UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARIANA ARAÚJO LUCENA

Revisão Sistemática da Literatura:

Os desafios encontrados na migração de uma Arquitetura Monolítica para uma orientada a Microsserviços

CAMPINA GRANDE - PB

2021

MARIANA ARAÚJO LUCENA

REVISÃO SISTEMÁTICA DA LITERATURA:
OS DESAFIOS ENCONTRADOS NA MIGRAÇÃO DE UMA ARQUITETURA MONOLÍTICA PARA UMA ORIENTADA A MICROSSERVIÇOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel ou Bacharela em Ciência da Computação.

Orientador: Professor João Arthur Brunet Monteiro

CAMPINA GRANDE - PB

2021

L935r Lucena, Mariana Araújo.
Revisão sistemática da literatura: os desafios encontrados na migração de uma arquitetura monolítica para uma orientada a microsserviços / Mariana Araújo. – 2021.

10 f.

Orientador: Prof. Dr. João Arthur Brunet Monteiro.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Arquitetura de software. 2. Revisão sistemática de literatura. 3. Arquitetura monolítica. 4. Computação em nuvem. 5. Integração contínua. 6. Entrega contínua. 7. Arquitetura de microserviços. I. Monteiro, João Arthur Brunet. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

MARIANA ARAÚJO LUCENA

Revisão Sistemática da Literatura: Os desafios encontrados na migração de uma Arquitetura Monolítica para uma orientada a Microsserviços

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel ou Bacharela em Ciência da Computação.

BANCA EXAMINADORA:

Professor João Arthur Brunet Monteiro

Orientador – UASC/CEEI/UFCG

Professora Patrícia Duarte de Lima Machado

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de outubro de 2021.

CAMPINA GRANDE - PB

ABSTRACT

Even with the great advance of technology and the contribution of large companies to adhere to or transitioning their monolithic codes to microservices, There are no set of rules or steps in the literature that can be followed to make the transition easier. The aim of this work is to prospectively the main challenges faced, such as: application security, data storage, management, monitoring and resource consumption. In addition to suggesting steps that will facilitate the migration. In order to achieve this accomplishment a literary collection that was related to the theme was obtained, the works were selected and the relevant data extracted. We conclude that for small applications or applications that are still in the beginning of development, the indicated architecture is the monolithic one, but with the application growth, it is desirable that the transition to microservices happens in the best possible way. This requires good test coverage before transitioning, as well as having knowledge of the system and how the relationship between the components will occur, a good process based on the ability to get feedback, such as building complete and informative logs , in addition to an agile methodology offering continuous integration and delivery.

Keywords: Software Architecture; Transition; Components.

Revisão Sistemática da Literatura: Os desafios encontrados na migração de uma Arquitetura Monolítica para uma orientada a Microsserviços

Mariana Araújo Lucena
mariana.lucena@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

João Arthur Brunet Monteiro
joao.arthur@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Mesmo com o grande avanço da tecnologia e a contribuição de grandes empresas em aderir ou transicionar seus códigos monólitos para microsserviços, ainda não há na literatura um conjunto de regras ou passos que possam ser seguidos para tornar a transição mais fácil. Este trabalho tem como objetivo investigar os principais desafios enfrentados, como: segurança da aplicação, armazenamento dos dados, gerenciamento, monitoramento e consumo de recursos. Além de sugerir etapas que irão facilitar a migração. Para alcançar este objetivo foi obtido um acervo literário que estivesse relacionado com o tema, os trabalhos foram selecionados e os dados relevantes, extraídos. Concluímos que para aplicações pequenas ou que ainda estão no início do desenvolvimento a arquitetura indicada é a monolítica, porém, com o crescimento da aplicação é desejável que a transição para microsserviços aconteça da melhor forma possível. Para isso, é necessário uma boa cobertura de testes antes de realizar a transição, assim como, ter conhecimento do sistema e como ocorrerá o relacionamento entre os componentes, um bom processo baseado na capacidade de obter feedback, como a construção de logs completos e informativos, além de uma metodologia ágil oferecendo integração e entrega contínua.

Palavras-chave

Arquitetura de software, Transição, Componentes.

1. INTRODUÇÃO

Uma aplicação monolítica é aquela na qual toda a base de código está contida em um só lugar, ou seja, todas as funcionalidades estão definidas no mesmo bloco [19]. Hoje é comum as pessoas associarem *softwares* com arquiteturas monolíticas a um sistema legado (sistema antigo e com tecnologias ultrapassadas) mas dependendo do contexto, por exemplo, para uma aplicação que está no início da vida, não muito grande, que não possui equipes multi territoriais, esta pode ser uma boa escolha arquitetural. Em contrapartida, à medida que os sistemas crescem, começam a aparecer problemas como dificuldades de entender o código, alto acoplamento entre módulos, aumento do tempo de implantação (*deploy)* e dependência de tecnologia a longo prazo.

Diante desses desafios, vê-se a necessidade em realizar a troca para uma arquitetura baseada em microsserviços, que são conjuntos de serviços interdependentes que se comunicam entre si organizados sobre a capacidade de negócio, geralmente com *deploy* automatizado e controle descentralizado de linguagens de dados [7]. A ideia é que os módulos do monolítico sejam refatorados em pequenas unidades que executam em programas independentes da maneira mais desacoplada possível, permitindo que seus processos de construção (*build*), implantação (*deploy*) e escalabilidade (*scaling*) sejam independentes.

Assim sendo, não é surpresa que grandes corporações como Google, Ebay, Netflix, Amazon, Spotify, PayPal, Uber e LinkedIn tenham realizado sérios esforços para mover suas aplicações monolíticas para uma arquitetura orientada a microsserviços. Apesar dos benefícios decorrentes, esse novo modelo arquitetural não afasta um conjunto de desafios advindos da tarefa de migração [2]. O problema comum de todas as companhias é que, identificar quais componentes das aplicações legadas podem ser convertidas, de forma coesa, em unidades de microsserviço independente, é uma tarefa complexa e manual que requer análises em várias dimensões na arquitetura do sistema e cai fortemente no conhecimento e experiência de quem os está fazendo [14].

As questões de pesquisa elaboradas para nortear o presente estudo foram: Quais são os principais desafios enfrentados no momento de refatorar aplicações de arquitetura monolítica para microsserviços? Quais os principais passos comumente utilizados pelos autores ao realizar a transição entre arquiteturas? Com base nelas, este estudo se concentra na realização de um compilado de passos que foram realizados nas transições estudadas, assim como, os principais desafios encontrados ao realizar a citada tarefa. Por ser uma área relativamente nova e pela dificuldade em realizar a transição percebe-se que ainda há falta de conhecimento na literatura sobre como o processo ocorre.

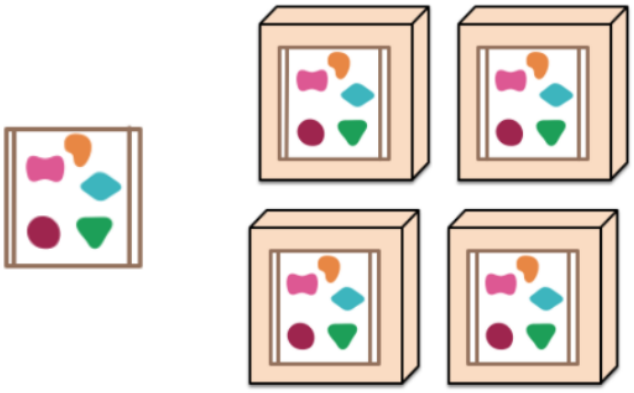
2. FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta a fundamentação teórica para o entendimento deste trabalho. Estão descritos abaixo os principais conceitos encontrados durante a revisão bibliográfica.

2.1 Principais conceitos

Monolítico: Monolítico é por definição semelhante a um monolito, este que por sua vez tem a característica de ser um monumento ou obra constituída por um só bloco. Análogo ao monolito, a arquitetura monolítica é um sistema único, não dividido, que roda em um único processo, uma aplicação de software em que diferentes componentes estão ligados a um único programa dentro de uma única plataforma. Geralmente, essa arquitetura subdivide o código em camadas, sendo elas: apresentação, que é onde o usuário visualiza o sistema, negócio, onde contém a lógica da aplicação e dados, que contém as classes responsáveis pela conexão com o sistema de armazenamento de dados. Uma aplicação monolítica coloca toda a sua funcionalidade em um único processo e escala replicando o monólito em vários servidores, como mostra a figura 1.

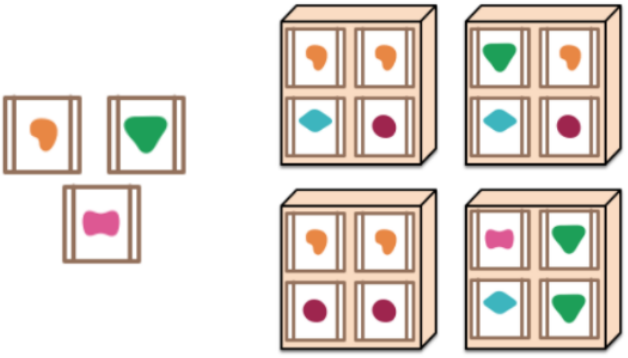
Figura 1 - Arquitetura Monolítica



Fonte: LEWIS; FOWLER (2014).

Microsserviços: Apesar do termo ter surgido apenas em 2011, a proposta já havia sido discutida em 1978 por McIlroy, Pinsen e Tague, quando publicaram um artigo com o contexto de situações mais flexíveis e em escalas diversas, que pode ser pensado como base para este formato de arquitetura. Microsserviços, que é um estilo arquitetônico originado de Arquiteturas Orientadas a Serviços (SOAs) [12], consiste em uma abordagem arquitetônica nativa de cloud na qual um único aplicativo é composto de muitos componentes ou serviços menores que são implementáveis de forma independente e têm acoplamento fraco. Nesta arquitetura cada funcionalidade é colocada em um serviço separado e, quando dimensionado, os serviços são distribuídos entre os componentes, replicando conforme surge a necessidade, como mostra a figura 2.

Figura 2 - Arquitetura de Microsserviços



Fonte: LEWIS; FOWLER (2014).

Computação em nuvem *(Cloud computing)*: A *Amazon Web Services (AWS)*, uma das maiores plataformas desse segmento, define computação em nuvem como a entrega de recursos de TI sob demanda por meio da Internet com definição de preço de pagamento conforme o uso. Em vez de comprar, ter e manter datacenters e servidores físicos, você pode acessar serviços de tecnologia, como capacidade computacional, armazenamento e bancos de dados, conforme a necessidade, usando um provedor de nuvem. A disponibilização de recursos de hardware traz vários aspectos, como: a sensação de que os recursos computacionais disponíveis são infinitos e disponíveis sob demanda, a possibilidade de alocar mais recursos a qualquer instante e de forma rápida, bem como desalocar, e pagar apenas pelo uso desses recursos [1].

Integração contínua: Segundo *Thought Works,* empresa global em tecnologia de informação, integração contínua (CI) é uma prática de desenvolvimento que requer que os desenvolvedores integrem o código em um repositório compartilhado várias vezes ao dia. Para isso não existe a necessidade de muitas ferramentas, em tese, mas alguns auxílios que são padrão em práticas de desenvolvimento são requeridos.

Entrega contínua *(Devops)*: Baseada em processos e ferramentas que automatizam tarefas como desenvolvimento, testes e *deploy* em produção, a entrega contínua é destinada a liberar componentes de *softwares* mais rapidamente e com maior frequência, ajudando a reduzir custos, tempo e riscos. Baseia-se em práticas enxutas e ágeis previamente estabelecidas [6].

3. METODOLOGIA

Para a realização deste trabalho foi utilizada uma revisão sistemática de literatura, que tem como propósito agregar experiências em uma determinada extensão de diferentes estudos para alcançar um objetivo de pesquisa específico [3], a fim de explorar de maneira metodológica e organizada o conhecimento existente nas áreas relacionadas.

Foi utilizado também, o método qualitativo, que tem o objetivo de demonstrar dados, indicadores e tendências [15], formulado através da análise de trabalhos científicos sobre o tema. Aplicamos os procedimentos: a formulação das questões de pesquisa; busca nas bases de dados científicas através das palavras-chave selecionadas; avaliação da qualidade dos trabalhos selecionando-os através dos critérios de inclusão e exclusão; extração dos dados e interpretação dos resultados [11].

3.1 Questões de pesquisa

- **Objetivo 1:** Quais são os principais desafios enfrentados no momento de refatorar aplicações de arquitetura monolítica para microsserviços?
- **Objetivo 2:** Quais os principais passos comumente utilizados pelos autores ao realizar a transição entre arquiteturas?

3.2 Seleção de estudos primários

Na fase de coleta dos materiais foram utilizados os principais bancos de dados científicos, são eles: a Biblioteca Digital Brasileira de Teses e Dissertações, Google Acadêmico e Portal de Periódicos da CAPES.

As pesquisas nas bases de dados, citadas anteriormente, foram feitas utilizando a seguinte *String* de busca: {“microsserviço” OR “monólito” OR “refatoramento” OR “migração”} AND {“microservice” OR “monolithic” OR “monolith” OR “refactoring” OR “migration”} AND {“monolithic architecture” OR “software architecture” OR “migration to microservice” OR “refactoring a monolith” OR “monolith transition to microservice” OR “from monolith to microservices"} AND {“arquitetura monolítica” OR “arquitetura de software” OR “migração para microsserviço” OR “refatoramento monolito” OR “transição de monolito para microsserviço”}

Na etapa de refinamento dos materiais escolhidos, todos os artigos encontrados foram selecionados para leitura do título e resumo. Após esta etapa, alguns estudos foram desconsiderados para a etapa de extração dos dados devido ao fato de não apresentarem os pré-requisitos apresentados posteriormente. Para facilitar o refinamento foram desenvolvidos critérios de inclusão e exclusão. Os critérios de inclusão podem ser encontrados a seguir:

C1. Ano de publicação maior ou igual 2010

C2. Está disponível em inglês ou português

C3. Descrever a transição de um software monolítico para microsserviço

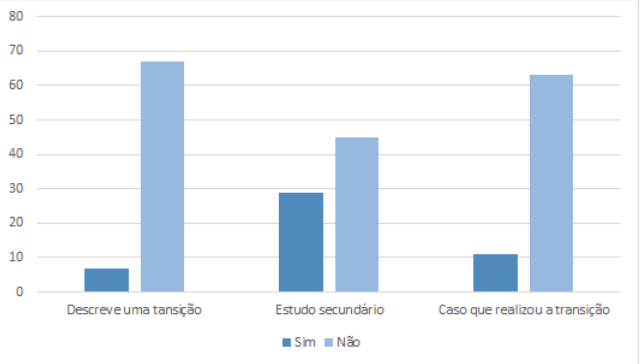
C4. Ser um estudo secundário sobre ferramentas de apoio ao processo de transição de um software monolítico para microsserviço

C5. Descrever um caso que realizou a transição entre as arquiteturas monolítica para microsserviços

Já para os critérios de exclusão foram consideradas as negações das proposições C1 a C5.

Ao final da etapa de seleção foi obtido um acervo literário constituído por 74 artigos, dos quais 30 foram aceitos, ou seja, satisfizeram o primeiro e segundo critérios e, pelo menos, um dos outros critérios. A figura abaixo mostra a relação entre os critérios de inclusão 3, 4 e 5.

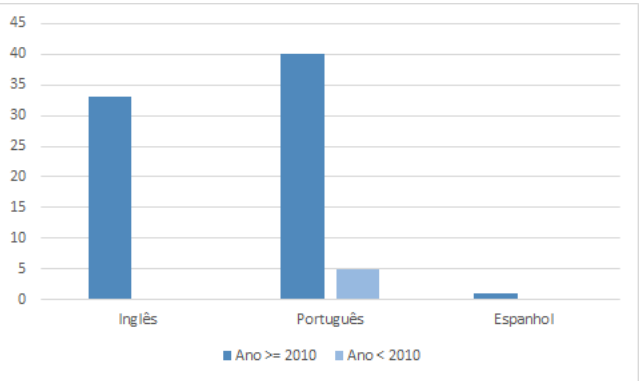
Figura 3 - Relação dos artigos selecionados



Fonte: Elaborado por Mariana Araújo (2021).

Dos trabalhos aceitos, cerca de 64,5% são dissertações de graduação ou pós-graduação e o restante, artigos científicos. Do acervo total, 44 artigos foram rejeitados, já que não satisfizeram os critérios um ou dois ou não satisfizeram todos os outros subsequentes. Destes, 6,75% foram excluídos por serem trabalhos publicados anteriormente ao ano de 2010 e 2,70% não estavam no idioma inglês ou português, como mostra o gráfico de colunas abaixo, que relaciona as variáveis ano de publicação e idioma.

Figura 4 - Relação entre as variáveis ano de publicação e idioma dos artigos



Fonte: Elaborado por Mariana Araújo (2021).

3.3 Extração dos dados

Para a extração dos dados foi utilizada a ferramenta MAXQDA, que tem como objetivo auxiliar na análise de dados qualitativos e métodos mistos em pesquisas acadêmicas, científicas e comerciais. Nele foram criados códigos que auxiliaram a identificar os principais pontos dentro dos artigos estudados. Foram destacados 273 fragmentos, distribuídos entre os códigos criados para facilitar o mapeamento de assuntos relevantes ao tema.

4. RESULTADOS

Esta seção apresenta os resultados da comparação dos 30 artigos analisados, dentre esses, 6 mencionaram os principais desafios

encontrados ao realizarem as refatorações de aplicações monolíticas existentes para a arquitetura de microsserviços e 17 descreveram o passos que a maioria dos autores consideraram para realizar a citada transição.

Determinar o nível de modularidade, coesão e acoplamento dos componentes é um dos aspectos mais importantes que a arquitetura deve transmitir. O objetivo de uma arquitetura de software é ter a quantidade certa de módulos que tenham a coesão correta, sejam reutilizáveis, fáceis de entender e manter [17].

Não é interessante iniciar um novo projeto já com a arquitetura de microsserviços, mesmo que exista a certeza que o sistema será robusto o suficiente. Uma das maneiras mais comuns de migração para arquitetura de microsserviços é a partir de um monólito, onde, em quase todos os casos de microsserviços bem-sucedidos, foram quando um software monolítico ficou muito complexo e grande [7].

- Cobertura de testes: Miika Kalske, em seu trabalho "Challenges When Moving from Monolith to Microservice Architecture (2017)" recomenda que para realizar a transição é necessário saber que há novos desafios de refatoração e organizacionais. Refatorar o software existente pode ser uma tarefa amedrontadora. Para realizá-la com sucesso, primeiramente, uma boa cobertura de testes é necessária. Caso contrário, existe a possibilidade de que, durante a introdução de microsserviços, novos bugs possam atingir as funcionalidades existentes.

- Divisão dos serviços: a maior dificuldade na divisão do software em componentes é a especificação do tamanho de cada componente. Existem interpretações variadas de qual deve ser o tamanho de um serviço e a falta de padrões adequados para ajudar com as especificações tornam essa tarefa ainda mais difícil [22]. Dentro de monólitos, é possível identificar candidatos a componentes para migração para microsserviços com base no mapeamento das dependências entre bancos de dados, funções de negócios e fachadas [12]..

- Relacionamento dos componentes: O pesquisador Raphael Henrique, cita em seu artigo, "Modelo de migração do monolítico ao microsserviço (2019)", que após o levantamento e armazenamento de funcionalidades candidatas, é importante refinar o relacionamento entre as partes componentes da aplicação. Com a realização do estudo do sistema, é possível identificar como é o fluxo da informação da entrada, processamento, transformação e saída para outros componentes ou usuários. Um dos métodos que possibilitam essa visão das funcionalidades e o fluxo de dados entre elas, é um diagrama de fluxo de dados que pode ser usado para representar sistemas monolíticos da perspectiva de processos de negócio (CHEN et al, 2017).

- Incorporação de mecanismos de registro: Sara Hassan, et. al. no seu artigo intitulado "Microservice transition and its granularity problem: A systematic mapping study (2020)", sugerem a técnica de microsservitização. Para eles, é importante que mecanismos robustos de registro e descoberta em arquiteturas de microsserviço sejam incorporados para garantir um registro atualizado dos microsserviços "ativos" atualmente.

- Expertise dos desenvolvedores: outro aspecto importante a se considerar é o conhecimento a respeito da aplicação da equipe de desenvolvimento, a execução de um design de microsserviço exige que toda a equipe entenda as complexidades que o design possui [7]. Uma compreensão desalinhada de como a arquitetura funciona e como os diferentes serviços devem ser divididos levará a uma aplicação ineficiente e difícil de manter [21].

- Gerenciamento de dados: Manuel Mazzara et. al. mencionam no artigo "Microservices: Migration of a Mission Critical System", que para o banco de dados, quanto menos for alterado maiores serão as chances de proporcionar o que é conhecido por Zero Downtime, que consiste em fazer o deploy sem que a aplicação, que já encontra-se em produção, pare de funcionar, a fim que a troca fique transparente para o usuário. Uma das estratégias para se alcançar esse objetivo é o Blue-Green Deployment [10], onde uma cópia do banco será criada, alterada e rotulada com uma cor. Um deles é rotulado de green e o outro de blue. Diante disto, apenas um dos bancos terá permissão para responder pelas requisições, enquanto o outro será manipulado com o propósito de sofrer as alterações programadas. Quando as alterações forem concluídas chaveiam-se as demandas para o banco que foi atualizado tendo apenas a preocupação de sincronizar os dados manipulados entre o banco que estava em operação com o banco que estava em manutenção.

- Software poliglota: com microsserviços é possível misturar várias linguagens, estruturas de desenvolvimento e tecnologias de armazenamento de dados. Uma vez que um sistema é composto de vários serviços independentes, os desenvolvedores têm a liberdade para escolher com qual tecnologia querem trabalhar dentro de cada componente. Isso permite escolher uma ferramenta apropriada para cada serviço, usando a linguagem e bibliotecas mais adequadas para determinados tipos de problemas (CONTINO, 2014; HAMPSHIRE; NEWMAN, 2015; RICHARDSON, 2016).

- Mapeamento de execução: um bom processo é baseado na sua capacidade de obter feedback do sistema [9]. Por isso, é aconselhável que a equipe de desenvolvimento construa logs completos e informativos, se baseando em métricas, para que favoreçam a observalidade dos microsseviços, através deles será possível extrair informações relevantes sobre o estado atual da aplicação. Facilitando o monitoramento dos microsserviços e tornando possível realizar ações preventivas, corretivas ou de melhorias.

- Integração Contínua e Entrega Contínua: O mestre em computação, Luis Heustákio cita em seu artigo "Uma abordagem de migração para arquitetura de microservices a partir de aplicações monolíticas em produção (2017)", que para aderir ao estilo arquitetural baseado em microsserviços é recomendado implantar, além de uma infraestrutura de servidores, um sistema de integração entre as equipes de desenvolvimento dos diversos componentes. Como o ciclo rápido de mudança é um dos principais pontos dos microsserviços, as implantações precisam ser rápidas e suaves. A entrega contínua significa que a equipe de desenvolvimento pode fazer todos os tipos de alterações no software com rapidez e segurança, sem interromper outras partes do programa. O monitoramento contínuo dá aos desenvolvedores feedback relacionado ao desempenho e possíveis erros nas operações [7].

- Devops: Taylor Rodrigues em "Método de migração de sistemas monolíticos legados para a arquitetura de microsserviços (2021)" defende a proposta DevOps porque ela visa facilitar a

comunicação e colaboração entre as equipes de desenvolvimento (Dev) e operações (Ops) de modo a garantir um fluxo contínuo de entrega de software. Em geral, isto é alcançado por meio de conceitos como integração contínua (CI) e entrega contínua (CD). O uso de culturas ágeis como *DevOps*, práticas ágeis (entrega contínua) e uso de arquitetura nativas para nuvem como microsserviços permite que as empresas entreguem *softwares* com mais valor de negócio, melhor experiência do usuário e redução do *time-to-market* para seus clientes.

- Computação em nuvem: Luis Heustákio também cita em seu trabalho "Uma abordagem de migração para arquitetura de microservices a partir de aplicações monolíticas em produção (2017)", que toda aplicação sofre cargas de trabalho diferentes ao longo de seu funcionamento e que o grande poder da computação em nuvem é aumentar ou diminuir os recursos de acordo com a demanda de uso, cobrando apenas pelo que foi usado. Além disso, a computação em nuvem propõe uma alta disponibilidade de recursos computacionais (processamento, armazenamento, memória e rede). Logo, percebe-se que o desenvolvimento de microsserviços em nuvem torna o dimensionamento automático de recursos muito mais fácil e econômico.

Marina Medeiros no artigo "Análise do Processo de Migração de Sistemas de Arquitetura Monolítica para Microsserviço (2019)", recomenda que os próximos 4 pontos sejam seguidos para que a migração obtenha sucesso.

- Métricas: todos os times devem entrar em acordo sobre quais são as métricas necessárias para que se possa utilizar um serviço, e é muito importante que todos possam ver como está a saúde dos serviços em um único lugar.

- *Rollback:* é importante se preparar para o caso do seu serviço ter problemas de performance, e é preciso ter em mãos mecanismos para agir rápido sobre esse problema;

- Independência dos serviços: manter detalhes de implementação privados é uma forma de não criar dependência de outros serviços nesses detalhes, e assim garantir a independência de implementação de todos.

- Versionamento de código: quando for preciso realizar mudanças em seus serviços, tenha mecanismos para informar futuras depreciações, e dê tempo para que os outros serviços se atualizem de suas mudanças.

- Desafios:

Os pontos que exigem uma atenção maior no momento de realizar a transição podem ser divididos em três categorias: design, desenvolvimento e operação. Durante a fase de design, as maiores preocupações estão relacionadas à arquitetura e segurança da aplicação. Na fase de desenvolvimento, a atenção precisa está voltada aos serviços, armazenamento e testes. A evolução individual e a próprio passo de cada serviço acaba por gerar um problema na manutenção e evolução dos cenários de teste. Diferentes serviços necessitam de diferentes ambientes de execução, o que aumenta a complexidade dos testes. Aliado a isso está o fato de que devem ser testados fluxos que dependem da execução de diferentes serviços. Cada serviço pode ser testado de forma isolada, porém problemas podem emergir repentinamente após a interação entre os serviços. Uma vez que o aplicativo está em operação, as preocupações estão associadas ao gerenciamento de aplicativos, monitoramento e consumo de recursos [20].

Como cita André Stangarlin no artigo "Uma Abordagem para Testes de Desempenho de Microservices (2016)", o custo operacional é outro fator desafiador, quando comparados a uma única aplicação monolítica, os microservices representam um grande custo extra para instalação das aplicações nos servidores. O problema cresce potencialmente com o aumento no número de serviços. Há também um aumento na complexidade operacional e de desenvolvimento. O desenvolvimento demanda um maior conhecimento de ferramentas em múltiplos campos de conhecimento. Nesse sentido, a equipe de desenvolvimento deve dominar as tecnologias para instalar, otimizar, executar e manter seus serviços dentro dos requisitos exigidos.

André também cita que há esforço duplicado, já que em muitas ocasiões, diferentes serviços dependem das mesmas funções. O problema é que os serviços utilizam diferentes tecnologias e muitas são incompatíveis. Sendo assim, não é possível compartilhar uma biblioteca entre os serviços.

5. CONCLUSÃO

Devido a facilidade de desenvolvimento e acesso a ferramentas, o modelo monolítico é amplamente recomendado, como a melhor opção para iniciar implementações de sistemas, mesmo aqueles que pretendem escalar. Porém, na medida que o sistema cresce, existe a necessidade em fazer a transição para microsserviços, visto que, os problemas encontrados em utilizar um monólito para aplicações grandes começam a surgir, como a dificuldade em adicionar novas funcionalidades, os deploys são cada vez mais demorados e imprevisíveis, as ações para resolução de problemas e quebras são demoradas, entre outros.

O foco da pesquisa era unificar as tarefas realizadas pela maioria dos autores ao realizar a transição entre as arquiteturas supracitadas e quais os principais pontos de atenção que devem ser levados em consideração. Conseguir alcançar os benefícios de uma arquitetura de microsserviço é um desafio devido à complexidade da arquitetura, já que não existe manual ou regras claras de como realizar a transição. Tarefas como uma boa cobertura de testes no monólito antes de realizar a transição, a forma em que os serviços serão divididos e como vai ocorrer o relacionamento entre eles, o conhecimento dos desenvolvedores a respeito do código, dentre outros, devem ser consideradas no momento da mudança. Os principais alertas estão no contexto de segurança da aplicação, armazenamento dos dados, testes de sistema, gerenciamento de aplicativos, monitoramento e consumo dos recursos.

A maior parte do foco, é claro, estará no lado técnico dos desafios, mas as organizações não devem esquecer a lei de Conway, que indica que as organizações projetam sistemas que refletem sua própria estrutura de comunicação. Portanto, os desafios técnicos e organizacionais precisam ser resolvidos para ter sucesso com os microsserviços.

6. REFERÊNCIAS

[1] ARMBRUST, M. e. a. Above the Clouds: A Berkeley View of Cloud Computing. [S.l.], 2009

[2] BALALAIE, A.; ABBASHEYDARNOORI; JAMSHIDI, P. Microservices architecture enables devops: An experience report on migration to a cloud-native architecture. In: . [S.l.]: IEEE Software, 2016.

[3] BUDGEN, David; BRERETON, Pearl. Performing systematic literature reviews in software engineering. In: 28TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. EUA, N.Y. Proceedings of the 28th international conference on software engineering (icse '06). EUA, N.Y: IEEE Xplore, 2016.

[4] CHEN, Rui; LI Shanshan; LI, Zheng. From Monolith to Microservices: A Dataflow-Driven Approach. In: 24th Asia-Pacific Software Engineering Conference (APSEC). China, Nanjing, 2017. Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC). China, Nanjing, IEEE Xplore, 2017.

[5] CONTINO. The Benefits Of MicroServices, 2014. Disponível em: <http://contino.co.uk/the-benefits-of-microservices/> Acesso em: 1 set. 2021.

[6] EBERT, C., Gallardo; HERNANTES, J., & SERRANO, N. (2016). DevOps Obtido de Banco de dados SCOPUS. Disponível em <https://www.scopus.com>. Acesso em: 15 ago. 2021.

[7] FOWLER, M., LEWIS, J. Microservices. Disponível em: http://martinfowler.com/articles/microservices.html. Acesso em 26 ago. 2021.

[8] HAMPSHIRE, Andy. Getting Started With Microservices. Disponível em: <https://dzone.com/refcardz/getting-started-with-microservices> Acesso em: 1 set. 2021

[9] IKAKLI, Nadareishvili; MITRA, Ronnie; MCLARTY, Matt e AMUNDSEN, Mike: Microservice Architecture: Aligning Principles, Practices, and Culture. página 146, 2016. 10, 18, 20, 32, 34, 65

[10] JONG, M. de; DEURSEN, A. van; CLEVE, A. Zero-downtime sql database schema evolution for continuous deployment. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track. Piscataway, NJ, USA: IEEE Press, 2017. (ICSE-SEIP '17), p. 143–152. ISBN 978-1-5386-2717-4.

[11] KITCHENHAM, B., PRETORIUS, R., BUDGEN, D., BRERETON, O.P., TURNER, M., NIAZI, M., LINKMAN, S. Systematic literature reviews in software engineering – A tertiary study, 52, 792-805. 2010

[12] LEVCOVITZ, A., TERRA, R. e VALENTE, MT, "Para uma técnica para extração de microsserviços de sistemas empresariais monolíticos," em III Workshop de Visualização, Evolução e Manutenção de Software (VEM), 2015, pp. 97–104.

[13] MACKENZIE, Matthew; LASKEY, Ken; MCCABE, Francis; BROWN, Peter; METZ, Rebekah Metz e HAMILTON, Booz Allen. Modelo de referência para arquitetura orientada a serviços 1.0.OASIS Standard, 12, 2006.

[14] MAZLAMI, Genc; CITO Jürgen; LEITNER, Philipp. Extraction of Microservices from Monolithic Software Architectures, IEEE International Conference on Web Services (ICWS), Honolulu, HI, 2017, p. 524-531.

[15] MINAYO, Maria Cecília de Souza. O desafio do conhecimento. 11 ed. São Paulo: Hucitec, 2008.

[16] NEWMAN, Sam. Building Microservices. Sebastopol: O'Reilly Media, 2015. p. 280.

[17] PRESSMAN, R. (2010). Ls. Engenharia de software: a abordagem de um profissional (7ª ed.). Nova York: McGraw-Hill.

[18] RICHARDSON, Chris. Microservices: Decomposição de Aplicações para Implantação e Escalabilidade, 2016. Disponível em: <http://www.infoq.com/br/articles/microservices-intro> Acesso em: 1 set. 2021 A.

[19] SANTOS, Lucas. Microserviços : dos grandes monólitos às pequenas rotas. 2017. Disponível em: <https://medium.com/trainingcenter/microservi%C3%A7os-dos-grandes-mon%C3%B3litos-%C3%A0s-pequenas-rotas-adb70303b6a3>. Acesso em: 30 ago. 2021.

[20] SOLDANI, J., TAMBURRI, D. A., & VAN Den Heuvel, W. -J. (2018). The pains and gains of microservices: A systematic grey literature review

[21] TAIBI, Lenarduzzi and PAHL (2017) conducted an empirical investigation in the motivations, issues and processes related to microservice architecture migrations (Taibi, Lenarduzzi, & Pahl, 2017).

[22] ZIMMERMANN, O. (2017). Princípios de microsserviços: abordagem ágil para o desenvolvimento de serviços e desdobramento, desenvolvimento