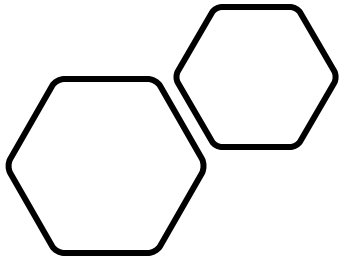
Arquitetura de Micros serviços

Prof. Bernardo Copstein

Prof. Júlio Machado

Criando os primeiros microsserviços



Leituras recomendadas:

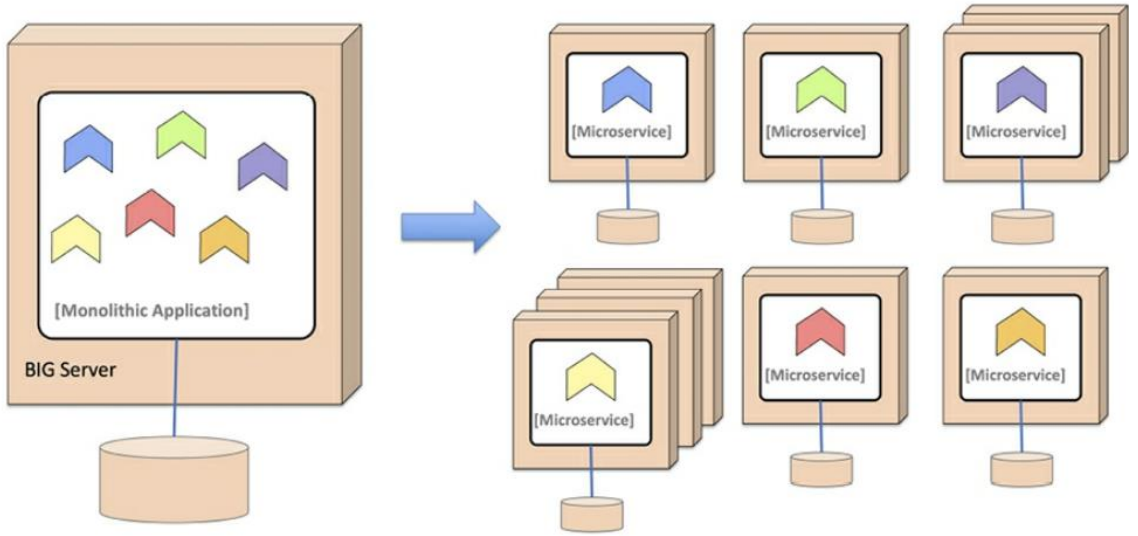
- Larsson, Magnus. Microservices with Spring Boot 3 and Spring Cloud: Build resilient and scalable microservices using Spring Cloud, Istio, and Kubernetes. Packt Publishing. Edição do Kindle.

Detalhando a definição de microsserviço

- Um microsserviço é basicamente um componente autônomo de software que pode ser atualizado, substituído e escalado de forma independente. Para poder agir como um elemento autônomo ele deve atender certos critérios:
  - Precisa atender a uma “arquitetura de não compartilhamento”, isto é, microsserviços não compartilham dados em bancos de dados, apenas se comunicam por troca de mensagens através de interfaces bem definidas.
  - A comunicação pode deve ocorrer através de interfaces bem definidas tanto usando APIs e serviços síncronos como por mensagens assíncronas (preferível).
  - As APIs e formatos de mensagens usados devem ser estáveis e bem documentados e devem evoluir seguindo uma estratégia de versionamento pré-definida.
  - Devem ser “instalados” como processos separados. Cada instância de microsserviço executa em um processo separado, por exemplo, um Docker container.
  - As instancias de um microsserviço devem ser “stateless” de maneira que qualquer instancia pode tratar qualquer requisição destinada para ele.

Qual o tamanho de um microsserviço?

- Pequeno suficiente para “caber na cabeça” de um único desenvolvedor
- Grande o suficiente para não comprometer o desempenho (ou seja, a latência) e/ou a consistência dos dados (chaves estrangeiras SQL entre dados armazenados em diferentes microsserviços não são mais algo que você pode considerar garantido)



Desafios com microsserviços

- Muitos componentes pequenos que usam comunicação síncrona podem levar a um problema de “falha em cadeia”, em especial quando sob carga elevada.
- Manter a configuração de vários elementos pequenos atualizada pode ser um desafio.
- É difícil rastrear uma requisição que é processada por vários componentes (por exemplo para análise de causa raiz) se cada um armazena “logs” locais.
- Analisar o uso de hardware por componente pode ser complexo.
- A configuração e o gerenciamento manual de muitos componentes pequenos pode ser custosa e sujeita a erros.

As 8 falácias da computação distribuída

1. A rede está disponível
2. A latência é zero
3. A largura de banda é infinita
4. A rede é segura
5. A topologia da rede não muda
6. Só tem um usuário administrador
7. O custo de transporte é zero
8. A rede é homogênea

Tolerância a falhas

- A melhor política quando se constroem aplicações distribuídas é sempre partir do princípio que algo pode dar errado
- No lado do servidor a arquitetura tem de ser capaz de prever as falhas e reiniciar os serviços com problemas
- No lado do cliente deve-se garantir que não são enviadas requisições para serviços com problemas

Usando o Spring-Boot

- Já usamos o Spring-Boot na implementação de um serviço como um monolito e vamos continuar usando na implementação de microsserviços.
- Vamos entender um pouco mais sobre a forma como esse framework está organizado visando facilitar o desenvolvimento de serviços e microsserviços.
- Vamos destacar dois aspectos:
  - O padrão de projeto “Configuração sobre Configuração”
  - O uso de arquivos “.jar” robustos (“fat jar files”)

O padrão Configuração sobre Configuração

- O Spring-Boot utiliza várias pré definições padrão para configurar tanto seus módulos internos como os produtos de terceiros tais como as bibliotecas de “log” ou conexões com bancos de dados. Isso minimiza a necessidade de configuração inicial simplificando o uso do framework.
- Quando necessário, entretanto, cada convenção pode ser sobreposta escrevendo-se o código de configuração respectivo.

Configuração sobre configuração na prática

- A configuração automática ocorre de várias formas a partir da classe anotada com “@SpringBootApplication”. Essa anotação provê várias funcionalidades:
  - Habilita o “component scanning”, isto é, o Spring-Boot procura por components Spring e classes de configuração no pacote e subpacotes da aplicação.
  - A classe da aplicação transforma-se em uma classe de configuração.
  - Ativa a autoconfiguração: o Spring-Boot procura por arquivos JAR no “classpath” que possam ser configurados automaticamente. Por exemplo se o “Tomcat” estiver no classpath, ele será automáticamente configurado e incluído no arquivo “.jar” de distribuição da aplicação como um servidor Web embutido.
- Se for o caso de usar componentes declarados em outros pacotes, basta usar a anotação “@ComponentScan”.
- Se for o caso de sobrescrever configurações padrão, basta anotar uma classe com “@Configuration” que ela será ativada pelo mecanismo de “componente scan” no início da aplicação.

“Fat jar files”

- O mecanismo de configuração automática do Spring-Boot facilita a criação de arquivos de distribuição que já contenham todas as tecnologias necessárias para a execução da aplicação (como é o caso do TomCat no desenvolvimento dos serviços e microsserviços).
- Isso tem implicações:
  - Vantagem: simplifica enormemente a execução de uma aplicação. Normalmente um comando simples como “mvn spring-bot:run” dispara o serviço e todos os mecanismos de apoio necessários (servidor web, servidor de banco de dados, gateway etc)
  - Desvantagem: o tempo de carga do arquivo “.jar” pode ser um pouco maior que o de outras tecnologias

Primeiro roteiro

- O primeiro roteiro mostra como disparar dois microsserviços de forma manual
  - Observe como o primeiro é completamente autônomo e possui seu próprio BD
  - Observe como o segundo depende do primeiro mas apenas por troca de mensagens
- Um sistema de monitoramento dos microsserviços é incluído de forma automática pelo simples fato de ser listado no arquivo ".pom" (incluído no classpath).
- Veja o roteiro no Moodle