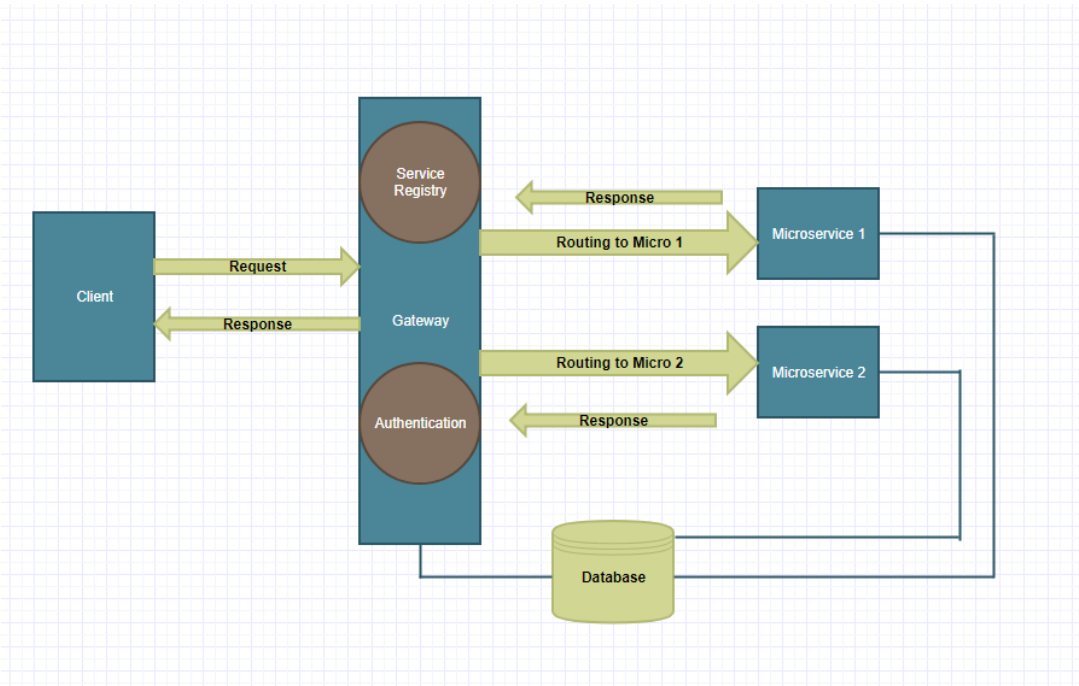
Spring-boot microservice with centralized authentication (ZUUL+Eureka+JWT)

arjun
nagathankandy
Oct 22, 2018 · 8 min read

We are going to discuss an architecture in which one microservice will act as a gateway service. Which does central authentication, redirect an incoming request to other microservices. The main advantage of this architecture you can easily add multiple microservices to the system and all authentication, authorization will be taken care from a central unit.

1. gateway service: Which will act as a central authentication unit, ZUUL proxy server for redirection.
2. cart service: This will act as a eureka client, which receive a request from gateway service.

Let's design the architecture like below. Don’t think too much we can split it and work nicely.



Architecture

First Step: Create a gateway using spring-boot microservice. Add ZUUL, Eureka server dependency to it. So gateway will act as ZUUL proxy server as well as service registry for Eureka client. Don’t worry we can dig deeper into it.

Second Step: Create one microservice with Eureka client dependency and register them in Eureka server.

Third Step: Test the application.

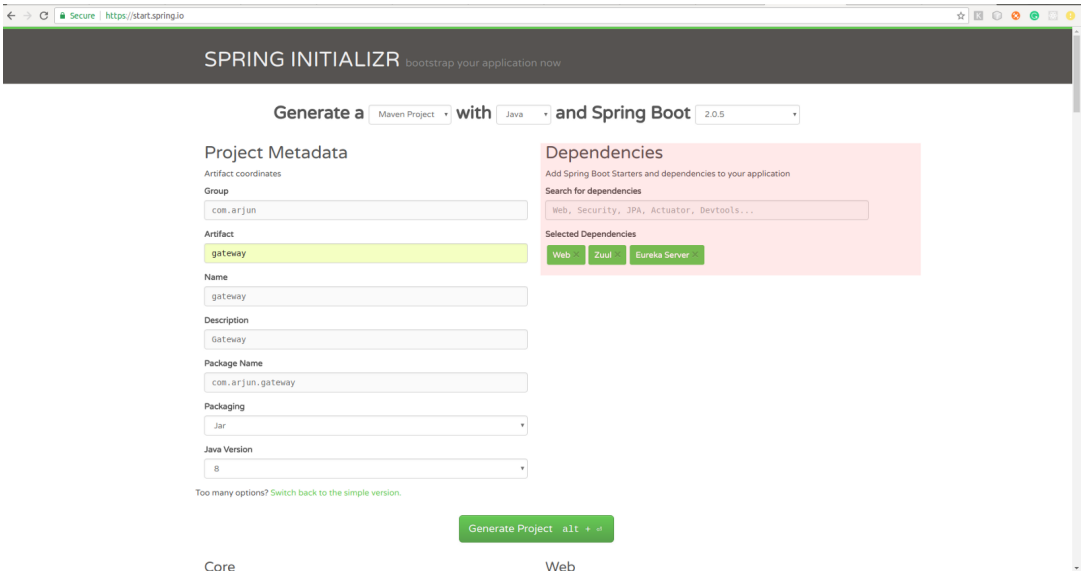
Fourth Step: Now add Spring security dependency in the gateway and check how it is affecting our application.

Fifth Step: Make the changes test the application. Yes, we are done !!!!

Let ‘s start coding.

First Step

Go to https://start.spring.io to create spring initializer. Add ZUUL, Eureka server, web dependency to it.



gateway with ZUUL, Eureka server, web-dependency

Download and unzip the file. open it in the editor. check your pom.xml to check the dependencies.

Now we need to make this gateway microservice as a ZUUL proxy server as well as Eureka server registry. Okay, let's do it.

Go to the main application file and add below annotations

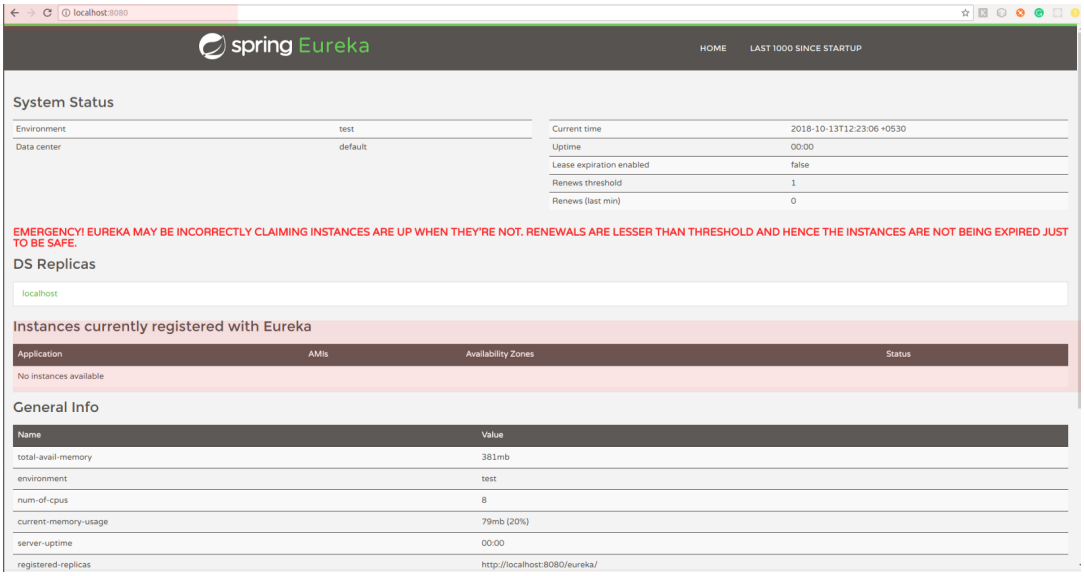
```
@EnableZuulProxy  // act as zuul proxy.
@EnableEurekaServer //for making this application as eureka server.

package com.arjun.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@EnableZuulProxy  // act as zuul proxy.
@EnableEurekaServer //for making this application as eureka server.
@SpringBootApplication
public class GatewayApplication {

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

```
    }
}
```

Fine now we need to add some properties to our application.properties file.

```
#Name of the application
spring.application.name=gateway
#This is a eureka server so no need to register
eureka.client.register-with-eureka=false
#This is a eureka server no need to fetch registry
eureka.client.fetch-registry=false
#Register url for client
eureka.client.service-url.defaultZone=http://localhost:8080/eureka
#timeout
eureka.server.wait-time-in-ms-when-sync-empty=0
#All url come with prefix/api will interpret
zuul.prefix=/api
```

We are done. Just start the application and go to the URL http://localhost:8080 .



As you can see in the picture. No instances are available. Which means service registry is fine and it is looking for the Eureka client to register. But currently, we don't have any Eureka clients. So far so good let's go and create a client

When I am saying Eureka client don't confuse it with front-end client. Eureka clients are microservices. It is a server side thing. But with respect to the gateway (Eureka server), these microservice will act as the client.

We have completed the first step. My commit till this point is below.

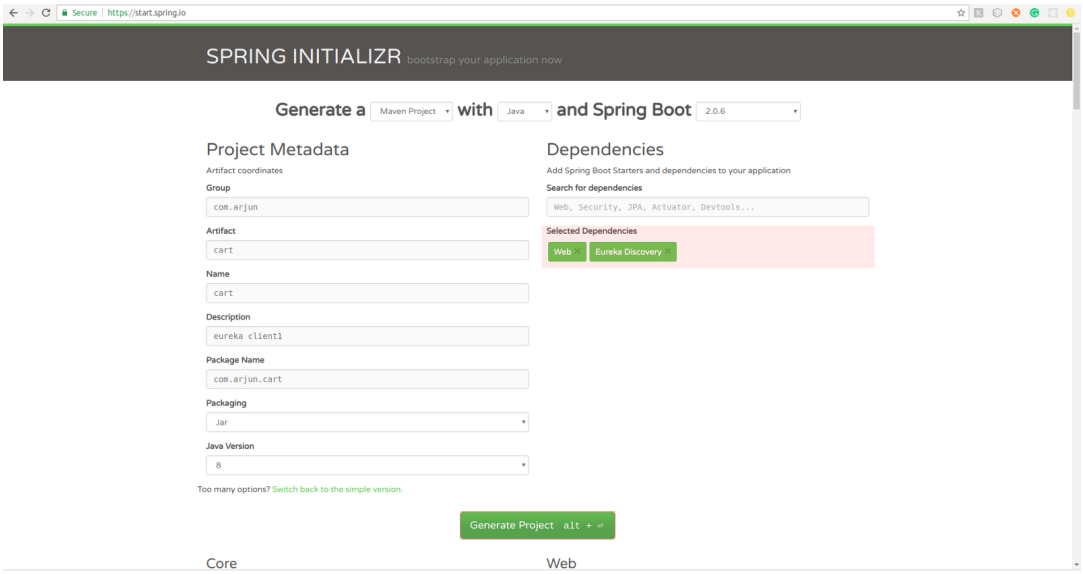
> ***https://github.com/arjunbalussery/gateway/commit/a17f1f0d6567313dbb61769a76ef4ae2898b6c68***

Second Step

Create a microservice with Eureka client dependency and register it to the Eureka server. Make use of ZUUL proxy to redirect the request from the gateway to the client microservice. Excited?? Let's make our hand a little dirty

client microservice. Excited?!!. Let's make our hand a little dirty.

First create a microservice with Eureka client dependency (Eureka Discovery) as below.

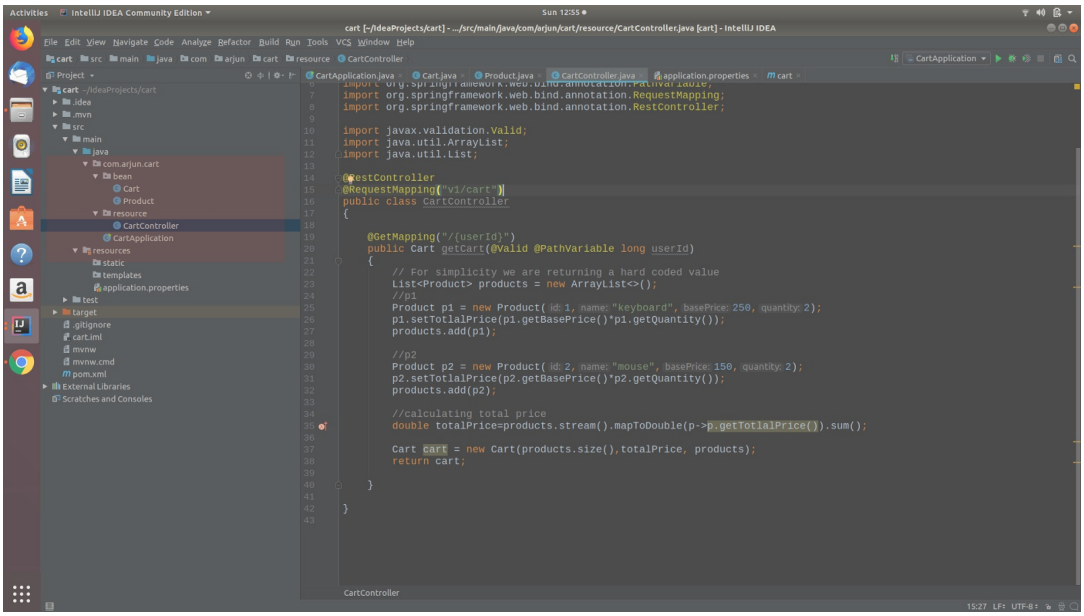


We are going to create a cart microservice which perform all operation related with cart.

For simplicity, I am adding only one operation which retrieves cart details for a particular user. for that

1 Added 2 bean classes product, Cart

2 Added a cartController class



CartController will look like below.

```
@RestController
@RequestMapping("v1/cart")
public class CartController
{

    @GetMapping("/{userId}")
```

```
    public Cart getCart(@Valid @PathVariable long userId)
    {
        // For simplicity we are returning a hard coded value
        List<Product> products = new ArrayList<>();
        //p1
        Product p1 = new Product(1,"keyboard",250,2);
        p1.setTotlalPrice(p1.getBasePrice()*p1.getQuantity());
        products.add(p1);

        //p2
        Product p2 = new Product(2,"mouse",150,2);
        p2.setTotlalPrice(p2.getBasePrice()*p2.getQuantity());
        products.add(p2);

        //calculating total price
        double totalPrice=products.stream().mapToDouble(p->p.getTotlalPrice()).sum();

        Cart cart = new Cart(products.size(),totalPrice, products);
        return cart;

    }

}
```

At this point, you can test this application separately. For that, you need to comment below dependency (Eureka Client) and import the changes. Because we still not added this microservice as a eureka Client.

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

let's test using below url.

> ***Request*** *: GET, url : http://localhost:8080/v1/cart/123*
>
> ***Response*** *: {"totalItems":2,"totalPrice":800.0,"products": [{"id":1,"name":"keyboard","basePrice":250.0,"quantity":2,"totlalPrice":500.0}, {"id":2,"name":"mouse","basePrice":150.0,"quantity":2,"totlalPrice":300.0}]}*

Fine great. Next register this microservice as a eureka client. How do we do that? It is really simple.

1) Uncomment the above dependency and import the changes.

2) Go to the main application file and add @EnableEurekaClient Annotation. Like below.

```
package com.arjun.cart;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@EnableEurekaClient
@SpringBootApplication
public class CartApplication {

    public static void main(String[] args) {
```

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

3) Now we need to consider a couple of things. Eureka server is running on 8080 port. So we need another port for our client application. let's give this 8081 port.

4) Ask cart service (Eureka client)to register to gateway (Eureka server)service

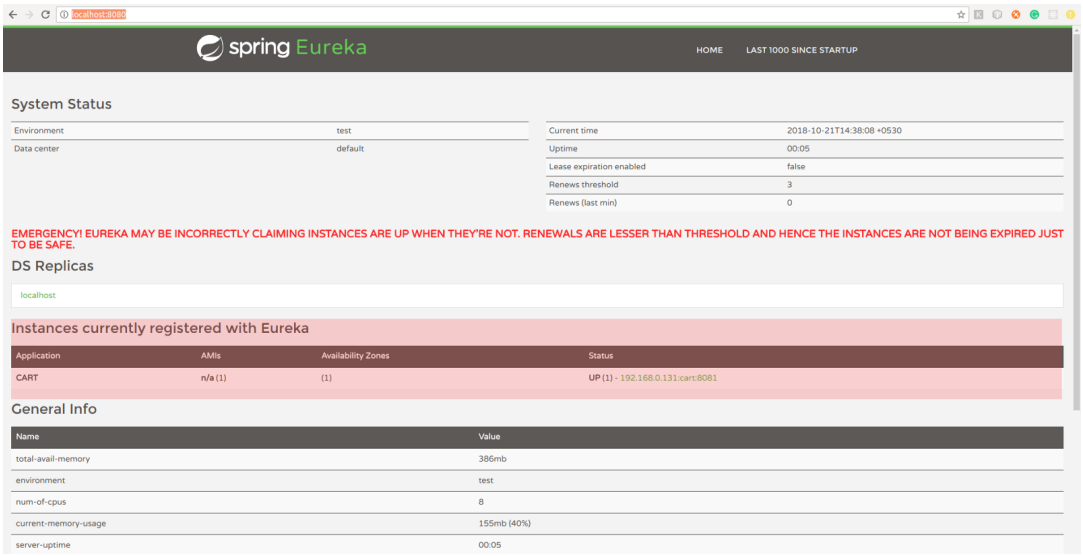
How do we do that? Let's use our application.properties for the above configurations.

```
#Name of the application
spring.application.name=cart
server.port=8081
#Eureka server url for registering
#This is eureka client
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
#register url to server
eureka.client.serviceUrl.defaultZone=http://localhost:8080/eureka/
instance.hostname=localhost
```

Fine. Looks great. let's start testing whether everything works till now.

First start gateway service, then start our cart application. To check whether out client service register to the server goes to the below URL.

http://localhost:8080/ you can see the cart application is registered on gateway service.



All looks fine. My commit for cart service is below.

> ***https://github.com/arjunbalussery/cart/commit/8f7fea38fcfc6c9cd84a4bbcd8933b935aafbe64***

Now we have registered cart service to the gateway. The next problem is how to access cart through gateway service ?. Here ZUUL proxy will help us. So **let's do the changes in the gateway service to accommodate this.** Alter the application.properties file and

add below properties.

```
zuul.routes.middleware.path=/cart/**
zuul.routes.middleware.url=http://localhost:8081/
```

It means whichever URL coming with **/cart/**** redirect it to **http://localhost:8081/**

is that enough? yes, we implemented it.

Third Step: Test the application.

Let's test it. Start gateway, cart services and hit gateway with below URL

http://localhost:8080/api/cart/v1/cart/123

Check you are getting the response which you got earlier.

> ***Request*** *: GET, url : http://localhost:8080/api/cart/v1/cart/123*
>
> ***Response*** *: {"totalItems":2,"totalPrice":800.0,"products": [{"id":1,"name":"keyboard","basePrice":250.0,"quantity":2,"totlalPrice":500.0}, {"id":2,"name":"mouse","basePrice":150.0,"quantity":2,"totlalPrice":300.0}]}*

Let's split the URL to understand much better.

http://localhost:8080/api/ **->** *help to reach the gateway.*

http://localhost:8080/api**/cart/** **->** *This helps ZUUL proxy to redirect the request to cart service by appending the remaining part. ie finally it will be converted to* **http://localhost:8081/v1/cart/123**

All looks fine. My commit for gateway service changes is below.

> ***https://github.com/arjunbalussery/gateway/commit/f0043ffce9148cea12e7945bc78ac571a7e5a39b***

Fourth Step: Add Spring security dependency in the gateway and check how it is affecting our application.

Great. We are going to extend the spring security filter and implements JWT token-based authentication(JSON Web tokens). We will be using MongoDB to store user information, generated JWT tokens. So we need to add a couple of dependencies and import those changes to our application. Let's do it step by step.

Step 1: Add below dependency in pom.xml

1) spring-security

```
<dependency>
  <groupId>org.springframework.security</groupId>
```

```
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
 </dependency>
```

2) spring-data-mongo

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

3) JWT

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

What we are going to implement.

1. Signin request accepts username and password. If username and password are valid it will generate a jwt token and sent back the token in the response header.
2. All other request headers should contain the JWT token which we have generated from signin service, it will be validated. If validation fails an exception will be thrown from filter service.
3. Request with valid JWT token will be processed.
4. Request header should contain → **Authorization : valid-jwt-token** key value pairs

Step 2: Create User bean, Role bean, User service, User repository to store and retrieve user information.

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/bean/auth/User.java
https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/bean/auth/Role.java
https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/security/UserService.java
https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/repository/UserRepository.java

Step 3: Implements Spring UserDetails interface to implement spring security.

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/bean/auth/MongoUserDetails.java

Step 4: Create a security package and add Jwt token filter, jwt token configuration, jwt token provider and a websecurityConfig

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/security/JwtTokenFilter.java

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/security/JwtTokenFilterConfigurer.java

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/security/JwtTokenProvider.java

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/security/WebSecurityConfig.java

Step 5: Create a signin, signout mechanism

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/controller/LoginController.java

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/service/serviceimpl/LoginService.java

https://github.com/arjunbalussery/gateway/blob/master/src/main/java/com/arjun/gateway/repository/JwtTokenRepository.java

Step 6: Add MongoDB Configurations in the application.properties

```
#spring data mongo
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=root
spring.data.mongodb.password=root
spring.data.mongodb.database=test
spring.data.mongodb.port=27017
spring.data.mongodb.host=localhost
```

I am not going too much deeper. You can search in the medium for JWT token implementation. If You really want me to explain deeper, please put in the comment section will write an article on the same.

Super we have added spring security using JWT token. Everything nice and well. let's go for testing.

Fifth Step: Test our application.

Let’s test it. Start gateway, cart services and hit gateway with below URL

http://localhost:8080 .

Do you able to see our cart service registered on the gateway service?. Yes it is registered.

That's fine. But let me ask you one thing. We have mentioned earlier all request except signin request should contain JWT token in their request header. But here we haven't sent any headers in the **http://localhost:8080** request. Then how it worked?

For registering to gateway our cart service should request with below URL **http://localhost:8080/eureka/** here also we are not sending any headers. Still, it worked. Does our security has any problem? What do you think?

No, our security don't have any problem. Go and check in the WebSecurityConfig.java. There you can see below code. This help us to achieve this.

```
@Override
public void configure(WebSecurity web) throws Exception {
    // Allow eureka client to be accessed without authentication
    web.ignoring().antMatchers("/*/")//
            .antMatchers("/eureka/**")//
            .antMatchers(HttpMethod.OPTIONS, "/**"); // Request type options should be allowed.
}
```

1. Call signin service (POST) with below body
   username:username
   password:password
2. copy the jwt token from the response header.
3. Call cart service with the request header **Authorization: jwtTokenrecieved**
4. Check we are getting the proper response from cart service.

Congratulations you have done it !!!

Ask all your suggestions, doubts in the comment section. HAPPY CODING !!!

Github

1. gateway
   https://github.com/arjunbalussery/gateway
2. cart
   https://github.com/arjunbalussery/cart

Java Spring Boot Jwt Zuul

91
claps

8


arjun nagathankandy
Senior Java Developer

Follow