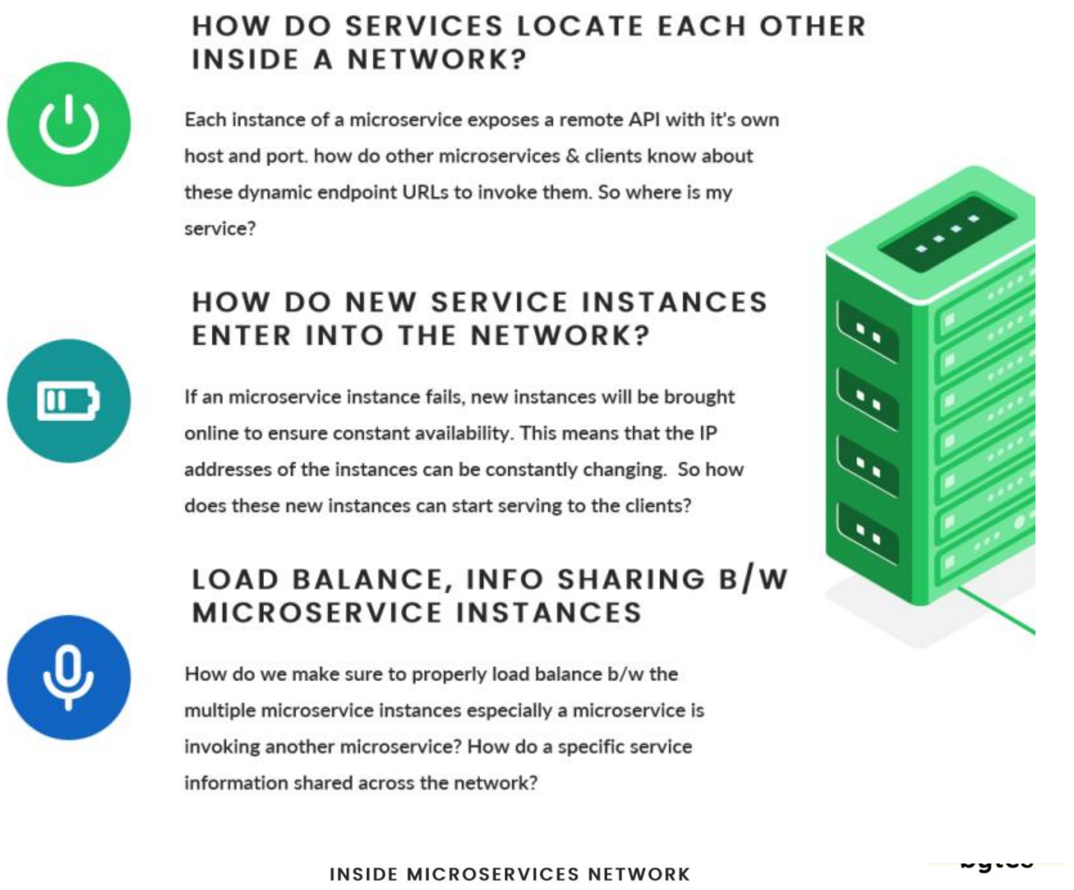
Service Discovery and Registration

-Shubham Ashtaputre

A] Necessity of Service Discovery and Registration

See the below image for issues that would occur if we don't use **Service Discovery and Registration**



HOW DO SERVICES LOCATE EACH OTHER INSIDE A NETWORK?

Each instance of a microservice exposes a remote API with it's own host and port. how do other microservices & clients know about these dynamic endpoint URLs to invoke them. So where is my service?

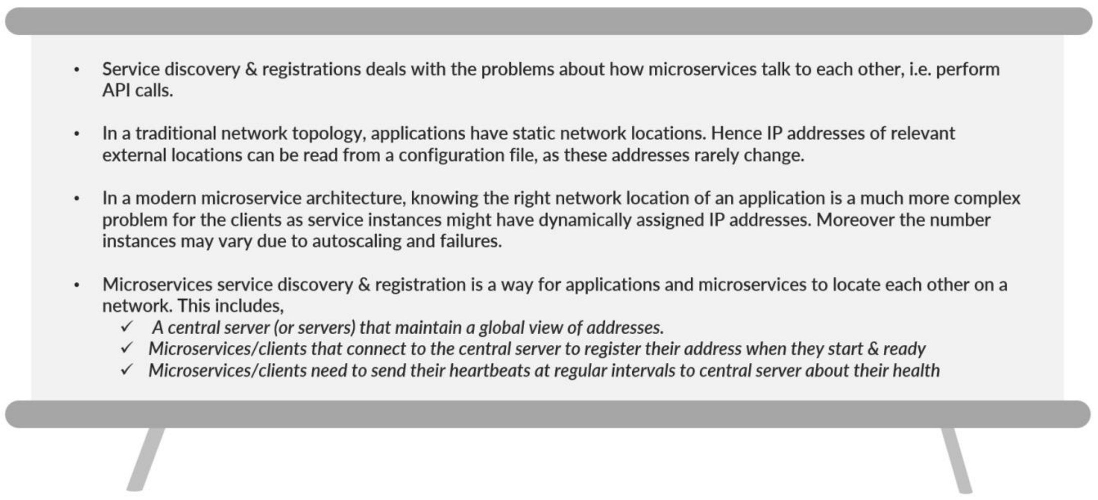
HOW DO NEW SERVICE INSTANCES ENTER INTO THE NETWORK?

If an microservice instance fails, new instances will be brought online to ensure constant availability. This means that the IP addresses of the instances can be constantly changing. So how does these new instances can start serving to the clients?

LOAD BALANCE, INFO SHARING B/W MICROSERVICE INSTANCES

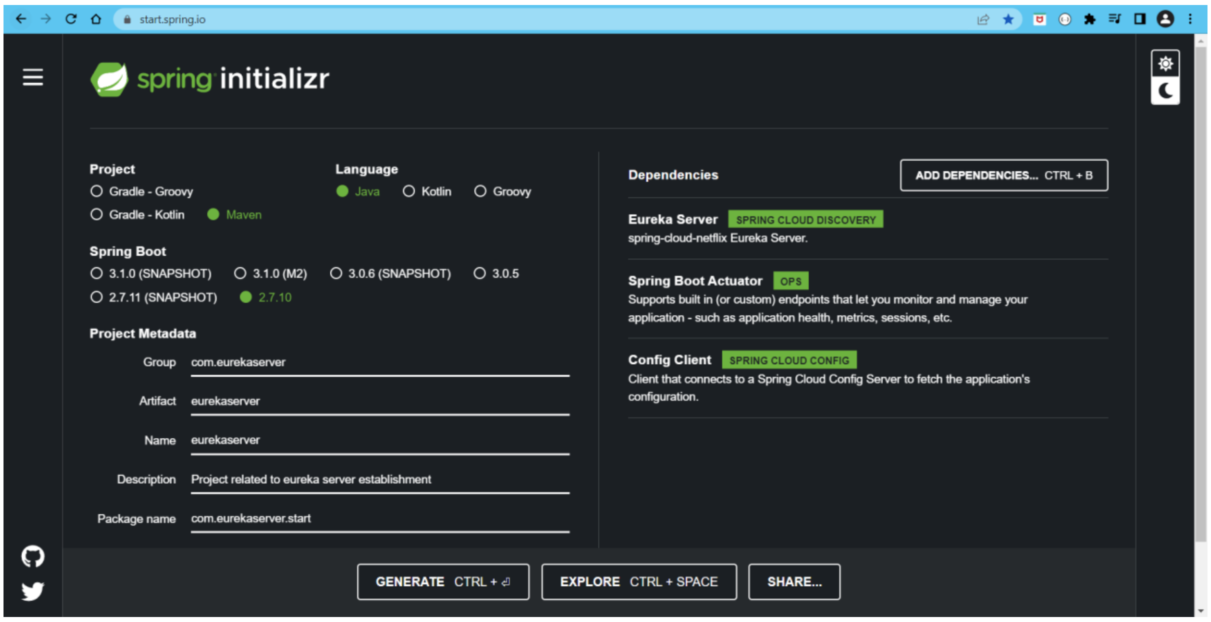
How do we make sure to properly load balance b/w the multiple microservice instances especially a microservice is invoking another microservice? How do a specific service information shared across the network?

INSIDE MICROSERVICES NETWORK



- Service discovery & registrations deals with the problems about how microservices talk to each other, i.e. perform API calls.
- In a traditional network topology, applications have static network locations. Hence IP addresses of relevant external locations can be read from a configuration file, as these addresses rarely change.
- In a modern microservice architecture, knowing the right network location of an application is a much more complex problem for the clients as service instances might have dynamically assigned IP addresses. Moreover the number instances may vary due to autoscaling and failures.
- Microservices service discovery & registration is a way for applications and microservices to locate each other on a network. This includes,
  - ✓ *A central server (or servers) that maintain a global view of addresses.*
  - ✓ *Microservices/clients that connect to the central server to register their address when they start & ready*
  - ✓ *Microservices/clients need to send their heartbeats at regular intervals to central server about their health*

B] Set Service discovery agent using Eureka server



Project
○ Gradle - Groovy ○ Gradle - Kotlin ● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.1.0 (SNAPSHOT) ○ 3.1.0 (M2) ○ 3.0.6 (SNAPSHOT) ○ 3.0.5 ○ 2.7.11 (SNAPSHOT) ● 2.7.10

Project Metadata

Group	com.eurekaserver
Artifact	eurekaserver
Name	eurekaserver
Description	Project related to eureka server establishment
Package name	com.eurekaserver.start

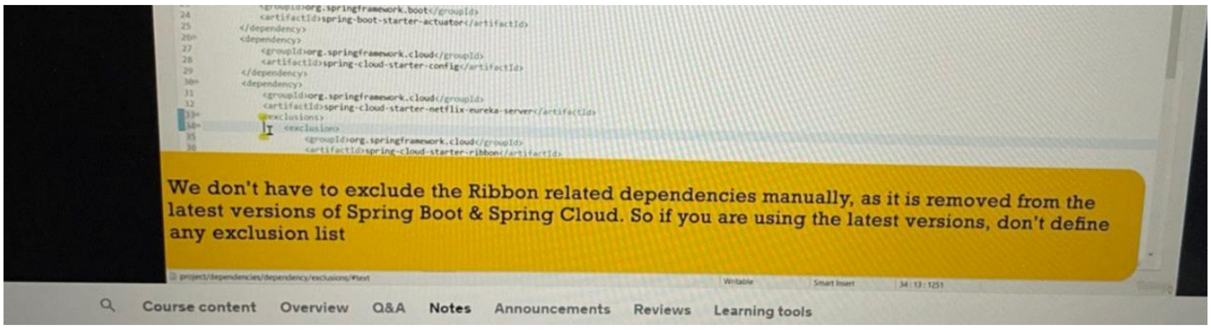
Dependencies

Eureka Server SPRING CLOUD DISCOVERY
spring-cloud-netflix Eureka Server.

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

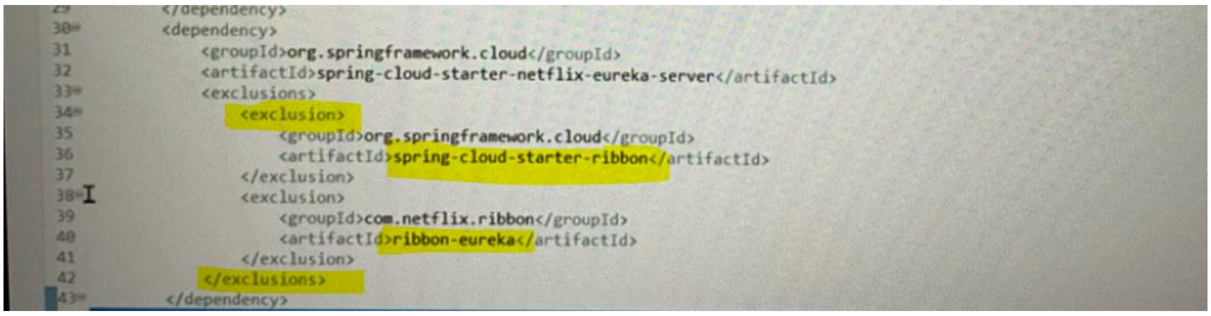
Config Client SPRING CLOUD CONFIG
Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

1] If you are using some old springboot version then you need to manually remove the ribbon dependency from eureka server as below:



```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <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>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-starter-ribbon</artifactId>
```

We don't have to exclude the Ribbon related dependencies manually, as it is removed from the latest versions of Spring Boot & Spring Cloud. So if you are using the latest versions, don't define any exclusion list

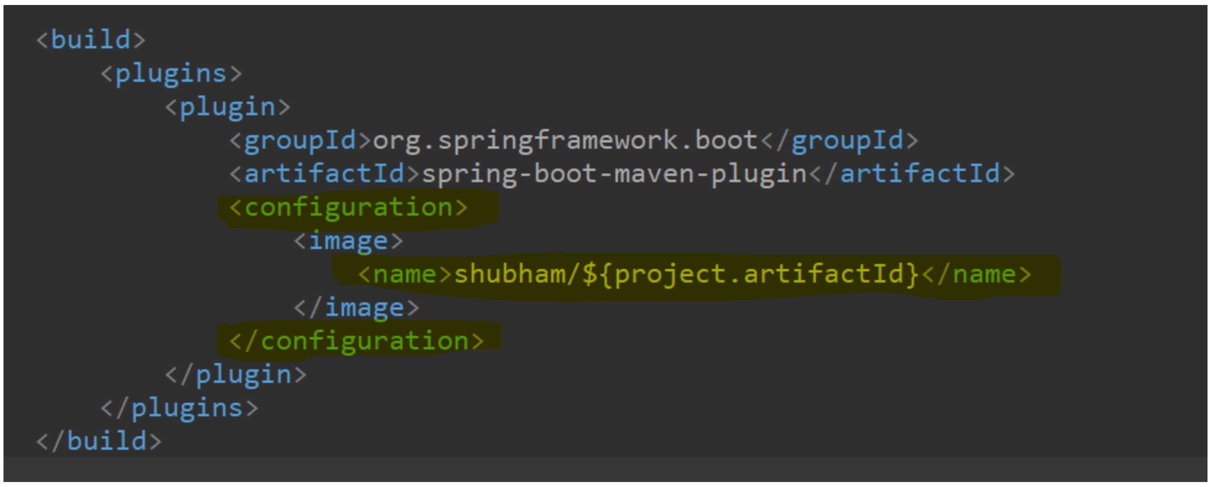


```
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-starter-ribbon</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>com.netflix.ribbon</groupId>
                    <artifactId>ribbon-eureka</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

2] But if you are using latest spring-boot version then it is already taken care of

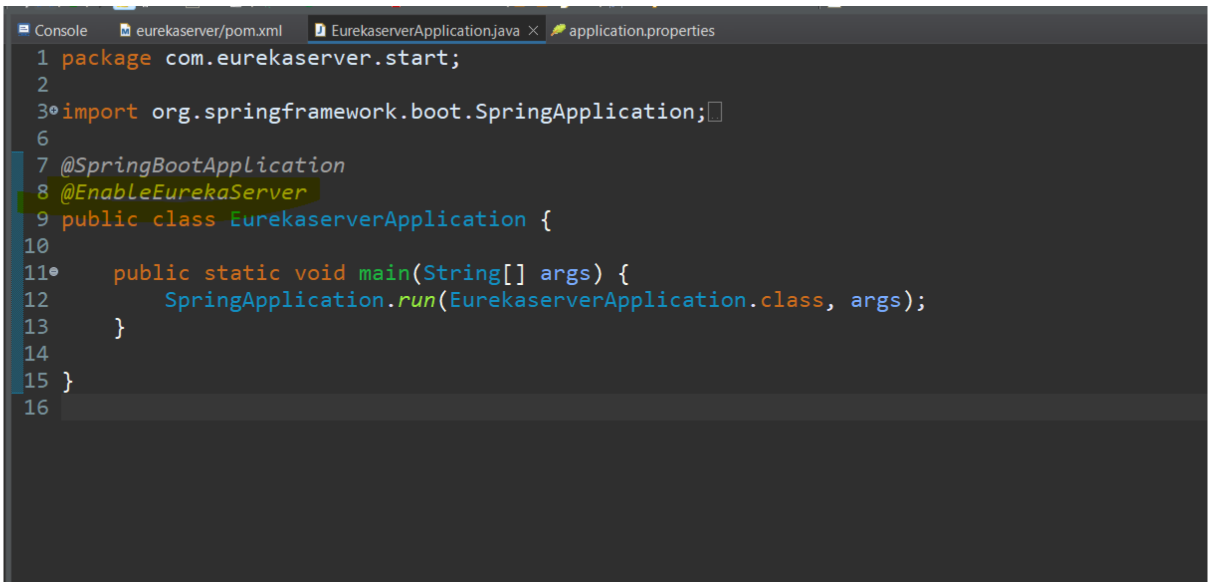
3] The reason we need to remove ribbon from eureka server is that we are going to use spring cloud load-balancer for our project

4] This will set how my image name of project



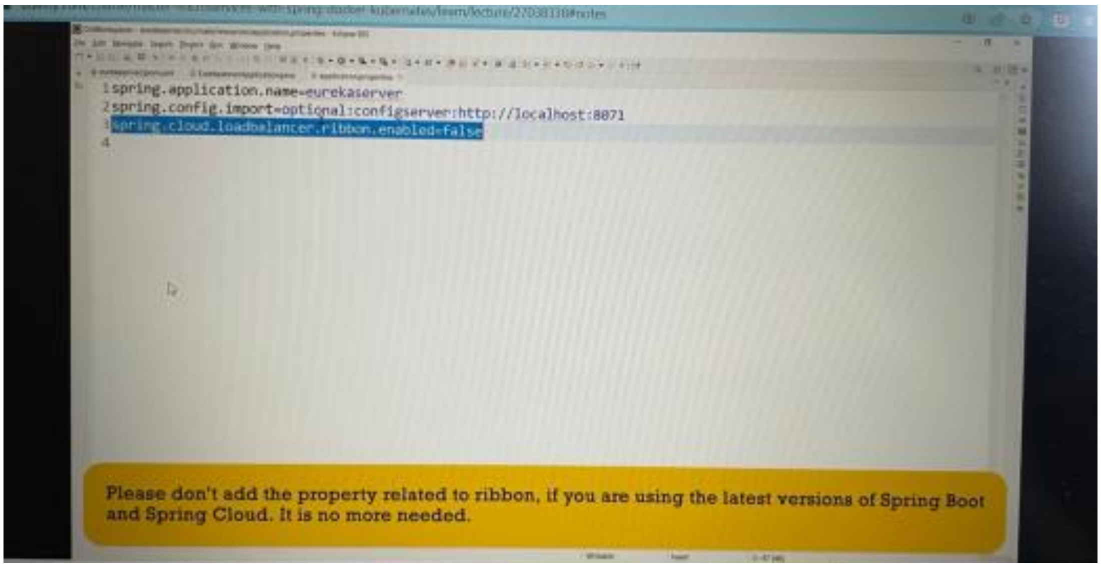
```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <image>
                    <name>shubham/${project.artifactId}</name>
                </image>
            </configuration>
        </plugin>
    </plugins>
</build>
```

5] Using **‘@EnableEurekaServer**’ annotation we will make our microservice act as service discovery agent using eureka server



```java
package com.eurekaserver.start;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableEurekaServer
public class EurekaserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaserverApplication.class, args);
    }

}

```

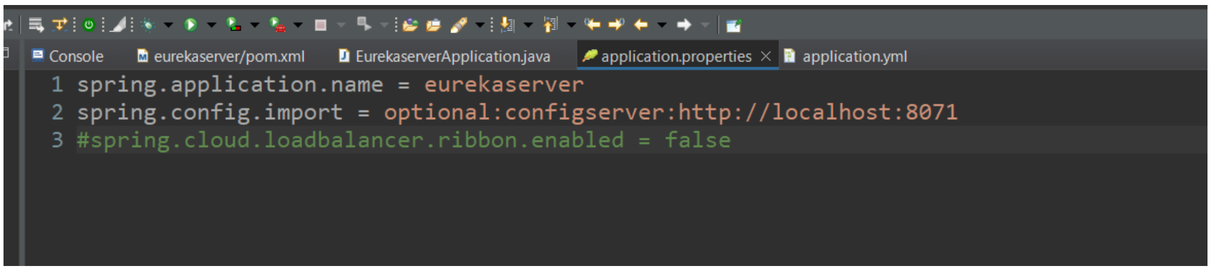
6] Make changes in the properties file:



```
spring.application.name=eurekaserver
spring.config.import=optional:configserver:http://localhost:8071
spring.cloud.loadbalancer.ribbon.enabled=false
```

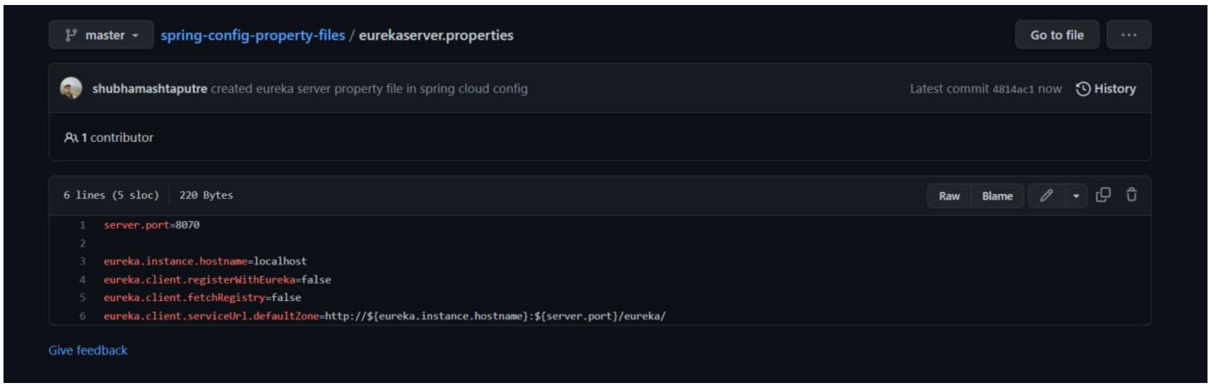
Please don't add the property related to ribbon, if you are using the latest versions of Spring Boot and Spring Cloud. It is no more needed.

7] Below are the changes that I had made inside properties file to connect to configuration server:



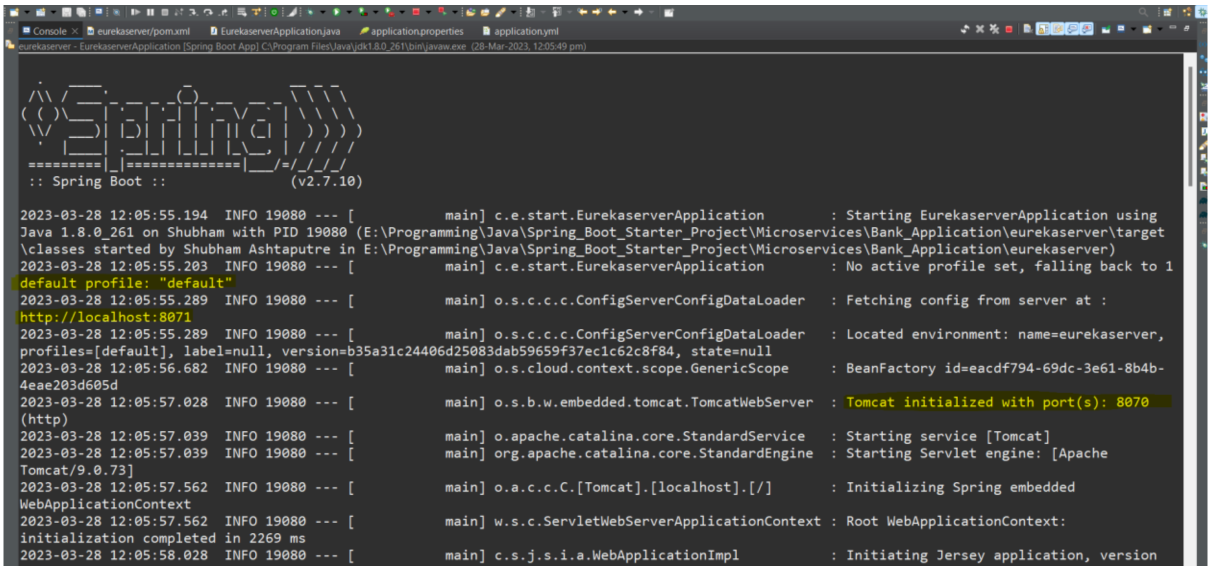
```
spring.application.name = eurekaserver
spring.config.import = optional:configserver:http://localhost:8071
#spring.cloud.loadbalancer.ribbon.enabled = false
```

8] Create eurekaserver.properties inside github location to read from spring cloud config


master
spring-config-property-files / eurekaserver.properties
Go to file
shubhamashtaputre created eureka server property file in spring cloud config
Latest commit 4814ac1 now
History
1 contributor
6 lines (5 sloc) | 220 Bytes
Raw
Blame
Give feedback

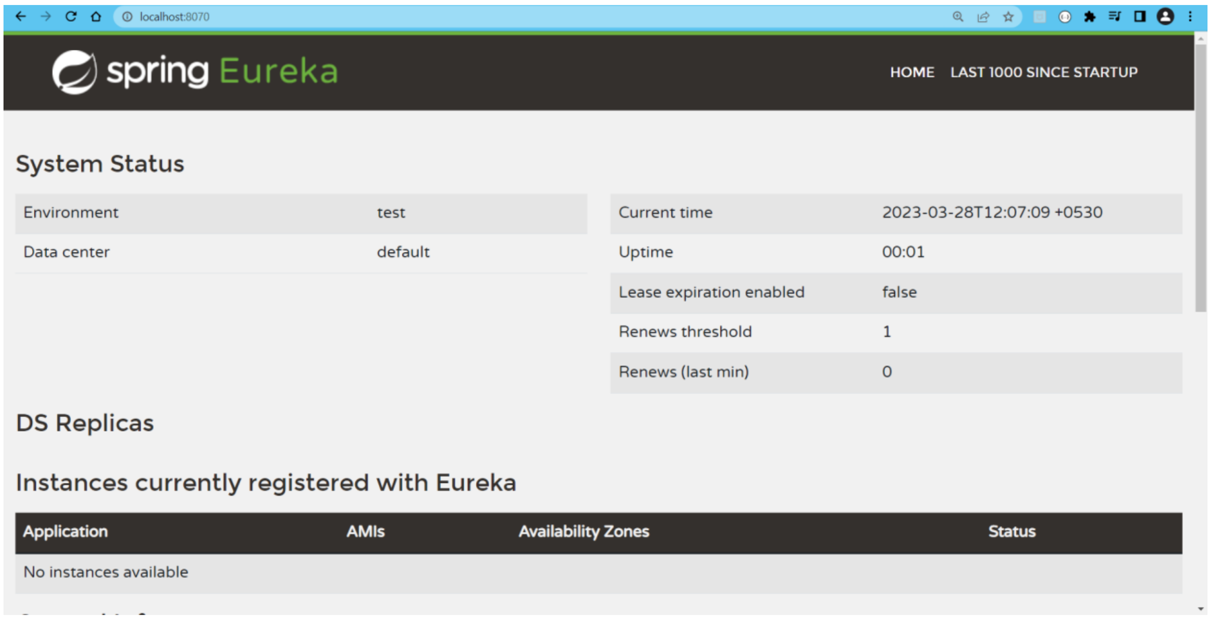
```
server.port=8070

eureka.instance.hostname=localhost
eureka.client.registerWithEureka=false
eureka.client.fetchRegistry=false
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/
```

9] Now, start your config server first as we are going to read eureka server properties from the config server and then start eureka server as below:


Console
eurekaserver/pom.xml
EurekaserverApplication.java
application.properties
application.yml
eurekaserver - EurekaserverApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_261\bin\javaw.exe (28-Mar-2023, 12:05:49 pm)

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::               (v2.7.10)

2023-03-28 12:05:55.194  INFO 19080 --- [           main] c.e.start.EurekaserverApplication        : Starting EurekaserverApplication using
Java 1.8.0_261 on Shubham with PID 19080 (E:\Programming\Java\Spring_Boot_Starter_Project\Microservices\Bank_Application\eurekaserver\target
\classes started by Shubham Ashtaputre in E:\Programming\Java\Spring_Boot_Starter_Project\Microservices\Bank_Application\eurekaserver)
2023-03-28 12:05:55.203  INFO 19080 --- [           main] c.e.start.EurekaserverApplication        : No active profile set, falling back to 1
default profile: "default"
2023-03-28 12:05:55.289  INFO 19080 --- [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Fetching config from server at :
http://localhost:8071
2023-03-28 12:05:55.289  INFO 19080 --- [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=eurekaserver,
profiles=[default], label=null, version=b35a31c24406d25083dab59659f37ec1c62c8f84, state=null
2023-03-28 12:05:56.682  INFO 19080 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=eacdf794-69dc-3e61-8b4b-
4eae203d605d
2023-03-28 12:05:57.028  INFO 19080 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8070
(http)
2023-03-28 12:05:57.039  INFO 19080 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-03-28 12:05:57.039  INFO 19080 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache
Tomcat/9.0.73]
2023-03-28 12:05:57.562  INFO 19080 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded
WebApplicationContext
2023-03-28 12:05:57.562  INFO 19080 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
initialization completed in 2269 ms
2023-03-28 12:05:58.028  INFO 19080 --- [           main] c.s.j.s.i.a.WebApplicationImpl           : Initiating Jersey application, version
```

10] Check the eureka server localhost you eureka server is up and running and our service discovery agent is setup



System Status

Environment	test
Data center	default

Current time	2023-03-28T12:07:09 +0530
Uptime	00:01
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

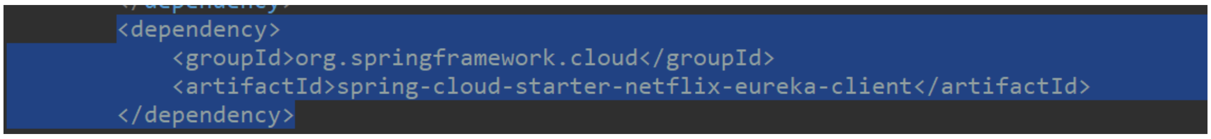
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

C] Make changes in the Account microservice to connect and register to Eureka server

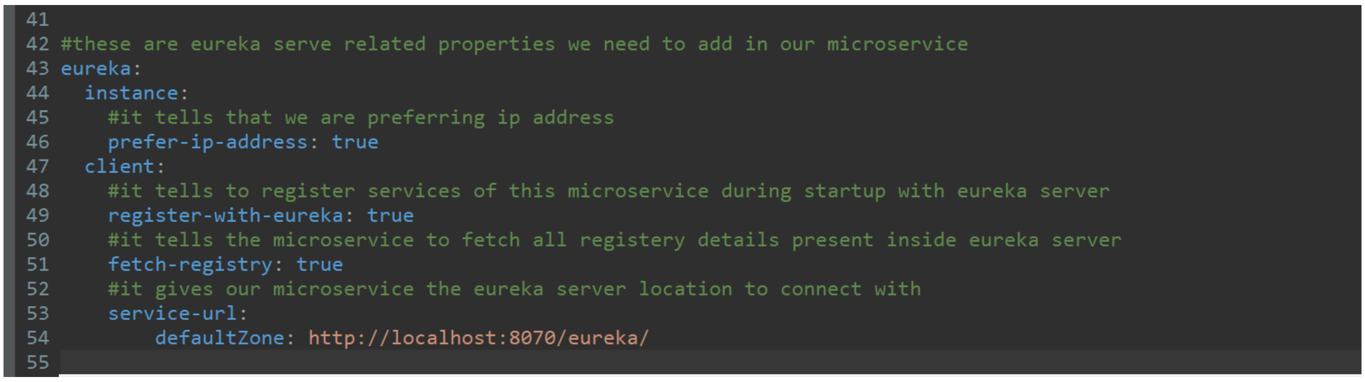
1] Add the below dependency into account pom.xml file:



```
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
```

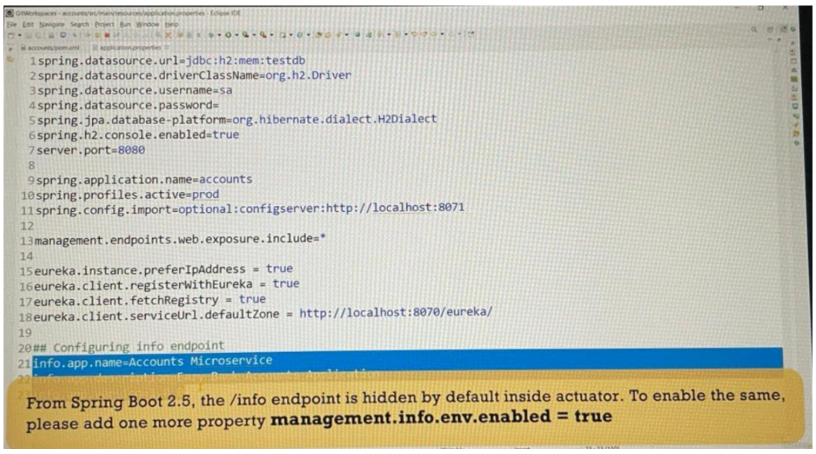
1.1] As we need to connect to eureka server so we need an client for connection

1.2] Also add eureka server related dependency as:



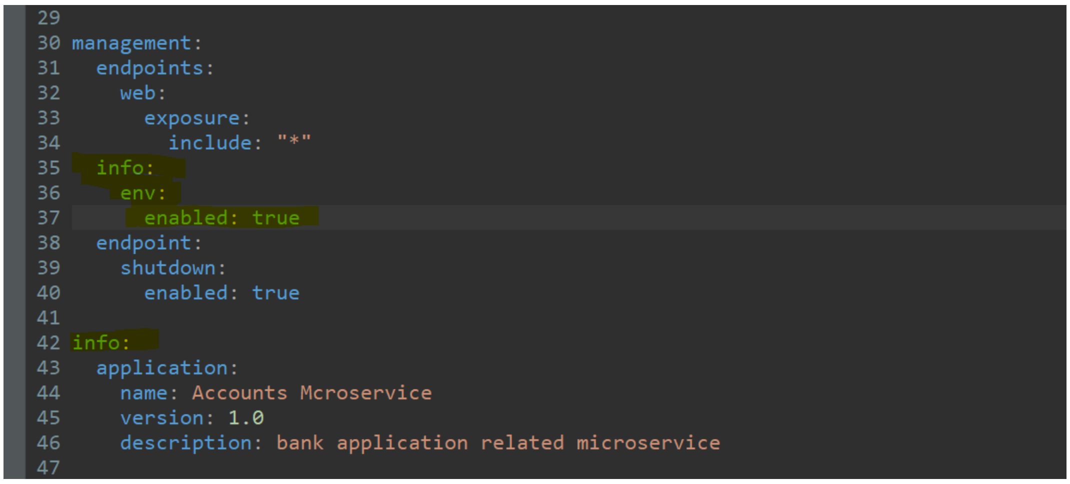
```

#these are eureka serve related properties we need to add in our microservice
eureka:
  instance:
    #it tells that we are preferring ip address
    prefer-ip-address: true
  client:
    #it tells to register services of this microservice during startup with eureka server
    register-with-eureka: true
    #it tells the microservice to fetch all registery details present inside eureka server
    fetch-registry: true
    #it gives our microservice the eureka server location to connect with
    service-url:
        defaultZone: http://localhost:8070/eureka/

```

1.3] Enable the info configuration inside the yml file as:



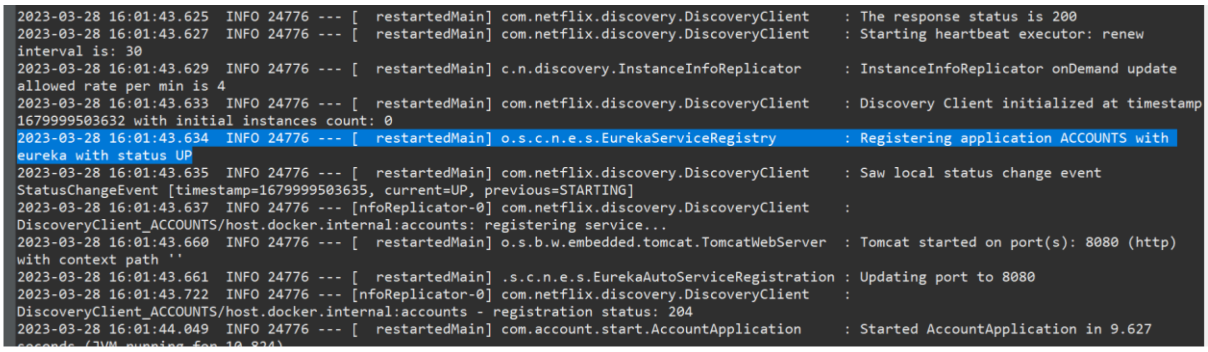
```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
server.port=8080

spring.application.name=accounts
spring.profiles.active=prod
spring.config.import=optional:configserver:http://localhost:8071

management.endpoints.web.exposure.include=*

eureka.instance.preferIpAddress = true
eureka.client.registerWithEureka = true
eureka.client.fetchRegistry = true
eureka.client.serviceUrl.defaultZone = http://localhost:8070/eureka/

## Configuring info endpoint
info.app.name=Accounts Microservice
```

From Spring Boot 2.5, the /info endpoint is hidden by default inside actuator. To enable the same, please add one more property **management.info.env.enabled = true**



```

management:
  endpoints:
    web:
      exposure:
        include: "*"
  info:
    env:
      enabled: true
  endpoint:
    shutdown:
      enabled: true

info:
  application:
    name: Accounts Mcroservice
    version: 1.0
    description: bank application related microservice

```

2] Start the account microservice along with eureka server and spring cloud config server too:

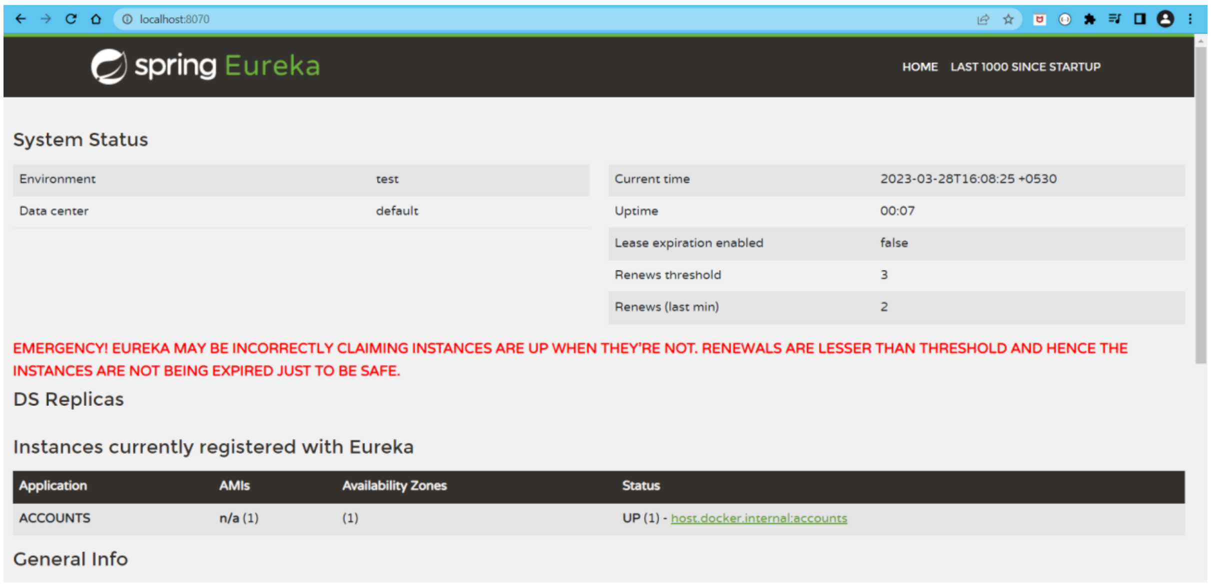


```
2023-03-28 16:01:43.625  INFO 24776 --- [  restartedMain] com.netflix.discovery.DiscoveryClient    : The response status is 200
2023-03-28 16:01:43.627  INFO 24776 --- [  restartedMain] com.netflix.discovery.DiscoveryClient    : Starting heartbeat executor: renew
interval is: 30
2023-03-28 16:01:43.629  INFO 24776 --- [  restartedMain] c.n.discovery.InstanceInfoReplicator     : InstanceInfoReplicator onDemand update
allowed rate per min is 4
2023-03-28 16:01:43.633  INFO 24776 --- [  restartedMain] com.netflix.discovery.DiscoveryClient    : Discovery Client initialized at timestamp
1679999503632 with initial instances count: 0
2023-03-28 16:01:43.634  INFO 24776 --- [  restartedMain] o.s.c.n.e.s.EurekaServiceRegistry        : Registering application ACCOUNTS with
eureka with status UP
2023-03-28 16:01:43.635  INFO 24776 --- [  restartedMain] com.netflix.discovery.DiscoveryClient    : Saw local status change event
StatusChangeEvent [timestamp=1679999503635, current=UP, previous=STARTING]
2023-03-28 16:01:43.637  INFO 24776 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    :
DiscoveryClient_ACCOUNTS/host.docker.internal:accounts: registering service...
2023-03-28 16:01:43.660  INFO 24776 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http)
with context path ''
2023-03-28 16:01:43.661  INFO 24776 --- [  restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8080
2023-03-28 16:01:43.722  INFO 24776 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    :
DiscoveryClient_ACCOUNTS/host.docker.internal:accounts - registration status: 204
2023-03-28 16:01:44.049  INFO 24776 --- [  restartedMain] com.account.start.AccountApplication     : Started AccountApplication in 9.627
```

As you an see our account microservice got registered with eureka server

3] Now if I visit my eureka server dashboard I will my account service got registered with it as below:

http://localhost:8070/



localhost:8070

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2023-03-28T16:08:25 +0530
Uptime	00:07
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	2

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

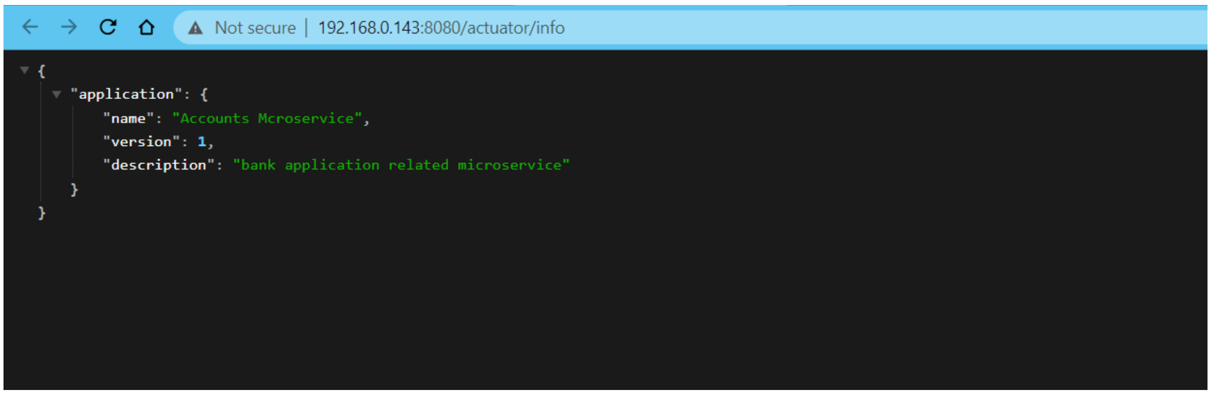
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTS	n/a (1)	(1)	UP (1) - host.docker.internal:accounts

General Info

If you click on the status url 'host.docker.internal:accounts' you will get the info about project



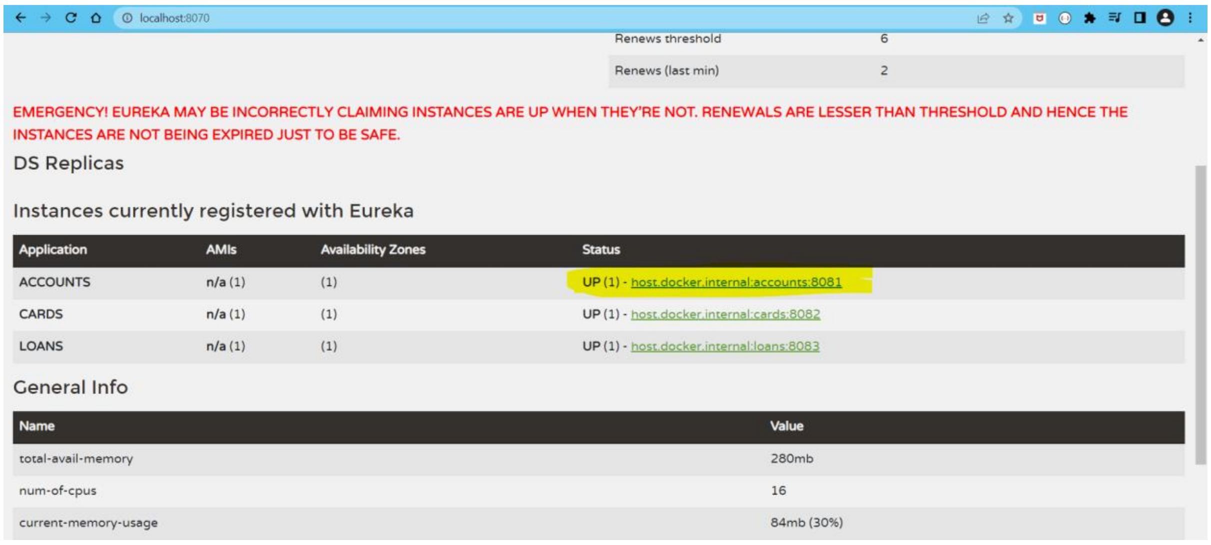
Not secure | 192.168.0.143:8080/actuator/info

```
{
    "application": {
        "name": "Accounts Mcroservice",
        "version": 1,
        "description": "bank application related microservice"
    }
}
```

Here, in eureka my apps url is 'host.docker.internal:accounts' instead of 'localhost' because docker is installed in my system and it treats docker internal url as my host url, but if docker was not installed in my system then I would have got '**localhost**' instead of '**host.docker.internal**'

D] Safely De-register microservice from the Eureka server:

1] If you go to your eureka server registered account microservice and hit the url



localhost:8070

Renews threshold	6
Renews (last min)	2

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

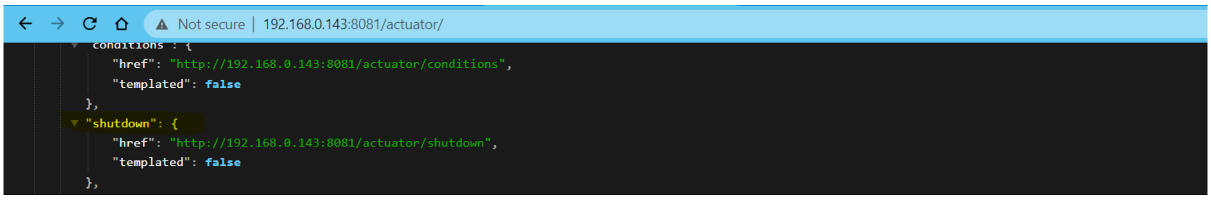
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTS	n/a (1)	(1)	UP (1) - host.docker.internal:accounts:8081
CARDS	n/a (1)	(1)	UP (1) - host.docker.internal:cards:8082
LOANS	n/a (1)	(1)	UP (1) - host.docker.internal:loans:8083

General Info

Name	Value
total-avail-memory	280mb
num-of-cpus	16
current-memory-usage	84mb (30%)

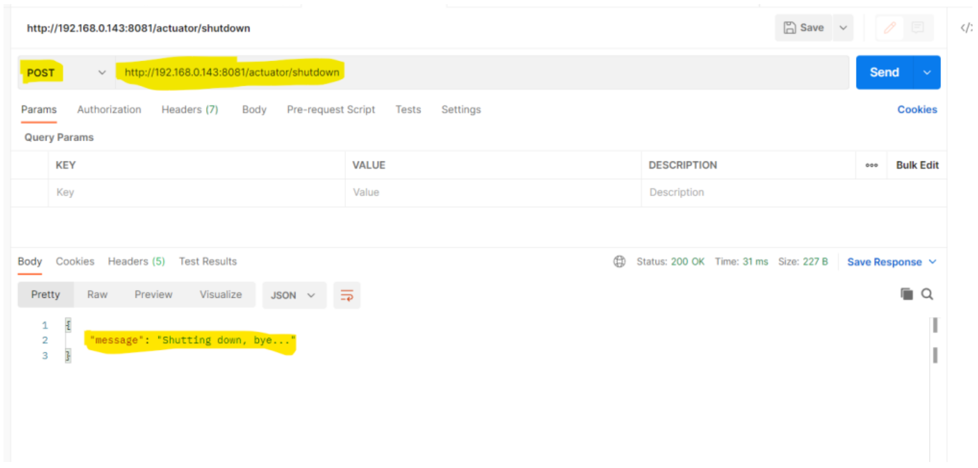
And then call all actuator url as below you will see a shutdown url present inside actuator



Not secure | 192.168.0.143:8081/actuator/

```
  "conditions": {
    "href": "http://192.168.0.143:8081/actuator/conditions",
    "templated": false
  },
  "shutdown": {
    "href": "http://192.168.0.143:8081/actuator/shutdown",
    "templated": false
  },
```

As this shutdown URL is of POST method so directly calling that URL using get method on browser will give us error we need tool like POSTMAN to call that URL using POST method as below:



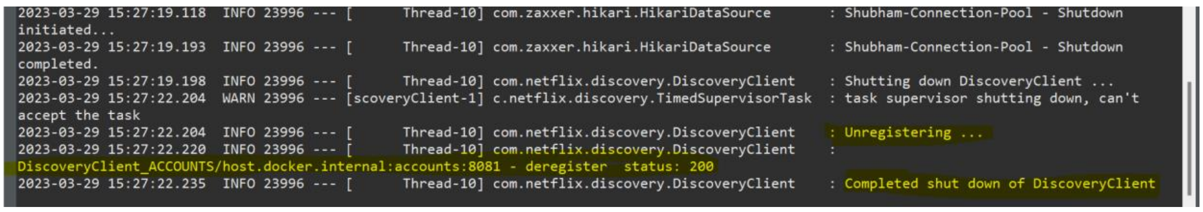
http://192.168.0.143:8081/actuator/shutdown

POST http://192.168.0.143:8081/actuator/shutdown

Status: 200 OK Time: 31 ms Size: 227 B

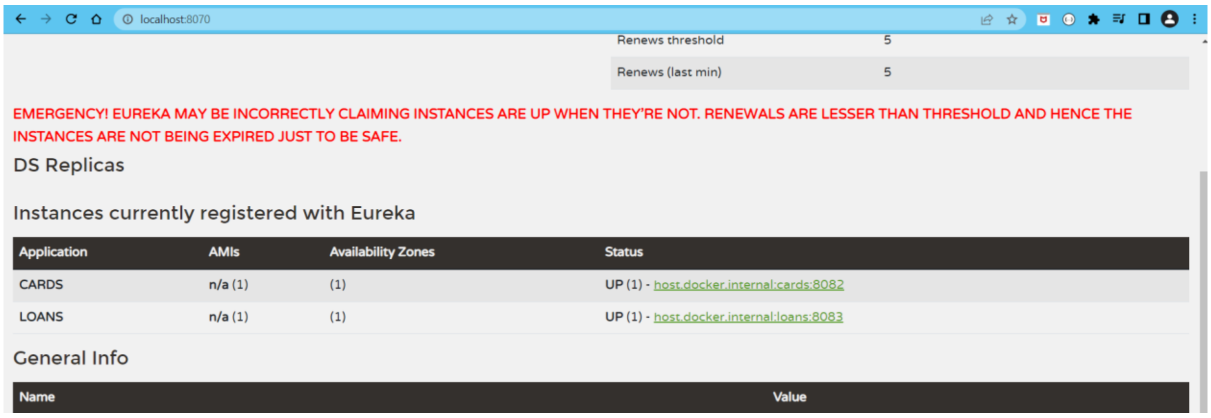
```
{
  "message": "Shutting down, bye..."
}
```

Here the account microservice got successfully de-registered from the eureka server as below:



```
2023-03-29 15:27:19.118  INFO 23996 --- [      Thread-10] com.zaxxer.hikari.HikariDataSource       : Shubham-Connection-Pool - Shutdown
initiated...
2023-03-29 15:27:19.193  INFO 23996 --- [      Thread-10] com.zaxxer.hikari.HikariDataSource       : Shubham-Connection-Pool - Shutdown
completed.
2023-03-29 15:27:19.198  INFO 23996 --- [      Thread-10] com.netflix.discovery.DiscoveryClient    : Shutting down DiscoveryClient ...
2023-03-29 15:27:22.204  WARN 23996 --- [scoveryClient-1] c.netflix.discovery.TimedSupervisorTask  : task supervisor shutting down, can't
accept the task
2023-03-29 15:27:22.204  INFO 23996 --- [      Thread-10] com.netflix.discovery.DiscoveryClient    : Unregistering ...
2023-03-29 15:27:22.220  INFO 23996 --- [      Thread-10] com.netflix.discovery.DiscoveryClient    :
DiscoveryClient_ACCOUNTS/host.docker.internal:accounts:8081 - deregister  status: 200
2023-03-29 15:27:22.235  INFO 23996 --- [      Thread-10] com.netflix.discovery.DiscoveryClient    : Completed shut down of DiscoveryClient
```

This response is taken from Account microservice consoled



localhost:8070

Renews threshold	5
Renews (last min)	5

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CARDS	n/a (1)	(1)	UP (1) - host.docker.internal:cards:8082
LOANS	n/a (1)	(1)	UP (1) - host.docker.internal:loans:8083

General Info

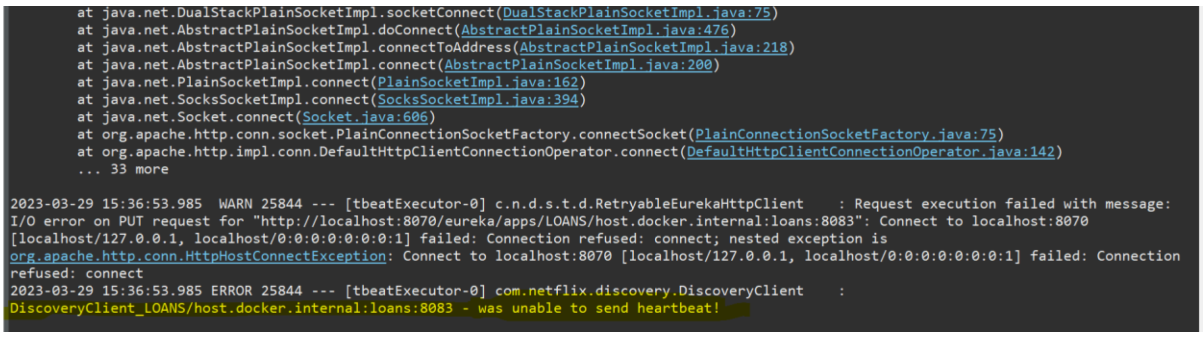
Name	Value

As seen in Eureka server this Account microservice is successfully deregistered

E] How to see the Heartbeats from microservice that wants to register itself with Eureka server:

1] Here, heart beats indicates the health of a particular microservice that Eureka server monitor to check their health at an interval of 30seconds, and suppose in interval of 90 seconds if some microservice doesn't give any response then Eureka server de-register that microservice from itself

2] If I close the eureka service, then I can see the heartbeats from microservice trying to connect with Eureka server after 30seconds interval as below:

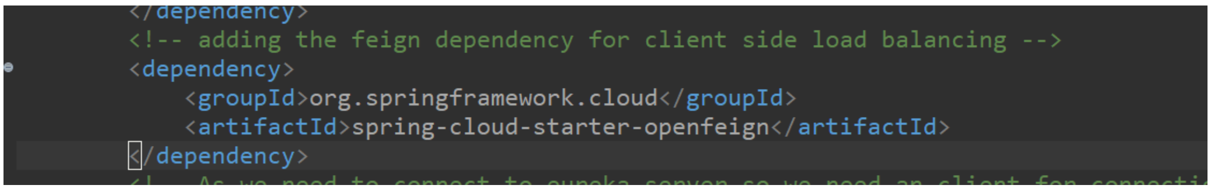


```
        at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:75)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:476)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:218)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:200)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:162)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:394)
        at java.net.Socket.connect(Socket.java:606)
        at org.apache.http.conn.socket.PlainConnectionSocketFactory.connectSocket(PlainConnectionSocketFactory.java:75)
        at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.java:142)
        ... 33 more

2023-03-29 15:36:53.985  WARN 25844 --- [tbeatExecutor-0] c.n.d.s.t.d.RetryableEurekaHttpClient    : Request execution failed with message:
I/O error on PUT request for "http://localhost:8070/eureka/apps/LOANS/host.docker.internal:loans:8083": Connect to localhost:8070
[localhost/127.0.0.1, localhost/0:0:0:0:0:0:0:1] failed: Connection refused: connect; nested exception is
org.apache.http.conn.HttpHostConnectException: Connect to localhost:8070 [localhost/127.0.0.1, localhost/0:0:0:0:0:0:0:1] failed: Connection
refused: connect
2023-03-29 15:36:53.985 ERROR 25844 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient    :
DiscoveryClient_LOANS/host.docker.internal:loans:8083 - was unable to send heartbeat!
```

F] Using Feign Client to invoke service from other microservices:

1] Here we are going to do client side load balancing to avoid load on eureka server

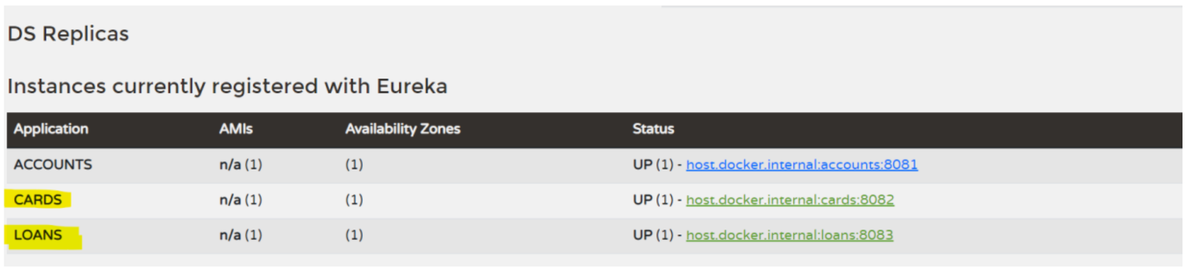
2] Add feign dependency into account microservice pom.xml file as below:



```xml
        </dependency>
        <!-- adding the feign dependency for client side load balancing -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>
        <!-- As we need to connect to eureka server so we need an client for connecti
```

3] Here, I'am not adding this dependency in to loan and card microservice as I want to invoke all other service from Account microservice only i.e. the loan and card service registered inside eureka server should be accessed by account microservice

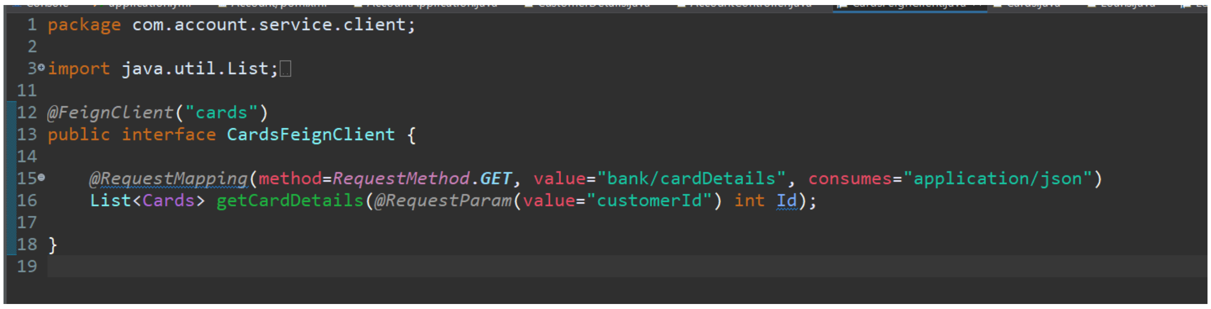
4] Next create two interface related to service access of card and loan app registered in eureka server as below:



DS Replicas

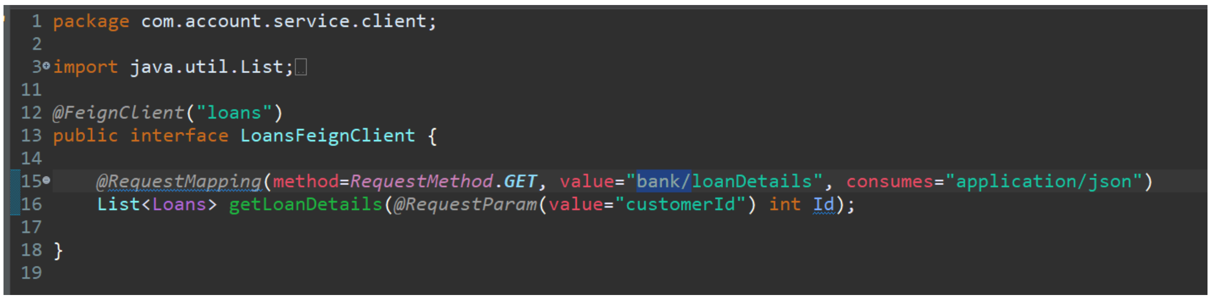
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTS	n/a (1)	(1)	UP (1) - host.docker.internal:accounts:8081
CARDS	n/a (1)	(1)	UP (1) - host.docker.internal:cards:8082
LOANS	n/a (1)	(1)	UP (1) - host.docker.internal:loans:8083



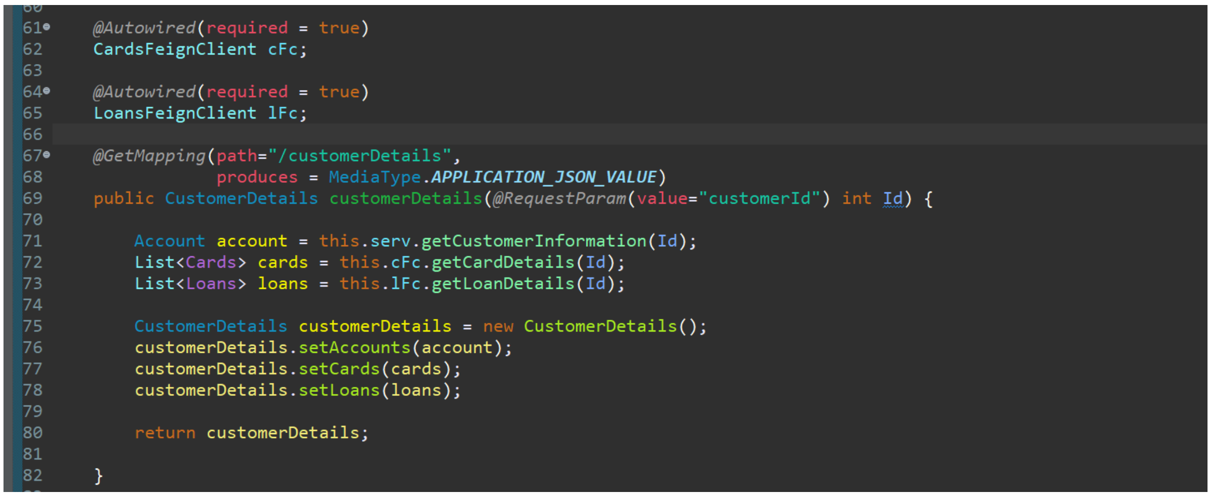
```java
package com.account.service.client;

import java.util.List;

@FeignClient("cards")
public interface CardsFeignClient {

    @RequestMapping(method=RequestMethod.GET, value="bank/cardDetails", consumes="application/json")
    List<Cards> getCardDetails(@RequestParam(value="customerId") int Id);

}
```

Here inside **@FeignClient("cards")** here, cards is the application name present inside eureka server, and inside **@RequestingMapping** value parameter i.e. **value="bank/cardDetails"** this is the controller URL of card microservice from which we will fetch cards details



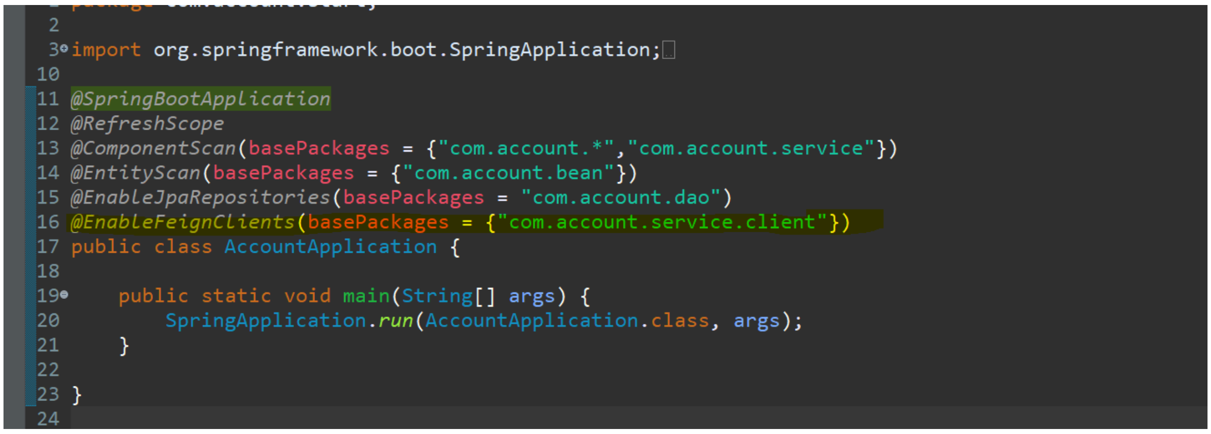
```java
package com.account.service.client;

import java.util.List;

@FeignClient("loans")
public interface LoansFeignClient {

    @RequestMapping(method=RequestMethod.GET, value="bank/loanDetails", consumes="application/json")
    List<Loans> getLoanDetails(@RequestParam(value="customerId") int Id);

}
```

5] Add a new Get method inside Accounts controller



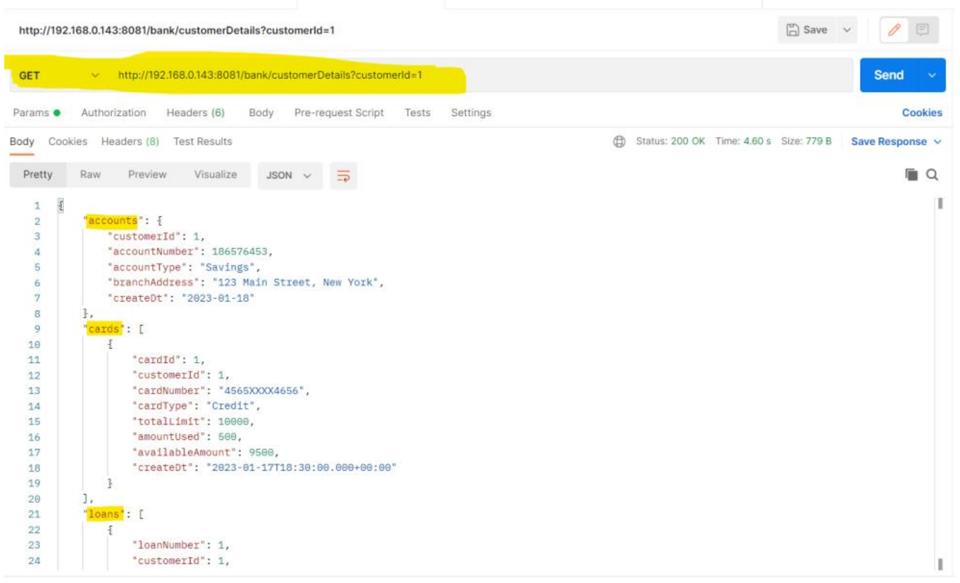
```java
    @Autowired(required = true)
    CardsFeignClient cFc;

    @Autowired(required = true)
    LoansFeignClient lFc;

    @GetMapping(path="/customerDetails",
                produces = MediaType.APPLICATION_JSON_VALUE)
    public CustomerDetails customerDetails(@RequestParam(value="customerId") int Id) {

        Account account = this.serv.getCustomerInformation(Id);
        List<Cards> cards = this.cFc.getCardDetails(Id);
        List<Loans> loans = this.lFc.getLoanDetails(Id);

        CustomerDetails customerDetails = new CustomerDetails();
        customerDetails.setAccounts(account);
        customerDetails.setCards(cards);
        customerDetails.setLoans(loans);

        return customerDetails;

    }
```

6] Enable the feign client using **@EnableFeignClient** annotation to use this feign client feature



```java

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@RefreshScope
@ComponentScan(basePackages = {"com.account.*","com.account.service"})
@EntityScan(basePackages = {"com.account.bean"})
@EnableJpaRepositories(basePackages = "com.account.dao")
@EnableFeignClients(basePackages = {"com.account.service.client"})
public class AccountApplication {

    public static void main(String[] args) {
        SpringApplication.run(AccountApplication.class, args);
    }

}
```

7] Now run the spring cloud server, eureka server, loan, card and account application to see the result as below:



http://192.168.0.143:8081/bank/customerDetails?customerId=1

GET http://192.168.0.143:8081/bank/customerDetails?customerId=1

Status: 200 OK Time: 4.60 s Size: 779 B

```json
{
    "accounts": {
        "customerId": 1,
        "accountNumber": 186576453,
        "accountType": "Savings",
        "branchAddress": "123 Main Street, New York",
        "createDt": "2023-01-18"
    },
    "cards": [
        {
            "cardId": 1,
            "customerId": 1,
            "cardNumber": "4565XXXX4656",
            "cardType": "Credit",
            "totalLimit": 10000,
            "amountUsed": 500,
            "availableAmount": 9500,
            "createDt": "2023-01-17T18:30:00.000+00:00"
        }
    ],
    "loans": [
        {
            "loanNumber": 1,
            "customerId": 1,
```

Now as you can see when I try to access the details of an customer from account microservice I will also get his details related to cards and loans from account microservice only

8] This is how we use **Feign Client** to invoke other microservice from one microservice