaning the existing file

right click on the project --->

run as ---> maven build..> goals: clean package

1 Run on Server

2 Java Application

Ju 3 JUnit Test

m2 4 Maven build...

m2

5 Maven clean

m2

6 Maven generate-sources

m2

7 Maven install

m2 8 Maven test

m2

9 Maven verify

Spring Boot App

Spring DevTools Client

What is the procedure of giving our own war /jar file name while packaging the code using the maven?

Ans)

<build>

<finalName>BillingService</finalName>

</build>