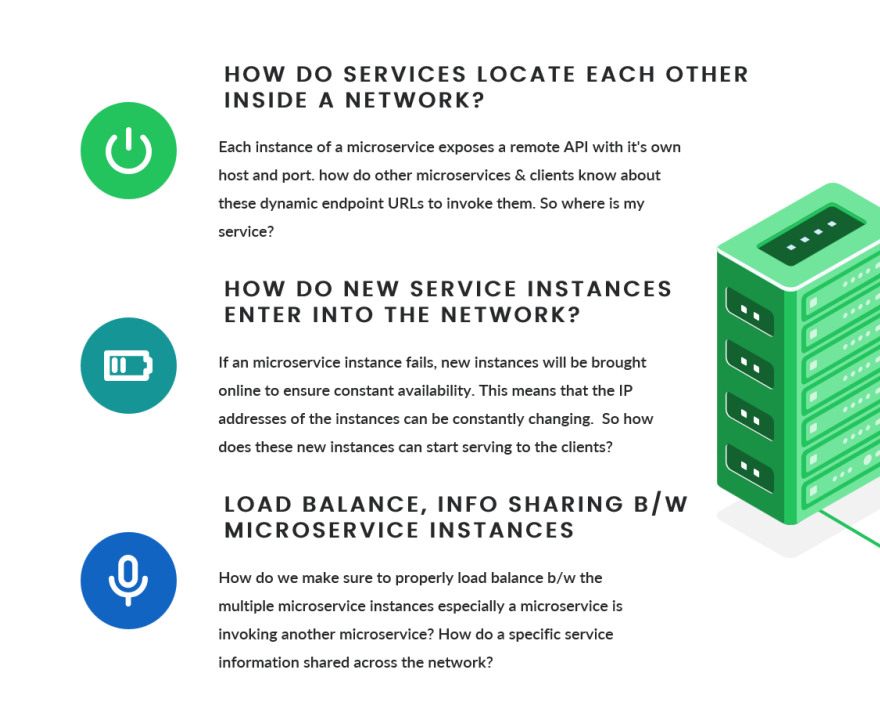
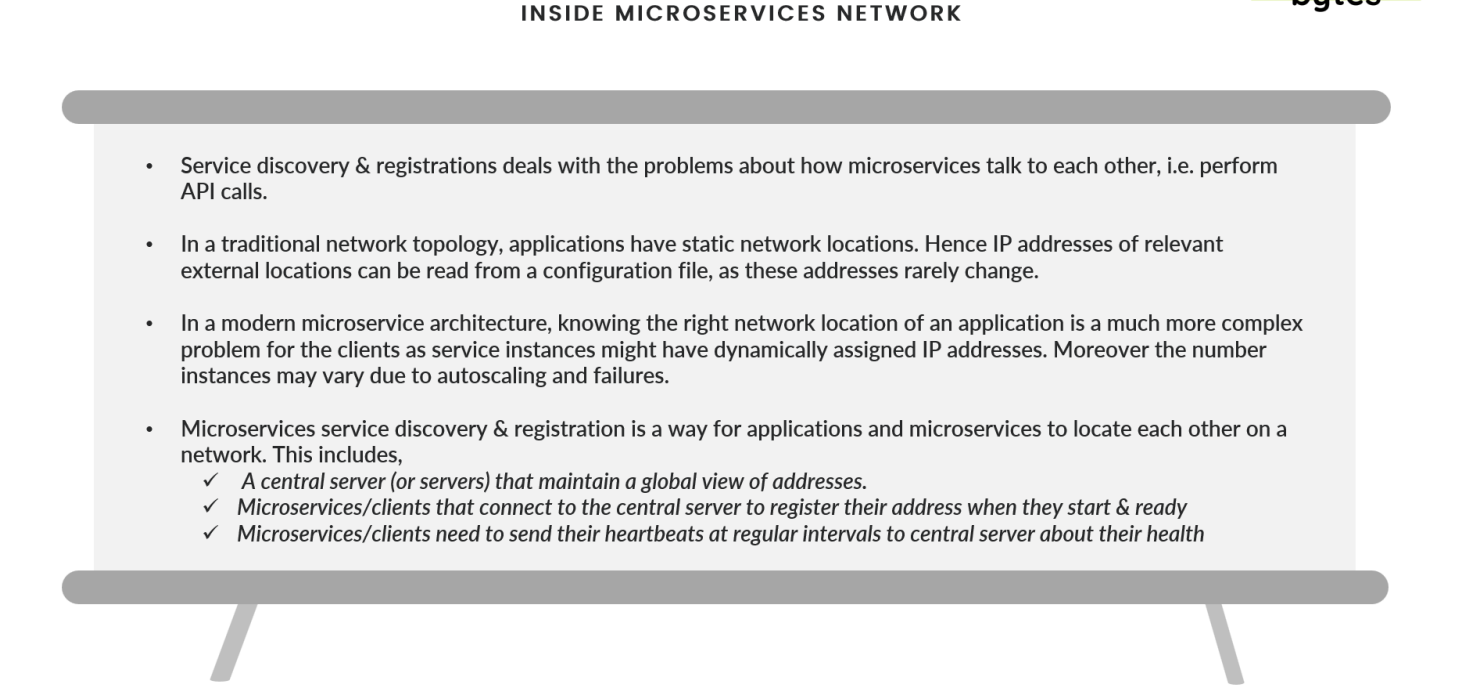
**Service Discovery and Registration**

**-Shubham Ashtaputre**

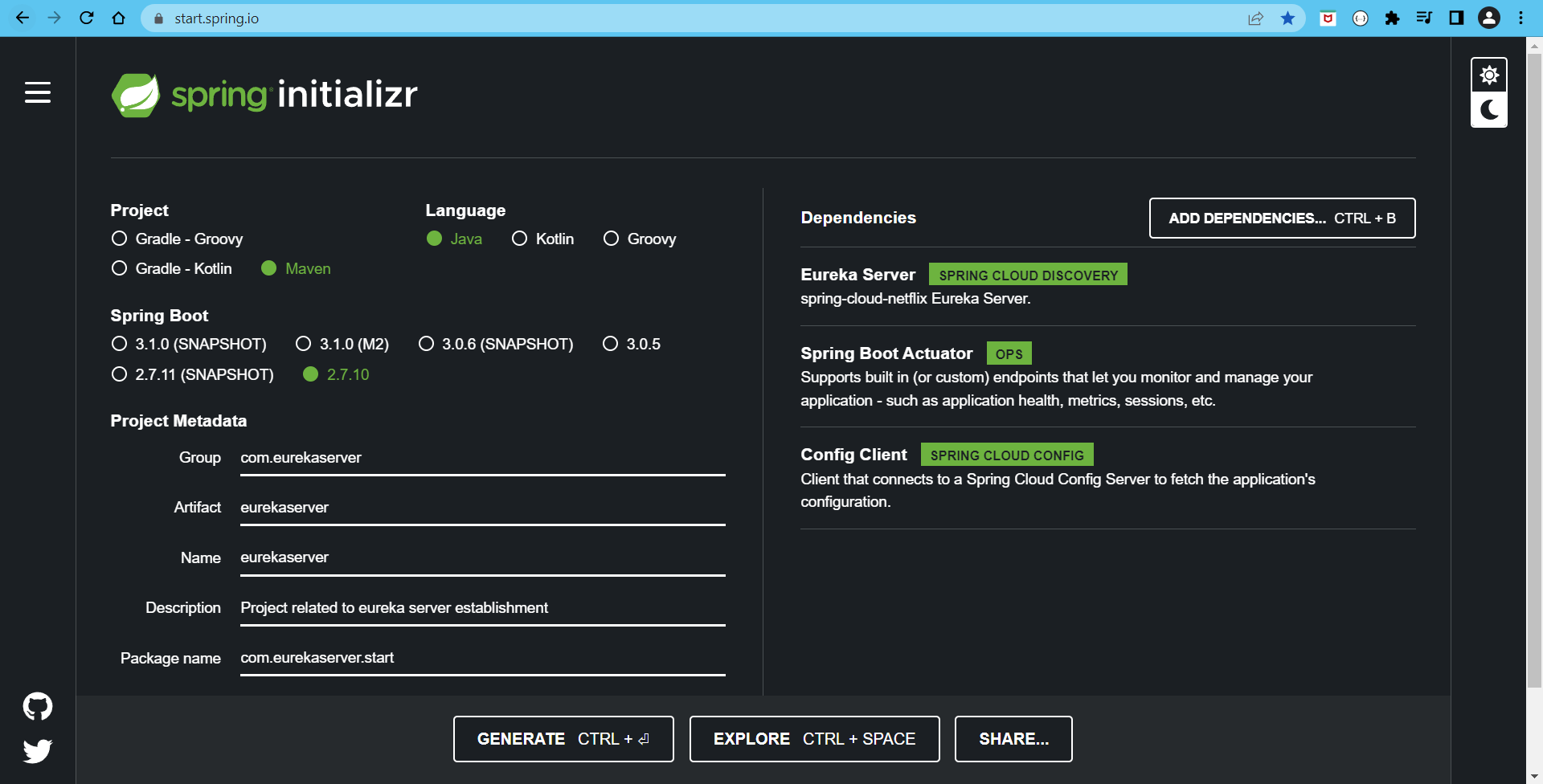
**A] Necessity of Service Discovery and Registration**

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

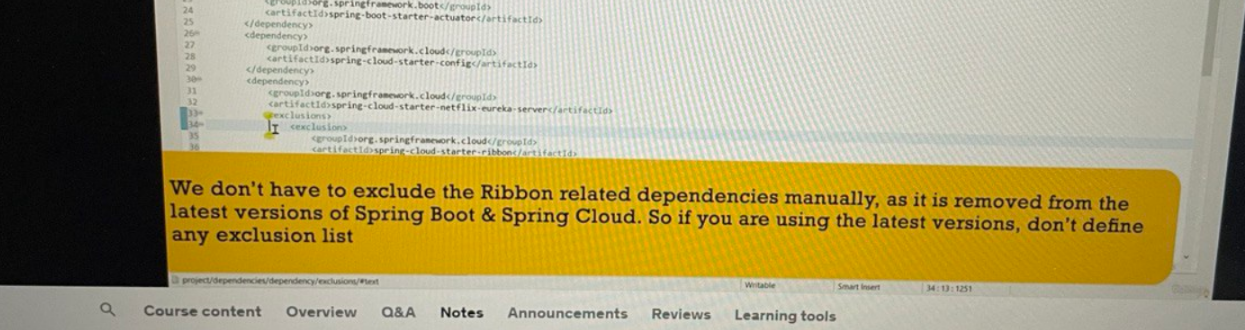


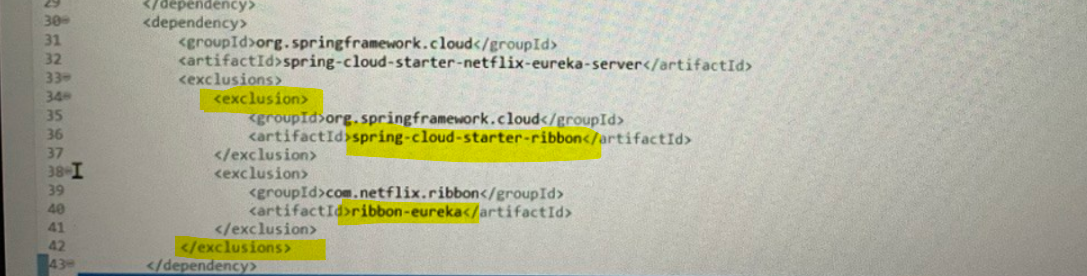


**B] Set Service discovery agent using Eureka server**



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

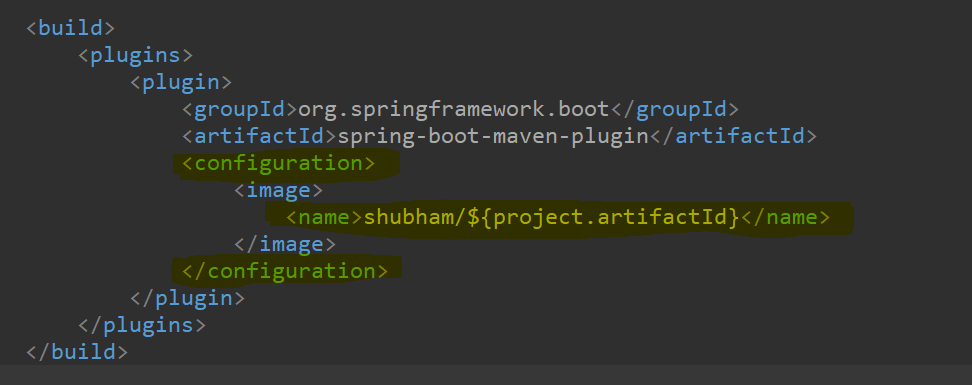




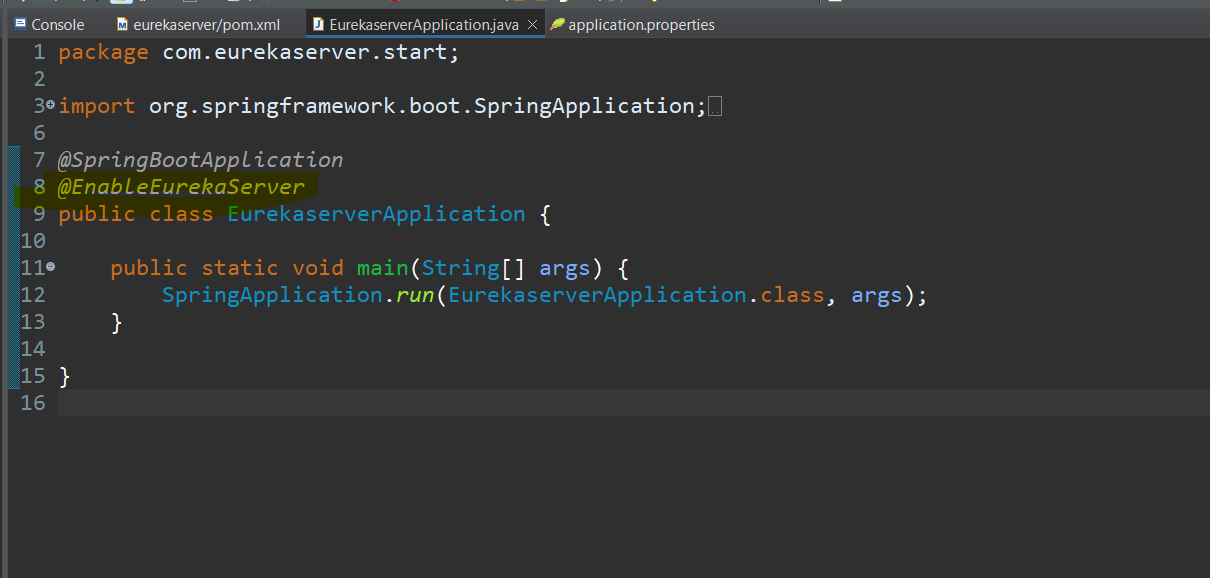
2] But if you are using latest spring-boot version then it is already taken care of

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

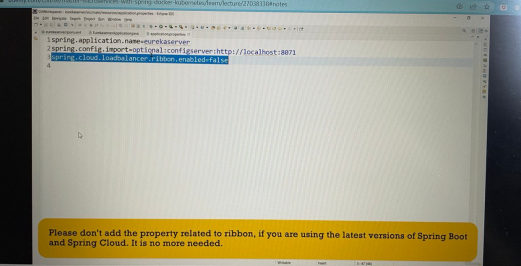
4] This will set how my image name of project



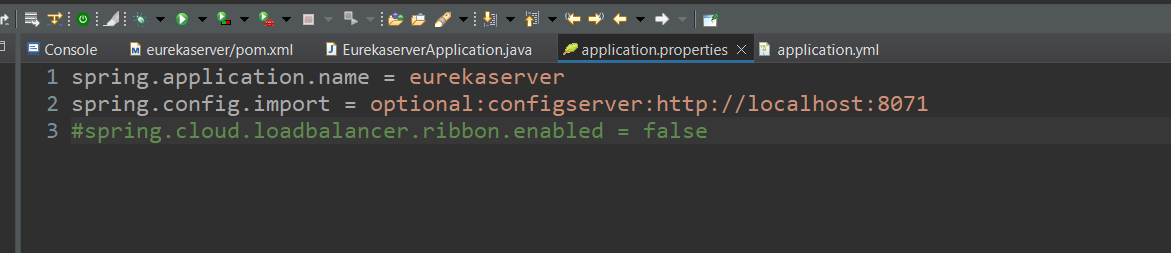
5] Using **‘@EnableEurekaServer**’ annotation we will make our microservice act as service discovery agent using eureka server



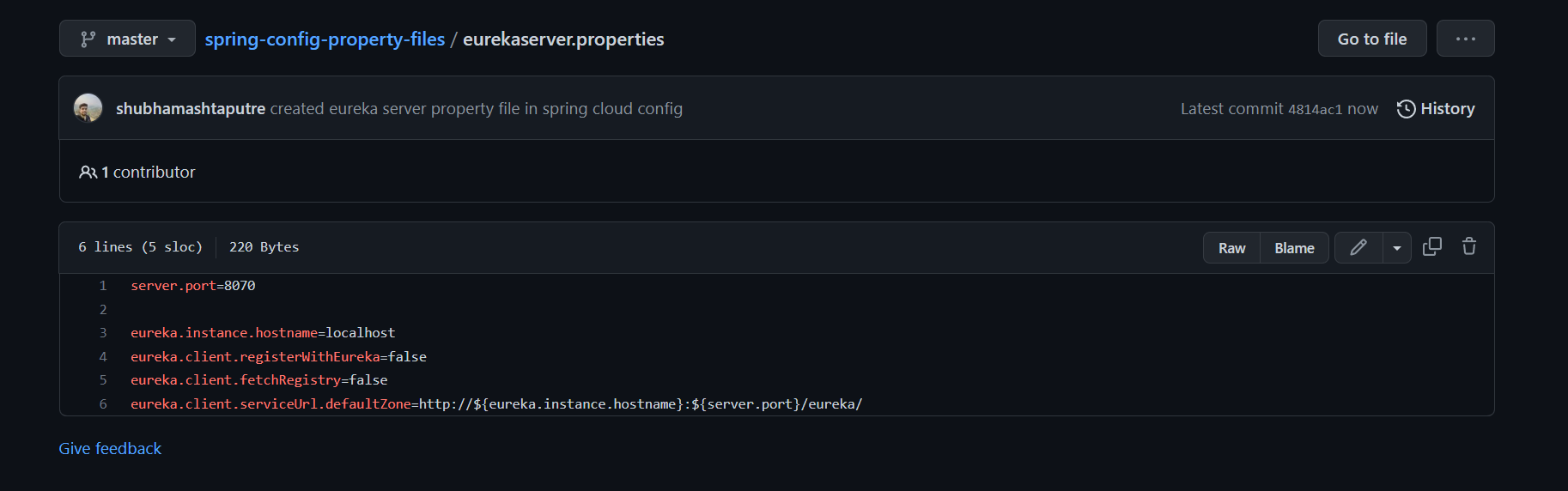
6] Make changes in the properties file:



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



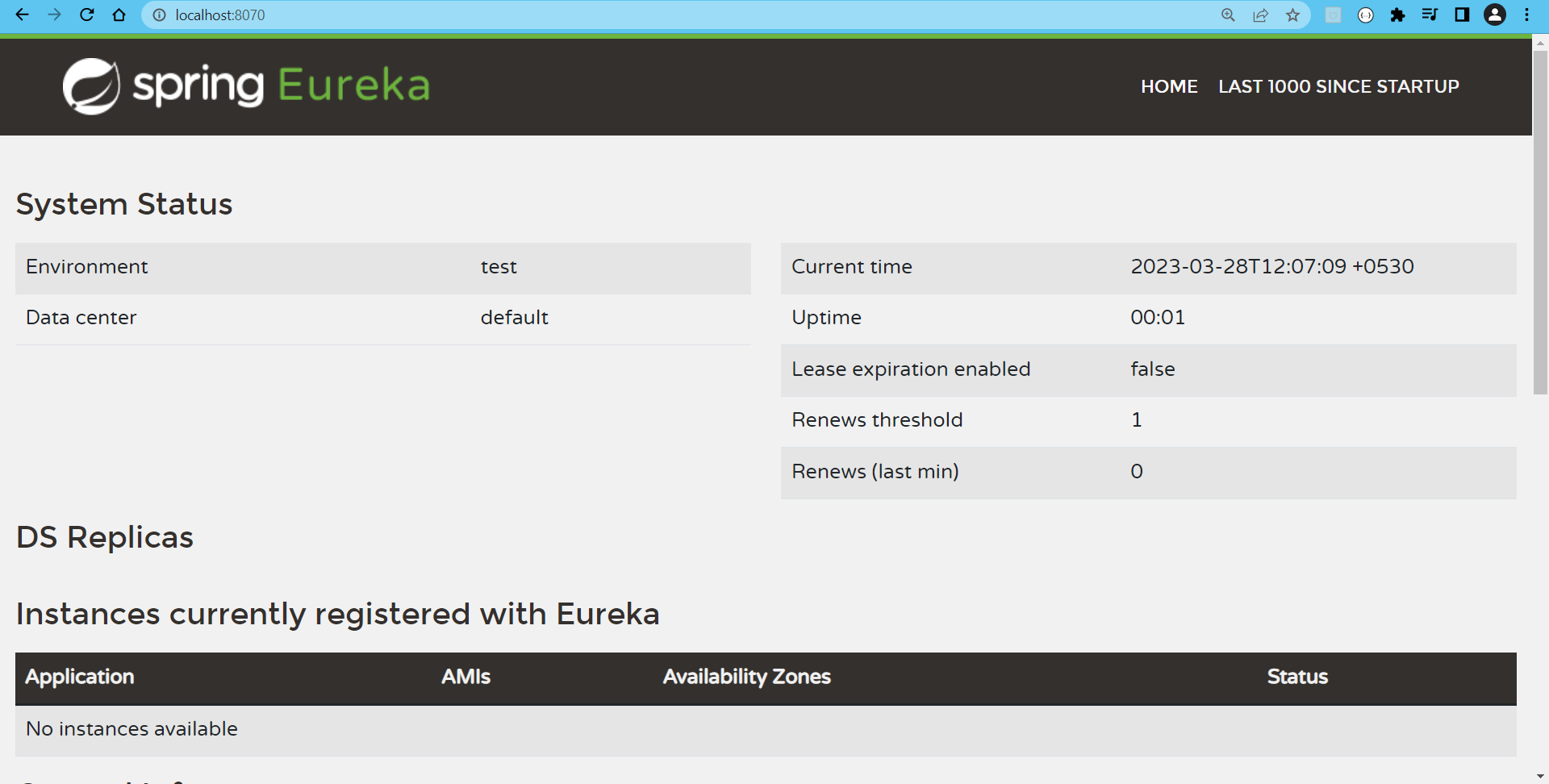
8] Create eurekaserver.properties inside github location to read from spring cloud config



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:

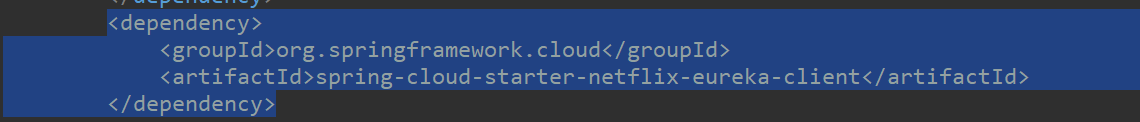


10] Check the eureka server localhost you eureka server is up and running and our service discovery agent is setup



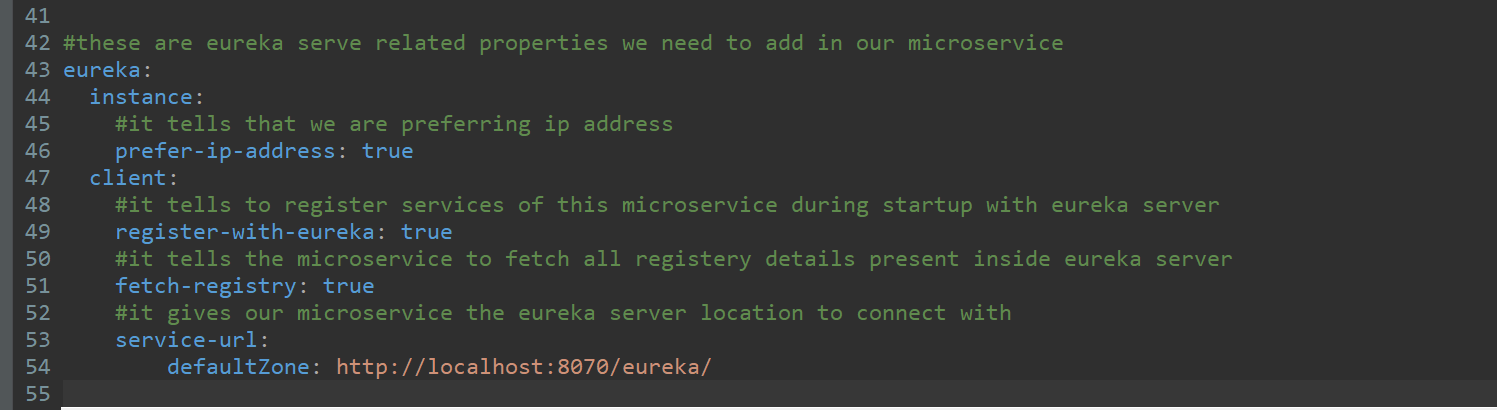
**C] Make changes in the Account microservice to connect and register to Eureka server**

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

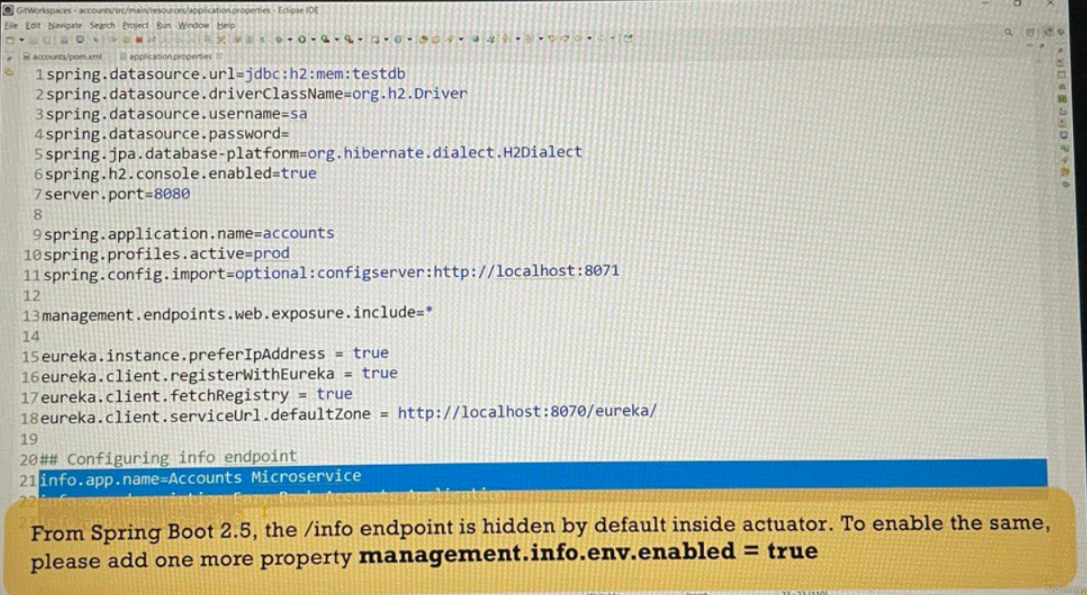


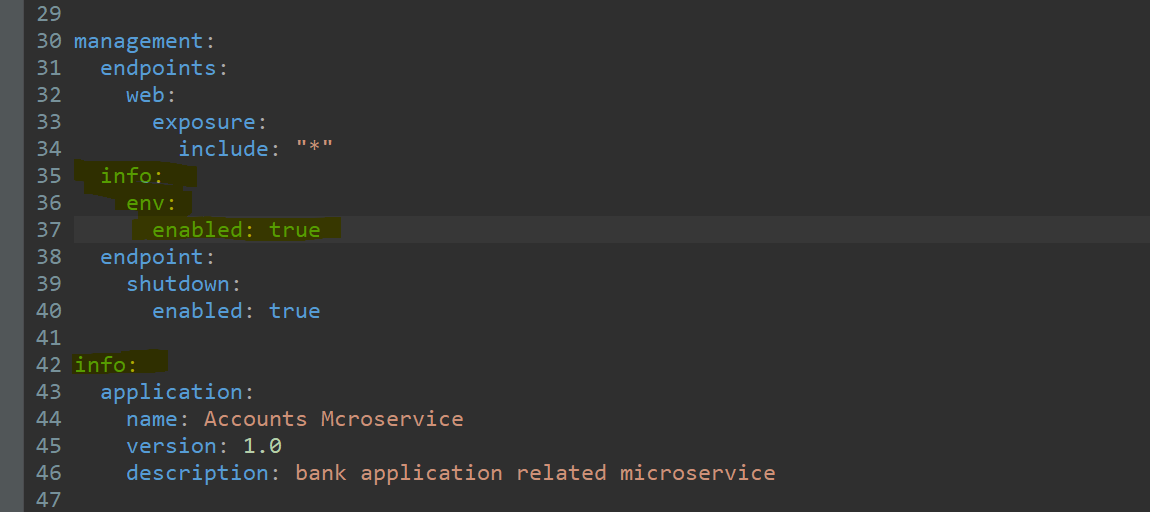
1.1] As we need to connect to eureka server so we need an client for connection

1.2] Also add eureka server related dependency as:

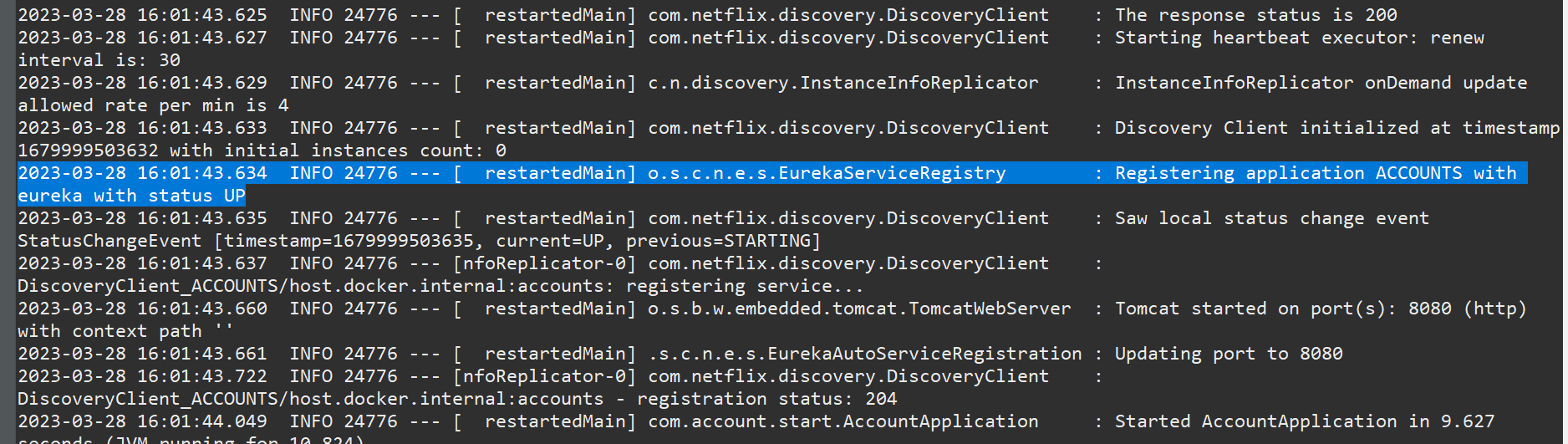


1.3] Enable the info configuration inside the yml file as:





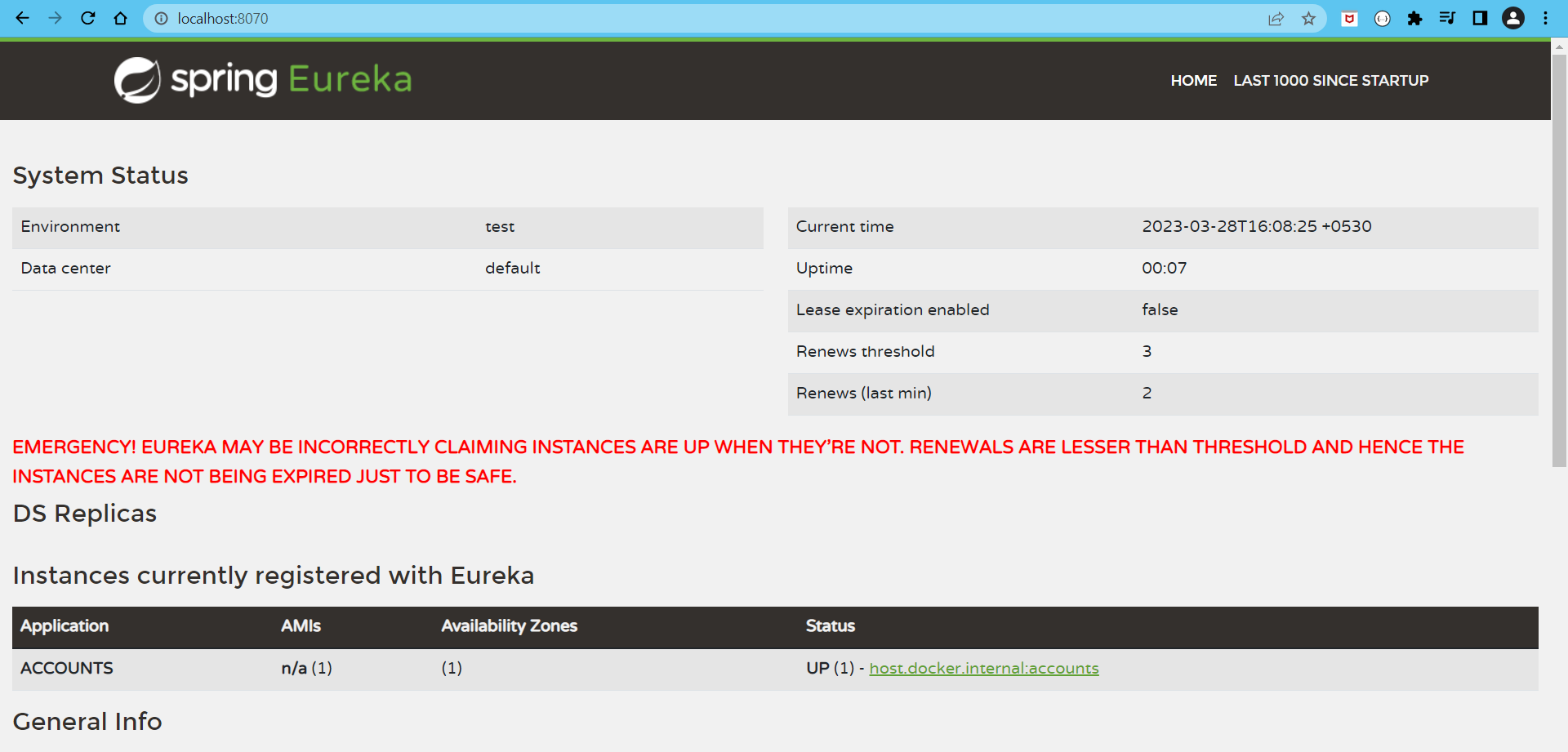
2] Start the account microservice along with eureka server and spring cloud config server too:



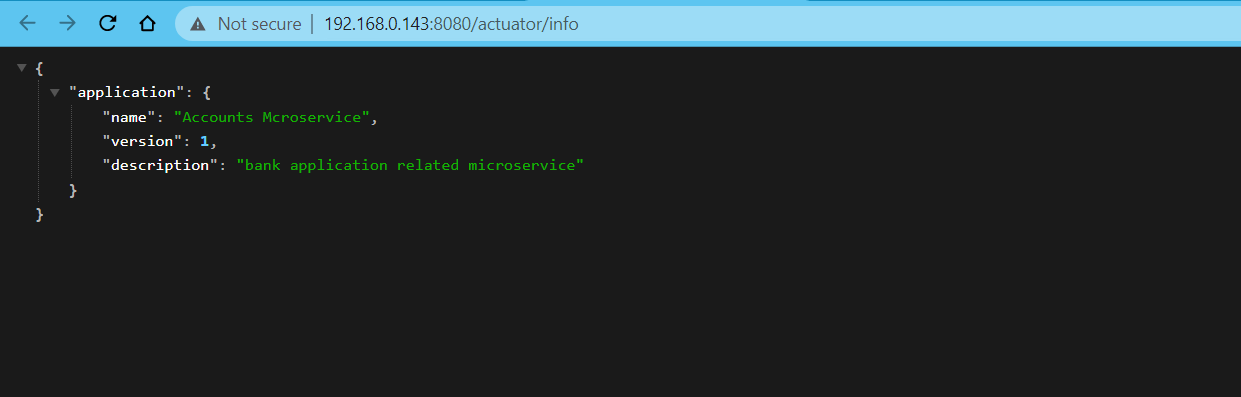
As you an see our account microservice got registered with eureka server

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

<http://localhost:8070/>



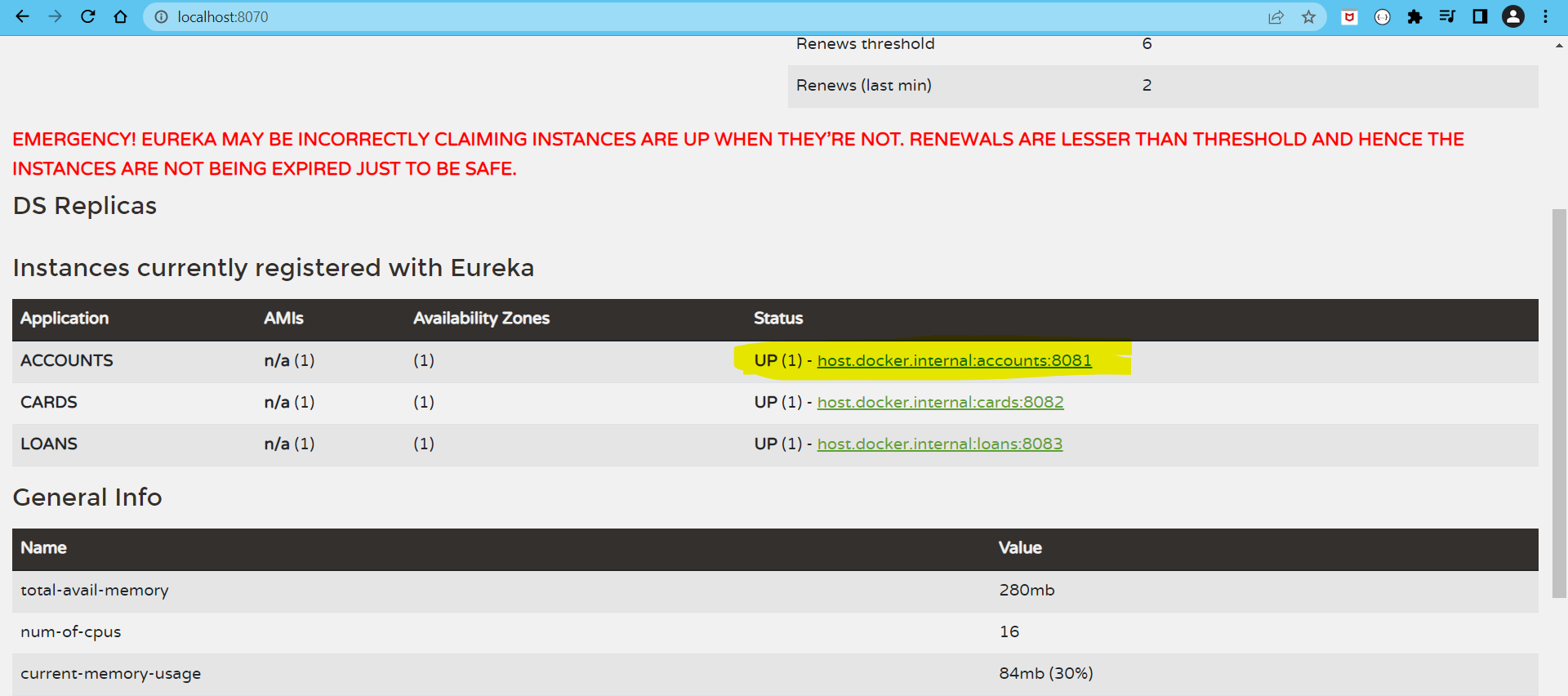
If you click on the status url ‘[host.docker.internal:accounts](http://192.168.0.143:8080/actuator/info" \t "_blank)’ you will get the info about project



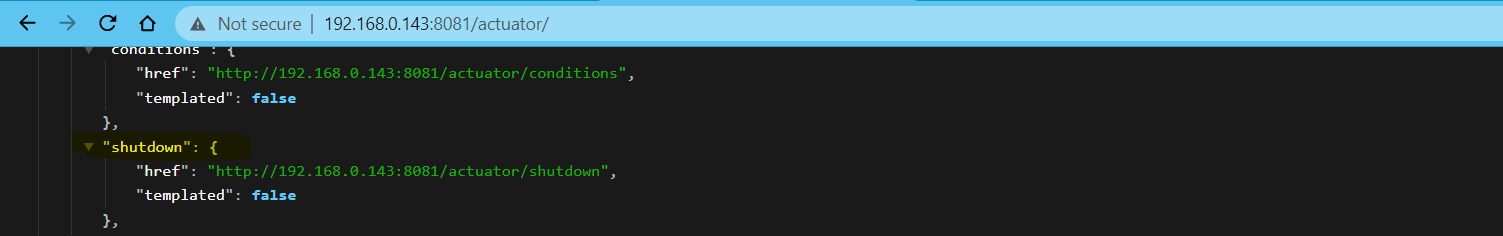
Here, in eureka my apps url is ‘[host.docker.internal:accounts](http://192.168.0.143:8080/actuator/info" \t "_blank)’ instead of ‘localhost’ because docker is installed in my system and it treats docker internal url as my host url, but if docker was not installed in my system then I would have got ‘**localhost**’ instead of ‘**host.docker.internal**’

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

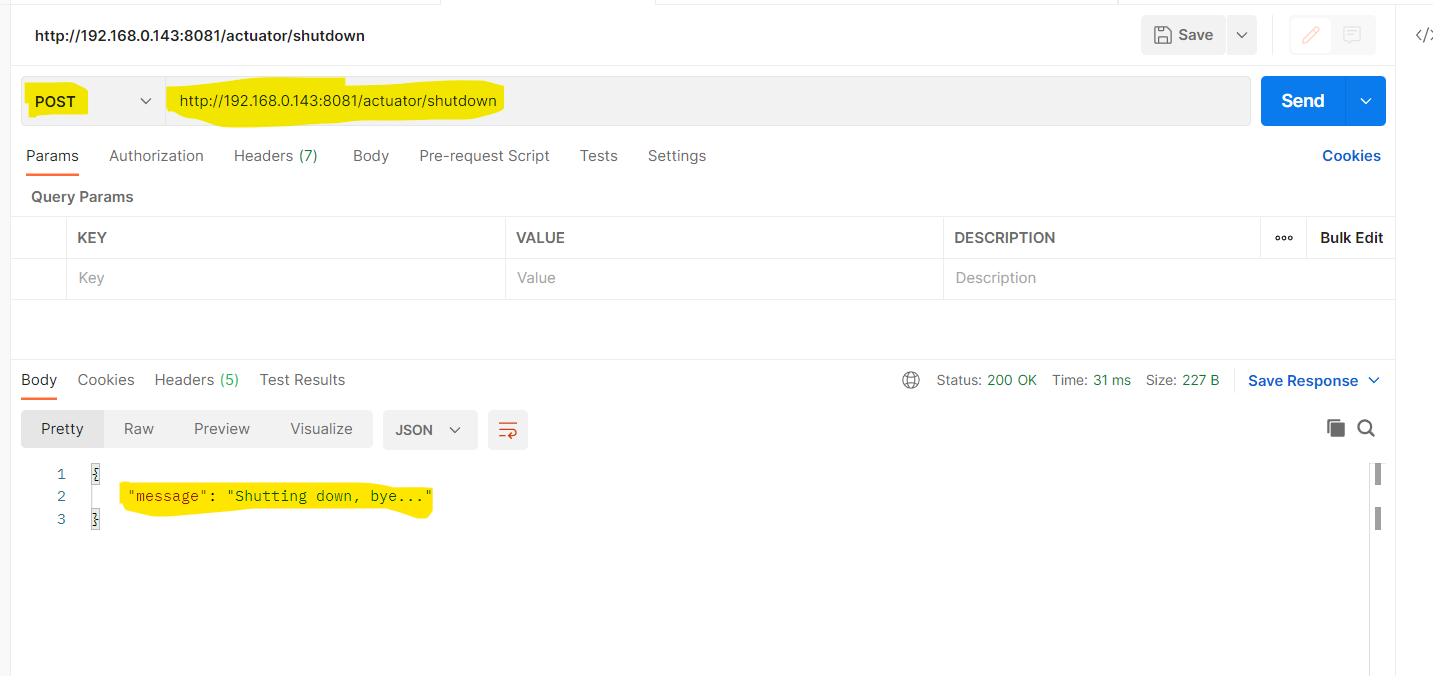
1] If you go to your eureka server registered account microservice and hit the url



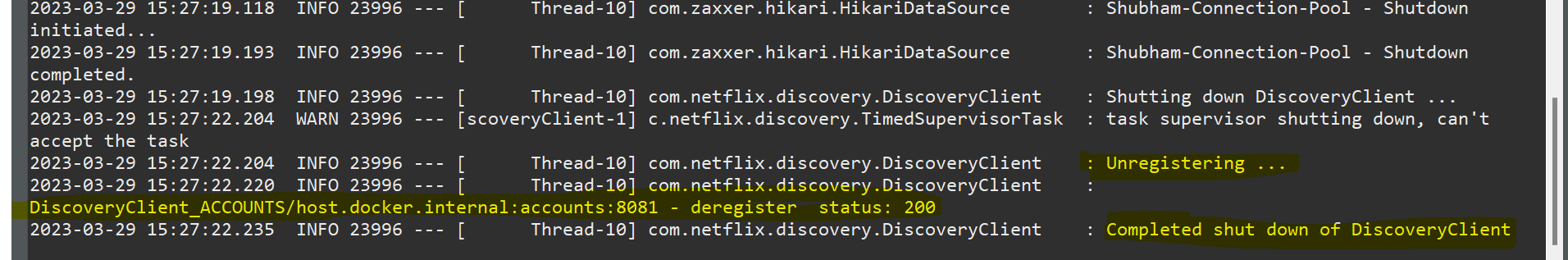
And then call all actuator url as below you will see a shutdown url present inside actuator



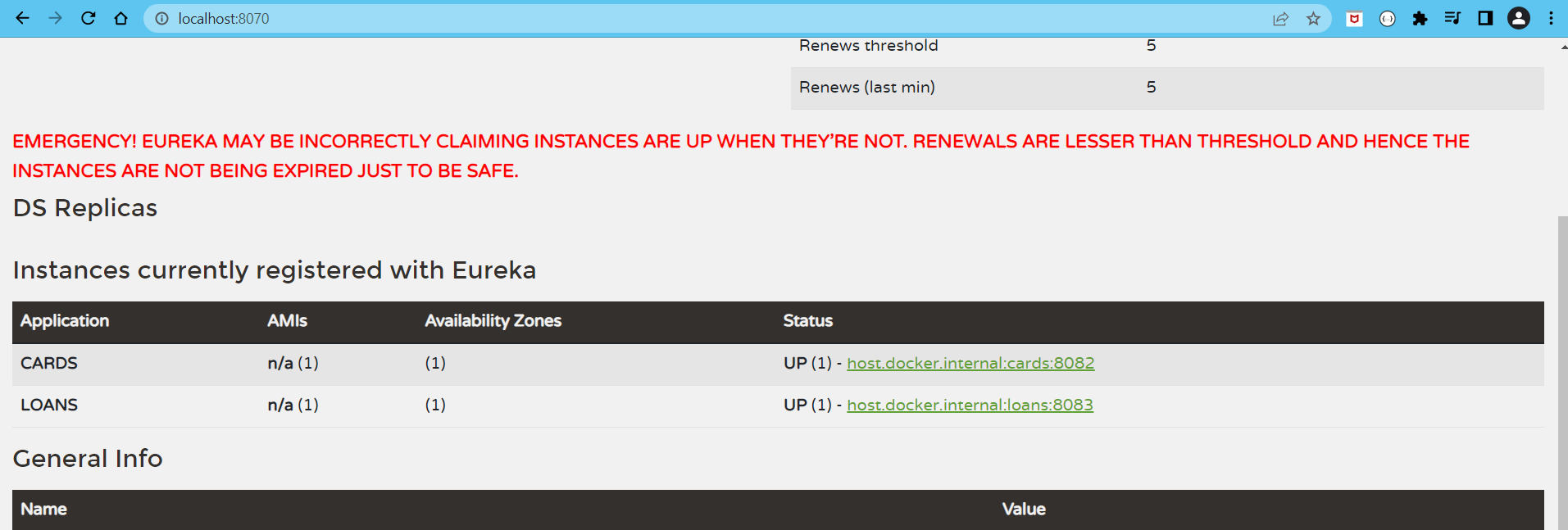
As this shutdown URL is of POST method so directly calling that URL using get method on browser will give us error we need tool like POSTMAN to call that URL using POST method as below:



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



This response is taken from Account microservice consoled

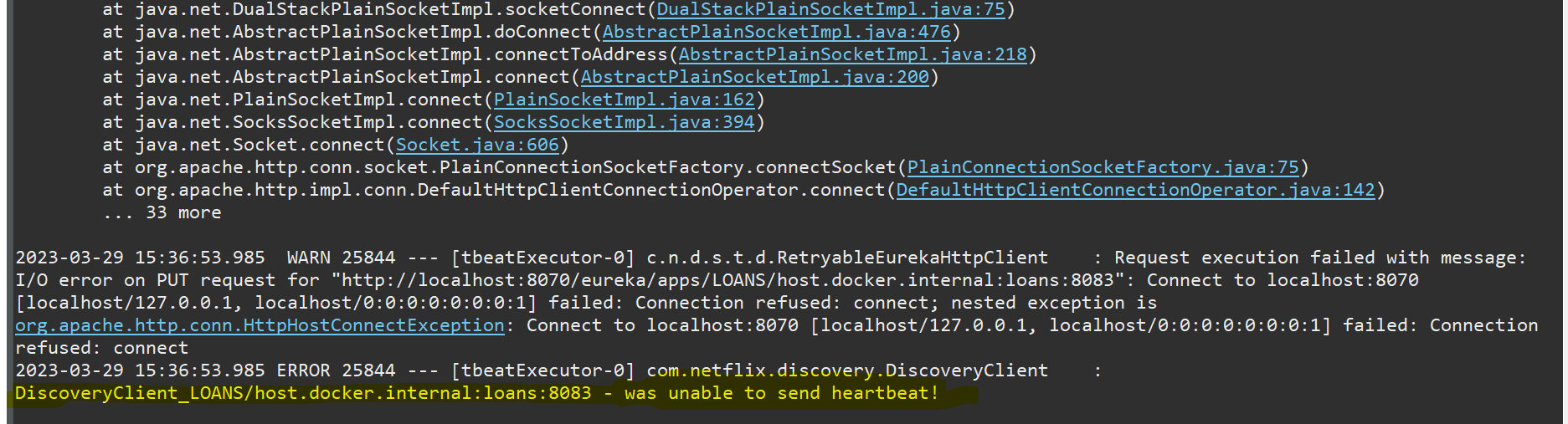


As seen in Eureka server this Account microservice is successfully deregistered

**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

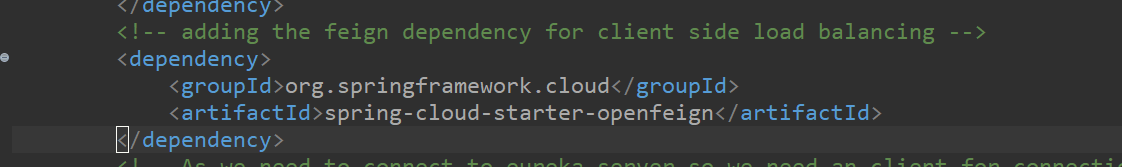
2] If I close the eureka service, then I can see the heartbeats from microservice trying to connect with Eureka server after 30seconds interval as below:



**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

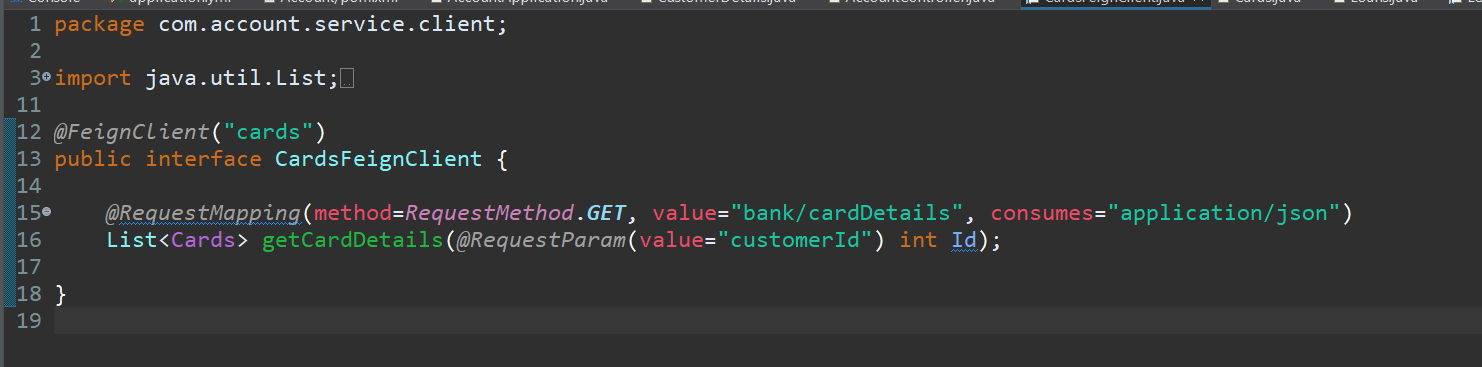
2] Add feign dependency into account microservice pom.xml file as below:



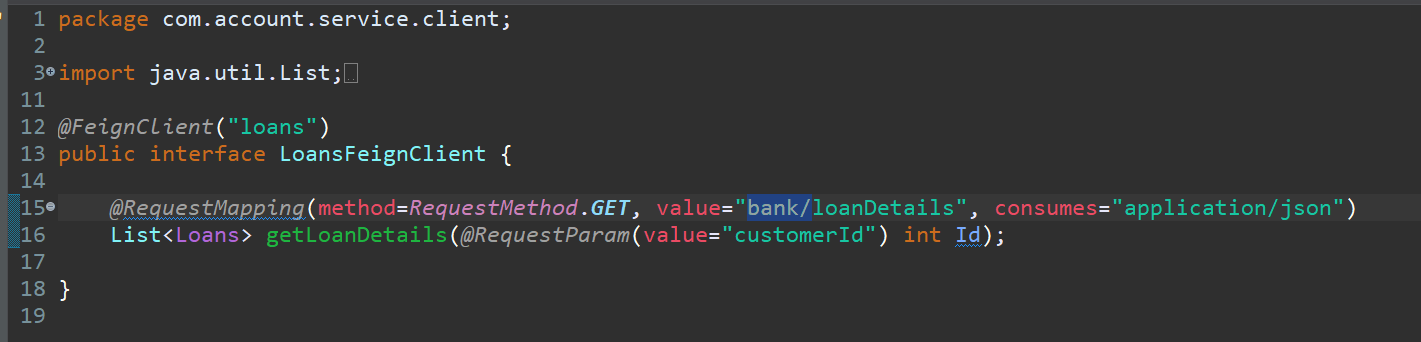
3] Here, I’am not adding this dependency in to loan and card microservice as I want to invoke all other service from Account microservice only i.e. the loan and card service registered inside eureka server should be accessed by account microservice

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

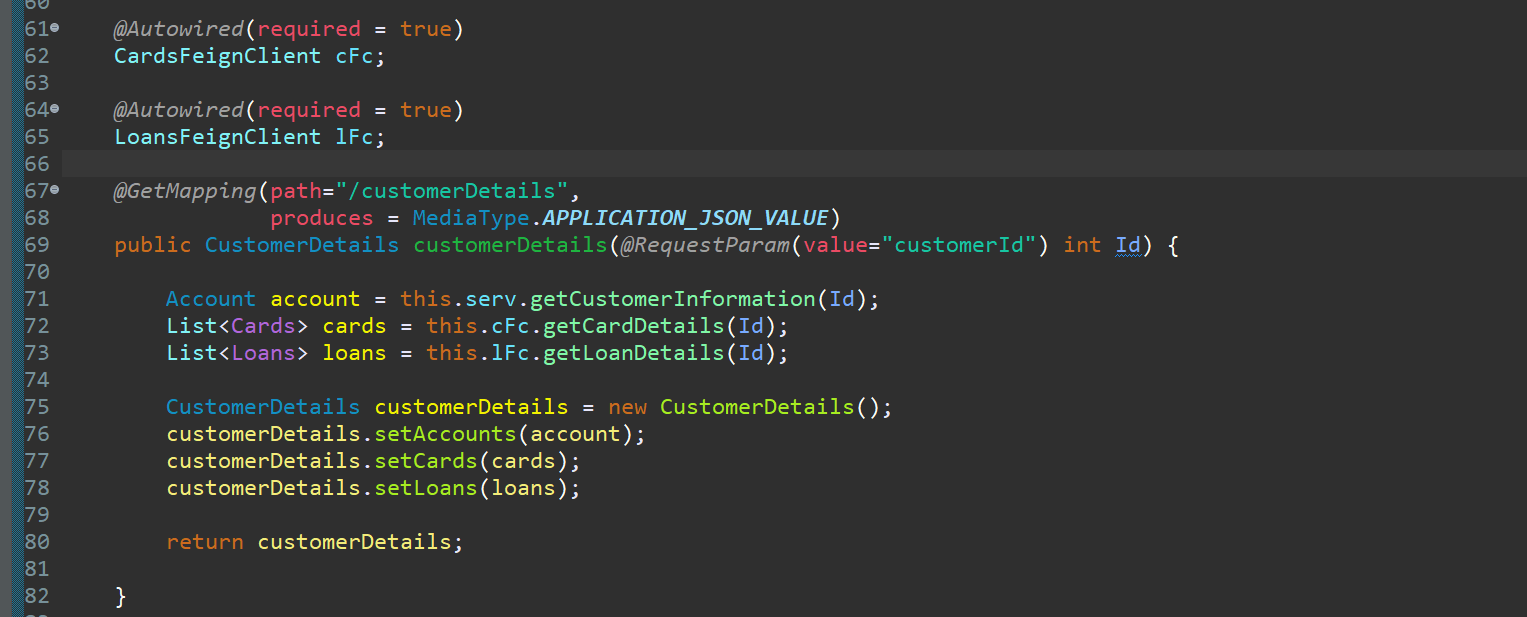




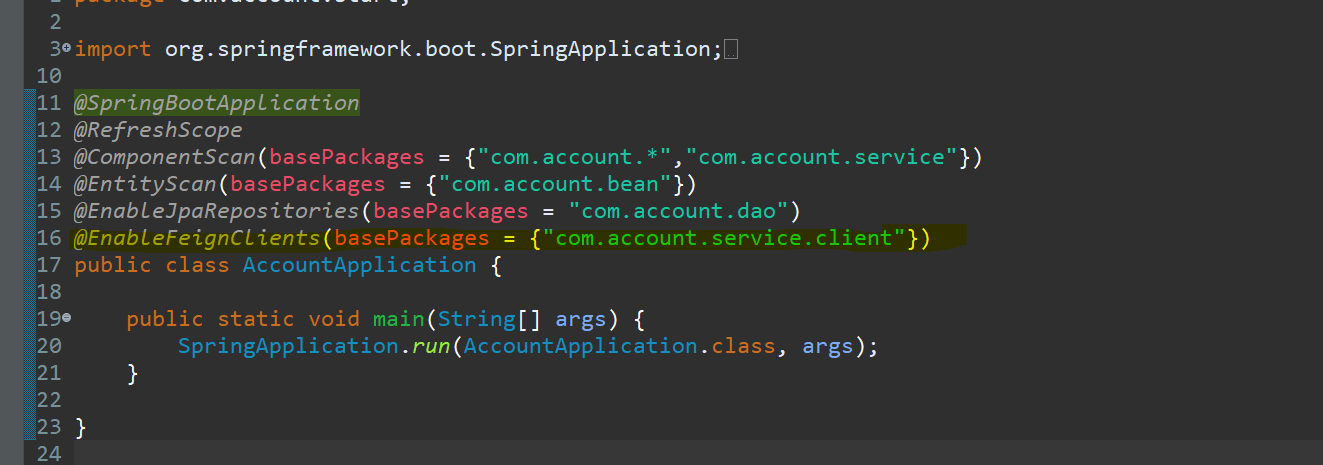
Here inside **@FeignClient(“cards”)** here, cards is the application name present inside eureka server, and inside **@RequestingMapping** value parameter i.e. **value=”bank/cardDetails”** this is the controller URL of card microservice from which we will fetch cards details



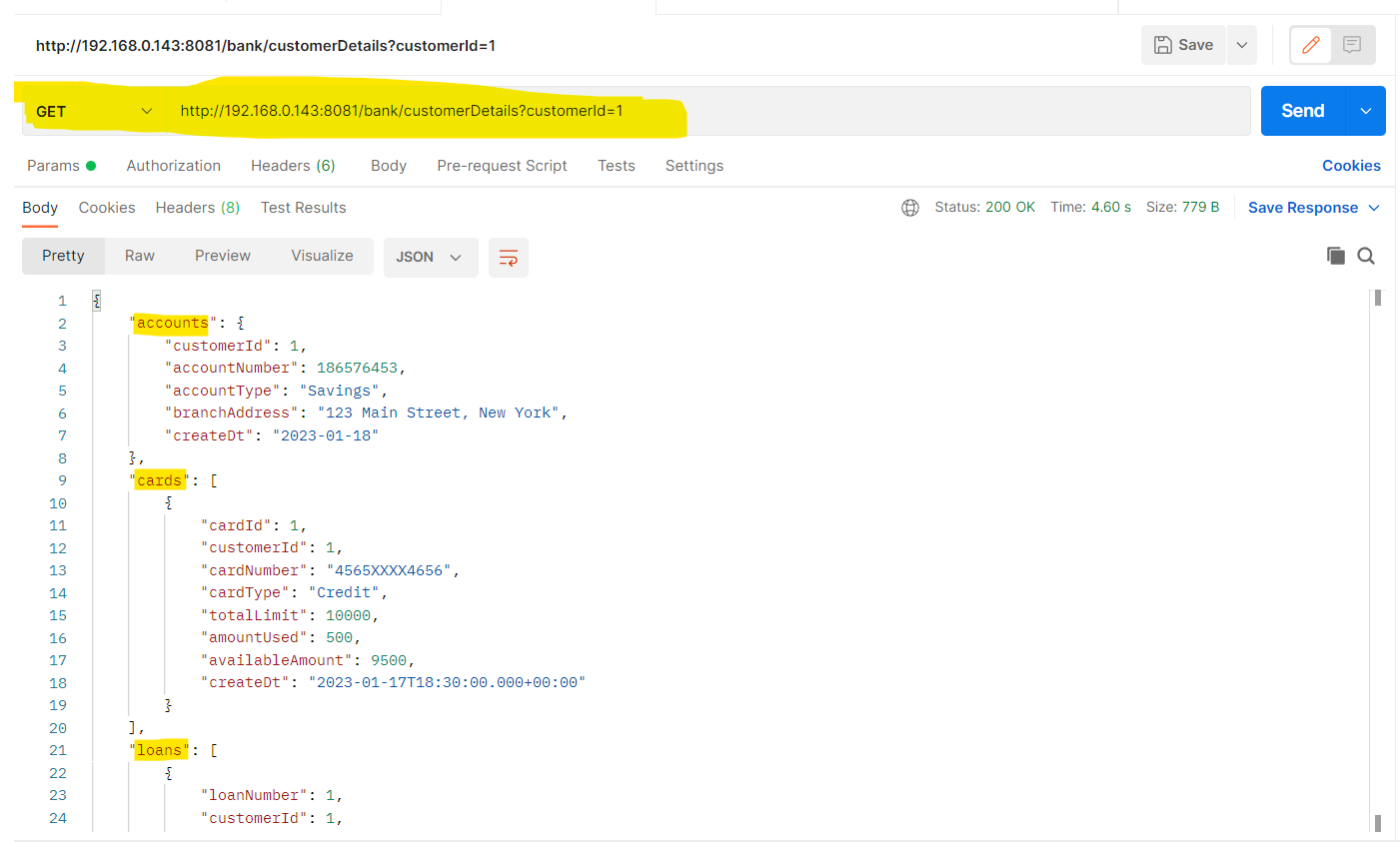
5] Add a new Get method inside Accounts controller



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



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



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