Spring Cloud Netflix e Eureka

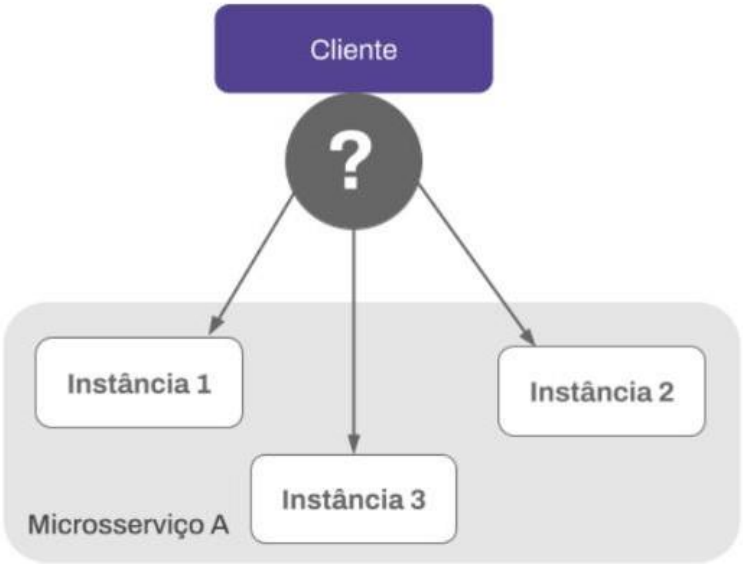
Começaremos a analisar cada um dos componentes da solução que a Netflix desenvolveu em sua arquitetura de microsserviços. Convidamos você a ver este vídeo para conhecer um pouco da história desse framework e de um de seus componentes, chamado Eureka. Não deixe de assistir!

Registro e descoberta de microsserviços: quais problemas vem resolver?

O estilo de arquitetura de microsserviços não é tanto sobre a construção de serviços individuais, mas sobre como tornar as interações entre os serviços confiáveis ??e tolerantes a falhas.

Nessa arquitetura, cada microsserviço fica em um endereço IP e uma porta, normalmente atribuídos dinamicamente, dificultando que um cliente faça uma solicitação a um microsserviço que, por exemplo, exponha uma API REST sobre HTTP.

Consideremos o seguinte diagrama. Temos um serviço com três instâncias, cada uma em um local diferente. Como o cliente poderia saber o endereço de cada um deles? E se formos um passo além do endereço, como ele saberia qual é a instância com menor carga para processar a requisição enviada?



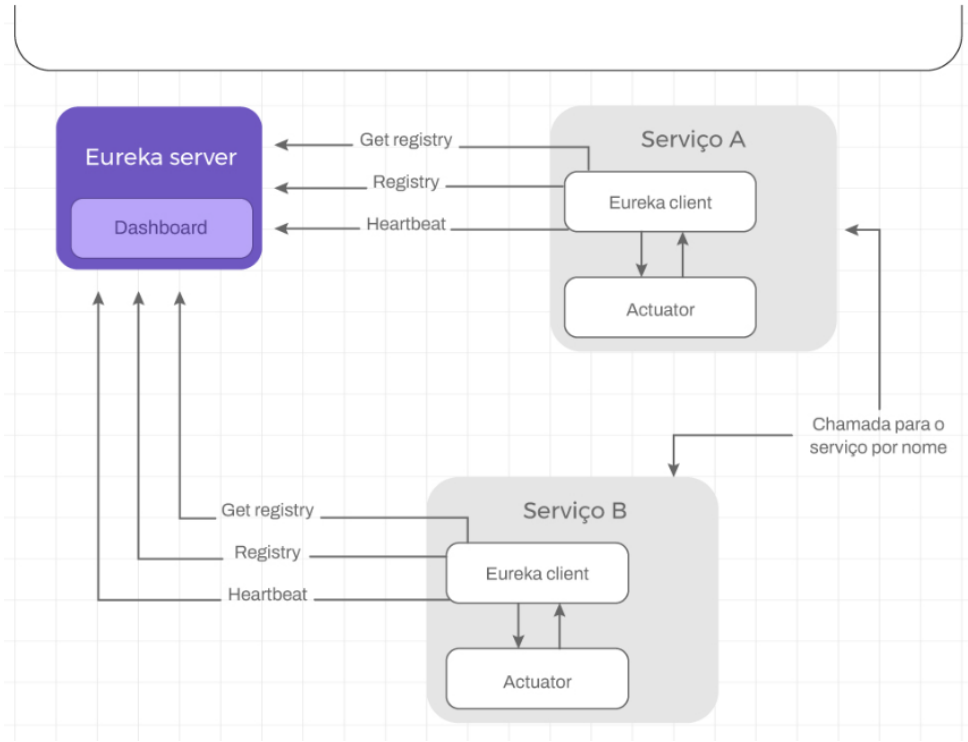
Perante este problema, devemos adicionar um novo componente à nossa arquitetura que nos permita:

- Registrar automaticamente os microsserviços e suas instâncias assim que estiverem em execução e cancelar o registro assim que eles pararem de ser executados ou pararem de responder.
- O cliente deve poder enviar uma solicitação aos microsserviços sem saber sua localização.
- As solicitações para instâncias de um microsserviço devem poder ser balanceadas por um balanceador de carga.

No Spring Cloud esse componente é o Eureka, que veremos em detalhes abaixo, mas primeiro, gostaríamos de lhe dizer de onde vem a palavra. Tudo começa quando Hiero II recebeu uma coroa e queria saber se era de ouro ou se tinha partes que não eram, então ele instruiu Arquimedes a fazer uma invenção para esse fim. Ele não conseguia encontrar a solução e o tempo o pressionava. Então, ele tomou um banho e notou como a água se movia dependendo de sua densidade. Assim, lhe ocorreu que poderia aplicar o mesmo princípio à coroa. É assim que a anedota conta que ele saiu correndo gritando “Eureka” (encontrei-o).

Arquitetura Eureka server

Podemos dividir o Eureka em dois grandes componentes: o Eureka client, responsável por publicar as informações do microsserviço em que se encontra; e o Eureka server, encarregado de coletar as informações de todos os clientes.



Monitorando a integridade de nossos microsserviços com o Spring Boot Actuator (heartbeat)

Em um ecossistema de microsserviços e instâncias, é necessário acompanhá-los, seja para controle de erros, para balanceamento de carga ou até mesmo para analisar métricas, assim podemos determinar se precisam de mais instâncias de um determinado serviço complementar.

Pode haver vários cenários de problemas em uma instância, mas os mais comuns são erros de exceção de aplicativo, erros de timeout e erros de instância. Erros de aplicativo são exceções programadas. Os problemas de instância podem ser, por exemplo, que a instância fique sem espaço em disco ou memória. Enquanto os erros de timeout acontecem quando a resposta chega após um determinado limite de tempo.

Obviamente, você não pode monitorar todos os endpoints e enviar dados “inertes” todas as vezes. Na prática, são usados endpoints especiais que retornam informações críticas sobre o status da instância e do serviço.

A solução oferecida pelo framework Spring para este problema é o Spring Actuator, que basicamente gera os endpoints necessários para esse monitoramento de maneira automática. No que lhe concerne, as respostas desses endpoints podem ser “pesquisadas” por balanceadores de carga ou outros aplicativos de gerenciamento de tráfego, como o Eureka, para checarem se a aplicação está saudável.

O Spring Boot Actuator fornece funcionalidades prontas para a produção que monitora nosso aplicativo, coleta métricas, entende e analisa nosso tráfego e o status do banco de dados, tudo isso sem exigir que nós mesmos o implementemos.

Os Actuators são usados principalmente para expor informações operacionais sobre nosso aplicativo em execução (health, metrics, info, dump, env, etc.) por uma API REST.

Endpoints úteis	
/health	Mostra informações sobre a saúde do nosso aplicativo. Um simples "status" se não estivermos autenticados, ou informações muito mais detalhadas se estivermos autenticados no aplicativo.
/info	Exibe informações arbitrárias do nosso aplicativo.
/metrics	Mostra as informações de métricas do nosso aplicativo.
/trace	Mostra informações de rastreamento (por padrão, as últimas solicitações HTTP).
/serviceregistry	Mostra o status do registro no Eureka server.

Por padrão, apenas o endpoint /health está habilitado para consumi-lo, para habilitar o restante temos que configurar a seguinte propriedade em application.properties:

```
management.endpoints.web.exposure.include=serviceregistry,health,info
```

Por exemplo, com esta configuração estamos habilitando os endpoints /serviceregistry, /health e /info. Caso queiramos habilitar todos, entramos:

```
management.endpoints.web.exposure.include=*
```

Se você quiser saber mais sobre os endpoints oferecidos pelo Actuator, recomendamos que acesse o seguinte link.

Por outro lado, o servidor Eureka usa os endpoints /health e /info para obter informações sobre os microsserviços.

Por padrão, se efetuarmos uma solicitação HTTP usando o método GET para o endpoint /health, ele responderá:

```
{
"status" : "UP"
}
```

No caso de /info, podemos personalizar a resposta definindo certas propriedades, por exemplo:

```
info.app.name=meu-servico
info.app.description=Servico testando Eureka
info.app.version=1.0.0
```

A resposta será:

```
{
"app" : {
"version" : "1.0.0",
"description" : "Servico testando Eureka",
"name" : "meu-servico"
}
}
```

Dependendo da versão do Actuador, o endpoint /info pode não ser ativado por padrão

para ler as variáveis carregadas a partir do application.properties. Para habilitá-lo, devemos adicionar o seguinte no referido arquivo:

```
management.info.env.enabled = true
```

Como adicionamos actuator ao nosso projeto?

Para isso, devemos usar Maven:

```
org.springframework.boot
spring-boot-starter-actuator
```

Posteriormente, atualizamos as dependências, executamos o projeto e teremos os endpoints prontos para consumir.

Conclusão

Eureka é uma solução de descoberta e registro de serviços que nos fornece recursos de tempo de execução robustos, resilientes e tolerantes a falhas. Com o Spring Cloud, fica fácil configurar um servidor Eureka e adaptar microsserviços para funcionar como clientes Eureka e para que possam ser registrados. Isso nos permite acompanhar as instâncias disponíveis.

Em breve, veremos como podemos dimensionar um microsserviço e enviar solicitações HTTP para suas instâncias passando por um balanceador de carga que nos ajudará a redirecioná-lo para a instância menos carregada.

Codificando juntos

Agora vamos ver como todos os conceitos que aprendemos nesta aula se reúnem no vídeo de live coding, onde mostraremos como implementar o Eureka passo a passo.