**Microservices**

-If I want to exclude the default tomcat server and use the jetty server how I can do that by going to the pom.xml and adding an exclusion   
<depandancy>

<groupid></groupid>

<artifactid><artifactid>

<exclusions>

<exclusion>

<groupid>org.springframework.boot</groupid>

<artifactid>spring.strater.web.tomcat<artifactid>

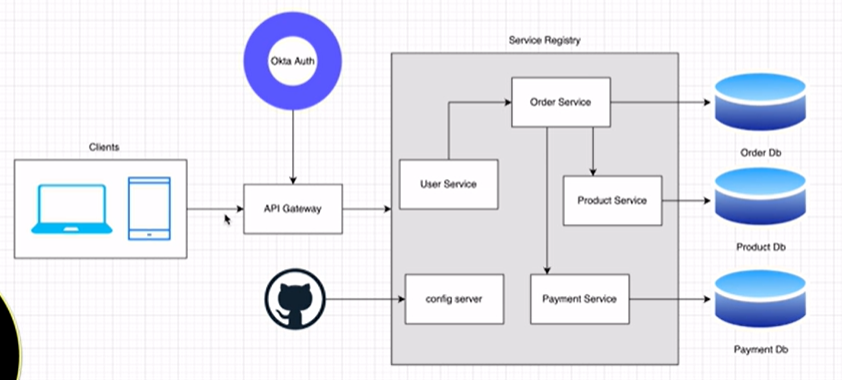
</exclusion>

</exclusions>  
</dependency>

* Actuator will give us endpoint to monitor our application
* When you have a requirement to have a mandatory field go for @PathVariable

If fields are not mandatory go with @Requestpram()

In request, param is used when you have to provide value after the question mark and we can make them not required by required filed false and defaultValue as =””

Microservice   


* **How to register any client with eureka server by**

eureka:

instance:

prefer-ip-address: true

client:

fetch-registry: true

register-with-eureka: true

service-url:

defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

CONFIG SERVER

* When you want to centralize all common configuration we will use config server so create new config Server microservice and add two dependency config server and eureka client dependency.
* Then in main class of this config server annotate with enable config server
* In application.yaml

server:

port: 9296

spring:

application:

name: CONFIG\_SERVER

cloud:

config:

server:

git:

uri: https://github.com/shubhamthakur614/product-app-config

default-label: master //if your git has main our master check

clone-on-start: true

* And create repository in GitHub and add file and above common configuration to that file and hide all this data from each microservice except the config-server
* To add this common configuration to each file we need to add dependency to each microservice will be “CONFIG\_CLIENT”.
* And then in each application.yaml file add this property

Spring:

config:

import: configserver:http://localhost:9296

**Open Feign Client Communication**

* First add the open feign client dependency and annotate the main class with @EnableFeignclient
* After that in @EnableFeignClient(name=”PRODUCT\_SERVICE/product”)

And copy past the method that you want to call only method not body

And put in this interface

* After in serviceImpl create Autowiring of this interface

Then we can check the method by calling

* Suppose if we get any exception as we are not handling the product exception in order so we have to create one pkg in external as decoder->customErrorDecoder to decode error or exception in frontend as well
* So create customErrorDecoder which implements ErrorDecoder interface
* It has one decode method that we have to override so what ever exception feign client get we can get and show

**Distributed log Tracing**

* Install zipkin server to get all logs through docker
* Open docker desktop and use command
* Docker run -d -p 9411:9411 openzipkin/zipkin
* Before we can use zipking client and sleuth dependency to perform distributed log tracing but now in spring 3 this are deprecated
* So we can use zipking dependency which consist of Actuator,Micrometer-tracing-bridge, zipkin-reporter-brave

Add this to all the application.yaml file  
management:

tracing:

sampling:

probability: 1.0

**To Implement API GATEWAY IN Application**

* Need this dependency and create new project
* Lombook, cloud bootstrap, eureka client, Zipkin, actuator, gateway, spring reactive web, config client

management:

endpoints:

web:

exposure:

include: '\*'

server:

port: 9090

spring:

application:

name: API-GATEWAY

config:

import: configserver:http://localhost:9296

cloud:

gateway:

mvc:

routes:

- id: ORDER-SERVICE

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

- id: PRODUCT-SERVICE

uri: lb://PRODUCT-SERVICE

predicates:

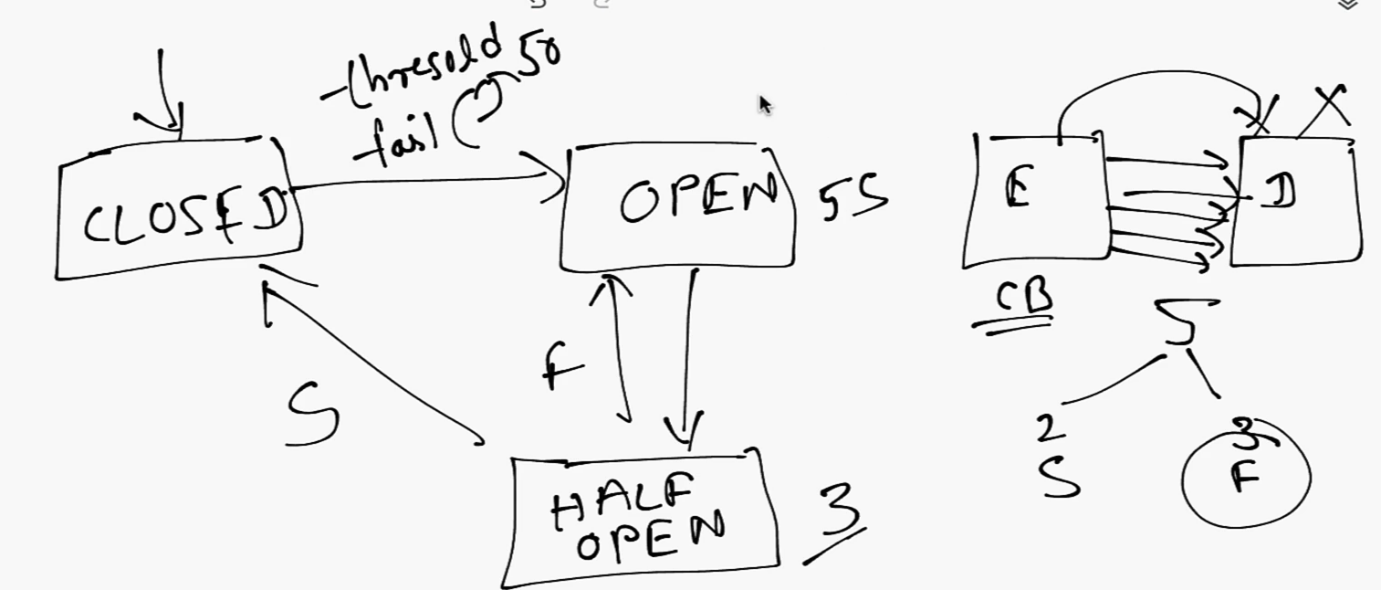
- Path=/product/\*\*

**CIRCUIT BREAKER**

**1)fallback method**

**2)Circuit Breaker**

**3)Retry Mechanisms**

**4)Rate Limiter**

* Resilience 4j gives us ability to implement circuit breaker
* In that main class add this bean

@Bean

**public** Customizer<Resilience4JCircuitBreakerFactory> defaultCustomizer() {

**return** factory -> factory.configureDefault(id -> **new** Resilience4JConfigBuilder(id)

.circuitBreakerConfig(CircuitBreakerConfig.*ofDefaults*()).build());

}

IN APPLICATION.YaML

cloud:

gateway:

mvc:

routes:

- id: ORDER-SERVICE

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

filters:

- name: CircuitBreaker

args:

name: ORDER-SERVICE

fallbackuri: forward:/orderServiceFallback

Security(AUTH2 (OKTA))

* In previous in monolithic applications we will do session-based a

Authentication. For that particular user session will be created within that session interval users are allowed to do the operation.

* In microservices architecture creating session for each service is not possible that why we are using OAUTH2 where it will give authentication through authentication code.
* How it will work we have one authentication server when we send a request to server this request can contatin credential so authentication server will create authentication code which is nothing but jwt token.This jwt token consist of 3 things header,body,signature.This jwt token which will be created will be unique.
* This authcode created contain an a code which contatin code validadity and code that we have to pass.
* Here we are using Okta its 3rd party authentication tool