ISSN 2316-2872
T.I.S. São Carlos, v. 4, n. 3, p. 209-215, set-dez 2015

Desenvolvimento de Aplicações e Micro Serviços: Um estudo de caso

Pedro Felipe Marques Moreira , Delano Medeiros Beder

Resumo: Não existe arquitetura mágica que soluciona todos os problemas enfrentados pela engenharia de software. É importante que arquitetos e engenheiros de software conheçam diferentes arquiteturas bem como suas vantagens e desvantagens para utilizá-las da melhor maneira possível dentro de cada cenário. Este trabalho tem por objetivo apresentar a arquitetura de micro serviços, com um levantamento teórico e implementação de um micro serviço utilizando o framework Spring Boot e apresentar quais as vantagens e desvantagens dessa arquitetura, mostrando como micro serviços podem diminuir a complexidade de sistemas complexos.

Palavras-Chave: micro serviço, arquitetura de software, framework Spring Boot.

Development of Applications and Micro Services: A study case

Abstract: *There is no magic architecture that solves all the problems faced by software engineering. It is important that architects and software engineers know different architectures as well as their advantages and disadvantages in order to use them in the best possible way within each scenario. This work aims to present the micro services architecture, with a theoretical research and implementation of a micro service using Spring Boot framework and show the advantages and disadvantages of this architecture, showing how micro services can reduce the complexity of systems.*

Keywords: *micro service, software architecture, Spring Boot framework.*

I. Introdução

Segundo Richards (2015), existem diversas abordagens e estilos arquiteturais para a construção de sistemas computacionais. Conhecer as vantagens e desvantagens de cada arquitetura visando a melhor escolha para um dado cenário é fundamental para o sucesso de um negócio.

Este trabalho apresenta a arquitetura de micro serviços por um estudo conceitual, onde as principais fontes de informação são oriundas de artigos publicados por engenheiros e arquitetos de software, tais como, Martin Fowler, Sam Newman e Chris Richardson.

A partir do levantamento teórico será apresentado as principais vantagens e desvantagens da utilização de micro serviços, com comparações com sistemas monolíticos. Não é objeto deste trabalho a realização de estudos comparativos entre sistemas monolíticos e micro serviços, mas algumas comparações serão apresentadas.

Foi implementado um micro serviço, que será consumido por uma aplicação web monolítica que utiliza a tradicional arquitetura de referência em camadas Java EE (ORACLE, 2015), com o objetivo de apresentar a vivência prática do funcionamento da interação desse sistema com um micro serviço.

Ao final serão apresentados alguns trabalhos que de alguma forma se relacionam com o tema apresentado por este, bem como sugestões de trabalhos futuros.

Este trabalho está dividido da seguinte forma. Na seção 2 é realizado um levantamento teórico sobre micro serviços, explicando o que é, quais suas vantagens e desvantagens. Na seção 3 é descrito como foi o processo de desenvolvimento do micro serviço, apresentado o estilo arquitetural do sistema e o porquê da utilização de Spring Boot (WOODS, 2015) para a construção do micro serviço. Na seção 4 são descritos os resultados obtidos com a análise em cima do estudo conceitual sobre o tema, do processo de desenvolvimento do micro serviço e implementação junto ao um sistema monolítico. Na seção 5 são apresentados alguns trabalhos correlatos e realizado uma breve análise crítica sobre os mesmo. Finalmente, são apresentadas as considerações finais e sugestões para futuros trabalhos na seção 6.

II. Revisão Bibliográfica

Com o crescimento de um sistema, o código base aumenta.

Departamento de Computação - Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – São Carlos – SP – Brasil
Autor para correspondência: pedrofelipemm@gmail.com, delano@dc.ufscar.br

Torna-se cada vez mais difícil mantê-lo. É comum o desenvolvimento de código duplicado, módulos altamente acoplados, métodos com centenas de linhas de código realizando mais processos do que deveria.

Em aplicações monolíticas, na tentativa de combater tais problemas, muitas vezes são criados módulos, abstrações e bibliotecas com código comum. Tais processos nem sempre são tão reutilizáveis como deveriam ser, e quando são, ficam limitadas a serem utilizados dentro do escopo do sistema, uma vez que são empacotados dentro de um único artefato.

A) Micro Serviços

Segundo Newman (2015a), micro serviços são serviços com poucas responsabilidades, pequenos e autônomos que podem trabalhar de forma independente ou em conjunto com outros serviços. Ao contrário de sistemas monolíticos, cada micro serviço é empacotado em um artefato separado, e portanto pode-se realizar a implantação de cada um independentemente.

Como bem explicado por Newman (2015a, p. 17), um assunto que é bastante debatido é: quão pequeno é pequeno? É fácil perceber quando um serviço é grande demais, possui muitas dependências ou faz muitas coisas, logo ele poderia ser quebrado em dois ou mais serviços. Então quando um serviço não parecer grande demais, já é pequeno o suficiente.

Quanto menor o serviço, maior a maximização dos benefícios e minimização das desvantagens da arquitetura orientada a micro serviço. Serviços extremamente pequenos tornam-se mais independentes, porém a complexidade de desenvolvimento e deploy aumentam consideravelmente.

Cada micro serviço é uma entidade separada e autônoma, eles podem ser implantados em diferentes máquinas, e ainda assim trabalhar de forma coletiva, sendo capaz de executar seus processos de forma independente caso algum outro serviço falhe. Para garantir a separação entre eles, toda a comunicação é realizada por chamadas de rede.

Ao realizar a modelagem de um micro serviço deve-se pensar no que faz sentido expor para terceiros e no que faz sentido esconder, encapsulando a lógica de execução dos consumidores.

Outro ponto de atenção é como será realizada a comunicação entre eles. Ao escolher um protocolo de comunicação padronizado e um formato de dados multiplataforma como por exemplo, HTTP (Hypertext Transfer Protocol) e JSON (JavaScript Object Notation) respectivamente, torna-se também possível que cada micro serviço seja escrito em uma linguagem diferente.

“A regra de ouro: Pode-se fazer uma mudança para um serviço e implantá-lo por si só sem mudar nada além disso?” (NEWMAN, 2015a, p. 18, tradução nossa). Caso a resposta seja sim, é um sinal que o serviço é pequeno o suficiente, o que o fará tirar bastante proveito das vantagens dessa arquitetura.

"[...] Steve reitera que em sua opinião, micro serviços não são novos e é de fato apenas uma abordagem de entrega orientada a serviço." (LITTLE, 2015, tradução nossa). Pode-se dizer que micro serviços nada mais são do que serviços bem definidos, pouco acoplados, seguindo os princípios da boa e velha arquitetura orientada a serviços (SOA), com um grau de maturidade maior, justamente por sua independência e capacidade de trabalhar em conjunto com outros micro serviços ou de forma autônoma ao mesmo tempo.

Farcic (2015) também afirma que a maioria das melhores práticas podem ser aplicadas a serviços de modo geral, porém com micro serviços elas tornam-se ainda mais importantes.

“Em primeiro lugar, implementações SOA tradicionais tem sido construídas em torno de monolíticos [...]” (WEIR, 2015, tradução nossa). Alguns problemas solucionados por micro serviços que ainda persistem em tradicionais serviços SOA são:

- Serviços gigantescos: Apesar da antiga aplicação monolítica ter sido quebrada em diferentes serviços, ainda é bastante comum encontrar serviços realizando mais processos do que deveria.
- Orquestração: Em aplicações SOA é bastante comum a existência de um orquestrador para enfileirar os processos de maneira correta, tornando parte do código procedural, ao invés de possuir inteligência para lidar com diferentes cenários.
- Alta complexidade: Embora a proposta seja justamente o contrário, todos esses serviços enormes sendo orquestrados por uma unidade central tornam-se extremamente complexos, difícil de ser mudado e testado.

B) Vantagens

As vantagens no uso de micro serviços podem ser caracterizadas como heterogeneidade tecnológica, resiliência, escalabilidade e facilidade de implantação.

Heterogeneidade Tecnológica

Com um sistema composto por múltiplos serviços colaborativos, é possível que eles sejam escritos em linguagens diferentes. Dessa maneira, pode-se escolher a melhor linguagem/ferramenta para a execução das tarefas.

Uma abordagem para um rede social seria ter um micro serviço responsável por armazenar as relações entre os usuários em um banco de dados orientado a grafos, enquanto outro micro serviço poderia armazenar os posts em um banco de dados orientado a documentos.

Uma grande vantagem dos micro serviços é a possibilidade de testar novas tecnologias e ferramentas com baixo risco (FOWLER, 2015). Caso algo dê errado basta reescrever uma pequena parte do código. Enquanto que nas aplicações monolíticas, a adoção de uma nova ferramenta pode impactar grande parte do sistema, o que requer muitos esforços em testes, e estes nem sempre são suficientes.

Não importa quantas e quais tecnologias, linguagens, ferramentas e padrões são utilizados, os micro serviços bem escritos são autônomos, trabalham em harmonia e realizam suas tarefas de maneira elegante, tornando os tecnologicamente heterógenos, como se todos utilizassem a mesma linguagem. São de fato, partes de um todo.

Resiliência

Resiliência é a habilidade dos sistemas de recuperação de falhas. Em sistemas monolíticos se uma parte falha, o sistema todo para (RICHARDSON , 2015). É comum rodar o mesmo sistema em múltiplas máquinas, para que na falha de uma instância tenha outras operando.

Com micro serviços, caso um ou mais falhe é possível isolar totalmente o problema. Com perda parcial das capacidades de um sistema.

A Netflix é um bom exemplo de utilização dessa arquitetura. Se o serviço de recomendações falhar, o serviço de streaming continua operando normalmente como se nada tivesse acontecido.

No entanto é preciso ter cuidado e entender os problemas gerados por múltiplas fontes de dados, assim como máquinas, redes também falham. Saber como lidar com possíveis falhas é a chave para a construção de um sistema resiliente.

Escalabilidade

Em aplicações monolíticas é necessário escalar tudo ou nada. Com micro serviços pode-se escolher quais partes precisam ser atualizadas (RICHARDSON, 2015). Por exemplo, se um determinado serviço é responsável por replicação de dados em diferentes bases, não faz sentido realizar um upgrade no processador da máquina que ele está sendo executado.

Facilidade de Implantação

É possível realizar a implantação somente da feature que será lançada. Se algo der errado é fácil de isolar o problema, executar procedimentos de rollback, ou simplesmente parar o micro serviço problemático. Justamente por sua facilidade e controle na hora de realizar o deploy, é muito simples a execução de várias entregas menores.

Enquanto que com tradicionais aplicações monolíticas as implantações são longas, demoradas e pelo risco de que um único componente pode comprometer o funcionamento do sistema todo, geralmente são realizadas poucas entregas com muitas features e correções, o que requer muitos testes.

C) Desvantagens

As desvantagens no uso de micro serviços podem ser caracterizadas como complexidade de desenvolvimento, chamadas remotas e gerenciamento de múltiplos bancos de dados e transações.

Complexidade de Desenvolvimento

Umas das principais reclamações em relação a micro serviços referem-se ao aumento da complexidade de desenvolvimento. É muito mais fácil escrever sistemas em um único projeto, realizando chamadas internas entre classes onde geralmente o único requisito para tal é a importação de uma classe em outra.

Quando trabalha-se com diferentes projetos e módulos é necessário gerenciar as dependências entre eles. Mesmo que atualmente exista excelentes ferramentas para gestão de dependências e automação de builds como o Maven, por exemplo, ainda é necessário sempre manter as versões atualizadas.

Em projetos onde muitos módulos estão sendo desenvolvidos paralelamente a boa comunicação entre equipes é de suma importância. Mesmo os micro serviços sendo escritos para trabalharem de maneira autônoma, com ou sem as partes com as quais eles interagem, se os diferentes módulos do projeto não forem atualizados os micro serviços não tirarão proveito das novas features que forem lançadas.

Chamadas Remotas

Um ponto importante que merece atenção é sobre a comunicação entre os micro serviços, pois chamadas remotas são muito mais custosas do que chamadas a classes internas.

Por isso é importante definir o escopo dos serviços. Se forem pequenos demais a comunicação via rede se tornará um problema. O ideal é que um micro serviço seja pequeno o suficiente para executar sua funcionalidade, sem a realização de processos adicionais que não fazem parte de seu propósito e diminuem sua coesão.

Gerenciamento de múltiplos bancos de dados e transações

Gerenciar diversos bancos de dados sendo consumidos por múltiplos micro serviços não é uma tarefa trivial. É necessário um grande esforço inicial para realizar todas as configurações necessárias e ainda têm alto custo para manter vários bancos de dados funcionando.

Controles transacionais em sistemas distribuídos são extremamente complexos. Em sistemas monolíticos há diversos frameworks, como Spring, que fornece recursos transacionais e suporte a rollback. Em sistemas distribuídos essa tarefa se torna bem mais complicada, uma vez que precisa ser realizada uma série de tratamentos em caso de falha.

III. Desenvolvendo micro serviços com Spring Boot

A) Spring Boot

A principal motivação do projeto Spring Boot é a facilidade de utilização e configuração. Ao contrário das tradicionais aplicações desenvolvidas com Spring onde é necessário extensos XMLs ou classes de configuração para um simples Hello World, com Spring Boot é possível iniciar uma aplicação com poucas linhas de código, permitindo que o desenvolvedor foque mais nas características do negócio do que na infraestrutura da aplicação.

Spring Boot também possui uma excelente documentação que pode ser encontrada no seguinte endereço: <http://docs.spring.io/spring-boot/docs>, possui não apenas explicações detalhadas das funcionalidades do framework, mas também exemplos com código fonte.

Ao iniciar o desenvolvimento de uma aplicação com Spring Boot não é necessário realizar nenhuma configuração, o próprio framework já possui uma configuração padrão para

tudo. Essas configurações automáticas são realizadas com bases nas bibliotecas disponíveis no classpath da aplicação.

Em termos práticos, se o projeto estiver usando Maven (PIVOTAL SOFTWARE, 2015), ao adicionar a dependência spring-boot-starter-web automaticamente já será configurada uma aplicação web com o container Tomcat embarcado disponível na porta 8080. Para mudar a porta da aplicação é necessário apenas uma linha de código.

Justamente por ser bastante personalizável e pelo seu mecanismo de autoconfiguração com base no classpath, ao subir uma aplicação sem nenhuma biblioteca adicional o framework irá realizar uma configuração mínima, tornando-a bastante leve.

Com base nessas características, essa tecnologia foi escolhida para o desenvolvimento do micro serviço apresentado nesse trabalho. Tornando-o simples e leve.

B) Micro serviço desenvolvido

O micro serviço nomeado Scorecard Microservice foi desenvolvido para ser consumido pela aplicação RCMS (Researcher Content Management System) da UFSCar. Seu objetivo é realizar consultas na base do RCMS e tratar esses dados, transformando em indicadores estatísticos que podem ser utilizados para alimentar relatórios e gráficos como é o caso do sistema RCMS.

O micro serviço apresenta dois recursos RESTful (Representational State Transfer) e por serem expostos como Web Service Rest, os mesmos podem ser consumidos por qualquer outra aplicação que possua seu endereço.

O objetivo de um desses recursos é a obtenção de quantos trabalhos acadêmicos foram produzidos pelos integrantes do RCMS, quantidade por pesquisador e o total do grupo. E o outro mostra quantos trabalhos acadêmicos foram produzidos em cada ano pelos membros do RCMS.

O sistema RCMS consome o micro serviço diretamente, não tendo sido implementado um Gateway por se tratar de um serviço simples, o que traria mais malefícios do que benefícios (ALAGARASAN, 2015). Tal implementação poderá ser feita em trabalhos futuros para a realização de análises de quais benefícios pode-se obter com esse componente.

C) Arquitetura do projeto

Inicialmente o sistema RCMS foi desenvolvido como uma aplicação web monolítica, utilizando JAVA 8, Java EE 7, Spring 4, JSF 2, PrimeFaces 5.2 e Bootstrap 3.2.0. Todo o projeto e suas dependências foram empacotadas em um único artefato no formato .war, e implantado no servidor de aplicações Glassfish 4.

Como ilustrado na figura 1, neste artefato está contido todos os recursos web tais como páginas HTML, folhas de estilo e arquivos Javascript, bem como Web Services, componentes e serviços do Spring, entidades, toda a camada de persistência e acesso ao banco de dados.

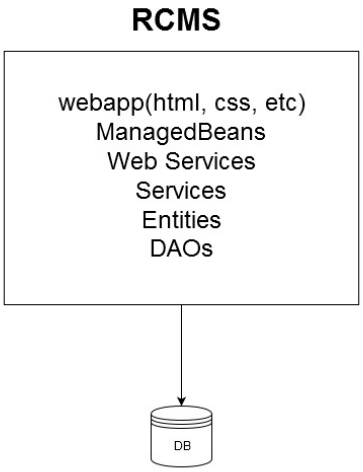

Figura 1. Arquitetura inicial do sistema.

É comum a modularização de sistemas em três camadas, separando em: camada web, serviços e persistência. De fato, o projeto está separado desta forma, porém não houve a necessidade de quebrá-lo em mais de um artefato considerando que a aplicação está implantada em um único servidor, os benefícios obtidos por esse processo seriam menores do que seus custos.

Porém com o crescimento do sistema chegando a mais de cinquenta páginas HTML, um pouco mais de cinquenta entidades, e algumas dezenas de DAOs, serviços e managed beans, a manutenção e implementações de novas funcionalidades se tornou cada vez mais difícil.

Esta foi uma oportunidade para o início do desenvolvimento de serviços utilizando a arquitetura orientada a micro serviços. A Figura 2 mostra como se apresenta a arquitetura do sistema após a implementação de um micro serviço.

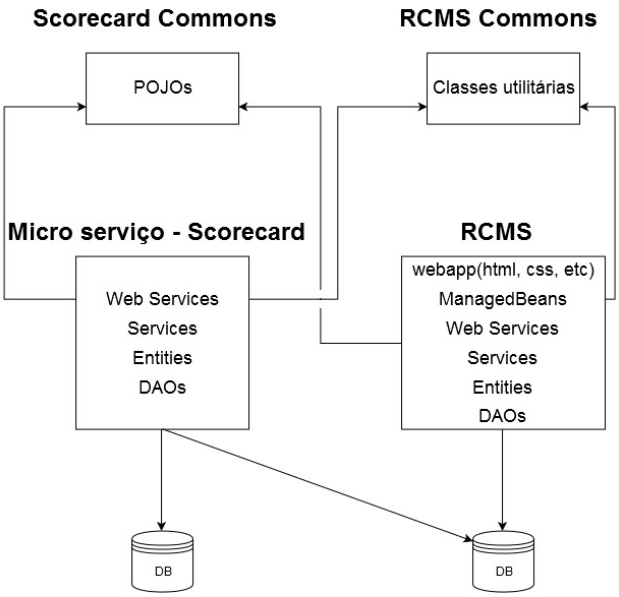

Figura 2. Arquitetura final do sistema.

O projeto RCMS foi modularizado permitindo a adição de novos módulos, com os detalhes desse processo explanado na próxima seção. Também foi realizado a extração de classes utilitárias, para que eles possam ser reutilizadas em outros projetos.

Foi criado um segundo projeto para o desenvolvimento do micro serviço Scorecard. Nele está contido o Web Service RESTful, serviços, entidades e DAOs. Diferente do RCMS, todos esses componentes ficam limitados ao escopo do Scorecard. Há apenas um Web Service e serviço, dois DAOs e algumas entidades para representar o modelo dados.

Também foi extraído para um projeto comum os POJOs (Plain Old Java Object) que representam os dados expostos pelo Web Service, esse é reusado tanto pelo Scorecard, quanto pelo RCMS que consome esse serviço.

O Scorecard possui sua própria base de dados, tornando-o assim independente do RCMS, caso o banco de dados do RCMS esteja indisponível, ainda assim é possível utilizar o micro serviço. Como os dados do RCMS são persistidos em sua própria base, há uma camada de integração no micro serviço que é responsável por realizar a leitura dos dados de uma base e cópia para a outra.

Parte da arquitetura do Scorecard foi realizada com base nos padrões apresentados por Gupta (2015).

É importante ressaltar que é gerado um artefato para o micro serviço separado do RCMS, permitindo que o primeiro seja implantado de maneira independente em relação ao segundo.

D) Processo de desenvolvimento

Antes do início do desenvolvimento do micro serviço foi realizado a modularização do sistema RCMS, com o objetivo de extrair componentes comuns que possam ser reutilizados por outros módulos, como por exemplo, o micro serviço que foi desenvolvido ou qualquer outro projeto futuro e também para facilitar a inserção de novos módulos, componentes e projetos no RCMS.

O projeto utiliza o Apache Maven para gerenciamento dependências, módulos e automação de build. Após a modularização o RCMS ficou com a seguinte estrutura:

- rcms: projeto onde está definido o Parent POM, com as declarações dos módulos, propriedades dos projetos, dependências genéricas e plug-ins.
- rcms-app: onde está o sistema monolítico RCMS, com componentes de persistência, serviços, web services, componentes de integração com o Lattes e recursos web estáticos, como páginas HTML, Javascript, folhas de estilo, etc.
- rcms-commons: projeto com componentes genéricos compartilhados entre diversos módulos do sistema, tais como, classes para manipulação de arquivos e validação de objetos.
- rcms-scorecard: nesse módulo encontra-se o micro serviço responsável pela obtenção de informações estatísticas.
- rcms-scorcard-commons: módulo com os POJOs utilizados para exposição de dados pelo web service RESTFUL e pelo próprio RCMS que consome essas informações, evitando assim a duplicação de código

Uma das vantagens da utilização de micro serviços é que eles são independentes um dos outros. Para isso é considerada a boa prática definida por Fowler e Lewis (2015), que cada micro serviço tenha sua própria base dados.

Como o objetivo desse micro serviço é tratar os dados do RCMS, é necessário que ele também faça consultas na base de dados do RCMS. Para torná-lo independente foi criado um segundo banco de dados exclusivo para o micro serviço e também um componente de integração para realizar a leitura na base do RCMS e escrita na base do Scorecard.

É realizado a leitura de cinco tabelas no banco de dados do RCMS, sendo elas: AutorProducao, CitacaoBibliografica, Pesquisador, Producao e Usuario. Já na base do Scorecard é salva apenas as informações tratadas que serão expostas pelos web services, por exemplo, quantidade de publicações realizadas pelo grupo de pesquisa.

Dessa forma, mesmo que o RCMS e seu banco não estejam disponíveis o micro serviço continua a operar normalmente, permitindo que futuras aplicações possam ser servidas por tais serviços.

Os web services expõem essas informações via JSON. Foi escolhido esse formato de dados pois ele é mais leve do que XML (EXtensible Markup Language) e consome menos recursos computacionais para ser serializado e desserializado tornando a comunicação de rede mais rápida.

Foi criado um client HTTP para que o RCMS possa consumir os dados expostos pelo micro serviço. Esse componente é responsável por realizar uma chamada síncrona no web service do scorecard e realizar a desserialização e conversão do JSON para objetos.

O RCMS também realiza um tratamento simples em caso de falha dos micro serviços. Esses dados são exibidos em gráficos em uma página do sistema, caso haja algum erro ou indisponibilidade do serviço, é exibido uma mensagem de erro com o endereço dos serviços indisponíveis.

IV. Resultados

As nomenclaturas monolítico e micro serviço não são dois extremos, porém formam um contraste interessante uma vez que muitas características de uma arquitetura diferem uma das outras.

Existem muitos sistemas que não se enquadram em nenhuma das duas arquiteturas, sejam eles por simplesmente utilizarem uma outra arquitetura ou por misturar as características de várias na tentativa de extrair o melhor de cada uma para um determinado cenário.

Com o protótipo apresentado neste trabalho foi possível levantar quais os pontos fortes e fracos de sistemas orientados a micro serviços, cenários onde pode-se extrair ao máximo de suas vantagens bem como onde recomenda-se não utilizá-lo por seus custos serem maiores que seus benefícios.

É também notável que a estratégia de implementar micro serviços em conjunto com sistemas monolíticos funciona de modo muito satisfatório, assim como sugerido por renomados

autores como Martin Fowler, Chris Richardson e Sam Newman.

Com a adoção de micro serviços é nítido o ganho que obtém-se com a velocidade de entrega de novas funcionalidades, especialmente em cenários onde tem-se implantado uma boa estratégia de entrega contínua.

É possível realizar alterações, manutenções, novas implantações entre outros procedimentos em pontos específicos do sistema sem comprometer ou mesmo parar outras funcionalidades que estão em produção.

"Com o tempo muitas vezes é difícil manter uma boa estrutura modular, tornando mais difícil de manter mudanças que deveriam afetar apenas um módulo dentro desse módulo." (FOWLER; LEWIS, 2015, tradução livre). Em tradicionais sistemas monolíticos, principalmente em sistemas de grande porte, há um certo receio em realizar alterações em funcionalidades antigas e muitas vezes acopladas a outras funcionalidades, pois uma mudança pode afetar diversos módulos, podendo gerar erros em vários pontos, sem contar o custo de testar novamente todas as partes afetadas.

Com micro serviços operando independentemente uns dos outros é possível alterar as regras de negócios pontuais, sem a necessidade de realizar toda a implantação do sistema novamente. Apenas o serviço que sofreu a alteração precisa ser reimplantado.

De maneira similar, a tecnologia e as necessidades dos clientes ou usuários mudam muito rápido, muitas vezes a arquitetura ou um framework que foi escolhido no começo do projeto já não é o mais apropriado.

Com micro serviços é possível adotar uma nova tecnologia e testá-la em partes do sistema sem comprometer o todo. Caso dê errado basta voltar apenas uma parte do sistema ao estado anterior. Se decidir por adotar essa nova tecnologia, pode-se implementá-la no resto do sistema aos poucos, ou até mesmo, ambas podem coexistir.

Desde que os micro serviços utilizem um protocolo de comunicação comum, por exemplo, HTTP, pode-se até mesmo utilizar linguagens de programação diferentes, aproveitando o melhor de cada uma para cada cenário.

Escalabilidade é outra grande vantagem dessa arquitetura. Considerando por exemplo, dois serviços, onde um utiliza bastante processamento de dados e um outro usa muitos recursos de leitura de disco. Pode-se implantar ambos em servidores diferentes, ou mesmo em alguma máquina na nuvem, com a infraestrutura apropriada para cada aplicação.

A forma com que será realizado a comunicação entre os micro serviços não deve ser negligenciada. Uma vez que a quantidade de chamadas remotas realizadas por diferentes serviços implantados em diferentes servidores podem tornar-se em um gargalo para qualquer sistema distribuído.

Na tentativa de minimizar o tráfego de rede recomenda-se a utilização de estrutura de dados mais simples. Um formato de dado muito comum e bastante utilizado é o JSON, utilizando como protocolo de comunicação o HTTP.

A complexidade de desenvolvimento de micro serviços em relação a sistemas monolíticos também é mais alta, esse problema tende a agravar-se especialmente em equipes não muito experientes com essa arquitetura.

Geralmente no início do desenvolvimento de um projeto é difícil ter uma visão clara de todas as fronteiras e responsabilidades dos serviços e componentes que serão desenvolvidos. O que pode levar a construção de micro serviços realizando mais processos do que deveria.

Justamente por essa dificuldade, não aconselhável iniciar um projeto totalmente orientado a micro serviços. Inicialmente, além de ser mais simples e produtivo o desenvolvimento de um sistema monolítico, os maiores ganhos com micro serviços são obtidos com sistemas grandes e complexos.

Sam Newman (2015b) recomenda a implementação de micro serviços em sistemas que já estão em desenvolvimento. Dessa forma é minimizado gradativamente os problemas gerados por grandes aplicações monolíticas e obtêm-se as vantagens dos micro serviços.

Nesse processo existem duas estratégias interessantes a serem seguidas. A primeira é o desenvolvimento de serviços futuros como micro serviços. A segunda é a decomposição de serviços existentes transformando-os em micro serviços.

Ambas as estratégias funcionam muito bem em sistemas existentes pois não são muito intrusivas, podendo ser realizadas gradativamente e em paralelo com o desenvolvimento de novas funcionalidades.

V. Trabalhos relacionados

Fernandez-Villamor, Iglesias e Garijo (2010) desenvolveram um ambiente para a descrição de micro serviços através proxies para outros serviços externos, onde a função do micro serviço é realizar a descrição de serviços. Algumas palavras chaves são usadas como parâmetros de entrada, é então realizada uma busca em serviços da Google e Flickr, que retornam as descrições dos serviços. Esta implementação mostra-se bastante efetiva, principalmente por seu alto nível de reusabilidade e independência.

O trabalho desenvolvido por Levcovitz, Terra e Valente (2015) descreve como identificar e definir micro serviços em sistemas monolíticos. O processo é divido em seis etapas: mapear os bancos de dados em subsistemas, criar um grafo de dependências, identificar os pares de serviços, selecionar os pares, identificar os candidatos a serem transformados em micro serviços e criação da API Gateway. Esse procedimento apresenta-se eficaz em casos onde os sistemas são grandes e complexos, o que torna sua análise bem mais complicada. Essa técnica foi realizada com sucesso em um grande sistema monolítico bancário. É ressaltado também que a decomposição de serviços para micro serviços pode ser feita de maneira incremental, assim como sugerido por Newman (2015a).

As principais diferenças do estudo realizado nesse trabalho é que o micro serviço foi desenvolvido para ser consumido por uma aplicação monolítica. Trata-se de um serviço relativamente simples, permitindo a realização mais fácil de uma análise conceitual sobre a arquitetura.

VI. Conclusões e trabalhos futuros

Nesse trabalho foi realizado um estudo teórico e prático sobre micro serviços, onde um micro serviço foi desenvolvido e implementado um em um sistema já existente analisando suas vantagens e desvantagens. O sistema monolítico RCMS consome o serviço diretamente, que é exposto por um web service.

Algumas vantagens da arquitetura de micro serviços são: escalabilidade, autonomia de serviços, possibilidade de uso de diversas tecnologias, tolerância a falha, diminuição da complexidade de sistemas grandes, etc.

O que torna essa arquitetura excelente para ser implementada em sistemas monolíticos cuja complexidade está crescendo. Dessa forma é possível, por exemplo, reduzir sua complexidade e testar novos padrões e tecnologias antes que a manutenção do sistema torne-se algo complexo e caro demais para ser realizada.

As principais desvantagens são: comunicação via rede e chamadas remotas podem aumentar o tempo de resposta, aumento de complexidade de desenvolvimento para projetos pequenos.

Nesse contexto, a utilização de micro serviços não é recomendada em sistemas pequenos, pois o custo de criação e manutenção de um ambiente para a utilização dessa arquitetura pode não ser viável para um sistema simples.

Como proposta de futuros trabalhos, pode ser realizado a análise e implementação de um API Gateway. Seu desenvolvimento é fortemente recomendado, especialmente em casos em que muitos micro serviços são expostos ou em que eles sirvam diferentes tipos de aplicações.

Através desse Gateway também é interessante a implementação de mecanismos de segurança especialmente em caso que os endpoints dos serviços possuam endereços públicos.

Uma extensão possível do trabalho é a decomposição de algum serviço existente do RCMS para mensurar quais os reais ganhos e custos dessa abordagem.

É importante que os arquitetos de sistemas conheçam diferentes arquiteturas e tecnologias e saibam a mais apropriada para cada cenário. Não há solução mágica que resolva todos os problemas.

Dessa forma esse trabalho apresenta mais uma forma de desenvolver softwares com a finalidade de aumentar a caixa de ferramentas de arquitetos e desenvolvedores de sistemas.

O micro serviço desenvolvido bem como o sistemas RCMS pode ser encontrado no seguinte endereço: <https://github.com/pedrofelipemm/rcms>.

Referências

ALAGARASAN, V. Seven Microservices Anti-patterns. Disponível em: <http://www.infoq.com/articles/seven-uservices-antipatterns>. Acesso em: 25 de ago. 2015.

FARCIC, V. Monolithic Servers vs Microservices. Disponível em: <http://www.javacodegeeks.com/2015/01/monolithic-servers-vs-microservices.html>. Acesso em: 20 de ago. 2015.

FOWLER, M; Lewis, J. Microservices. Disponível em: <http://martinfowler.com/articles/microservices.html>. Acesso em: 30 de ago. 2015.

FOWLER, M. Microservice Trade-Offs. Disponível em: <http://martinfowler.com/articles/microservice-trade-offs.html>. Acesso em: 12 de jul. 2015.

GUPTA, A. Refactor your Java EE application using Microservices and Containers . Microservice Design Patterns. Disponível em: <http://www.javacodegeeks.com/2015/04/microservice-design-patterns.html>. Acesso em: 10 de jul. 2015.

Levcovitz, A.; Terra, R.; Valente, M. C. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. III Workshop Brasileiro de Visualização de Software, Evolução e Manutenção, 2015.

LITTLE, M. Microservices and SOA. Disponível em: <http://www.infoq.com/news/2014/03/microservices-soa>. Acesso em: 30 de ago. 2015.

NEWMAN, S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2015a.

NEWMAN, S. Microservices For Greenfield? Disponível em: <http://samnewman.io/blog/2015/04/07/microservices-for-greenfield/>. Acesso em: 20 de ago. 2015b.

ORACLE. Java Platform, Enterprise Edition: The Java EE Tutorial. Disponível em: <https://docs.oracle.com/javaee/7/tutorial/overview003.htm>. Acesso em: 20 de set. 2015.

Pivotal Software. Building an Application with Spring Boot. Disponível em: <https://spring.io/guides/gs/spring-boot/>. Acesso em: 24 de set. 2015.

RICHARDS, M. Software Architecture Patterns. Sebastopol: O'Reilly Media, 2015.

RICHARDSON, C. Microservices: Decomposição de Aplicações para Implantação e Escalabilidade. Disponível em: <http://www.infoq.com/br/articles/microservices-intro>. Acesso em: 01 de jul. 2015.

WEIR, L. A Word About Microservice Architectures and SOA. Disponível em: <http://soacommunity.com/index.php/magazine/articles/236-articles-microservicesweir>. Acesso em: 13 de nov. 2015.

Woods, D. Criando micro serviços com Spring Boot. Disponível em: <http://www.infoq.com/br/articles/boot-microservices>. Acesso em: 24 de set. 2015.