

Migração de Sistemas Monolíticos para Microserviços

Guilherme L. D. Villaca¹

¹Universidade Estadual do Oeste do Paraná (Unioeste)
Cascavel – PR – Brazil

guidvillaca@gmail.com

Abstract. *Migrating monolithic systems to microservices has been a way for developers to solve old problems faced in the software lifecycle, such as scalability and maintainability. However, it is not simple to identify when these systems should migrate, and what the consequences are. In theory, applying microservices has a number of benefits compared to monolithic systems, especially when it is previously planned and starts development from scratch. In practice, issues like sharing a database, team relocation, infrastructure investment, developer team re-education are no easy task.*

Resumo. *Migrar sistemas monolíticos para microserviços tem sido uma maneira para desenvolvedores resolverem velhos problemas enfrentados na durante o ciclo de vida de um software, como escalabilidade e manutenibilidade. Porém não é simples identificar quando estes sistemas devem migrar, e quais as consequências disso. Na teoria, aplicar microserviços trás uma série de benefícios se comparado aos sistemas monolíticos, ainda mais quando se planeja previamente e inicia o desenvolvimento do zero. Na prática, questões como dividir uma base de dados, remanejamento de equipe, investimento em infraestrutura, reeducação da equipe de desenvolvedores, não é tarefa fácil.*

Keywords: sistemas monolíticos, microserviços, reúso de software, reengenharia, migração, banco de dados, manutenibilidade, escalabilidade.

1. Introdução

A técnica de desenvolvimento de software baseado em reúso é um processo de design que pode resultar em redução de tempo e custos de desenvolvimento além de aumentar a flexibilidade, manutenibilidade e confiabilidade do software. A principal razão para pensar em reúso é evitar trabalhos repetidos no desenvolvimento de software e usar do conhecimento e experiência adquiridos previamente e concentrar os esforços em partes mais críticas da aplicação. Seu foco é não necessitar mais desenvolver um software do zero [Yang et al.].

Reengenharia pode ser uma forma para atingir o reuso de software e para entender os conceitos ocultos no domínio da aplicação. Seu objetivo é obter e manter o conhecimento incluído nos sistemas legados, usando-o como base para a continuidade e evolução estruturada do sistema. O código legado tem lógicas programadas, decisões de projeto, requisitos de usuários e regras de negócio que podem ser recuperadas e reconstruídas sem perder a semântica [Garcia et al. 2004]

Os sistemas legados são tradicionalmente construídos de forma monolítica, ou seja, um

sistema composto por módulos que não são independentes da aplicação a que pertencem [Dragoni et al. 2017]. Para resolver as limitações dos sistemas monolíticos, os Microserviços estão sendo amplamente utilizados pela indústria por ser uma técnica que estimula uma melhor modularização e gestão através de serviços menores e autônomos [Carvalho et al. 2019]. Ter um sistema mais adaptável a mudanças é a melhor estratégia para modernizar sistemas legados [Kamimura et al. 2019].

O objetivo deste trabalho é verificar como seria a migração de sistemas monolíticos para microserviços em um ambiente com múltiplos sistemas interligados. Em um cenário como esse como deve ocorrer a migração? de uma forma mais gradual e a longo prazo? Somente as pequenas partes interligadas poderiam se tornar microserviços? Apenas novas funcionalidades devem ser desenvolvidas com a nova arquitetura? Desenvolver tudo do zero? Apesar de existir no estado da arte guias e processos para migração de sistemas únicos, não há uma definição sobre como proceder em um cenário como este.

2. Desenvolvimento

2.1. Revisão Bibliográfica

O desenvolvimento de sistemas pelas organizações geralmente segue um padrão onde o desenvolvimento começa pela solução específica de um problema e a partir daí novas soluções podem ser adicionadas, formando um ou mais sistemas maiores e complexos. Com o passar dos anos estes sistemas podem gerar diversos problemas para a equipe de desenvolvedores. Se a implementação de uma nova funcionalidade ou correção de uma falha tiver qualquer barreira, seja por código mal escrito, complexidade estrutural, rotatividade na equipe onde perde-se conhecimento de domínio, tem-se um dos seguintes problemas:

- **Manutenibilidade:** Que é a capacidade de um software ser modificado. Se um software não for construído com capacidade para lidar com a demanda de constantes mudanças, certamente irá consumir mais recursos e tornar a manutenção uma tarefa tediosa [Velmourougan et al. 2014].
- **Escalabilidade:** É a capacidade de um software para acomodar crescimento em termos de tamanho e complexidade do problema e em um segundo aspecto está relacionado a uma medida de eficácia quando usada em um contexto maior em escopo e complexidade do que o contexto para o qual foi originalmente projetado [Ibrahim et al. 2009].
- **Confiabilidade:** Eficácia do sistema quanto a estar livre de falhas por um período de tempo. É difícil atingir a confiabilidade de um software, pois quanto mais complexo, menos confiável ele se torna.[Pan 1999].
- **Qualidade:** Garantia de que o software atenda a todos os requisitos e a capacidade do software de atender às necessidades dos usuários.

Algumas das motivações para migração de sistemas monolíticos para microserviços é a delegação de responsabilidades, pois microserviços podem ser desenvolvidos por times de desenvolvedores diferentes e independentes, reduzindo sobrecarga de comunicação e necessidade de coordenação entre as equipes [Taibi et al.]. Ainda de acordo com [Taibi et al.], apesar dos custos iniciais do desenvolvimento com microserviços serem maiores, a longo prazo esse custo é menor, se comparado com sistemas monolíticos, devido a redução, principalmente, de manutenibilidade.

Um outro benefício na utilização de microserviços é a possibilidade de utilizar novas tecnologias. Com serviços independentes, não importa como esse serviço é implementado, portanto é possível escolher diferentes tecnologias que se adequem às necessidades específicas daquela funcionalidade. Isso também se aplica a banco de dados. Não há necessidade de se prender a uma tecnologia [Richter et al. 2017].

Um sistema monolítico, geralmente, tem uma base de dados e a recomendação é a separação dessa base onde cada microserviço tenha a sua própria base de dados. Portanto, ao migrar um monolítico para microserviços, deve-se também migrar a base de dados. Cada microserviço deve ser responsável pelos seus próprios dados, ou seja, independentes para que qualquer mudança não afete o sistema todo [Kholy and Fatatry 2019]. Mesmo que uma entidade seja comum a dois ou mais microserviços, ainda assim pode haver diferenças entre elas, o que corrobora a necessidade de manter a separação.

É comum haver uma resistência em equipes de desenvolvimento, isso ocorre devido a dificuldade em aceitar mudanças, seja de arquitetura e tecnologias, ou ainda a apego ao software desenvolvido [Taibi et al.].

2.2. Trabalhos Relacionados

Existem alguns trabalhos que focaram na migração de sistemas monolíticos para microserviços.

Na dissertação "Um guia para apoiar a migração de sistemas de software legados para arquitetura baseada em microserviços" o autor propõe um roteiro com etapas a se seguir para migração de sistemas legados, focando em identificar funcionalidades candidatas a serem modularizadas no contexto de microserviços. O autor também faz a aplicação em um teste real deste roteiro.

No trabalho "Modernização de sistemas legados para disponibilização em dispositivos móveis com arquitetura baseada em Microservices" o autor propõe uma migração para atender a distribuição e independência de um sistema legado para disponibilização em dispositivos móveis. Para o estudo de caso foi utilizado um sistema acadêmico.

Foi identificado outros trabalhos similares, porém todos eles aplicados a um sistema específico. Neste contexto a migração é mais simples comparada ao cenário que é o foco deste trabalho, onde vários sistemas devem ser afetados pela arquitetura de microserviços.

2.3. Metodologia

De acordo com [Runeson et al. 2012] em uma pesquisa de estudo de caso, nós escolhemos estudar um fenômeno que ocorre em um ambiente da vida real. Nossa "unidade de análise" pode ser algum aspecto de um projeto de engenharia de software, uma metodologia de engenharia de software e seu uso dentro de uma organização, o todo ou uma parte específica de um novo projeto em desenvolvimento ou manutenção de um projeto em andamento.

Hoje o Núcleo de Tecnologia da Informação (NTI) da UNIOESTE experiencia um cenário com vários sistemas monolíticos que sofrem por problemas de manutenibilidade, confiabilidade entre outros. O objetivo analisar a arquitetura existente e propor uma nova arquitetura que atenda aos principais conceitos de microserviços.

Inicialmente fizemos pesquisas em portais de artigos e periódicos para entender como estão os estudos atuais na área, buscamos pelos assuntos: *microservices*, *software reliability*, *legacy system*, sistemas monolíticos, *software reuse*, *software maintainability*.

3. Resultados Esperados

Após os estudos e análises da arquitetura existena no estudo de caso, espera-se a migração de uma aplicação, em todo ou em parte, para microserviços. Em um segundo momento, analisar os efeitos da migração, benefícios e dificuldades encontradas.

4. Cronograma de Execução

1. Levantar um histórico de todos os sistemas existentes no NTI, tendo um resumo do que cada sistema se propõe e as principais funcionalidades;
2. Identificar qual a interligação entre estes sistemas, onde é perceptível códigos duplicados, funcionalidades duplicadas em um ou mais sistemas e a dependência entre eles e identificar os principais candidatos à migração para Microserviços;
3. Fazer um levantamento da base de dados e quais as consequências da separação, sugerida pela arquitetura em Microserviços;
4. Aplicar em um sistema a migração para microserviços de forma gradual, num primeiro momento migrando a aplicação e a separação do banco de dados em uma segunda etapa.

Referências

- Carvalho, L., Garcia, A., Assunção, W. K. G., Bonifácio, R., Tizzei, L. P., and Colanzi, T. E. (2019). Extraction of configurable and reusable microservices from legacy systems: An Exploratory Study. pages 1–6.
- Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, pages 195–216.
- Garcia, V. C., Lucrédio, D., Do Prado, A. F., De Almeida, E. S., and Alvaro, A. (2004). Using reengineering and aspect-based techniques to retrieve knowledge embedded in object-oriented legacy system. *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI-2004*, pages 30–35.
- Ibrahim, H., Far, B. H., and Eberlein, A. (2009). Scalability Improvement in Software Evaluation Methodologies. *2009 IEEE International Conference on Information Reuse & Integration*, pages 236–241.
- Kamimura, M., Yano, K., Hatano, T., and Matsuo, A. (2019). Extracting Candidates of Microservices from Monolithic Application Code. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2018-Decem*:571–580.
- Kholy, M. E. and Fatatry, A. E. (2019). Framework for Interaction between Databases and Microservice Architecture. *IT Professional*, 21(5):57–63.
- Pan, J. (1999). Software Reliability. pages 1–9.
- Richter, D., Konrad, M., Utecht, K., and Polze, A. (2017). Highly-Available Applications on Unreliable Infrastructure: Microservice Architectures in Practice. *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017*, pages 130–137.
- Runeson, P., Host, M., Rainer, A., and Regnell, B. (2012). *in Software in Software*.

- Taibi, D., Lenarduzzi, V., and Pahl, C. Processes , Motivations , and Issues for Migrating to Microservices Architectures : An Empirical Investigation.
- Velmourougan, S., Dhavachelvan, P., Baskaran, R., and Ravikumar, B. (2014). Software development Life cycle model to improve maintainability of software applications. *2014 Fourth International Conference on Advances in Computing and Communications*, pages 270–273.
- Yang, X., Chen, L., Wang, X., and Cristoforo, J. A Dual-Spiral Reengineering Model for Legacy System.