INTERFACE TECNOLÓGICA

ARQUITETURA DE MICROSSERVIÇOS

MICROSERVICE ARCHITECTURE

Melissa Tidori da Conceição – melissatidori@gmail.com
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

Giuliano Scombatti Pinto – giuliano.pinto@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

DOI: 10.31510/infa.v18i2.1186
Data de submissão: 27/08/2021
Data do aceite: 03/11/2021
Data da publicação: 30/12/2021

RESUMO

Quando se trata de desenvolvimento de *software*, existem várias alternativas de arquiteturas, porém algumas das mais conhecidas para desenvolvimento de *software* são: Microsserviços e Monolítico. Assim, este artigo tem como objetivo apresentar o conceito de microsserviços, demonstrando sua estrutura e componentes, bem como as vantagens de se trabalhar com esta arquitetura, apontando seu ciclo de desenvolvimento desde a etapa da coleta de requisitos e formação de protótipos até a entrega final do *software*. A metodologia utilizada é a pesquisa bibliográfica. Os resultados apontam que há diversos desafios organizacionais que as empresas enfrentam ao optar por essa arquitetura, pois se a mesma não gerenciar e administrar os problemas que aparecerem e não souber liderar as equipes de desenvolvimento, a arquitetura acaba se tornando complexa a ponto de não garantir a entrega do *software*. Com isso, foi possível concluir que esta arquitetura, quando gerenciada com eficiência, garante sucesso para as equipes, diminuindo as falhas, obtendo excelentes resultados em menor tempo e garantindo qualidade no *software* que será desenvolvido.

Palavras-chave: Arquitetura de Microsserviços. Arquitetura Monolítica. Arquitetura de Software. Escalabilidade. Migração.

ABSTRACT

When it is about to software development, there are many alternative architectures, but the best known for software development are: Microservices and Monolithic. This article aims to present the concept of microservices, demonstrating its structure and components, as well as the advantages of working with this architecture, pointing out its development cycle from the stage of requirements gathering and prototyping to the final delivery of the software. The methodology used is bibliographic research. The results show that there are several organizational challenges that companies face when choosing this architecture, because if you don't know how to manage the problems that arise and does not know how to lead the development teams, the architecture ends up becoming complex to the point of not guaranteeing

delivery of the software. With that, it was possible to conclude that this architecture, when managed efficiently, guarantees success for the teams, reducing failures, obtaining excellent results in less time and guaranteeing quality in the software that will be developed.

Keywords: Microservice Architecture. Monolithic Architecture. Software Architecture. Scalability. Migration.

1 INTRODUÇÃO

Segundo Ballard (2016), a arquitetura de microsserviços é uma abordagem para criar aplicações em que cada função é denominada de serviço e é criada e implantada de maneira independente, ou seja, cada serviço pode falhar que não irá afetar os outros serviços.

Portanto, pode-se considerar que a arquitetura de Microsserviços é muito importante no contexto das aplicações desenvolvidas pelas empresas de *software* e a falta de conhecimento dela por parte de alguns desenvolvedores pode torná-la um tanto quanto complexa e difícil de ser mantida.

O objetivo deste trabalho é discorrer sobre a arquitetura de Microsserviços, abordando o funcionamento da arquitetura, juntamente com seus desafios organizacionais e os benefícios em utilizá-la, com uma breve abordagem sobre arquitetura monolítica.

A metodologia utilizada no presente artigo é a pesquisa bibliográfica, realizada em livros, artigos e websites.

2 ARQUITETURA DE SOFTWARE

Segundo Picoli (2018), arquitetura de software trata-se do mapeamento dos componentes de um software. Seria como projetar uma casa, na qual o arquiteto utiliza algumas ferramentas para criar cômodos de uma forma disposta, fazendo com que agrade o morador. No caso do software, seus elementos e componentes devem garantir a disposição da melhor forma também, garantindo o sucesso no projeto e que o usuário tenha uma experiencia otimizada.

A arquitetura de software é importante, pois ela é responsável pelo desempenho, a robustez e a capacidade de distribuição e manutenção de um sistema (BOSCH, 2000). Sendo assim, uma arquitetura de sistema é considerada uma influência dominante.

INTERFACE TECNOLÓGICA

2.1 Arquitetura Monolítica

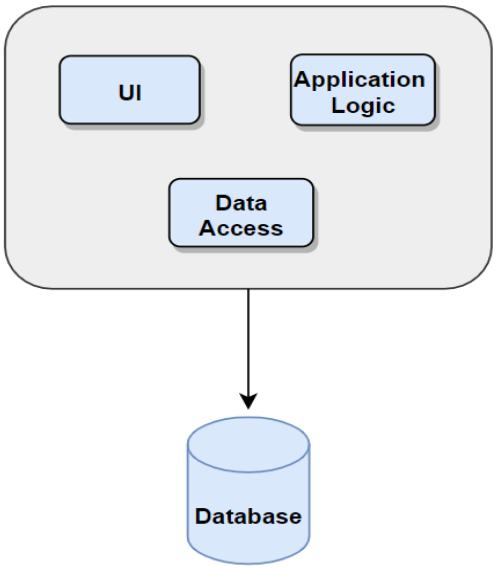
Um sistema de uma arquitetura monolítica é um modelo unificado composto por apenas uma peça. Esse sistema é projetado para ser independente, nas quais seus componentes são interconectados e interdependentes. Sendo assim, se algum componente do programa for atualizado, todo o resto deve ser reescrito (WIGMORE, 2016).

No entanto, há vantagens em utilizar uma arquitetura monolítica, pois sistemas monolíticos possuem melhor rendimento do que abordagens modulares, podendo ser mais fácil de testar e depurar a aplicação.

Aplicações monolíticas são consideradas simples de se desenvolver, pelo fato de serem mais simples e fáceis de implantar, pois o banco de dados atualiza todas as funcionalidades ao mesmo tempo. Esse sistema simplifica a reutilização de códigos dentro de seu próprio sistema (NEWMAN, 2020).

A Figura 1 representa uma aplicação monolítica, nas quais todos os componentes do sistema estão dentro de apenas uma aplicação acessando um único banco de dados.

Figura 1 - Arquitetura Monolítica.



Fonte: Elaborado pela autora (2021)

Ao contrário da arquitetura de Microsserviços, a arquitetura monolítica possui apenas um banco de dados, o que torna mais fácil de se desenvolver e dar manutenção. Em um sistema como este, é necessário apenas um *deploy*, ou seja, existe apenas um sistema para subir.

Segundo Richardson (2014), se uma aplicação monolítica se tornar muito grande, haverá problemas, pois dificulta o entendimento e manutenção de códigos para os desenvolvedores, além de não poder implementar novas funcionalidades, já que é preciso atualizar o sistema inteiro para isso, consumindo muito tempo, se tornando complexo e arriscado.

2.2 Arquitetura de microsserviços

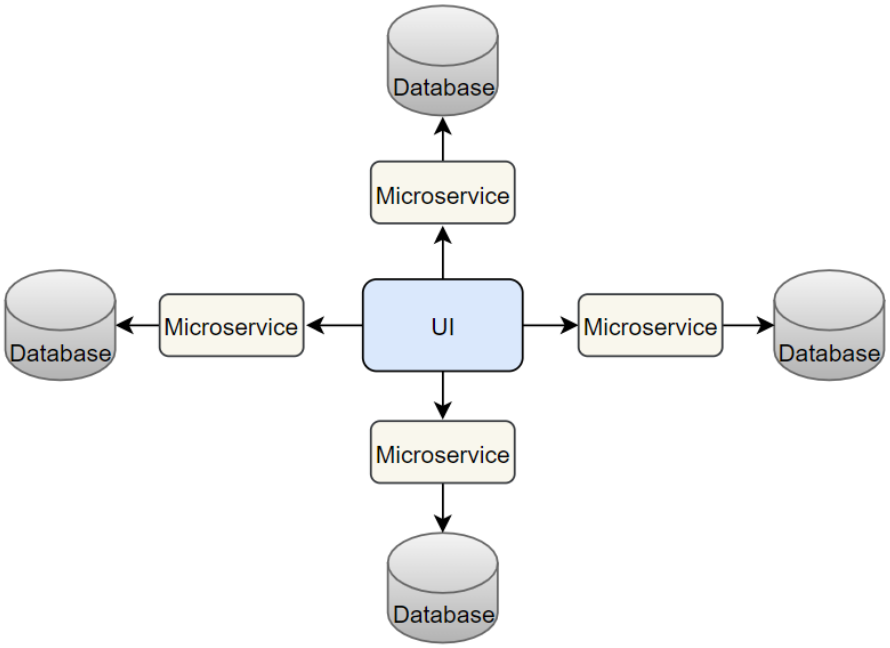
Microsserviços é uma arquitetura para a criação de aplicações, sendo elas desmembradas em componentes completamente independentes, com um alto grau de acoplamento, ou seja, são componentes separados que trabalham juntos, realizando as mesmas tarefas (NEWMAN, 2020).

Segundo Atkisson (2017), microsserviços são métodos de dividir um aplicativo em um conjunto de serviços pequenos e leves, que se comunicam através do protocolo de redes HTTP (Hypertext Transfer Protocol).

Essa arquitetura de software é um propulsor da inovação tecnológica, permitindo que sejam desenvolvidas e apresentadas em menor tempo, garantindo que se um serviço individual falhar, o resto da aplicação continuará funcionando normalmente.

A figura 2 representa uma aplicação de microsserviços, nas quais existem vários componentes e cada um possui seu próprio banco de dados.

Figura 2 – Arquitetura de Microsserviços



Fonte: Elaborado pela autora (2021)

Uma aplicação com microsserviços é desmembrada em componentes independentes, em que cada componente é um microsserviço. Sendo assim, cada microsserviço possui seu próprio banco de dados, o que torna essa arquitetura totalmente independente. (KANCZUK, 2020).

2.2.1 Ciclo de desenvolvimento

Um ciclo de vida de uma aplicação em microsserviços possui várias etapas, desde a coleta de requisitos e protótipos até a entrega final da aplicação em um ambiente de produção. Segundo Asher (2019) os 8 passos para o ciclo desenvolvimento, são:

1- **Conceito:** O conceito inicial de um ciclo de vida de um microsserviços é necessário para esclarecer e alinhar os requisitos do mercado, documentando todas as informações coletadas em alguma ferramenta.

2- **Especificação API:** É muito importante que o *design* de um protótipo da API (Application Programming Interface) seja adequado, pois um bom *design* torna o processo como um todo mais fácil. Para o desenvolvimento desse *design* existem linguagens específicas usadas para *designs* de APIs.

3- **Modelo API:** A etapa de modelagem de uma API é importante porque validar o *design* da API gera perspectiva do cliente, desbloqueando seus desenvolvedores, escrevendo códigos ou criando protótipos.

4- **Protótipo de Serviço:** O protótipo é necessário para permitir que as equipes validem seus códigos o mais cedo possível, evitando grandes problemas futuros, permitindo operações em nuvem, qualidade do sistema e segurança.

5- **Versão Alpha/Beta:** Um serviço não pode estar pronto para produção se não tiver sido utilizado por usuários reais. Com isso é mais fácil de encontrar problemas e *bugs* no sistema.

6- **Teste de Sistema:** Testar o sistema é uma das etapas mais importantes para encontrar erros, latências e vulnerabilidades de segurança. A garantia de qualidade do sistema é baseada em correções funcionais, escalabilidade, robustez e segurança. Por isso, testar o sistema antes de subir para produção é importante.

7- **Produção:** Mesmo com o sistema já em produção, ocorrem *releases* de atualizações, por isso ainda é importante fazer testes, melhorias, verificações do sistema e possuir provisionamento caso seja necessário.

8- **Fim do Serviço:** O fim do serviço é considerado a última etapa, nas quais os serviços da equipe não são mais necessários, pois o sistema já se encontra em produção funcionando corretamente, necessitando apenas de releases caso seja necessário.

2.2.2 Desafios Organizacionais

Para que haja a migração de uma arquitetura monolítica para microsserviços, é importante ter em mente que ocorrerão mudanças não somente nas aplicações, mas também no modo como as pessoas trabalham e suas culturas (ATKISSON, 2017). Assim, é relevante considerar os pontos a seguir:

- **Compilação:** Uma compilação pode gerar inúmeras dependências afetando os dados da aplicação. Por isso é necessário que haja dedicação de tempo para identificar essas dependências.

- **Testes:** Pelo fato de que uma falha nos testes de integração pode encadear outras falhas adiante, é importante saber que os testes podem se tornar mais difíceis e importantes.

- **Controle de Versão:** Em relação à atualização para versões novas, é muito provável que a compatibilidade com as versões anteriores seja rompida, entretanto é possível resolver este problema utilizando a lógica condicional e colocando várias versões ativas no ar.

- **Implantação:** As configurações iniciais de uma implantação sempre se tornam um grande desafio, por isso é necessário investir em automação por conta da complexidade da implantação manual do serviço.

- **Geração de logs:** Pelo fato de que um sistema de microsserviços é distribuído, cresce a necessidade de ter *logs* centralizados para unificar tudo, facilitando o gerenciamento da escala, pois logar em milhares de serviços para rastrear os *logs* se torna inútil.

- **Monitoramento:** é de extrema importância ter um monitoramento centralizado do sistema para identificar as fontes de problemas. Monitorar vários microsserviços é uma tarefa difícil, sendo necessário algo para agregar os dados do serviço.

- **Depuração:** uma depuração não funciona com centenas de serviços, então não há possibilidades de realizar depurações, pois a depuração remota não é viável em muitos microsserviços.

- **Conectividade:** As arquiteturas de um microsserviço deverão ter incluído um registro de descoberta de serviço, porque há pulos de rede entre cada serviço, adicionando latência e perda de dados entre eles. (ATKISSON, 2017)

3 PROCEDIMENTOS METODOLÓGICOS

A metodologia deste estudo se baseia em revisão bibliográfica, especialmente consulta de livros, levantamentos de artigos, trabalhos de conclusão de curso, dissertações e teses, além de materiais especializados na área da Tecnologia da Informação, com o intuito de verificar os trabalhos acadêmicos, além de entender toda a questão de uma arquitetura de microsserviços.

4 RESULTADOS

4.1 Vantagens da Arquitetura de microsserviços

Há muitas vantagens em se utilizar microsserviços em uma aplicação, pois ela é bastante flexível, garantindo a integração entre os módulos e obtendo sucesso no projeto, Segundo Ivory (2021) alguma delas são:

- **Produtividade:** com o uso de microsserviços, a produtividade da equipe aumenta, pois ocorre a divisão do programa em serviços independentes, ou seja, cada módulo pode ser codificado de forma independente, facilitando e acelerando o desenvolvimento do software.

- **Performance:** pelo fato de que os módulos são independentes, fica mais fácil de aproveitar vantagens da computação paralela, pois quando se separa uma aplicação em diferentes serviços, surge a possibilidade de otimizar o uso dos núcleos de processamento, aumentando a performance.

- **Escalabilidade:** com o uso de microsserviços, surge a facilidade da escalabilidade dos projetos, pelo fato de que cada componente pode ser escalável de forma individual e quando for necessário, é possível a adição de novos componentes sem precisar alterar o resto do sistema.

- **Resiliência:** resiliência basicamente é a capacidade de impedir que alguns erros pontuais virem falhas no sistema, permitindo se recuperar do "*bug*" e retornando ao estado funcional.

- **Flexibilidade:** os módulos de microsserviços se comunicam através de APIs, sendo possível utilizar diversas tecnologias e linguagens de programação para cada serviço, com maior flexibilidade para as equipes desenvolverem os componentes do software.

- **Integração:** o uso de um microsserviço facilita a coordenação de diferentes equipes de desenvolvimento pelo fato de que cada time desenvolve os módulos de forma independente e simultânea. Com isso, a tomada de decisões fica acelerada.

- **Reutilização:** como os serviços são independentes, é possível reutilizá-los em outras finalidades, ou seja, o módulo de um projeto pode ser reaproveitado em outros softwares da empresa. Com isso, se ganha a aceleração dos processos de desenvolvimento e evita o desperdício de recursos.

4.2 Desvantagens

Pelo fato de uma arquitetura de microsserviços ser formada por partes autônomas, o conjunto do software pode se tornar complexo.

Não é indicado migrar para microsserviços se não há um sistema que seja complexo para gerenciar, como por exemplo, a arquitetura monolítica. Com problemas de complexidade, surgem dificuldades com a governança do projeto, pois a governança pode acabar deixando a desejar. Mudanças de hábitos e culturas são pontos extremamente difíceis de serem concretizados, principalmente em empresas de grande porte ou equipes antigas que já têm sua forma de trabalho enraizada (FOWLER, 2017).

Com uma arquitetura de microsserviços também podem surgir problemas com segurança, ou seja, se os microsserviços não estiverem bem definidos, documentados e estandardizados por meio de APIs seguras, surgem problemas com a segurança, o que torna o software sujeito a ataques.

5 CONCLUSÃO

Esse estudo procurou avaliar conceitos básicos e mais específicos sobre microsserviços, abordando o que é, para que serve, como funciona, as vantagens, desvantagens e os desafios para se implementar a arquitetura.

Após todo o estudo, pode-se concluir que a arquitetura de microsserviços é muito complexa de ser implementada, principalmente se o time que for aplicá-lo não for maduro ou experiente com a arquitetura, fazendo com que o projeto talvez fracasse, tendo muito retrabalho pela frente e prejuízos com ele.

Também é importante ter em mente que essa arquitetura não vai resolver todos os problemas de um projeto, tudo dependerá do contexto e requisitos do sistema, mas tudo indica que essa arquitetura é ideal quando se espera uma alta demanda, ou seja, muitas requisições nos serviços e resiliência (quando algo falhar, o sistema não pode parar, pararia apenas o serviço que falhou).

Contudo, quando a arquitetura é gerenciada e implementada de maneira correta, a chance de sucesso aumenta e, consequentemente, o projeto tende a dar certo, diminuindo as falhas, obtendo melhores resultados em menor e tempo e garantindo uma boa qualidade no software que foi desenvolvido.

REFERÊNCIAS

ATKISSON, Brian. **The Truth about Microservices**. 2017. Disponível em: https://developers.redhat.com/blog/2017/05/04/the-truth-about-microservices?extIdCarryOver=true&sc_cid=701f2000001OH6fAAG. Acesso em: 18 maio 2021.

ASHER, David. **A Lifecycle for microservices**. jun. 2019. Disponível em: https://pt.slideshare.net/ProductCampBoston/a-lifecycle-for-microservices. Acesso em: 27 jun. 2021.

BALLARD, Deon. **Intro to Microservices**. 24 mar. 2016. Disponível em: https://www.redhat.com/pt-br/blog/intro-microservices. Acesso em: 28 jun. 2021.

BOSCH, Jan. **Design and Use of Software Architectures:** Adopting and evolving a product-line approach. 1ª. ed. Pearson Education Limited, 2000.

IVORY. **Microsserviços: o que são e 7 benefícios de implementá-los.** 12 mar. 2021. Disponível em: https://www.ivoryit.com.br/blog/microsservicos. Acesso em: 25 jun. 2021.

FOWLER, Susan. **Microsserviços Prontos Para a Produção**: Construindo Sistemas Padronizados em uma Organização de Engenharia de Software. 1ª. ed. Novatec Editora, 2017.

KANCZUK, Daniel. **O que são microsserviços e como funcionam?.** 16 jun 2020. Disponível em: https://blog.geekhunter.com.br/arquitetura-de-microsservicos-x-arquitetura-monolitica/. Acesso em: 30 jun. 2021.

NEWMAN, Sam. **Migrando Sistemas Monolíticos Para Microsserviços**: Padrões Evolutivos Para Transformar seu Sistema Monolítico. 1ª. ed. Novatec Editora, 2020.

PICOLI, Cristian. **Arquitetura de software:** Saiba o que é e qual a sua importância. 2018. Disponível em: https://atmdigital.com.br/saiba-o-que-e-arquitetura-de-software-e-qual-a-sua-importancia/. Acesso em: 27 jul. 2021.

RICHARDSON, Chris. **Microservices:** Decomposição de Aplicações para Implantação e Escalabilidade. 2014. Disponível em: <https://www.infoq.com/br/articles/microservices-intro>. Acesso em: 04 jun. 2017.

WIGMORE, Ivy. **Monolithic architecture**. 10 maio 2016. Disponível em: https://whatis.techtarget.com/definition/monolithic-architecture. Acesso em: 28 jun. 2021.