Proposta de Trabalho
MAC0499 - Trabalho de Formatura Supervisionado
Instituto de Matemática e Estatística - Universidade de São Paulo
29 de Abril de 2022

Lucas Moretto da Silva
Supervisores: Alfredo Goldman Vel Lejbman, João Francisco Lino Daniel

1 Introdução

O mundo de software atual está focado cada vez mais no desenvolvimento de aplicações distribuídas. Nesse contexto, arquitetura de microsserviços é um tema cada vez mais em evidência.

Esse estilo de arquitetura traz diversas vantagens para os desenvolvedores e para a aplicação, mas, carrega consigo diversas responsabilidades que não podem ser ignoradas. A integração e comunicação entre microsserviços é uma dessas responsabilidades e, se bem desenvolvida, proporciona a independência e autonomia do sistema. Além disso, tendo em vista a complexidade de uma aplicação estruturada em microsserviços, podemos dizer que padrões de arquitetura são essenciais para o desenvolvimento bem-sucedido de sistemas baseados nesse estilo.

Neste trabalho pretendemos realizar uma série de experimentos que explicitem a análise de trade-offs arquiteturais focando no escopo de integração e comunicação entre microsserviços. Estudaremos os padrões de comunicação entre microsserviços (síncrono e assíncrono), desenvolveremos aplicações que fazem uso desses diferentes estilos e realizaremos a análise de trade-offs para identificar como a escolha de um padrão pode influenciar a estruturação de uma aplicação e as decisões de um arquiteto de software.

Espera-se, como resultado, ter em mãos uma análise bem estruturada dos impactos arquiteturais decorrentes da utilização de cada forma de integração entre microsserviços que seja útil para auxiliar engenheiros de software na tomada de decisão em seus projetos.

2 Background

2.1 Arquitetura de microsserviços

A arquitetura de microsserviços (MSA) refere-se a um estilo arquitetural para o desenvolvimento de aplicações. Neste estilo arquitetural, o sistema é desenvolvido em diversas partes pequenas, cada qual executada em um processo isolado - essa parte recebe o nome de “microsserviço”. Esse tem seu escopo definido através de um único domínio de responsabilidade. Nesse estilo de arquitetura, os microsserviços comunicam-se entre si por meio de interfaces simples e leves para resolver problemas de negócios, montar respostas provenientes de diferentes partes do sistema, etc..

A arquitetura de microsserviços foi desenvolvida para superar as dificuldades apresentadas por abordagens arquiteturais monolíticas. A arquitetura monolítica é caracterizada pelo desenvolvimento da aplicação em um único código-fonte que é executado em um único processo contendo todos os componentes de software de um aplicativo: interface de usuário, camada de negócios e interface de dados. Construir uma aplicação como uma única unidade traz diversas limitações, como inflexibilidade, falta de confiabilidade, dificuldade de dimensionamento, desenvolvimento lento e assim por diante. Foi para contornar esses problemas que a arquitetura de microsserviços foi criada.

A divisão de uma aplicação em microsserviços permite um desenvolvimento mais rápido, detecção e resolução de bugs mais eficientes, manutenção mais simples, flexibilidade e maior disponibilidade e escalabilidade.

2.1.1 Características de microsserviços

Autonomia

Cada serviço pertencente a uma arquitetura de microsserviços deve ser desenvolvido, implantado, operado e dimensionado sem afetar o funcionamento dos demais. Os componentes não precisam compartilhar nenhum código ou implementação uns com os outros. Qualquer comunicação entre microsserviços acontece diretamente por meio de APIs simples e bem definidas ou indiretamente por infraestruturas de mensageria do tipo “dumb pipe” .

Especificação

Cada serviço é projetado com um conjunto de recursos e se concentra na solução de um problema específico. Se os desenvolvedores contribuírem com mais código para um serviço ao longo do tempo e o serviço se tornar complexo, ele pode ser dividido em serviços menores.

Agilidade

Os microsserviços promovem a organização de um time de desenvolvimento em equipes pequenas e independentes que se apropriam de seus serviços. As equipes agem dentro de um contexto pequeno e bem compreendido, e têm o poder de trabalhar de forma mais independente e rápida, ajudando a reduzir o tempo de desenvolvimento.

Elasticidade

Os microsserviços permitem que cada serviço seja dimensionado independentemente para atender à demanda de recursos do aplicativo. Isso permite que as equipes dimensionem as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade se um serviço tiver um pico de demanda e reduzam a oferta em momentos de baixa demanda.

Fácil implantação

Os microsserviços permitem integração e entrega contínuas, facilitando a experimentação de novas ideias e a reversão se algo não funcionar. O baixo custo da falha permite a experimentação, facilita a atualização do código e acelera o tempo de entrega de novas releases.

Liberdade de tecnologias

Dentro de uma arquitetura monolítica, assim que a stack de tecnologias da aplicação é definida, ela terá de ser carregada e suportada até o fim do desenvolvimento. Pode ser que a escolha das ferramentas utilizadas não beneficiem parte das funcionalidades do sistema, mas, como o monolito é um artefato único, a equipe de desenvolvimento fica presa a decisão feita no começo do projeto.

Numa arquitetura de microsserviços as equipes têm a liberdade de escolher a melhor ferramenta para resolver seus problemas ou desenvolver as funcionalidades da aplicação. Como consequência, as equipes que criam microsserviços podem sempre escolher a melhor ferramenta para cada trabalho.

Código reutilizável

Dividir o software em módulos pequenos e bem definidos permite que as equipes reutilizem funções para diversos propósitos. Um serviço escrito para uma determinada função pode ser usado como um bloco de construção para outro recurso.

Resiliência

A independência de cada serviço aumenta a resistência a falhas de uma aplicação. Em uma arquitetura monolítica, se um único componente falhar, pode causar a falha de todo o sistema. Com os microsserviços, os componentes lidam com a falha total do serviço degradando a funcionalidade e não travando a aplicação inteira.

Existem diversas estratégias que podem ser utilizadas para garantir a tolerância a falhas em microsserviços. Por exemplo, pode-se utilizar uma estratégia de abertura e fechamento de circuito (*circuit breaking*) onde, caso ocorra falhas no sistema, “abre-se” o circuito, impedindo comunicação entre os serviços que falharam por um determinado período de tempo. O sistema continua monitorando o ponto de falha e, quando as coisas voltarem ao normal, o circuito é “fechado” para restabelecer a funcionalidade padrão.

Pode-se utilizar estratégias de *timeouts*, que definem que não se deve esperar por uma resposta de serviço por um período de tempo indefinido — lance uma exceção em vez de esperar muito. Isso garante que o sistema não fique preso em um estado intermediário, continuando a consumir recursos do aplicativo. Quando o tempo limite for atingido, a comunicação é liberada novamente.

Além disso, existem estratégias mais simples, como utilização de *retry*, que simplesmente indica para tentar novamente uma conexão assim que a falha acontece. Isso é muito útil em caso de problemas temporários com um dos microsserviços.

2.1.2 Desafios inerentes a arquitetura de microsserviços

Complexidade da solução

Na arquitetura de microsserviços uma aplicações possui muito mais componentes e camadas que funcionam em conjunto do que um monolito equivalente. Cada microsserviço é mais simples, mas o sistema como um todo é mais complexo de se desenvolver e manter.

Integração/Comunicação entre microsserviços

Embora os microsserviços possam existir e funcionar de forma independente, em muitas situações precisam interagir e se comunicar com outros microsserviços para realizar algum processo. Isso requer o desenvolvimento de APIs funcionais, simples e bem definidas que atuem como canal de comunicação entre vários serviços que constituem a aplicação.

Além disso, como existirão diversos microsserviços, a comunicação entre eles pode ficar bastante intensa para a rede e consumir muitos recursos. Outro problema que pode surgir é a questão da latência de resposta. Se a cadeia de dependências do serviço ficar muito longa (o serviço 1 chama 2, que chama 3...), a latência adicional pode se tornar um problema para a aplicação.

Logs distribuídos

Microsserviços são aplicações independentes e cada um deles geralmente possui um mecanismo de log distinto. Por consequência grandes volumes de dados de log são gerados de forma não centralizada e não estruturada, o que pode acarretar em problemas para organizar e manter os registros de uma mesma aplicação.

Dependências cíclicas entre serviços

Uma dependência cíclica refere-se ao acoplamento entre dois ou mais serviços na aplicação. Pode ser que durante o desenvolvimento existam serviços tão dependentes um do outro que podem deixar a solução mais complexa e dificultar a manutenção da aplicação.

Garantir consistência de dados

Cada microsserviço é responsável por seu próprio mecanismo de persistência de dados, com seus próprios banco de dados independentes, etc. Como resultado, manter a consistência dos dados entre diferentes microsserviços pode ser um desafio.

2.2 Padrões de comunicação em microsserviços

Serviços podem se comunicar por meio de diferentes formas, cada uma visando cenários e objetivos diferentes. Podemos classificar as formas de comunicação em três eixos: comunicação síncrona, comunicação assíncrona e híbrida.

2.2.1 Comunicação síncrona

Protocolo síncrono é aquele no qual os dados são transmitidos de um serviço a outro através de uma interação blocante. Geralmente essa comunicação é feita utilizando REST ou gRPC. Dizemos que é blocante pois, na comunicação entre dois serviços, o cliente envia uma requisição e aguarda uma resposta do serviço. Isso porque o protocolo utilizado (HTTP/HTTPS) é síncrono, então o código cliente só pode continuar sua execução quando receber a resposta do servidor HTTP.

2.2.2 Comunicação assíncrona

Na comunicação assíncrona, um cliente envia uma requisição, mas não para sua execução para esperar uma resposta. Geralmente esse tipo de comunicação é feita utilizando-se o protocolo AMQP (*Advanced Message Queuing Protocol*).

O AMQP foi criado como um protocolo aberto que permite a interoperabilidade de mensagens entre sistemas, independente do cliente produtor das mensagens e da plataforma utilizada. Além de definir o protocolo da camada de rede da aplicação, o AMQP também especifica uma arquitetura de alto nível para componentes intermediários das mensagens.

Nessa definição estão especificados um conjunto de recursos que devem ser disponibilizados por uma implementação de um servidor (chamado *broker*) compatível com AMQP (como RabbitMQ), regras de como as mensagens devem ser processadas, roteadas, armazenadas, etc.

O funcionamento de um sistema seguindo o protocolo AMQP ocorre da seguinte forma: um cliente chamado produtor envia uma mensagem para o broker. Estes distribuem cópias das mensagens para filas, dependendo das regras definidas pelo mesmo e da chave fornecida na mensagem. Por fim, a mensagem é consumida por um cliente chamado consumidor.

2.2.3 Comunicação híbrida

Entre dois serviços podemos ter apenas um tipo de comunicação de cada vez (ou síncrona ou assíncrona). Porém, uma aplicação arquitetada seguindo o estilo de microsserviços pode ser composta por uma vasta coleção de componentes.

Sendo assim, uma aplicação baseada em microsserviços geralmente usa uma combinação desses estilos de comunicação, variando a utilização de acordo com os cenários e especificações de negócio.

2.3 Trabalhos relacionados

O artigo de Thatiane de Oliveira Rosa *et al.* [12] define e exemplifica uma técnica sistemática para identificar padrões de arquitetura que afetam um determinado conjunto de atributos previamente escolhidos pelo arquiteto. A técnica é exposta em um passo a passo bem definido e exemplificado de forma geral, levando em consideração diversas características do estilo MSA.
O artigo de S. Hassan e R. Bahsoon [13], explora formas para desenvolvimento de sistemas em microsserviços e aplica técnicas para análise de trade-offs arquiteturais durante o desenvolvimento de uma sistema teste para streaming de músicas. Os autores expõem como essas técnicas podem ser abordadas pelos arquitetos durante o desenvolvimento do software.

3 Proposta

3.1 Metodologia

Tendo em vista a complexidade de uma aplicação estruturada em microsserviços, podemos dizer que padrões de arquitetura são essenciais para o desenvolvimento bem-sucedido de sistemas baseados nesse estilo.

A arquitetura de microsserviços ainda é um estilo de desenvolvimento muito recente. Os padrões e tecnologias atuais não solucionam todos os problemas da arquitetura, e estes, quando existem, também são recentes e necessitam de novas técnicas. Os autores de [12] apresentam um método para análise de trade-offs arquiteturais que auxiliam o arquiteto de software a tomar decisões e optar por padrões que melhor se adequam às necessidades do sistema.

Neste trabalho pretendemos realizar uma série de experimentos que explicitem a análise de trade-offs arquiteturais focando no escopo de integração entre microsserviços. Mais especificamente, pretendemos:

- Estudar os padrões de comunicação entre microsserviços;
- Implementar uma mesma aplicação com versões que fazem uso de diferentes estilos de comunicação;
- Realizar a análise de trade-offs arquiteturais com base em alguns atributos qualitativos previamente definidos. No momento já temos alguns desses especificados, como, por exemplo, complexidade da solução técnica envolvida, performance do sistema, heterogeneidade de tecnologias a disposição da solução, mas poderemos incluir outros;
- Quanto às tecnologias, focaremos em comparar REST, para comunicação síncrona, e AMQP (utilizando Apache Kafka ou RabbitMQ), para comunicação assíncrona.

3.2 Resultados Esperados

Pelo fim do trabalho pretendemos ter em mãos uma análise bem estruturada dos impactos arquiteturais decorrentes da utilização de cada padrão de integração entre microsserviços.

Esperamos que nossa análise possa auxiliar arquitetos de software definir a forma de integração que melhor se adequa a suas necessidades e que o ajude a evidenciar os impactos de padrões de arquitetura em certos atributos de qualidade investigados.

4 Cronograma

1. Definição das tecnologias
2. Definição da aplicação e sua modelagem
3. Implementação da aplicação com integração síncrona
4. Implementação da aplicação com integração assíncrona
5. Testes de performance
6. Análise os trade-offs arquiteturais das soluções
7. Escrita da monografia

Tarefas	Mai	Jun	Jul	Ago	Set	Out	Nov
1	✔						
2	✔						
3		✔	✔				
4			✔	✔			
5				✔	✔		
6		✔	✔	✔	✔	✔	
7	✔	✔	✔	✔	✔	✔	✔

5 Referências

[1] Rodgers, Peter. "Service-Oriented Development on NetKernel- Patterns, Processes & Products to Reduce System Complexity Web Services Edge 2005 East: CS-3" (último acesso em 27 de Abril de 2022)

[2] Lewis, James . "Micro services - Java, the Unix Way". (último acesso em 27 de Abril de 2022)

[3] George, Fred. "MicroService Architecture: A Personal Journey of Discovery". (último acesso em 27 de Abril de 2022)

[4] MW Team. "What are microservices". (último acesso em 27 de Abril de 2022)

[5] Newman, Sam. Building Microservices: Designing fine-grained systems. [s.l.]: O'reilly Media, Inc., 2015

[6] Newman, Sam. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. [s.l.]: O'reilly Media, Inc., 2019
[7] Richardson, Chris. Microservices Patterns: With Examples in Java, 2018

[8] Bellemare, Adam. Building Event-Driven Microservices: Leveraging Organizational Data at Scale. 2020

[9] Richards, Mark. Ford, Neal. Fundamentals of Software Architecture: An Engineering Approach, 2020

[10] Fowler, Martin. "Microservices" (último acesso em 27 de Abril de 2022)

[11] Vogels, Werner. "Amazon, microservices and the birth of AWS cloud computing" (último acesso em 28 de Abril de 2022)

[12] Thatiane de Oliveira Rosa, João Francisco Lino Daniel, Eduardo Martins Guerra, and Alfredo Goldman. 2020. A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes. In European Conference on Pattern Languages of Programs 2020 (EuroPLoP '20), July 1-4, 2020, Virtual Event, Germany. , 8 pages. 10.1145/3424771.3424809

[13] S. Hassan and R. Bahsoon, "Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap," 2016 IEEE International Conference on Services Computing (SCC), 2016, pp. 813-818, doi: 10.1109/SCC.2016.113.

[14] Behara, Samir. "Making Your Microservices Resilient and Fault Tolerant" (último acesso em 29 de Abril de 2022)

[15] Johansson, Lovisa. "What is AMQP and why is it used in RabbitMQ?" (último acesso em 29 de Abril de 2022)

[16] Ledbrook, Peter. "Understanding AMQP, the protocol used by RabbitMQ" (último acesso em 29 de Abril de 2022)