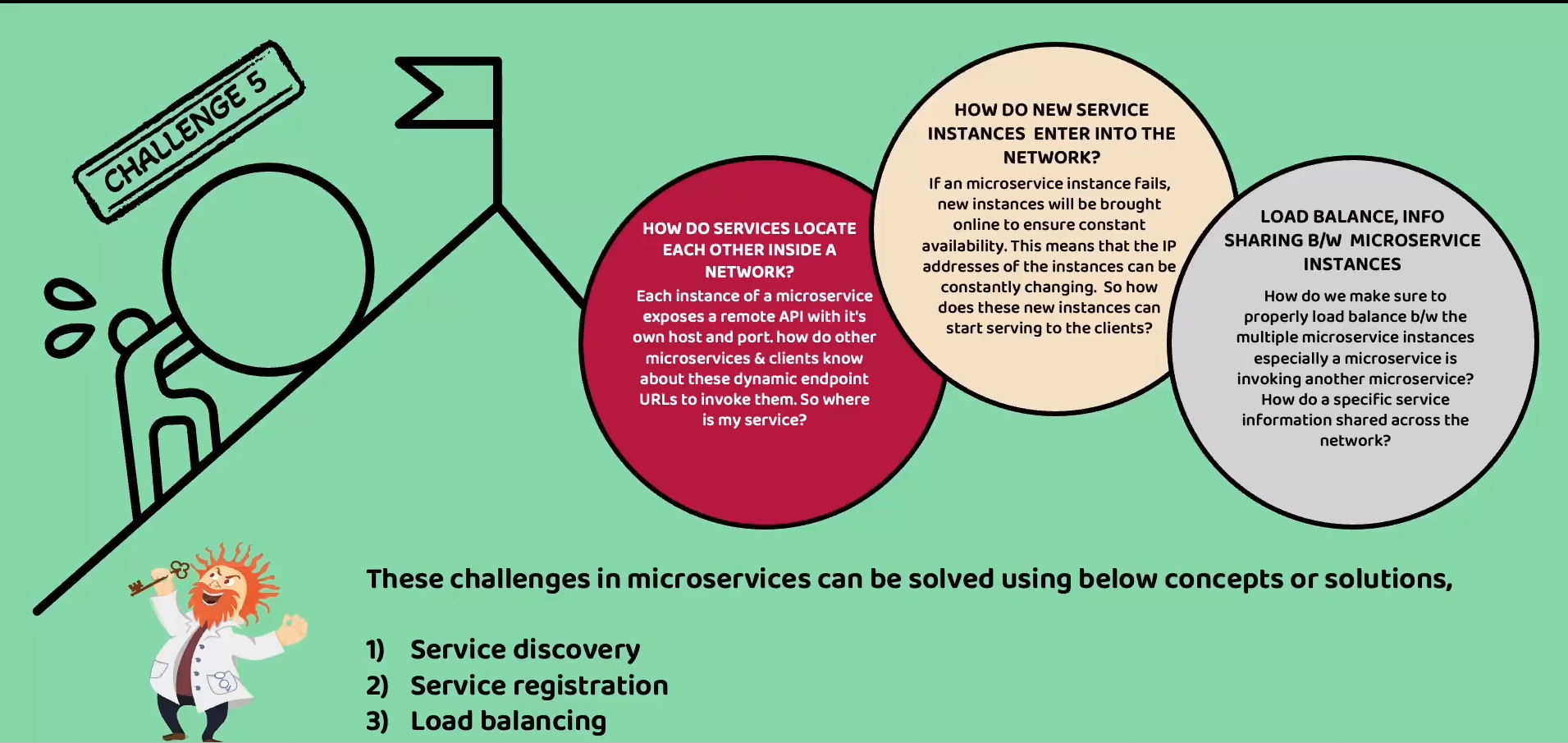
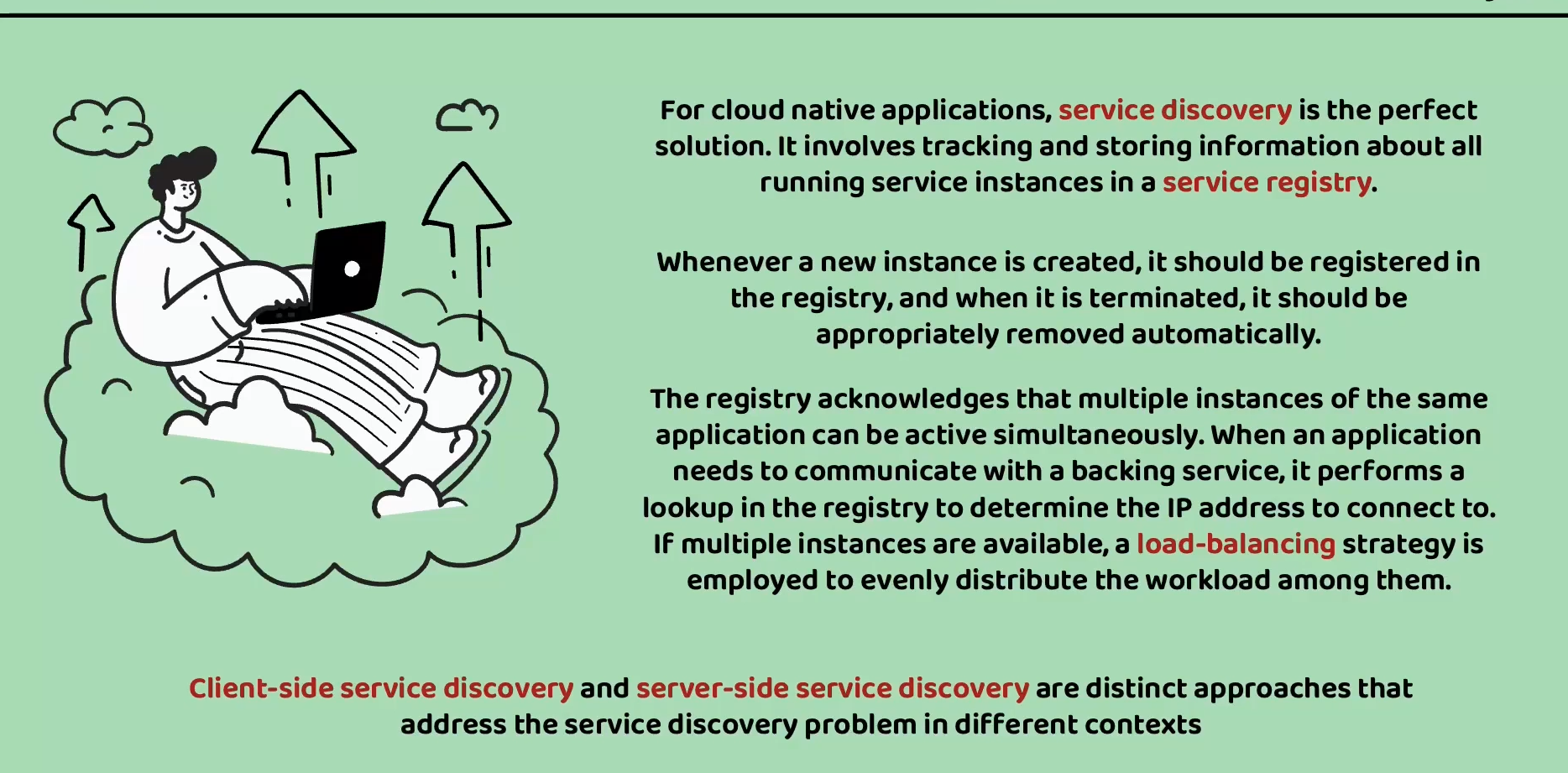
Service Discovery Eureka Server

Sometimes there is a requirement that where microservices have to communicate with each other in microservices network such communication is called internal communication. Such communication is possible when each service know about the ip address and port number of all other microservices. However this is not possible in microservices environment as there may be multiple instances of microservices be running. This issue can be overcome by using Eureka server where all the services registered to it at the start up and share the information like ip address, port number and all. Which is required for internal communication. These microservices keep sending heartbeats to the service discovery registry server for every intervals. Also these microservices deregister itself whenever microservice is shutdown.

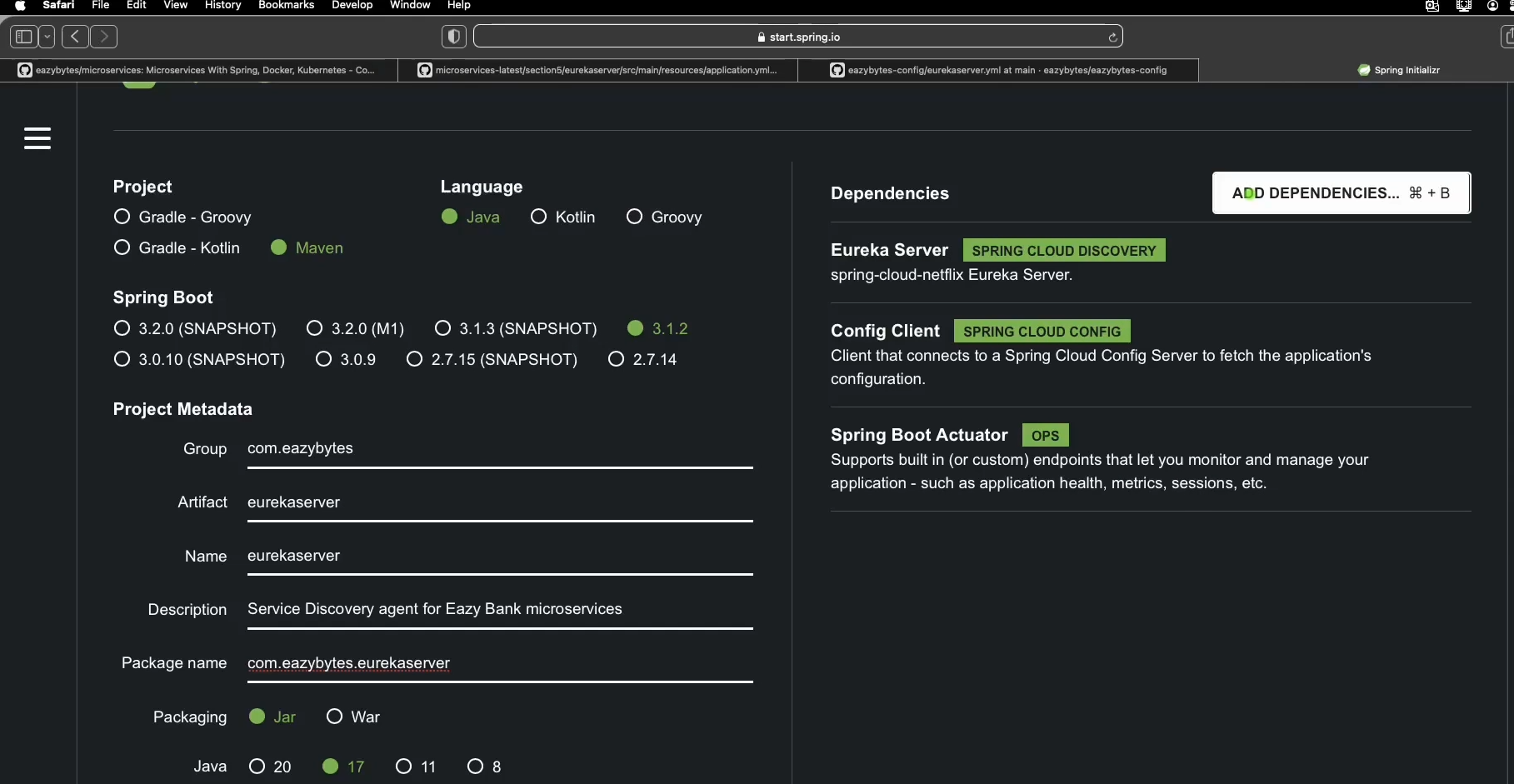




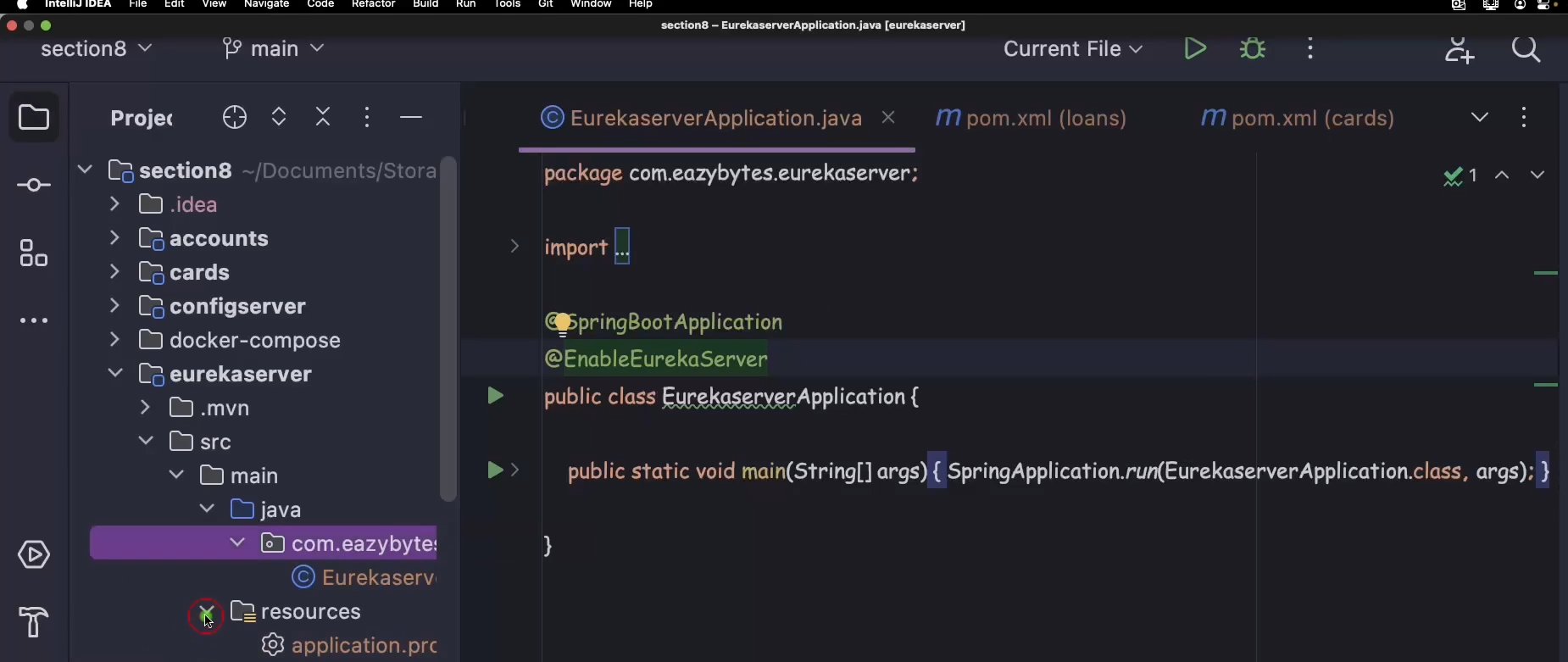
All the microservices register with client side service discovery. And this server contains service discovery node that maintains all the microservices information. When ever any microservice wants to communicate with other microservice then first it communicates with service discovery server through logical name to get all the ip addresses of that particular microservice then load balancing is done at the client side microservice. Also these microservice cache these details so that there is no frequent requests to service discovery server. And also this cache gets refreshed any change in service discovery server



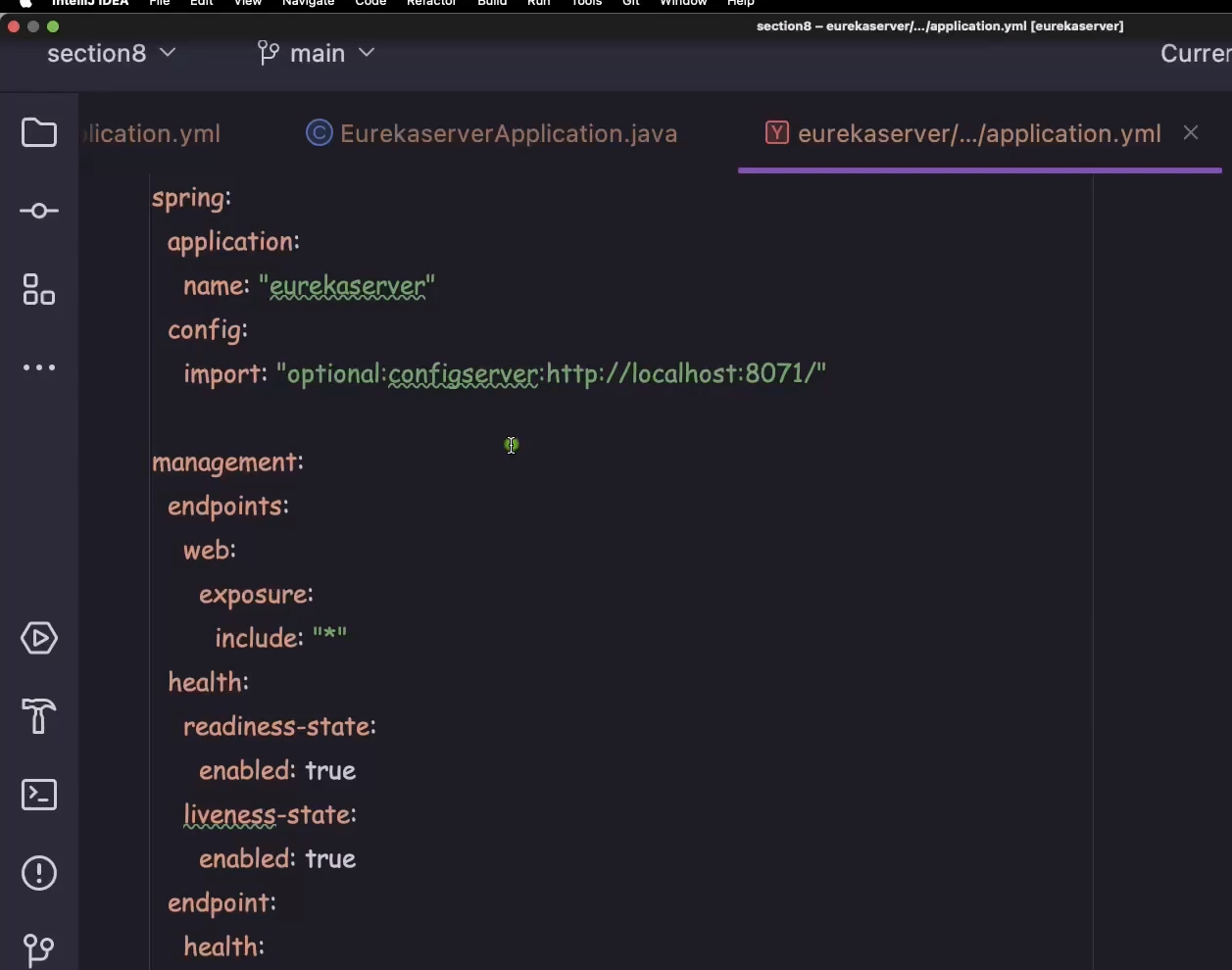
Now lets build service discovery server with below details



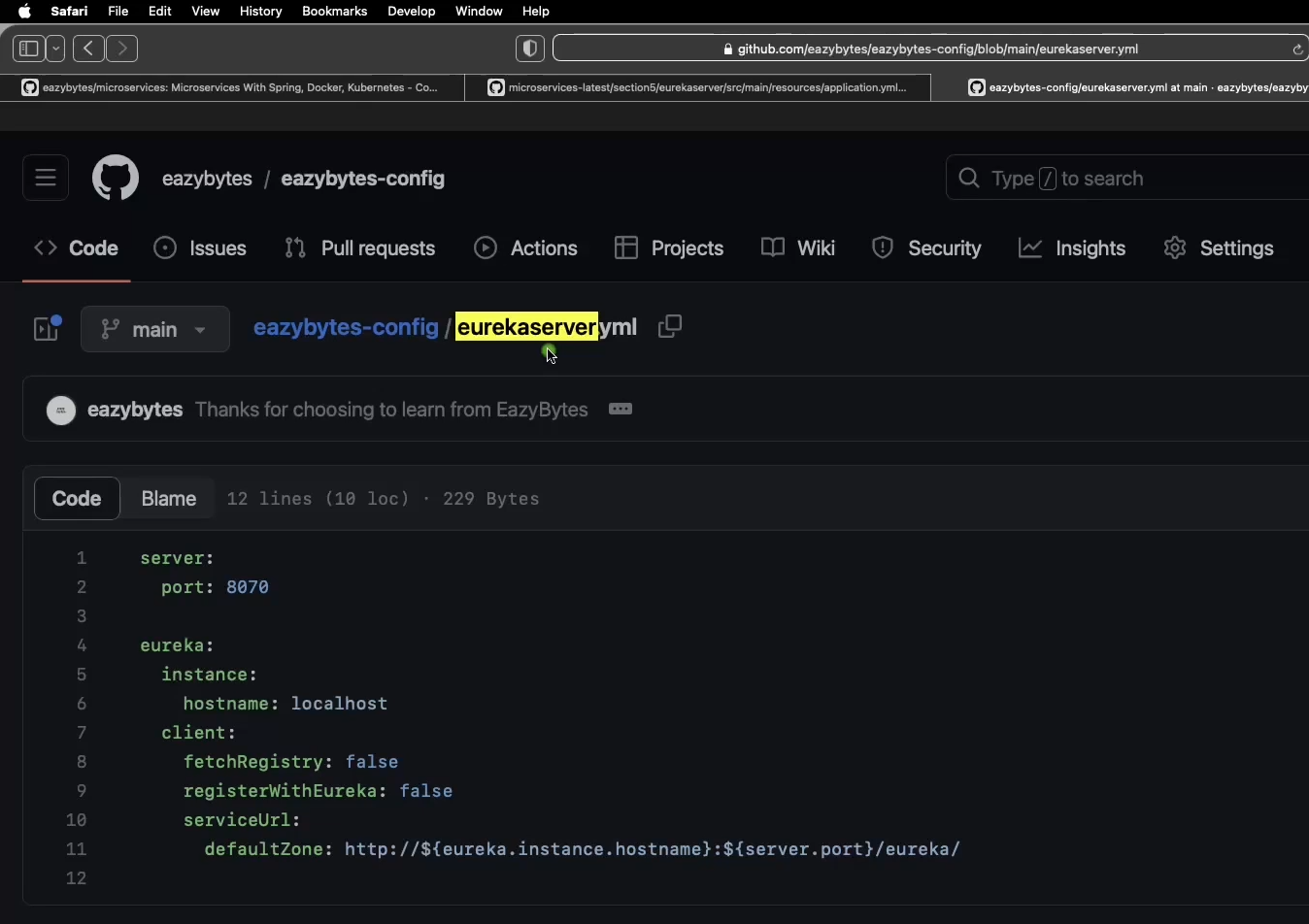
Now we need to add annotation @EnableEurekaServer to main class as below which enables this application to act as a service discovery server



Now add the properties related to the service discovery server such as application name, config server and health related properties. Health related bcoz service discovery server should start before all other microservices so that we can configure in docker compose file.



And eureka config file in git hub repo

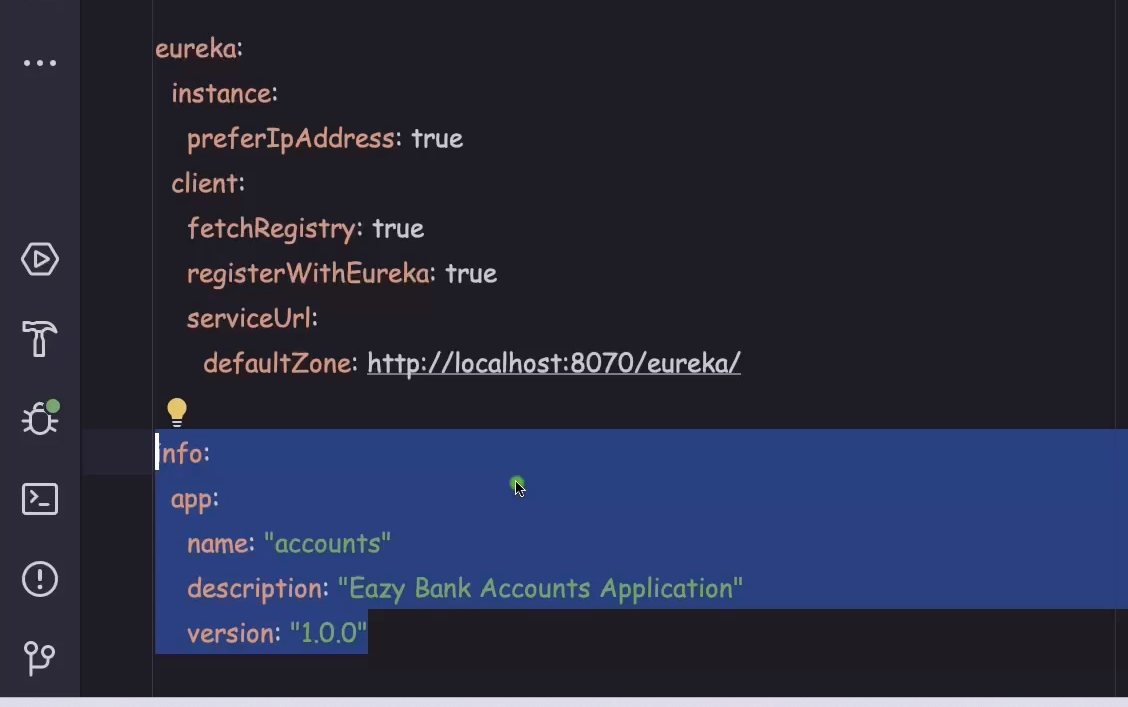


Here fetchregistry is false bcoz it doesnot communicate with other services hence fetching of all the microservices details is not required and registerWithEureka is also false bcoz itself is server

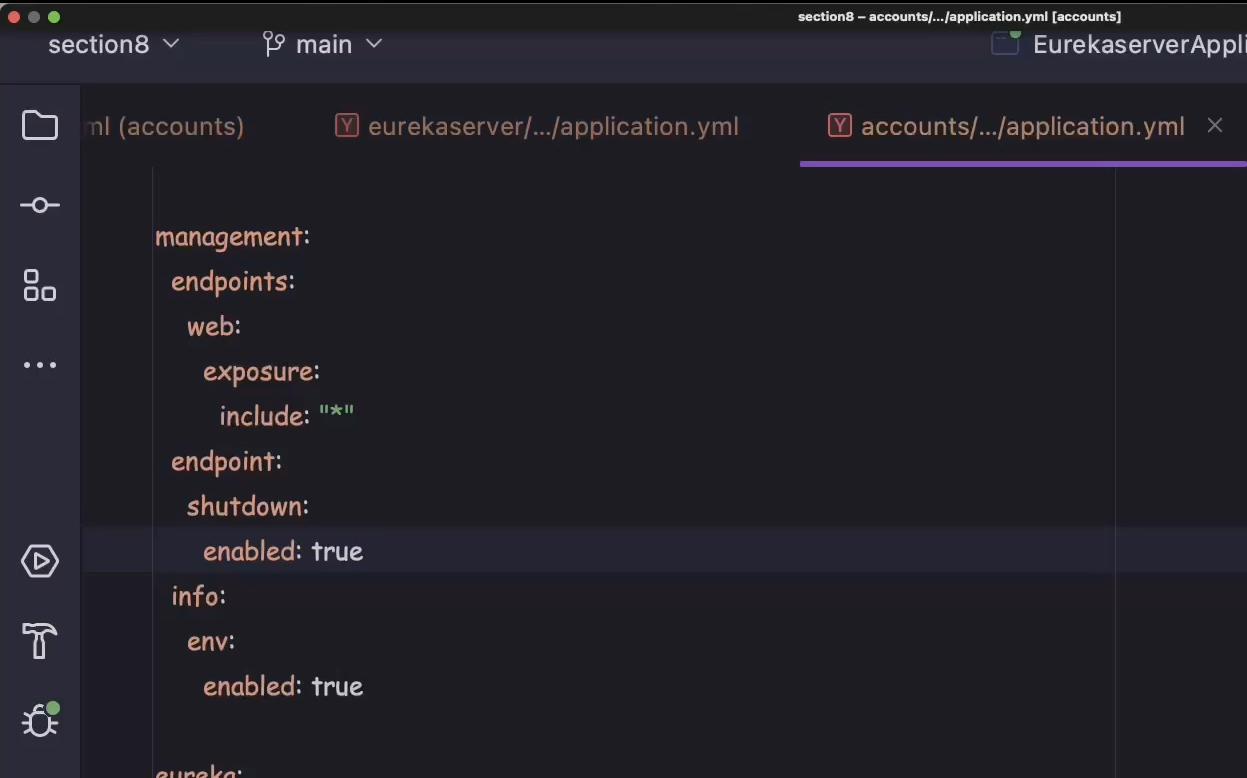
Now start the config server and then service discovery when you access eureka server in browser it will display the dashboard and also the number of instances registered to it.

Now lets make changes to our services. We need to add Eureka client config dependency spring-cloud-starter-netflix-eureka-client. And add the below properties in service application property file

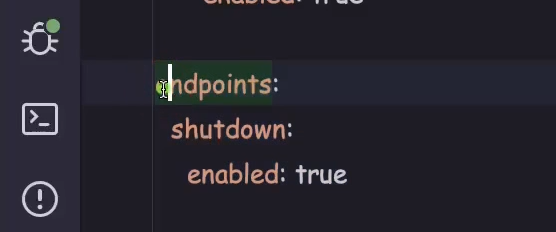
preferIpAddress is enabled true bcoz microservice will try to register with eureka server with ip address instead of hostname. So that microservice can communicate using ip address



For info properties to work we need to enable it under management properties and also for shutdown so that it deregister from eureka server



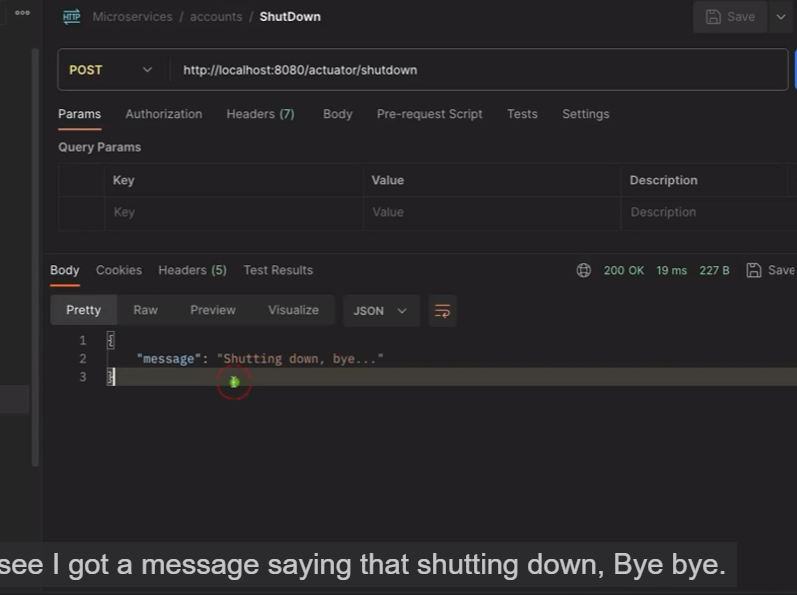
Also to expose shutdown endpoint we need to add properties as below



For more info

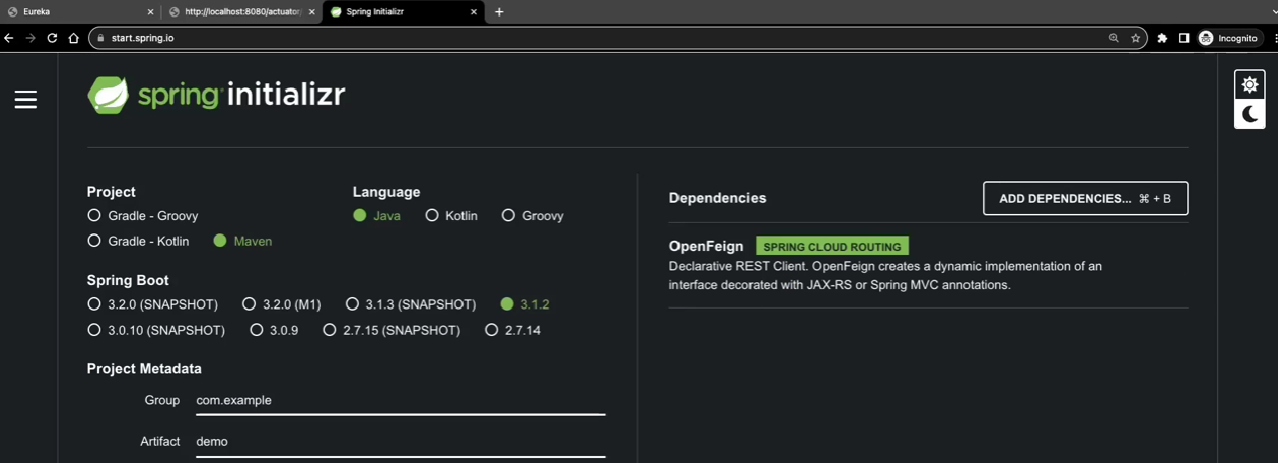
https://github.com/eazybytes/microservices/blob/3.2.3/section8/accounts/src/main/resources/application.yml

It is not good practice to directly shutdown the application. Instead we have to call shutdown which is the endpoint of the actuator we have added this management properties in the yaml file we just need to call this method from the postman. Post this it De registers from the Eureka server we can confirm this from dashboard



Now lets implement inter microservices communication using feign client

For that we need to add dependency Open Fiegn client goto spring initializr copy from expose



Lets add this dependency in all the services then in the main application file add the annotation @EnableFeignClients

Now in our microservice say accounts service which want to communicate with cards microservice to get cards details then we just need to create interface with abstract method whose signature is same as the endpoint defined in cards microservice. Notice that in @FiegnClient(“cards”) is the application name

