Microservices Project Steps :

**1.We create 2 services**

a. Departmentservice(port:8080) )(It is Eureka Client)

instance of Departmentservice(port:8082)(It is Eureka Client)

b. Employeeservice(port:8081) )(It is Eureka Client)

c. service-registry(port:8761)(It is Eureka Server)

Command To start the instance of Project JAR - ***java -jar departmentservice-0.0.1-SNAPSHOT.jar --server.port=8082***

**2.**We have created the instace of Department Service which is a replica od Department Service running on port 8082.

**3**.Netflix Eureka Server has a internal built in Load Balancer which can point to Departmentservice 8082 if Departmentservice 8080 is Down . just make change to Employeeservice(OpenFeign() Annotation).

**4.API Gateway**

MicroService-1

Port:8080

API

Gateway

Port:9191

MicroService-2

Port-8081

Client

MicroService-3

Service Registry

Or

Eureka Server

(Port 8761)

Porperties of API-GATEWAY:

spring.application.name=API-GATEWAY  
server.port=9191  
eureka.instance.client.serverUrl.defaultZone = http://localhost:8761/eureka/  
management.endpoints.web.exposure.include =\*  
  
## Routes for Employee Service  
spring.cloud.gateway.routes[0].id = EMPLOYEE-SERVICE  
##We can use below localhost 8081 as well but if we want to use the internal load balancer then use lb  
##spring.cloud.gateway.routes[0].uri= http://localhost:8081  
spring.cloud.gateway.routes[0].uri= lb://EMPLOYEE-SERVICE  
spring.cloud.gateway.routes[0].predicates[0]=PATH=/api/employee/\*\*  
  
##When user will make request to the http://localhost:9191/api/employee , this will route to employee service and  
## the user need not have to worry about the internal URL for employee i.e 8081 ,with help of api gateway{9191}  
## he will be routed to the Employee Service  
  
## Routes for Department Service  
spring.cloud.gateway.routes[1].id = DEPARTMENT-SERVICE  
##We can use below localhost 8081 as well but if we want to use the internal load balancer then use lb  
##spring.cloud.gateway.routes[1].uri= http://localhost:8080 0r http://localhost:8082(Instance of Department)  
spring.cloud.gateway.routes[1].uri= lb://DEPARTMENT-SERVICE  
spring.cloud.gateway.routes[1].predicates[0]=PATH=/api/departments/\*\*

We have changed the dependency from

<artifactId>spring-cloud-starter-gateway-mvc</artifactId>

To:

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

**6.Spring Cloud Bus**

***Docker Image Pull Commands***

***(Commands run in command prompt default directry)***

1. docker pull rabbitmq:3.12.13 (Once Time see Docker Desktop , there the image will be there).
2. docker run --rm -it -p 5672:5672 rabbitmq:3.12.13 (To start rabbitmq instance ).

**7.Distributed Tracing Problems**

If we have 100 of micreservices moduules in our project and if error occurs so it is quite difficult to trace that error using logs , and we can also find out how much time each microservice is taking .

M4

M3

M2

M1

M5

(Start of Chain) (End of Chain)

* What and why Distributed Tracing is used ?

Department

MicroService

Employee

MicroService

API

Gateaway

Client

Span id Span id Span id

Trace -> trace id

Distributed Tracing helps in

1.tracing end to end (from start to end) the request with common trace id and span id for each MS.

2.Gives us the time taken by each MS .

Two libraries are 1.Spring cloud Sleuth 2. Zipkin

3.Zipkin command from IDE terminal : java -jar zipkin-server-3.0.6-exec.jar

**8.Circuit Breaker Pattern:**

There are two methods by which it can be done like

1.Circuit Breaker 2.Retry Method

With the fallback methods we must pass the Exception in the fallback method or else it will throw the error and 500 error code and it will not work