Course 3 – Sprint 7: Register Microservices on a Netflix Eureka Discovery Server

We have already explored what is microservices and the use of Spring Cloud. Here we are looking at a high-level perspective of microservices, where we say that there are many small and independent applications running which have their own URLs to identify them and ports.

In such scenarios, we need to maintain and monitor them properly. We should also ensure that they are run synchronously. We make use of Eureka Server to resolve this problem.

What Is Netflix Eureka?

Eureka is one of those projects that is the most used for service registry & discovery in Microservices. It consists of the Eureka Server and Eureka clients. It's also a microservice to which all the other microservices registers. Eureka Clients are nothing but the independent microservices.

What is Service Registry in Microservices?

- Service Registry is the process of registering a microservice with Eureka Server.
- It serves like a registry or databases, which stores all info about the microservices involved in the application.
- Spring Cloud's annotation called **@EnableEurekaServer**, to stand up a registry with which other applications can communicate.

What is Service Discovery in Microservices?

Now that you have started a service registry, you can stand up a client that both registers itself with the registry and uses the Spring Cloud DiscoveryClient abstraction to interrogate the registry for its own host and port. The @EnableDiscoveryClient activates the Netflix Eureka DiscoveryClient implementation. (There are other implementations for other service registries, such as Hashicorp's Consul or Apache Zookeeper).

Demo:

Step 1: To **start from scratch**, move on to start with Spring Initializr.

Step 2: To **skip the basics**, do the following:

- Download and unzip the source repository for this guide, or clone it using Git:

  - **Git clone https://github.com/spring-guides/gs-service-registration-and-discovery.git**
- cd into gs-service-registration-and-discovery/initial.
- Jump ahead to Start a Eureka Service Registry.
- When you finish, you can check your results against the code in gs-service-registration-and-discovery/complete.

We need to add below dependency in pom.xml

```
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2020.0.0</spring-cloud.version>
</properties>
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
<dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
</dependency>
<build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
    <repositories>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
        </repository>
    </repositories>
```

Step 3: In Application.properties file,

We need to update the properties file for this project to indicate that it is a discovery server and not a client.

```
eureka.instance.hostname=localhost
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
server.port=8000
spring.application.name=DiscoveryServer
```

Step 4:

a) Start a Eureka Service Registry

```
package com.example.serviceregistrationanddiscoveryservice;

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

@EnableEurekaServer
@SpringBootApplication
public class ServiceRegistrationAndDiscoveryServiceApplication {
    public static void main(String[] args) {
SpringApplication.run(ServiceRegistrationAndDiscoveryServiceApplication.class, args);
    }
}
```

When the registry starts, it will complain (with a stack trace) that there are no replica nodes to which the registry can connect. In a production environment, you will want more than one instance of the registry. For our simple purposes, however, it suffices to disable the relevant logging.

By default, the registry also tries to register itself, so you need to disable that behavior as well.

It is a good convention to put this registry on a separate port when using it locally.
Add some properties to eureka service/src/main/resources /application.properties to handle all of these requirements, as the following listing shows:

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

b) Discover the services

```
package com.example.serviceregistrationanddiscoveryclient;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class ServiceRegistrationAndDiscoveryClientApplication {

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

```
@RestController
class ServiceInstanceRestController {

 @Autowired
 private DiscoveryClient discoveryClient;

 @RequestMapping("/service-instances/{applicationName}")
 public List<ServiceInstance> serviceInstancesByApplicationName(
         @PathVariable String applicationName) {
     return this.discoveryClient.getInstances(applicationName);
 }
}
```

Whatever implementation you choose, you should soon see eureka-client registered under whatever name you specify in the spring.application.name property. This property is used a lot in Spring Cloud, often in the earliest phases of a service's configuration.

The eureka-client defines a Spring MVC REST endpoint (ServiceInstanceRestController) that returns an enumeration of all the registered ServiceInstance instances at http://localhost:8080/service-instances/a-bootiful-client. See the Building a RESTful Web Service guide to learn more about building REST services with Spring MVC and Spring Boot.

Test the Application

Test the end-to-end result by starting the eureka-service first and then, once that has loaded, starting the eureka-client.

To run the Eureka service with Maven, run the following command in a terminal window (in the /complete directory):

```
./mvnw spring-boot:run -pl eureka-service
```

To run the Eureka client with Maven, run the following command in a terminal window (in the /complete directory):

```
./mvnw spring-boot:run -pl eureka-clientCOPY
```

The eureka-client will take about a minute to register itself in the registry and to refresh its own list of registered instances from the registry. Visit the eureka-client in the browser, at http://localhost:8080/service-instances/a-bootiful-client. There, you should see the ServiceInstance for the eureka-client reflected in the response. If you see an empty <List> element, wait a bit and refresh the page.

Pls refer to this website for sample demo

https://javatechonline.com/how-to-register-discover-microservices-using-netflix-eureka/