ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL DOS SANTOS KRUG

ENTENDENDO VANTAGENS E DESVANTAGENS EM APLICAÇÕES MONOLÍTICAS VERSUS O USO DE MICRO-SERVIÇOS

Porto Alegre
2024

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ENTENDENDO VANTAGENS E DESVANTAGENS EM APLICAÇÕES MONOLÍTICAS VERSUS O USO DE MICRO-SERVIÇOS

DANIEL DOS SANTOS KRUG

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Afonso Henrique Corrêa de Sales

Porto Alegre
2024

Ficha Catalográfica

K94e Krug, Daniel dos Santos

Entendendo vantagens e desvantagens em aplicações monolíticas versus o uso de micro-serviços / Daniel dos Santos Krug. – 2024.
94.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Afonso Henrique Corrêa de Sales.

1. Arquitetura de micro-serviços. 2. Aplicações monolíticas. 3. Tomada de decisão de arquitetura de software. I. Sales, Afonso Henrique Corrêa de. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

DANIEL DOS SANTOS KRUG

ENTENDENDO VANTAGENS E DESVANTAGENS EM APLICAÇÕES MONOLÍTICAS VERSUS O USO DE MICRO-SERVIÇOS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado(a) em 26 de Março de 2024.

BANCA EXAMINADORA:

Prof. Dr. Igor F. Steinmacher (Northern Arizona University)

Prof. Dr. Rafael Prikladnicki (PPGCC/PUCRS)

Prof. Dr. Afonso Henrique Corrêa de Sales (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Dedico este trabalho a todos meus colegas e ex-colegas que de alguma forma ajudaram a moldar o profissional que sou hoje.

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”
(Martin Fowler)

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a minha esposa Naiara por sempre me incentivar a continuar estudando e buscar melhorar sempre; meu filho Lucas por me mostrar um novo mundo doce e desafiador; e minha filha Júlia por mostrar que amor não se divide, e sim se multiplica. Agradeço também ao Professor Afonso Henrique Corrêa de Sales, meu orientador, por todas as oportunidades e pelos ensinamentos.

ENTENDENDO VANTAGENS E DESVANTAGENS EM APLICAÇÕES MONOLÍTICAS VERSUS O USO DE MICRO-SERVIÇOS

RESUMO

A arquitetura de micro-serviços surgiu como uma alternativa à arquitetura de monolitos. Com os monolitos, as aplicações são desenvolvidas em blocos inteiros que se comunicam internamente, gerenciam seus dados geralmente em um único banco de dados, e cada nova funcionalidade exige o *deploy* da aplicação como um todo. Por outro lado, os micro-serviços dividem a aplicação em blocos menores com responsabilidades únicas, utilizando mecanismos de comunicação leves e gerenciando seus próprios dados. Esta nova arquitetura tem diversas vantagens, mas também apresenta algumas desvantagens. A partir da compreensão dessas vantagens e desvantagens, o objetivo principal desta pesquisa é identificar como as duas arquiteturas têm sido utilizadas nas práticas profissionais e como a academia pode ajudar no entendimento dos problemas associados à utilização da arquitetura de micro-serviços e propôr soluções para os mesmos. Desta forma, o objetivo deste trabalho é a geração de um guia de recomendações para ajudar na compreensão de quando uma arquitetura pode ser mais apropriada em relação a outra. Para atingir este objetivo, foi realizada uma revisão multivocal da literatura e levantamento com especialistas através de *survey*.

Palavras-Chave: Arquitetura de Micro-serviços, Aplicações Monolíticas, Tomada de Decisão de Arquitetura de *Software*.

UNDERSTANDING ADVANTAGES AND DISADVANTAGES IN MONOLITHIC APPLICATIONS VERSUS THE USE OF MICROSERVICES

ABSTRACT

Microservices architecture emerged as an alternative to monolith architecture. With monoliths, applications are developed in entire blocks that communicate internally, manage their data usually in a single database, and each new feature demands the deployment of the application as a whole. On the other hand, microservices splits the application into smaller blocks with unique responsibilities, using lightweight communication mechanisms and managing their own data. This new architecture has several advantages, but it also has some disadvantages. Based on an understanding of these advantages and disadvantages, the main goal of this research is to identify how the two architectures have been used in professional practice and how academia can help to understand the problems associated with the microservice architecture and propose solutions to them. The aim of this work is to generate a guide of recommendations to help understand when one architecture may be more appropriate than the other. To achieve this goal, a multivocal literature review and a survey of experts were carried out.

Keywords: Micro-services Architecture, Monolithic Applications, Software Architecture Decision Making.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

CD – *Continuous Delivery*

CI – *Continuous Integration*

DDD – *Domain Driven Design*

DEVOPS – Combinação das palavras *development* e *operations*

ERP – *Enterprise Resource Planning*

JSON – JavaScript Object Notation

JWT – *JSON Web Tokens*

RLM – Revisão de Literatura Multivocal

SDN – *Software-Defined Networking*

TLS – *Transport Layer Security*

VPN – *Virtual Private Network*

SUMÁRIO

1. INTRODUÇÃO

A Arquitetura de Micro-serviços é uma arquitetura de desenvolvimento de *Software*, na área da Engenharia de *Software*, que consiste em dividir as aplicações que antigamente eram blocos inteiros - ou monolitos [42] - em serviços menores capazes de funcionar e de se comunicar independentemente uns dos outros. Cada um desses serviços pode ser escrito em linguagens diferentes, e se comunicam através de mecanismos leves (APIs). Além disso, podem gerenciar suas próprias base de dados e ter seus modelos de dados específicos para o seu propósito. Micro-serviços são comumente utilizados com contêineres como *Docker* e *Kubernetes* [7], o que permite sua execução isolada, facilitando o seu *deploy* em diferentes ambientes de execução.

Alguns dos primeiros estudos acadêmicos relacionados à micro-serviços datam da metade da década de 2010 [53, 48] enquanto os primeiros estudos referenciando as diferenças entre micro-serviços e monolitos datam de 2016 [31]. Em 2015, Brian Carter [13] menciona as evoluções obtidas ao separar monolitos em aplicações menores, utilizando o termo micro-arquitetura ou micro-aplicações, referindo-se ao oposto de monolitos. James Lewis e Martin Fowler [19] discutem o tema desde março de 2014. Entretanto não há um claro indício de quando o termo foi cunhado pela primeira vez.

A arquitetura de monolitos precede a arquitetura de micro-serviços e é menos complicada: É mais simples de desenvolver, testar e fazer *deploy* [14, 54], visto que a aplicação está contida em um único módulo. Porém, aplicações possuem uma tendência natural de crescimento em tamanho e eventualmente uma aplicação utilizando esta arquitetura de monolitos torna-se muito grande. Quando isso acontece, as desvantagens da arquitetura de monolitos se sobrepõem às suas vantagens, por exemplo: códigos complexos e incompreensíveis tornam muito mais difícil o desenvolvimento de novas funcionalidades; a escalabilidade é limitada a réplicas da aplicação sendo instanciadas em novos servidores; dificuldade no *deploy* onde qualquer pequena mudança requer o *deploy* da aplicação inteira; entre outras [1].

A adoção da arquitetura de micro-serviços na engenharia de *software* soluciona alguns desses problemas relacionados aos monolitos. Ela acarreta em diversas vantagens, mas também algumas desvantagens. Estudos recentes [47, 56] focam em elencar estas, a fim de entender como a indústria está adotando esta arquitetura, quais são os benefícios que encontram e as dificuldades que necessitam superar.

As principais vantagens listadas são: entregas mais rápidas, aumento da escalabilidade e maior autonomia [35]. Estas vantagens estão associadas aos benefícios de limite consistente entre módulos - que tem origem em práticas como o DDD [37] e seus *bounded contexts* - ou seja, cada módulo deve ter um único propósito, o que reforça a estrutura modular, sendo esta importante para times maiores de desenvolvimento; *deploy*

independente, ou seja, por serem serviços separados uns dos outros, é menos provável que alguma falha no lançamento de um serviço cause maiores danos à aplicação como um todo e; a variedade de tecnologias, podendo assim diversificar as linguagens de programação utilizadas no desenvolvimento do software, assim como diferentes *frameworks* de desenvolvimento e tecnologias de armazenamento de dados [19].

Por outro lado, o uso da arquitetura de micro-serviços também está associada a desvantagens. Alguns estudos buscam entender como os praticantes podem ter o maior entendimento sobre elas e qual a solução que deve ser adotada. Um exemplo de desvantagem é a segurança [57]. No modelo tradicional, o desenvolvedor e sua equipe necessitam se preocupar com a segurança da aplicação como um todo. Entretanto, no modelo de micro-serviços, cada módulo da aplicação necessita da sua própria segurança e esta por vezes acaba sendo negligenciada [57]. Outra desvantagem associada ao uso de micro-serviço é a performance da rede [36], uma vez que as comunicações entre serviços necessitam ser feitas através do envio de chamadas de APIs pela rede, enquanto na arquitetura de monolitos essa comunicação é feita em memória. Além disso, há também o problema da consistência de dados [30]: Na arquitetura de monolitos, os dados da aplicação são armazenados de forma centralizada, normalmente em um único banco de dados. Na arquitetura de micro-serviços, cada módulo é responsável pelo gerenciamento dos seus dados e modelos de dados. Apesar de isso poder ser uma vantagem, pode trazer problemas maiores de garantia de consistência de informação entre os serviços [38], onde os serviços precisam garantir a correta execução uns dos outros para poder validar suas transações.

Estas desvantagens citadas são amplamente estudadas e podem ser mitigadas antes e durante o tempo do desenvolvimento das aplicações que usam a arquitetura de micro-serviços. Contudo, o fato de a equipe de desenvolvimento ter essas preocupações a mais quando utiliza esta arquitetura, torna o desenvolvimento mais complexo e custoso: Mais esforço é necessário para o desenvolvimento da camada de comunicação entre os serviços; mais desenvolvimento é necessário para a segurança de cada um dos módulos; mais serviços, ou formas de prevenir falhas, são necessários para garantir a consistência de dados, entre outros. Entender os problemas associados à arquitetura de micro-serviços é o motivador desta pesquisa.

É importante ressaltar que a pesquisa apresentada não visa desincentivar o uso de micro-serviços, visto que quando esta arquitetura é bem aplicada pela equipe de desenvolvimento, as vantagens que esta arquitetura acarreta fazem com que seja a melhor escolha para a maioria dos casos.

1.1 Proposta de Pesquisa

Diante do observado nos estudos apresentados pela fundamentação desta pesquisa e, principalmente pelo que se mostrou nos exemplos de vantagens do uso de micro-serviços, há comprovada vantagem na adoção desta arquitetura. Entretanto, observa-se também que a adoção desta arquitetura pode trazer complexidades que não são exploradas pela comunidade científica.

A partir do entendimento anterior e considerando que até o momento não foi encontrado nenhum trabalho na literatura com foco nos problemas associados à arquitetura de micro-serviços e suas possíveis soluções, a proposta desta dissertação foi de realizar uma pesquisa de cunho exploratório para entender estes problemas efetivamente e como se pode minimizá-los, propondo um guia de recomendações de boas práticas dado as características do *software*.

1.2 Questão de Pesquisa

A questão principal de pesquisa deste trabalho foi estruturada da seguinte maneira:

“Como apoiar o desenvolvedor praticante da arquitetura de micro-serviços na hora da decisão sobre sua utilização e como mitigar possíveis problemas associados à esta arquitetura?”

1.3 Objetivos da Pesquisa

1.3.1 Objetivo Geral

Diante do exposto, o objetivo geral desta pesquisa foi de tentar entender quando o uso da arquitetura de monolitos é satisfatória no desenvolvimento de uma aplicação e quando a utilização de micro-serviços é a melhor opção.

1.3.2 Objetivos Específicos

OBJ1: Obter o conhecimento sobre como as práticas acadêmicas contribuem para uma melhor compreensão das duas arquiteturas propostas;

OBJ2: Obter uma visão detalhada das práticas diárias de uso das arquiteturas na indústria;

OBJ3: Elaborar um guia para auxiliar na tomada de decisão entre a utilização de arquiteturas monolíticas ou micro-serviços, fornecendo boas práticas que visam prover alternativas aos problemas relacionados ao uso da arquitetura de micro-serviços.

1.4 Estrutura do Documento

Neste contexto, esta dissertação se estrutura conforme segue: a Seção 2 apresenta a metodologia delineada para esta pesquisa. A Seção 3 apresenta os resultados obtidos com a análise das etapas metodológicas propostas. Na Seção 4 são discutidos os resultados deste estudo, bem como apresenta-se um guia de boas práticas que visam prover alternativas aos problemas relacionados ao uso da arquitetura de micro-serviços, para tomada de decisão entre o uso das arquiteturas, além das possíveis ameaças à validade deste estudo. E, por fim, a conclusão é apresentada na Seção 5 e os trabalhos futuros a serem desenvolvidos.

2. METODOLOGIA

As etapas metodológicas a seguir estão diretamente relacionadas aos objetivos específicos desta pesquisa e seguem as etapas apresentadas na Figura 2.1. Estas etapas contribuíram diretamente com o entendimento inicial do tema e subsidiou o planejamento e a realização da etapa seguinte. Inicialmente, foi conduzida uma Revisão de Literatura Multivocal (RLM) proposto por Garousi, Felderer e Mäntylä [21], com foco na execução do *snowballing* [55] - para a obtenção de estudos acadêmicos sobre o tema da pesquisa - assim como nas diretrizes da *Grey Literature Review*, que considera o conhecimento disponível em fontes não acadêmicas, que são relevantes para a indústria de *software*.

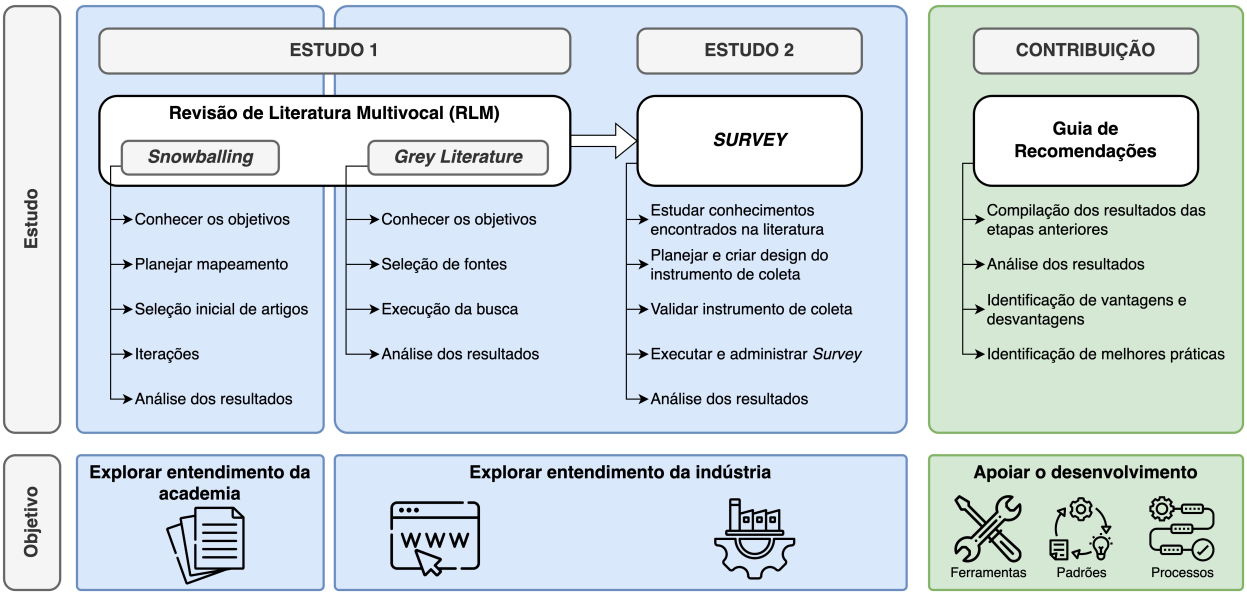

Figura 2.1 – Etapas da metodologia.

2.1 Condução do *Snowballing*

Com o objetivo de contemplar o entendimento da comunidade acadêmica sobre os problemas em aplicações quando é feita a escolha pela arquitetura de micro-serviços, foi executada a metodologia do *snowballing*. *Snowballing* [55] é o nome dado à metodologia de estudo sistemático que preconiza, a partir de uma seleção inicial de artigos, a utilização de suas referências e citações para uma revisão sistemática. Ao uso de referências e citações dá-se o nome, respectivamente de *backward snowballing* e *forward snowballing*. Cada seleção de artigos que partem de referências e citações de uma seleção anterior é considerada uma “iteração”, conforme Figura 2.2. As iterações são realizadas

até que não haja mais artigos relacionados ao tema pesquisado e que contemplem os critérios de inclusão e exclusão.

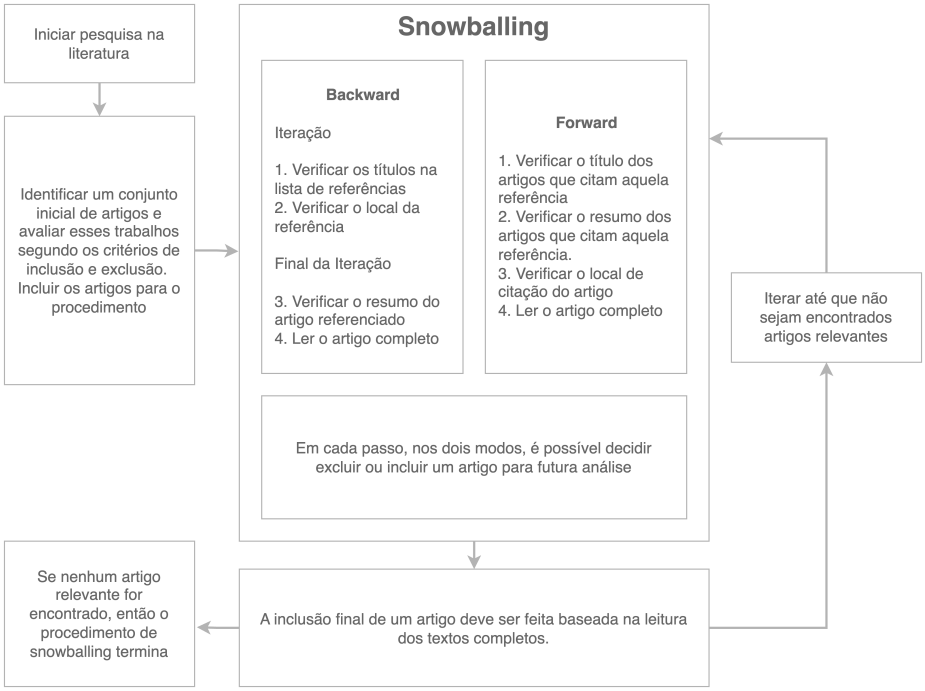

Figura 2.2 – Procedimento do *snowballing*. Traduzido de Wholin [55].

O método de *snowballing* segue o preconizado por Wohlin [55], que infere que, para uma primeira análise e aplicação do método, deve-se obter uma seleção inicial de trabalhos. Para a realização desta seleção inicial, foram elencadas duas bases de dados, *Association of Computer Machinery (ACM) - Digital Library*[1] e *IEEE Xplore*[2]. Os termos utilizados para a busca contemplam as seguintes *strings*:

(“Microservice” OR “Micro-service*” OR “Micro service*”)*

AND (“monolith”)*

AND (“disadvantages”)

[1]https://dl.acm.org/
[2]https://ieeexplore.ieee.org/

2.1.1 Critérios de Inclusão

Os critérios de inclusão determinados para a escolha inicial do *snowballing* e das iterações foram:

- Artigos que citem vantagens e desvantagens do uso de micro-serviços;
- Artigos com idioma em inglês ou português.

2.1.2 Critérios de Exclusão

Os critérios de exclusão estabelecidos para o *snowballing* estão listados a seguir:

- Trabalhos de *grey literature*, que serão adicionados pela revisão de *grey literature*;
- Estudos que não estejam nos idiomas inglês ou português;
- Dissertações e testes;
- Livros.

Os resultados obtidos com a realização da Revisão Sistemática utilizando o *Snowballing* estão dispostos na Seção 3.1.

2.2 Condução da Revisão da *Grey Literature*

O termo "*grey literature*" engloba várias definições que têm diferentes graus de conexão entre si. A primeira definição oficial foi apresentada na Terceira Conferência Internacional sobre Literatura Cinzenta, realizada em 1997, em Luxemburgo. Essa definição conceituava a categoria como estudos primários não publicados, contendo informações produzidas em diversos setores, como governo, acadêmico, negócios, indústria eletrônica e materiais não controlados por veículos comerciais impressos. Mais tarde, na mesma conferência, em 2004, a definição foi expandida para incluir relatórios acadêmicos e industriais, além de dissertações e teses.

Em 2010, durante a conferência realizada em Praga, houve uma adição ao conceito de literatura cinzenta. Dessa vez, o foco era nas questões de propriedade intelectual relacionadas aos trabalhos provenientes da internet. De acordo com essa nova definição, a literatura cinzenta abrangia documentos em todos os níveis governamentais, acadêmicos, empresariais e industriais, tanto em formato impresso quanto eletrônico, desde que

fossem protegidos por direitos de propriedade intelectual e possuíssem qualidade suficiente para serem coletados e preservados por bibliotecas ou instituições, sem estarem sob o controle de empresas cuja atividade principal não fosse a publicação. É importante ressaltar que essa definição não considerava postagens de *blogs* ou *tweets* como literatura cinzenta.

Somente em 2017, a análise da inclusão desse tipo de literatura começou a ser abordada por Adams *et al.* [2] em suas diretrizes para a inclusão de literatura cinzenta em estudos organizacionais. Essa análise dividiu o termo em diferentes níveis, levando em conta o grau de credibilidade científica, conforme pode ser observado na Figura 2.3. O primeiro nível (*tier 1*) incluía recursos com alto controle de fontes, como livros, revistas, relatórios governamentais e *white papers*. O segundo nível (*tier 2*) abrangia materiais classificados pelos autores como tendo credibilidade moderada, como relatórios anuais de empresas, materiais jornalísticos, apresentações, vídeos, sites de perguntas e respostas, e artigos derivados de enciclopédias virtuais (como a *Wikipedia*). A terceira camada (*tier* 3) correspondia à literatura de menor credibilidade acadêmica, como *blogs*, *tweets*, *e-mails*, entre outros.

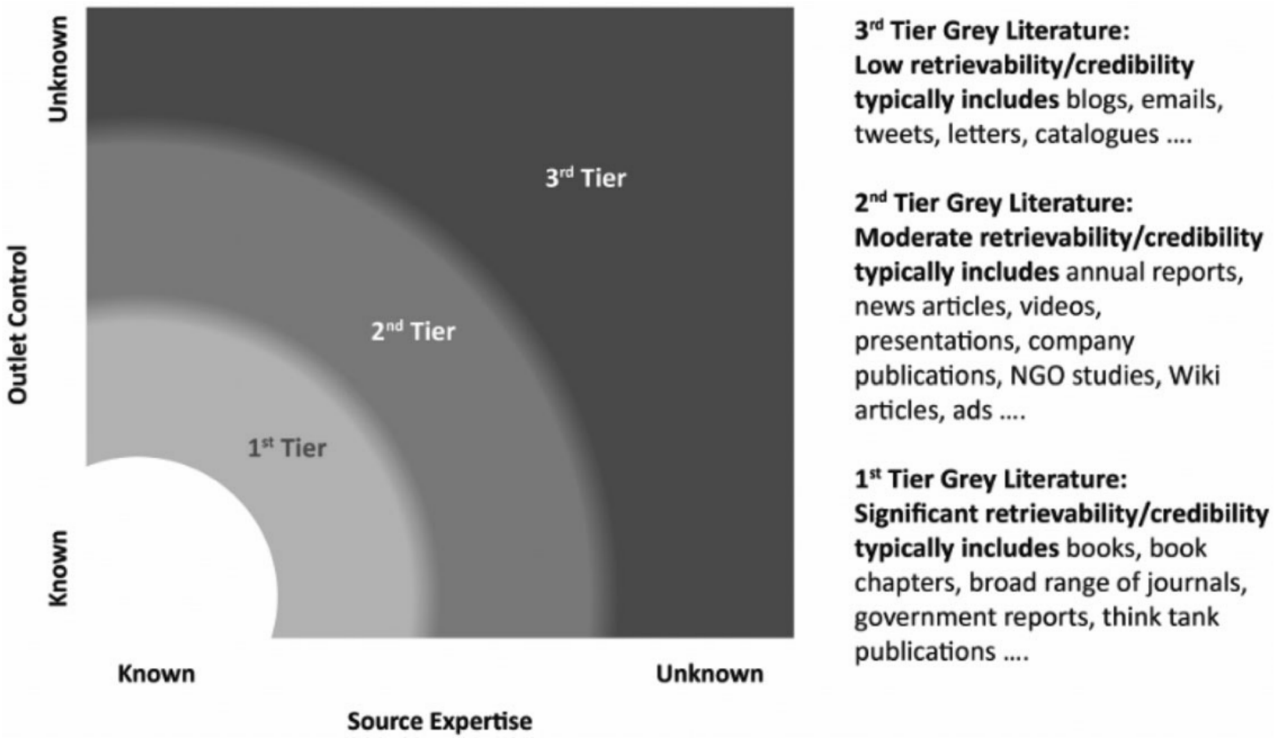

Figura 2.3 – Níveis de classificação para *grey literature* [2]

No contexto da Engenharia de *Software*, Garousi *et al.* [21] adaptaram o conceito de Adams *et al.* [2] para simplificar o termo, considerando literatura cinzenta todos os materiais que não passaram por revisão por pares. Essa simplificação foi necessária devido à quantidade significativa de conteúdo relevante, porém fora do escopo acadêmico, existente nessa área.

Dessa forma, a revisão da literatura cinzenta foi conduzida como complemento aos resultados obtidos por meio do método *snowballing* formando uma Revisão de Literatura Multivocal. Esse método foi escolhido devido ao fato de o tema desta pesquisa estar bastante presente nas práticas da indústria. A justificativa de uso do método vai ao encontro do descrito por Garousi *et al.* [21]. Além de Garousi *et al.* [21], outros pesquisadores também já fizeram esta ponte entre a necessidade de estudos na Engenharia de *Software* que fossem mais representativos da realidade de pessoas que estão no mercado de trabalho [27, 46].

A etapa de revisão de *grey literature* iniciou-se em paralelo ao método de *snowballing*. Para a seleção de trabalhos, uma combinação de *strings* foi desenvolvida, sendo esta utilizada no mecanismo de busca do *Google*[3]. A *string* utilizada foi a descrita a seguir:

(“comparison” “advantages” “disadvantages” “microservices” “monolith” site:“endereço de domínio pretendido”)

A escolha pela inclusão do termo *advantages* e *disadvantages* se deu devido ao fato de esta busca poder trazer resultados que mostrem as vantagens e desvantagens do uso desta arquitetura. Da mesma forma, a inclusão do termo *monolith* e *microservices* foi motivada pela busca por artigos que façam a comparação entre uma arquitetura e a outra. A escolha do uso do filtro por domínio se deu pela necessidade de refinamento da pesquisa, devido à quantidade de resultados obtidos. Os seis sites selecionados foram escolhidos por serem reconhecidos pela comunidade de desenvolvimento de *software* e estão listados a seguir:

- **Stack Overflow blog**: https://stackoverflow.blog/
- **Site Point**: https://www.sitepoint.com/
- **Geeks For Geeks**: https://www.geeksforgeeks.org/
- **freeCodeCamp**: https://www.freecodecamp.org/
- **DZone**: https://dzone.com/
- **Medium**: https://medium.com/

2.2.1 Critérios de Inclusão - Revisão da *Grey Literature*

Para a seleção dos trabalhos advindos de *grey literature*, os seguintes critérios de inclusão foram utilizados:

[3] https://www.google.com/

- Conteúdos em texto;
- Textos oriundos dos *blogs* selecionados, devido ao fato de haver uma grande quantidade de conteúdo compartilhado pela comunidade nesse tipo de plataforma;
- Conteúdos que abordem a temática da comparação das arquiteturas de micro-serviços e monolitos;
- Textos em idioma inglês ou português.

2.2.2 Critérios de Exclusão - Revisão da *Grey Literature*

Os itens que basearam os critérios de exclusão dos trabalhos de *grey literature* são os seguintes:

- Conteúdos de áudio ou vídeo sem transcrição de texto;
- Textos não oriundos de *blogs*;
- Textos duplicados;
- Textos que não apresentem comparação entre as arquiteturas propostas;
- Textos em outros idiomas que não em inglês ou português.

Os resultados obtidos com a realização da Revisão da *Grey Literature* estão dispostos na Seção 3.3.

2.3 *Survey*

Embora a etapa anterior de Revisão de Literatura Multivocal, utilizando as técnicas de *snowballing* e *grey literature* por si só já seja capaz de agregar valor a esta pesquisa [43], foi decidido aumentar a fonte de dados realizando uma pesquisa utilizando a metodologia de *survey*.

Nesta etapa, o objetivo foi identificar, usando uma abordagem qualitativa exploratória, a partir de entrevista aplicada a praticantes da indústria, o entendimento sobre a utilização das arquiteturas de *software* propostas. Pesquisa qualitativa denota um tipo de pesquisa que produz resultados que não resultam de procedimentos estatísticos ou outros que se quantificam mas que qualificam o fenômeno de investigação [16]. Estudos desta natureza referem-se à pesquisa sobre a vida das pessoas, experiências, comportamentos, emoções, e sentimentos, bem como funcionamento organizacional, movimentos

sociais, fenômenos culturais, e interações entre povos [16]. Desta forma, a pesquisa qualitativa envolve a interpretação dos assuntos tratados, estudando as coisas em suas configurações naturais na tentativa de interpretar o fenômeno conforme relatos trazidos pelas pessoas participantes da pesquisa [50].

Para a realização do estudo de campo desta pesquisa, foi utilizado o método de *survey*, que baseado no tema desta pesquisa, utilizou um questionário, que foi aplicado aos participantes durante sessões de entrevistas, a fim de traduzir o objetivo deste trabalho em questões para serem respondidas por uma amostra de profissionais de diferentes níveis de experiência na temática deste estudo. A formulação do questionário foi planejada de forma que as perguntas resultassem em respostas que agregassem valor a este trabalho e de que também fossem motivadoras, de forma que as pessoas estivessem mais engajadas a fornecer respostas completas e precisas se pudessem ver que os resultados do estudo provavelmente serão úteis para a comunidade. O embasamento para a criação das perguntas propostas aos entrevistados foi proveniente da análise dos resultados das metodologias aplicadas em passos anteriores nesta mesma pesquisa.

Também, para propor-se um instrumento de valor para uma amostra representativa, foi aplicado um piloto com os primeiros entrevistados, de forma que se pudesse identificar problemas com o questionário, bem como o procedimento de taxa de acompanhamento, sendo também este piloto útil para contribuir para a avaliação da confiabilidade [45].

2.3.1 Seleção dos Entrevistados

A seleção dos entrevistados foi feita de maneira intencional, com base na disponibilidade e interesse dos participantes. Optou-se por entrevistar nove profissionais que se voluntariaram para participar, todos integrantes do time em que o pesquisador está inserido, garantindo, assim, um nível significativo de engajamento e familiaridade com os temas abordados. Embora a amostragem tenha sido composta por colegas do mesmo ambiente de trabalho, o grupo selecionado reflete uma diversidade de níveis de experiência anteriores e posições dentro da área de atuação, o que agrega riqueza às perspectivas apresentadas. Ademais, a voluntariedade dos participantes assegurou maior liberdade e profundidade nas respostas, aspectos essenciais para a natureza qualitativa exploratória da pesquisa. Em estudos qualitativos, a amostra não se baseia em representatividade estatística, mas na capacidade de fornecer *insights* detalhados e profundos sobre o fenômeno investigado [16]. A escolha desse grupo, portanto, foi estratégica para captar experiências diretamente relacionadas ao objeto de estudo, oferecendo uma visão valiosa e contextualizada sobre as arquiteturas de software investigadas.

2.3.2 Entrevistas

Foram conduzidas entrevistas semiestruturadas seguindo os guias associados à Engenharia de *Software* [25] com 9 participantes que aceitaram o convite. As entrevistas foram conduzidas, transcritas e analisadas pelo autor deste estudo. Para as entrevistas, foram utilizadas questões que foram elaboradas com base nos resultados das etapas metodológicas anteriores. A todos os entrevistados foi aplicado o conjunto inicial de perguntas, porém, como é comum em entrevistas semiestruturadas, novas questões foram surgindo durante as entrevistas com base nas respostas de perguntas específicas. Assim, os entrevistados puderam falar livremente sobre suas percepções e opiniões.

As entrevistas foram realizadas em duas etapas e foram aplicadas para cada entrevistado da seguinte maneira. Na primeira etapa, foi apresentado o formulário de consentimento (o qual se encontra descrito no Apêndice B) e tão logo foi explicado como seria conduzida a entrevista. Em seguida, foi iniciada a entrevista que foi guiada por um questionário semiestruturado em 11 partes, com 22 questões sobre suas experiências profissionais e conhecimentos específicos sobre as arquiteturas de *software* abordadas neste estudo. Este questionário pode ser visto no Apêndice A.

2.3.3 Decodificação dos Dados na Análise dos Resultados

A análise dos resultados apresentada nesta dissertação foi conduzida utilizando a técnica de decodificação temática [11], que consiste em identificar padrões, categorias e temas emergentes a partir das falas dos entrevistados. Essa abordagem permitiu uma organização sistemática das respostas, facilitando a compreensão dos pontos de convergência e divergência nas percepções dos entrevistados.

No processo de decodificação, as entrevistas foram transcritas e lidas de forma cuidadosa. As informações relevantes foram, então, agrupadas em códigos representando ideias ou temas recorrentes. Por exemplo, nas discussões sobre a arquitetura de software, surgiram temas como facilidade de desenvolvimento, manutenção, escalabilidade, performance e complexidade. Cada código foi atribuído a trechos específicos das entrevistas, conforme as menções dos entrevistados.

Dessa forma, a técnica de decodificação possibilitou uma análise detalhada das opiniões e experiências dos entrevistados, identificando tanto os pontos comuns quanto as nuances específicas de cada perspectiva. O uso dessa técnica facilitou a organização das respostas em categorias temáticas e contribuiu para a estruturação lógica e clara dos resultados apresentados.

3. RESULTADOS

Este estudo foi concebido com o objetivo primordial de conduzir uma Revisão de Literatura Multivocal, adotando a metodologia de *snowballing* como estratégia para a coleta e análise de dados acadêmicos [55]. O *snowballing* é uma técnica de pesquisa que envolve a expansão da amostra por meio de citações e referências obtidas a partir de estudos iniciais. Essa abordagem é particularmente útil quando se deseja alcançar uma compreensão profunda de um campo de estudo específico, pois permite que os pesquisadores identifiquem e avaliem uma gama mais ampla de literatura relevante [55].

Além do uso do *snowballing*, este estudo também incorporou uma análise da *grey literature* [21]. A *grey literature* refere-se a fontes de informação que não são controladas por editoras comerciais, o que pode incluir relatórios, teses, conferências e outras formas de literatura acadêmica. A inclusão da *grey literature* nesta pesquisa permite uma visão mais abrangente do campo de estudo, pois abrange fontes que podem não estar presentes em canais de publicação tradicionais. Este ponto é particularmente relevante, considerando que o tema desta pesquisa é frequentemente discutido no contexto profissional.

Por fim, complementando as estratégias de pesquisa mencionadas, também foi conduzida uma pesquisa (*survey*) com profissionais que estão ativamente envolvidos no campo do desenvolvimento de *software*. Esta pesquisa destinou-se a coletar opiniões e percepções desses profissionais sobre o tema em questão, proporcionando percepções da prática profissional, que pode enriquecer a compreensão teórica obtida através da Revisão de Literatura Multivocal. Combinando esses diferentes métodos de pesquisa, este estudo visa fornecer uma visão aprofundada sobre a utilização das arquiteturas de *software* mencionadas.

3.1 Snowballing

Para o estudo sistemático utilizando a técnica de *snowballing*, a seleção inicial de artigos foi extraída das bases de dados *Association of Computer Machinery (ACM) - Digital Library* e *IEEE Xplore*. Para alcançar essa seleção inicial, foi empregada a *string* de busca previamente citada, com os filtros sendo aplicados de acordo com os critérios de inclusão previamente definidos (veja Seção 2.1). Como resultado desse processo, um total de 41 artigos foi identificado. Destes, 28 eram da *Association of Computer Machinery (ACM) - Digital Library* e os 13 restantes eram da base de dados *IEEE Xplore*. Todos os resumos (*abstracts*) desses artigos foram lidos e analisados cuidadosamente.

Após a análise dos resumos, 29 artigos da lista inicial foram excluídos. A razão principal para a remoção desses artigos foi que suas temáticas não estavam alinhadas com o foco principal desta pesquisa. Adicionalmente, alguns artigos foram excluídos com base nos critérios de inclusão e exclusão estabelecidos previamente. Isso resultou em um total de 12 artigos que foram selecionados para a primeira iteração do processo de revisão do *snowballing*. A Tabela 3.1 apresenta uma visão detalhada da distribuição desses artigos.

ID	Ref.	Título	Ano
S1	[34]	Methods and process of service migration from monolithic architecture to microservices	2022
S2	[22]	The Comparison of Microservice and Monolithic Architecture	2020
S3	[52]	Container-based Video Streaming Service	2022
S4	[41]	Reliability Evaluation of Microservices and Monolithic Architectures	2022
S5	[23]	Comparing Interservice Communications of Microservices for E-Commerce Industry	2022
S6	[51]	A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges	2023
S7	[4]	Performance Analysis of Microservice Design Patterns	2019
S8	[3]	An application of microservices architecture pattern to create a modular computer numerical control system	2017
S9	[8]	A Multi-way Access Portal Website Construction Scheme	2022
S10	[54]	A Feature Table Approach to Decomposing Monolithic Applications into Microservices	2021
S11	[49]	Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages	2017
S12	[17]	A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes	2020

Tabela 3.1 – Seleção inicial de artigos pelo *snowballing*

A execução da busca inicial do *snowballing* possibilitou a obtenção de 12 artigos que foram analisados para extrair as vantagens e desvantagens presentes nas arquiteturas de monolitos e de micro-serviços. Este trabalho de identificação e categorização dessas características foi essencial para aprofundar a compreensão destas duas arquiteturas e fornecer uma base sólida para o objetivo principal desta pesquisa.

Um dos principais pontos de interesse deste estudo é entender como a adoção da arquitetura de micro-serviços pode acarretar em um tempo maior de desenvolvimento de *software*. Para ilustrar essa questão de maneira mais clara e objetiva, as desvantagens associadas ao uso da arquitetura de micro-serviços, que foram mencionadas nos artigos selecionados, foram listadas e apresentadas na Tabela 3.2.

É importante ressaltar que esta tabela não pretende oferecer uma visão definitiva ou exaustiva, mas sim proporcionar um ponto de partida para uma discussão mais aprofundada sobre o tema. Cada desvantagem listada representa uma característica que deve ser cuidadosamente considerada por equipes de desenvolvimento e tomadores de decisão ao avaliar a adoção da arquitetura de micro-serviços em seus projetos.

Desvantagem no uso de micro-serviços	**Citações**
Complexidade operacional	6
Desafios de consistência de dados	4
Custo inicial de adoção	3
Overhead de rede	3
Aumento da complexidade de *DevOps*	3
Desafios de comunicação	2
Dificuldades na depuração	2
Monitoramento mais complexo	1
Dificuldades de versionamento	1
Padrões de segurança desafiadores	1

Tabela 3.2 – Tabela inicial de desvantagens associadas ao uso de micro-serviços

Após a seleção inicial dos artigos e análise dos seus conteúdos, deu-se início ao processo de iteração que envolveu o uso das técnicas de *backward* e *forward snowballing*. Esta abordagem permitiu a inclusão de 14 artigos adicionais ao conjunto inicial, resultando em um aumento significativo de fontes de informação para a pesquisa.

Os artigos adicionados passaram por uma análise detalhada, durante a qual foram extraídos os principais pontos relativos às desvantagens do uso de micro-serviços. Notavelmente, já na segunda iteração do processo de *snowballing*, não foram identificadas novas desvantagens além das já listadas previamente, o que sugere uma saturação de informações sobre os tópicos de interesse.

Dado que as iterações subsequentes não produziram novas informações, optou-se por finalizar as iterações propostas pelo *snowballing*. Essa decisão foi baseada na avaliação de que mais iterações não trariam novos dados significativos, e que o tempo e recursos investidos seriam melhor utilizados na análise dos dados já coletados.

Os 26 artigos selecionados para a pesquisa foram lidos na íntegra para garantir uma compreensão completa de seu conteúdo e do contexto em que foram escritos. A lista completa desses artigos pode ser encontrada na Tabela 3.3.

ID	Ref.	Título	Ano
S1	[34]	Methods and process of service migration from monolithic architecture to microservices	2022
S2	[22]	The Comparison of Microservice and Monolithic Architecture	2020
S3	[52]	Container-based Video Streaming Service	2022
S4	[41]	Reliability Evaluation of Microservices and Monolithic Architectures	2022
S5	[23]	Comparing Interservice Communications of Microservices for E-Commerce Industry	2022
S6	[51]	A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges	2023
S7	[4]	Performance Analysis of Microservice Design Patterns	2019
S8	[3]	An application of microservices architecture pattern to create a modular computer numerical control system	2017
S9	[8]	A Multi-way Access Portal Website Construction Scheme	2022
S10	[54]	A Feature Table Approach to Decomposing Monolithic Applications into Microservices	2021
S11	[49]	Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages	2017
S12	[17]	A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes	2020
S13	[44]	Migrating to a microservice architecture: benefits and challenges	2023
S14	[18]	Migrating Towards Microservice Architectures: An Industrial Survey	2018

S15	[40]	Migrating from monolithic architecture to microservices: A Rapid Review	2019
S16	[6]	Comparative Analysis of Microservices and Monolithic Architecture	2022
S17	[15]	Analyzing Microservices and Monolithic Systems: Key Factors in Architecture, Development, and Operations	2023
S18	[9]	Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation	2022
S19	[28]	Deployment and communication patterns in microservice architectures: A systematic literature review	2021
S20	[32]	Performance Characterization of Communication Protocols in Microservice Applications	2021
S21	[36]	Microservices: architecture, container, and challenges	2020
S22	[26]	Microservices: The Journey So Far and Challenges Ahead	2018
S23	[33]	Microarchitectural Analysis and Characterization of Performance Overheads in Service Meshes with Kubernetes	2023
S24	[12]	Microservices approach for the internet of things	2016
S25	[24]	Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap	2016
S26	[5]	Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture	2016

Tabela 3.3 – Seleção final de artigos pelo *snowballing*

Em relação às desvantagens associadas ao uso da arquiteturas de micro-serviços, uma lista completa com todas as desvantagens identificadas, bem como a frequência de suas ocorrências em todos os artigos analisados, está disponível na Tabela 3.4. Esta tabela fornece uma visão abrangente das percepções e experiências relatadas na literatura sobre o uso dessas arquiteturas.

Desvantagem no uso de micro-serviços	**Citações**
Complexidade operacional	20
Custo inicial de adoção	12
Desafios de consistência de dados	11
Overhead de rede	9

Aumento da complexidade de *DevOps*	9
Monitoramento mais complexo	5
Desafios de comunicação	4
Dificuldades na depuração	4
Dificuldades de versionamento	4
Padrões de segurança desafiadores	4

Tabela 3.4 – Tabela final de desvantagens associadas ao uso de micro-serviços

3.2 Análise dos resultados do *snowballing*

Nesta seção, será apresentada uma análise dos resultados obtidos através do processo de *snowballing*. O objetivo aqui é entender os desafios e as desvantagens associados à implementação e uso da arquitetura de micro-serviços, conforme relatado na literatura recente.

Esta análise centra-se em problemas identificados nos artigos científicos selecionados, com um foco particular entender como as práticas acadêmicas percebem estes problemas e como podem ajudar a mitigar estes.

3.2.1 Complexidade operacional

A complexidade operacional emerge como uma questão crítica na arquitetura de micro-serviços, uma vez que a adoção dessa abordagem resulta em uma proliferação de serviços autônomos que necessitam ser gerenciados e coordenados. Cada um desses serviços deve ser concebido, implementado, implantado, escalado e monitorado de forma independente, o que amplifica consideravelmente a carga de trabalho operacional.

Em contraste com as arquiteturas de monolitos tradicionais, onde as operações estão centralizadas e o gerenciamento é relativamente simples e direto, os micro-serviços introduzem uma magnitude de complexidade operacional que pode ser desafiadora. Isto é especialmente verdadeiro quando se considera a necessidade de garantir a comunicação eficiente entre serviços, gerenciar a consistência dos dados entre diferentes serviços e lidar com questões de falha e recuperação de serviço.

Essa complexidade operacional pode levar a um aumento no tempo de desenvolvimento de *software*, pois a equipe de desenvolvimento precisa dedicar esforços significativos para gerenciar e coordenar esses serviços. Não só a implementação inicial, mas

também a manutenção e a evolução do sistema podem exigir mais tempo devido à necessidade de lidar com a interação entre os serviços, a consistência dos dados e a resiliência do sistema.

Portanto, embora a arquitetura de micro-serviços possa trazer benefícios significativos em termos de escalabilidade e flexibilidade, é importante reconhecer a complexidade operacional como uma consequência inerente que pode afetar o tempo de desenvolvimento e requerer uma gestão cuidadosa para garantir a eficácia da adoção dessa arquitetura.

3.2.2 Custo inicial de adoção

O custo inicial de adoção é um problema notório associado à arquitetura de micro-serviços. Essa arquitetura, embora flexível e escalável, demanda um investimento inicial significativo que vai além da mera capacidade financeira. Esse custo se manifesta em diversos aspectos, incluindo a necessidade de adquirir competências técnicas específicas, implementar novas ferramentas e processos, além de lidar com questões de segurança e gerenciamento de dados distribuídos.

A complexidade da arquitetura de micro-serviços é um dos principais fatores que contribui para o custo inicial de adoção. A decomposição de um aplicativo em diferentes serviços requer uma compreensão sólida dos domínios do problema e uma capacidade de projetar componentes independentes, mas interconectados. Este processo pode ser demorado e exige um alto nível de competências, o que pode atrasar significativamente o tempo de desenvolvimento do *software*.

A necessidade de adotar novas ferramentas e tecnologias também é uma contribuição significativa para o custo inicial. A arquitetura de micro-serviços requer o uso de ferramentas avançadas de integração, entrega e implantação contínuas, monitoramento de serviços, gerenciamento de contêineres, dentre outros. Essas ferramentas demandam investimento, tanto financeiro quanto de tempo, para aquisição e aprendizado.

Além disso, a arquitetura de micro-serviços apresenta desafios únicos em termos de segurança e gerenciamento de dados. O gerenciamento de dados distribuídos pode ser complexo e requer soluções robustas e bem pensadas. Da mesma forma, a necessidade de garantir a segurança em um ambiente distribuído pode ser um desafio considerável, exigindo esforços adicionais e, consequentemente, aumentando o tempo de desenvolvimento.

Portanto, embora a arquitetura de micro-serviços possa oferecer vantagens substanciais em termos de escalabilidade e flexibilidade, o custo inicial de adoção é uma consideração importante que pode levar a um tempo de desenvolvimento mais longo. Isso

precisa ser cuidadosamente considerado ao avaliar a viabilidade e apropriabilidade desta arquitetura para um determinado projeto ou organização.

3.2.3 Desafios de consistência de dados

A arquitetura de micro-serviços, embora esteja associada à diversas vantagens, como isolamento de falhas e escalabilidade, também apresenta desafios significativos. Entre eles, os desafios de consistência de dados surgem como um problema recorrente. Este problema está intrinsecamente ligado ao princípio fundamental dos micro-serviços, que é a descentralização do controle de dados.

Em um ambiente de micro-serviços, cada serviço gerencia seu próprio banco de dados, levando à fragmentação dos dados. Isso representa um desafio para manter a consistência de dados entre os vários serviços. Em particular, operações que envolvem várias transações em diferentes serviços podem levar à inconsistência de dados se uma das transações falhar, um problema conhecido como atomicidade.

Adicionalmente, a gestão de consistência de dados em uma arquitetura de micro-serviços pode acarretar em um aumento no tempo de desenvolvimento de *software*. A necessidade de garantir a consistência de dados entre os diversos serviços exige um esforço de coordenação adicional, podendo exigir a implementação de mecanismos complexos de sincronização de dados, como a coordenação de transações distribuídas.

Portanto, embora a arquitetura de micro-serviços possa trazer benefícios significativos em termos de flexibilidade e escalabilidade, a complexidade associada à manutenção da consistência dos dados pode resultar em tempos de desenvolvimento prolongados, sendo este um fator que deve ser levado em consideração ao se optar por este tipo de arquitetura.

3.2.4 *Overhead* de rede

Uma vez que os micro-serviços operam de maneira distribuída, exigindo comunicação contínua pela rede para funcionar corretamente, o uso de micro-serviços resulta em uma maior quantidade de chamadas de rede, cada uma com seu próprio custo em termos de latência e largura de banda.

Este aumento na comunicação de rede não apenas adiciona complexidade ao projeto e à manutenção do sistema, mas também pode resultar em um tempo de resposta mais lento, especialmente quando os serviços estão geograficamente dispersos ou quando a rede está congestionada. Isso pode levar a um desempenho degradado, que precisa ser cuidadosamente gerenciado para garantir a qualidade do serviço.

Além disso, o *overhead* de rede também pode contribuir para o prolongamento do ciclo de desenvolvimento do *software*. O projeto, teste e depuração de sistemas distribuídos são inerentemente mais complexos e demorados do que seus equivalentes monolitos, devido à necessidade de sincronizar e coordenar vários serviços. Isso pode levar a um aumento no tempo de desenvolvimento, à medida que os engenheiros precisam navegar por essas complexidades adicionais.

Em suma, o *overhead* de rede é um desafio significativo na arquitetura de micro-serviços, com implicações potenciais tanto para o desempenho do sistema quanto para o tempo de desenvolvimento do *software*. Portanto, deve ser cuidadosamente considerado ao optar por esta arquitetura.

3.2.5 Aumento da complexidade de *DevOps*

O aumento da complexidade em *DevOps* é um problema recorrente e notável associado à implementação da arquitetura de micro-serviços. A arquitetura de micro-serviços, embora ofereça benefícios significativos em termos de escalabilidade e desacoplamento de serviços, também implica uma sobrecarga adicional nos procedimentos de *DevOps*, exigindo um grau mais elevado de coordenação e gerenciamento.

Na arquitetura de micro-serviços, cada serviço é desenvolvido, implantado e escalado de forma independente, o que pode resultar em um grande número de *pipelines* de CI/CD (Integração Contínua/Entrega Contínua) para gerenciar. Além disso, a necessidade de coordenação entre diferentes serviços e a garantia de compatibilidade entre as interfaces de serviço aumentam a complexidade do gerenciamento de configurações. Isso pode resultar em um aumento significativo do tempo de desenvolvimento de *software*, particularmente em projetos maiores com muitos serviços interdependentes.

Outra área onde a complexidade aumenta é o monitoramento e o gerenciamento de falhas. Cada micro-serviço precisa ser monitorado individualmente, e a detecção e correção de falhas pode se tornar complexa devido à natureza distribuída dos serviços. Isso pode levar a uma maior sobrecarga de trabalho para as equipes de *DevOps*, potencialmente prolongando o ciclo de desenvolvimento do *software*.

Embora o aumento da complexidade em *DevOps* possa ser mitigado até certo ponto com o uso de ferramentas e práticas apropriadas, é inevitável que essa complexidade adicional imponha um ônus sobre as equipes de desenvolvimento e operações, potencialmente aumentando o tempo e os recursos necessários para o desenvolvimento de *software* na arquitetura de micro-serviços. Portanto, é crucial que a adoção dessa arquitetura seja cuidadosamente considerada, levando em conta tanto os benefícios potenciais quanto os desafios que ela apresenta.

3.2.6 Monitoramento mais complexo

O monitoramento em uma arquitetura de micro-serviços representa um desafio não trivial, especialmente quando comparado com a arquitetura de monolitos. Nesse ambiente descentralizado e altamente distribuído, cada serviço opera de forma independente, possuindo seu próprio conjunto de indicadores de desempenho e métricas, o que torna a atividade de monitoramento consideravelmente mais complexa.

Nesse contexto, a dificuldade em rastrear transações que atravessam vários micro-serviços, identificar a causa raiz dos problemas de desempenho ou disponibilidade, e gerenciar a quantidade volumosa de dados de *log* gerados pelos diversos serviços torna-se uma tarefa complexa. Essa complexidade adicional não só exige ferramentas de monitoramento mais sofisticadas e robustas, mas também demanda mais tempo e recursos da equipe de desenvolvimento para configurar, gerenciar e interpretar o monitoramento de forma eficaz.

Portanto, apesar das vantagens oferecidas pela arquitetura de micro-serviços, como a possibilidade de escalabilidade e a independência entre serviços, a complexidade inerente ao monitoramento pode prolongar o tempo de desenvolvimento e implantação de *software*, uma vez que a implementação de um sistema de monitoramento eficaz e confiável se torna uma etapa crítica e indispensável do processo. Além disso, a necessidade de equipe especializada para gerir tal complexidade pode implicar em custos adicionais e em um maior tempo de resposta a possíveis problemas identificados através do monitoramento.

3.2.7 Desafios de comunicação

A arquitetura de micro-serviços, embora vantajosa em muitos aspectos, acarreta uma série de desafios que podem influenciar negativamente o tempo de desenvolvimento do *software*. Um desses obstáculos é o desafio de comunicação entre os serviços.

Na arquitetura de micro-serviços, diferentes serviços funcionam de maneira independente uns dos outros, cada um com seu próprio banco de dados e lógica de negócios. Esta independência funcional, apesar de ser atraente em termos de escalabilidade e isolamento de falhas, introduz uma complexidade considerável no que diz respeito à comunicação entre os serviços. Os micro-serviços devem interagir entre si para formar um sistema coeso, e a coordenação dessas interações é um desafio significativo.

A comunicação entre micro-serviços geralmente é realizada através de APIs e protocolos de rede, que precisam ser cuidadosamente projetados para garantir a integridade dos dados e a eficiência da comunicação. Além disso, é necessário implementar

mecanismos de tolerância a falhas e estratégias de recuperação, pois uma falha em um único serviço pode ter um efeito cascata em todo o sistema.

Essa complexidade adicional na coordenação e gerenciamento das interações entre os serviços pode resultar em um aumento no tempo de desenvolvimento de *software*. Em comparação com a arquitetura de monolitos, onde as diferentes partes do *software* interagem através de chamadas de função internas, a arquitetura de micro-serviços requer uma abordagem muito mais sofisticada e cuidadosa para a comunicação entre serviços. Portanto, a introdução de micro-serviços pode exigir mais tempo e esforço de desenvolvimento para garantir uma comunicação eficaz e confiável entre os serviços.

3.2.8 Dificuldades na depuração

A arquitetura de micro-serviços, embora seja reconhecida por sua escalabilidade e modularidade, implica um conjunto de desafios inerentes. Um desses desafios é a dificuldade na depuração. Quando surge um problema, pode ser complexo e demorado rastrear a sua origem, pois cada serviço pode ser implementado usando diferentes linguagens de programação, ter seus próprios *logs* e possuir seu próprio ciclo de vida.

A depuração em uma arquitetura de micro-serviços requer a correlação de *logs* e dados de rastreamento de vários serviços, uma tarefa que pode rapidamente se tornar complexa e pesada. Esta complexidade adicional na depuração de erros pode resultar em um aumento no tempo de desenvolvimento do *software*, pois os desenvolvedores precisam dedicar uma quantidade significativa de tempo para analisar e entender a origem do problema.

Além disso, a falta de um estado global compartilhado e a falta de sincronia entre os micro-serviços também podem tornar a depuração mais desafiadora. Erros que ocorrem devido a problemas de temporização ou falhas de comunicação entre serviços podem ser particularmente difíceis de reproduzir e, portanto, de depurar.

Portanto, enquanto a arquitetura de micro-serviços oferece muitas vantagens, as dificuldades na depuração representam um desafio significativo que pode resultar em um tempo de desenvolvimento de *software* mais extenso. A solução deste problema requer ferramentas e práticas adequadas de depuração distribuída, assim como uma compreensão profunda da interação entre os serviços.

3.2.9 Dificuldades de versionamento

O versionamento em uma arquitetura de micro-serviços representa um desafio complexo e multifacetado. A natureza distribuída desta arquitetura implica em um número

significativamente maior de componentes interdependentes, cada um com seu próprio ciclo de vida e versões. Gerenciar e coordenar as versões desses serviços pode se tornar uma tarefa arduamente complexa, principalmente quando comparado à arquitetura de monolitos, onde a aplicação é tratada como uma única entidade.

No âmbito dos micro-serviços, ocorrem frequentemente situações onde um serviço precisa ser atualizado, enquanto outros serviços que dependem dele ainda estão operando com versões anteriores. A coordenação dessas atualizações, garantindo que todas as dependências estejam sincronizadas, é um desafio notável. Sem um gerenciamento cuidadoso, isso pode levar a incompatibilidades de versões, resultando em falhas de integração, *bugs* e, em alguns casos, paralisação de serviços.

Além disso, o versionamento em micro-serviços requer um esforço de desenvolvimento consideravelmente maior. As equipes de desenvolvimento precisam dedicar uma parcela substancial de seu tempo para rastrear e coordenar as versões dos serviços, o que inevitavelmente prolonga o ciclo de desenvolvimento. Isso poderia ser usado para aprimorar funcionalidades, corrigir *bugs* ou melhorar a performance da aplicação em uma arquitetura de monolitos.

Portanto, embora a arquitetura de micro-serviços possa oferecer vantagens significativas em termos de escalabilidade e isolamento de falhas, as dificuldades de versionamento representam um obstáculo substancial que exige um investimento de tempo considerável por parte das equipes de desenvolvimento. Isso, por sua vez, pode levar a um aumento no tempo total de desenvolvimento de *software*.

3.2.10 Padrões de segurança desafiadores

A arquitetura de micro-serviços, por sua natureza descentralizada e distribuída, apresenta uma complexidade adicional aos protocolos de segurança tradicionais, exigindo a implementação de um modelo de segurança mais sofisticado e elaborado.

Nessa arquitetura, cada micro-serviço é responsável por sua própria base de dados e lógica de negócio, podendo ser desenvolvido e implantado de maneira autônoma. Embora essa característica traga benefícios como robustez e escalabilidade, ela também cria várias superfícies de ataque potenciais para agentes mal-intencionados, pois cada serviço representa um ponto de acesso potencial ao sistema. A necessidade de proteger cada um desses pontos de acesso aumenta a complexidade do sistema de segurança, exigindo um controle rigoroso de acesso, autenticação robusta e medidas de segurança de dados eficazes.

Além disso, a comunicação entre os micro-serviços - geralmente realizada através de APIs - precisa ser segura para prevenir ataques de interceptação e manipulação de

dados. Isso requer a implementação de protocolos de segurança de comunicação adicionais, como criptografia de transporte e assinaturas digitais.

A necessidade de implementar e gerenciar essas medidas de segurança complexas pode resultar em um aumento no tempo de desenvolvimento do *software*. Os desenvolvedores precisam gastar um tempo significativo projetando, implementando, testando e mantendo os sistemas de segurança para cada micro-serviço, além de seu trabalho habitual de desenvolvimento de funcionalidades. Isso pode levar a um ciclo de desenvolvimento de *software* mais longo, aumentando o tempo de lançamento para o mercado e potencialmente afetando a competitividade da empresa.

Portanto, embora a arquitetura de micro-serviços traga vários benefícios em termos de escalabilidade e robustez, os padrões de segurança desafiadores representam um obstáculo significativo que deve ser cuidadosamente gerenciado para garantir a integridade e segurança do sistema.

3.2.11 Conclusão da análise do *snowballing*

A análise de *snowballing* revelou uma série de desafios inerentes à implementação e uso da arquitetura de micro-serviços, conforme identificado na literatura científica recente. A complexidade operacional, o custo inicial de adoção, os desafios de consistência de dados, o *overhead* de rede, o aumento da complexidade de *DevOps*, o monitoramento mais complexo, os desafios de comunicação, as dificuldades na depuração, as dificuldades de versionamento e os padrões de segurança desafiadores são obstáculos significativos que podem afetar o tempo de desenvolvimento e a eficácia da adoção desta arquitetura.

Cada um desses desafios apresenta implicações importantes para as equipes de desenvolvimento e operações, exigindo estratégias de gerenciamento cuidadosas, ferramentas apropriadas e um alto nível de expertise para garantir a eficácia da adoção da arquitetura de micro-serviços. Embora a arquitetura de micro-serviços ofereça benefícios significativos em termos de escalabilidade, flexibilidade e isolamento de falhas, é essencial que as organizações estejam cientes dos potenciais desafios associados e estejam preparadas para gerenciá-los efetivamente.

Em suma, a arquitetura de micro-serviços, apesar de suas vantagens, apresenta uma série de desafios que podem resultar em um aumento no tempo de desenvolvimento de *software*. Portanto, a decisão de adotar essa arquitetura deve ser cuidadosamente considerada, levando em conta tanto os benefícios potenciais quanto os desafios inerentes. Futuras pesquisas nessa área são necessárias para desenvolver estratégias, práticas e ferramentas mais eficazes para gerenciar esses desafios e maximizar os benefícios da arquitetura de micro-serviços.

3.3 *Grey Literature*

Os resultados obtidos pela busca de *blogs* na internet totalizaram um número de 90 artigos. Destes 90 artigos, 68 foram obtidos dos *blogs* **Medium**, 1 do **Site Point**, 6 do **Geeks For Geeks**, 1 do **freeCodeCamp**, 13 do **DZone**, e 1 do **Stack Overflow blog**.

Para a seleção dos artigos de acordo com os critérios de inclusão e exclusão (veja Seção 2.2), foi realizada a leitura dos textos e a classificação das desvantagens mencionadas por eles. Como alguns dos *blogs* continha uma grande quantidade de resultados, esta classificação das desvantagens do uso da arquitetura de micro-serviços foi útil para gerar uma base de conhecimento sobre estas, o que mostrou que a indústria tem uma percepção abrangente sobre os problemas relacionados ao uso desta arquitetura. O número final de textos selecionados foi de 27 artigos, de acordo com os critérios de exclusão propostos. Dos 27 artigos, 20 foram obtidos do *blog* **Medium**, 1 do **Site Point**, 2 do **Geeks For Geeks**, 1 do **freeCodeCamp**, 3 do **DZone**, e nenhum do **Stack Overflow blog**, como visto na Tabela 3.5.

Site	**Resultados da busca**	**Textos selecionados**
Medium	68	20
DZone	13	3
Geeks For Geeks	6	2
Site Point	1	1
freeCodeCamp	1	1
Stack Overflow blog	1	0
Total	**90**	**27**

Tabela 3.5 – Resultados da execução do *Grey Literature*

3.4 Análise dos resultados da *Grey Literature*

Da mesma forma como foi encontrado na academia com a utilização do *snowballing*, o estudo dos textos selecionados pela realização da análise da *grey literature* revela uma série de desafios associados à implementação e utilização da arquitetura de micro-serviços. Estes desafios, embora variem em grau e complexidade, destacam as dificuldades inerentes a essa arquitetura e as questões que precisam ser consideradas ao adotá-la.

Primeiramente, a complexidade na implementação e manutenção é citada em 25 dos textos analisados. Este desafio é devido ao fato de que a arquitetura de micro-serviços envolve a criação e gestão de múltiplos serviços independentes, cada um com seu próprio ciclo de vida. Isso pode levar a uma complexidade significativa tanto na fase de implementação quanto na manutenção desses serviços.

Em segundo lugar, os desafios no gerenciamento de estado distribuído são mencionados em 18 dos textos. Este problema surge porque, em uma arquitetura de micro-serviços, o estado da aplicação é geralmente distribuído entre vários serviços. Isso pode tornar o gerenciamento do estado da aplicação complexo e propenso a erros.

Além disso, as dificuldades na automação de *deploy* e configuração são mencionadas em 13 dos textos. A automação é essencial para gerenciar eficientemente o grande número de serviços em uma arquitetura de micro-serviços. No entanto, a automação pode ser difícil devido à complexidade e à diversidade dos serviços.

Os custos adicionais de infraestrutura, citados em 12 dos textos, também são uma preocupação. Como cada serviço em uma arquitetura de micro-serviços geralmente é executado em seu próprio ambiente, isso pode levar a um aumento dos custos de infraestrutura.

Outros problemas identificados nos textos incluem o aumento da latência de rede devido à comunicação entre serviços (11 textos), a necessidade de lidar com uma diversidade de tecnologias (11 textos), as dificuldades na realização de testes abrangentes e eficazes (11 textos), os desafios de segurança decorrentes da complexidade da arquitetura de micro-serviços (8 textos), a necessidade de um sistema de ponto de entrada para facilitar a comunicação e o controle entre os serviços (3 textos), a falta de treinamento adequado para as equipes de desenvolvimento (2 textos) e os desafios na gestão e controle do versionamento de código (2 textos).

A distribuição desses problemas pode ser visualizada na Tabela 3.6. Isso oferece uma visão abrangente dos desafios associados à arquitetura de micro-serviços e pode ajudar a guiar futuras pesquisas e práticas nesse campo.

Problemas Elencados	**Citações**
Alta complexidade na implementação e manutenção	25
Desafios no gerenciamento de estado distribuído	18
Dificuldades na automação de *deploy*	13
Custos adicionais de infraestrutura	12
Aumento da latência de rede devido à comunicação	11
Necessidade de lidar com a diversidade de tecnologias	11
Dificuldades na realização de testes abrangentes	11
Desafios de segurança decorrentes da complexidade	8
Necessidade de um sistema de ponto de entrada	3

Falta de treinamento adequado para equipes	2
Gestão e controle do versionamento de código	2

Tabela 3.6 – Problemas elencados na revisão da *Grey Literature* com a utilização da arquitetura de micro-serviços

3.5 *Survey*

Nesta seção, são apresentados os resultados obtidos a partir da realização de uma *survey*, aplicada em forma de entrevistas semiestruturadas. As perguntas foram cuidadosamente elaboradas com base nos resultados obtidos nas etapas metodológicas anteriores.

Nas entrevistas, os participantes foram convidados a compartilhar suas experiências, percepções e opiniões em relação aos temas identificados, oferecendo assim uma riqueza de dados qualitativos que foram posteriormente analisados.

Os resultados apresentados a seguir são uma síntese das respostas e pontos de vista expressos pelos participantes durante as entrevistas. Eles proporcionam um olhar detalhado e abrangente sobre o tema desta pesquisa. É importante ressaltar no entanto que, enquanto os resultados oferecem uma visão valiosa, eles são baseados nas perspectivas dos participantes e, portanto, estão sujeitos à interpretação.

3.6 Análise dos resultados da *Survey*

3.6.1 Dados demográficos

A análise dos dados demográficos revelou uma variedade de experiências e perspectivas dos participantes da pesquisa. As respostas dos entrevistados, que variam de estagiários a líderes técnicos e especialistas, mostram um amplo espectro de experiência no campo do desenvolvimento de *software*. A Tabela 3.7 resume os papéis e a experiência dos entrevistados.

Entrevistado	**Papel dos Entrevistados**	**Anos de Experiência**
E1	Desenvolvedor *Backend*	10
E2	Especialista em Desenvolvimento de *Software*	5
E3	Desenvolvedor de *Software*	8

E4	Desenvolvedor de *Software*	3
E5	Especialista em Desenvolvimento de *Software*	5
E6	Estagiário de Desenvolvimento de *Software*	1
E7	Engenheiro de *Software*	10
E8	Desenvolvedor de *Software* Sênior	16
E9	Desenvolvedor e Líder Técnico	13

Tabela 3.7 – Dados Demográficos: Experiências dos entrevistados

No que diz respeito ao setor em que os entrevistados atuam, houve uma representação equilibrada entre os que trabalham no desenvolvimento de *software* ERP e os que atuam no setor de viagens corporativas. Isso sugere que as percepções e experiências compartilhadas pelos entrevistados podem ser generalizáveis para esses setores.

Em termos de experiência, a maioria dos entrevistados tem uma considerável quantidade de anos trabalhando na área de desenvolvimento de *software*, com uma média de 7,9 anos. O entrevistado com a menor quantidade de anos de experiência tem apenas 1 ano como estagiário de desenvolvimento de *software*, enquanto o com mais experiência tem 16 anos como desenvolvedor de *software* sênior. Essa variedade na experiência dos entrevistados pode trazer percepções valiosas, pois proporciona uma compreensão mais ampla das diferenças entre as arquiteturas de micro-serviços e monolitos.

Dada a diversidade de experiência e setores representados pelos entrevistados, as respostas coletadas oferecem uma visão abrangente e variada das diferenças entre as arquiteturas de micro-serviços e monolitos, bem como do impacto que a utilização de micro-serviços pode ter no tempo de desenvolvimento.

3.6.2 Experiência com arquiteturas de *software*

A análise das respostas dos entrevistados revela um amplo espectro de experiências e percepções em relação às arquiteturas de micro-serviços e de monolitos. Todos os entrevistados possuem experiência com ambas as arquiteturas, indicando uma relevância contínua de ambos os modelos na indústria de *software*.

Há um consenso geral entre os entrevistados de que micro-serviços são mais escaláveis, uma observação que é apoiada por vários estudos (E1, E2, E8). Além disso, a modularidade dos micro-serviços permite um gerenciamento mais simples, especialmente em projetos de grande escala (E2, E3, E4, E5). Como E5 destacou, *“a manutenção de micro-serviços é muito mais simples, principalmente em projetos desconhecidos, visto que os serviços estão isolados”*.

Por outro lado, os entrevistados também apontaram que os monolitos podem ser mais fáceis de desenvolver e testar (E2, E3, E4), possivelmente devido à menor complexidade inerente em ter um repositório de código e um banco de dados unificados. Além disso, a comunicação interna é mais simplificada em monolitos, como E4 mencionou, *"em monolitos a comunicação é mais simplificada devido a esta ser feita internamente, e não por chamadas por rede"*.

No entanto, os entrevistados também reconheceram as limitações da arquitetura de monolitos. E8 observou que a velocidade de entrega pode ser mais lenta em monolitos e que a escalabilidade é geralmente mais limitada. Além disso, E7 destacou *"a arquitetura de monolitos é centralizada, enquanto a arquitetura de micro-serviços tem serviços mais distribuídos"*, o que pode limitar a distribuição e independência dos serviços nos monolitos.

A análise dessas respostas sugere que a escolha entre micro-serviços e monolitos depende fortemente do contexto do projeto. Para projetos menores, os monolitos podem ser mais adequados devido à sua simplicidade de desenvolvimento e teste. Por outro lado, para projetos de grande escala com muitos desenvolvedores e funcionalidades diferentes, os micro-serviços podem oferecer vantagens significativas em termos de escalabilidade, manutenção e velocidade de entrega.

3.6.3 Perguntas específicas: Desenvolvimento e manutenção

Na análise das respostas dos entrevistados, emergem perspectivas variadas sobre as facilidades e desafios do desenvolvimento de novas funcionalidades em arquiteturas de micro-serviços e monolitos. De acordo com a maioria dos respondentes (E3, E4, E5, E6 e E8), a arquitetura de micro-serviços facilita o desenvolvimento de novas funcionalidades, principalmente para aplicações grandes. Isto é confirmado pela observação do E7, *"depende de cada projeto: Se o projeto for pequeno, então monolitos acaba sendo mais rápido para desenvolver"*.

Por outro lado, E1 e E2 mostram preferência pelo monolito para o desenvolvimento de novas funcionalidades. E9 amplia a discussão ao ressaltar a importância do contexto, tamanho da equipe, expectativas do cliente e necessidades do produto. Este entrevistado também destaca a relevância da maturidade do time e as complexidades associadas à divisão de bancos de dados em micro-serviços.

> *"Na minha opinião, depende muito do contexto. Normalmente é mais fácil começar com um monolito, especialmente se você tem uma equipe pequena, pois isso faz sentido. No entanto, se você já começa com várias equipes, geralmente é necessário iniciar com microsserviços para evitar muitas dores de cabeça, e você precisa ter uma ideia clara de como dividir os serviços e as responsabilidades entre eles. Também é importante considerar o tamanho que o software vai alcançar;*

> *não faz sentido criar uma complexidade enorme com microsserviços se um serviço monolítico poderia atender às expectativas do cliente e às necessidades do produto.”* (Entrevistado 9)

Quanto à manutenção do sistema a longo prazo, a maioria dos entrevistados (E1, E2, E3, E4, E5, E6 e E9) concorda que a arquitetura de micro-serviços demanda menos esforço. E7, no entanto, acredita que os monolitos são mais fáceis de manter a longo prazo.

Em suma, as opiniões dos entrevistados sobre desenvolvimento e manutenção refletem a complexidade da escolha entre arquiteturas de micro-serviços e monolitos. Vários fatores influenciam essa decisão, incluindo o tamanho do projeto, a natureza do sistema, a maturidade da equipe e as expectativas do cliente. Isso sugere a necessidade de uma abordagem flexível e contextualizada para a escolha da arquitetura de *software*.

3.6.4 Perguntas específicas: Escalabilidade e performance

Na seção de perguntas específicas referentes à escalabilidade e performance, os entrevistados forneceram perspectivas valiosas sobre suas experiências no trabalho com arquiteturas de monolitos e micro-serviços.

Quando perguntados sobre qual arquitetura tem se provado mais escalável para lidar com um aumento significativo na demanda de usuários ou dados, a maioria dos entrevistados (E1 a E7) convergiu na opinião de que a arquitetura de micro-serviços apresenta uma maior escalabilidade. E8 destacou que a manutenção de um fluxo de micro-serviços pode ser desafiadora se o sistema não estiver bem estruturado ou documentado, mas ainda assim, advoga por micro-serviços para serviços muito grandes. E9 reforçou a ideia de que a arquitetura baseada em micro-serviços é mais escalável, principalmente se a comunicação entre os micro-serviços for assíncrona, como através de mensagens em fila.

Em termos de performance, alguns entrevistados (E1, E2, E3, E6 e E8) indicaram que observaram melhor performance com micro-serviços. E5, no entanto, ressaltou *“micro-serviços tem uma complexidade maior, então é mais difícil conseguir gerenciar e mais difícil de alocar recursos, o que acarreta em a performance ser prejudicada em comparação a monolitos.”*. E7 trouxe uma perspectiva interessante: *“Já vi projetos onde monolitos serviu muito bem e acabou não tendo muito impacto em questão de performance, mas quando você migrou para o microsserviço acabou tendo um alto custo para ter a mesma performance.”*. E9 discutiu a diferença na facilidade de testar a aplicação como um todo, afirmando que os monolitos geralmente são mais simples para subir a aplicação inteira em uma única máquina, enquanto que com os micro-serviços pode ser necessário conectar-se a múltiplos serviços.

Em resumo, a análise dessas respostas sugere que, embora a arquitetura de micro-serviços possa oferecer maior escalabilidade e performance em determinados contextos, ela também apresenta uma complexidade adicional que pode dificultar o gerenciamento e a alocação de recursos. Além disso, as necessidades específicas do projeto, a estruturação e documentação do sistema, e a forma como a comunicação entre os micro-serviços é implementada, podem influenciar significativamente a escalabilidade e performance da arquitetura. Portanto, é importante considerar esses fatores ao escolher entre uma arquitetura de micro-serviços e monolito para um projeto específico.

3.6.5 Perguntas específicas: Complexidade

Na seção dedicada à complexidade, um tema central que surgiu das respostas dos entrevistados foi a questão da compreensão e manutenção do código-fonte e da lógica de negócios em diferentes arquiteturas.

Os entrevistados apresentaram opiniões variadas sobre se as arquiteturas de monolitos ou de micro-serviços são mais fáceis de entender e manter. E1, E3, E7 e E8 preferiram a arquitetura de monolitos, citando que a centralização do código e das regras de negócios facilita a compreensão e a manutenção. Como E8 observou, *“as regras de negócio e os fluxos estão todos contidos dentro de um mesmo pacote, o que facilita a visualização do fim a fim de um fluxo de negócio em termos de código”*.

Por outro lado, E2, E4 e E6 favoreceram a arquitetura de micro-serviços. Eles argumentaram que a separação de responsabilidades e componentes pode facilitar a manutenção e a compreensão. E2 mencionou que a arquitetura de micro-serviços facilita a compreensão *“pela separação de responsabilidades”*.

No entanto, é importante destacar o ponto de vista do E5 e E9, que argumentaram que a facilidade de compreensão e manutenção depende mais da implementação do projeto e de quem está analisando do que da arquitetura em si, conforme mencionado por E9: *“a facilidade de compreensão e manutenção depende muito de como a arquitetura é implementada e gerenciada”*.

Em suma, a complexidade e a facilidade de manutenção e compreensão podem variar consideravelmente entre arquiteturas de monolitos e de micro-serviços. Não existe uma resposta definitiva, pois a escolha entre micro-serviços e monolitos depende muito do contexto específico, incluindo a natureza do projeto, a equipe de desenvolvimento e as necessidades do negócio.

3.6.6 Perguntas específicas: Testes e entregas contínuas

A análise das respostas dos entrevistados revela uma divisão entre as opiniões sobre qual arquitetura facilita mais os testes automatizados e o *deploy* contínuo. Alguns entrevistados (E1, E3, E4, E5, E6 e E9) consideram que a arquitetura de micro-serviços é mais favorável, enquanto outros (E2, E7 e E8) acreditam que a arquitetura de monolitos proporciona um ambiente mais facilitado para essas tarefas. E7 adiciona à esta questão: *"Monolitos por que o código está centralizado. Micro-serviços também é possível, mas requer mais tempo de desenvolvimento."*

Os entrevistados que defenderam a arquitetura de micro-serviços para este tema argumentaram que, por serem partes menores para se fazer o *deploy*, eles são mais rápidos para fazer o *deploy* (E3 e E5) e permitem lidar com menos dependências (E9). Por outro lado, os que defenderam os monolitos sustentam que esta arquitetura facilita a realização de testes automatizados e *deploys* contínuos, conforme E8: *"envolve apenas um artefato, então você terá o build e os testes em um lugar só. Quando você roda o pacote e tudo está funcionando, você sabe que o código vai funcionar quando for implantado"*.

Quanto à estabilidade e confiabilidade do sistema ao fazer o *deploy*, boa parte dos entrevistados (E1, E3, E4 e E8) concorda que os micro-serviços oferecem vantagens. Eles destacam que os micro-serviços normalmente têm mais automações (E1) e que, se algo der errado, o problema estará localizado em apenas uma parte do sistema (E3). Um entrevistado mencionou que, na experiência dele, um micro-serviço não é pior do que um monolito em termos de estabilidade e confiabilidade (E8).

No entanto, vale ressaltar que a escolha entre micro-serviços e um sistema monolítico depende de vários fatores, incluindo o tamanho e a maturidade da equipe de desenvolvimento, a complexidade do sistema e as expectativas de escalabilidade e manutenção a longo prazo (E9). Isso indica que, embora existam tendências gerais, a escolha da arquitetura mais adequada pode variar dependendo do contexto específico.

3.6.7 Perguntas específicas: Tamanho de projeto

Em relação à adequação da arquitetura para projetos de diferentes tamanhos e complexidades, a opinião dos entrevistados foi unânime. Os nove entrevistados concordaram que a arquitetura de monolitos é mais adequada para projetos pequenos e de baixa complexidade, enquanto a arquitetura de micro-serviços é mais adequada para projetos grandes e de alta complexidade.

Para projetos pequenos e de baixa complexidade, os entrevistados indicaram preferência pela arquitetura de monolitos. Esta preferência pode ser atribuída à simplici-

dade de implementação e manutenção desta arquitetura, que não requer a coordenação entre diferentes serviços, como é o caso dos micro-serviços. Além disso, projetos pequenos e de baixa complexidade, geralmente, não necessitam das vantagens oferecidas pelos micro-serviços, como escalabilidade e independência de serviços.

Por outro lado, para projetos grandes e de alta complexidade, os entrevistados manifestaram uma preferência clara pela arquitetura de micro-serviços. Esta preferência pode ser justificada pela capacidade dos micro-serviços de lidar com a complexidade através da decomposição do sistema em serviços menores e independentes. Esta decomposição facilita o desenvolvimento, pois permite que diferentes equipes trabalhem em diferentes serviços de forma simultânea e sem interferências. Além disso, a arquitetura de micro-serviços permite a escalabilidade de partes específicas do sistema, o que é especialmente útil em projetos grandes.

Esses resultados estão em linha com as descobertas da literatura existente, que aponta os micro-serviços como uma solução eficaz para a gestão da complexidade em projetos de *software* de grande escala. No entanto, também enfatiza a importância de considerar o tamanho e a complexidade do projeto ao escolher a arquitetura apropriada, visto que a adoção de micro-serviços pode trazer desafios adicionais e aumentar o tempo de desenvolvimento em projetos menores ou menos complexos.

3.6.8 Perguntas específicas: Tempo de desenvolvimento

A análise das respostas dos entrevistados revela uma variedade de opiniões sobre o impacto da arquitetura de micro-serviços no tempo de desenvolvimento de *software* em comparação com a arquitetura de monolitos.

Vários entrevistados notaram que, enquanto a arquitetura de micro-serviços pode inicialmente levar mais tempo para ser configurada e desenvolvida devido à complexidade de manter múltiplos serviços (E1), o potencial para o desenvolvimento paralelo e a escalabilidade a longo prazo pode compensar o tempo de desenvolvimento inicial (E2 e E5). Isto é, embora a configuração inicial de um monolito possa ser mais rápida, a complexidade aumenta à medida que o projeto cresce, tornando a manutenção e a expansão mais desafiadoras.

> *"Eu acredito que a arquitetura de monolitos é mais rápido pra começar a desenvolver, pra começar um projeto, porque tu só configura um repositório, um banco de dados, um deploy e assim por diante. Mas com o passar do tempo, conforme o projeto for crescendo e ficando mais complexo, os micro-serviços são mais simples."* (Entrevistado 2)

Além disso, E3 destaca que a complexidade dos micro-serviços pode ser ampliada quando há muitos serviços diferentes para coordenar e diferentes equipes de desen-

volvimento envolvidas. Este é um ponto importante, pois sugere que a gestão eficaz de projetos e equipes é crucial para aproveitar os benefícios dos micro-serviços.

Por outro lado, E6 discorda, sugerindo que os micro-serviços podem ser desenvolvidos mais rapidamente. No entanto, ele não fornece uma justificativa para essa opinião, o que limita nossa capacidade de entender sua perspectiva.

Conforme comenta E7: *"existe um aumento do tempo de desenvolvimento quando utilizada a arquitetura de micro-serviços, mas não é tão significativo"*. Além disso, ele aponta que o uso de micro-serviços pode resultar em custos operacionais e de infraestrutura mais altos.

E8 oferece uma visão detalhada de como a eficácia da cooperação da equipe e a burocracia organizacional podem afetar o tempo de desenvolvimento em uma arquitetura de micro-serviços. Ele destaca que, em um ambiente burocrático, a necessidade de cumprir os requisitos de entrega pode aumentar o tempo de desenvolvimento, especialmente se várias equipes precisam coordenar seus esforços.

Por fim, E9 destaca a importância de ter uma infraestrutura de qualidade ao implementar uma arquitetura de micro-serviços. Ele menciona que, se a infraestrutura estiver bem abstraída, um sistema baseado em micro-serviços pode ser muito eficiente e produtivo.

Em suma, as opiniões dos entrevistados variam, mas a maioria concorda que, embora a arquitetura de micro-serviços possa levar mais tempo para ser implementada inicialmente, sua escalabilidade e potencial para desenvolvimento paralelo podem compensar o tempo extra gasto no início do projeto. Além disso, a eficácia da gestão do projeto, a qualidade da infraestrutura e a burocracia organizacional foram identificadas como fatores que podem influenciar o tempo de desenvolvimento.

3.6.9 Outras considerações

Através da análise das respostas coletadas na pesquisa realizada, podemos observar que a escolha entre a arquitetura de micro-serviços e a arquitetura de monolitos é influenciada por uma variedade de fatores que vão além das questões puramente técnicas. As decisões são muitas vezes moldadas pelo contexto do projeto, pela natureza do negócio e pelas características da equipe.

Tal como E3 aponta, a escolha da arquitetura deve ser feita de acordo com o projeto e sua complexidade. E2 acrescenta que o tamanho da equipe também influencia a decisão, sugerindo monolitos para equipes pequenas e micro-serviços para projetos que vão escalar. Isto é corroborado pelo E7, que argumenta que monolitos são mais adequados para projetos pequenos, devido às considerações de manutenção e configuração, enquanto micro-serviços fazem mais sentido quando se precisa de um sistema que escale.

A relação entre o domínio do negócio e a escolha da arquitetura é outro tema recorrente entre os entrevistados. E8 argumenta que a escolha da arquitetura de *software* deve ser influenciada pelo domínio de negócio, pois ter um time que compreende profundamente o negócio pode trazer percepções valiosas para a definição da arquitetura. Este ponto é reforçado pelo E4, que considera o domínio do negócio como um dos principais fatores a serem considerados na escolha da arquitetura.

Quando questionados sobre vantagens e desvantagens específicas, os entrevistados destacaram vários pontos. E1 notou que monolitos são mais simples, enquanto micro-serviços são mais complexos. E7 apontou: *"O lado bom da arquitetura de micro-serviços é que pode se utilizar diversas tecnologias para o desenvolvimento de cada serviço, onde uma linguagem mais apropriada para cada tipo de tarefa pode ser selecionada, e esta pode ser diferente do resto do sistema."* Por outro lado, E9 destacou a dificuldade de manutenção dos monolitos, especialmente à medida que envelhecem e se tornam sistemas legados.

Os entrevistados também compartilharam exemplos concretos de projetos onde uma arquitetura se destacou em relação à outra. E3 mencionou um cenário onde um projeto precisava de um processamento específico de volume de dados, para o qual foi desenvolvido um micro-serviço que posteriormente se integrava ao monolito. E7 compartilhou um exemplo de um serviço desenvolvido utilizando a arquitetura de micro-serviços, que permitiu uma maior independência de infraestrutura.

> *"Houve um cenário onde um projeto precisava de um processamento específico de volume de dados em que a escolha da tecnologia não permitia que essa função fosse integrada ao monolito. Para este problema, foi desenvolvido um micro-serviço que fazia essa função e posteriormente se integrava ao monolito."* (Entrevistado 3)

Em suma, a análise das respostas dos entrevistados sugere que a escolha entre a arquitetura de micro-serviços e a arquitetura de monolitos é multifacetada e depende de uma série de fatores, incluindo a natureza do projeto, a experiência da equipe e o domínio do negócio. Ambas as arquiteturas têm suas vantagens e desvantagens, e a decisão deve ser baseada em uma avaliação cuidadosa desses aspectos em relação às necessidades específicas do projeto.

3.6.10 Conclusão da análise da *survey*

Através da análise das entrevistas, fica evidente que a escolha entre uma arquitetura de monolitos e de micro-serviços depende de uma variedade de fatores. Estes incluem o tamanho e a complexidade do projeto, a experiência e maturidade da equipe de

desenvolvimento, o domínio do negócio, assim como as necessidades de escalabilidade, manutenção e testes.

No que diz respeito ao tempo de desenvolvimento, a opinião predominante é que a arquitetura de micro-serviços pode levar mais tempo para ser implementada inicialmente, devido à sua complexidade e à necessidade de coordenação entre múltiplos serviços. No entanto, uma vez implementados, os micro-serviços podem oferecer benefícios significativos em termos de escalabilidade, manutenção e desenvolvimento paralelo, que podem compensar o tempo adicional investido na sua implementação inicial.

As opiniões dos entrevistados sugerem que a arquitetura de monolitos pode ser mais adequada para projetos menores e menos complexos, devido à sua simplicidade de implementação e manutenção. Por outro lado, a arquitetura de micro-serviços parece ser preferida para projetos de maior escala e complexidade, devido à sua capacidade de lidar com a complexidade através da decomposição do sistema em serviços menores e independentes.

Importante ressaltar que os entrevistados concordam que a escolha da arquitetura deve ser feita com base no contexto específico do projeto. Cada arquitetura tem suas vantagens e desvantagens, e a melhor escolha depende das necessidades e características específicas do projeto, da equipe de desenvolvimento e do domínio do negócio.

Assim, com base nesta análise, podemos concluir que não existe uma resposta definitiva para a questão de se a utilização da arquitetura de micro-serviços acarreta em um aumento do tempo de desenvolvimento de *software*. Em vez disso, a resposta depende do contexto específico do projeto.

No entanto, a análise das entrevistas oferece percepções valiosas para a criação de um guia de recomendações a ser utilizado por equipes de desenvolvimento de *software*. Este guia poderá incluir, por exemplo, recomendações para considerar o tamanho e a complexidade do projeto, a experiência e maturidade da equipe, o domínio do negócio, assim como as necessidades de escalabilidade, manutenção e testes, ao escolher entre uma arquitetura de monolitos e de micro-serviços. Além disso, deverá incluir a sugestão de considerar a arquitetura de micro-serviços para projetos de maior escala e complexidade, e a arquitetura de monolitos para projetos menores e menos complexos.

Em suma, esta pesquisa oferece uma contribuição valiosa para o entendimento das diferenças entre as arquiteturas de micro-serviços e de monolitos, e fornece perspectivas úteis para a tomada de decisões sobre a escolha da arquitetura em projetos de desenvolvimento de *software*.

4. DISCUSSÃO

Dados os objetivos específicos delineados para esta pesquisa (Seção 1.3), identificou-se na literatura (OBJ1) e nas práticas da indústria (OBJ2) diversas características relacionadas ao uso da arquitetura de micro-serviços e de monolitos. Por meio de uma Revisão de Literatura Multivocal, foram encontrados os principais desafios associados à adoção de micro-serviços, conforme reportado em estudos recentes. Os resultados apontam para uma série de obstáculos operacionais e estratégicos que podem influenciar negativamente o tempo de desenvolvimento e a eficácia da implementação dessa arquitetura. Isso inclui complexidades adicionais em termos de operações de *DevOps*, monitoramento, comunicação entre serviços, depuração, versionamento e segurança, como demonstra a Figura 4.1. Apesar dos benefícios inegáveis, como escalabilidade e isolamento de falhas, a transição para micro-serviços exige um planejamento cuidadoso e uma compreensão aprofundada dos desafios envolvidos.

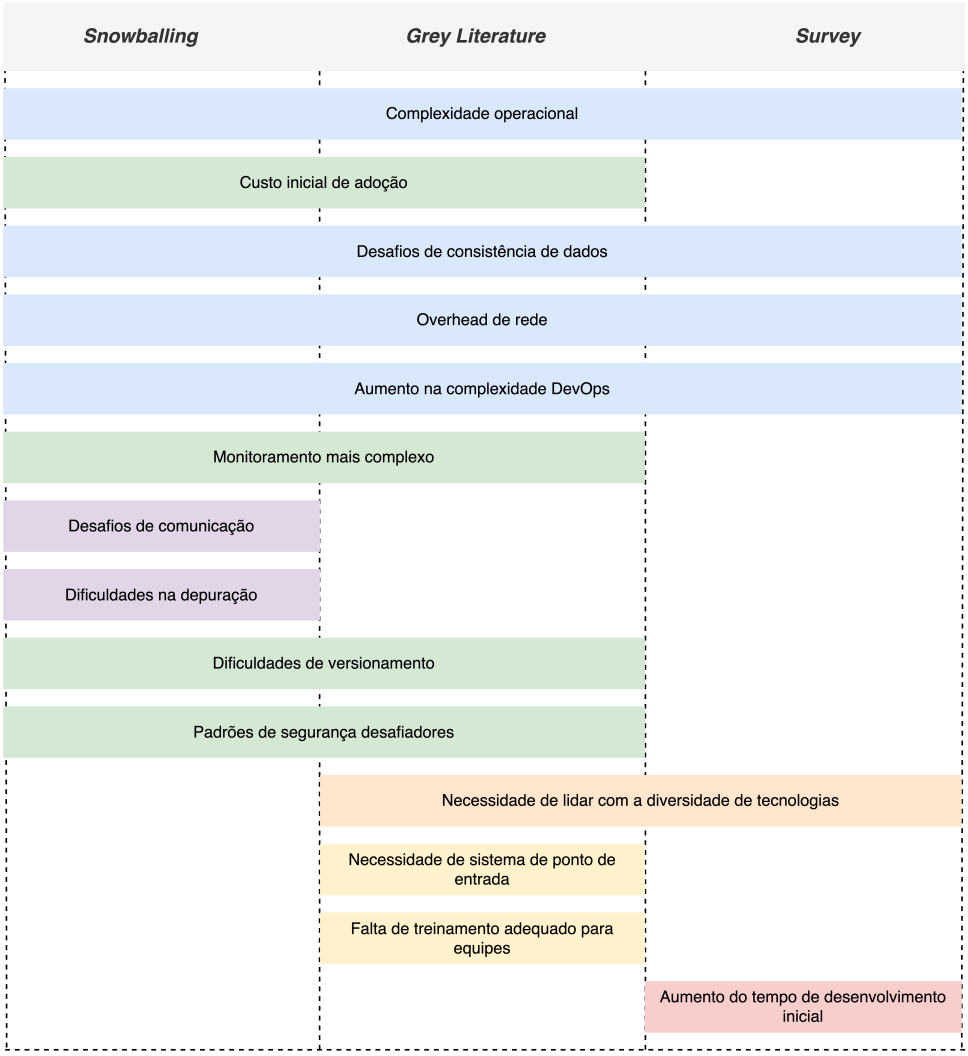

Figura 4.1 – Desafios relacionados a micro-serviços e em quais etapas metodológicas foram encontrados.

Complementando essa análise, uma *survey* com especialistas em desenvolvimento de *software* foi realizada para capturar percepções práticas e avaliar como a teoria se alinha às experiências de campo. Os profissionais entrevistados reforçam a ideia de que a escolha entre arquiteturas monolíticas e de micro-serviços não é binária, mas sim influenciada por um espectro de fatores, como o tamanho e a complexidade do projeto, a maturidade da equipe e as particularidades do domínio do negócio. A pesquisa revela um consenso de que os micro-serviços podem exigir mais tempo para uma implementação inicial adequada, mas oferecem vantagens a longo prazo que podem compensar o investimento inicial.

Esta pesquisa, portanto, se propôs a elucidar o dilema enfrentado por muitas organizações no que tange à escolha da arquitetura de *software* mais adequada, fornecendo um guia de recomendações sobre a escolha destas arquiteturas (OBJ3), com o objetivo de contribuir com conhecimentos que possam orientar desenvolvedores, arquitetos de *software* e tomadores de decisão na seleção da abordagem arquitetural que melhor se alinhe às necessidades e objetivos de seus projetos, alinhados com a realidade prática das operações de desenvolvimento de *software*.

Conforme mencionado pelos entrevistados E8 e E9, corroborados também por dados extraídos da Revisão de Literatura Multivocal ([5], [24], [33], [26], [28], [9], [15], [6], [44] [39], [29], [58]), foram identificadas três categorias principais de soluções para os problemas em questão. Essas categorias são: *Ferramentas*, *Processos* e *Padrões*, descritas a seguir:

- **Ferramentas:** Esta categoria se refere às soluções tecnológicas que podem ser aplicadas para superar desafios específicos no uso de microserviços. Logo, inclui-se ferramentas de software para monitoramento, teste, implantação e gerenciamento de micro-serviços.

- **Processos:** Aborda as melhorias ou alterações nos procedimentos operacionais que podem facilitar o uso eficaz de micro-serviços. Desta forma, envolve a adoção de novos métodos de trabalho, como DevOps ou Agile, ou a implementação de novas práticas de gestão.

- **Padrões:** Diz respeito à implementação de normas ou convenções de design que podem ajudar a garantir a coesão e a eficiência na arquitetura de micro-serviços. Em sendo assim, inclui-se a adoção de padrões de design específicos, convenções de nomenclatura ou diretrizes de codificação.

4.1 Guia de Recomendações

Conforme exposto nas etapas anteriores desta pesquisa, este guia visa fornecer uma discussão detalhada e recomendações práticas sobre a escolha entre a arquitetura de micro-serviços e monolitos. Para atingir este objetivo, serão detalhados os problemas relacionados ao uso da arquitetura de micro-serviços encontrados nesta pesquisa, de acordo com os dados obtidos nas etapas metodológicas anteriores, conforme apresentado na Figura 4.1. Além disso, serão apresentadas também boas práticas que são eficazes para solucionar o problema mencionado. Este guia pretende oferecer um quadro de referência para ajudar os tomadores de decisão a entender as implicações de cada opção, considerando o contexto específico de sua organização e as necessidades de seus sistemas de *software*.

4.1.1 Complexidade operacional

A complexidade operacional é um desafio comum que surge ao se adotar a arquitetura de micro-serviços em comparação com a arquitetura de monolitos ([5, 24, 33, 26, 28, 9, 15, 6, 44, 39, 29, 58] e E1, E2, E3, E5, E8, E9). Este problema é caracterizado pela necessidade de desenvolver e manter um grande número de serviços independentes que compõem o sistema como um todo, ao invés de lidar com uma única aplicação monolítica.

Na arquitetura de micro-serviços, cada serviço executa um processo de negócio distinto, opera de forma independente e se comunica com outros serviços por meio de APIs. Este modelo oferece muitos benefícios, como a capacidade de escalar, desenvolver e implantar cada serviço de forma independente, bem como o uso de diferentes tecnologias para cada serviço conforme apropriado. No entanto, gerenciar muitos serviços pequenos pode ser muito mais complexo do que gerenciar uma única aplicação utilizando a arquitetura de monolitos.

A complexidade operacional decorre de várias fontes. Primeiramente, cada micro-serviço pode ter seus próprios requisitos de infraestrutura, o que significa que você precisa gerenciar várias configurações de ambiente de execução. Além disso, monitorar e solucionar problemas em um sistema de micro-serviços pode ser desafiador, já que você precisa rastrear as interações entre vários serviços ao invés de um único fluxo de controle. Também há a questão da consistência de dados, pois cada serviço pode ter seu próprio banco de dados, e garantir a consistência entre eles tende a ser desafiador.

Uma abordagem comum para lidar com a complexidade operacional é adotar ferramentas e práticas que foram projetadas especificamente para ambientes de micro-serviços. Por exemplo, as plataformas de orquestração de contêineres podem automatizar

o gerenciamento e a escalabilidade dos micro-serviços. Para monitoramento e solução de problemas, as ferramentas de observabilidade podem ajudar a rastrear as interações entre os serviços e identificar os gargalos de desempenho. E para a consistência de dados, padrões podem ser usados para garantir a consistência em transações que abrangem vários serviços.

Em resumo, a complexidade operacional é um desafio significativo na arquitetura de micro-serviços, mas com as ferramentas e práticas corretas, é possível gerenciar essa complexidade e aproveitar os benefícios que os micro-serviços podem oferecer.

Problema
"Complexidade operacional"

Caracterizado pela necessidade de gerenciar muitos serviços independentes. Decorre da necessidade de gerenciar requisitos de infraestrutura distintos, monitorar e resolver problemas em um sistema complexo, e garantir a consistência de dados entre múltiplos serviços.

Recomendação
Adoção de ferramentas e práticas projetadas para micro-serviços. Plataformas de orquestração de contêineres podem gerenciar e escalar serviços, as ferramentas de observabilidade podem auxiliar no monitoramento e solução de problemas, e padrões podem ser estabelecidos para garantir a consistência de dados em transações que envolvem vários serviços.

Categoria
Ferramentas

4.1.2 Custo inicial de adoção

A arquitetura de micro-serviços implica em um alto investimento em termos de tempo, esforço e recursos. Neste sentido, inclui-se o tempo e os recursos necessários para projetar e implementar a nova arquitetura, treinar a equipe no novo paradigma e migrar os dados e funcionalidades existentes para os novos serviços. Além disso, a adoção de micro-serviços também pode exigir a compra de novas ferramentas ou tecnologias, bem como a implementação de novos processos para gerenciar e monitorar os muitos serviços diferentes ([26, 28, 9, 18, 44, 20, 10]).

Uma boa prática para mitigar este problema é adotar uma abordagem gradual para a adoção de micro-serviços. Em vez de tentar mover todo o sistema de uma só vez, pode ser mais eficaz começar por dividir apenas um ou dois aspectos do sistema em

micro-serviços e aprender com essa experiência. Essa abordagem pode permitir que a organização ganhe uma melhor compreensão dos desafios e benefícios da arquitetura de micro-serviços antes de se comprometer com uma adoção em larga escala.

Além disso, é importante garantir que a organização tem a infraestrutura e as habilidades necessárias para gerenciar uma arquitetura de micro-serviços antes de fazer a transição. Com isso, pode-se incluir investimentos em ferramentas de gerenciamento de micro-serviços, bem como treinar a equipe em práticas de desenvolvimento e operações de micro-serviços.

Portanto, enquanto a arquitetura de micro-serviços pode oferecer muitos benefícios, também vem com um custo inicial de adoção que pode ser um desafio para muitas organizações. No entanto, com uma abordagem cuidadosa e gradual, é possível mitigar esse desafio e aproveitar os benefícios dos micro-serviços.

Problema
"Custo inicial de adoção"

A transição do modelo de monolito para o de micro-serviços, que divide a aplicação em serviços menores e independentes, pode implicar em dificuldades de escalabilidade e alterações no sistema, podendo exigir novas ferramentas, tecnologias e implementação de processos.

Recomendação
Adoção de uma abordagem gradual, começando por dividir apenas alguns aspectos do sistema em micro-serviços. Além disso, é crucial investir em ferramentas de gerenciamento de micro-serviços e treinar a equipe em práticas de desenvolvimento e operações de micro-serviços, preparando a organização para a transição.

Categoria
Ferramentas, Processos

4.1.3 Desafios de consistência de dados

O desafio da consistência de dados é um dos problemas enfrentados quando se utiliza a arquitetura de micro-serviços, em comparação com a arquitetura de monolitos. Esse desafio surge devido à natureza descentralizada e distribuída dos micro-serviços (que pode inclusive estar em regiões distintas), onde cada serviço pode ter seu próprio banco de dados ou estado de persistência. Isto é, em um cenário de micro-serviços, os dados não residem em um único lugar, mas são distribuídos entre vários serviços ([26, 9, 15, 18, 20] e E1).

Essa configuração pode levar a problemas de consistência de dados. Por exemplo, se um serviço faz uma alteração em um dado e outro serviço que utiliza o mesmo dado não é informado sobre essa alteração, pode haver inconsistências. Esse problema é exacerbado em ambientes de alta latência ou quando ocorrem falhas de rede, onde as atualizações de dados podem não ser propagadas para todos os serviços em tempo hábil.

Em contraste, na arquitetura de monolitos, todos os componentes do aplicativo compartilham o mesmo banco de dados. Dessa forma, qualquer alteração em um dado é imediatamente refletida em todo o aplicativo, garantindo a consistência dos dados.

Para resolver o desafio de consistência de dados na arquitetura de micro-serviços, existem várias boas práticas que podem ser seguidas. Uma delas é a implementação de padrões específicos de *design*, onde uma operação que envolve vários serviços é dividida em várias transações locais. Se uma dessas transações falhar, as transações compensatórias são disparadas para desfazer as transações anteriores, mantendo a consistência dos dados.

Outra boa prática é a utilização de eventos ou mensagens para comunicar alterações entre serviços. Por exemplo, se um serviço altera um dado, ele pode emitir um evento que informa os outros serviços sobre a alteração. Isso permite que todos os serviços que dependem desse dado se atualizem e mantenham a consistência dos dados.

Além disso, é recomendável limitar o escopo dos dados gerenciados por cada serviço, para minimizar a necessidade de coordenação e sincronização entre serviços. Cada micro-serviço deve ser o único responsável por seus próprios dados, reduzindo as dependências e tornando o sistema mais resiliente.

Problema
“Desafios de consistência de dados”

Cada serviço pode ter seu próprio banco de dados, tornando os dados descentralizados e distribuídos. Esse cenário pode levar à inconsistência de dados, pois se um serviço altera um dado e os outros não são informados, surgem inconsistências. O problema aumenta em ambientes de alta latência ou falhas de rede, pois as atualizações podem não ser propagadas em tempo hábil.

Recomendação
Adoção de padrões de *design* específicos dividindo operações em transações locais. Se uma falha, transações compensatórias são disparadas para manter a consistência dos dados. A utilização de eventos para comunicar alterações entre serviços também é recomendada. Além disso, é aconselhável limitar o escopo dos dados gerenciados por cada serviço, tornando cada micro-serviço responsável por seus próprios dados, reduzindo dependências e aumentando a resiliência.

Categoria
Ferramentas, Padrões, Processos

4.1.4 *Overhead* de rede

Overhead de rede é a sobrecarga adicional que é imposta sobre a comunicação de dados em uma rede devido ao processo de empacotamento, desempacotamento, encaminhamento, processamento e entrega dos dados. Em uma arquitetura de micro-serviços, os serviços estão geralmente distribuídos e precisam se comunicar entre si através da rede. Cada solicitação e resposta entre micro-serviços implica em uma série de operações de rede, como estabelecimento de conexão, transferência de dados, espera por resposta, entre outras, o que pode resultar em um aumento no tempo total de processamento ([24, 26, 32, 15, 20, 10] e E4, E9).

Este é um problema porque o *overhead* de rede pode resultar em latência adicional, o que pode afetar adversamente o desempenho e a eficiência do sistema como um todo. Além disso, a comunicação de rede entre micro-serviços também pode ser afetada por questões como a confiabilidade da rede e a segurança dos dados durante a transmissão.

Uma boa prática para lidar com o desafio do *overhead* de rede em arquiteturas de micro-serviços é a adoção de padrões e técnicas de *design* que minimizam a comunicação entre serviços. Por exemplo, o uso de padrões de *design* como o “CQRS” (*Command Query Responsibility Segregation*) e “*Event Sourcing*” pode ajudar a reduzir a necessidade

de comunicação síncrona entre serviços, substituindo-a por comunicação assíncrona baseada em eventos.

Outra prática recomendada é o uso de técnicas de compactação de dados para reduzir a quantidade de dados que precisam ser transmitidos pela rede. A utilização de protocolos de comunicação mais eficientes, como o gRPC, em vez de HTTP/JSON, também pode ajudar a reduzir o *overhead* de rede.

Além disso, a localização física dos serviços também pode ser considerada. Por exemplo, serviços que precisam se comunicar frequentemente podem ser colocados mais próximos um do outro na rede para reduzir a latência. Isso pode ser realizado através de uma abordagem de *design* chamada "*affinity-based routing*".

Por último, mas não menos importante, o monitoramento constante e a observação dos padrões de tráfego da rede podem ajudar a identificar gargalos e otimizar a configuração da rede para melhor atender às necessidades dos micro-serviços.

Problema
"*Overhead* de rede"

Refere-se à sobrecarga adicional na comunicação de dados em uma rede, causada por processos como empacotamento, desempacotamento e entrega de dados. Em uma arquitetura de micro-serviços, essa sobrecarga pode resultar em latência extra, afetando o desempenho e a eficiência do sistema como um todo. Além disso, a comunicação entre micro-serviços pode ser afetada por questões como confiabilidade da rede e segurança dos dados durante a transmissão.

Recomendação
Adoção de padrões de *design* que minimizam a comunicação entre serviços, como CQRS e Event Sourcing; o uso de técnicas de compactação de dados e protocolos de comunicação mais eficientes, como o gRPC; a consideração da localização física dos serviços para reduzir a latência; e o monitoramento constante dos padrões de tráfego de rede para identificar e resolver gargalos.

Categoria
Ferramentas, Padrões

4.1.5 Aumento da complexidade de *DevOps*

DevOps é uma prática que visa unir o desenvolvimento de *software* (Dev) e as operações de TI (Ops), com o objetivo de entregar *software* de alta qualidade de forma

mais rápida e eficiente. No entanto, quando aplicado em um ambiente de micro-serviços, o *DevOps* pode se tornar significativamente mais complexo ([5, 26, 15, 39, 20] e E8).

A complexidade aumentada surge porque, em vez de lidar com um único aplicativo monolítico, a equipe de *DevOps* deve agora gerenciar uma infinidade de serviços menores, cada um com seu próprio ciclo de vida, dependências e exigências de infraestrutura. Isso pode levar a desafios em termos de monitoramento, implantação, escalabilidade e manutenção. Por exemplo, a implantação de atualizações pode se tornar um processo muito mais complexo, pois é preciso garantir que todas as peças do sistema continuem a funcionar em harmonia mesmo após a atualização de um único serviço.

Isso é um problema porque pode levar a erros, ineficiências e atrasos na entrega de *software*. A coordenação de vários micro-serviços pode ser uma tarefa complicada que requer uma quantidade significativa de tempo e recursos. Além disso, a complexidade adicional pode tornar o sistema mais difícil de entender e gerenciar, o que pode resultar em problemas de desempenho e segurança.

Para lidar com a complexidade aumentada de *DevOps* em um ambiente de micro-serviços, uma boa prática é adotar ferramentas e práticas que facilitem o gerenciamento, a implantação e o monitoramento de serviços micro. Esse fato pode incluir o uso de contêineres, para empacotar e isolar serviços, o que pode facilitar a implantação e a escalabilidade. Além disso, ferramentas de orquestração podem ser usadas para automatizar o gerenciamento e a escalabilidade de serviços em contêineres.

Outra prática recomendada é adotar uma abordagem de integração contínua e entrega contínua (CI/CD). Esse fato envolve a automatização do processo de teste e implantação de *software*, o que pode ajudar a garantir que todos os serviços funcionem corretamente juntos e reduzir o risco de erros e atrasos.

Também é importante investir em monitoramento e *logs* robustos para acompanhar o desempenho e o status de cada serviço. Neste sentido, este fato pode ajudar a identificar e resolver rapidamente quaisquer problemas que surjam, antes que eles possam afetar o resto do sistema.

Problema
"Aumento da complexidade de *DevOps*"

DevOps é uma prática que une desenvolvimento de *software* e operações de TI para entregar *software* de alta qualidade rapidamente. No entanto, em um ambiente de micro-serviços, a complexidade aumenta pois há muitos serviços menores para gerenciar, cada um com seu próprio ciclo de vida e exigências. Isso pode causar desafios em monitoramento, implantação, escalabilidade e manutenção, levando a erros, ineficiências e atrasos na entrega de *software*.

Recomendação
Adoção de ferramentas e práticas que facilitem o gerenciamento, implantação e monitoramento dos serviços. Isso inclui o uso de contêineres e ferramentas de orquestração, a prática de integração contínua e entrega contínua (CI/CD), e investimento em monitoramento robusto e *logs* para rastrear o desempenho e status de cada serviço, identificando e resolvendo rapidamente quaisquer problemas.

Categoria
Ferramentas

4.1.6 Monitoramento mais complexo

Na arquitetura de monolitos, todos os componentes da aplicação são compactados em uma única unidade indivisível. Isso significa que o monitoramento e a manutenção são centralizados e, portanto, mais simples. No entanto, com a arquitetura de micro-serviços, a aplicação é dividida em vários serviços menores e independentes, cada um com sua própria base de código e infraestrutura. Embora esta abordagem ofereça várias vantagens, como a possibilidade de escalar serviços individualmente e a agilidade para atualizar ou alterar um serviço sem impactar os outros, também torna o monitoramento muito mais complexo. Cada micro-serviço pode ter seu próprio conjunto de métricas, *logs* e rastreios para monitorar, o que pode resultar em uma grande quantidade de dados para gerenciar ([5, 24, 12, 26, 28, 9, 15, 39, 29, 20]).

Esse aumento na complexidade do monitoramento pode dificultar a identificação de problemas e a tomada de decisões informadas. Por exemplo, se um serviço específico está falhando ou se comportando de maneira inadequada, pode ser difícil identificar o problema sem uma visibilidade clara de como todos os serviços estão operando juntos. Não há uma forma clara de definir métricas que sejam abrangentes e funcionais para todos os serviços, devido à natureza da arquitetura, onde cada serviço vai atender demandas distintas, em tempo e com capacidades próprias a eles. Além disso, também pode ser

mais desafiador identificar gargalos de desempenho ou problemas de segurança em uma arquitetura de micro-serviços devido à sua natureza distribuída.

Para enfrentar esse desafio, uma boa prática é a implantação de uma solução de monitoramento distribuído. Este tipo de solução pode coletar, correlacionar e visualizar dados de vários serviços, proporcionando uma visão unificada do desempenho do sistema. Isso pode facilitar a identificação de problemas e a análise de tendências ao longo do tempo. Além disso, a implementação de alertas automatizados pode ajudar a identificar problemas em tempo real, permitindo uma resposta rápida.

Além disso, a adoção de uma abordagem padronizada para registrar e rastrear informações em todos os serviços pode ser benéfica. Envolvendo o uso de formatos de *log* consistentes para entender como as solicitações se movem através de diferentes serviços. Essas práticas podem facilitar a análise de dados e a identificação de problemas.

Em resumo, embora o monitoramento seja mais complexo em uma arquitetura de micro-serviços, a implementação de estratégias e ferramentas adequadas pode ajudar a gerenciar essa complexidade e garantir que a aplicação continue a funcionar de forma eficiente e confiável.

Problema
"Monitoramento mais complexo"

Monitoramento em arquitetura de micro-serviços é um desafio devido à independência de cada serviço. A individualidade de cada serviço, com suas próprias métricas, *logs* e rastreios, resulta em uma grande quantidade de dados para gerenciar. Isso pode tornar mais difícil identificar problemas, gargalos de desempenho ou questões de segurança, pois a visibilidade de como todos os serviços operam juntos pode ser limitada.

Recomendação
Adoção de uma solução de monitoramento distribuído para uma visão unificada do sistema. Utilização de alertas automatizados para identificar problemas em tempo real. Adoção de uma abordagem padronizada para registrar e rastrear informações em todos os serviços, usando formatos de *logs* consistentes e rastreamento distribuído. Essas práticas melhoram a análise de dados e a identificação de problemas.

Categoria
Ferramentas

4.1.7 Desafios de comunicação

O desafio da comunicação é um dos problemas mais significativos quando se trabalha com uma arquitetura de micro-serviços, em comparação com uma arquitetura de monolitos. Na arquitetura de monolitos, todos os componentes do sistema estão interligados e funcionam como uma única unidade. Portanto, a comunicação entre diferentes partes do sistema é geralmente direta e simplificada ([5, 24, 12, 33, 26, 32, 28, 9, 15]).

No entanto, na arquitetura de micro-serviços, cada serviço é um componente separado e independente que funciona em seu próprio processo. Esses serviços precisam se comunicar entre si para realizar tarefas do sistema. A comunicação entre os serviços é feita através de protocolos de rede, geralmente HTTP/REST ou mensagens assíncronas. Este é o ponto onde o desafio da comunicação se torna evidente. A comunicação entre os serviços pode ser complexa, lenta e propensa a falhas de rede. Além disso, o encadeamento de chamadas entre vários serviços pode levar a problemas de latência e dificuldades de rastreamento.

Este desafio de comunicação é um problema porque pode afetar negativamente a eficiência e a confiabilidade do sistema. Se um serviço não for capaz de se comunicar de maneira eficaz com outro, pode haver atrasos na execução das tais tarefas. Além disso, se um serviço falhar, pode ser difícil identificar a causa do problema devido à complexidade da comunicação entre os serviços.

Uma boa prática para lidar com este desafio é a implementação de padrões de *design* de comunicação bem estabelecidos, como a comunicação síncrona e assíncrona. A comunicação síncrona é útil para operações que precisam de uma resposta imediata, enquanto a comunicação assíncrona pode ser usada para operações que podem ser processadas em segundo plano. Além disso, é importante implementar mecanismos de tolerância a falhas, como o padrão de *circuit breaker*, para prevenir a falha de um serviço que afeta o sistema inteiro.

Outra prática recomendada é a utilização de ferramentas de rastreamento e monitoramento para acompanhar as chamadas entre os serviços. Isso pode ajudar a identificar rapidamente qualquer problema de comunicação entre os serviços. Além disso, a implementação de uma camada de orquestração pode facilitar a coordenação e a comunicação entre os serviços, o que pode ajudar a simplificar a complexidade da comunicação.

Problema
"Desafios de comunicação"

Diferentemente da arquitetura de monolitos, onde a comunicação entre componentes é direta e simplificada, na arquitetura de micro-serviços, cada serviço é um componente independente que precisa se comunicar com outros através de protocolos de rede. Isso pode tornar a comunicação complexa, lenta e sujeita a falhas de rede. O desafio é agravado por problemas de latência e dificuldades de rastreamento em chamadas encadeadas entre serviços. Isso pode afetar a eficiência e a confiabilidade do sistema, causando atrasos na execução das tarefas e dificuldade em identificar falhas.

Recomendação
Implementação de padrões de *design* de comunicação, como a comunicação síncrona e assíncrona. Além disso, a implementação de mecanismos de tolerância a falhas, como o padrão de *circuit breaker*, pode prevenir que a falha de um serviço afete o sistema inteiro. A utilização de ferramentas de rastreamento e monitoramento também é recomendada para identificar rapidamente problemas de comunicação. A implementação de uma camada de orquestração pode facilitar a coordenação e a comunicação entre os serviços, simplificando a complexidade da comunicação.

Categoria
Ferramentas, Padrões

4.1.8 Dificuldades na depuração

Em sistemas monolíticos, a depuração é geralmente direta. O fluxo de controle e os dados passam de uma parte do sistema para outra em um único processo, facilitando a observação e o rastreamento dos problemas. No entanto, em uma arquitetura de micro-serviços, o sistema é dividido em muitos serviços separados, cada um rodando em seu próprio processo e se comunicando com os outros por meio de chamadas de rede. Isso torna a depuração muito mais complexa ([32, 15, 18]).

Este é um problema porque a depuração é um aspecto crucial do desenvolvimento e manutenção de *software*. Quando um *bug* é encontrado, os desenvolvedores precisam ser capazes de rastrear o problema até sua fonte para corrigí-lo. Em uma arquitetura de micro-serviços, isso pode significar rastrear um problema através de vários serviços, cada um potencialmente rodando em uma máquina ou processo diferente, e cada um com seus próprios registros de *log*. Esse fato pode ser uma tarefa demorada e complexa, que pode retardar significativamente o processo de desenvolvimento.

Uma boa prática para lidar com esse desafio é implementar uma estratégia sólida de rastreamento distribuído e gerenciamento de *logs*. O rastreamento distribuído envolve a anexação de um identificador único a todas as chamadas de rede relacionadas a uma única transação ou pedido do usuário. Logo, permite-se que os desenvolvedores rastreiem o fluxo de uma transação através de vários serviços.

As ferramentas de gerenciamento de *log* podem coletar, indexar e analisar registros de todos os serviços em um único lugar, ou seja, é possível tornar mais fácil para os desenvolvedores localizar e analisar os problemas. Além disso, é importante que os micro-serviços sejam projetados com a falha em mente. Isso significa implementar mecanismos de isolamento de falhas, como *timeouts* e *circuit breakers*, para evitar que os problemas em um serviço se espalhem para outros.

Além disso, a adoção de padrões e protocolos de comunicação consistentes em todos os serviços também pode ajudar a simplificar a depuração. Se todos os serviços usam o mesmo protocolo para se comunicar e seguem as mesmas convenções para formatos de mensagem e códigos de erro, isso pode tornar mais fácil para os desenvolvedores entender o que está acontecendo quando problemas precisam ser corrigidos.

Problema
"Dificuldades na depuração"

A depuração em sistemas de micro-serviços é complexa devido à divisão do sistema em muitos serviços separados, cada um rodando em seu próprio processo e se comunicando por meio de chamadas de rede. Isso torna difícil rastrear problemas até sua origem, o que é crucial para o desenvolvimento e manutenção do *software*.

Recomendação
Adoção de uma estratégia sólida de rastreamento distribuído e gerenciamento de *logs* associado ao uso de ferramentas para coletar e analisar registros de todos os serviços em um só lugar. Implementação de micro-serviços com a falha em mente, adotando mecanismos de isolamento de falhas. Adoção de padrões e protocolos de comunicação consistentes para simplificar a depuração.

Categoria
Ferramentas, Padrões, Processos

4.1.9 Dificuldades de versionamento

Este problema se refere à complexidade adicional que é introduzida quando os serviços são divididos em componentes menores, cada um com seu próprio ciclo de vida, em comparação com a arquitetura de monolitos, onde cada mudança de código resulta em uma nova versão do todo ([12, 9, 6, 29, 20, 10]).

Na arquitetura de micro-serviços, cada serviço pode ser desenvolvido, implantado e escalado de forma independente. Isso significa que diferentes equipes podem trabalhar em diferentes serviços ao mesmo tempo, cada uma com sua própria agenda e necessidades. Como resultado, pode ser difícil acompanhar quais versões de quais serviços estão atualmente em uso, e quais devem ser usadas juntas. Além disso, quando um serviço é atualizado, pode haver consequências inesperadas para outros serviços que dependem dele.

Esse é um problema significativo, pois pode levar a inconsistências e falhas no sistema. Por exemplo, se um serviço é atualizado para uma nova versão que não é compatível com um serviço dependente, o serviço dependente pode parar de funcionar corretamente. Além disso, o versionamento inadequado pode tornar difícil rastrear a causa de um problema quando ele ocorre, pois não está claro qual versão de qual serviço está causando o problema.

Uma boa prática para gerenciar o versionamento em uma arquitetura de micro-serviços é implementar uma estratégia de versionamento semântico. Isso implica em dar a cada versão um número significativo que indica o grau de mudança do serviço, por exemplo, um aumento no número de versão maior indica uma alteração incompatível.

Outra abordagem é o uso de um registro de serviço ou um orquestrador de serviços, que pode rastrear quais versões de quais serviços estão atualmente em uso. Isso pode ajudar a garantir que somente combinações compatíveis de serviços sejam usadas juntas.

Além disso, é importante implementar práticas robustas de teste e integração contínua. Ao testar os serviços juntos antes de eles serem implantados, você pode identificar e corrigir problemas de incompatibilidade antes que eles afetem o sistema em produção.

Problema
“Dificuldades de versionamento”

O problema remete à complexidade na gestão de versões em arquiteturas de micro-serviços. Diferentes equipes trabalham em serviços distintos que são desenvolvidos, implantados e escalados independentemente, tornando difícil rastrear as versões de serviços em uso. Isso pode levar a inconsistências, falhas no sistema e dificuldade em identificar a causa dos problemas devido ao versionamento inadequado.

Recomendação
Para gerenciar efetivamente o versionamento em arquiteturas de micro-serviços, pode-se implementar uma estratégia de versionamento semântico e utilizar um registro de serviço ou orquestrador. Além disso, é crucial implementar práticas robustas de teste e integração contínua para identificar e corrigir problemas de incompatibilidade antes que afetem a produção.

Categoria
Ferramentas, Padrões

4.1.10 Padrões de segurança desafiadores

Diferentemente da arquitetura monolítica, onde toda a aplicação é encapsulada em uma única unidade, os micro-serviços são distribuídos por natureza. Cada serviço é um componente separado que pode ser desenvolvido, implantado e escalado independentemente. Essa descentralização pode complicar a implementação de medidas de segurança que são consistentes em toda a aplicação. Por exemplo, cada serviço pode ter suas próprias políticas de autenticação e autorização, o que pode levar a inconsistências e possíveis vulnerabilidades ([26, 28, 15, 29, 20]).

Além disso, a comunicação entre serviços em uma arquitetura de micro-serviços ocorre através da rede, o que pode ser um ponto de entrada para ataques. Em uma arquitetura de monolitos, a maior parte da comunicação ocorre dentro do mesmo processo, o que geralmente é mais seguro.

Este é um problema que aumenta a superfície de ataque potencial para um invasor. Cada serviço individual pode se tornar um ponto de entrada, e uma vez que um serviço é comprometido, o invasor pode ser capaz de se mover lateralmente através da rede para outros serviços. Isso pode resultar em violações de dados, interrupção do serviço e outros problemas graves.

Uma boa prática para lidar com esse desafio é implementar um plano de segurança abrangente que aborde tanto a segurança em nível de serviço quanto a segurança em nível de aplicação. A segurança em nível de serviço pode incluir coisas como autenticação e autorização robustas, criptografia em repouso e em trânsito, e regularmente

escanear e corrigir vulnerabilidades no código e nas dependências. A segurança em nível de aplicação pode envolver a implementação de um API *gateway* para gerenciar o tráfego entre serviços, usando uma rede privada virtual (VPN) ou uma rede virtual privada de *software* (SDN) para isolar a comunicação entre serviços, e monitorar e registrar a atividade da rede para detectar e responder a incidentes de segurança.

Outra prática recomendada é a adoção de um padrão de "autenticação de serviço a serviço", como o uso de *tokens* JWT (*JSON Web Tokens*) ou certificados mútuos TLS para garantir que apenas serviços autorizados possam se comunicar entre si.

Além disso, a implementação de princípios de segurança de design, como o menor privilégio, o design seguro desde o início e a defesa em profundidade, podem ajudar a garantir que a segurança seja uma consideração primordial em todas as fases do ciclo de vida do desenvolvimento de *software*.

Em resumo, enquanto a arquitetura de micro-serviços oferece muitos benefícios, ela também apresenta desafios de segurança que exigem uma atenção cuidadosa. Com um planejamento e implementação cuidadosos, no entanto, é possível construir aplicações de micro-serviços que são robustas, escaláveis e seguras.

Problema

"Padrões de segurança desafiadores"

Os micro-serviços, ao contrário de uma arquitetura monolítica, são distribuídos e podem ser desenvolvidos e escalados independentemente. Essa descentralização pode dificultar a implementação de medidas de segurança consistentes, resultando em possíveis vulnerabilidades e aumentando a superfície de ataque para invasores. A comunicação entre serviços ocorre através da rede, o que pode ser um ponto de entrada para ataques.

Recomendação

As melhores práticas para lidar com os desafios de segurança em micro-serviços incluem a implementação de um plano de segurança abrangente com medidas em nível de serviço e de aplicação, a adoção de "autenticação de serviço a serviço", e a implementação de princípios de segurança de design. Isso pode incluir autenticação robusta, criptografia, escaneamento de vulnerabilidades, uso de API *gateways*, redes privadas virtuais, monitoramento e registro de atividade de rede, e design seguro desde o início.

Categoria

Ferramentas, Padrões, Processos

4.1.11 Necessidade de lidar com a diversidade de tecnologias

Este desafio surge porque, ao contrário da arquitetura de monolitos, onde todo o sistema é desenvolvido usando uma única pilha tecnológica, a arquitetura de micro-serviços permite (e muitas vezes encoraja) o uso de diferentes tecnologias para diferentes serviços ([58, 20, 10] e E3, E5, E7).

Este problema é significativo por várias razões. Primeiro, aumenta a complexidade do sistema. Cada tecnologia vem com sua própria curva de aprendizado, e é necessário que a equipe de desenvolvimento esteja familiarizada com todas as tecnologias usadas.

A diversidade de tecnologias pode levar a dificuldades de integração. Cada tecnologia pode ter suas próprias peculiaridades e maneiras de fazer as coisas, o que pode levar a problemas quando se tenta fazer diferentes tecnologias trabalharem juntas. Isso pode resultar em um aumento no tempo de desenvolvimento e na possibilidade de erros.

Além disso, a manutenção de um sistema com uma diversidade de tecnologias pode ser complexa. A correção de *bugs*, a implementação de novos recursos e a atualização do sistema podem ser mais complexas e demoradas do que em um sistema construído com uma única tecnologia.

Para resolver este problema, uma boa prática é limitar a diversidade de tecnologias usadas. Embora a arquitetura de micro-serviços permita o uso de diferentes tecnologias para diferentes serviços, e isso seja também uma vantagem desta arquitetura, isso não significa que seja necessário ou benéfico usar uma tecnologia diferente para cada serviço. Ao limitar o número de tecnologias usadas, é possível reduzir a complexidade do sistema e tornar o desenvolvimento e a manutenção mais gerenciáveis.

Além disso, é importante garantir que a equipe de desenvolvimento esteja adequadamente treinada em todas as tecnologias usadas. Esse fato pode levar a um maior investimento em treinamento e desenvolvimento profissional para a equipe. Logo, não apenas ajudará a reduzir a possibilidade de erros, mas também aumentará a eficiência do desenvolvimento.

Problema
"Necessidade de lidar com a diversidade de tecnologias"

A arquitetura de micro-serviços permite o uso de diferentes tecnologias para diferentes serviços, aumentando a complexidade do sistema. A diversidade tecnológica pode levar a dificuldades de integração, aumento no tempo de desenvolvimento e possibilidade de erros. Além disso, a manutenção de um sistema com várias tecnologias pode ser desafiadora, tornando a correção de bugs e a implementação de novos recursos mais complexas e demoradas.

Recomendação
Limitar o número de tecnologias usadas, reduzindo assim a complexidade do sistema e tornando o desenvolvimento e a manutenção mais gerenciáveis. É crucial garantir que a equipe de desenvolvimento esteja adequadamente treinada em todas as tecnologias usadas, possivelmente investindo em treinamento e desenvolvimento profissional, para reduzir a possibilidade de erros e aumentar a eficiência do desenvolvimento.

Categoria
Processos

4.1.12 Necessidade de um sistema de ponto de entrada

Nas arquiteturas de micro-serviços, as funcionalidades do sistema são divididas em serviços menores e independentes, cada um com suas próprias responsabilidades. Essa natureza descentralizada dos micro-serviços torna a gestão das solicitações de serviço um desafio complexo. Um sistema de ponto de entrada, também conhecido como API *Gateway*, é necessário para gerenciar como as solicitações dos clientes são roteadas para os micro-serviços apropriados ([29, 20]).

Este é um problema porque sem um ponto de entrada eficaz, a coordenação entre diferentes micro-serviços pode se tornar caótica e ineficiente. Os clientes podem ter que lidar com vários pontos de contato, cada um correspondendo a um micro-serviço diferente, o que pode resultar em uma má experiência de usuário. Além disso, sem um sistema de ponto de entrada, pode ser difícil implementar recursos como balanceamento de carga, tolerância a falhas e autenticação de usuários de maneira consistente em todos os micro-serviços.

Um API *Gateway* atua como um ponto de entrada único para todas as solicitações de clientes, roteando cada solicitação para o micro-serviço apropriado. Isso simplifica a interface do cliente, pois ele não precisa se preocupar com a localização ou o protocolo de cada micro-serviço.

Um API *Gateway* também pode fornecer funcionalidades adicionais que melhoram a robustez e a eficiência do sistema como um todo. Por exemplo, ele pode implementar um balanceamento de carga para distribuir a carga de trabalho entre várias instâncias de um micro-serviço, aumentando a capacidade de lidar com grandes volumes de tráfego. Ele também pode fornecer tolerância a falhas, redirecionando as solicitações para instâncias alternativas de um micro-serviço se uma falhar. Além disso, um API *Gateway* pode centralizar a autenticação e a autorização de usuários, garantindo que apenas os clientes autenticados possam acessar os micro-serviços.

Problema
"Necessidade de um sistema de ponto de entrada"

Em arquiteturas de micro-serviços, gerenciar solicitações de serviço é complexo devido à natureza descentralizada. Sem um ponto de entrada eficaz, a coordenação entre micro-serviços pode ser caótica, resultando em uma má experiência de usuário e dificuldade em implementar funcionalidades como balanceamento de carga e autenticação de usuários.

Recomendação
A implementação de um API *Gateway* ajuda a superar esses desafios. Ele age como um ponto de entrada único, roteando solicitações para o micro-serviço correto e simplificando a interface do cliente. Além disso, fornece funcionalidades extras como balanceamento de carga, tolerância a falhas e centralização da autenticação, melhorando a robustez e eficiência do sistema.

Categoria
Ferramentas

4.1.13 Falta de treinamento adequado para equipes

Enquanto a arquitetura de micro-serviços oferece vários benefícios, como escalabilidade e flexibilidade, ela também traz uma série de novos conceitos e práticas que as equipes de desenvolvimento precisam aprender e adotar [20].

A falta de treinamento adequado para as equipes é um problema porque a arquitetura de micro-serviços exige uma compreensão sólida de uma variedade de princípios e tecnologias. Isso inclui, mas não se limita a, conceitos como o design orientado a domínio, a programação baseada em eventos, a integração contínua e entrega contínua, e o gerenciamento de contêineres e orquestração. Sem o conhecimento adequado dessas áreas, as equipes de desenvolvimento podem encontrar dificuldades em projetar, implementar e manter serviços eficazes. Além disso, sem o treinamento adequado, as equipes podem

não ser capazes de aproveitar ao máximo os benefícios da arquitetura de micro-serviços e podem até introduzir novos riscos e complicações no sistema.

Para solucionar este problema, é essencial implementar um programa de treinamento abrangente e contínuo. Pode-se começar com uma formação básica sobre os princípios da arquitetura de micro-serviços, seguida de treinamentos mais avançados sobre as tecnologias e ferramentas específicas que serão usadas. Além disso, deve ser dada ênfase à prática, com as equipes tendo a oportunidade de aplicar o que aprenderam em projetos reais.

Também é útil promover uma cultura de aprendizado contínuo dentro da organização. Esse fato pode ser alcançado através de práticas como a realização regular de *workshops* ou sessões de compartilhamento de conhecimento, a criação de canais de comunicação onde os membros da equipe possam fazer perguntas e compartilhar perspectivas, e o incentivo à participação em conferências e seminários da indústria.

Finalmente, é importante lembrar que o treinamento adequado não é apenas sobre a aquisição de conhecimentos técnicos. As equipes também precisam ser treinadas em novas formas de trabalhar, como a colaboração e o desenvolvimento ágil, que são cruciais para o sucesso dos micro-serviços.

Problema
“Falta de treinamento adequado para equipes”

A falta de treinamento adequado em arquitetura de micro-serviços é um problema, pois requer uma sólida compreensão de diversos princípios e tecnologias. Sem este conhecimento, as equipes podem enfrentar dificuldades em desenvolver serviços eficazes e não aproveitar os benefícios dos micro-serviços, introduzindo riscos ao sistema.

Recomendação
Para resolver este problema, é crucial implementar um programa de treinamento abrangente e contínuo, promovendo uma cultura de aprendizado. Este programa deve incluir formação básica e avançada em micro-serviços, treinamento prático em projetos reais, *workshops* regulares e incentivo à participação em eventos da indústria. As equipes também devem ser treinadas em novas formas de trabalho, como colaboração e desenvolvimento ágil.

Categoria
Processos

4.1.14 Aumento do tempo de desenvolvimento inicial

O desenvolvimento inicial em uma arquitetura de micro-serviços pode ser mais demorado do que em uma arquitetura monolítica. Isso ocorre porque cada serviço precisa ser desenvolvido separadamente, com seu próprio conjunto de requisitos, e muitas vezes com suas próprias tecnologias e linguagens de programação. Além disso, os serviços precisam ser projetados para trabalhar juntos, o que requer um tempo considerável para design e coordenação. O desenvolvimento de infraestrutura para suportar a comunicação entre os serviços também pode ser complexo e demorado. Se somam a este tópico as automações necessárias para monitoramento, integração contínua e entrega contínua e gerenciamento da infraestrutura (E1, E2, E3, E4, E5, E7, E8, E9).

Esse aumento no tempo de desenvolvimento inicial é um problema porque pode aumentar os custos do projeto, prolongar o tempo até o lançamento do produto e diminuir a velocidade em que novos recursos podem ser entregues. Além disso, pode ser mais difícil justificar o investimento inicial em uma arquitetura de micro-serviços se os benefícios não forem imediatamente aparentes.

Apesar desses desafios, existem várias boas práticas que podem ajudar a mitigar o impacto do aumento do tempo de desenvolvimento inicial na arquitetura de micro-serviços. É importante planejar cuidadosamente o design e a coordenação dos serviços. É possível incluir a definição de padrões para comunicação entre serviços e a criação de um plano detalhado para o desenvolvimento e implantação de cada serviço.

Usar uma abordagem incremental para o desenvolvimento pode ser útil. Isso pode significar começar com uma arquitetura monolítica e, em seguida, separar gradualmente os serviços à medida que a aplicação cresce e as necessidades se tornam mais claras. Logo, este fato permite que a equipe ganhe experiência com a arquitetura de micro-serviços e minimize o risco de investir muito tempo e recursos no desenvolvimento de serviços que podem não ser necessários.

A utilização de ferramentas e plataformas que facilitam o desenvolvimento de micro-serviços também é benéfica. Isso pode incluir plataformas de desenvolvimento de *software* que suportam a arquitetura de micro-serviços, bem como ferramentas para gerenciamento de contêineres e orquestração de serviços.

Por fim, é importante ter uma equipe de desenvolvimento bem treinada e experiente, que entenda os desafios da arquitetura de micro-serviços e saiba como mitigá-los. Incluí-se a este fato o fornecimento de treinamento e educação para a equipe, bem como a contratação de desenvolvedores com experiência em micro-serviços.

Problema
"Aumento do tempo de desenvolvimento inicial"

A arquitetura de micro-serviços, embora poderosa, pode ter um desenvolvimento inicial mais demorado e complexo, aumentando os custos do projeto e prolongando o tempo de lançamento do produto. Isso ocorre pela necessidade de desenvolver cada serviço separadamente, de projetar a interação entre eles e de criar uma infraestrutura de comunicação adequada.

Recomendação
Adoção de um planejamento cuidadoso do design e coordenação dos serviços, uso de uma abordagem incremental para o desenvolvimento, utilização de ferramentas e plataformas que facilitem o desenvolvimento de micro-serviços e treinamento e educação adequados para a equipe de desenvolvimento, incluindo a contratação de profissionais com experiência em micro-serviços.

Categoria
Ferramentas, Padrões, Processos

4.2 A decisão de arquitetura

Conforme discutido nas seções anteriores, a transição para uma arquitetura de micro-serviços traz consigo uma série de desafios que precisam ser superados. Estes incluem a complexidade operacional, o custo inicial de adoção, os desafios de consistência de dados, o *overhead* de rede, a complexidade de *DevOps*, o monitoramento mais complexo, os desafios de comunicação, as dificuldades de depuração, as dificuldades de versionamento, os padrões de segurança desafiadores, a necessidade de lidar com a diversidade de tecnologias, a necessidade de um sistema de ponto de entrada, a falta de treinamento adequado para equipes e o aumento do tempo de desenvolvimento inicial.

Apesar desses desafios, a escolha da arquitetura deve ser fundamentada em uma análise cautelosa das necessidades e objetivos do projeto, assim como das habilidades e recursos da equipe de desenvolvimento. Diversas recomendações foram apresentadas para enfrentar esses desafios, abrangendo ferramentas, padrões e processos variados. A aplicação dessas recomendações, contudo, será determinada pelo contexto específico da organização e pelos requisitos do sistema de software.

A discussão apresentada revela que o sucesso na adoção de uma arquitetura de micro-serviços é resultado de um entendimento profundo desses desafios, e de medidas de mitigação eficazes. Além disso, é essencial considerar o ciclo de vida do projeto e a necessidade de manter, desenvolver e escalar o sistema ao longo do tempo. Assim, embora a transição para micro-serviços apresente obstáculos, a superação dessas barreiras pode trazer benefícios em termos de escalabilidade, flexibilidade e isolamento de falhas.

4.3 Ameaças à Validade

Quaisquer descobertas e percepções deste estudo são limitadas pela natureza das metodologias de pesquisa que foram aplicadas. Cada metodologia tem suas limitações associadas, mas algumas destas são mais geralmente aplicadas.

Primeiramente, este estudo poderia ter sido afetado ao viés de pesquisa, onde o pesquisador espera um resultado e assim interpreta as descobertas a partir dessa perspectiva. Este estudo colocou em prática técnicas que reduzem esse viés, como análise por mais de um autor e discussão dos resultados.

Dado que este estudo fez uso de entrevistas, gravações e transcrições, há a limitação em relação à transcrição das entrevistas e à interpretação das respostas pelo pesquisador. Mesmo que se tente ser o mais imparcial possível, o viés na análise dos resultados não pode ser totalmente eliminado.

É importante frisar que este estudo envolveu entrevistas com apenas nove especialistas na área. Embora esse número possa parecer modesto, a profundidade e a amplitude das percepções fornecidas por esses especialistas experientes enriquecem significativamente esta pesquisa. Sua vasta experiência e profundo entendimento das arquiteturas de *software* conferem às descobertas desta pesquisa uma profundidade que pode não ser alcançada através de grupos de participantes maiores e menos especializados. Esta abordagem destaca o princípio da qualidade sobre a quantidade, garantindo que as conclusões sejam tanto informadas quanto diversificadas.

Além disso, o campo em rápida evolução da arquitetura de *software* significa que os resultados podem ser limitados ao contexto ou perder relevância ao longo do tempo. Reconhecer estas limitações é crucial para uma compreensão equilibrada do escopo e aplicabilidade do estudo.

5. CONCLUSÃO

A pesquisa realizada nesta dissertação proporcionou uma análise abrangente sobre as arquiteturas de micro-serviços e monolitos, considerando suas vantagens e desvantagens. Os resultados obtidos através das metodologias de Revisão de Literatura Multivocal, encapsulando uma revisão sistemática por *snowballing* e *grey* literature, e a execução de uma *survey* com profissionais da indústria de *software* indicam que, embora a arquitetura de micro-serviços ofereça benefícios significativos em termos de escalabilidade, flexibilidade e isolamento de falhas, ela também apresenta desafios que necessitam ser considerados quando é feita a adoção por esta arquitetura.

Os desafios associados à arquitetura de micro-serviços, como a complexidade operacional, o custo inicial de adoção, os desafios de consistência de dados, o *overhead* de rede e o aumento da complexidade de *DevOps*, exigem uma gestão cuidadosa e podem demandar um investimento significativo de tempo e recursos. No entanto, para projetos grandes e complexos, os micro-serviços podem ser a melhor opção, oferecendo vantagens que superam as desvantagens.

Esta dissertação contribui para a literatura existente ao fornecer uma visão detalhada das implicações da escolha arquitetural. As descobertas reforçam a necessidade de uma avaliação criteriosa das necessidades do projeto e dos recursos disponíveis antes de decidir entre micro-serviços e monolitos.

5.1 Trabalhos Futuros

Para os trabalhos futuros, recomenda-se a continuação da investigação sobre as arquiteturas de micro-serviços e monolitos, com um foco particular na mitigação dos desafios identificados. Pesquisas adicionais poderiam explorar novas estratégias e ferramentas que facilitam a transição de monolitos para micro-serviços, bem como práticas que promovem a eficiência no desenvolvimento e manutenção de arquiteturas baseadas em micro-serviços.

Outra área de interesse é o desenvolvimento de *frameworks* ou plataformas que automatizem aspectos do desenvolvimento de micro-serviços, reduzindo a complexidade operacional e acelerando o ciclo de desenvolvimento. Além disso, estudos de caso detalhados em diferentes contextos de indústria poderiam fornecer dados práticos sobre como as empresas estão abordando os desafios associados a cada arquitetura e quais soluções estão sendo eficazes.

Finalmente, a pesquisa futura poderia se concentrar na educação e treinamento de desenvolvedores para trabalhar com micro-serviços, identificando as habilidades es-

senciais e criando materiais educacionais que preparem melhor os profissionais para os desafios desse paradigma arquitetural.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] “Microservices vs. monolithic architecture | Atlassian”. Accessed on 05.02.2024.

[2] Adams, R. D.; Smart, P.; Huff, A. S. “Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies”, *International Journal of Management Reviews*, vol. 19, 2016, pp. 432–454.

[3] Afanasev, M. Y.; Fedosov, Y. V.; Krylova, A. A.; Shorokhov, S. A. “An application of microservices architecture pattern to create a modular computer numerical control system”. In: 2017 20th Conference of Open Innovations Association (FRUCT), 2017, pp. 10–19.

[4] Akbulut, A.; Perros, H. G. “Performance analysis of microservice design patterns”, *IEEE Internet Computing*, vol. 23–6, 2019, pp. 19–27.

[5] Balalaie, A.; Heydarnoori, A.; Jamshidi, P. “Microservices architecture enables devops: Migration to a cloud-native architecture”, *IEEE Software*, vol. 33–3, 2016, pp. 42–52.

[6] Benavente, V.; Yantas, L.; Moscol, I.; Rodriguez, C.; Inquilla, R.; Pomachagua, Y. “Comparative analysis of microservices and monolithic architecture”. In: 2022 14th International Conference on Computational Intelligence and Communication Networks (CICN), 2022, pp. 177–184.

[7] Bernstein, D. “Containers and cloud: From lxc to docker to kubernetes”, *IEEE Cloud Computing*, vol. 1–3, 2014, pp. 81–84.

[8] Bian, Y.; Ma, D.; Zou, Q.; Yue, W. “A multi-way access portal website construction scheme”. In: 2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD), 2022, pp. 589–592.

[9] Blinowski, G.; Ojdowska, A.; Przybyłek, A. “Monolithic vs. microservice architecture: A performance and scalability evaluation”, *IEEE Access*, vol. 10, 2022, pp. 20357–20374.

[10] Bouttier, S. “Microservices Architecture From A to Z - The Startup - Medium”. Accessed on 05.02.2024, dec 16 2021.

[11] Braun, V.; Clarke, V. “Using thematic analysis in psychology”, *Qualitative Research in Psychology*, vol. 3–2, 2006, pp. 77–101, https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa.

[12] Butzin, B.; Golatowski, F.; Timmermann, D. “Microservices approach for the internet of things”. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 2016, pp. 1–6.

[13] Carter, B. “Growing software applications by incremental development of heterogeneous micro-applications using cellular regeneration concepts”. In: 2015 Annual Global Online Conference on Information and Computer Technology (GOCICT), 2015, pp. 1–5.

[14] Chen, R.; Li, S.; Li, Z. “From monolith to microservices: A dataflow-driven approach”. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017, pp. 466–475.

[15] Christian, J.; Steven; Kurniawan, A.; Anggreainy, M. S. “Analyzing microservices and monolithic systems: Key factors in architecture, development, and operations”. In: 2023 6th International Conference of Computer and Informatics Engineering (IC2IE), 2023, pp. 64–69.

[16] Corbin, J.; Strauss, A. “Qualitative research”, *Techniques and procedures for developing grounded theory*, vol. 3, 2008.

[17] de Oliveira Rosa, T.; Daniel, J. a. F. L.; Guerra, E. M.; Goldman, A. “A method for architectural trade-off analysis based on patterns: Evaluating microservices structural attributes”. In: 2020 European Conference on Pattern Languages of Programs 2020, 2020.

[18] Di Francesco, P.; Lago, P.; Malavolta, I. “Migrating towards microservice architectures: An industrial survey”. In: 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 29–2909.

[19] Fowler, M. “Microservice Trade-Offs”. Accessed on 05.02.2024.

[20] Freeman, J. “Microservices vs. Monoliths - John Freeman - Medium”. Accessed on 05.02.2024, nov 19 2018.

[21] Garousi, V.; Felderer, M.; Mäntylä, M. V. “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering”, *Information and Software Technology*, vol. 106, 2019, pp. 101–121.

[22] Gos, K.; Zabierowski, W. “The comparison of microservice and monolithic architecture”. In: 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2020, pp. 150–153.

[23] Gördesli, M.; Varol, A. “Comparing interservice communications of microservices for e-commerce industry”. In: 2022 10th International Symposium on Digital Forensics and Security (ISDFS), 2022, pp. 1–4.

[24] Hassan, S.; Bahsoon, R. “Microservices and their design trade-offs: A self-adaptive roadmap”. In: 2016 IEEE International Conference on Services Computing (SCC), 2016, pp. 813–818.

[25] Hove, S.; Anda, B. “Experiences from conducting semi-structured interviews in empirical software engineering research”. In: 11th IEEE International Software Metrics Symposium (METRICS’05), 2005, pp. 10 pp.–23.

[26] Jamshidi, P.; Pahl, C.; Mendonça, N. C.; Lewis, J.; Tilkov, S. “Microservices: The journey so far and challenges ahead”, *IEEE Software*, vol. 35–3, 2018, pp. 24–35.

[27] Kamei, F. K. “The use of grey literature review as evidence for practitioners”, *SIGSOFT Softw. Eng. Notes*, vol. 44–3, oct 2020, pp. 23.

[28] Karabey Aksakalli, I.; Çelik, T.; Can, A. B.; Tekinerdoğan, B. “Deployment and communication patterns in microservice architectures: A systematic literature review”, *Journal of Systems and Software*, vol. 180, 2021, pp. 111014.

[29] Karagözgil, M. “Understanding Microservice Design Patterns - Murat Karagözgil - Medium”. Accessed on 05.02.2024, oct 27 2023.

[30] Kholy, M. E.; Fatatry, A. E. “Framework for interaction between databases and microservice architecture”, *IT Professional*, vol. 21–5, 2019, pp. 57–63.

[31] Knoche, H. “Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices”. In: 2016 7th ACM/SPEC on International Conference on Performance Engineering, 2016, pp. 121–124.

[32] Kumar, P. K.; Agarwal, R.; Shivaprasad, R.; Sitaram, D.; Kalambur, S. “Performance characterization of communication protocols in microservice applications”. In: 2021 International Conference on Smart Applications, Communications and Networking (SmartNets), 2021, pp. 1–5.

[33] Kurpad, S.; BT, S.; Vijaykumar, S.; Jain, S.; Kalambur, S. “Microarchitectural analysis and characterization of performance overheads in service meshes with kubernetes”. In: 2023 3rd Asian Conference on Innovation in Technology (ASIANCON), 2023, pp. 1–6.

[34] Kyryk, M.; Tymchenko, O.; Pleskanka, N.; Pleskanka, M. “Methods and process of service migration from monolithic architecture to microservices”. In: 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2022, pp. 553–558.

[35] Lewis, J. “Microservices”. Accessed on 05.02.2024.

[36] Liu, G.; Huang, B.; Liang, Z.; Qin, M.; Zhou, H.; Li, Z. “Microservices: architecture, container, and challenges”. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020, pp. 629–635.

[37] Merson, P.; Yoder, J. “Modeling microservices with ddd”. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), 2020, pp. 7–8.

[38] Munaf, R. M.; Ahmed, J.; Khakwani, F.; Rana, T. “Microservices architecture: Challenges and proposed conceptual design”. In: 2019 International Conference on Communication Technologies (ComTech), 2019, pp. 82–87.

[39] Naizer, S. “Microservices vs Monolithic : which Architecture is the best”. Accessed on 05.02.2024, jun 22 2022.

[40] Ponce, F.; Márquez, G.; Astudillo, H. “Migrating from monolithic architecture to microservices: A rapid review”. In: 2019 38th International Conference of the Chilean Computer Science Society (SCCC), 2019, pp. 1–7.

[41] Raharjo, A. B.; Andyartha, P. K.; Wijaya, W. H.; Purwananto, Y.; Purwitasari, D.; Juniarta, N. “Reliability evaluation of microservices and monolithic architectures”. In: 2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM), 2022, pp. 1–7.

[42] Richardson, C. “Pattern: Monolithic architecture (2014)”. Accessed on 05.02.2024.

[43] Runeson, P.; Höst, M. “Guidelines for conducting and reporting case study research in software engineering”, *Empirical software engineering*, vol. 14, 2009, pp. 131–164.

[44] Salii, S.; Ajdari, J.; Zenuni, X. “Migrating to a microservice architecture: benefits and challenges”. In: 2023 46th MIPRO ICT and Electronics Convention (MIPRO), 2023, pp. 1670–1677.

[45] Singer, J.; Sim, S. E.; Lethbridge, T. C. “Software Engineering Data Collection for Field Studies”. London: Springer London, 2008, pp. 9–34.

[46] Soldani, J. “Grey literature: A safe bridge between academy and industry?”, *SIGSOFT Softw. Eng. Notes*, vol. 44–3, oct 2020, pp. 11–12.

[47] Soldani, J.; Tamburri, D. A.; Van Den Heuvel, W.-J. “The pains and gains of microservices: A systematic grey literature review”, *Journal of Systems and Software*, vol. 146, 2018, pp. 215–232.

[48] Stubbs, J.; Moreira, W.; Dooley, R. “Distributed systems of microservices using docker and serfnode”. In: 2015 7th International Workshop on Science Gateways, 2015, pp. 34–39.

[49] Taibi, D.; Lenarduzzi, V.; Pahl, C.; Janes, A. “Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages”. In: 2017 XP2017 Scientific Workshops, 2017.

[50] Taylor, G. R. “Integrating quantitative and qualitative methods in research”. University press of America, 2005.

[51] Velepucha, V.; Flores, P. “A survey on microservices architecture: Principles, patterns and migration challenges”, *IEEE Access*, vol. 11, 2023, pp. 88339–88358.

[52] Vidiečcan, M.; Bobák, M. “Container-based video streaming service”. In: 2022 IEEE 22nd International Symposium on Computational Intelligence and Informatics and 8th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics (CINTI-MACRo), 2022, pp. 000191–000196.

[53] Viennot, N.; Lécuyer, M.; Bell, J.; Geambasu, R.; Nieh, J. “Synapse: A microservices architecture for heterogeneous-database web applications”. In: 2015 Tenth European Conference on Computer Systems, 2015.

[54] Wei, Y.; Yu, Y.; Pan, M.; Zhang, T. “A feature table approach to decomposing monolithic applications into microservices”. In: 2021 12th Asia-Pacific Symposium on Internetware, 2021, pp. 21–30.

[55] Wohlin, C. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: 2014 18th International Conference on Evaluation and Assessment in Software Engineering, 2014.

[56] Wu, M.; Zhang, Y.; Liu, J.; Wang, S.; Zhang, Z.; Xia§, X.; Mao, X. “On the way to microservices: Exploring problems and solutions from online qa community”. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, pp. 432–443.

[57] Yarygina, T.; Bagge, A. H. “Overcoming security challenges in microservice architectures”. In: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), 2018, pp. 11–20.

[58] Yevtushenko, S. “Sober Look at Microservices - Sergiy Yevtushenko - Medium”. Accessed on 05.02.2024, mar 14 2022.

APÊNDICE A – ENTREVISTA PARA COLETA DE DADOS JUNTO A ESPECIALISTAS

Título	**Roteiro semi-estruturado - Questionário**
Formato	• Roteiro semi-estruturado para entrevista presencial ou via video-conferência. • Este questionário está dividido em 3 etapas: – Etapa 1: Dados demográficos; – Etapa 2: Questionário específico sobre experiências; – Etapa 3: Observações gerais • Com duração prevista entre 30 e 45 minutos, incluindo a leitura do TCLE e introdução do estudo realizado.
Objetivo	• O objetivo da realização das entrevista é a obtenção de um melhor entendimento sobre a percepção dos praticantes do desenvolvimento de software sobre a utilização das arquiteturas de software de micro serviços e de monolitos. Com este entendimento, obter o embasamento necessário para a geração de um guia de recomendações sobre a utilização destas arquiteturas, bem como quando um desenvolvedor deve optar pela utilização de uma ou de outra, com base nas características do software a ser desenvolvido.
Participantes	• Desenvolvedores de software com experiência relevante nas arquiteturas mencionadas no estudo

Questões	• Dados demográficos: 1. Qual é o seu cargo ou função dentro da empresa? 2. Há quantos anos você trabalha na área de desenvolvimento de software? 3. Em qual setor ou indústria sua empresa opera?

- Experiências específicas:

    1. Você já trabalhou em projetos que usam a arquitetura de micro-serviços? Se sim, quantos?
    2. Você já trabalhou em projetos que usam a arquitetura monolítica? Se sim, quantos?
    3. Na sua opinião, quais são as principais diferenças entre micro-serviços e monolitos?
    4. Na sua experiência, qual arquitetura facilita o desenvolvimento de novas funcionalidades?
    5. Qual arquitetura requer menos esforço para manutenção do sistema a longo prazo?
    6. Qual arquitetura se mostrou mais escalável para lidar com um aumento significativo na demanda de usuários ou dados?
    7. Em termos de desempenho, você observou diferenças significativas entre as arquiteturas nos projetos em que trabalhou? Quais diferenças?
    8. Na sua opinião, qual arquitetura torna o código fonte e a lógica de negócio mais fáceis de entender e manter?
    9. Qual arquitetura facilita o teste automatizado e a implantação contínua?
    10. Você notou diferenças na estabilidade e confiabilidade do sistema ao implantar com cada uma das arquiteturas? Quais diferenças?
    11. Na sua opinião, qual arquitetura é mais adequada para projetos pequenos e de baixa complexidade?
    12. E qual arquitetura é mais adequada para projetos grandes e de alta complexidade?
    13. Com base na sua experiência geral, você tende a recomendar o uso de micro-serviços ou monolitos para a maioria dos projetos?
    14. Você acredita que a escolha da arquitetura deve ser influenciada pelo domínio de negócio no qual o software será aplicado? Por quê?
    15. Com base na sua experiência, você acredita que o desenvolvimento de software usando a arquitetura de micro-serviços geralmente resulta em um aumento geral no tempo de desenvolvimento em comparação com a arquitetura monolítica? Se sim, até que ponto você percebe essa diferença no tempo de desenvolvimento?

	• Considerações Finais: 1. Existem outras vantagens ou desvantagens que você considera importante mencionar sobre o uso de micro-serviços ou monolitos? Quais são eles? 2. Existem cenários específicos em que uma arquitetura é claramente superior à outra? Quais são eles? 3. Se você tiver, compartilhe exemplos específicos de projetos em que você trabalhou onde uma arquitetura se destacou em comparação com a outra.

APÊNDICE B – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO (TCLE)

Nós, Daniel dos Santos Krug (aluno de mestrado) e Afonso Henrique Correa de Sales (professor orientador), responsáveis pela pesquisa “Entendendo vantagens e desvantagens em aplicações monolíticas versus o uso de micro serviços”, estamos fazendo um convite para você participar como voluntário neste estudo.

Esta pesquisa pretende gerar um guia de recomendações sobre a utilização das arquiteturas de micro serviços e monolíticas. Acreditamos que ela seja importante porque foram identificados problemas com o uso da arquitetura de micro serviços. Para sua realização será feito o seguinte: Será realizada uma entrevista entre o entrevistado e entrevistador preferencialmente de forma remota (online). O tempo estimado para a realização desta entrevista é de no máximo 45 minutos. Sua participação consistirá em responder ao questionário proposto. É possível que aconteçam os seguintes desconfortos ou riscos: Você pode se sentir constrangido com o teor das perguntas propostas. Há também o risco da quebra de sigilo. Você tem o direito de pedir uma indenização por qualquer dano que, comprovadamente, resulte da sua participação no estudo. O benefício que esperamos com o estudo é a obtenção de um melhor entendimento sobre a percepção dos praticantes do desenvolvimento de software sobre a utilização das arquiteturas de software de micro serviços e de monolitos. Com este entendimento, espera-se obter o embasamento necessário para a geração de um guia de recomendações sobre a utilização destas arquiteturas, bem como quando um desenvolvedor deve optar pela utilização de uma ou de outra, com base nas características do software a ser desenvolvido.

Durante todo o período da pesquisa você tem o direito de esclarecer qualquer dúvida ou pedir qualquer informação sobre o estudo, bastando para isso entrar em contato, com Daniel dos Santos Krug no telefone Removido por privacidade ou Afonso Sales no telefone Removido por privacidade a qualquer hora. Em caso de algum problema relacionado com a pesquisa você terá direito à assistência gratuita que será prestada pelo pesquisador. Você tem garantido o seu direito de não aceitar participar ou de retirar sua permissão, a qualquer momento, sem nenhum tipo de prejuízo ou retaliação, pela sua decisão. Se, por algum motivo, você tiver despesas decorrentes da sua participação neste estudo com transporte e/ou alimentação, você será reembolsado adequadamente pelos pesquisadores.

As informações desta pesquisa serão confidenciais, e serão divulgadas apenas em eventos ou publicações científicas, não havendo identificação dos participantes, a não ser entre os responsáveis pelo estudo, sendo assegurado o sigilo sobre sua participação. Caso você tenha qualquer dúvida quanto aos seus direitos como participante de pesquisa, entre em contato com o Comitê de Ética em Pesquisa da Pontifícia Universidade

Católica do Rio Grande do Sul (CEP-PUCRS) em (51) 33203345, Av. Ipiranga, 6681/prédio 50 sala 703, CEP: 90619-900, Bairro Partenon, Porto Alegre – RS, e-mail: cep@pucrs.br, de segunda a sexta-feira das 8h às 12h e das 13h30 às 17h. O Comitê de Ética é um órgão independente constituído de profissionais das diferentes áreas do conhecimento e membros da comunidade. Sua responsabilidade é garantir a proteção dos direitos, a segurança e o bem-estar dos participantes por meio da revisão e da aprovação do estudo, entre outras ações. Ao assinar este termo de consentimento, você não abre mão de nenhum direito legal que teria de outra forma. Não assine este termo de consentimento a menos que tenha tido a oportunidade de fazer perguntas e tenha recebido respostas satisfatórias para todas as suas dúvidas. Se você concordar em participar deste estudo, você rubricará todas as páginas e assinará e datará duas vias originais deste termo de consentimento. Ao assinar e rubricar todas as páginas deste documento, você, de forma voluntária e esclarecida, nos autoriza a utilizar todas as informações de natureza pessoal que constam em seu formulário, imagens e voz, para finalidade de pesquisa e realização deste estudo. Você receberá uma das vias para seus registros e a outra será arquivada pelo responsável pelo estudo.

Eu, _____, após a leitura deste documento e de ter tido a oportunidade de conversar com o pesquisador responsável, para esclarecer todas as minhas dúvidas, acredito estar suficientemente informado, ficando claro para mim que minha participação é voluntária e que posso retirar este consentimento, a qualquer momento, sem penalidades ou perda de qualquer benefício. Estou ciente também dos objetivos da pesquisa, dos procedimentos aos quais serei submetido, dos possíveis danos ou riscos deles provenientes e da garantia de confidencialidade e esclarecimentos sempre que desejar.

Diante do exposto expresso minha concordância de espontânea vontade em participar deste estudo, autorizando o uso, compartilhamento e publicação dos meus dados e informações de natureza pessoal para essa finalidade específica.

_____

Assinatura do participante da pesquisa

_____

Assinatura de uma testemunha

DECLARAÇÃO DO PROFISSIONAL QUE OBTEVE O CONSENTIMENTO

Expliquei integralmente este estudo ao participante (e/ou ao seu tutor). Na minha opinião e na opinião do participante (e/ou do tutor), houve acesso suficiente às informações, incluindo riscos e benefícios, para que uma decisão consciente seja tomada.

Data: _____

Daniel dos Santos Krug	Afonso Henrique Correa de Sales
Aluno de Mestrado em Ciência da Computação	Professor do PPGCC – Escola Politécnica

APÊNDICE C – PARECER DE APROVAÇÃO DO PROJETO GERADO PELO COMITÊ DE ÉTICA E PESQUISA

PARECER CONSUBSTANCIADO DO CEP

DADOS DO PROJETO DE PESQUISA

Título da Pesquisa: Entendendo vantagens e desvantagens em aplicações monolíticas versus o uso de micro-serviços

Pesquisador: AFONSO HENRIQUE CORREA DE SALES

Área Temática:

Versão: 2

CAAE: 74791523.5.0000.5336

Instituição Proponente: Pontifícia Universidade Católica do Rio Grande do Sul

Patrocinador Principal: UNIAO BRASILEIRA DE EDUCACAO E ASSISTENCIA

DADOS DO PARECER

Número do Parecer: 6.482.063

Apresentação do Projeto:

As informações elencadas nos campos "Apresentação do Projeto", "Objetivo da Pesquisa" e "Avaliação dos Riscos e Benefícios" foram retiradas do arquivo Informações Básicas da Pesquisa (PB_INFORMAÇÕES_BÁSICAS_DO_PROJETO_2217782.pdf, de 25/10/2023).

Introdução

A Arquitetura de Micro-serviços é uma abordagem de desenvolvimento de software que consiste em dividir aplicações grandes - ou Monolitos - em serviços menores e independentes, capazes de se comunicar entre si. Cada serviço pode ser escrito em uma linguagem diferente e se comunica através de APIs leves. Esses serviços podem gerenciar suas próprias bases de dados e possuir modelos de dados específicos. A utilização de contêineres, como Docker e Kubernetes, é comum na implantação de micro-serviços, pois permite a execução isolada e facilita o seu deploy em diferentes ambientes. Estudos acadêmicos relacionados aos micro-serviços surgiram nadécada de 2010, enquanto as diferenças entre micro serviços e monolitos foram discutidas a partir de 2016. A adoção da arquitetura de micro-serviços resolve problemas relacionados aos monolitos, como complexidade de código, escalabilidade limitada e dificuldades no deploy. Algumas das principais vantagens dos micro-serviços incluem entregas mais rápidas, escalabilidade e autonomia. No entanto, também existem desvantagens, como segurança, desempenho de rede e consistência de dados, que podem ser mitigadas durante o desenvolvimento. Esta pesquisa propõe caracterizar a

utilização da arquitetura de micro serviços em comparação a de monolitos no campo da Engenharia de Software, procurando identificar, analisar e interpretar.
Hipótese: A adoção da arquitetura de micro-serviços pode aumentar o tempo de desenvolvimento de software devido a desafios como complexidade inicial, gerenciamento de comunicação, testes e implantação mais complexos, evolução constante, gerenciamento de dados e monitoramento mais detalhado. No entanto, esses desafios podem ser mitigados com uma estratégia de desenvolvimento cuidadosa.
Metodologia Proposta: Para obtenção do embasamento necessário, as metodologias selecionadas foram a análise da literatura com o uso do Snowbolling, uma análise baseada na revisão sistemática da Grey Literature e por fim, a realização de uma Survey, tendo como participantes desenvolvedores da área de software.

Objetivo da Pesquisa:
Objetivo Primário:
Entender a questão do tempo do aumento do tempo de desenvolvimento de software relacionado ao uso da arquitetura de micro-serviços em
comparação ao uso da arquitetura de monolitos.
Objetivo Secundário:
Gerar um guia de recomendações a ser utilizado por desenvolvedores praticantes da arquitetura de micro-serviços e monolitos sobre qual
arquitetura é a mais adequada para o software a ser desenvolvido baseado em suas características.

Avaliação dos Riscos e Benefícios:
Os riscos associados à pesquisa são mínimos, como o constrangimento ao responder alguma das questões propostas. Existe minimamente também o risco de quebra de sigilo dos respondentes.

Comentários e Considerações sobre a Pesquisa:
Sem comentários adicionais.

Considerações sobre os Termos de apresentação obrigatória:
Todos os termos foram apresentados.

Conclusões ou Pendências e Lista de Inadequações:
Não há pendências.

Considerações Finais a critério do CEP:
Diante do exposto, o CEP-PUCRS, de acordo com suas atribuições definidas na Resolução CNS n°

466 de 2012, Resolução 510 de 2016 e da Norma Operacional n° 001 de 2013 do CNS, manifesta-se pela aprovação do projeto de pesquisa "Entendendo vantagens e desvantagens em aplicações monolíticas versus o uso de micro-serviços" proposto pelo pesquisador AFONSO HENRIQUE CORREA DE SALES com número de CAAE 74791523.5.0000.5336.

Este parecer foi elaborado baseado nos documentos abaixo relacionados:

Tipo Documento	Arquivo	Postagem	Autor	Situação
Informações Básicas do Projeto	PB_INFORMAÇÕES_BÁSICAS_DO_PROJETO_2217782.pdf	25/10/2023 11:54:34		Aceito
Outros	Encaminhamento_ao_CEP_pdf.doc	24/10/2023 21:58:16	Daniel Krug	Aceito
Orçamento	Orcamento_Assinado_png.doc	24/10/2023 21:52:00	Daniel Krug	Aceito
TCLE / Termos de Assentimento / Justificativa de Ausência	TCLE_termo_concentimento_livre_esclarecido_revisado_LGPD_2023_v1.doc	23/10/2023 21:58:54	Daniel Krug	Aceito
Solicitação Assinada pelo Pesquisador Responsável	Encaminhamento_ao_CEP.pdf	04/10/2023 13:49:38	Daniel Krug	Aceito
Declaração do Patrocinador	Orcamento_Assinado.png	01/10/2023 15:22:10	Daniel Krug	Aceito
Outros	Curriculo_lattes.docx	01/10/2023 15:05:08	Daniel Krug	Aceito
Outros	Documento_Unificado_Projeto_Pesquisa_1695833048592.pdf	01/10/2023 15:01:17	Daniel Krug	Aceito
Solicitação registrada pelo CEP	Carta_Aprovacao_Comissao_Cientifica_1695833048592.pdf	01/10/2023 14:59:38	Daniel Krug	Aceito
Declaração de concordância	Carta_de_Anuencia_Escola_Politecnica_Daniel_Krug.pdf	01/10/2023 14:58:14	Daniel Krug	Aceito
Projeto Detalhado / Brochura Investigador	Projeto_Pesquisa_CEP_Daniel_dos_Santos_Krug.docx	21/09/2023 21:12:10	Daniel Krug	Aceito
Folha de Rosto	folhaDeRosto.pdf	21/09/2023 21:10:42	Daniel Krug	Aceito

Situação do Parecer:
Aprovado

Necessita Apreciação da CONEP:

Não

PORTO ALEGRE, 02 de Novembro de 2023

Assinado por:
Karen Cherubini
(Coordenador(a))

Endereço: Av. Ipiranga, n° 6681, Prédio 50, sala 703
Bairro: Partenon **CEP:** 90.619-900
UF: RS **Município:** PORTO ALEGRE
Telefone: (51)3320-3345 **Fax:** (51)3320-3345 **E-mail:** cep@pucrs.br

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Pesquisa e Pós-Graduação
Av. Ipiranga, 6681 – Prédio 1 – Térreo
Porto Alegre – RS – Brasil
Fone: (51) 3320-3513
E-mail: propesq@pucrs.br
Site: www.pucrs.br