CONFIGURATION AND EXECUTION STEPS

JAVA:

1. JAVA\_HOME- C:\Program Files\Java\jdk1.8.0\_181

2. Path - C:\Program Files\Java\jdk1.8.0\_181\bin

MAVEN:

1. M2\_HOME - C:\Softwares\apache-maven-3.6.3-bin\apache-maven-3.6.3

2. Path - C:\Softwares\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin

SPRING BOOT AND MICROSERVICES:

PROJECT CREATION AND IMPORT:

1. Go to Spring Initializer, Spring boot project can be maven project(pom.xml) or gradle project(build.gradle)
2. Extract project into workspace
3. Import the project in eclipse
4. In command prompt, we go into the project location

>mvn clean install

- it will download all dependency and put in .m2

1. Right click project - Maven - Update project - Check Force Update on snapshot Release - click ok
2. If still errors occur, go to Problems tab - delete all errors
3. In Spring Boot, we always have to run main program

POSSIBLE DEPENDENCIES:

Spring Web – Provides the embedded servers and used to create Restful applications with Spring MVC.

Spring Dev Tools – To restart the application and reload the browser automatically.

Spring Data JPA – Java Persistence Unit – Simple to write the queries to the database

MySQL Driver – Used to interact with MySQL Database.

Validation – To validate the inputs

Eureka Server – Will create a service registry

Eureka Discovery Client – Allow the services to be registered in the service registry

Gateway -

Open Feign – To interact with external services

Spring Cloud Load Balancer – Load Balancing

Resilience 4J – Circuit Breaker

JSP:

As the Spring Boot doesn’t know about Jsp, we have to let it know through dependency.

For that, we have to include following dependencies in pom.xml

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

We have to include the following lines in application.properties file.

spring.mvc.view.prefix=/WEB-INF/view/

spring.mvc.view.suffix=.jsp

MySQL:

Have to include the following lines in application.properties file

spring.datasource.url=jdbc:mysql://localhost:3306/database\_name

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.show-sql = true

spring.jpa.hibernate.ddl-auto = update

#dialect will generate the query based on particular db

spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect

EUREKA SERVER:

While creating the project, we have to include **Eureka Server** dependency.

In the main class, we have to place @EnableEurekaServer above the class name.

We have to include the following lines in application.properties file

server.port=8761

eureka.client.register-with-eureka=false //to prevent eureka server to register with itself.

eureka.client.fetch-registry=false //to prevent the registry to fetch the details of itself.

EUREKA CLIENT:

While creating the project, we have to include **Eureka Discovery Client**  dependency.

In the main class, we have to place @EnableEurekaClient above the class name.

We have to include the following lines in application.properties file

server.port=8081

spring.application.name=order-service

eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

OPEN FEIGN:

While creating the project, we have to include **Open Feign** dependency.

We have to do the following steps in the project to which external service is needed.

In the main class, we have to place @EnableFeignClients above the class name.

For each client we need to interact with, have to create an interface.

Above the interface, we need to place @FeignClient and have to include the required method signatures.

Ex:

@FeignClient(name = "http://ORDER-SERVICE/orders")

**public** **interface** OrderClient

{

@GetMapping("/port")

**public** String getInfo();

}

LOAD BALANCER – CLIENT SIDE:

While creating the project, we have to include **Cloud LoadBalancer** dependency.

We have to make sure that we followed Eureka server, Eureka client and feign client steps.

We have to instantiate multiple instances of required external service we need to access.

To do above step, we have to run external service spring boot application through command prompt.

Command to run the SBA:

mvn clean install //it will access test cases tooo

mvn clean install –DskipTests //to skip test cases

mvn spring-boot:run -Dspring-boot.run.arguments=--server.port=8010

So, one of the multiple instances of external service will be accessed randomly.

CIRCUIT BREAKER – RESILIENCE 4J:

While creating the project, we have to include **Resilience4J** dependency.

We have to make sure that we followed Eureka server, Eureka client and feign client steps.

We have to include the following lines in application.properties file.

resilience4j.circuitbreaker.configs.default.register-health-indicator=true

resilience4j.circuitbreaker.configs.default.sliding-window-size=10

resilience4j.circuitbreaker.configs.default.minimum-number-of-calls=5

resilience4j.circuitbreaker.configs.default.permitted-number-of-calls-in-half-open-state=3

resilience4j.circuitbreaker.configs.default.automatic-transition-from-open-to-half-open-enabled=true

resilience4j.circuitbreaker.configs.default.wait-duration-in-open-state=5

resilience4j.circuitbreaker.configs.default.failure-rate-threshold=50

resilience4j.circuitbreaker.configs.default.event-consumer-buffer-size=10

resilience4j.circuitbreaker.configs.default.record-exceptions=org.springframework.web.client.HttpServerErrorException,java.util.concurrent.TimeoutException,java.io.IOException

resilience4j.circuitbreaker.instances.mainService.base-config=default

We have to follow below steps to use circuit breaker.

--- We have to autowire CircuitBreakerFactory.

--- Using that autowired object, we have to create circuit breaker using create() method.

--- To run the circuit breaker, we have to pass 2 arguments.

1. The service needs to be called.
2. The fall back method needs to be invoked in case the called service is not available (Throwable).

--- The fall back method needs to be defined.

Ex:

@Service

**public** **class** UserService

{

@Autowired

CircuitBreakerFactory circuitBreakerFactory;

@Autowired

OrderClient orderClient;

**public** String getInfo()

{

CircuitBreaker circuitBreaker = circuitBreakerFactory.create("circuitbreaker");

**return** circuitBreaker.run(() -> orderClient.getInfo(), throwable -> getDefaultInfo());

}

**public** String getDefaultInfo()

{

System.***out***.println("From fallback ======================>>");

**return** "Order-service is down, Please try after some time.";

}

}