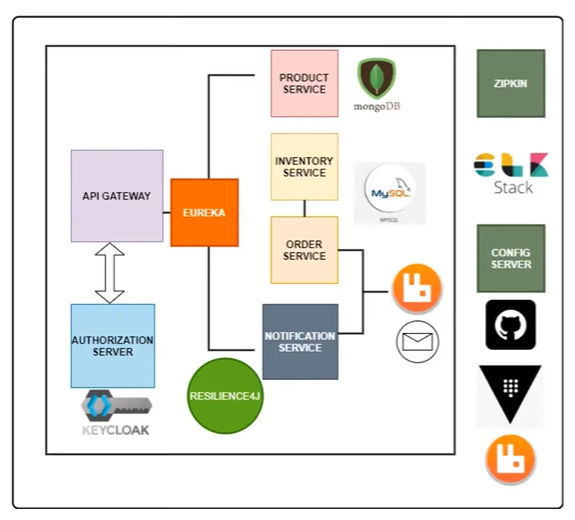
Online Shopping



Zipkin – Distributed Tracing

ELK – Logging

Resilience4J –

Ports-

Product-service : 9001

Lombok, Spring Web, Spring Data Mongo DB

1. Product Service Class – in Model Package

* Uses Mongo DB – configure that in Application.properties
* Model > Product – Add Lombok properties for Data/Builder/NoArgs/AllArgs constructor
* @Document(value="product") – MongoDB document linkage

1. ProductRepository interface – extends MongoDBRepository
2. ProductController –

@RequiredArgsConstructor – Creates argument constructor for the class

**private** **final** ProductRepository productRepository;

//Why is it not autowired

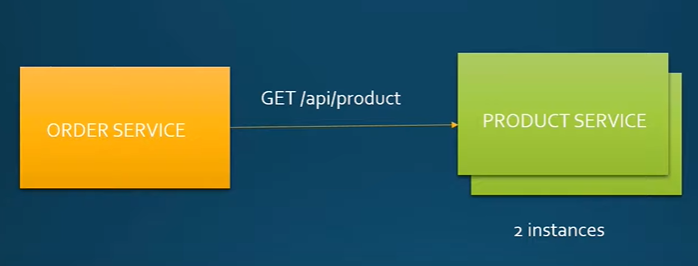
npx kill-port 8080 – to kill what’s running on 8080

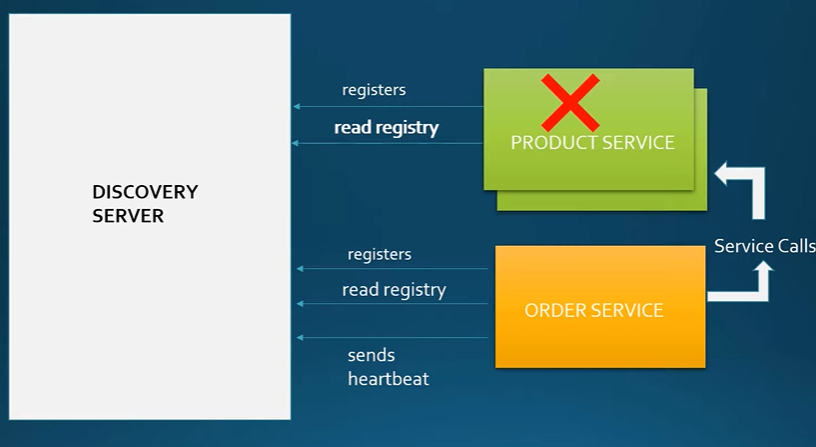
Order-Service : 9002

Lombok, SpringWeb, Spring Data JPA, MySQL Driver

create database `order-service`; - use backticks

Service Discovery : Automatic detection of all available services.



Service Registry – Process of services registering to discovery server

Services keep a local cache of registry. Do some client side load balancing in case of multiple instances

If no heartbeat – client is down

**Eureka-Service**

Eureka CServer Dependency

@EnableEurekaServer

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

Local copy of registry not needed

@EnableEurekaClient – For the clients

spring.application.name=order-service

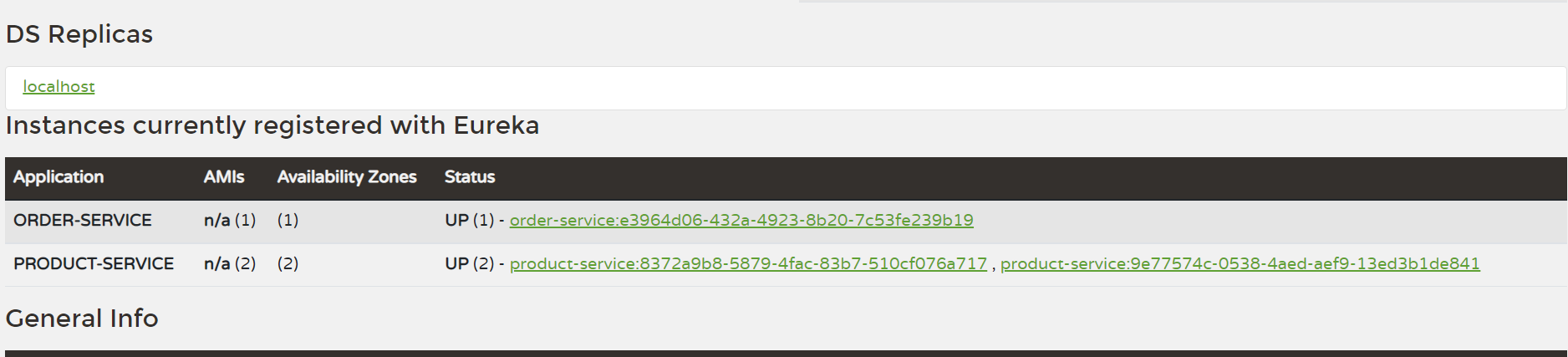
Eureka Client Dependency

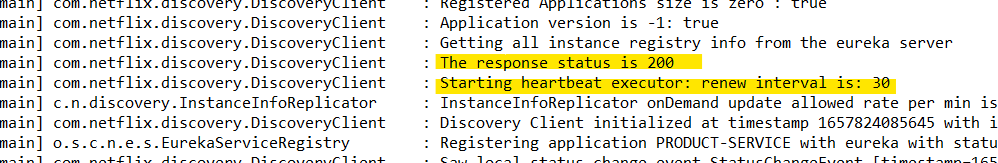
For Product Service:

server.port=0 – To assign a port number dynamically

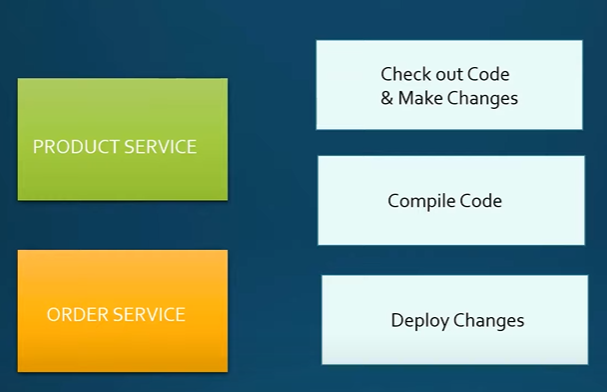
-We would be creating multiple instances for this

eureka.instance.instance-id=${spring.application.name}:${random.uuid}

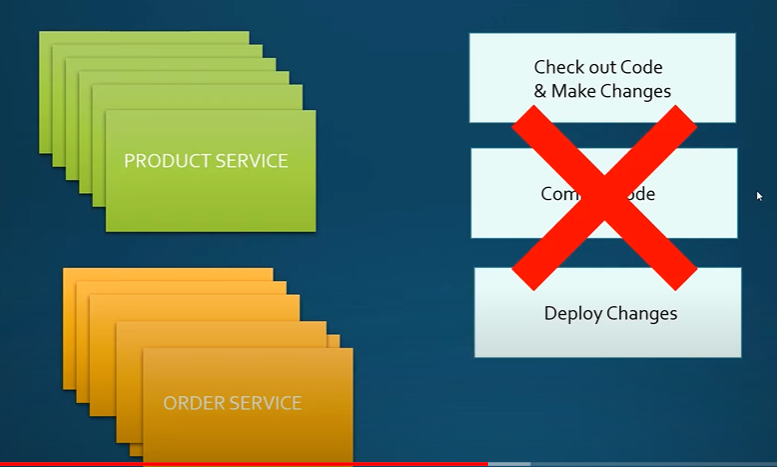




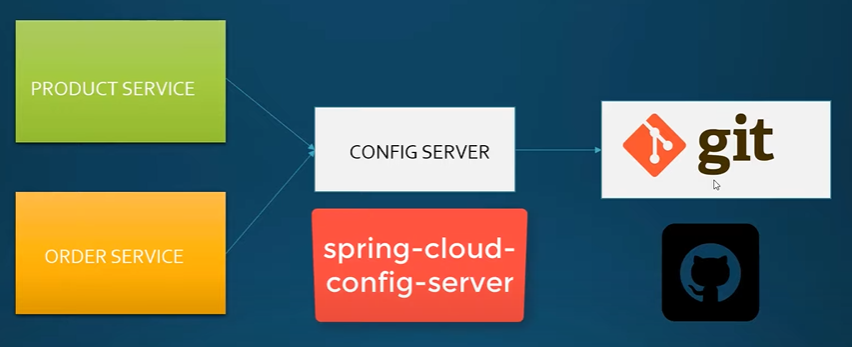
**Centralized Configuration**

****

**Config changes difficult for multiple instances**

****

**Dynamically load changes instead of restarting the instances.**

****

**Can store config in git repo and native files**

@EnableConfigServer

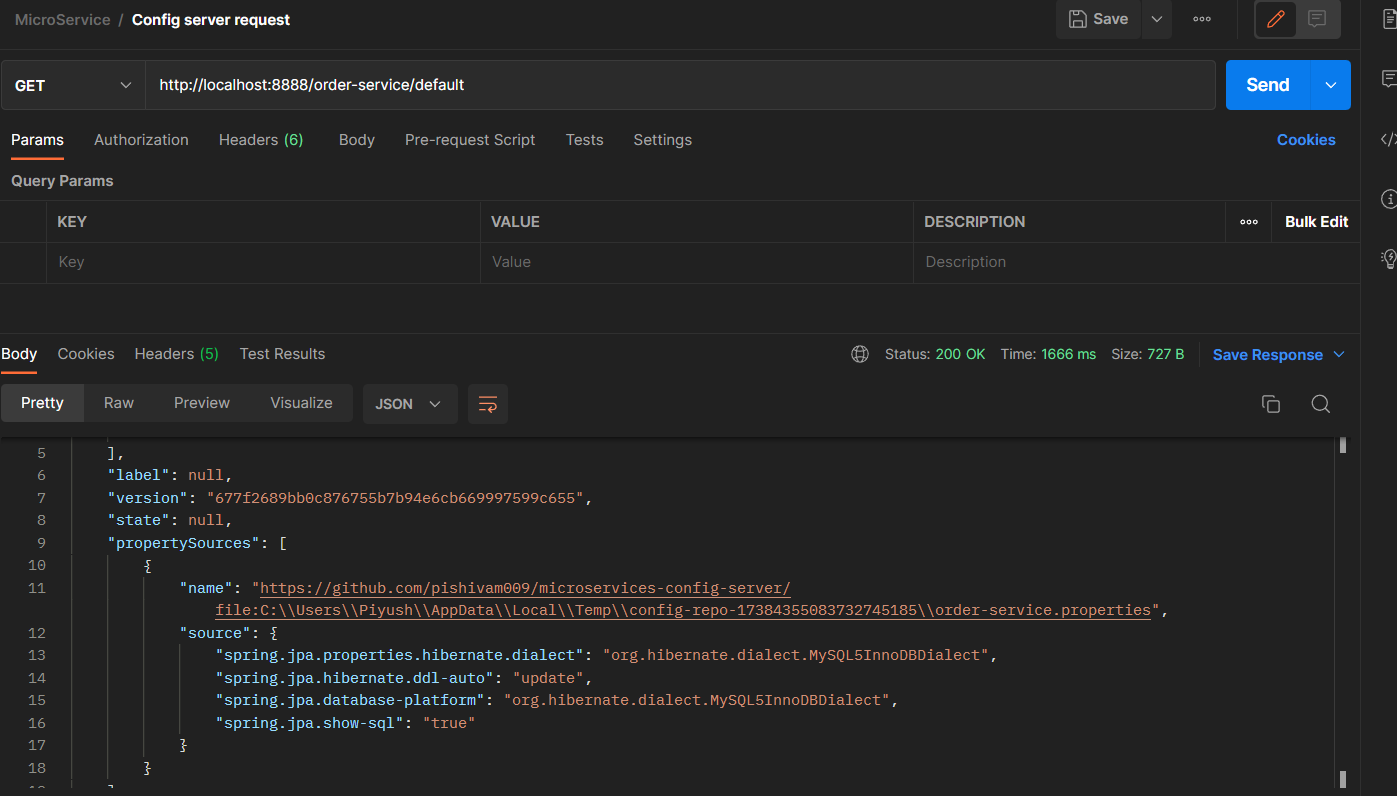
SpringCloud ConfigServer , Spring Actuator

Properties moved to new git repo

spring.cloud.config.server.git.uri=https://github.com/pishivam009/microservices-config-server

spring.cloud.config.server.git.clone-on-start=true

spring.cloud.config.server.git.default-label=master



Order-service -> Name of the properties file

Default- profile name

Prod>order-service.properties

Access > order-service/prod

Add Config-client dependency to Order Service and Product Service

Cloud-starter-bootstrap – legacy

New: <https://docs.spring.io/spring-cloud-config/docs/3.0.0/reference/html/#config-data-import>

Refresh Config

@RefreshScope - Beans annotated this way can be refreshed at runtime and any components that are using them will get a new instance on the next method call, fully initialized and injected with all dependencies.

Change test.name in product-service

Call /actuator/refresh

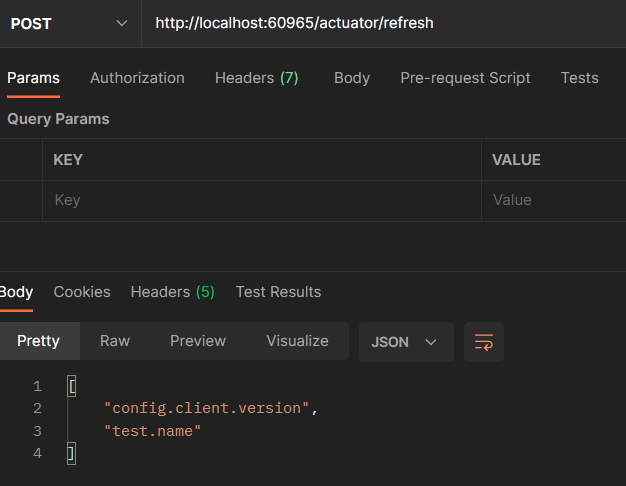
Add actuator dependency to product/Order service. That is needed to trigger config refresh

<https://stackoverflow.com/questions/61139281/spring-cloud-config-client-not-picking-values-from-config-server>

One Extra Space!!! – Stupid

Add management.endpoints.web.exposure.include=\* in product/order service for refresh

Test.name value is refreshed.



Store secrets in vault.

choco install vault

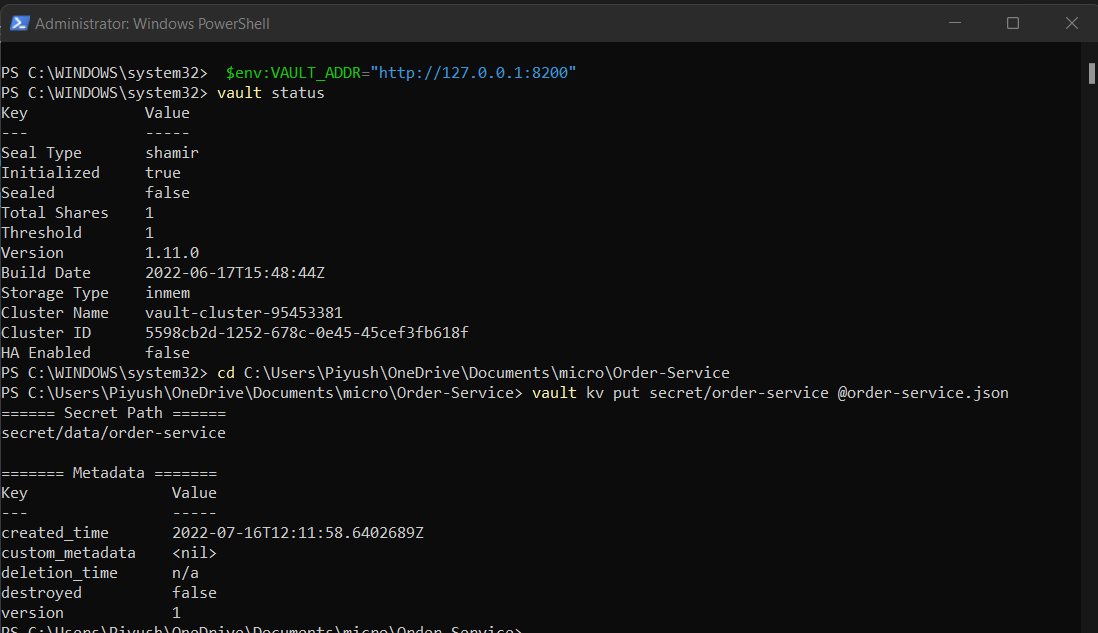
vault server –dev

set VAULT\_ADDR=http://127.0.0.1:8200

vault kv put secret/order-service @order-service.json

Having issues. Omitting for now. Vault can be integrated along with config server too.

https://www.youtube.com/watch?v=p65u4t26BBc



vault kv get secret/order-service

Same for product-service

Add spring cloud vault dependency

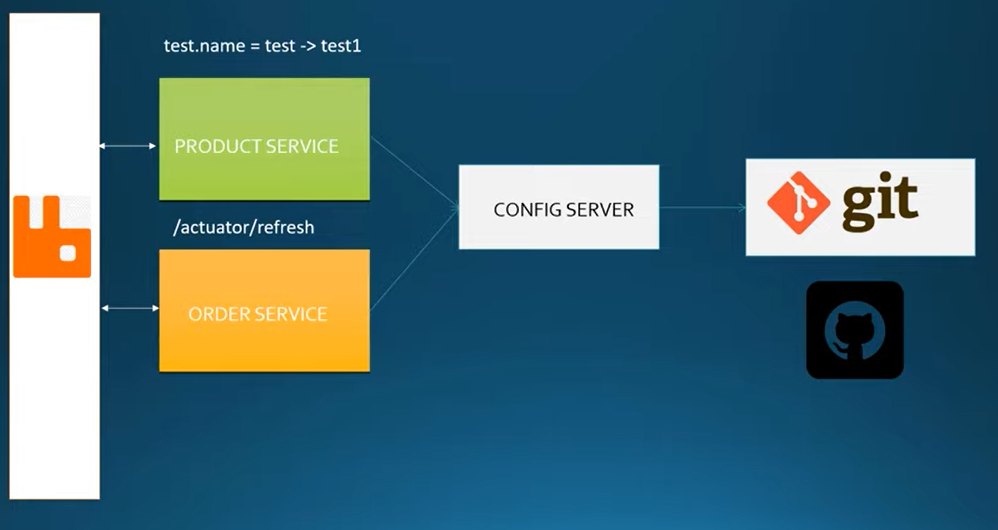
* Not working properly. Removing for now. Read further on it



@RefreshScope needs actuator/scope to be called.

Instead, use MsgBroker – RabbitMQ or Kafka – to broadcast changes to all these microservices

Subscribe to msg broker



Add spring cloud bus dependency

spring-cloud-starter-bus-amqp

Download RabbitMQ – used as message Broker

Call actuator/busrefresh – POST

Broadcast the changes. Other services will be made aware of the changes.

API Gateway

Make request to API gateway and then it gets routed

Also takes care of authentication, monitoring, rate limiting.

Gateway, Eureka Client

spring.cloud.gateway.discovery.locator.enabled=true

Similar to enable eureka client – but specific to Gateway

spring.cloud.gateway.routes[0].uri=http://localhost:port

or

spring.cloud.gateway.routes[0].uri=http://lb:PRODUCT\_SERVICE (Picks from EUREKA Service)

To use in small letters

spring.cloud.gateway.routes[0].predicates[0]=Path=/api/product

when this api is called, redirected to Product Service

spring.cloud.loadbalancer.ribbon.enabled – Netfilx Ribbon is deprecated.

Netflix Ribbon

Netflix Ribbon is a Part of Netflix Open Source Software (Netflix OSS). It is a cloud library that provides the client-side load balancing. It automatically interacts with Netflix Service Discovery (Eureka) because it is a member of the Netflix family. The Ribbon mainly provides client-side load balancing algorithms.

It starts Netty Server, not tomcat

Add eureka.instance.hostname=localhost – Otherwise it will lead to discovery issues as it picks hostname from os like ACER.msc.net and then cannot find it.

2022-07-23 00:50:39.854 ERROR 17768 --- [ctor-http-nio-3] a.w.r.e.AbstractErrorWebExceptionHandler : [1ad93ec8-2] 500 Server Error for HTTP GET "/api/product"

java.net.UnknownHostException: Failed to resolve 'PIYUSH-ACER.mshome.net' after 4 queries

at io.netty.resolver.dns.DnsResolveContext.finishResolve(DnsResolveContext.java:1047) ~[netty-resolver-dns-4.1.78.Final.jar:4.1.78.Final]

Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException:

Error has been observed at the following site(s):

\*\_\_checkpoint ⇢ org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]

\*\_\_checkpoint ⇢ HTTP GET "/api/product" [ExceptionHandlingWebHandler]

Original Stack Trace:

at io.netty.resolver.dns.DnsResolveContext.finishResolve(DnsResolveContext.java:1047) ~[netty-resolver-dns-4.1.78.Final.jar:4.1.78.Final]

at io.netty.resolver.dns.DnsResolveContext.tryToFinishResolve(DnsResolveContext.java:1000) ~[netty-resolver

Install Keycloak

Identity and access management server

Using keycloak 16 –standalone.bat –Djboss.http.port=8180

Install Java 11 too

Create Realm – A container where you can store user/client/login info

Add Client ID – spring-cloud-gateway-client

Keeping access Type confidential- as we want authentication to be done between applications.

If want it via Angular, or front end – choose public

Switch off Direct Access Grant- tries to authenticate using resource owner password grant, Standard flow – tries to authenticate using browser.

Get secret token

Add Test User, configure password

Click on realm. Go to OpenIDEndPoint configuration

Add Spring Boot Starter Security, OAuth2 resource Server – to verify access tokens incoming from KeyCloak

spring.security.resourceserver.jwt.issuer-uri=http://localhost:8180/auth/realms/microservices-realm

URI details available in <http://localhost:8180/auth/realms/microservices-realm/.well-known/openid-configuration>

Define rpod service as resource servers. All should be resource servers only

In API-gateway…add resourceserver configs

@Configuration

@EnableWebFluxSecurity

**public** **class** SecurityConfig {

@Bean

**public** SecurityWebFilterChain springSecurityWebFilterChain(ServerHttpSecurity serverHttpSecurity) {

//Lambda of type AuthoriseExchaneSpec. Make sure all exchange requests are authenticated

//Enable resource server capabilities

//Using jwt as token

serverHttpSecurity.authorizeExchange(exchange -> exchange.anyExchange().authenticated())

.oauth2ResourceServer(ServerHttpSecurity.OAuth2ResourceServerSpec::jwt);

//Disable CSRF as REST API used

serverHttpSecurity.csrf().disable();

**return** serverHttpSecurity.build();

}

spring.security.oauth2.resourceserver.jwt.issuer-uri=http://localhost:8180/auth/realms/microservices-realm In Product-service too

SecurityConfig inside product-service

Get oauth token in Postman. Use it in header to make API request