Arquitetura de MicroServices com Spring Cloud e Spring Boot — Parte 3

João Rafael Campos da Silva Follow
Jul 22, 2017 · 5 min read

Sumário

Subindo um Eureka Server e conectando-o ao Config Server

Olá pessoal, hoje vamos criar nosso Service Registry, que será responsável por gerenciar o status e a localização dos Microservices em nossa rede. Para isso usaremos o Eureka, uma ferramenta desenvolvida pela Netflix e disponibilizada para a comunidade.

O Eureka é um serviço REST (Representational State Transfer) que é usado principalmente na nuvem AWS (Amazon Web Services) para localizar serviços com o objetivo de balanceamento de carga e failover de servidores.

O conjunto de ferramentas Spring Cloud possui uma implementação do Eureka para que possamos subir um Eureka Server utilizando uma

aplicação Spring Boot.

Gerando a aplicação no Spring Initializr

Acesse o site do Spring Initializr e preencha as configurações como na imagem. Escolhemos qual vai ser o gerenciador de dependências *(Maven)*, a linguagem de programação *(Java)*, as dependências necessárias para o projeto *(Web, Actuator, Config Client e Eureka Server)* e configuramos os metadados do Maven. Por fim geramos o projeto clicando em *Generate Project, Ctrl + Enter ou Command + Enter*. *(**Atenção para a versão do Spring, estamos usando a 1.5 nesse projeto**)*

Você se lembra da pasta delivery, que criamos para colocar o nosso config server? Então, é lá que iremos descompactar o nosso Eureka Server.

Implementando a Aplicação

Como na parte 2 dessa série de stories já entramos em detalhes sobre os arquivos que são gerados e os que são desnecessários não falaremos mais disso para que essa story fique mais concentrada no conteúdo proposto.

Vamos direto ao ponto, observando o `pom.xml` e explicando as dependências inéditas.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
        <modelVersion>4.0.0</modelVersion>

        <groupId>com.coderef</groupId>
        <artifactId>delivery-eureka-server</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>delivery-eureka-server</name>
        <description>Eureka Server</description>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.5.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
                <spring-cloud.version>Dalston.SR1</spring-cloud.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-eureka-server</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-config</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <dependencyManagement>
                <dependencies>
                        <dependency>
                                <groupId>org.springframework.cloud</groupId>
```

```
                        <artifactId>spring-cloud-dependencies</artifactId>
                        <version>${spring-cloud.version}</version>
                        <type>pom</type>
                        <scope>import</scope>
                    </dependency>
                </dependencies>
        </dependencyManagement>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
</project>
```

pom.xml hosted with ♡ by GitHub view raw

Comparando com o projeto do Config Server, temos apenas duas novas dependências. Explicarei elas abaixo:

- **spring-cloud-starter-config:** Como estamos utilizando um Config Server para centralizar nossa aplicação, o Spring fornece esta biblioteca para que a conexão com esse servidor seja simples, somente configurando no arquivo `bootstrap.yml` qual o host do Config Server.
- **spring-cloud-starter-eureka-server:** Essa dependência é a mais importante. Ela torna possível transformar a aplicação Spring Boot em um Servidor Eureka somente com algumas configurações nos properties e uma anotação na classe principal.

Como já sabemos, toda aplicação Spring Boot tem uma classe principal, temos que adicionar a ela a anotação `@EnableEurekaServer` para que seja iniciado um Servidor do Eureka.

```
package com.coderef.delivery;

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

@SpringBootApplication
@EnableEurekaServer
public class DeliveryEurekaServerApplication {

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

DeliveryEurekaServerApplication.java hosted with ♡ by GitHub view raw

Agora precisamos configurar o nosso `bootstrap.yml` . Como disse anteriormente já entramos em detalhes para que serve cada arquivo nas

stories anteriores, portanto irei somente explicar o conteúdo.

Como a configuração da aplicação ficará no repositório do Config Server precisamos criar somente o `bootstrap.yml` localmente.

Crie o arquivo `bootstrap.yml` no diretório `delivery-eureka-server/src/main/resources`.

```yaml
spring:
  application:
    name: delivery-eureka-server
  cloud:
    config:
      uri: http://localhost:9090
```

bootstrap.yml hosted with ♡ by GitHub view raw

A configuração do nome da aplicação nós já conhecemos, porém temos algo novo neste `bootstrap.yml`. Configuramos a propriedade `spring.cloud.config.uri` que indica o host do Config Server e para que na hora da inicialização a aplicação peça a ele suas propriedades.

Mas como o Config Server sabe qual é o arquivo em seu repositório que representa as configurações específicas dessa aplicação que está se conectando?

Bem, é ai que entra uma das responsabilidades da propriedade `spring.application.name`. Quando a aplicação `delivery-eureka-server` estiver iniciando ela consultará a aplicação `delivery-config-server`, perguntando a ela se existe em seu repositório um arquivo com o nome configurado em `spring.application.name` e com a extensão `properties` ou `yml`.

Mas espera um pouco, não criamos esse arquivo no repositório do Config Server ainda, estão vamos lá.

No repositório de configurações do Config Server, no mesmo local onde criamos o `sample-config-app.yml` na parte 2 dessa série, crie o arquivo `delivery-eureka-server.yml` com o conteúdo abaixo:

```yaml
server:
  port: 9091

eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: true
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

```
12    server:
13      wait-time-in-ms-when-sync-empty: 3000
```

delivery-eureka-server.yml hosted with ♡ by GitHub view raw

Neste arquivo temos varias configurações novas, vamos passar uma a uma para descobrirmos do que se trata:

- **eureka.instance.hostname:** Host onde o será iniciado o servidor do Eureka.
- **eureka.client.registerWithEureka:** Mesmo sendo um Servidor do Eureka, essa aplicação não deixa de ser um Microservice como os outros, portanto ele deve se auto registrar no eureka.
- **eureka.client.fetchRegistry:** Os clientes do Eureka obtêm as informações de registro do servidor e as armazenam no local. Depois disso, os clientes usam essa informação para encontrar outros serviços. Como essa aplicação é o próprio Servidor do Eureka, desabilitamos esta busca.
- **eureka.client.serviceUrl.defaultZone:** Nada mais do que a localização do Eureka Server, já que essa aplicação vai se auto registrar.
- **eureka.server.wait-time-in-ms-when-sync-empty:** Tempo em milissegundos que o Eureka Server irá esperar entre as sincronizações com os clientes.

Agora temos dois arquivos de configuração em nosso repositório, como você pode ver abaixo.

Certo! Como você já deve ter percebido temos uma dependência entre nossos dois Microservices criados até agora. Dependência em uma Arquitetura de Microservices não é uma coisa boa, porém se trata de um servidor de configuração que futuramente será clusterizado na nuvem e não iremos mais subir ele localmente.

Mas por enquanto sempre que quisermos subir Eureka Server precisamos antes iniciar o Config Server.

Para testarmos se já esta tudo funcionando corretamente, inicie o Config Server e depois o Eureka Server executando a classe principal, como vimos na story anterior.

Observe o log da inicialização do Eureka Server e confira se contém essas duas linhas:

```
Fetching config from server at: http://localhost:9090 (Início do Log)
Tomcat started on port(s): 9091 (http) (Fim do Log)
```

Isso significa que o Eureka Server está iniciado e funcionando. Acesse o endereço `http://localhost:9091/` e confira se o painel do Eureka aparece.

Aqui temos muita informação, tanto de serviços registrados quanto do consumo da maquina host e etc.

A seção mais importante para nós no momento é a `Instances currently registered with Eureka`. Aqui obtemos informações preciosas dos nossos Microservices registrados no Eureka, como por exemplo:

- O Status `UP` ou `DOWN`.
- Quantidade de instâncias do mesmo Microservice registradas. Importante lembrar que o próprio Eureka irá realizar o load balancer entre elas quando for solicitada uma instância pelo Gateway por exemplo.
- Host e Porta onde a instância está escutando.

Como só temos o Microservice do próprio Eureka Server registrado, não temos uma visão do quanto essa tela é importante ainda, mas de agora pra frente todo Microservice que nós implementarmos será registrado no Eureka, revelando assim todo o seu poder.

Por enquanto é isso pessoal, na próxima story iremos implementar o nosso Servidor de Autorização, e claro, conecta-lo ao nosso Config Server e Eureka Server. Até a volta.

Repositórios do Projeto

Configurações: https://github.com/rafaelcam/delivery-configs
Projeto: https://github.com/rafaelcam/delivery

Referências

https://cloud.spring.io/spring-cloud-netflix/
https://github.com/Netflix/eureka

Thanks to Diego Brener da Silva and Diego Rampim.

Spring Cloud Spring Boot Eureka Netflixoss Java