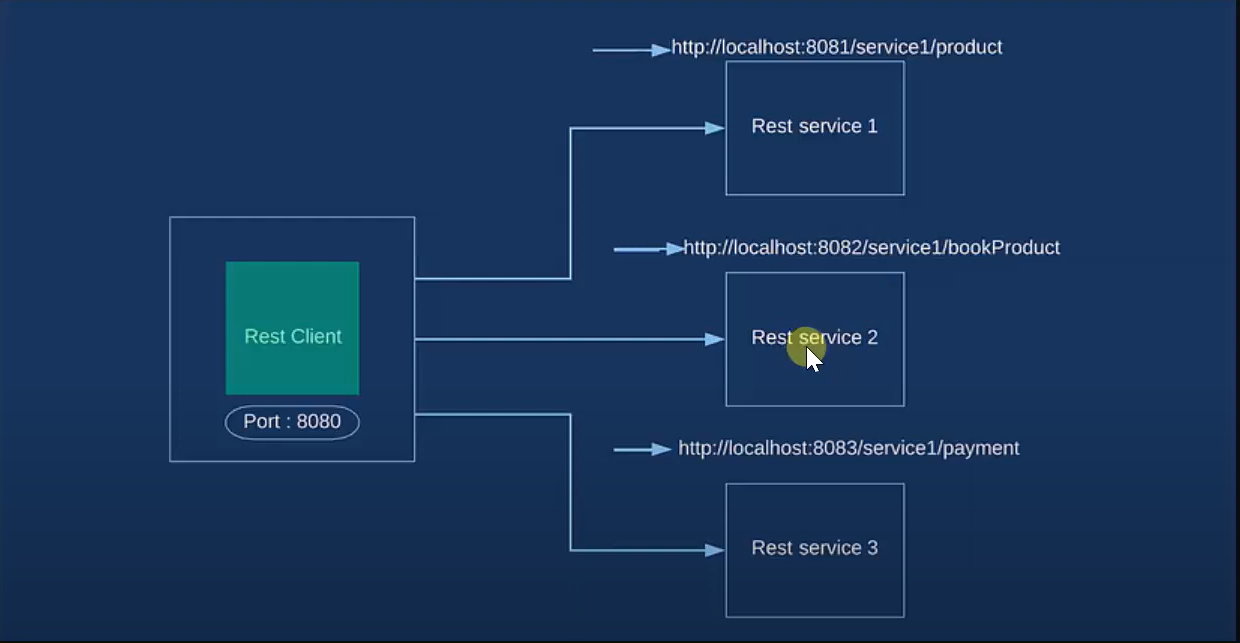
Eureka server -> It’s server registry

Orthodox ways -> this is tight coupled as we need to remember the concerned urls exactly

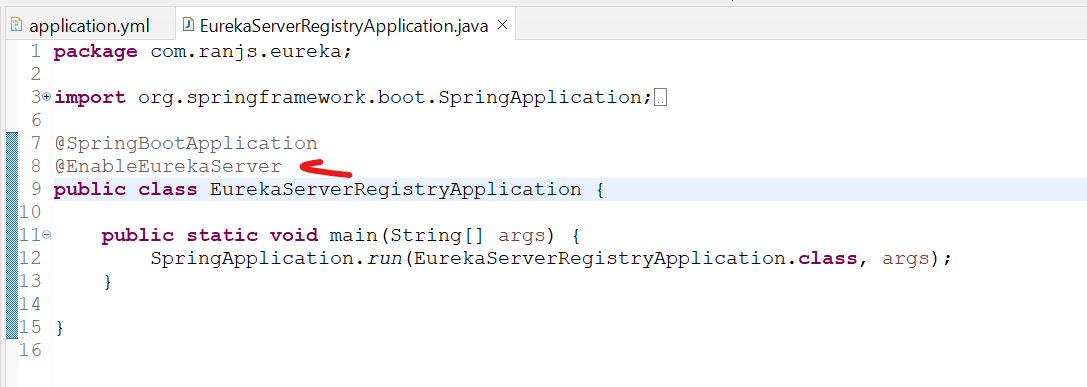


Microservices -> using eureka server, we register all the service which we need to use for our client



To make eureka server -> need to eureka server dependency in pom.xml

Need to annotate eureka serer config class with @EnableEurekaServer

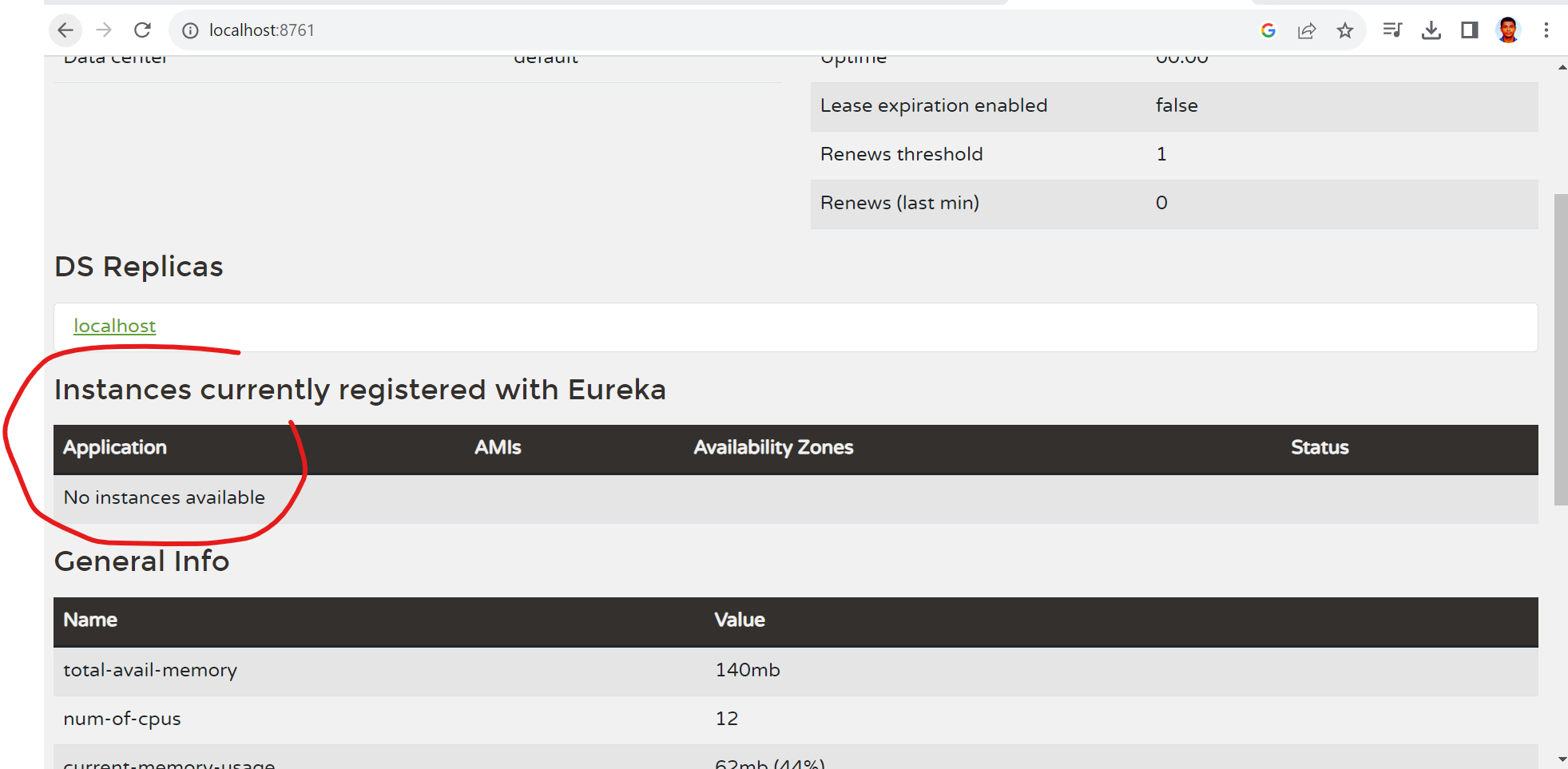


Create yml file in resource folder:-



And start the springbootapplication

Hit localhost:portno/



Initially no instance available, once we register some services they will reflect there.

To register service with eureka server -> need to add eureka discovery client in pom.xml

Need to add eureka client dependency in each service.

Need to annotate the each service config file with @EnableDiscoveryClient

Earlier we annotate with @ EnableEurekaClient but now @EnableDiscoveryClient.

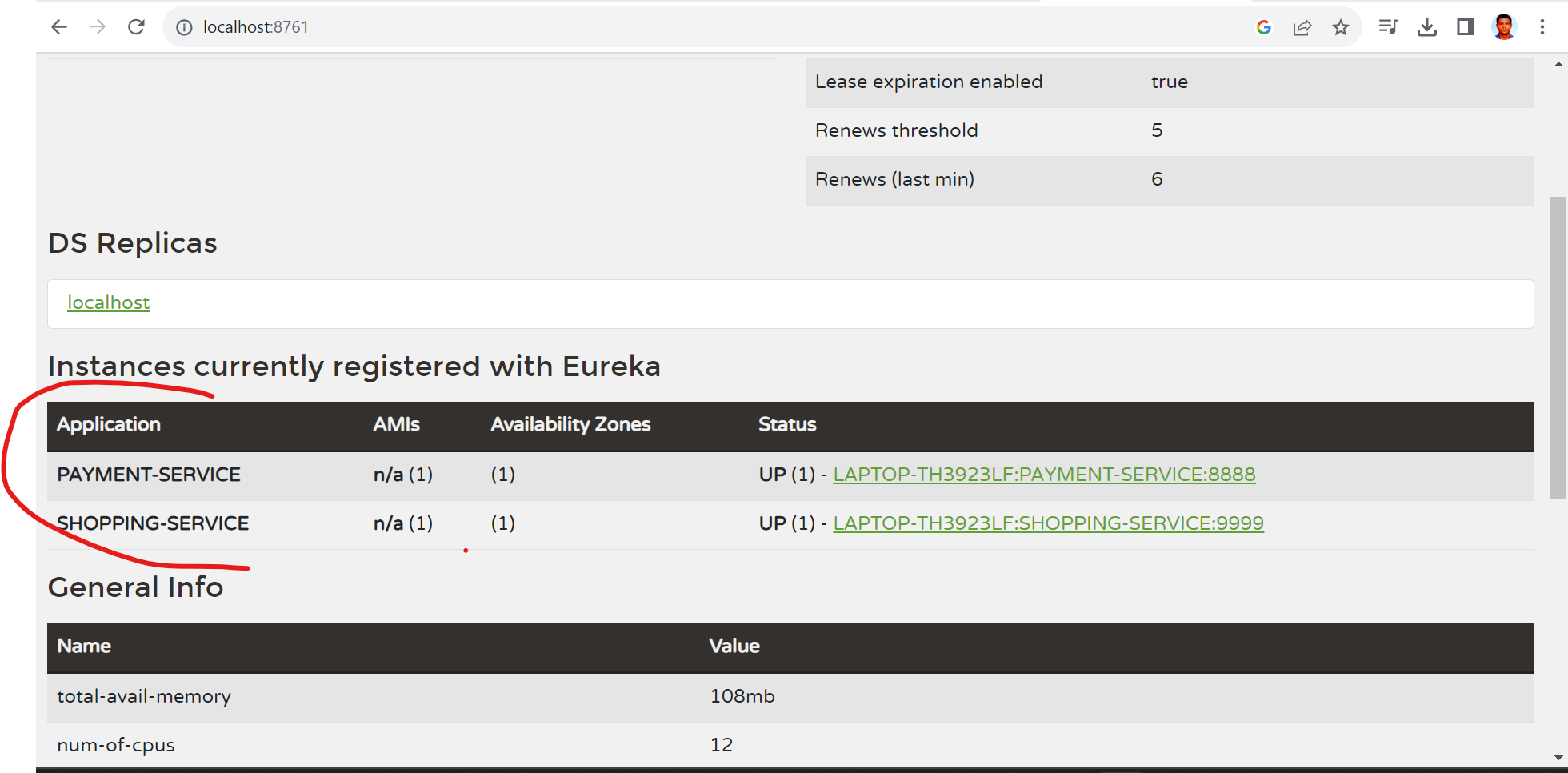
Add yml file in individual service like :-



Note: - **before starting individual service, ensure eureka service registry is running already**.

Now we can see the service which we register in registered with service registry: -





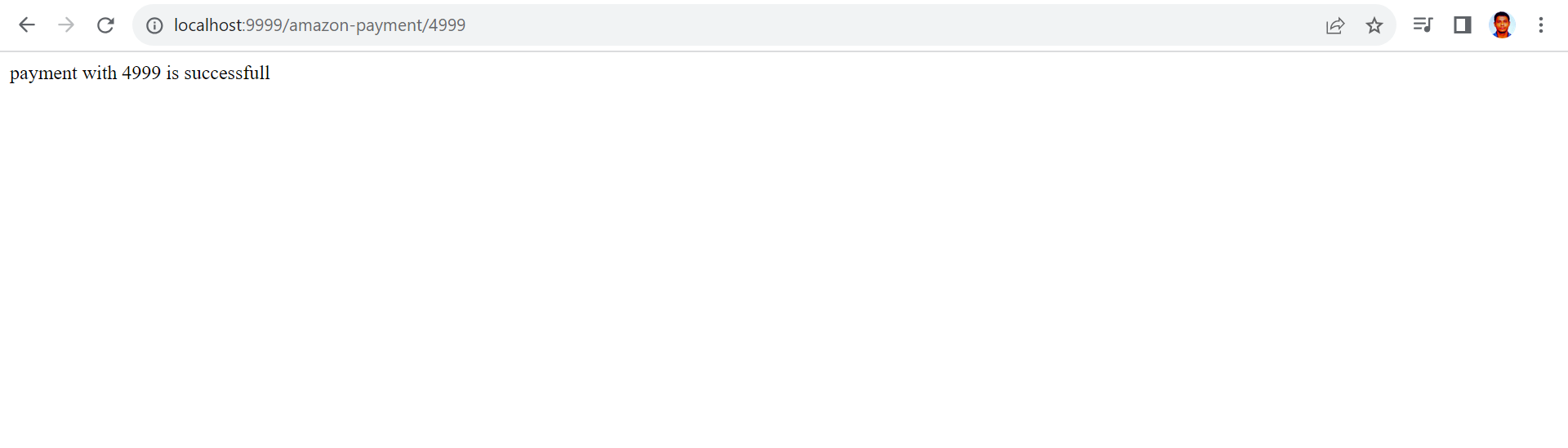
Old urls were:-

<http://localhost:8888/>..... & http://localhost:9999/....

So, our new urls will be

<http://PAYMENT-SERVICE:8888/>..... & <http://SHOPPING-SERVICE:9999/>....

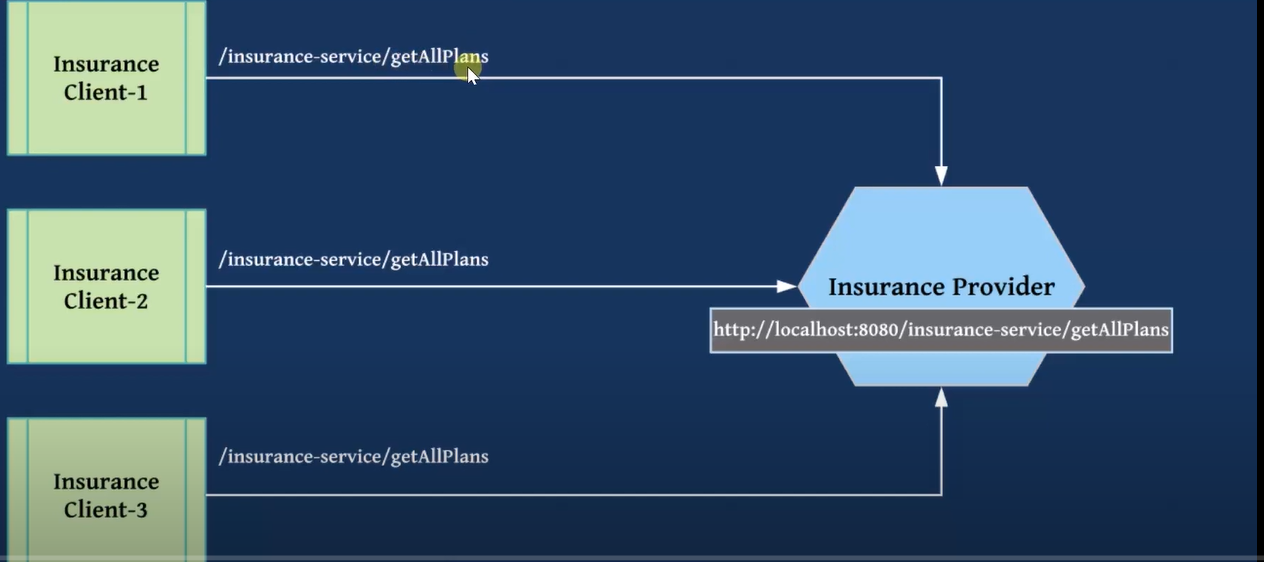
<http://localhost:9999/amazon-payment/4999>



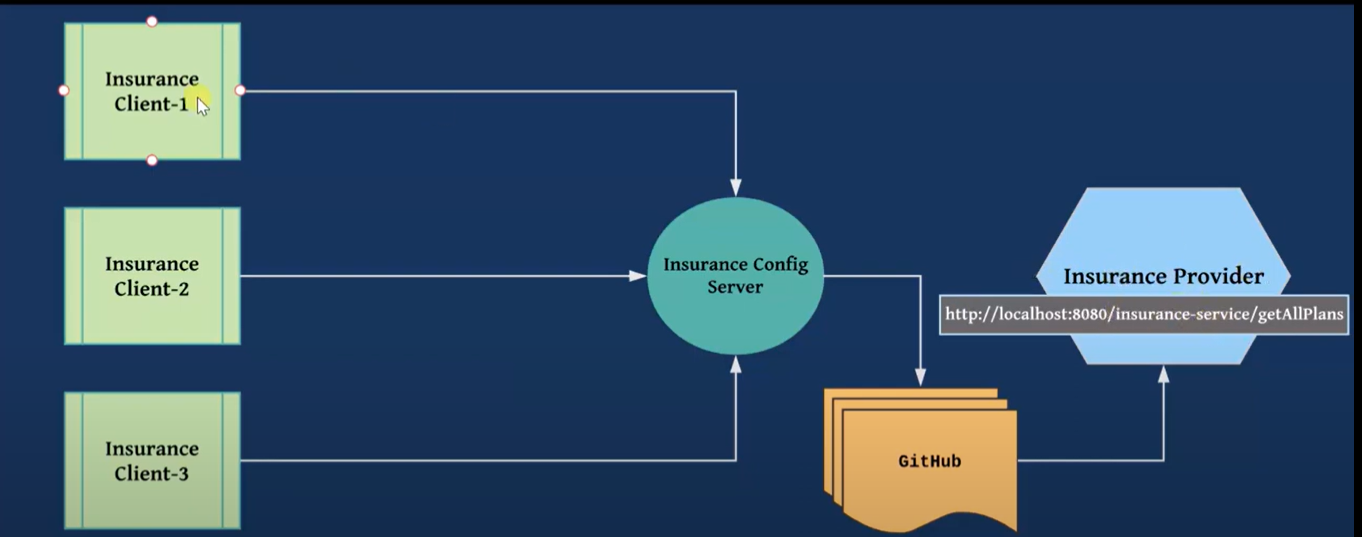
So, working fine.

**Spring Cloud Config** provides server-side and client-side support for externalized configuration in a distributed system. With the Config Server, you have a **central place to manage external properties for applications across all environments**.

Ex. In below all insurance client is calling end-point /getAllPlans but let’s say Insurance Provider updated its url to /getAllUpdatedPlans then we need to make changes in all in insurance client viz 1,2 and 3.



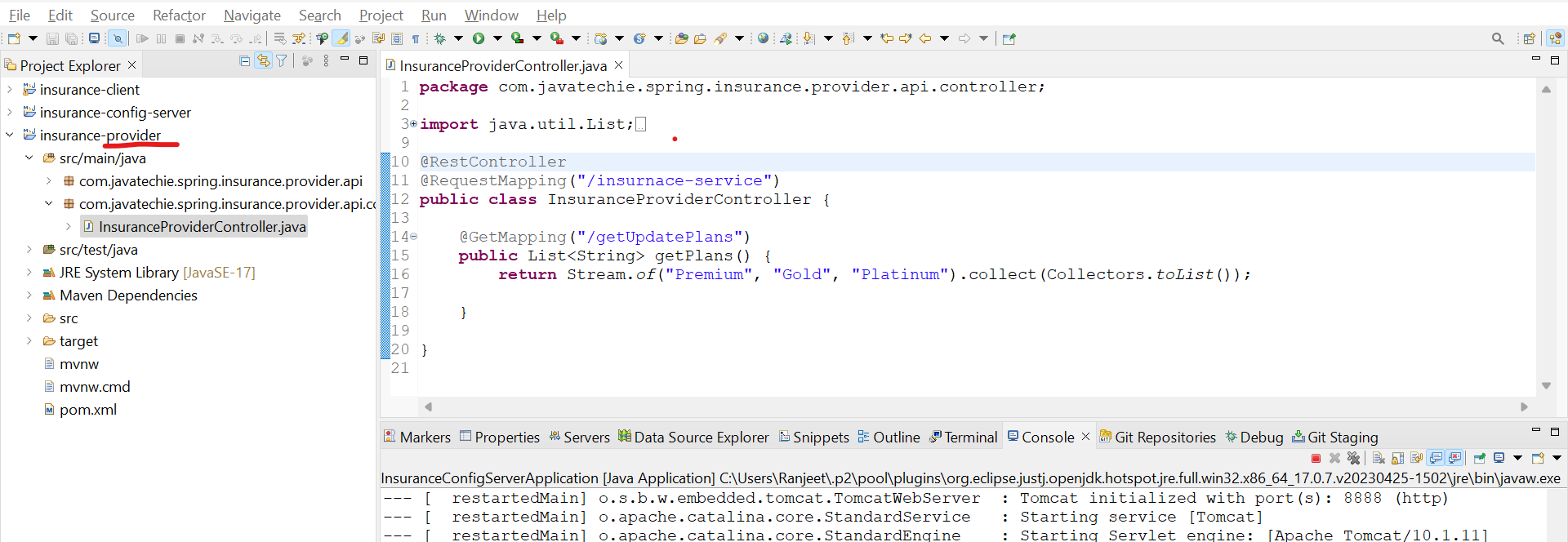
Using spring cloud config:-



All clients won’t directly calling the insurance provider url and instead it will interact with centralized config server. Config server acts as entry point.

🡪step 1.

Create a new repository in github namely insurance-provider which will run at 8080 default port



🡪 step2.

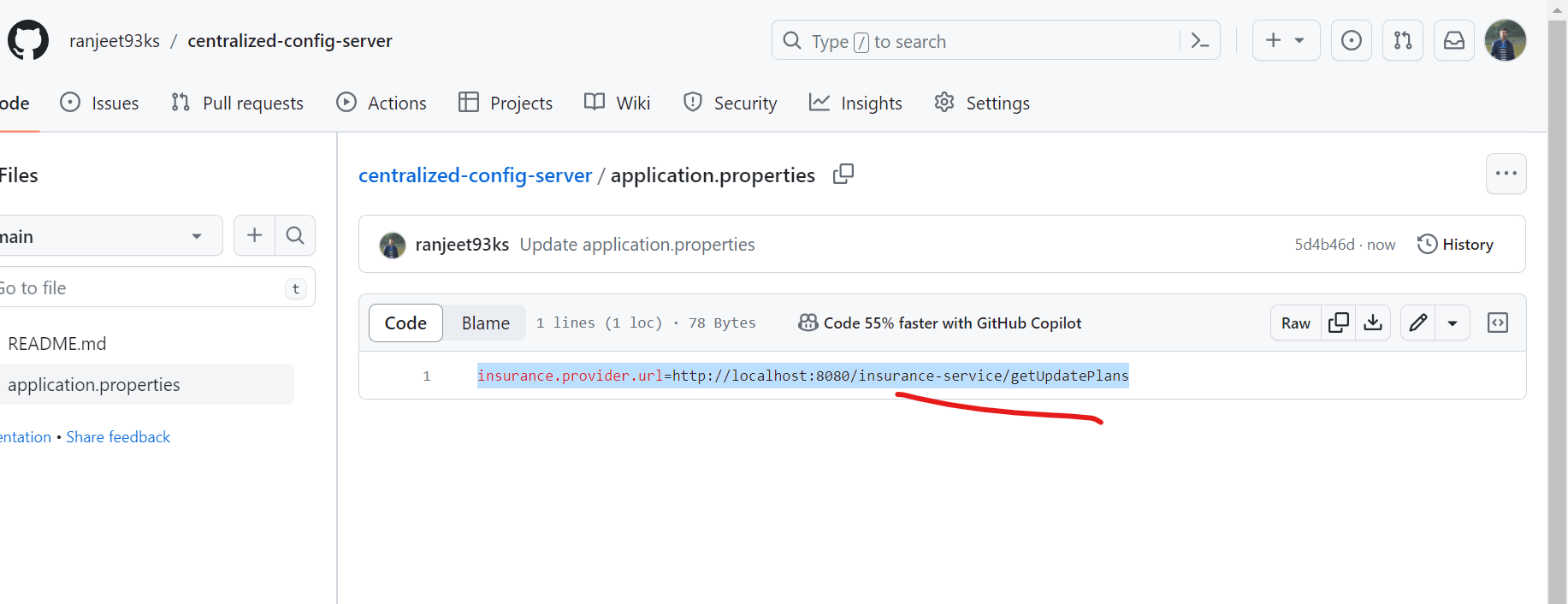
Create a new repository in github namely https://github.com/ranjeet93ks/**centralized-config-server** and add README file

Now, create a

ADD FILE -> Create new file

application.properties

And add 🡪 insurance.provider.url=http:localhost:8080/insurance-service/getUpdatePlans

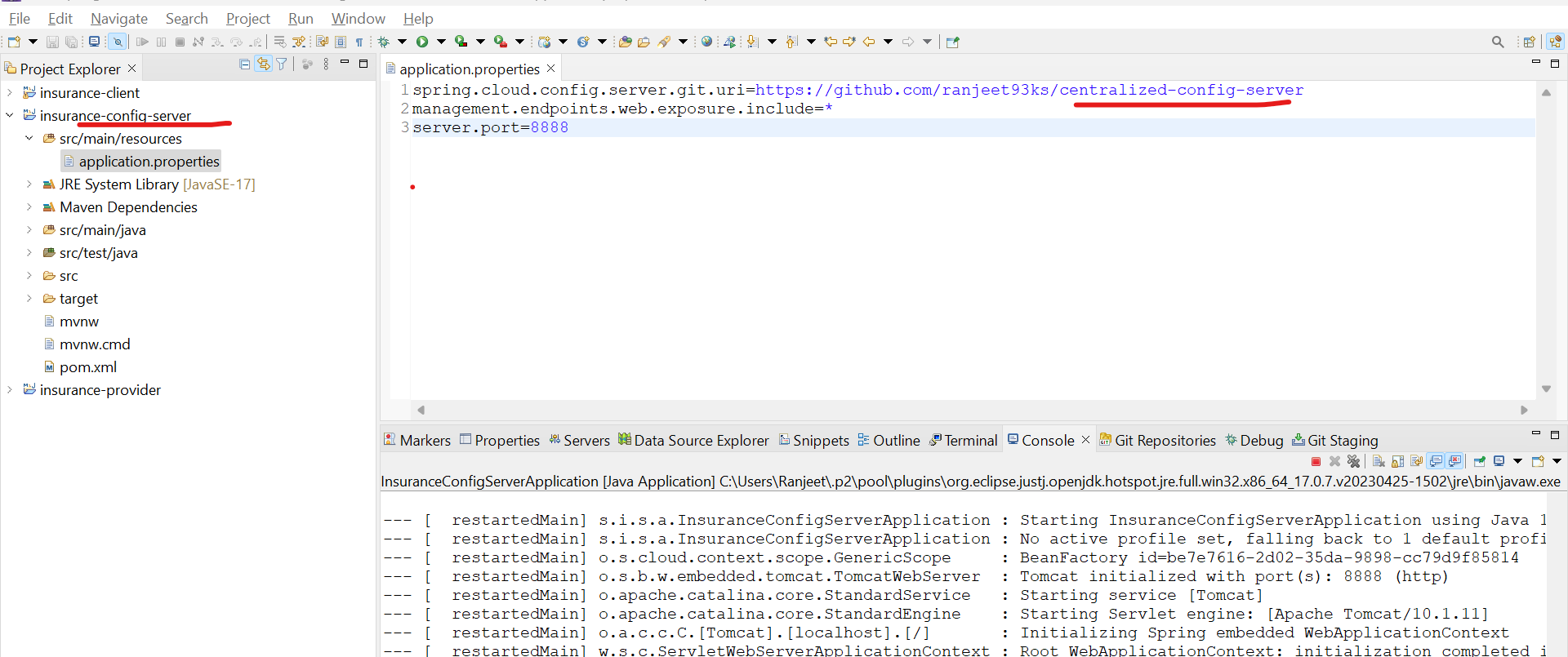


🡪Step3 .

Now create a new sprinboot job namely **insurance-config-server**

And in its application.properties add url of cofig server

spring.cloud.config.server.git.uri=https://github.com/ranjeet93ks/centralized-config-server

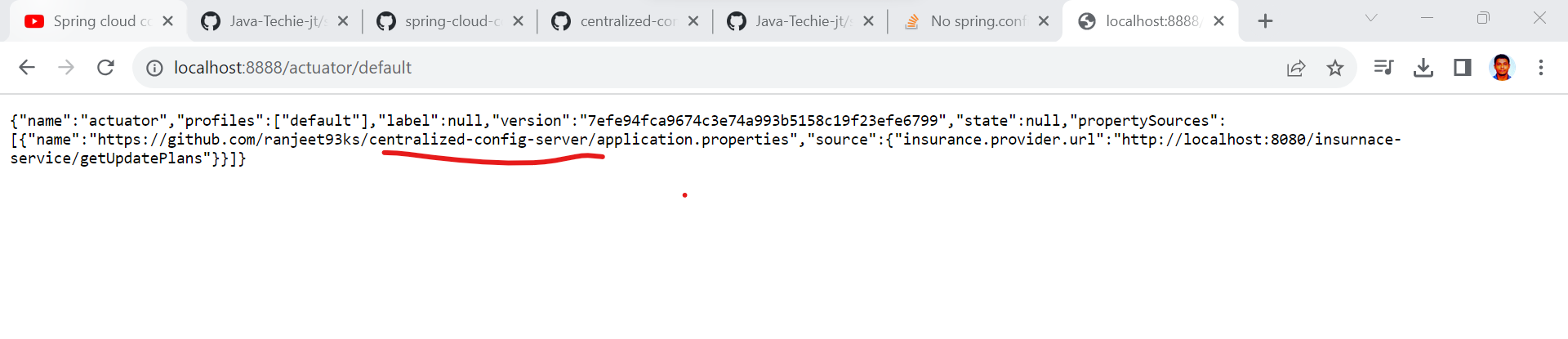


Add config-server dependency

And add @EnableConfigServer in springbootapplication class

Once start the springbootapplication class

Hit :- <http://localhost:8888/actuator/default>



We will read details from application.properties of config-server from github

🡪step4.

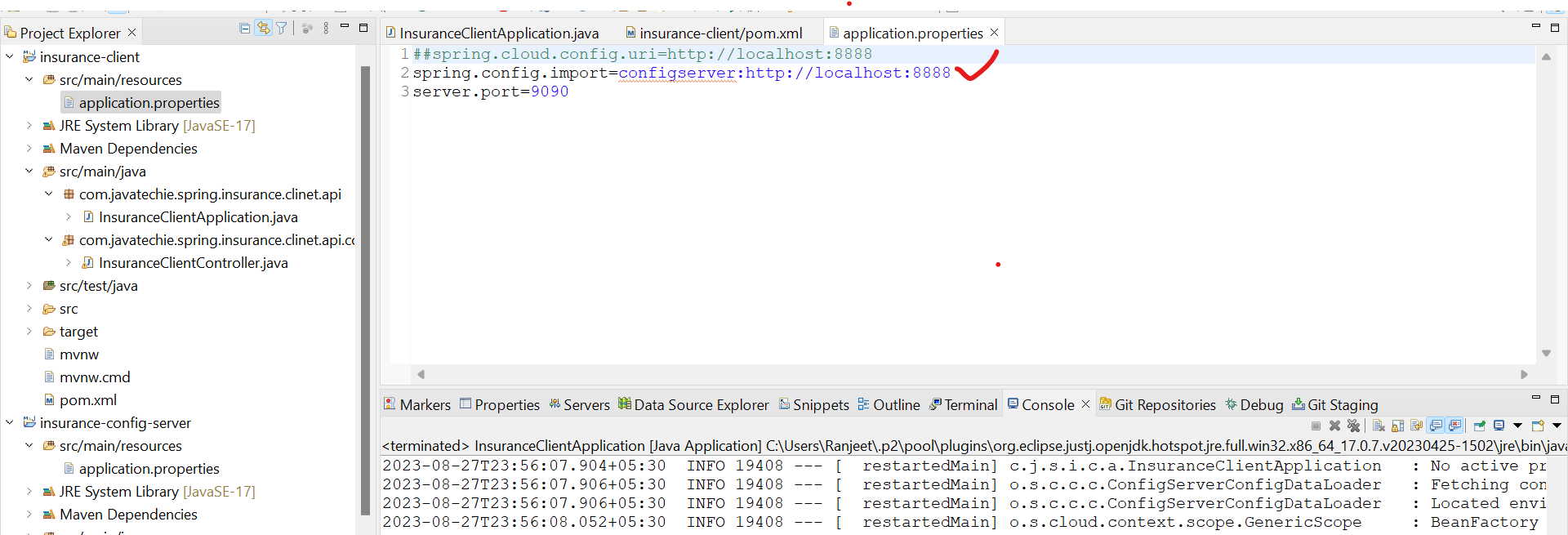
Now, create new client springboot service namely **insurance-client**

Add config client in dependency

In app.properties

Need to add include configserver like in our case insurance-config-server which runs on 8888 port and read properties from github()

spring.config.import=configserver:http://localhost:8888



Now, start this client application as well

Finally hit: - <http://localhost:9090/getPlan>

Finally, let’s say in insurance-provider url from

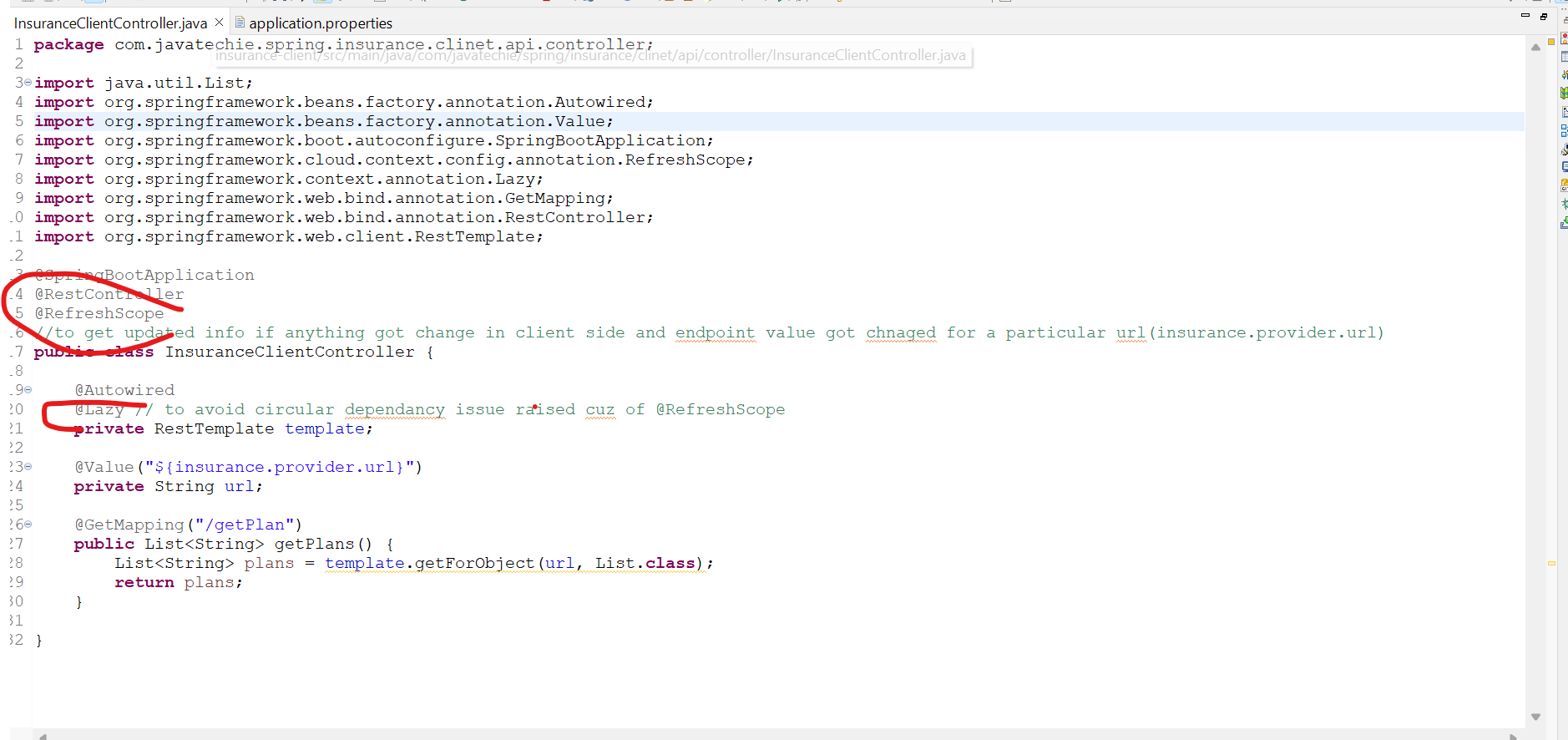
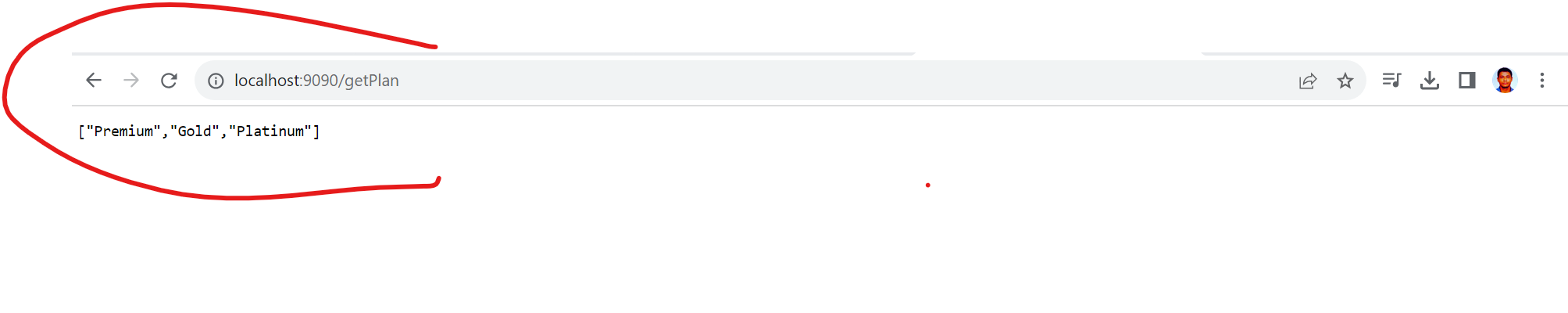
<http://localhost:8080/insurance-service/getUpdatePlans> changed to <http://localhost:8080/insurance-service/getNewPlans>

and in centralised config server also we changed to

insurance.provider.url=http://localhost:8080/insurance-service/getNewPlans

So this to be reflected instantly in client side, we added @RefreshScope

@Lazy // to avoid circular dependancy issue raised cuz of @RefreshScope

Note:-

1.client provider provides service at 8080 port (<http://localhost:8080/insurance-service/getUpdatePlans>)

2.this getUpdatePlans url has been added in centralized-config-server in github.

insurance.provider.url=http://localhost:8080/insurance-service/getUpdatePlans

3. a local config server service has been created and in its app.properties we added the location of centralized server which will run at 8888

spring.cloud.config.server.git.uri=https://github.com/ranjeet93ks/centralized-config-server

4. we create the insurance-client springboot service which will call insurance-provider’s end-point

And in app.properties we need to let them to look for http:localhost:8888 which local config server

And finally client will utilize the provider end-points via local config server and local config server then get the exact endpoint from centralized config server

Reference :- <https://www.youtube.com/watch?v=x1BR0D-buQg&list=PLVz2XdJiJQxz3L2Onpxbel6r72IDdWrJh&index=2>