UNIVERSIDADE DO VALE DO TAQUARI
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ARQUITETURA DE MICROSSERVIÇOS: DESAFIOS E OPORTUNIDADES NA CONSTRUÇÃO DE SISTEMAS ESCALÁVEIS E RESILIENTES

LUCAS MICHEL DA SILVA

Lajeado, novembro de 2023

LUCAS MICHEL DA SILVA

ARQUITETURA DE MICROSSERVIÇOS: DESAFIOS E OPORTUNIDADES NA CONSTRUÇÃO DE SISTEMAS ESCALÁVEIS E RESILIENTES

Monografia apresentada na disciplina de Trabalho de Conclusão de Curso II, do curso de Engenharia de Computação, da Universidade do Vale do Taquari - Univates, como parte da exigência para a obtenção do título de bacharel em Engenharia de Computação.

Orientador: Prof. Me. Edson Moacir Ahlert

Lajeado, novembro de 2023

LUCAS MICHEL DA SILVA

ARQUITETURA DE MICROSSERVIÇOS: DESAFIOS E OPORTUNIDADES NA CONSTRUÇÃO DE SISTEMAS ESCALÁVEIS E RESILIENTES

A Banca examinadora abaixo aprova a Monografia apresentada no componente curricular Trabalho de Conclusão de Curso II, do Curso de Engenharia de Computação, da Universidade do Vale do Taquari - Univates, como parte da exigência para a obtenção do título de Bacharel em Engenharia de Computação:

Prof. Me. Luis Antônio Schneiders
Universidade do Vale do Taquari - Univates

Prof. Dr. Alexandre Stürmer Wolf
Universidade do Vale do Taquari - Univates

Lajeado, novembro de 2023

RESUMO

Este estudo explora a relevância e a dinâmica da arquitetura de microsserviços como fundamento para a criação de sistemas informáticos que sejam ao mesmo tempo escaláveis e dotados de alta disponibilidade. A pesquisa avança por meio de um exame meticuloso dos desafios e vantagens que caracterizam a implementação e a gestão dessa arquitetura. Combinando métodos qualitativos e quantitativos, e valendo-se de análises de literatura e casos práticos relevantes, o trabalho propõe um compêndio de diretrizes práticas e fundamentação teórica visando facilitar a transição para ou a adoção de microsserviços. Propõe-se a criação de uma arquitetura de microsserviços voltada para a integração com o ChatGPT, abrangendo os conceitos e tarefas necessários para atingir os objetivos estabelecidos. As conclusões deste estudo destacam a eficácia da arquitetura de microsserviços em termos de escalabilidade e resiliência. Fazendo utilização da abordagem científica, foram explorados os desafios práticos e teóricos, fornecendo percepções tanto para o meio acadêmico quanto profissional. Os desafios enfrentados durante a pesquisa revelaram oportunidades para investigações e desenvolvimentos futuros na área de microsserviços.

Palavras-chave: arquitetura de microsserviços, sistemas escaláveis, resiliência, sistemas distribuídos.

ABSTRACT

This study explores the relevance and dynamics of microservices architecture as a foundation for creating computer systems that are both scalable and highly available. The research progresses through a meticulous examination of the challenges and advantages that characterize the implementation and management of this architecture. Combining qualitative and quantitative methods, and utilizing analyses of literature and relevant practical cases, the work proposes a compendium of practical guidelines and theoretical foundation aimed at facilitating the transition to or adoption of microservices. The creation of a microservices architecture focused on integration with ChatGPT is proposed, covering the necessary concepts and tasks to achieve the established objectives. The conclusions of this study highlight the effectiveness of microservices architecture in terms of scalability and resilience. Using a scientific approach, practical and theoretical challenges were explored, providing insights for both the academic and professional community. The challenges encountered during the research revealed opportunities for future investigations and developments in the field of microservices.

Keywords: microservices architecture, scalable systems, resilience, distributed systems.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interfaces* – Interfaces de Programação de Aplicativos
BFF	*Backend For Frontend* – Backend para Frontend
MSA	*Microservice architecture* – Arquitetura de microsserviços
UML	*Unified Modeling Language* Linguagem de modelagem unificada

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o estilo de arquitetura de microsserviços (Microservices Architecture - MSA) ganhou popularidade significativa como uma arquitetura de desenvolvimento de software. Isso, juntamente com o desenvolvimento de software automatizado e os processos operacionais, capacita as organizações a publicar novos recursos e aprimoramentos de maneira rápida e eficiente em seus aplicativos. Como resultado, a MSA tornou-se a escolha preferida para otimizar a implantação de softwares modulares.

Segundo Lewis e Fowler (2014), a arquitetura de microsserviços é uma técnica de arquitetura de software em que um aplicativo é desmembrado em componentes menores, independentes e que se comunicam entre si por meio de APIs. Cada componente, ou serviço, é responsável por uma funcionalidade específica, permitindo que o aplicativo como um todo seja escalável e resiliente.

Para Newman (2015), essa abordagem concentra-se na construção de sistemas a partir de serviços independentes e autônomos que podem ser implantados, escalados e atualizados de forma independente. Essa abordagem oferece várias vantagens em relação às arquiteturas monolíticas tradicionais, incluindo maior flexibilidade, escalabilidade e resiliência.

Além disso, a arquitetura de microsserviços permite que as equipes de desenvolvimento possam trabalhar em serviços independentes, facilitando a inovação e a implementação rápida de novos recursos.

No entanto, a implementação de uma arquitetura de microsserviços pode ser desafiadora e apresentar problemas, como a complexidade da comunicação entre serviços e a necessidade de garantir a segurança e a privacidade dos dados em um ambiente distribuído. De acordo com Dragoni *et al.* (2017), muitas empresas e desenvolvedores ainda têm dúvidas sobre como implementar uma arquitetura de microsserviços de forma eficiente e escalável.

Apesar das vantagens oferecidas pela arquitetura de microsserviços, ainda há muitas questões a serem abordadas na sua implementação. Entre elas, é necessário pensar em como garantir a consistência dos dados em um ambiente

distribuído, além de lidar com a complexidade da comunicação entre os serviços. Nesse sentido, um ponto de atenção relevante seria como garantir a integridade e a disponibilidade dos serviços em uma arquitetura de microsserviços, sem comprometer a segurança e a privacidade dos dados dos usuários.

1.1 Problema de pesquisa

O presente trabalho pretende fornecer uma visão geral sobre a arquitetura de microsserviços, seus benefícios e desafios, e fornecer orientações práticas para a implementação de uma MSA bem-sucedida.

A partir de um estudo do estado da arte, serão apresentadas as principais características da arquitetura de microsserviços e seus principais benefícios em relação às abordagens tradicionais. Também serão discutidos os principais desafios da implementação de uma arquitetura de microsserviços, incluindo a comunicação entre serviços, a gestão de dados distribuídos e a necessidade de garantir a segurança e privacidade dos dados.

Por fim, serão apresentadas algumas orientações práticas para a implementação de uma arquitetura de microsserviços bem-sucedida, incluindo as melhores práticas de design, a escolha das ferramentas e tecnologias mais adequadas e os principais desafios de operação e monitoramento em um ambiente de microsserviços.

O problema de pesquisa abordado nesta monografia surge a partir da seguinte questão: como a arquitetura de microsserviços pode contribuir na construção de sistemas escaláveis e resilientes?

Este trabalho tem como foco a implementação bem-sucedida de uma arquitetura de microsserviços, analisando suas principais características e benefícios, além dos desafios envolvidos em sua implementação e operação.

Com o uso crescente de arquiteturas de microsserviços em empresas e organizações, torna-se fundamental compreender as melhores práticas e estratégias para garantir eficiência e segurança na adoção destes sistemas.

Assim, este estudo se dedica a oferecer contribuições teóricas e práticas para a implementação eficaz de uma arquitetura de microsserviços, visando enriquecer tanto a comunidade acadêmica quanto a profissional com orientações neste campo.

1.2 Objetivos

Os próximos tópicos apresentam os objetivos a serem alcançados através deste projeto.

1.2.1 Objetivo geral

O principal objetivo deste trabalho é oferecer diretrizes práticas e recomendações para uma implementação bem-sucedida e robusta de uma arquitetura baseada em microsserviços.

1.2.2 Objetivos específicos

- Investigar as características fundamentais da arquitetura de microsserviços, abordando suas vantagens e desvantagens em comparação com as metodologias convencionais de desenvolvimento de software.
- Examinar os desafios enfrentados na implementação de uma arquitetura de microsserviços, considerando aspectos como comunicação entre serviços, gerenciamento de dados distribuídos e a garantia de segurança e privacidade dos dados.
- Identificar as melhores práticas no projeto de microsserviços, contemplando a seleção de tecnologias e ferramentas apropriadas, a administração de dados e a garantia de qualidade do sistema.
- Proporcionar diretrizes práticas e eficazes para a implementação bem-sucedida de uma arquitetura de microsserviços, abrangendo fases como planejamento, desenvolvimento, testes e operação.
- Arquitetar e desenvolver uma arquitetura de microserviços com foco na integração com o ChatGPT para avaliar os resultados obtidos ao implantar

em um contexto real, analisando os efeitos na qualidade, escalabilidade, desempenho e segurança do sistema desenvolvido.

1.3 Estrutura do trabalho

O estudo em questão está estruturado em seis partes principais, com foco em microsserviços. No primeiro capítulo, são apresentados os objetivos da pesquisa, o público-alvo e uma introdução ao tema dos microsserviços.

O segundo capítulo aborda o referencial teórico, examinando o que foi estudado para a aplicação da arquitetura de microsserviços e as tecnologias utilizadas.

No terceiro capítulo, são comparados trabalhos que utilizaram a abordagem de desenvolvimento baseada em microsserviços e que possuem recursos e funcionalidades similares. Esta análise permite destacar as diferenças e semelhanças entre os diferentes estudos, mostrando o que cada um deles abrange ou não em termos de metodologia e abordagem.

O capítulo quatro fornece uma descrição detalhada das metodologias adotadas neste estudo, incluindo os métodos científicos utilizados, a abordagem de pesquisa escolhida, os objetivos do estudo e os procedimentos técnicos empregados na pesquisa em análise.

O capítulo cinco detalha os passos seguidos para o desenvolvimento da integração com o ChatGPT. Ao final, são apresentados os desafios encontrados durante o processo, bem como os testes realizados e os resultados obtidos a partir deles.

O capítulo seis oferece as conclusões finais. Em seguida, são apresentadas as fontes bibliográficas que serviram de base para a fundamentação teórica deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os tópicos relevantes ao tema em estudo, por meio de uma revisão bibliográfica fundamentada na literatura especializada, em pesquisas e debates de outros autores acerca da temática abordada neste trabalho. Dessa forma, busca-se construir um embasamento teórico sólido para o desenvolvimento da pesquisa em questão.

2.1 Conceitos e características de arquitetura de microsserviços

A arquitetura de microsserviços é um estilo de projeto de software que enfatiza a decomposição de aplicativos em serviços menores e independentes, cada um com responsabilidades específicas e bem definidas (LEWIS; FOWLER, 2014). Essa abordagem contrasta com as arquiteturas monolíticas tradicionais, nas quais todos os componentes de uma aplicação estão interconectados em uma única unidade. A ideia central dos microsserviços é permitir a construção de sistemas mais flexíveis e escaláveis, que possam ser facilmente adaptados às mudanças nas necessidades do negócio (NEWMAN, 2015).

Os princípios fundamentais da arquitetura de microsserviços incluem a modularidade, o desacoplamento e a comunicação entre serviços (NEWMAN, 2015). A modularidade permite que cada serviço seja desenvolvido, implantado e mantido de forma independente, o que facilita a escalabilidade e a manutenção. O desacoplamento é uma característica importante para garantir a independência entre os serviços, permitindo que as mudanças em um serviço não afetem os outros serviços. A comunicação entre os serviços é geralmente realizada por meio de protocolos leves e padronizados, como HTTP e REST (RICHARDSON, 2018).

A arquitetura de microsserviços apresenta vários benefícios, como maior escalabilidade, flexibilidade e facilidade de manutenção, além de implantação independente de serviços (RICHARDSON, 2018). No entanto, também existem desafios associados a essa abordagem, como a complexidade na coordenação e monitoramento dos serviços e a necessidade de lidar com questões de consistência e tolerância a falhas. Algumas práticas recomendadas e padrões em arquitetura de

microsserviços incluem o uso de API Gateway, Circuit Breaker e Service Registry e Discovery para lidar com esses desafios (RICHARDSON, 2018).

Em conclusão, a arquitetura de microsserviços representa uma abordagem moderna e promissora para o desenvolvimento de sistemas de software, oferecendo diversos benefícios em relação às arquiteturas monolíticas tradicionais. Esses benefícios incluem maior escalabilidade, flexibilidade e facilidade de manutenção, bem como implantação independente de serviços.

No entanto, a adoção dessa abordagem também traz desafios, como a complexidade na coordenação e monitoramento dos serviços e a necessidade de lidar com questões de consistência e tolerância a falhas. É importante considerar as práticas recomendadas e os padrões em arquitetura de microsserviços para enfrentar esses desafios e garantir o sucesso na implantação de sistemas baseados nessa abordagem.

2.1.1 Conceitos básicos de arquitetura de microsserviços: definição, objetivos e principais características.

A arquitetura de microsserviços tem sido amplamente adotada nos últimos anos, proporcionando uma abordagem alternativa à arquitetura monolítica tradicional (NEWMAN, 2015). Microsserviços são definidos como pequenos serviços independentes e coesos que, juntos, formam um sistema maior e mais complexo. Eles são projetados com um foco no domínio e são autônomos, o que significa que cada serviço é responsável por uma funcionalidade específica e pode ser desenvolvido, implantado e dimensionado independentemente dos outros (NEWMAN, 2015; LEWIS; FOWLER, 2014).

Os principais objetivos da arquitetura de microsserviços incluem escalabilidade, resiliência, desacoplamento e facilidade de manutenção (NEWMAN, 2015). A escalabilidade é alcançada através da capacidade de dimensionar serviços individuais de acordo com a demanda, sem afetar os outros serviços no sistema. A resiliência é obtida através do isolamento e da tolerância a falhas entre os serviços, permitindo que o sistema continue funcionando mesmo quando alguns dos serviços

enfrentam problemas (DRAGONI *et al*., 2017). O desacoplamento é alcançado através da separação das responsabilidades entre os serviços, promovendo a modularidade e a flexibilidade. Além disso, a arquitetura de microsserviços facilita a manutenção, permitindo que as equipes de desenvolvimento trabalhem em serviços individuais sem a necessidade de compreender todo o sistema (RICHARDSON, 2018).

Algumas práticas e padrões recomendados para a arquitetura de microsserviços incluem o Domain-Driven Design (DDD), que auxilia na modelagem de serviços com base no domínio do problema (EVANS, 2016), e o uso de Circuit Breaker e API Gateway para melhorar a resiliência e gerenciamento de comunicação entre os serviços (RICHARDSON, 2018). A adoção desses padrões e práticas pode ajudar a garantir que os sistemas de microsserviços sejam eficientes, escaláveis e resilientes.

Em resumo, a arquitetura de microsserviços oferece uma abordagem inovadora e eficiente para o desenvolvimento de sistemas de software, proporcionando escalabilidade, resiliência, desacoplamento e facilidade de manutenção. Ao adotar os princípios fundamentais da arquitetura de microsserviços e aplicar práticas e padrões recomendados, como o Domain-Driven Design, Circuit Breaker e API Gateway, os desenvolvedores e arquitetos de software podem criar sistemas complexos e robustos que atendem às necessidades em constante evolução dos negócios.

A compreensão desses conceitos e características é essencial para aproveitar ao máximo as vantagens oferecidas pela arquitetura de microsserviços e garantir a eficiência, escalabilidade e resiliência dos sistemas desenvolvidos.

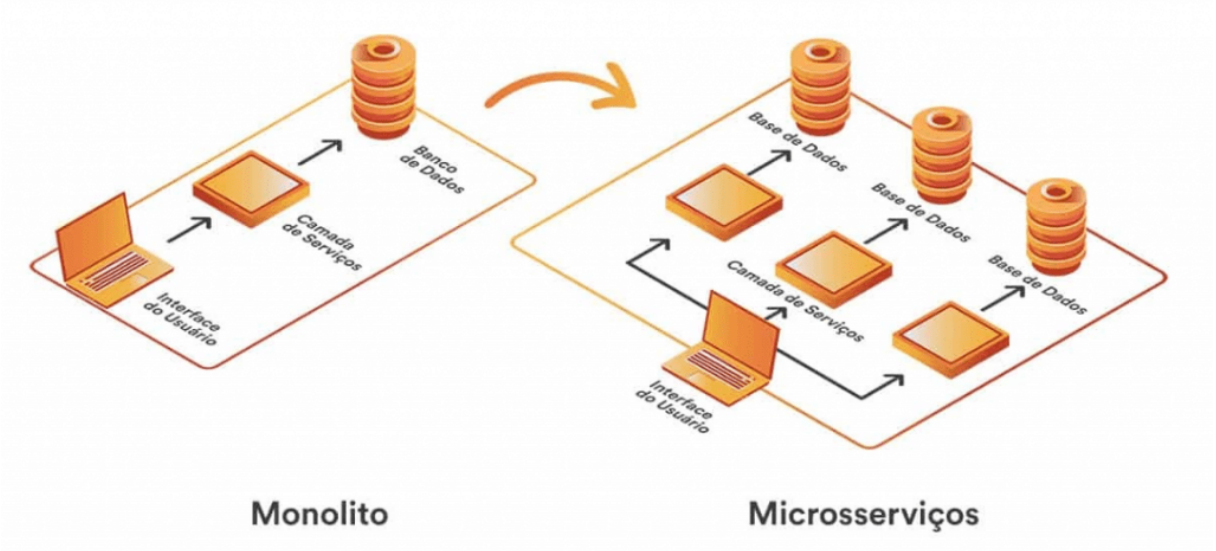
2.1.2 Diferenças entre arquiteturas monolíticas e de microsserviços

Arquiteturas de software são um elemento crucial para o desenvolvimento de aplicações empresariais. As arquiteturas monolíticas são um padrão consolidado, que por muitos anos foram a opção preferida para a construção de aplicações de grande escala. No entanto, com o aumento da complexidade dos sistemas, surgiram

novas alternativas para abordar esses desafios. Nesse contexto, as arquiteturas de microsserviços vêm ganhando cada vez mais espaço, e se apresentam como uma opção viável para lidar com a escalabilidade e a manutenibilidade de aplicações empresariais.

De acordo com Newman (2015), em arquiteturas monolíticas, todo o código da aplicação está contido em um único repositório, em um único deploy e é executado em um único processo. Por outro lado, em arquiteturas de microsserviços, cada serviço é executado em um processo separado, e pode ser implantado e escalado independentemente. Dessa forma, as arquiteturas de microsserviços tornam possível a construção de sistemas complexos, dividindo-os em serviços menores e mais gerenciáveis.

Figura 1 - Comparação entre monolito e microsserviços



Fonte: Supero (2020).

A modularização das aplicações em serviços independentes é uma das principais vantagens das arquiteturas de microsserviços (DRAGONI *et al*., 2017). Ao separar as funcionalidades em módulos, os desenvolvedores podem trabalhar de maneira mais eficiente e independente, facilitando a escalabilidade e a manutenção do sistema. Além disso, as arquiteturas de microsserviços também permitem que diferentes serviços sejam desenvolvidos em diferentes linguagens de programação, o que é uma vantagem em sistemas de grande escala.

No entanto, as arquiteturas de microsserviços também apresentam alguns desafios. Segundo Lewis e Fowler (2014), uma das principais dificuldades na adoção de arquiteturas de microsserviços é a complexidade do gerenciamento de vários serviços independentes, que requer uma abordagem mais avançada para o monitoramento e o gerenciamento de falhas. Além disso, as arquiteturas de microsserviços podem aumentar a complexidade do desenvolvimento e implantação, exigindo uma estratégia bem definida para lidar com essas questões.

Em resumo, as arquiteturas de microsserviços apresentam diversas vantagens em relação às arquiteturas monolíticas, como modularidade, escalabilidade e independência de linguagem de programação. No entanto, a adoção dessas arquiteturas também apresenta desafios, como o gerenciamento de vários serviços independentes e a complexidade do desenvolvimento e implantação. Por isso, é importante avaliar cuidadosamente as vantagens e desafios de cada abordagem antes de tomar uma decisão de arquitetura.

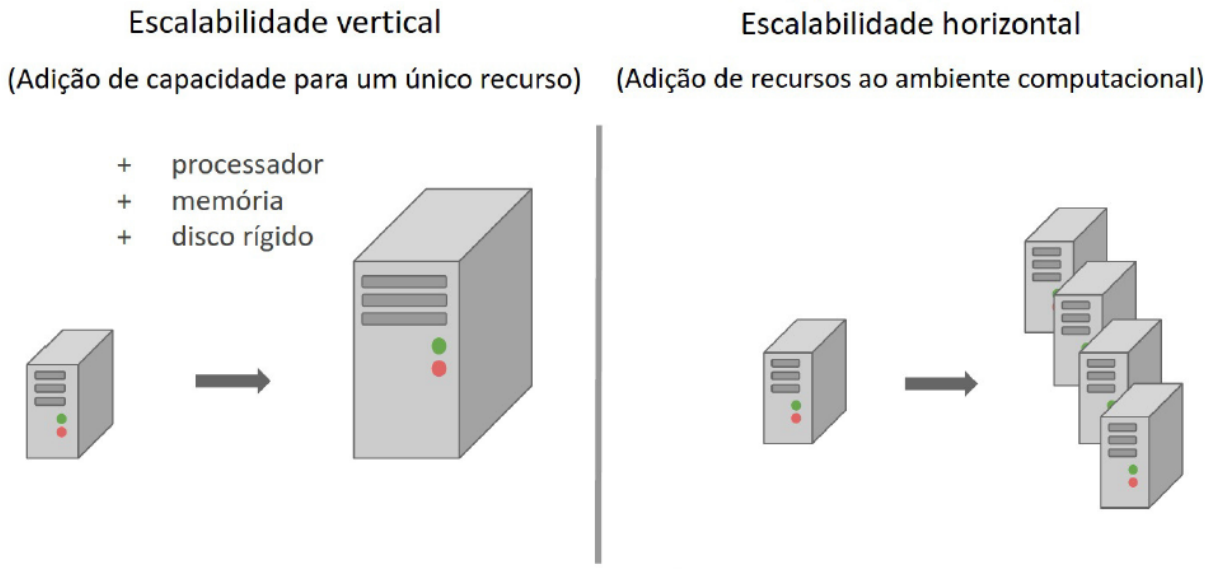
2.1.3 Escalabilidade horizontal e vertical em arquiteturas de microsserviços

Escalabilidade é um aspecto importante em sistemas distribuídos e arquiteturas de microsserviços, permitindo que o sistema aumente sua capacidade para lidar com um aumento na demanda. Duas abordagens comuns para escalabilidade são a escalabilidade horizontal e a escalabilidade vertical. A escalabilidade horizontal envolve adicionar mais instâncias de um serviço para lidar com o aumento na demanda, enquanto a escalabilidade vertical envolve aumentar a capacidade de processamento de cada instância do serviço (NEWMAN, 2015).

A escalabilidade horizontal é uma das principais vantagens das arquiteturas de microsserviços, permitindo que os serviços sejam dimensionados independentemente uns dos outros. Isso permite que as empresas criem sistemas altamente escaláveis e resilientes. No entanto, a escalabilidade horizontal também apresenta desafios, como a necessidade de sincronização de dados entre os serviços e o aumento da complexidade da infraestrutura de rede (LEWIS; FOWLER, 2014).

Por outro lado, a escalabilidade vertical pode ser mais simples de implementar, mas tem um limite na capacidade de processamento de cada instância do serviço. Isso pode levar a gargalos de desempenho e limitações na escalabilidade. As arquiteturas de microsserviços podem aproveitar a escalabilidade vertical usando a abordagem de escalonamento automático baseado em métricas. Isso envolve aumentar automaticamente a capacidade de processamento de uma instância do serviço com base em métricas de desempenho (DRAGONI *et al.*, 2017).

Figura 2 - Escalabilidade Vertical x Escalabilidade Horizontal



Fonte: Marquesone (2016).

Em geral, tanto a escalabilidade horizontal quanto a vertical são importantes para a escalabilidade em arquiteturas de microsserviços. A escolha entre as abordagens deve ser feita com base nos requisitos de desempenho e na arquitetura do sistema. A escalabilidade horizontal é mais adequada para sistemas com muitos serviços pequenos e independentes, enquanto a escalabilidade vertical é mais adequada para serviços grandes e com alta demanda (BEHARA; KHANDRIKA, 2018).

2.1.4 Desacoplamento de serviços e modularidade em microsserviços

O desacoplamento de serviços e a modularidade são duas características fundamentais de uma arquitetura de microsserviços. O desacoplamento refere-se à independência dos serviços, que podem ser desenvolvidos, implantados e escalados

de forma independente, sem afetar outros serviços no sistema. A modularidade, por sua vez, refere-se à divisão do sistema em módulos independentes, que podem ser atualizados ou substituídos sem afetar outros módulos. O desacoplamento de serviços e a modularidade são aspectos essenciais para alcançar a flexibilidade e escalabilidade desejadas em um sistema de microsserviços.

Para alcançar o desacoplamento de serviços em um sistema de microsserviços, é necessário usar protocolos de comunicação e APIs padronizadas, como o REST ou gRPC, e também ferramentas de orquestração de contêineres, como o Kubernetes. Além disso, a modularidade pode ser alcançada através de uma abordagem baseada em domínio, onde cada serviço é responsável por um conjunto específico de funcionalidades relacionadas a um determinado domínio de negócios.

De acordo com Newman (2015), o desacoplamento de serviços e a modularidade são fundamentais para alcançar a escalabilidade horizontal, uma vez que permitem que serviços individuais possam ser escalados independentemente uns dos outros. Além disso, Lewis e Fowler (2014) destacam que a modularidade é uma forma de aumentar a coesão e reduzir o acoplamento entre os módulos, o que facilita o desenvolvimento, implantação e manutenção do sistema como um todo.

Para finalizar, a abordagem de microsserviços é uma escolha de arquitetura cada vez mais popular, mas é importante destacar que sua implementação bem-sucedida depende da adoção de boas práticas de design, desenvolvimento, implantação e manutenção. A modularidade e o desacoplamento de serviços são apenas duas dessas práticas essenciais, que devem ser consideradas ao projetar e implantar um sistema de microsserviços.

2.1.5 Modelagem e Documentação de Arquiteturas de microsserviços: C4, ArchiMate e UML

A documentação e visualização de arquiteturas de software são elementos essenciais no processo de desenvolvimento e manutenção de sistemas complexos. Existem diversos modelos e ferramentas disponíveis para auxiliar os arquitetos de

software nessa tarefa, cada um com suas vantagens e desvantagens. Entre os modelos mais utilizados estão o C4 Model, ArchiMate e UML.

O C4 Model é uma abordagem que se concentra em quatro níveis de abstração: contexto, contêiner, componente e código. Segundo Brown (2017), essa abordagem é simples e fácil de entender, permitindo que as equipes de desenvolvimento e arquitetura possam colaborar na criação e manutenção da documentação. O C4 Model utiliza diagramas simples, mas eficazes, para representar a arquitetura de software, facilitando a comunicação entre as equipes.

O ArchiMate é uma linguagem de modelagem de arquitetura de software que se concentra na modelagem de processos de negócios, sistemas de informação e tecnologia da informação (IT). De acordo com Jonkers *et al*. (2013), o ArchiMate é uma linguagem poderosa e versátil que pode ser usada para modelar diferentes perspectivas da arquitetura de software, desde a perspectiva do negócio até a perspectiva técnica. O ArchiMate também é integrado com outras linguagens, como o UML, permitindo a criação de modelos mais complexos.

O UML é uma linguagem gráfica padrão para modelagem de sistemas de software, que pode ser usada para modelar diferentes perspectivas da arquitetura de software, incluindo a estrutura estática e o comportamento dinâmico. Segundo Chonoles e Schardt (2003), o UML é uma linguagem poderosa e flexível, capaz de representar uma ampla variedade de sistemas, desde sistemas de software simples até sistemas complexos e críticos. O UML também é compatível com outras linguagens e padrões, o que permite a integração com outras ferramentas de modelagem.

Em resumo, a documentação e visualização de arquiteturas de software são elementos críticos para o desenvolvimento e manutenção de sistemas complexos. O C4 Model, ArchiMate e UML são algumas das abordagens mais comuns para a documentação e visualização de arquiteturas de software, cada uma com suas próprias vantagens e desvantagens. É importante avaliar cuidadosamente as

necessidades e objetivos de cada projeto antes de escolher o modelo de documentação e visualização a ser utilizado.

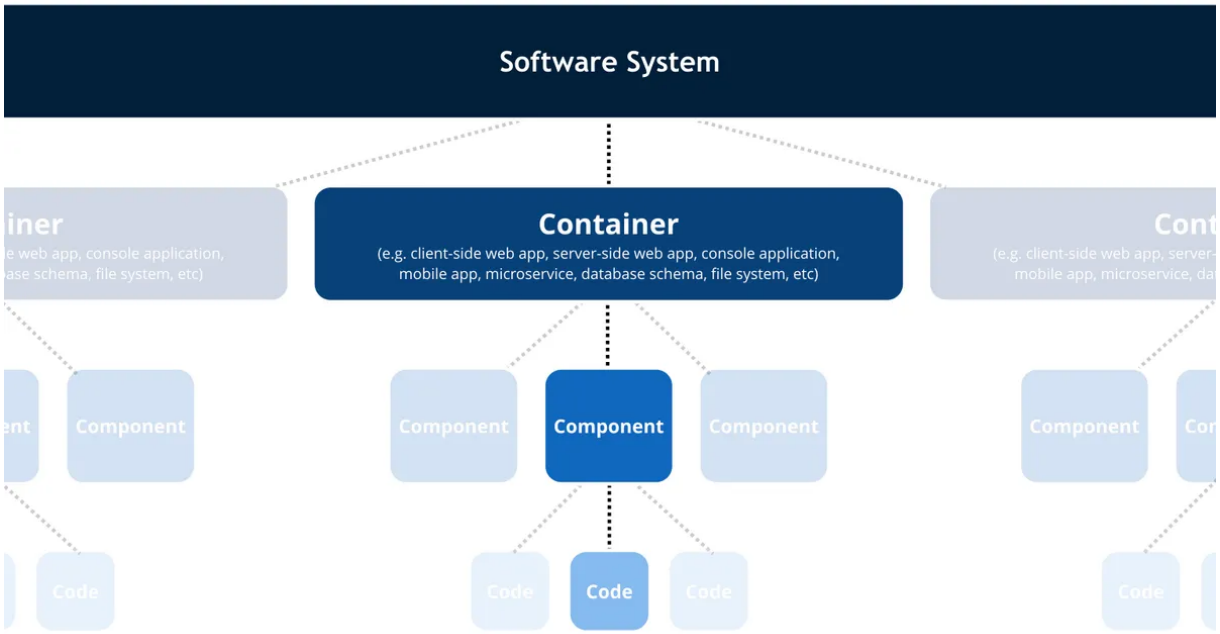
2.1.5.1 C4 Model aplicado a arquitetura de microsserviços

Com a popularização da abordagem de microsserviços na indústria de software, alguns desafios têm sido enfrentados na sua adoção, como o gerenciamento da complexidade arquitetural e a comunicação entre os serviços. Nesse sentido, o C4 Model tem sido uma ferramenta útil para mitigar esses desafios, permitindo a visualização da arquitetura de forma clara e concisa.

Segundo Brown (2017), o C4 Model é uma técnica de modelagem de arquitetura de software que permite a criação de modelos visuais simples, coesos e abrangentes, ajudando a alinhar o entendimento entre todos os stakeholders envolvidos. Além disso, o C4 Model também auxilia na tomada de decisões de design e na comunicação da arquitetura.

A aplicação do C4 Model na abordagem de microsserviços pode ser feita por meio de diagramas contextuais e de contêineres, permitindo a visualização das dependências entre os serviços e a estrutura modular dos mesmos. De acordo com Amundsen *et al.* (2016), a criação desses diagramas pode auxiliar na identificação de problemas de design e na criação de fronteiras bem definidas entre os serviços, permitindo a evolução independente de cada um. Além disso, o uso de diagramas de componentes e classes pode auxiliar no detalhamento da implementação interna de cada serviço, permitindo uma melhor compreensão do seu funcionamento.

Figura 3 - Estrutura de camadas do C4 Model



Fonte: Brown (2018).

O C4 Model também é útil para documentar a arquitetura dos microsserviços, permitindo que todos os stakeholders possam compreender a estrutura e funcionamento dos serviços. Segundo Skelton e Pais (2019), a documentação da arquitetura é um componente essencial para a comunicação e colaboração entre equipes de desenvolvimento e operações, garantindo que todos tenham um entendimento comum sobre a arquitetura. Além disso, a documentação pode auxiliar na identificação de problemas de design e na evolução dos serviços.

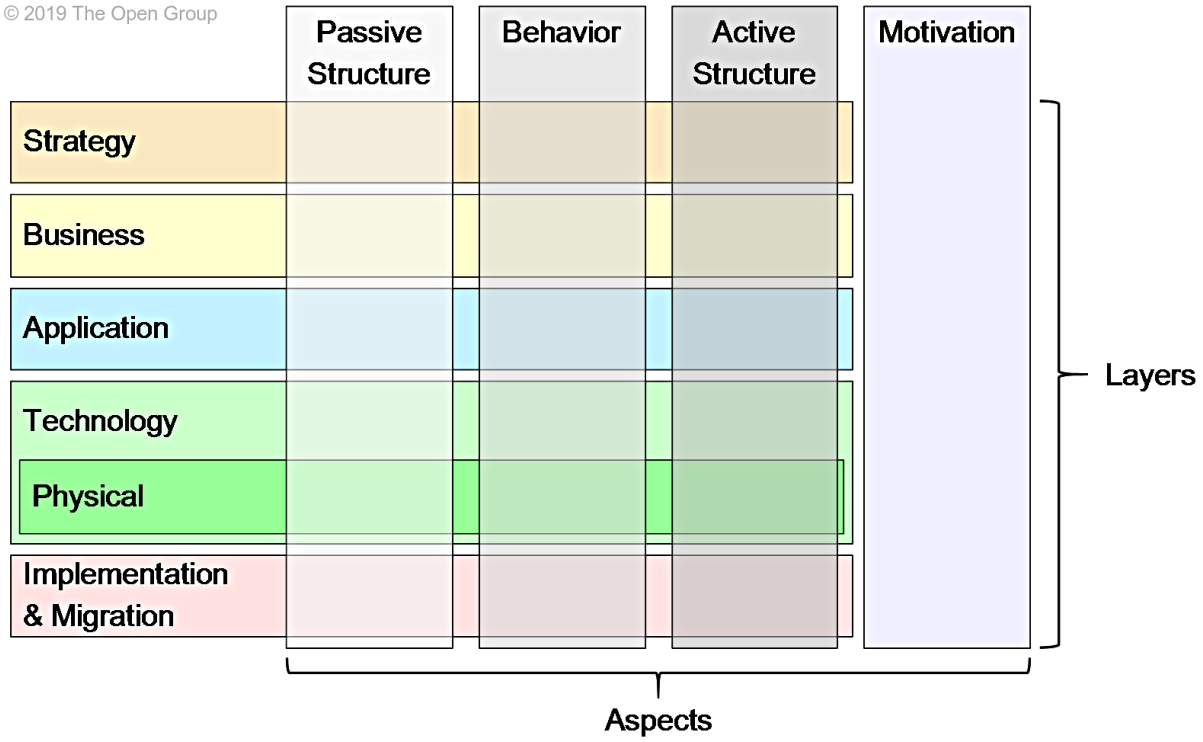
Por fim, a utilização do C4 Model na abordagem de microsserviços pode trazer benefícios para a evolução dos serviços. Segundo Behara e Khandrika (2018), a modularidade dos serviços permite a evolução independente de cada um, e o C4 Model pode ajudar a identificar os pontos de entrada e saída de cada serviço, permitindo a evolução sem impactos para outros serviços. Além disso, a visualização da arquitetura com o C4 Model pode ajudar a identificar onde os esforços de evolução devem ser concentrados.

2.1.5.2 ArchiMate aplicado a arquitetura de microsserviços

A modelagem de arquitetura empresarial é essencial para compreender a estrutura e as relações entre os componentes de um sistema complexo, como é o caso dos microsserviços. O ArchiMate é uma linguagem de modelagem gráfica padronizada e aberta, desenvolvida pelo Open Group, que oferece uma abordagem integrada para descrever a arquitetura de negócios, aplicativos e tecnologia em uma organização (THE OPEN GROUP, 2016).

Quando aplicado à arquitetura de microsserviços, o ArchiMate pode fornecer uma representação visual clara e sistemática dos serviços individuais e suas interações, facilitando a comunicação entre as partes interessadas e auxiliando no projeto e implementação de sistemas baseados em microsserviços (LANKHORST, 2017).

Figura 4 - Estrutura de Linguagem no ArchiMate



Fonte: The Open Group (2019).

O ArchiMate permite a modelagem de diferentes aspectos da arquitetura de microsserviços, incluindo serviços de aplicativos, interfaces, componentes de

infraestrutura e relacionamentos entre eles (THE OPEN GROUP, 2016). Além disso, a linguagem ArchiMate oferece suporte a múltiplos pontos de vista, permitindo que os arquitetos e desenvolvedores se concentrem em aspectos específicos da arquitetura, como segurança, desempenho e escalabilidade, sem se perderem em detalhes desnecessários (LANKHORST, 2017).

A aplicação do ArchiMate na arquitetura de microsserviços pode ajudar a garantir a consistência e a coerência do projeto, facilitando a identificação de problemas potenciais e a análise de impacto das mudanças propostas (LANKHORST, 2017).

Embora o ArchiMate seja uma linguagem poderosa e abrangente para modelar a arquitetura de microsserviços, é importante notar que o sucesso na aplicação do ArchiMate depende da compreensão clara dos conceitos de microsserviços e das melhores práticas de arquitetura (THE OPEN GROUP, 2016). Além disso, o uso adequado da linguagem ArchiMate requer treinamento e experiência, já que a complexidade da linguagem pode dificultar a comunicação eficaz entre as partes interessadas (LANKHORST, 2017).

2.1.5.3 UML aplicado a arquitetura de microsserviços

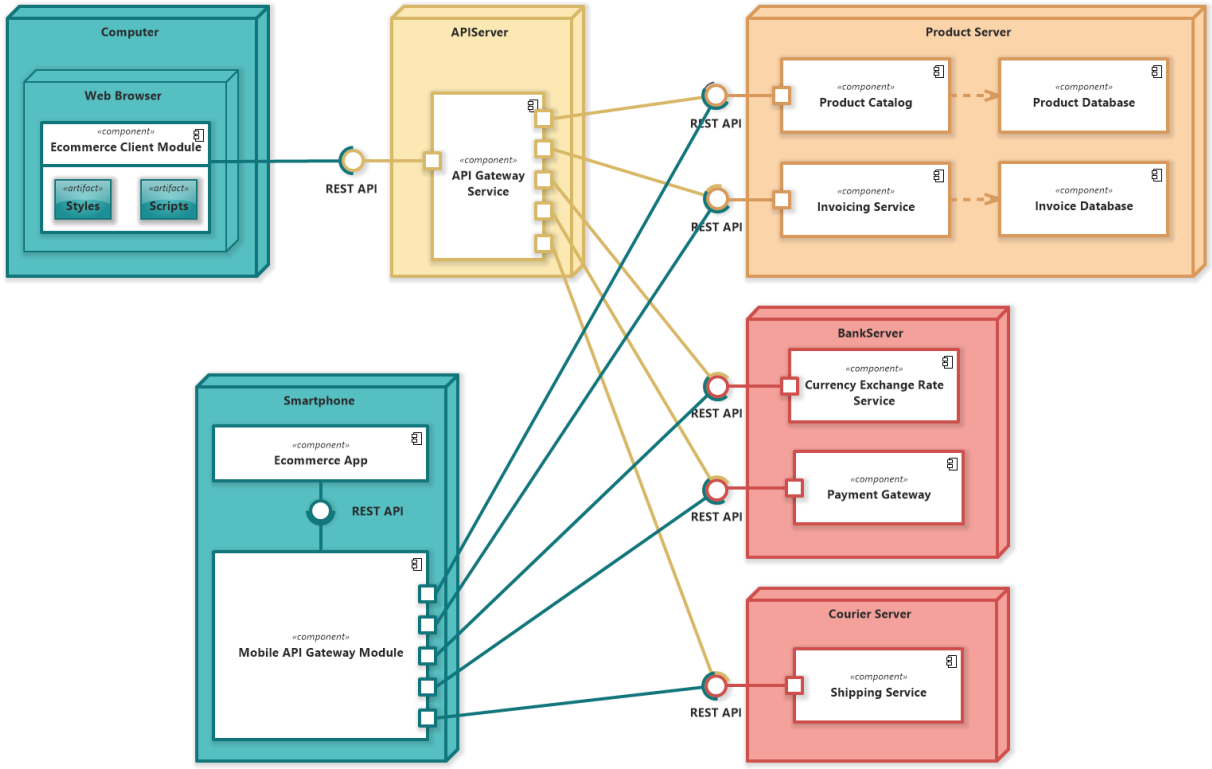
A UML (Unified Modeling Language) é uma linguagem gráfica de modelagem amplamente utilizada na engenharia de software para representar e documentar a arquitetura e o design de sistemas (OMG GROUP, 2015). Ao aplicar a UML à arquitetura de microsserviços, os desenvolvedores podem criar diagramas e modelos visuais que representam os componentes individuais dos serviços, suas interações e os aspectos funcionais e não funcionais dos sistemas baseados em microsserviços (AMBLER, 2004).

A utilização da UML na arquitetura de microsserviços pode ajudar a melhorar a compreensão e a comunicação entre os membros da equipe e as partes interessadas, além de apoiar a tomada de decisões durante a fase de projeto e implementação do sistema (HOHPE; WOOLF, 2015).

A UML oferece uma série de diagramas e notações que podem ser aplicadas à arquitetura de microsserviços, como diagramas de classes, diagramas de sequência, diagramas de componentes e diagramas de implantação, entre outros (OMG GROUP, 2015).

Esses diagramas permitem que os desenvolvedores capturem e representem diferentes aspectos dos sistemas baseados em microsserviços, como a estrutura dos serviços, as relações e dependências entre os serviços, o comportamento dos serviços e a infraestrutura de suporte (BELL, 2008). A utilização desses diagramas pode auxiliar na identificação e resolução de problemas de design e na análise do impacto das mudanças propostas (HOHPE; WOOLF, 2015).

Figura 5 - Exemplo de diagrama UML para arquitetura de microsserviços



Fonte: Rodina (2020).

No entanto, é importante notar que, embora a UML seja uma linguagem de modelagem rica e expressiva, ela pode não abordar completamente os desafios específicos associados à arquitetura de microsserviços, como a comunicação entre os serviços e a necessidade de balanceamento de carga e descoberta de serviços (RICHARDSON, 2018). Além disso, o uso efetivo da UML requer um conhecimento

profundo da linguagem e das melhores práticas de modelagem, e o treinamento adequado é essencial para garantir a comunicação eficaz e a aplicação correta dos diagramas e notações da UML (AMBLER, 2004).

2.2 Vantagens e desafios da abordagem de microsserviços

A abordagem de microsserviços é uma arquitetura de software que permite que sistemas complexos sejam construídos como um conjunto de serviços independentes que se comunicam entre si por meio de interfaces bem definidas. Essa abordagem vem sendo adotada cada vez mais pelas empresas, devido às vantagens que oferece, mas também apresenta desafios significativos a serem enfrentados (LEWIS; FOWLER, 2014).

Entre as principais vantagens da abordagem de microsserviços, pode-se destacar a escalabilidade e a resiliência do sistema. Como cada serviço é independente, é possível escalá-los separadamente, permitindo que a aplicação possa crescer conforme necessário, sem afetar os outros componentes. Além disso, se um serviço falhar, isso não afeta os demais, permitindo que a aplicação continue funcionando parcialmente (LEWIS; FOWLER, 2014).

Além disso, a abordagem de microsserviços exige uma arquitetura de rede robusta e confiável, uma vez que os serviços são independentes e se comunicam por meio de APIs (NEWMAN, 2015). Outro desafio é a gestão da infraestrutura, que se torna mais complexa com a abordagem de microsserviços. Cada serviço precisa ser implantado e gerenciado separadamente, o que aumenta a carga de trabalho da equipe de operações (LEWIS; FOWLER, 2014).

Por fim, é importante ressaltar que a abordagem de microsserviços também pode apresentar desafios na implementação da segurança, uma vez que há uma grande quantidade de serviços independentes a serem protegidos (DRAGONI *et at.*, 2017).

Neste contexto de microsserviços, as vantagens oferecidas incluem escalabilidade e resiliência do sistema, flexibilidade na implementação de diferentes

tecnologias e ferramentas em cada serviço e independência dos serviços, permitindo que a aplicação continue a funcionar mesmo se um serviço falhar.

No entanto, existem desafios significativos a serem enfrentados, como a complexidade da gestão do ambiente distribuído, a necessidade de garantir a integridade dos dados entre os serviços, a necessidade de uma arquitetura de rede robusta e confiável, a complexidade na gestão da infraestrutura e desafios na implementação de segurança devido à grande quantidade de serviços independentes.

Para superar esses desafios, é necessário ter uma equipe altamente capacitada e adotar boas práticas de gestão de projetos e de comunicação entre as equipes. Além disso, é importante avaliar cuidadosamente as opções disponíveis em termos de tecnologias e ferramentas a serem utilizadas na implementação dos serviços, e adotar uma abordagem ágil e iterativa no desenvolvimento dos serviços. A monitoração dos serviços em tempo real também é uma prática importante para garantir a qualidade e a disponibilidade do sistema.

2.2.1 Vantagens dos microsserviços em relação a outras arquiteturas, como monolíticas

A abordagem de microsserviços tem se mostrado uma alternativa viável às arquiteturas monolíticas devido às suas diversas vantagens. Segundo Newman (2015), os microsserviços possibilitam uma maior flexibilidade, escalabilidade e independência de desenvolvimento. Isso se deve ao fato de que cada serviço é responsável por uma funcionalidade específica, o que permite sua evolução de forma isolada, sem afetar o funcionamento dos demais serviços. Além disso, a modularização da aplicação em serviços menores facilita o gerenciamento do código fonte e a adoção de práticas de desenvolvimento ágil.

Outra vantagem dos microsserviços em relação às arquiteturas monolíticas é a possibilidade de utilizar tecnologias e ferramentas diferentes em cada serviço, de acordo com suas necessidades específicas (LEWIS; FOWLER, 2014). Isso permite a utilização de tecnologias mais modernas e adequadas para cada contexto, além de

facilitar a adoção de novas tecnologias de forma gradual e sem impactar toda a aplicação.

Contudo, a adoção de arquiteturas de microsserviços também apresenta desafios, como a complexidade do gerenciamento de múltiplos serviços interconectados (DRAGONI *et al*., 2017). O uso de tecnologias diferentes em cada serviço pode gerar dificuldades na integração e comunicação entre os serviços. Além disso, é preciso garantir a disponibilidade e a escalabilidade de cada serviço individualmente, o que pode demandar uma infraestrutura mais robusta e complexa.

Apesar dos desafios, as vantagens apresentadas pelos microsserviços têm atraído cada vez mais empresas e organizações para essa abordagem arquitetural. A utilização de boas práticas de desenvolvimento e o uso de ferramentas e técnicas específicas para o gerenciamento de microsserviços podem ajudar a mitigar os desafios e garantir uma transição bem-sucedida para essa abordagem.

2.2.2 Escalabilidade: como a arquitetura de microsserviços ajuda a escalar serviços de forma independente

A escalabilidade é um aspecto chave em sistemas de software modernos, e a arquitetura de microsserviços é uma abordagem que ajuda a garantir a escalabilidade de forma independente. De acordo com Lewis e Fowler (2014), a arquitetura de microsserviços é construída em torno de serviços independentes e de tamanho reduzido, que podem ser implantados e escalados independentemente. Isso é possível devido à estrutura de comunicação baseada em APIs e à descentralização do controle do sistema.

Para assegurar a escalabilidade dos serviços em uma arquitetura de microsserviços, é crucial considerar fatores como a orquestração de serviços, a seleção de ferramentas e tecnologias apropriadas e a estratégia de implantação. Em uma pesquisa conduzida por Balalaie, Heydarnoori e Jamshidi (2016), os autores examinaram o impacto de várias abordagens de orquestração de serviços na escalabilidade de sistemas de microsserviços. Eles descobriram que o emprego de ferramentas como o Kubernetes pode contribuir para garantir a escalabilidade,

embora seja essencial analisar meticulosamente as exigências do sistema e optar pela abordagem mais adequada.

Outro aspecto importante é a escolha de tecnologias adequadas para cada serviço. Como cada serviço em uma arquitetura de microsserviços é independente, é possível escolher tecnologias diferentes para cada um deles. No entanto, é importante garantir a compatibilidade entre os serviços e pensar na escalabilidade de cada tecnologia escolhida. De acordo com Newman (2015), a escolha de tecnologias de comunicação eficientes, como REST e mensageria, pode ajudar a garantir a escalabilidade e resiliência do sistema.

Por fim, é importante pensar em estratégias de deployment adequadas para garantir a escalabilidade dos serviços. Uma das abordagens mais utilizadas é o uso de containers, que permitem uma implantação rápida e fácil dos serviços, além de possibilitar a escalabilidade vertical e horizontal.

2.2.3 Desacoplamento: benefícios e desafios do desacoplamento de serviços em arquiteturas de microsserviços.

Arquiteturas de microsserviços têm como um de seus principais pilares o desacoplamento dos serviços, permitindo que cada um possa ser desenvolvido, testado e implantado independentemente dos outros. Conforme apontado por Newman (2015), essa abordagem traz benefícios como a capacidade de escalar partes específicas do sistema sem afetar os demais serviços, além de facilitar a manutenção e evolução do sistema como um todo. Contudo, o desacoplamento também pode trazer desafios, especialmente no que diz respeito à comunicação entre serviços, que precisa ser cuidadosamente planejada e gerenciada.

Uma das estratégias para lidar com os desafios do desacoplamento é a implementação de contratos entre os serviços, como proposto por Richardson (2018). Esses contratos definem a forma como os serviços devem interagir entre si, especificando os tipos de dados e as interfaces de comunicação que cada um deve fornecer e consumir. Dessa forma, cada serviço pode ser desenvolvido de forma

independente, mas ainda assim integrado de maneira confiável com os outros serviços.

Outra abordagem para garantir o desacoplamento é a utilização de um barramento de serviços, como proposto por Behara e Khandrika (2018). Esse barramento é responsável por intermediar a comunicação entre os serviços, garantindo que cada um possa ser implantado e atualizado independentemente dos outros. Além disso, o barramento pode implementar funcionalidades como roteamento, tradução de protocolos e controle de versão, facilitando ainda mais o gerenciamento de serviços em arquiteturas de microsserviços.

Por fim, é importante destacar que o desacoplamento em arquiteturas de microsserviços não é uma tarefa trivial e requer um cuidadoso planejamento e gerenciamento. Como afirmado por Newman (2015), é necessário pensar cuidadosamente sobre a forma como os serviços serão divididos e integrados, bem como sobre as estratégias de comunicação e gerenciamento de dados que serão utilizadas para garantir a interoperabilidade entre eles.

2.2.4 Resiliência: estratégias para lidar com falhas em serviços de microsserviços

A implementação de uma arquitetura de microsserviços traz muitos benefícios, como a escalabilidade e a flexibilidade dos serviços, porém, também apresenta desafios relacionados à manutenção e gerenciamento desses serviços, especialmente quando se trata de lidar com falhas.

Nesse contexto, diversos autores propõem estratégias para minimizar os efeitos de falhas em serviços de microsserviços. Segundo Newman (2015), é importante ter uma abordagem de design que preveja falhas nos serviços. Ele sugere o uso de bibliotecas e frameworks que ofereçam recursos de "*circuit breakers"*, como o Hystrix, para lidar com erros e falhas em serviços, evitando assim a propagação de falhas em cascata.

Outra estratégia utilizada por Cockcroft (2020) é a adoção de práticas de resiliência, tais como o uso de técnicas de "*retries e timeouts*", para evitar que as

falhas afetem diretamente os usuários. Ele também sugere o uso de testes de estresse e caos, que simulam situações extremas de falhas em serviços, permitindo que os desenvolvedores identifiquem e corrijam possíveis pontos de falhas. Além disso, Cockcroft (2020), enfatiza a importância do monitoramento constante dos serviços, utilizando ferramentas de monitoramento e *logging*, para que as falhas possam ser detectadas e corrigidas rapidamente.

2.2.5 Autenticação e autorização: como garantir a autenticação e autorização em arquiteturas de microsserviços

A autenticação e a autorização são aspectos críticos de segurança em sistemas de software, especialmente em arquiteturas de microsserviços, onde vários serviços independentes interagem entre si (NEWMAN, 2015). Autenticação refere-se ao processo de verificação da identidade de um usuário ou cliente, enquanto autorização diz respeito ao controle de acesso aos recursos e funcionalidades do sistema com base nas permissões atribuídas ao usuário ou cliente autenticado (RICHARDSON, 2018). Garantir a autenticação e autorização eficazes em arquiteturas de microsserviços é fundamental para proteger os serviços e dados contra acessos não autorizados e potenciais ameaças de segurança (NEWMAN, 2015).

Figura 6 - Exemplificação de autorização e autenticação

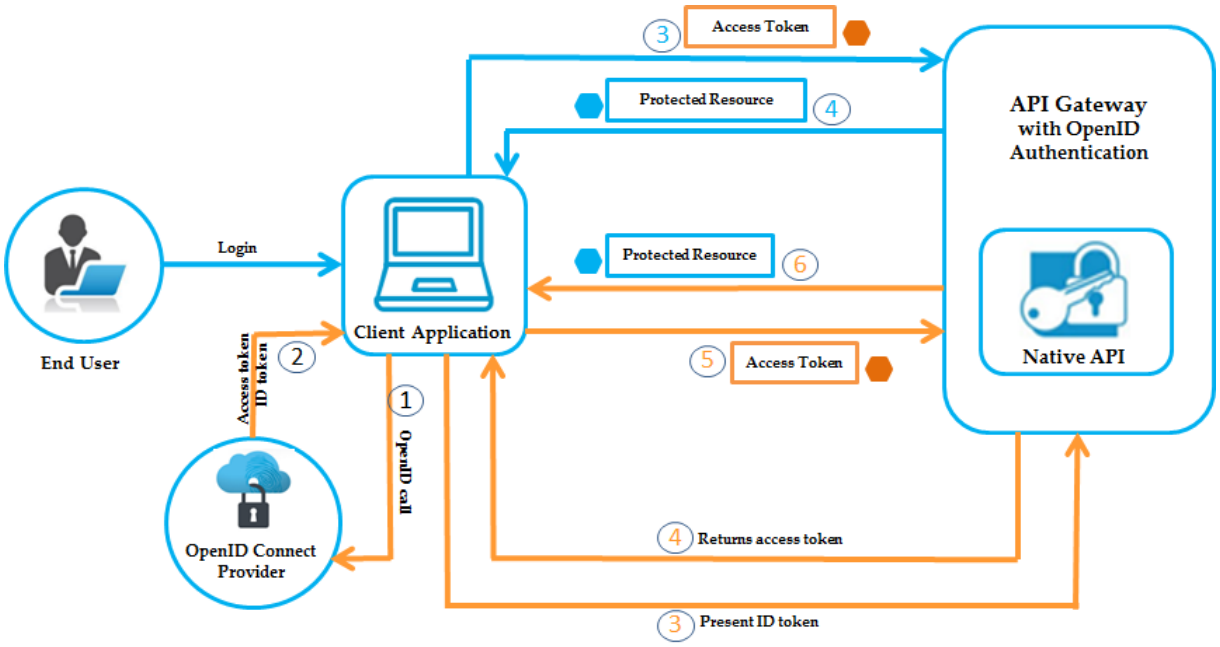
Fonte: Lima (2019).

Uma abordagem comum para lidar com autenticação e autorização em arquiteturas de microsserviços é a utilização de um serviço de gerenciamento de identidade e acesso centralizado, como o OAuth 2.0 ou OpenID Connect (HARDT,

2012). Esses protocolos baseados em token permitem que os usuários e clientes autentiquem-se e recebam tokens de acesso que podem ser usados para acessar os serviços e recursos do sistema (RICHARDSON, 2018).

Os serviços de microsserviços podem validar e verificar esses tokens de acesso para garantir que o usuário ou cliente tenha as permissões adequadas para realizar as ações solicitadas (NEWMAN, 2015). Essa abordagem centralizada simplifica a gestão de autenticação e autorização e promove a reutilização e a interoperabilidade entre os serviços (RICHARDSON, 2018).

Figura 7 - Caso de uso e fluxo de trabalho de autenticação OpenID



Fonte: SoftwareAG (2022).

Além disso, os padrões e práticas recomendadas para garantir a autenticação e autorização eficazes em arquiteturas de microsserviços incluem o uso de TLS (Transport Layer Security) para proteger a comunicação entre os serviços e a aplicação do princípio do menor privilégio, garantindo que os usuários e clientes tenham acesso apenas aos recursos e funcionalidades necessários para realizar suas tarefas (NEWMAN, 2015; RICHARDSON, 2018). Também é importante monitorar e auditar continuamente os eventos e atividades relacionados à autenticação e autorização para detectar e responder a possíveis violações de

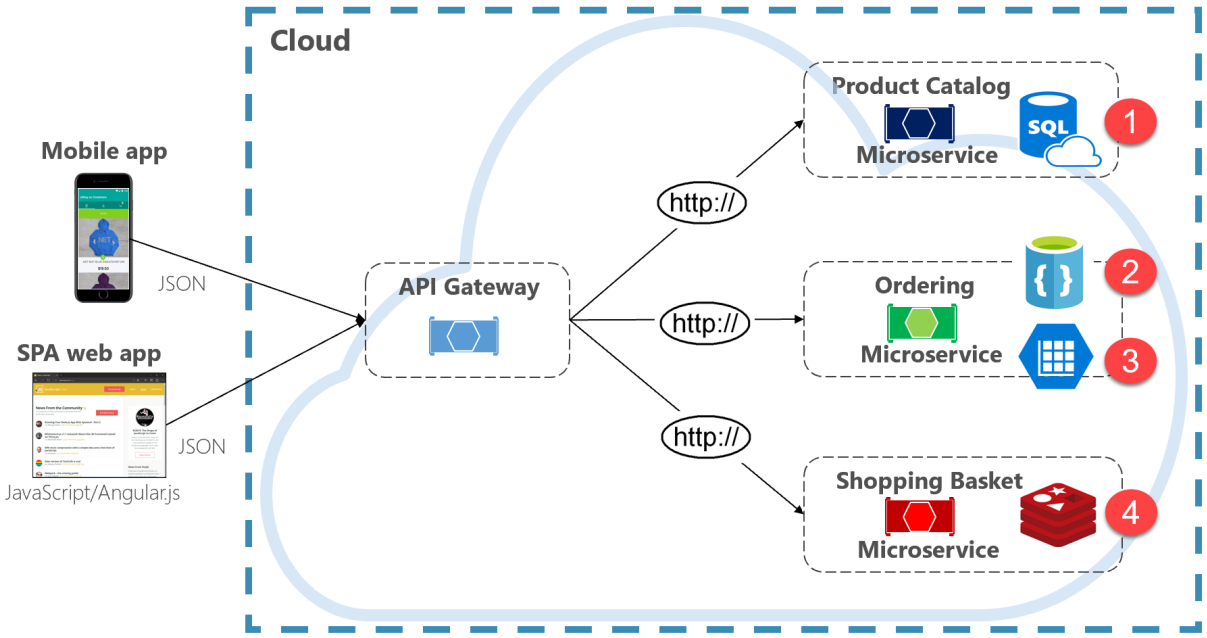
segurança e garantir a conformidade com as políticas e regulamentações aplicáveis (NEWMAN, 2015).

2.2.6 Gerenciamento de dados e persistência

O gerenciamento de dados e persistência em arquiteturas de microsserviços apresenta desafios únicos em comparação com sistemas monolíticos. Diversas estratégias e padrões foram propostos para lidar com esses desafios e garantir a eficiência e a consistência dos dados em um ambiente distribuído.

Um princípio fundamental é que cada micro serviço deve ser responsável por seu próprio armazenamento e gerenciamento de dados, o que implica ter um banco de dados dedicado para cada serviço (NEWMAN, 2015; LEWIS; FOWLER, 2014). Isso permite que os microsserviços evoluam e escalem de forma independente, reduzindo o acoplamento e a complexidade.

Figura 8 - Padrões de dados nativos de nuvem



Fonte: Microsoft (2022).

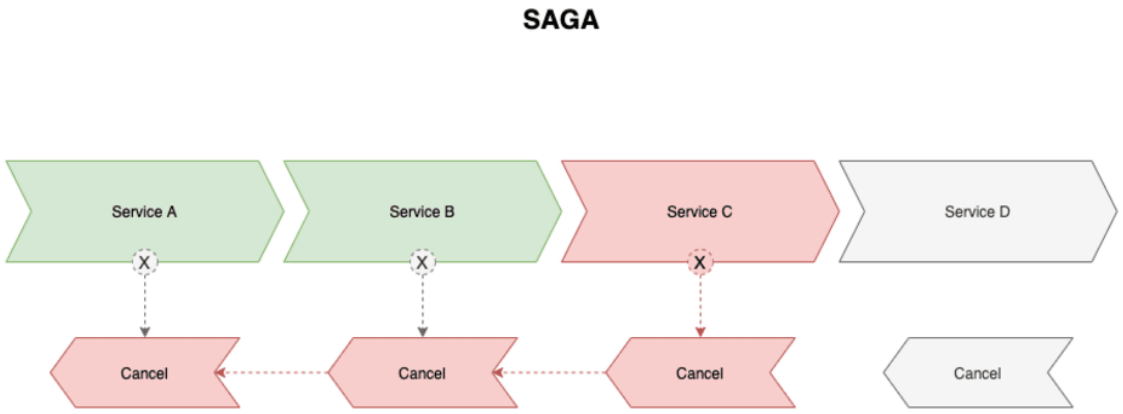
Em sistemas de microsserviços, é comum enfrentar problemas de consistência de dados devido à natureza distribuída dos serviços. Para lidar com essas questões, muitos sistemas adotam o conceito de eventual consistência, no qual os serviços são autorizados a ter dados inconsistentes temporariamente, com a

expectativa de que a consistência seja alcançada em algum momento no futuro (VOGELS, 2009). Essa abordagem permite maior disponibilidade e tolerância a falhas, mas pode exigir que os desenvolvedores lidem com situações em que os dados não estão imediatamente sincronizados entre os serviços.

Uma abordagem eficiente para lidar com transações distribuídas em sistemas de microsserviços é o padrão SAGA (RICHARDSON, 2018). O padrão SAGA divide uma transação de longa duração em várias etapas menores, chamadas de sagas. Cada etapa é uma transação local que pode ser desfeita por uma compensação correspondente.

Se uma etapa falhar, o sistema executará as compensações para todas as etapas concluídas anteriormente, garantindo assim a consistência do sistema. O padrão SAGA é particularmente útil em ambientes de microsserviços, onde a coordenação entre serviços é crucial para manter a consistência dos dados.

Figura 9 - SAGA Pattern aplicado em microsserviços



Fonte: Henrique (2021).

Além dessas estratégias, é importante considerar a integração de diferentes tecnologias de armazenamento e cache para melhorar o desempenho e a escalabilidade dos sistemas de microsserviços. Tecnologias como o Redis ou o Memcached podem ser empregadas para implementar o cache distribuído e reduzir a carga nos bancos de dados (NEWMAN, 2015). Além disso, elas também podem ser utilizadas para implementar padrões de cache de segundo nível, como o

Hibernate Second-Level Cache, que permite armazenar em cache os resultados de consultas realizadas pelo ORM Hibernate.

Em resumo, o gerenciamento de dados e persistência em arquiteturas de microsserviços exige a utilização de estratégias e padrões específicos para lidar com a complexidade e a distribuição dos dados. Bancos de dados dedicados, eventual consistência, padrão SAGA e cache distribuído são apenas algumas das abordagens importantes para garantir a eficiência e a consistência dos dados em um ambiente distribuído (NEWMAN, 2015; LEWIS; FOWLER, 2014; VOGELS, 2009; RICHARDSON, 2018).

2.3 Melhores práticas de design de microsserviços

Uma arquitetura de microsserviços bem-sucedida exige a adoção de melhores práticas de design, juntamente com o uso de ferramentas e tecnologias apropriadas. Um dos principais aspectos do design de microsserviços é a criação de serviços autônomos e independentes que possam ser implantados e gerenciados separadamente, garantindo assim a flexibilidade e escalabilidade do sistema (WOLFF, 2016).

Além disso, é crucial estabelecer APIs bem definidas e padronizadas para a comunicação entre os serviços, permitindo que diferentes equipes desenvolvam e mantenham os serviços sem problemas de integração e proporcionando flexibilidade e extensibilidade para adicionar novas funcionalidades sem afetar a integridade do sistema (WOLFF, 2016).

No que diz respeito à segurança, é importante garantir a segurança da comunicação entre os serviços e a proteção dos dados do sistema, utilizando técnicas de autenticação, autorização e criptografia (TAIBI; LENARDUZZI; PAHL, 2017).

A escolha de ferramentas e tecnologias adequadas, como ferramentas de automação e contêineres, também é fundamental para facilitar a implantação e o gerenciamento dos serviços, bem como aumentar a flexibilidade e a escalabilidade do sistema (WOLFF, 2016). Além disso, a realização de testes automatizados

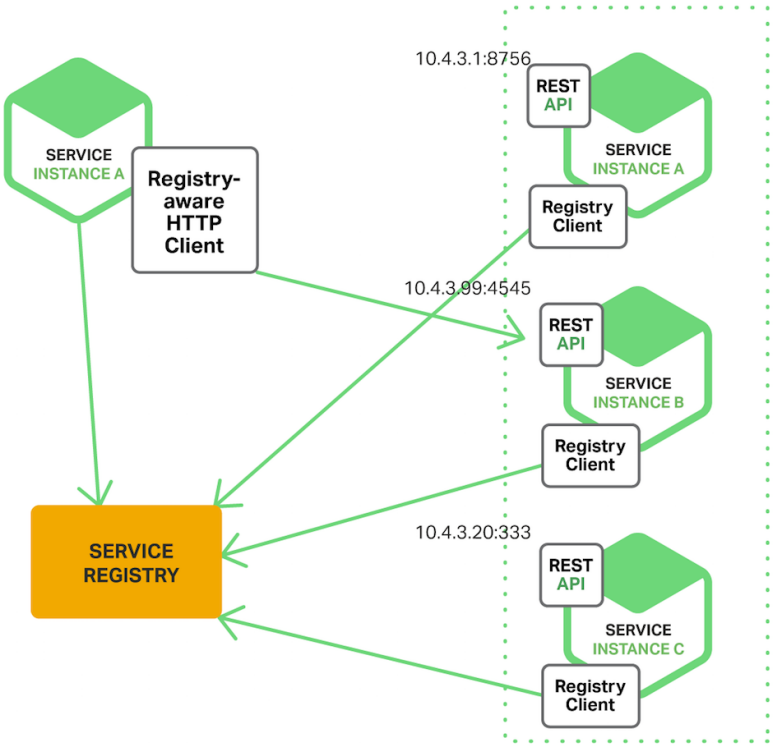
integrados ao processo de desenvolvimento e monitoramento em tempo real dos serviços é essencial para garantir a qualidade, a disponibilidade e o desempenho do sistema como um todo (WOLFF, 2016).

Em resumo, a implementação bem-sucedida de uma arquitetura de microsserviços envolve a adoção de melhores práticas de design e a seleção de ferramentas e tecnologias adequadas. Isso inclui a criação de serviços autônomos e independentes, o uso de APIs bem definidas e padronizadas, garantindo a segurança e proteção dos dados, e a escolha de ferramentas de automação e contêineres. A realização de testes automatizados e o monitoramento contínuo dos serviços são cruciais para manter a qualidade, disponibilidade e desempenho do sistema em geral.

2.3.1 Padrões de projeto: principais padrões de projeto para arquiteturas de microsserviços.

A arquitetura de microsserviços tem ganhado popularidade nos últimos anos, e o uso de padrões de projeto é fundamental para garantir a qualidade, flexibilidade e manutenção dessa arquitetura. Dentre os padrões de projeto mais utilizados em arquiteturas de microsserviços, destaca-se o padrão Service Registry. Segundo Newman (2015), o Service Registry é um componente que armazena informações sobre os serviços disponíveis, permitindo que os clientes descubram e se comuniquem com esses serviços de forma dinâmica e transparente.

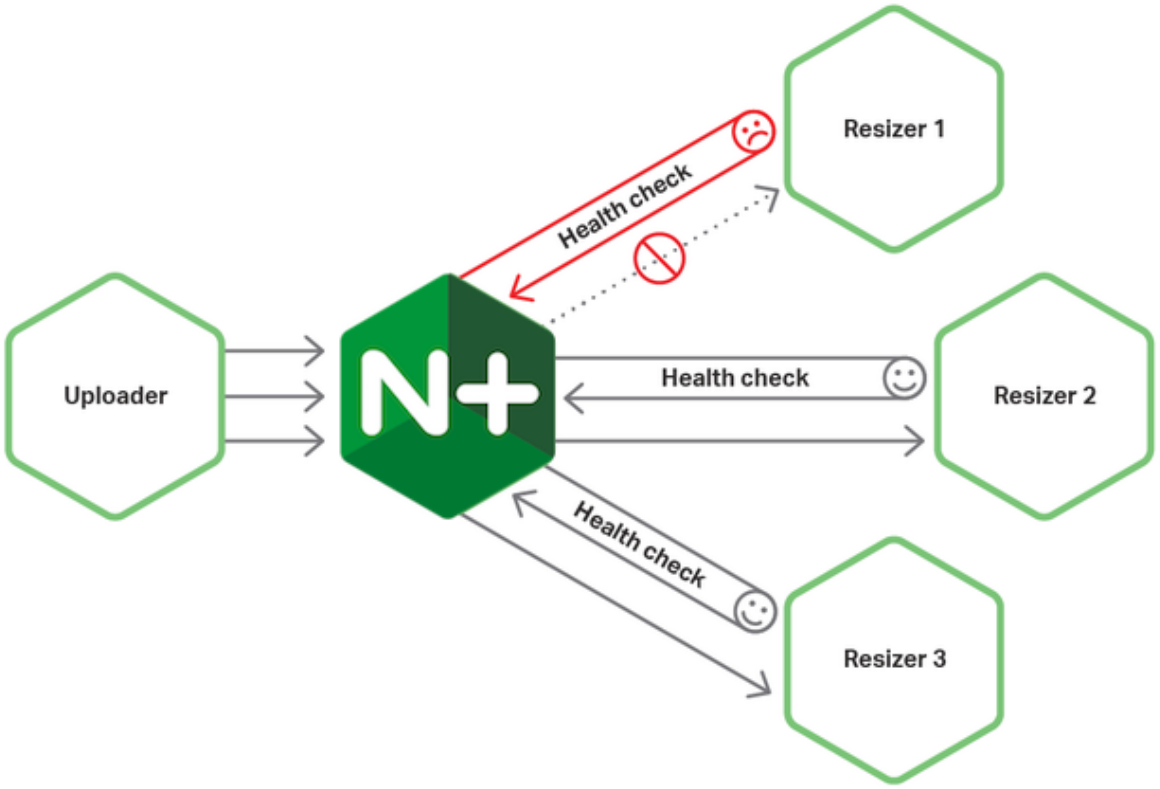
Figura 10 - Exemplificando o Service Registry



Fonte: Richardson (2015).

Outro padrão de projeto importante para arquiteturas de microsserviços é o Circuit Breaker. Conforme apontado por Lewis e Fowler (2014), o Circuit Breaker é um padrão que ajuda a lidar com falhas de serviços, evitando que elas se propaguem para outros serviços e causando a queda de toda a aplicação. O padrão consiste em monitorar o estado do serviço e, caso ele esteja indisponível, abrir um circuito que impede a comunicação com o serviço. Com isso, o sistema pode se adaptar às falhas de forma mais resiliente e tolerante a falhas.

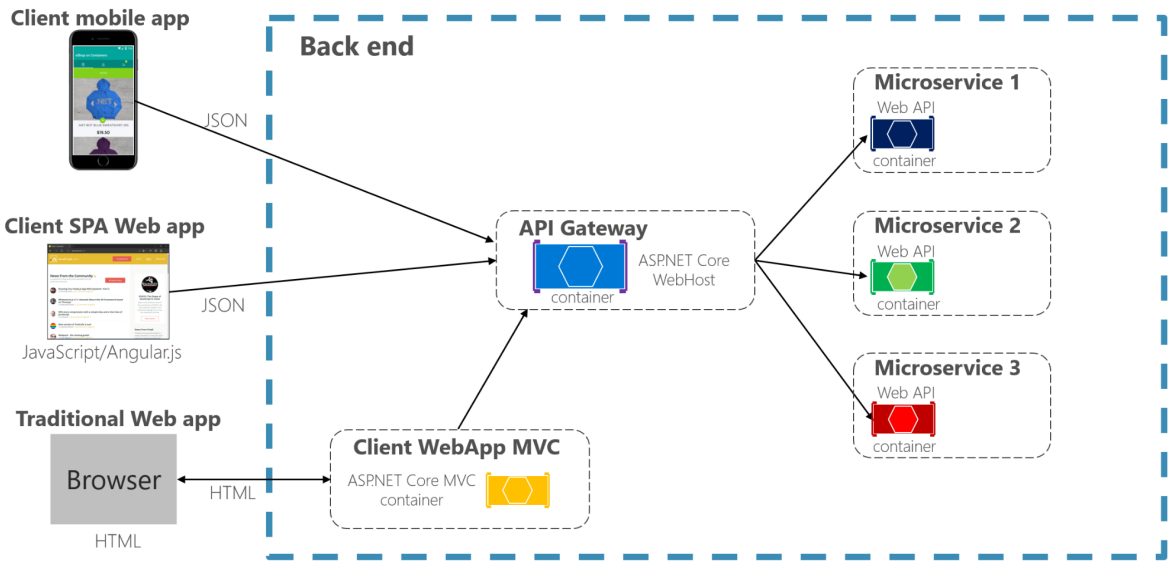
Figura 11 - Exemplificando o Circuit Breaker



Fonte: Stetson (2016).

Por fim, outro padrão de projeto importante para arquiteturas de microsserviços é o API Gateway. Segundo Newman (2015), o API Gateway é um componente que atua como ponto de entrada para a aplicação, recebendo e direcionando as requisições para os serviços correspondentes. Além disso, o API Gateway também pode ser utilizado para implementar autenticação e autorização, monitoramento e outras funcionalidades comuns a vários serviços da aplicação, centralizando essas funcionalidades em um único componente.

Figura 12 - Exemplificando o API Gateway



Fonte: Microsoft (2022).

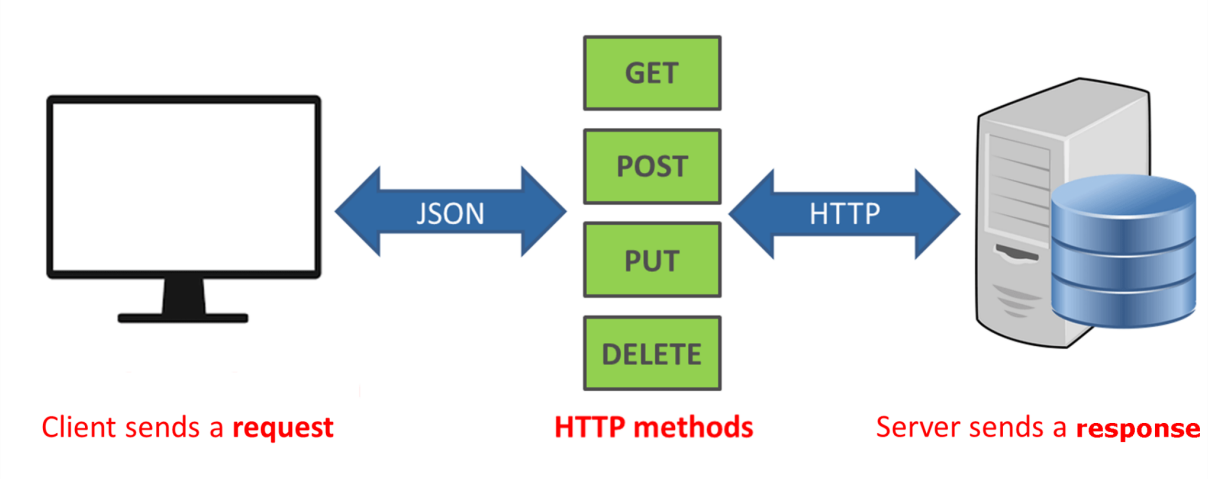
A utilização desses e outros padrões de projeto em arquiteturas de microsserviços contribui para a implementação de sistemas mais flexíveis, escaláveis e fáceis de manter. No entanto, é importante destacar que a escolha e implementação desses padrões deve ser feita com cuidado e atenção às características específicas da aplicação em questão.

2.3.2 Comunicação entre microsserviços: abordagens e ferramentas para garantir a comunicação entre os diferentes serviços

A comunicação entre microsserviços é uma parte crucial do projeto e implementação de sistemas baseados nessa arquitetura. Segundo Newman (2015), existem várias abordagens e ferramentas para garantir a comunicação eficiente e confiável entre os diferentes serviços.

Uma das abordagens mais comuns é utilizar APIs baseadas em REST (Representational State Transfer), que oferecem uma maneira simples e padronizada de trocar informações entre serviços usando o protocolo HTTP (RICHARDSON; AMUNDSEN, 2013).

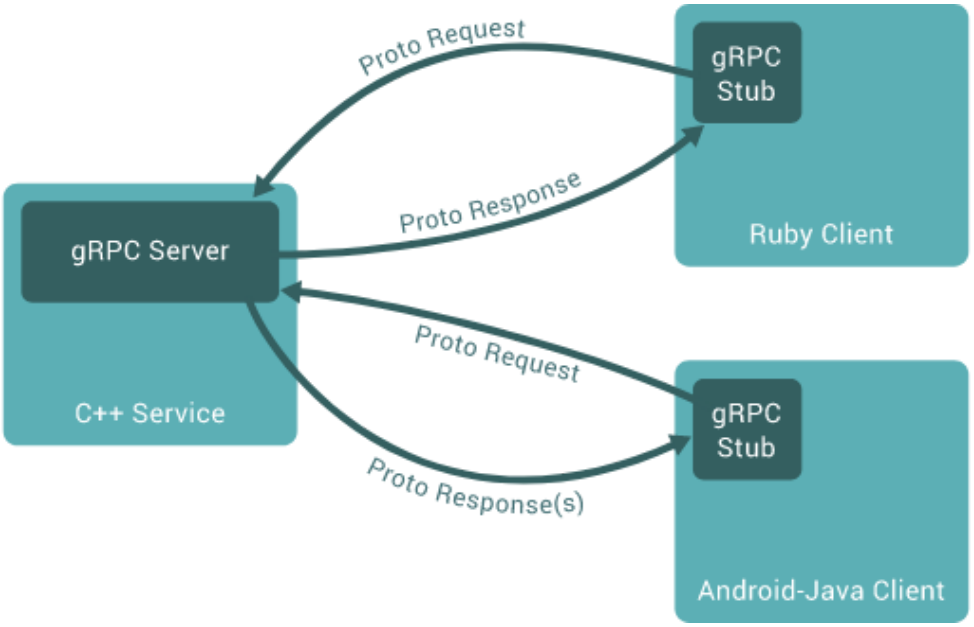
Figura 13 - Exemplificando o REST API



Fonte: Benharosh (2018).

Outra opção de comunicação é o uso de gRPC, um framework desenvolvido pelo Google que utiliza o protocolo HTTP/2 e Protocol Buffers para troca de dados entre serviços (GRPC, 2022). A abordagem gRPC permite comunicação mais eficiente e rápida, especialmente quando comparada ao uso de REST e JSON, além de suportar streaming bidirecional e outros recursos avançados.

Figura 14 - Introdução ao gRPC



Fonte: gRPC (2023).

Além das abordagens baseadas em protocolos síncronos, a comunicação assíncrona por meio de sistemas de mensagens também é amplamente utilizada em arquiteturas de microsserviços. Mensagens assíncronas, como as oferecidas por Apache Kafka ou RabbitMQ, proporcionam um desacoplamento maior entre os serviços e permitem lidar melhor com cenários de falhas e latência (LEWIS; FOWLER, 2014; KLEPPMANN, 2017).

Por fim, é importante destacar a importância de considerar padrões de design e melhores práticas para garantir a comunicação eficiente entre microsserviços. Além das ferramentas e abordagens mencionadas, é fundamental adotar estratégias de design como "*circuit breakers*", timeouts e mecanismos de retentativa para lidar com cenários de falha e garantir a resiliência dos sistemas (LEWIS; FOWLER, 2014; NYGARD, 2018).

2.3.3 Desenvolvimento e deployment: melhores práticas de desenvolvimento e deployment de microsserviços

Newman (2015) enfatiza a importância de uma cultura de automação entre as equipes de desenvolvimento de microsserviços, abrangendo desde a construção até a implantação e operação dos serviços. A utilização de ferramentas de automação de testes, integração contínua e implantação contínua é crucial para garantir a qualidade do código e a agilidade no processo de deployment. A adoção de práticas de DevOps, que integram desenvolvimento e operações, também é essencial para promover a colaboração entre as equipes e a entrega contínua dos serviços.

Chacon e Straub (2014) destacam que o sucesso na implementação de arquiteturas de microsserviços depende da adoção de boas práticas de desenvolvimento, como o uso de controle de versão e integração contínua. A automação de testes e o monitoramento constante da aplicação são vitais para garantir a qualidade e o bom funcionamento dos serviços.

Lewis e Fowler (2014) complementam essa perspectiva, ressaltando a importância do uso de tecnologias de conteinerização, como o Docker, para empacotar serviços e suas dependências. Isso facilita o isolamento dos serviços, a

escalabilidade e o deployment. A utilização de ferramentas de orquestração de containers, como o Kubernetes, também contribui para a gestão automatizada de serviços em larga escala, aumentando a eficiência e minimizando erros humanos.

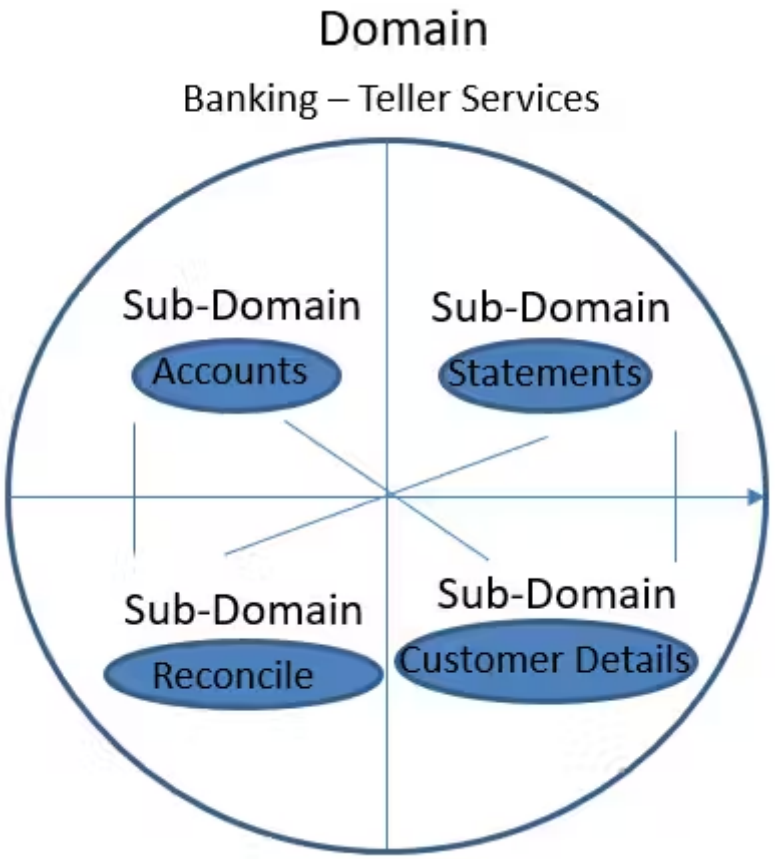
Por fim, Lewis e Fowler (2014) salientam a relevância de seguir práticas de design de APIs que permitam a evolução independente dos serviços. Isso envolve o uso de contratos de APIs bem definidos, versões de APIs e a adoção de princípios RESTful. Além disso, é crucial que a equipe de desenvolvimento tenha uma visão holística da arquitetura de microsserviços, assegurando que cada serviço seja responsável por uma única tarefa e que a comunicação entre eles ocorra de maneira eficiente e segura.

2.3.4 Domain-Driven Design: Aplicação em projetos de software complexos

Domain-Driven Design (DDD) é uma abordagem de desenvolvimento de software que visa a criação de aplicações que reflitam o domínio de negócios do cliente. O objetivo do DDD é criar um modelo de domínio rico e expressivo, que seja facilmente compreendido pelos desenvolvedores e especialistas em negócios. Em projetos de software complexos, o DDD pode ser especialmente útil, pois permite a modelagem do domínio de negócios de forma mais precisa e coerente (EVANS, 2016).

O uso do DDD em projetos de software complexos pode trazer muitos benefícios. Uma das principais vantagens é a possibilidade de lidar com a complexidade do negócio, que pode ser difícil de compreender e documentar. Com o DDD, é possível criar um modelo de domínio que seja claro e consistente, permitindo uma melhor comunicação entre desenvolvedores e especialistas em negócios (VERNON, 2013).

Figura 15 - Exemplo de DDD aplicado em microsserviços



Fonte: Ragan (2023).

Para aplicar o DDD em projetos de software complexos, é necessário seguir alguns princípios e padrões definidos na abordagem. Por exemplo, é importante definir os limites de contexto do domínio e criar uma linguagem ubíqua que seja compreendida por todos os envolvidos no projeto. Além disso, é importante criar um modelo de domínio que seja expressivo e coeso, utilizando técnicas como a identificação de agregados e a criação de eventos de domínio (FOWLER, 2002).

Apesar de trazer muitos benefícios, a aplicação do DDD em projetos de software complexos pode ser desafiadora. É necessário um bom entendimento do negócio e da tecnologia utilizada, além de uma equipe altamente qualificada e engajada no processo de desenvolvimento. É importante também considerar que o DDD não é a solução para todos os problemas de software, e que sua aplicação deve ser cuidadosamente avaliada para cada projeto em particular (FOWLER, 2002).

2.3.5 Segurança: melhores práticas para garantir a segurança em arquiteturas de microsserviços

A segurança é um aspecto fundamental em arquiteturas de microsserviços, uma vez que a comunicação entre os serviços pode ocorrer em redes inseguras e expor informações sensíveis. Para garantir a segurança em arquiteturas de microsserviços, algumas práticas são recomendadas. De acordo com Newman (2015), é essencial que cada microsserviço possua seu próprio controle de acesso, permitindo a autenticação e autorização de usuários para acessar os serviços. Além disso, é crucial utilizar protocolos de criptografia, como HTTPS e SSL/TLS, para assegurar a segurança da comunicação entre os serviços.

Outra prática significativa para garantir a segurança em arquiteturas de microsserviços envolve o uso de ferramentas de segurança, como firewalls, sistemas de detecção e prevenção de intrusões (IDS/IPS) e sistemas de monitoramento de segurança. Kim *et al.* (2016) salientam a importância de implementar essas ferramentas em cada camada da arquitetura de microsserviços, abrangendo a camada de rede, a camada de aplicação e a camada de dados. Ao fazê-lo, é possível monitorar o tráfego de dados e identificar possíveis ameaças à segurança.

Além disso, a implementação de uma política de segurança e a conscientização da equipe de desenvolvimento sobre as melhores práticas de segurança são fundamentais (NEWMAN, 2015). A equipe deve estar ciente dos riscos e das técnicas de proteção, garantindo que os microsserviços sejam projetados e desenvolvidos com a segurança em mente desde o início.

2.3.6 Gerenciamento Eficiente de APIs em Arquiteturas de microsserviços: Adotando o Versionamento Apropriado

O versionamento de APIs é uma prática essencial na manutenção e evolução de sistemas baseados em microsserviços. À medida que as APIs evoluem, é importante garantir que as mudanças não afetem negativamente os consumidores dessas APIs (WILDE; PAUTASSO, 2011). O versionamento de APIs permite que os desenvolvedores introduzam novas funcionalidades e corrijam problemas sem

quebrar os sistemas existentes, mantendo a compatibilidade com versões anteriores (WOOD *et al.*, 2016).

Existem várias abordagens para o versionamento de APIs, cada uma com suas vantagens e desvantagens. Algumas das abordagens mais comuns incluem o versionamento na URL, o versionamento no cabeçalho de aceitação e o versionamento semântico (SemVer) (WILDE; PAUTASSO, 2011; PRESTON-WERNER, 2013). O versionamento na URL envolve incluir o número da versão diretamente na URL da API, facilitando a identificação da versão em uso.

O versionamento no cabeçalho de aceitação, por outro lado, move a informação da versão para o cabeçalho HTTP, mantendo a URL limpa e garantindo que os recursos possam ser versionados individualmente. O SemVer estabelece um esquema de numeração baseado em três componentes: *major*, *minor* e *patch*, que indicam mudanças na API que podem ser incompatíveis, compatíveis ou correções de bugs, respectivamente (PRESTON-WERNER, 2013).

Uma das melhores práticas no versionamento de APIs é adotar uma estratégia de versionamento que seja fácil de entender e usar pelos consumidores da API e que minimize a possibilidade de erros (RICHARDSON; AMUNDSEN, 2013). Além disso, é importante comunicar claramente as mudanças nas APIs aos consumidores e fornecer documentação atualizada e acessível (RICHARDSON; AMUNDSEN, 2013). Também é crucial considerar o suporte a versões antigas das APIs e estabelecer um processo para descontinuar versões antigas de forma ordenada e transparente (WILDE; PAUTASSO, 2011).

Em resumo, o versionamento de APIs desempenha um papel crítico na manutenção e evolução de sistemas de microsserviços. Ao adotar uma estratégia de versionamento apropriada e seguir as melhores práticas, os desenvolvedores podem garantir que as mudanças nas APIs sejam gerenciadas de forma eficiente, minimizando o impacto nos consumidores e mantendo a estabilidade e confiabilidade do sistema como um todo.

2.4 Implementação bem-sucedida de uma arquitetura de microsserviços

Para uma implementação bem-sucedida de uma arquitetura de microsserviços, é crucial considerar vários aspectos. Um dos primeiros passos é identificar os serviços e suas responsabilidades, garantindo a modularidade e a escalabilidade do sistema (RICHARDSON, 2018). A delimitação clara das responsabilidades dos serviços permite uma divisão mais eficiente do trabalho entre as equipes e facilita a manutenção e evolução do sistema.

Outro aspecto relevante é a seleção das tecnologias e ferramentas a serem empregadas na implementação dos serviços. Uma avaliação criteriosa das opções disponíveis deve ser realizada, levando em conta fatores como desempenho, escalabilidade, segurança e facilidade de manutenção (LEWIS; FOWLER, 2014). A escolha adequada de tecnologias e ferramentas pode contribuir para o sucesso da arquitetura de microsserviços e a satisfação das necessidades dos usuários.

Adicionalmente, é fundamental adotar uma abordagem ágil e iterativa no desenvolvimento dos serviços, permitindo a adição gradual de novas funcionalidades e a identificação e correção rápidas de problemas (WOLFF, 2016). Essa abordagem também favorece a adaptação às mudanças nos requisitos do negócio e a entrega contínua de valor aos usuários.

Por fim, é importante ressaltar que a implementação bem-sucedida de uma arquitetura de microsserviços demanda planejamento cuidadoso e uma equipe altamente capacitada. Investir em treinamento e desenvolvimento dos profissionais, além de adotar boas práticas de gerenciamento de projetos e comunicação entre as equipes, é essencial (RICHARDSON, 2018).

Com base nesses autores, pode-se afirmar que a implementação de uma arquitetura de microsserviços é um processo complexo que exige atenção a diversos aspectos. Identificar corretamente os serviços e suas responsabilidades, escolher as tecnologias e ferramentas apropriadas, adotar uma abordagem ágil e iterativa no desenvolvimento, monitorar os serviços em tempo real e investir no treinamento e

capacitação da equipe são práticas essenciais para garantir a modularidade, escalabilidade, segurança e qualidade do sistema.

Além disso, elas permitem a adição de novas funcionalidades de maneira flexível e sem comprometer a integridade do sistema como um todo. Ressalta-se a importância do planejamento cuidadoso e da presença de uma equipe altamente capacitada e bem gerenciada no sucesso da implementação de uma arquitetura de microsserviços.

2.4.1 Monitoramento: ferramentas e técnicas para monitorar serviços de microsserviços

O monitoramento de serviços em uma arquitetura de microsserviços é uma atividade crítica para garantir a confiabilidade e a disponibilidade dos sistemas distribuídos. Segundo Calcote e Butcher (2019), o uso de ferramentas de monitoramento é essencial para observar o comportamento dos serviços em tempo real, coletar métricas de desempenho e diagnosticar falhas em diferentes camadas da infraestrutura. Além disso, a utilização de técnicas como *tracing*, *logging* e agregação de logs pode ajudar a entender a causa raiz dos problemas e facilitar a tomada de decisões para melhorias futuras.

Para implementar uma estratégia de monitoramento efetiva em uma arquitetura de microsserviços, é importante escolher as ferramentas certas para cada necessidade. Segundo Richardson (2018), o uso de um serviço de descoberta de serviços, como o Consul ou o Eureka, pode ajudar a rastrear a localização dos microsserviços em tempo de execução. Já a ferramenta Prometheus é recomendada para coletar e armazenar métricas, enquanto o Grafana pode ser utilizado para visualizar essas métricas em dashboards customizados.

Além das ferramentas, é importante ter em mente que o monitoramento deve ser integrado ao processo de desenvolvimento e operações dos microsserviços. De acordo com Angrist e Pischke (2014), a definição de indicadores de performance e a criação de testes de carga são exemplos de práticas que podem ajudar a identificar gargalos e pontos críticos do sistema antes mesmo de entrar em produção. Sayfan

(2018) ainda ressalta que a utilização de práticas de monitoramento em tempo real, como a implementação de alertas e a análise de logs, pode ajudar a garantir a escalabilidade e a resiliência dos microsserviços em cenários de alta demanda.

2.4.2 Testes de microsserviços

O teste de microsserviços é uma parte crucial do processo de desenvolvimento e manutenção dessas arquiteturas, garantindo que os serviços individuais e o sistema como um todo funcionem de maneira confiável e eficiente. Lewis e Fowler (2014) destacam a importância de testar microsserviços, dada a complexidade e a natureza distribuída dessas arquiteturas. Eles abordam os testes de unidade, testes de integração e testes de contrato, que são essenciais para garantir o bom funcionamento dos microsserviços.

Newman (2015) detalha várias abordagens para testar microsserviços, incluindo o uso de ferramentas como JUnit e Mockito, que são comumente usadas para escrever e executar testes de unidade e de integração. Essas ferramentas ajudam a verificar se os microsserviços funcionam corretamente, tanto isoladamente quanto em conjunto com outros serviços, contribuindo para a resiliência e escalabilidade da arquitetura.

Além disso, Newman (2015) destaca a importância do teste de contrato, especialmente no contexto de microsserviços, onde a comunicação entre diferentes serviços pode ser um ponto crítico. O Pact é uma ferramenta mencionada por Newman, que facilita a realização de testes de contrato, garantindo que os microsserviços interajam corretamente entre si e cumpram suas obrigações contratuais.

Dentro do escopo de testes de microsserviços, Richardson (2018) reitera a necessidade de testar os componentes individualmente, bem como em conjunto. Ele afirma que, além dos testes de unidade e de integração, é fundamental garantir que o sistema como um todo seja submetido a testes de ponta a ponta e testes de carga, para garantir a estabilidade e a escalabilidade da arquitetura em condições reais de uso.

Em resumo, os testes de microsserviços envolvem uma variedade de estratégias e ferramentas, como testes de unidade, testes de integração, testes de contrato e ferramentas como JUnit, Mockito e Pact. A aplicação adequada dessas práticas e ferramentas é essencial para garantir a qualidade e a confiabilidade dos microsserviços, promovendo a resiliência e escalabilidade da arquitetura. Ao seguir as orientações e práticas recomendadas por especialistas como Fowler, Lewis, Newman e Richardson, os desenvolvedores podem garantir que os microsserviços sejam bem testados e estejam prontos para operar em ambientes de produção.

2.5 Modelos de negócio que podem se beneficiar da arquitetura de microsserviços

Certos modelos de negócios podem se beneficiar significativamente da adoção de arquiteturas de microsserviços. Como afirmam Newman (2015) e Wolff (2018), microsserviços podem oferecer maior flexibilidade e escalabilidade em comparação com as arquiteturas monolíticas tradicionais. Por exemplo, empresas que oferecem serviços on-line, como plataformas de e-commerce, podem se beneficiar da separação em serviços independentes, cada um focado em uma função específica, permitindo atualizações e manutenções mais eficientes e evitando problemas de escalabilidade.

Além disso, a arquitetura de microsserviços pode ser vantajosa para empresas que lidam com grande quantidade de dados, como as de análise de dados, como aponta Indrasiri e Siriwardena (2018). Ao dividir um sistema em serviços menores e independentes, é possível escalar e processar grandes quantidades de dados de forma mais eficiente, permitindo a realização de análises mais rápidas e precisas.

Empresas que desejam integrar serviços de diferentes fontes também podem se beneficiar da arquitetura de microsserviços, como destaca a Daya *et al*. (2015). A integração de diferentes serviços pode ser feita de forma mais eficiente com a adoção de microsserviços, onde cada serviço é responsável por uma tarefa específica e pode ser facilmente substituído ou atualizado.

A adoção de microsserviços requer uma mudança significativa na cultura e no processo de desenvolvimento de software. Indrasiri e Siriwardena (2018) oferecem um guia prático para a implementação de arquitetura de microsserviços, incluindo as melhores práticas para projetar, desenvolver e implantar serviços independentes. O livro destaca as ferramentas e tecnologias que podem ser usadas para criar arquiteturas de microsserviços, bem como os desafios que as empresas podem enfrentar ao fazer essa transição.

Por fim, as empresas de tecnologia da informação também podem se beneficiar da arquitetura de microsserviços. Segundo Balalaie, Heydarnoori e Jamshidi (2016), a arquitetura de microsserviços pode ajudar a simplificar a gestão de aplicações complexas, permitindo que diferentes equipes possam trabalhar de forma independente e evoluir os serviços de forma mais rápida e eficiente. Além disso, a modularidade dos serviços pode permitir a criação de soluções mais customizadas para diferentes clientes e mercados.

2.5.1 Modelo de negócio Netflix

A Netflix foi fundada em 1998 como uma empresa de aluguel de filmes em mídia física. Os clientes podiam alugar filmes que eram enviados para seus endereços via correio. Nos anos seguintes, a empresa ganhou popularidade graças ao seu inovador modelo de negócios, que oferecia aluguel ilimitado sem taxas de atraso, devolução ou entrega. A partir de 2008, a Netflix começou a oferecer aos seus assinantes a opção de assistir filmes por streaming na internet.

Entre 2009 e 2010, a Netflix iniciou a migração de seu sistema para a nuvem. A decisão foi tomada devido ao tempo e aos altos custos associados à construção de data centers dedicados. A Amazon Web Services foi escolhida para hospedar sua infraestrutura, cobrando pelos serviços no mês seguinte ao uso, permitindo que a Netflix concentrasse seus investimentos na aquisição de novos conteúdos e nas despesas para entregá-los (BRAY, 2015).

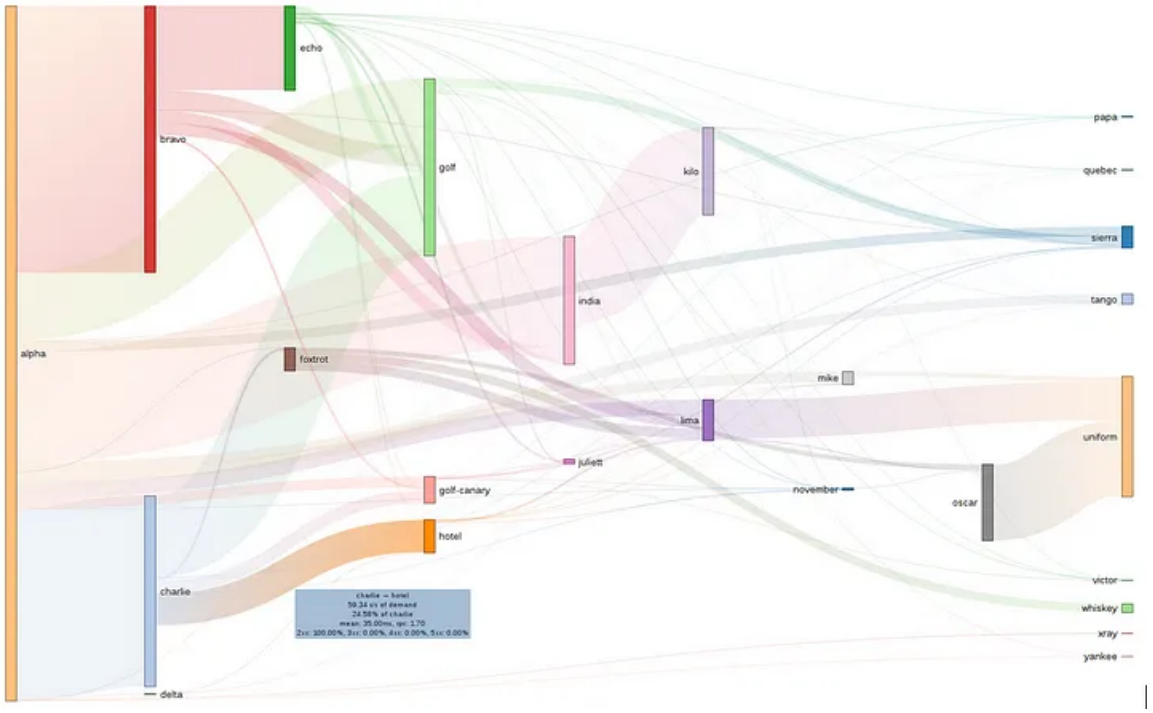
A transição para a nuvem ocorreu gradualmente. O primeiro componente da Netflix a ser executado na nuvem foi um serviço de preenchimento automático.

Quando os usuários começavam a digitar parte de uma palavra, o site sugeriria opções com base no texto inserido. Quando esse serviço foi implementado, todo o site ainda era hospedado em um data center dedicado. Embora fosse uma tecnologia simples, foi de grande importância para a equipe, pois os ensinou a implantar um sistema na nuvem, conectá-lo a um balanceador de carga e utilizar todas as ferramentas necessárias para isso. Apesar de ser um projeto pequeno, levou um mês para ser implementado com sucesso (BRAY, 2015).

A partir dessa experiência, a empresa começou a decompor seu sistema em diversos serviços. A arquitetura evoluiu de um único WebApp (.war) em 2008 para centenas de pequenos serviços em 2012. Inicialmente, a Netflix não se referia à sua nova arquitetura como microsserviços. Termos como "nativo na nuvem" ou "SOA de granularidade fina" eram usados. O termo microsserviços acabou sendo adotado pela Netflix após ser sugerido por membros da equipe da ThoughtWorks (BRAY, 2015).

O gráfico de dependência mostrado na Figura 16 ilustra a relação entre alguns dos microsserviços da Netflix. Ele apresenta alguns dos serviços utilizados pela empresa, como se comunicam entre si e quanto tempo gastam na comunicação.

Figura 16 - Relacionamento entre microsserviços



Fonte: Netflix (2015).

A Netflix tornou-se pioneira no desenvolvimento e implementação de novas arquiteturas e tecnologias de nuvem para operar em uma escala massiva - uma escala que testa os limites de várias tecnologias. Isso os levou a desenvolver muitas ferramentas próprias. Grande parte dessas tecnologias é disponibilizada ao público por meio do Netflix OSS (Netflix Open Source Software Center). O desafio não é apenas oferecer funcionalidades e gerenciar seu grande número de instâncias, mas também fornecer insights rápidos e acionáveis para uma arquitetura baseada em microsserviços de grande escala.

Ao longo dos anos, a Netflix se consolidou como um exemplo de sucesso na adoção de arquiteturas de microsserviços e tecnologias em nuvem. A empresa enfrentou vários desafios na transição, como a necessidade de reestruturar seu sistema e aprender a gerenciar e monitorar sua nova arquitetura distribuída. No entanto, a Netflix superou esses desafios e colheu os benefícios, incluindo maior escalabilidade, resiliência e flexibilidade (BRAY, 2015).

A experiência da Netflix com microsserviços e tecnologias em nuvem serve como um importante estudo de caso para outras organizações que desejam adotar

abordagens semelhantes. As lições aprendidas e as ferramentas desenvolvidas pela Netflix podem ser aplicadas em outros contextos e projetos, permitindo que outras empresas também aproveitem os benefícios das arquiteturas de microsserviços e da computação em nuvem.

2.5.2 Modelo de negócio Amazon

A Amazon, assim como a Netflix, é outra empresa de sucesso na adoção de arquiteturas de microsserviços. A migração da Amazon de uma arquitetura monolítica para microsserviços começou no início dos anos 2000, quando a empresa enfrentava problemas de escalabilidade e complexidade crescente de seus sistemas (O'HANLON, 2006). A mudança para microsserviços permitiu à Amazon escalar de forma eficiente e melhorar a resiliência de seus sistemas.

Werner Vogels, CTO da Amazon, descreve a importância da adoção de microsserviços para a empresa em seu artigo "A conversation with Werner Vogels" (2006). De acordo com O'Hanlon, a Amazon começou a desmembrar seu monólito em serviços menores e independentes, o que possibilitou uma melhor separação de responsabilidades e facilitou a colaboração entre equipes. Essa abordagem também permitiu a adoção de diferentes tecnologias e abordagens de desenvolvimento para cada serviço, melhorando a capacidade de inovação e a velocidade de entrega de novos recursos (O'HANLON, 2006).

Outro aspecto fundamental na abordagem de microsserviços da Amazon é a utilização de APIs bem definidas para a comunicação entre os serviços. Essas APIs promovem o desacoplamento entre os serviços e permitem que as equipes trabalhem de forma mais independente. De acordo com Bray (2015) em seu artigo "Microservices at Amazon", a Amazon segue uma política interna denominada "API Mandate", que exige que todas as equipes se comuniquem por meio de APIs, o que garante uma maior integração e modularidade dos sistemas.

Além disso, a Amazon utiliza sua própria plataforma de computação em nuvem, a Amazon Web Services (AWS), para implementar e gerenciar seus microsserviços. A AWS oferece uma variedade de serviços e ferramentas que facilitam a implantação, o monitoramento e a escalabilidade dos microsserviços,

como o Elastic Beanstalk, o Lambda e o EC2 Container Service (NEWMAN, 2015). Essa infraestrutura em nuvem permite que a Amazon aproveite os benefícios da computação em nuvem, como a elasticidade, a disponibilidade e a eficiência de custos.

A Amazon também utiliza práticas de DevOps e integração contínua/entrega contínua (CI/CD) para garantir a qualidade e a rápida entrega de seus microsserviços. De acordo com Jez Humble em "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" (HUMBLE; FARLEY, 2011), essas práticas ajudam a reduzir o tempo entre a concepção de novas funcionalidades e sua disponibilização aos usuários finais, ao mesmo tempo em que mantêm a qualidade e a estabilidade dos sistemas.

Em resumo, a Amazon é um exemplo de sucesso na adoção de arquiteturas de microsserviços. A empresa conseguiu superar desafios de escalabilidade e complexidade, adotando uma abordagem baseada em serviços menores e independentes, comunicando-se por meio de APIs bem definidas e utilizando a plataforma AWS para implantação e gerenciamento. As práticas de DevOps e CI/CD também desempenham um papel crucial na rápida entrega e na qualidade dos serviços da Amazon.

3 TRABALHOS RELACIONADOS

No capítulo anterior, exploramos os principais conceitos e ferramentas que fundamentam nosso estudo sobre arquiteturas de microsserviços e suas melhores práticas. Neste capítulo, examinaremos trabalhos relacionados ao desenvolvimento e implementação de sistemas baseados em microsserviços.

O objetivo é comparar as diferentes abordagens adotadas nesses trabalhos, identificando assim diferenças e pontos em comum, projetando possíveis resultados e verificando as estratégias mais eficientes. Essa análise nos permitirá estabelecer o melhor método para a realização do nosso estudo sobre arquiteturas de microsserviços e as melhores práticas para garantir sua implementação bem-sucedida.

3.1 Microservices: yesterday, today, and tomorrow

Em seu trabalho intitulado “Microservices: yesterday, today, and tomorrow” Dragoni *et al.* (2017) exploraram os desafios e benefícios associados à adoção da arquitetura de microsserviços em uma variedade de casos de uso. O objetivo principal do estudo foi analisar e avaliar os benefícios e desafios enfrentados pelas organizações ao adotar essa abordagem de arquitetura, bem como identificar padrões emergentes e práticas recomendadas para abordar esses desafios.

Diferentemente do nosso trabalho, que se concentra nos aspectos práticos e técnicos da implementação de microsserviços, o estudo focou mais nos aspectos organizacionais e gerenciais relacionados à adoção dessa arquitetura. No entanto, os resultados obtidos em seu trabalho fornecem uma base sólida para comparação e complementação ao nosso estudo.

Em seu estudo constataram que a adoção da arquitetura de microsserviços pode trazer benefícios significativos em termos de escalabilidade, flexibilidade e resiliência. No entanto, também identificaram desafios relacionados à complexidade na gestão e coordenação de microsserviços, bem como à segurança e ao

gerenciamento de dados distribuídos. Esses resultados são semelhantes aos encontrados em nosso trabalho, onde também discutimos a importância de abordar tais desafios para aproveitar ao máximo os benefícios oferecidos pelos microsserviços.

Um aspecto positivo do trabalho é a análise aprofundada dos fatores organizacionais e gerenciais que afetam a adoção da arquitetura de microsserviços. No entanto, o estudo não aborda detalhadamente os aspectos técnicos e práticos da implementação e teste de microsserviços, que é um foco importante do nosso trabalho. Ao considerar os resultados e conclusões, podemos aprimorar nosso estudo ao abordar os aspectos organizacionais e gerenciais mencionados em seu trabalho, enquanto fornecemos uma análise mais detalhada das soluções técnicas e práticas para enfrentar os desafios relacionados à arquitetura de microsserviços.

Ao utilizar as informações de seu trabalho, nosso estudo se beneficia ao abranger uma perspectiva mais ampla, combinando aspectos organizacionais, gerenciais e técnicos na análise dos desafios e benefícios da arquitetura de microsserviços. Ao mesmo tempo, nosso trabalho se diferencia ao fornecer uma abordagem mais prática para enfrentar os desafios técnicos, como testes e segurança, que são essenciais para a construção de sistemas escaláveis e resilientes.

Dessa forma, a contribuição do trabalho para o nosso estudo é dupla: complementa nossas descobertas ao trazer uma visão mais completa dos aspectos organizacionais e gerenciais envolvidos na adoção da arquitetura de microsserviços e nos permite posicionar nosso trabalho de forma mais precisa no contexto da pesquisa atual no campo dos microsserviços. Essa combinação de perspectivas fornece uma base sólida para o desenvolvimento de soluções eficientes e eficazes no âmbito da arquitetura de microsserviços.

3.2 From monolith to microservices

No trabalho desenvolvido por Aronen (2020), denominado “From monolith to microservices”, teve como principal questão examinar a migração de uma organização que busca alternativas para sua arquitetura monolítica e que já realizou a migração de algumas aplicações para microsserviços. O objetivo deste trabalho é avaliar como as teorias selecionadas na literatura funcionam na prática.

A conclusão do estudo mostra que a arquitetura atual na organização está mais próxima da Arquitetura Orientada a Serviços (SOA) do que da arquitetura de microsserviços. No entanto, os problemas causados pelo Enterprise Service Bus (ESB) e o acoplamento apertado dos serviços têm influenciado a transição para a arquitetura de microsserviços.

O estudo destaca que, em comparação com as aplicações monolíticas, os microsserviços são mais fáceis de manter, pois a funcionalidade de um único microsserviço é limitada e mais fácil de entender. Por outro lado, encontrar a causa de um problema é muito mais difícil em aplicações baseadas em microsserviços, devido à sua natureza distribuída e complexa.

Os resultados deste estudo fornecem uma visão prática das dificuldades encontradas durante a migração de uma aplicação monolítica para microsserviços. Um aspecto positivo destacado é a automação do desenvolvimento e das operações (DevOps), que é essencial no estilo de arquitetura de microsserviços e permite entregas contínuas. No entanto, a complexidade aumentada e os desafios na identificação e resolução de problemas são aspectos negativos que devem ser considerados ao se avaliar a migração para a arquitetura de microsserviços.

Neste trabalho, os autores demonstram que a migração de uma arquitetura monolítica para microsserviços pode ser benéfica no longo prazo, mas também destacam os desafios adicionais apresentados pela natureza distribuída e complexa dos microsserviços. Comparando com o nosso trabalho, que também busca analisar

os benefícios e desafios da implementação bem-sucedida de uma arquitetura de microsserviços, este estudo de caso oferece insights valiosos sobre os aspectos práticos e reais da migração e pode ser usado como um exemplo a ser considerado em nosso estudo.

3.3 Adoption Of The Microservice Architecture

Em seu estudo denominado “Adoption Of The Microservice Architecture”, Ndungu (2019) analisou a adoção da arquitetura de microsserviços, destacando os desafios enfrentados pelos arquitetos de sistemas e software no atual cenário tecnológico em rápida evolução. O estudo apresenta a arquitetura de microsserviços como uma solução inovadora para lidar com a complexidade do crescimento do código-fonte dos sistemas e discute suas características e atributos de qualidade, como acoplamento solto, alta coesão, resiliência e escalabilidade.

O trabalho destaca que a adoção da arquitetura de microsserviços é mais desafiadora do que outras abordagens de arquitetura, mas traz benefícios comerciais significativos, como um tempo de lançamento no mercado mais rápido. O objetivo principal deste estudo é fornecer uma compreensão detalhada dos conceitos e benefícios dos microsserviços, enfocando as estratégias de implementação e as melhores abordagens para adotar essa arquitetura. A pesquisa incentiva uma abordagem orientada a objetivos para a adoção de microsserviços, permitindo que as empresas e organizações obtenham o máximo valor comercial possível.

No trabalho, são discutidas várias maneiras de adotar a arquitetura de microsserviços e identificar os próprios microsserviços. O estudo conclui que não há uma maneira correta de implementar microsserviços; a arquitetura apenas fornece diretrizes para o projeto de sistemas. O sucesso na implementação depende da capacidade da organização de atingir seus objetivos e obter valor comercial com a arquitetura de microsserviços.

O trabalho enfatiza a importância da autonomia na arquitetura de microsserviços, explicando que os serviços devem lidar com apenas uma tarefa e executá-la excepcionalmente bem. Os microsserviços devem ser implantados de forma independente, e as equipes responsáveis pelo desenvolvimento e manutenção dos serviços devem ser distintas e independentes, com a liberdade de escolher as ferramentas e linguagens de programação que desejarem.

Outro aspecto importante abordado é a necessidade de cada microsserviço ter seu próprio armazenamento de dados e implementar seu próprio protocolo de comunicação, garantindo que os serviços não precisem de coordenação externa para se comunicarem entre si. A arquitetura de microsserviços também ajuda a alcançar atributos de qualidade desejáveis, como reusabilidade, modificabilidade, escalabilidade e resiliência.

O trabalho também aborda os desafios associados à adoção da arquitetura de microsserviços, enfatizando a importância de analisar cuidadosamente se os benefícios superam os custos e se a adoção dessa arquitetura trará resultados significativos a longo prazo. O estudo sugere que, ao decidir adotar a arquitetura de microsserviços, é crucial desenvolver um roteiro para o processo de adoção, considerando a estrutura e a cultura da organização, bem como os objetivos estratégicos da empresa. Isso inclui levar em conta a Lei de Conway, que afirma que o software reflete a estrutura de comunicação da organização. Para uma implementação bem-sucedida da arquitetura de microsserviços, é essencial que a organização considere seus níveis de comando e a estrutura de comunicação.

Ao estabelecer os limites do sistema e extrair os serviços, o trabalho sugere o uso da abordagem Domain Driven Design (DDD) para estabelecer um vocabulário de domínio e dividir os microsserviços de acordo com esses termos. A subdivisão adicional é fundamental para determinar contextos limitados e construir microsserviços que executem apenas uma função.

O estudo apresenta várias formas de adotar a arquitetura de microsserviços, permitindo que os arquitetos decidam se constroem um novo sistema de

microsserviços do zero ou se decompõem um sistema monolítico existente. São explicadas diferentes estratégias para decompor uma aplicação existente e as possíveis abordagens para adotar microsserviços, fornecendo estratégias práticas para a decomposição de uma aplicação monolítica.

Em suma, este trabalho aborda os desafios das aplicações monolíticas tradicionais e oferece soluções baseadas na adoção adequada da arquitetura de microsserviços. Ele fornece um caso a favor dos microsserviços, destacando suas características, vantagens e atributos de qualidade, enquanto também apresenta um caso contra os microsserviços, citando suas complexidades e custos.

A decisão de adotar a arquitetura de microsserviços deve ser baseada em uma análise cuidadosa dos prós e contras. Além disso, o trabalho demonstra os fatores essenciais que devem ser considerados caso uma empresa decida adotar a arquitetura de microsserviços e ilustra as possíveis abordagens e estratégias para a adoção bem-sucedida dos microsserviços.

Este trabalho é relevante para a nossa pesquisa, pois fornece informações valiosas sobre a adoção da arquitetura de microsserviços e destaca os benefícios e desafios associados à sua implementação. Ele oferece uma visão abrangente das melhores práticas e estratégias para adotar essa arquitetura, o que pode ser útil para comparar com as abordagens propostas em nosso estudo. Além disso, ao destacar os aspectos positivos e negativos da arquitetura de microsserviços, este trabalho também nos ajuda a entender como nosso próprio trabalho pode abordar e superar as limitações e desafios encontrados na adoção de microsserviços.

3.4 Investigating Quality Attributes and Best Practices of Microservices Architectures

No trabalho "Investigating Quality Attributes and Best Practices of Microservices Architectures", publicado por Faizan Zafar em 2022, o autor realiza uma análise aprofundada dos atributos de qualidade e das melhores práticas

relacionadas às arquiteturas de microservices. Este estudo é fundamental para compreender como os atributos de qualidade são afetados pelas práticas e decisões de projeto em tais arquiteturas e, também, para identificar as melhores práticas que podem ser adotadas para equilibrar os diferentes atributos de qualidade.

O estudo inicia-se com uma revisão da literatura sobre microservices, abordando os principais conceitos, características e benefícios dessa abordagem arquitetônica. O autor destaca que os *microservices* têm sido cada vez mais adotados por empresas e desenvolvedores, devido à sua capacidade de proporcionar escalabilidade, flexibilidade e facilidade de manutenção, entre outros benefícios.

Em seguida, analisa os atributos de qualidade associados às arquiteturas de microservices, incluindo desempenho, confiabilidade, segurança, manutenibilidade e usabilidade. O autor explica que, embora os microservices possam melhorar alguns desses atributos, também podem apresentar desafios em outros, como a complexidade de gerenciamento e a necessidade de lidar com questões de segurança em um ambiente distribuído.

Para identificar as melhores práticas que podem ser adotadas para otimizar os atributos de qualidade em arquiteturas de microservices, realiza uma análise sistemática da literatura, selecionando estudos de caso relevantes e artigos científicos que abordam esse tema. O autor analisa cada prática identificada, avaliando sua eficácia e aplicabilidade em diferentes cenários e contextos.

Com base nos resultados da análise, propõe um conjunto de melhores práticas para a arquitetura de microservices, incluindo:

1. Definir claramente os limites e responsabilidades de cada microservice;
2. Implementar padrões de comunicação eficientes entre os microservices;
3. Garantir a segurança dos dados e das comunicações em um ambiente distribuído;
4. Adotar práticas de DevOps e automação para facilitar a implantação e manutenção dos microservices;

5. Monitorar e ajustar continuamente os microservices para garantir a qualidade e o desempenho.

O trabalho de contribui para o campo de estudo dos microservices ao oferecer uma visão abrangente dos atributos de qualidade e melhores práticas relacionadas a essa arquitetura. Entretanto, o estudo não aborda especificamente a escalabilidade e adaptação da granularidade, temas que são centrais no contexto do nosso trabalho.

Em conclusão, o trabalho "Investigating Quality Attributes and Best Practices of Microservices Architectures" de Faizan Zafar (2022) é uma referência valiosa para pesquisadores e profissionais interessados em compreender e otimizar os atributos de qualidade em arquiteturas de microservices.

Embora não se concentre especificamente nos aspectos de escalabilidade e adaptação da granularidade, oferece insights úteis sobre as melhores práticas que podem ser aplicadas para garantir uma melhor qualidade geral dos microservices. Além disso, ao identificar e analisar as práticas que afetam diretamente os atributos de qualidade, o estudo de fornece uma base sólida para a compreensão das complexidades envolvidas na concepção e implementação de arquiteturas de microservices eficientes e eficazes.

Ao comparar o trabalho com o nosso estudo, podemos identificar algumas diferenças e pontos de interseção. Primeiramente, o foco de nosso trabalho é a análise sistemática da escalabilidade e adaptação da granularidade dos microservices, enquanto Zafar se concentra nos atributos de qualidade e melhores práticas. No entanto, é importante notar que os atributos de qualidade abordados por Zafar são também relevantes para a nossa pesquisa, uma vez que a escalabilidade e a granularidade afetam diretamente o desempenho, a confiabilidade e a manutenibilidade dos microservices.

Nesse sentido, os resultados apresentados podem ser utilizados como base para a comparação com o nosso trabalho, especialmente no que diz respeito às práticas que impactam os atributos de qualidade. Por exemplo, podemos analisar

como as práticas relacionadas à definição dos limites e responsabilidades dos microservices e à implementação de padrões de comunicação eficientes entre eles afetam a escalabilidade e a granularidade em diferentes cenários.

Em termos de pontos positivos do trabalho, destacamos a abordagem sistemática adotada pelo autor para identificar e analisar as melhores práticas, bem como a contribuição para o campo dos microservices, fornecendo insights valiosos sobre os atributos de qualidade e suas implicações. Quanto aos pontos negativos, podemos mencionar a falta de enfoque específico na escalabilidade e adaptação da granularidade, aspectos que são abordados de forma mais detalhada e aprofundada em nosso trabalho.

Assim, ao integrar as descobertas com as nossas próprias análises e resultados, esperamos enriquecer ainda mais o conhecimento no campo dos microservices e fornecer recomendações mais robustas e abrangentes para os profissionais e pesquisadores envolvidos no desenvolvimento e implantação de arquiteturas baseadas em microservices.

3.5 Systematic scalability analysis for microservices granularity adaptation design decisions

No artigo intitulado "Systematic scalability analysis for microservices granularity adaptation design decisions", Hassan, Bahsoon e Buyya (2021) abordam a importância da análise de escalabilidade na tomada de decisões de adaptação de granularidade para arquiteturas de microservices. O objetivo principal deste trabalho é fornecer uma abordagem sistemática para auxiliar na tomada de decisões informadas relacionadas à granularidade e escalabilidade de microservices, considerando as demandas e as restrições do ambiente.

O trabalho apresenta duas contribuições principais. Primeiro, os autores desenvolvem um catálogo de dimensões e métricas específicas para microservices relacionadas à escalabilidade, que é baseado em um estudo de mapeamento

sistemático da literatura sobre adaptação de granularidade em microservices. Este catálogo visa ajudar os arquitetos de software a identificar as dimensões e métricas relevantes que devem ser consideradas ao tomar decisões de adaptação de granularidade.

Em segundo lugar, os autores aplicam a análise de metas e obstáculos de escalabilidade no contexto da adaptação de granularidade de microservices. Esta análise é baseada no modelo KAOS (Keep All Objectives Satisfied) de modelagem orientada a metas. O objetivo dessa abordagem é ajudar os arquitetos de software a identificar e analisar sistematicamente os objetivos importantes para a escalabilidade de uma arquitetura de microservices e os obstáculos que podem impedir a satisfação desses objetivos.

A principal diferença entre este trabalho e outros estudos relacionados é a abordagem sistemática e orientada a metas proposta pelos autores, que permite uma análise mais detalhada e contextualizada das decisões de adaptação de granularidade em microservices. Além disso, a combinação do catálogo de dimensões e métricas com a análise de metas e obstáculos de escalabilidade oferece uma estrutura abrangente para abordar a complexidade das arquiteturas de microservices em cenários do mundo real.

Os resultados deste trabalho podem ser utilizados para comparação com outras abordagens, uma vez que os autores aplicam suas contribuições a um exemplo hipotético de arquitetura de microservices chamado Filmflix. A análise e discussão dos resultados mostram como a abordagem proposta leva a resultados mais informados do que uma avaliação ad hoc da escalabilidade.

Os pontos positivos do trabalho incluem a abordagem sistemática e orientada a metas, que permite uma análise aprofundada das decisões de adaptação de granularidade em microservices, e o catálogo de dimensões e métricas, que fornece uma base sólida para a identificação e avaliação das dimensões e métricas relevantes para a escalabilidade. Além disso, a aplicação da análise de metas e

obstáculos de escalabilidade no contexto de microservices é inovadora e promissora.

No entanto, há algumas limitações no trabalho. A abrangência e a aplicabilidade do catálogo de dimensões e métricas ainda precisam ser avaliadas em estudos futuros, e a metodologia pode ser aprimorada com a integração de outras dimensões arquitetônicas, como segurança, confiabilidade e desempenho. Além disso, a aplicação do trabalho em um único exemplo hipotético, o Filmflix, pode limitar a generalização dos resultados. Outros estudos de caso e cenários do mundo real devem ser explorados para validar e aprimorar a abordagem proposta.

Em nosso trabalho, pretendemos abordar algumas das limitações mencionadas, expandindo a aplicabilidade do catálogo de dimensões e métricas para outros cenários e avaliando sua eficácia em arquiteturas de microservices mais complexas. Além disso, nossa pesquisa visa integrar outras dimensões arquitetônicas, como segurança, confiabilidade e desempenho, proporcionando uma visão mais holística das decisões de adaptação de granularidade em microservices.

Em resumo, o trabalho "Systematic scalability analysis for microservices granularity adaptation design decisions" apresenta uma abordagem inovadora e sistemática para a tomada de decisões de adaptação de granularidade em arquiteturas de microservices, com ênfase na análise de escalabilidade.

A combinação do catálogo de dimensões e métricas com a análise de metas e obstáculos de escalabilidade oferece uma estrutura abrangente e promissora para enfrentar os desafios da escalabilidade em arquiteturas de microservices. No entanto, algumas limitações do trabalho, como a aplicabilidade do catálogo de dimensões e métricas e a necessidade de integrar outras dimensões arquitetônicas, fornecem oportunidades para futuras pesquisas e melhorias na área.

3.6 Comparativo entre os trabalhos relacionados

No Quadro 1 é realizado um comparativo que apresenta uma visão geral dos cinco trabalhos mencionados, incluindo os objetivos, metodologias, pontos positivos e negativos de cada um. Ele destaca as diferenças entre os trabalhos e pode ser usado como base para análise e comparação.

Quadro 1 – Comparativo dos trabalhos relacionados

Trabalho	**Objetivo**	**Metodologia**	**Pontos Positivos**	**Pontos Negativos**
1. Microservices: yesterday, today, and tomorrow	Analisar a evolução dos microservices, discutir os princípios e apresentar os desafios e oportunidades futuras.	Revisão bibliográfica e análise evolutiva.	Fornece uma visão abrangente da evolução dos microservices.	Falta de foco em aspectos técnicos e metodologias específicas.
2. From monolith to microservices	Abordar a migração de sistemas monolíticos para arquiteturas de microservices.	Estudo de caso e práticas recomendadas.	Discute a migração de sistemas monolíticos com profundidade.	Limitado a um estudo de caso; pode não ser generalizável.
3. Adoption of the Microservice Architecture	Investigar os fatores de adoção e as barreiras na transição para arquiteturas de microservices.	Pesquisa quantitativa e qualitativa.	Identifica fatores e barreiras para adoção de microservices.	Não aborda detalhadamente a implementação e práticas.
4. Investigating Quality Attributes and Best Practices of Microservices Architectures	Avaliar atributos de qualidade e práticas recomendadas para arquiteturas de microservices.	Revisão sistemática da literatura.	Foco em atributos de qualidade e práticas recomendadas.	Não se aprofunda em questões específicas de escalabilidade.
5. Systematic scalability analysis for	Propor uma abordagem sistemática	Catálogo de dimensões e métricas,	Abordagem sistemática e	Limitado em abordar outras

microservices granularity adaptation design decisions	para decisões de adaptação de granularidade em microservices.	análise de metas e obstáculos.	foco na escalabilidade.	dimensões arquitetônicas

Fonte: Do autor (2023).

O presente trabalho se baseia na análise da escalabilidade e adaptação da granularidade em arquiteturas de microservices, buscando propor soluções eficientes para enfrentar os desafios atuais no desenvolvimento de sistemas distribuídos. Uma semelhança importante é a abordagem sistemática adotada em "Systematic scalability analysis for microservices granularity adaptation design decisions", que também se concentra na escalabilidade e granularidade em microservices.

Como diferencial, este trabalho busca integrar várias dimensões arquitetônicas, tais como desempenho, segurança e tolerância a falhas, proporcionando uma abordagem mais abrangente e eficiente para a adaptação de granularidade. Além disso, a metodologia proposta enfatiza a importância da colaboração entre equipes multidisciplinares e o uso de métricas quantitativas para apoiar a tomada de decisões.

Entretanto, não encontramos trabalhos que abordem de forma integrada todas as dimensões arquitetônicas mencionadas e que considerem a colaboração entre diferentes especialistas como um elemento chave para o sucesso na adaptação da granularidade de microservices. Desta forma, nosso estudo se propõe a preencher essa lacuna na literatura e contribuir para o avanço do conhecimento no campo das arquiteturas de microservices.

4 MATERIAIS E MÉTODOS

Este segmento discute a abordagem metodológica e as técnicas de investigação adotadas para a realização deste estudo, detalhando os métodos empregados, bem como os propósitos específicos da pesquisa e os procedimentos técnicos correspondentes.

4.1 Pesquisa quanto aos métodos científicos

A metodologia científica aplicada nesta pesquisa envolve uma abordagem sistemática e estruturada para coletar, analisar e interpretar dados. O estudo utiliza uma combinação de métodos qualitativos e quantitativos, conforme sugerido por Creswell (2014), incluindo revisão da literatura e análise de documentos. Essa combinação de métodos permite uma compreensão mais profunda e abrangente do fenômeno estudado.

A escolha dos métodos científicos é orientada pela natureza do problema de pesquisa e pelos objetivos do estudo. De acordo com Gil (2022), a aplicação de métodos qualitativos e quantitativos de forma integrada pode enriquecer a análise e fornecer insights mais robustos sobre o objeto de estudo. Além disso, a combinação de diferentes métodos permite a triangulação dos dados, aumentando a confiabilidade dos resultados (CRESWELL, 2014).

Ao adotar essa abordagem metodológica, a pesquisa busca gerar conhecimentos relevantes e aplicáveis no contexto da implementação de arquiteturas de microsserviços, contribuindo para a identificação e disseminação de melhores práticas e aprimorando a compreensão dos atributos de qualidade associados a essa abordagem arquitetônica.

4.2 Pesquisa quanto ao modo de abordagem

A pesquisa adota uma abordagem quanti-qualitativa, que combina métodos quantitativos e qualitativos para investigar as características e melhores práticas associadas à arquitetura de microsserviços.

No método quantitativo, o foco da pesquisa foi a análise de dados obtidos a partir da experiência real dos usuários com a arquitetura de microsserviços, especialmente no que se refere a escalabilidade, uso de CPU e memória. Para isso, foram empregados testes práticos onde os usuários interagiram com o sistema sob condições controladas. Durante esses testes, métricas específicas, como a capacidade de escalonamento do sistema, a eficiência no uso da CPU e a gestão da memória, foram rigorosamente monitoradas e registradas.

Esses dados numéricos fornecem informações cruciais sobre o desempenho real do sistema em um ambiente de uso ativo, permitindo uma avaliação precisa de como a arquitetura de microsserviços se comporta na prática, não apenas em cenários teóricos ou simulados.

Por outro lado, no método qualitativo, foi aplicado um questionário direcionado a desenvolvedores, operadores e usuários finais que interagiram com a arquitetura de microsserviços. Este questionário incluiu perguntas abertas sobre suas experiências, percepções e desafios encontrados durante a implementação e operação dos microsserviços. As respostas forneceram insights valiosos sobre os aspectos práticos, como facilidade de uso, integração com outros sistemas e satisfação geral com a arquitetura. Esse feedback qualitativo foi crucial para entender como a arquitetura impacta os usuários em um nível mais subjetivo e pessoal.

A combinação dessas metodologias forneceu uma visão holística da arquitetura de microsserviços. Enquanto os testes quantitativos ofereceram dados objetivos sobre o desempenho e a escalabilidade, o questionário qualitativo revelou percepções e experiências dos usuários, destacando áreas de melhoria e sucesso. Essa abordagem integrada foi fundamental para compreender completamente os

aspectos técnicos e humanos da arquitetura de microsserviços, permitindo uma análise mais aprofundada e a identificação de oportunidades de otimização e inovação.

4.3 Pesquisa quanto aos fins da pesquisa

Este trabalho adota uma abordagem híbrida, combinando aspectos de pesquisas exploratórias, e bibliográficas. O foco exploratório se dedica a descortinar e entender profundamente as práticas de excelência e os atributos de qualidade na arquitetura de microsserviços, conforme delineado por Gil (2022). A abordagem exploratória é particularmente apropriada quando a literatura sobre o tema é escassa e há necessidade de uma exploração meticulosa do fenômeno.

A faceta descritiva do trabalho se esforça para elucidar as nuances e inter-relações entre os atributos de qualidade e as práticas consagradas na arquitetura de microsserviços. Seguindo a perspectiva de Gil (2022), a pesquisa descritiva oferece uma visualização organizada e minuciosa dos fenômenos em questão, facilitando a identificação de padrões emergentes e tendências.

Ao amalgamar os propósitos exploratório e descritivo, esta pesquisa proporciona uma análise robusta e bem contextualizada da arquitetura de microsserviços. Isso permite um entendimento mais aprofundado dos elementos que moldam o sucesso de sua implantação e manutenção. A integração destes objetivos de pesquisa amplifica o avanço do conhecimento no domínio, destacando lacunas na literatura existente e sugerindo diretrizes para a prática profissional.

Embora a vertente predominante desta pesquisa seja experimental, é imprescindível reconhecer a importância de uma descrição meticulosa dos processos e resultados. Tal descrição é crucial para garantir uma compreensão holística do fenômeno analisado.

4.4 Pesquisa quanto aos procedimentos técnicos

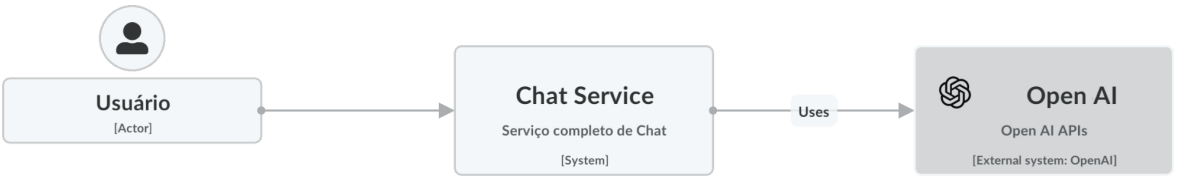
Dentro do escopo desta pesquisa, o principal objetivo foi desenvolver uma interface para o ChatGPT utilizando um microsserviço em GO, conforme ilustrado na

Figura 17. A decisão por essa tecnologia foi motivada por sua notável performance e confiabilidade, observadas em testes preliminares. Esta abordagem se mostrou eficaz, permitindo uma comunicação fluida e estável com o ChatGPT, potencializando suas capacidades e oferecendo uma solução robusta para os usuários.

O projeto englobou o desenvolvimento de um *Backend For Frontend* (BFF) e de um painel web, ambos construídos com Next JS, visando demonstrar a autenticação e autorização através do KeyCloak. Optou-se pelo Next JS devido à sua eficiência e flexibilidade na criação de aplicações, além de ser uma ferramenta com a qual já tenho familiaridade.

No que diz respeito à etapa de testes e análise dos resultados, foi planejada uma metodologia rigorosa para avaliar a eficácia e eficiência do microsserviço em GO desenvolvido. A intenção foi assegurar que o serviço cumpra todos os requisitos e casos de uso estabelecidos, identificando e corrigindo quaisquer problemas ou deficiências que possam surgir durante essa fase.

Figura 17 - Diagrama de contexto da aplicação



Fonte: Do Autor (2023).

Essas abordagens são complementadas com a pesquisa bibliográfica e o documental como principais métodos de investigação, o que permite uma análise abrangente e detalhada, fortalecendo a validade e a confiabilidade dos achados (CRESWELL, 2014; GIL, 2022; YIN, 2018).

4.5 Desenvolvimento do sistema

O sistema foi construído com base em uma arquitetura de microsserviços, onde cada serviço é dedicado a uma tarefa específica, como a comunicação com as APIs da OpenAI ou a gestão de chats, garantindo uma funcionalidade específica e otimizada.

Nas seções subsequentes, este documento detalha a arquitetura dos microsserviços, os fluxos de comunicação, e apresentará exemplos práticos da sua implementação e operação. A proposta é fornecer uma compreensão abrangente da eficiência e resiliência proporcionadas pela arquitetura de microsserviços neste projeto específico.

4.5.1 Ambientes de teste e desenvolvimento

No desenvolvimento de aplicações e sistemas, é essencial que diferentes ambientes sejam estabelecidos para permitir que equipes de desenvolvimento, testes e operações trabalhem de forma eficiente e independente. No contexto deste projeto, utilizamos Docker como ferramenta principal para criar e gerenciar esses ambientes, garantindo assim a consistência e a portabilidade da aplicação em diferentes estágios de desenvolvimento e produção.

O Docker é uma plataforma que permite que desenvolvedores criem, testem e implementem aplicações dentro de contêineres. Contêineres são ambientes isolados que contêm tudo o que uma aplicação precisa para ser executada: código, bibliotecas, dependências e assim por diante. Eles são leves e garantem que a aplicação funcione da mesma maneira, independentemente de onde seja executada (DOCKER, 2023).

Para o projeto, cada componente - o microsserviço em Go, o BFF e o Frontend em NextJS, e a solução de autenticação KeyCloak - foi encapsulado em seu próprio container Docker. Isso permite que desenvolvedores e testadores simulem o ambiente de produção em suas máquinas locais sem interferir uns com os outros.

Segue abaixo uma tabela que destaca as principais diferenças entre os ambientes de desenvolvimento e teste:

Quadro 2 - Comparação entre ambiente de desenvolvimento e teste

Critério	Ambiente de Desenvolvimento	Ambiente de Teste
Objetivo	Desenvolvimento e integração de novas funcionalidades.	Testar a funcionalidade e performance da aplicação.
Dados	Dados fictícios ou de amostra para simulação.	Dados de teste que imitam o ambiente de produção.
Atualizações	Frequentes, a cada nova funcionalidade ou correção.	Ocorre após o ambiente de desenvolvimento estar estável.
Acesso	Apenas para desenvolvedores.	Testadores, e em alguns casos, partes interessadas para UAT (Teste de Aceitação do Usuário).
Configuração do Contêiner	Configurado para depuração e desenvolvimento.	Configurado para simular o ambiente de produção o mais próximo possível.
Backup	Não é necessário.	Regularmente, dependendo da natureza dos testes.

Fonte: Do Autor (2023).

4.5.2 Definição do microsserviço proposto

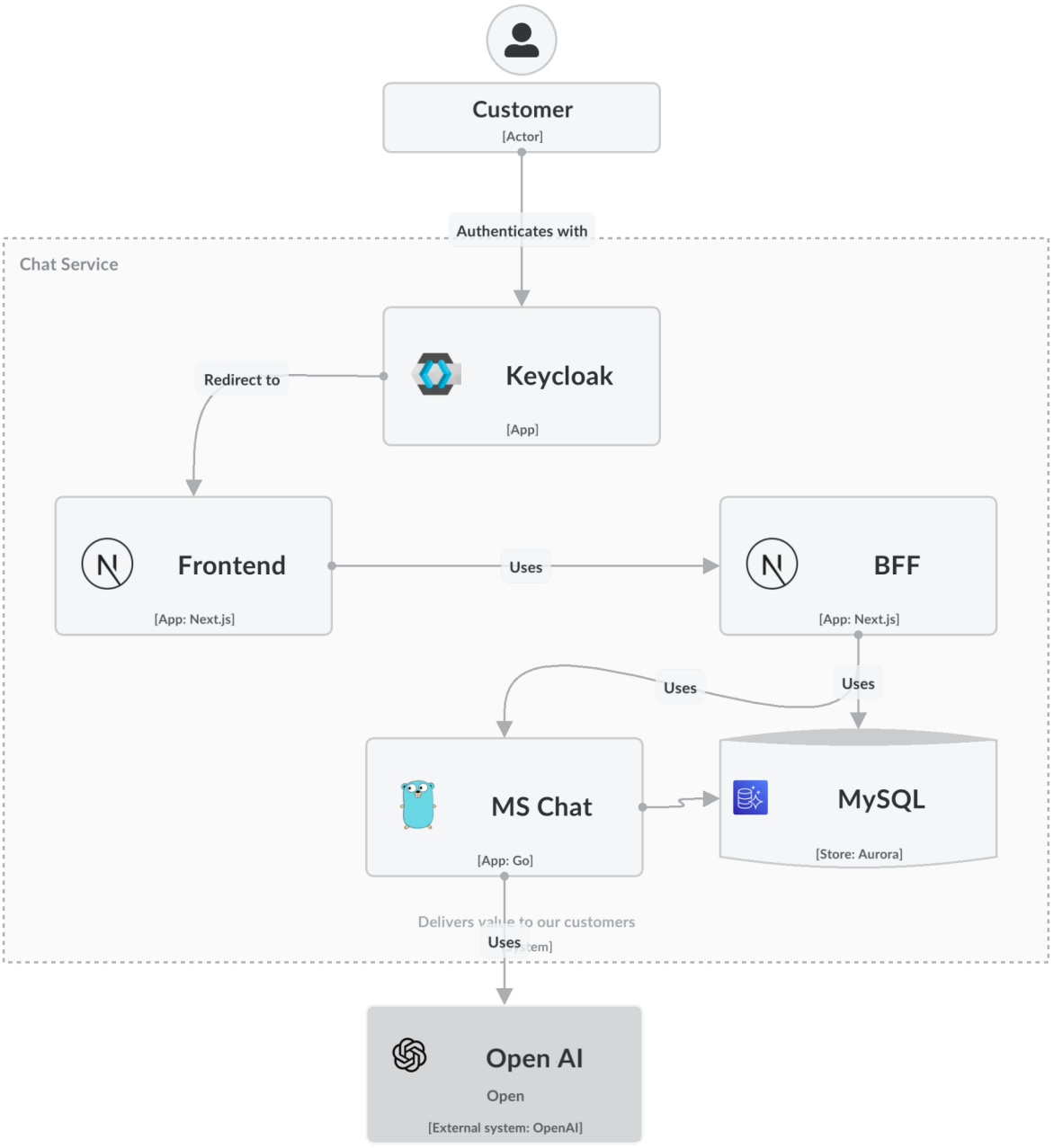
A criação de microsserviços é uma abordagem de design de arquitetura que segmenta uma aplicação em componentes menores e autônomos. Estes componentes, ou "microsserviços", são independentes entre si e se comunicam através de interfaces bem definidas, geralmente APIs. Neste projeto, o principal objetivo do microsserviço é intermediar a comunicação entre o Frontend e as APIs da OpenAI.

4.5.2.1 Visão geral da arquitetura

O microsserviço foi projetado para ser leve, escalável e altamente responsivo. Ele é responsável por gerenciar solicitações do Frontend, processá-las conforme necessário e se comunicar com a OpenAI.

Para uma visualização mais clara da arquitetura proposta, podemos observar o diagrama de container, exibido na Figura 18.

Figura 18 - Diagrama de container do MS Chat



Fonte: Do Autor (2023).

4.5.2.2 Casos de uso

O microsserviço foi desenvolvido com foco em atender a vários casos de uso relacionados à comunicação e processamento de solicitações entre o Frontend e a OpenAI:

1. Solicitação de Informação: Os usuários podem solicitar informações específicas que são recuperadas pela OpenAI através do microsserviço.
2. Processamento de Linguagem Natural: O microsserviço pode enviar textos do usuário para a OpenAI, que os analisa e retorna insights ou respostas relevantes.
3. Autenticação e Autorização: Embora a autenticação seja gerenciada pelo KeyCloak, o microsserviço valida as solicitações para garantir que sejam de usuários autenticados.

4.5.2.3 Requisitos

Os requisitos são as especificações detalhadas que o sistema deve cumprir. Eles estão categorizados em requisitos funcionais, que se referem a funções específicas do sistema, e requisitos não funcionais, que se referem a características de qualidade do sistema.

Quadro 3 - Requisitos funcionais

ID	Descrição Simplificada do Requisito Funcional	Prioridade
RF1	Processamento e transmissão de solicitações ao OpenAI.	Alto
RF2	Recebimento e encaminhamento de respostas do OpenAI.	Alto
RF3	Validação de solicitações de usuários autenticados.	Alto
RF4	Retorno de mensagens de erro ao Frontend em falhas.	Alto

Fonte: Do autor (2023).

Quadro 4 - Requisitos não funcionais

ID	Descrição Simplificada do Requisito Não Funcional	Prioridade
RNF1	Escalabilidade para muitos usuários simultâneos.	Alto
RNF2	Tempos de resposta abaixo de 500ms.	Alto
RNF3	Comunicações seguras com a OpenAI, usando protocolos criptografados.	Alto
RNF4	Disponibilidade mínima de 99,9% (Three-Nines).	Alto

Fonte: Do Autor (2023).

4.5.3 Desenvolvimento do Microsserviço em Go

A implementação de microsserviços em Go representa uma combinação de simplicidade, eficiência e alta performance, permitindo criar aplicações robustas e facilmente escaláveis.

4.5.3.1 Introdução ao Go e justificativa para sua escolha

Go, também conhecido como Golang, é uma linguagem de programação criada no Google, que combina a eficiência de linguagens compiladas, como C++, com a facilidade de uso de linguagens interpretadas, como Python. Suas principais características incluem:

Concorrência Inerente: Go foi projetado com suporte nativo para programação concorrente usando goroutines.

Eficiência: Sendo uma linguagem compilada, Go oferece um desempenho notável, especialmente importante para um microsserviço que precisa ser rápido e responsivo.

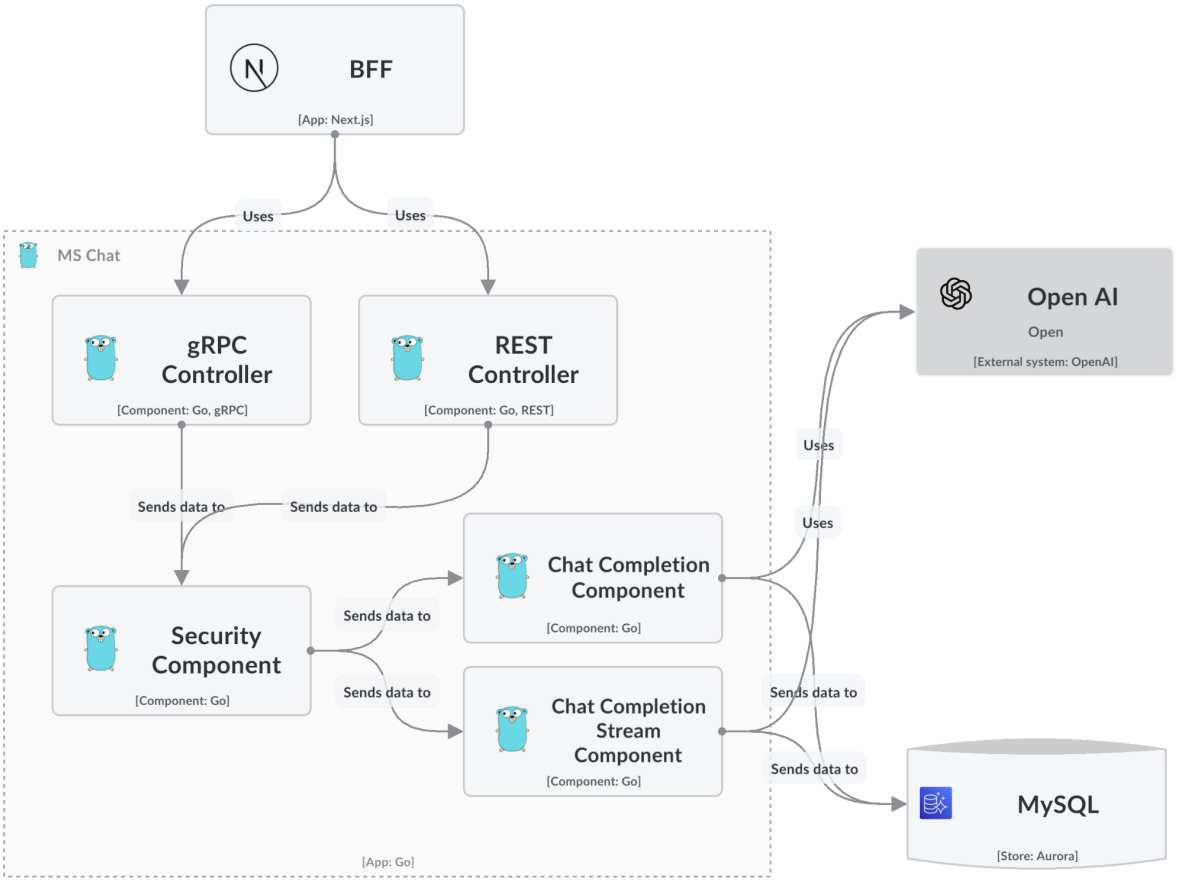
Simplicidade: A sintaxe clara e a falta de exceções tornam o Go fácil de ler e escrever.

A decisão de utilizar Go para o desenvolvimento do microsserviço baseou-se na necessidade de um sistema altamente responsivo, fácil de manter e que pode ser escalado com facilidade.

4.5.3.2 Estrutura e design do microserviço

O design do microsserviço foi concebido para ser modular e desacoplado, de forma que diferentes componentes possam funcionar independentemente, mas de forma integrada quando necessário, conforme Figura 19.

Figura 19 - Diagrama de componente do MS Chat



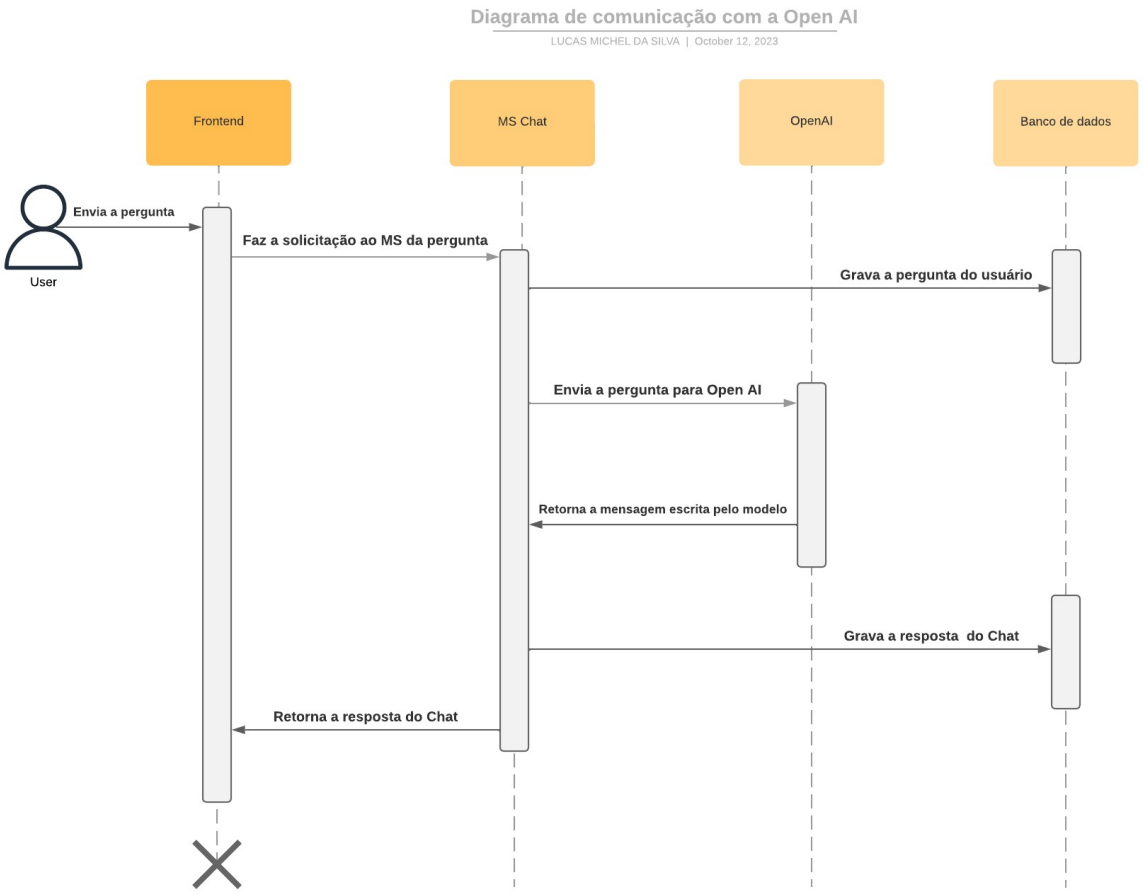
Fonte: Do Autor (2023).

Este diagrama ilustra a organização interna do microsserviço, demonstrando como as solicitações são processadas, quais componentes são responsáveis por diferentes tarefas e como a informação flui dentro do sistema.

4.5.3.3 Comunicação com as APIs da OpenAI

A comunicação entre o microsserviço e as APIs da OpenAI é essencial para a funcionalidade deste projeto. O microsserviço serve como intermediário, processando e encaminhando solicitações do frontend para a OpenAI e vice-versa.

Figura 20 - Diagrama de comunicação com a Open AI



Fonte: Do Autor (2023).

Este fluxo garante que as solicitações sejam processadas de maneira eficiente, e que as respostas da OpenAI sejam entregues ao usuário de forma rápida e precisa, tendo a responsabilidade de manter a continuidade nas conversas ao armazenar dados de chats e reenviar os contextos anteriores das interações. Para garantir a eficácia deste processo, o microsserviço avalia a contagem de tokens das conversas armazenadas. A OpenAI estabelece um limite máximo de tokens que o modelo pode processar de uma vez conforme o Quadro 2.

Quadro 5 - Listagem de modelos OpenAI

MODEL NAME	MAX TOKENS	TRAINING DATA
gpt-3.5-turbo	4,097 tokens	Up to Sep 2021
gpt-3.5-turbo-16k	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo-instruct	4,097 tokens	Up to Sep 2021
gpt-3.5-turbo-0613	4,097 tokens	Up to Sep 2021
gpt-3.5-turbo-16k-0613	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo-0301 (Legacy)	4,097 tokens	Up to Sep 2021
text-davinci-003 (Legacy)	4,097 tokens	Up to Jun 2021
text-davinci-002 (Legacy)	4,097 tokens	Up to Jun 2021

Fonte: OpenAI (2023).

Se uma conversa armazenada excede este limite, o microsserviço, aproveitando a eficiência do GO, adequadamente ajusta o contexto, removendo partes mais antigas até que a contagem de tokens esteja dentro do aceitável. Deste modo, assegura-se que a mensagem possa ser enviada sem comprometer a continuidade da conversa.

4.5.4 Desenvolvimento do BFF e Frontend em NextJS

NextJS, uma framework baseada em React, revolucionou a maneira como as aplicações da web são construídas, oferecendo versatilidade e um conjunto robusto de ferramentas para desenvolvimento.

4.5.4.1 Introdução ao NextJS e justificativa para sua escolha

NextJS é uma framework React com foco em produção e eficiência. Algumas de suas características mais marcantes incluem:

Renderização no lado do servidor (SSR): Essencial para SEO e para melhorar o tempo de carregamento inicial.

Roteamento: NextJS possui um sistema de roteamento embutido que facilita a criação de aplicações de página única (SPA).

Otimização de desempenho: A framework cuida automaticamente da divisão de código, o que torna o carregamento de páginas mais rápido.

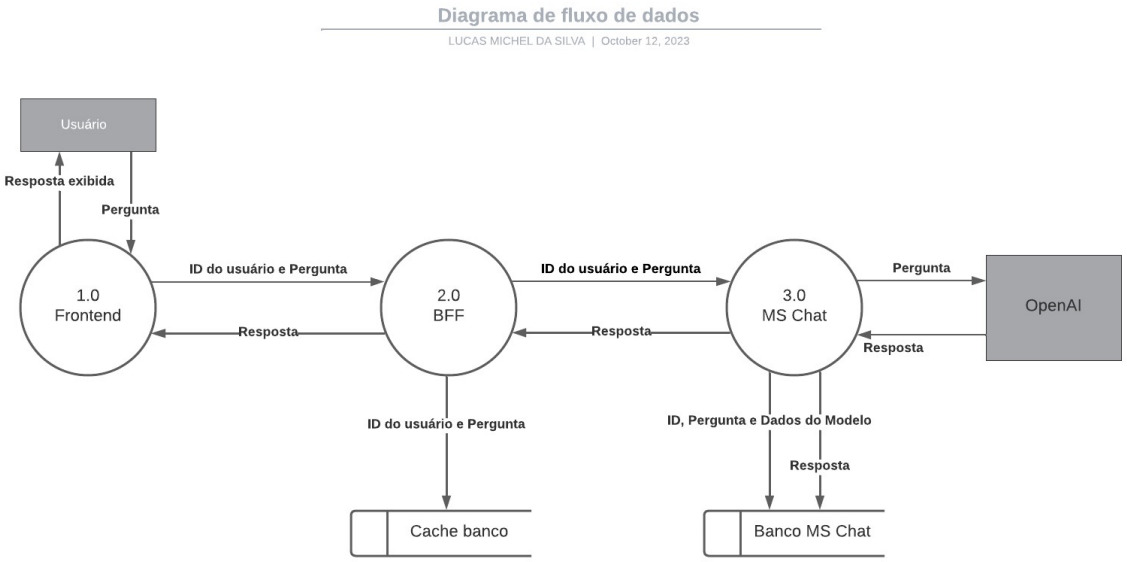
A decisão de utilizar NextJS foi motivada por sua capacidade de entregar conteúdo rapidamente, otimização para mecanismos de busca e sua integração suave com o React, permitindo uma experiência de desenvolvimento coesa.

4.5.4.2 Estrutura do BFF

O Backend For Frontend (BFF) age como uma camada intermediária entre o Frontend e os diversos serviços de backend, como o microsserviço. Ele consolida e otimiza as solicitações, garantindo que o Frontend receba os dados de forma eficiente.

Além disso, o BFF tem a capacidade de armazenar em cache os chats, assegurando uma resposta rápida e diminuindo a sobrecarga sobre o banco de dados. Para gerenciar esses dados armazenados, ele utiliza o Prisma, uma ferramenta ORM (Object Relational Mapping) moderna e eficiente, que facilita a interação com o banco de dados relacional MySQL. Esta integração assegura que as operações de CRUD (Create, Read, Update e Delete) sejam realizadas de forma otimizada e robusta.

Figura 21 - Diagrama de fluxo de dados



Fonte: Do Autor (2023).

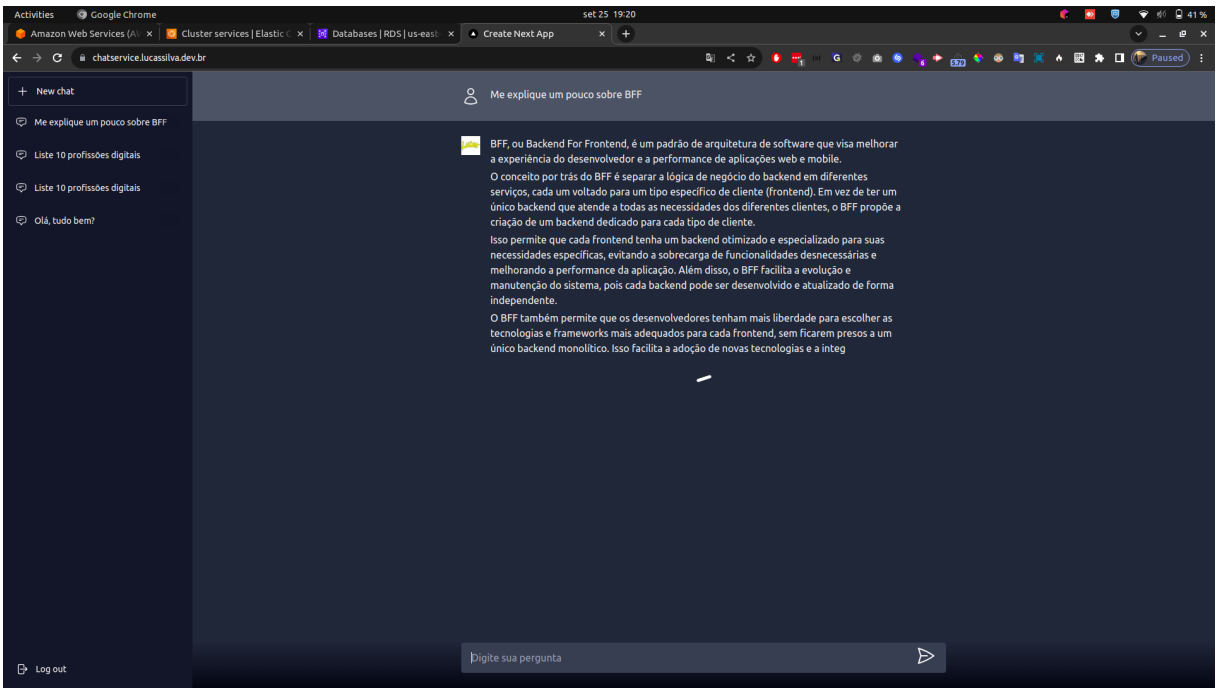
Este design facilita a manutenção, uma vez que as mudanças no backend ou no microsserviço não necessitam de alterações substanciais no Frontend, e vice-versa.

Quanto à comunicação, o BFF possui múltiplos endpoints. O protocolo gRPC, conhecido por sua alta performance e eficiência na transmissão de dados, é usado para a comunicação com o Microsserviço de Chat (MS Chat). Este canal garante a troca de informações em tempo real e de maneira confiável. Por outro lado, para funções como a listagem de chats e interações mais tradicionais, o BFF oferece endpoints HTTPS, proporcionando uma interface flexível e segura para os consumidores dessas APIs.

4.5.4.3 Design e funcionalidades do Frontend

O Frontend, construído com NextJS, foi projetado para ser intuitivo e eficiente, garantindo que os usuários tenham uma experiência fluida.

Figura 22 - Frontend do Chat



Fonte: Do Autor (2023).

O design foi orientado para garantir que os usuários possam interagir com a aplicação de maneira intuitiva, com ênfase em usabilidade e clareza.

Um dos principais atributos desse frontend é seu protocolo de autenticação. Cada usuário, antes de ter acesso a quaisquer recursos, é submetido a um processo de login, fundamentado pelo OpenID Connect. Este, um padrão moderno baseado no protocolo OAuth 2.0, não só assegura uma autenticação eficaz, mas também simplifica o gerenciamento de tokens, permitindo que o frontend obtenha informações do backend de forma segura e controlada.

4.5.5 Autenticação com KeyCloak

A autenticação é uma parte crucial de qualquer aplicação moderna. Ela garante que os usuários sejam quem dizem ser e que tenham acesso apenas aos recursos que lhes são permitidos. KeyCloak, uma solução open-source, surge como uma ferramenta robusta para gerenciar essa autenticação e autorização.

4.5.5.1 Visão geral e vantagens do KeyCloak

KeyCloak é uma solução de identidade e acesso que oferece recursos como Single Sign-On (SSO), autenticação de dois fatores e gerenciamento centralizado de políticas. Algumas das vantagens mais notáveis do KeyCloak incluem:

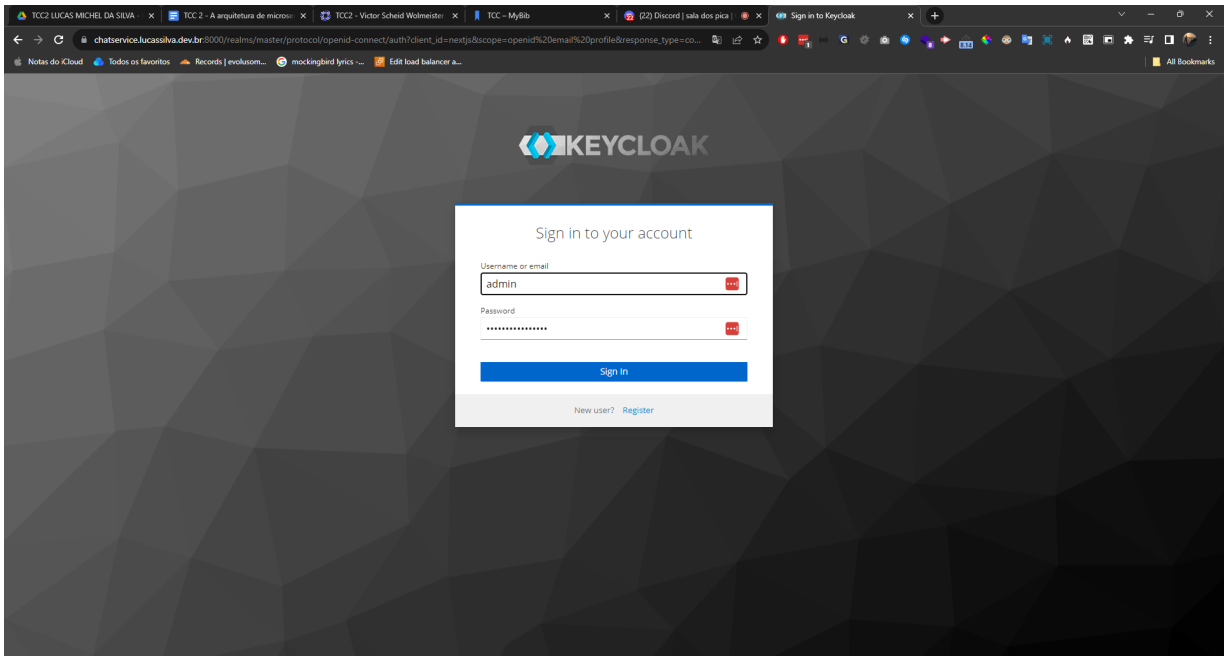
Configurabilidade: Permite personalizar fluxos de autenticação para atender a requisitos específicos.

Escalabilidade: KeyCloak pode ser facilmente escalado para atender a grandes volumes de usuários.

Integração: É altamente integrável com diferentes plataformas e aplicações, graças ao suporte a padrões como OpenID Connect, SAML 2.0 e OAuth 2.0.

Essas características tornam o KeyCloak uma opção ideal para projetos que requerem uma solução de autenticação segura e escalável.

Figura 23 - Tela de autenticação

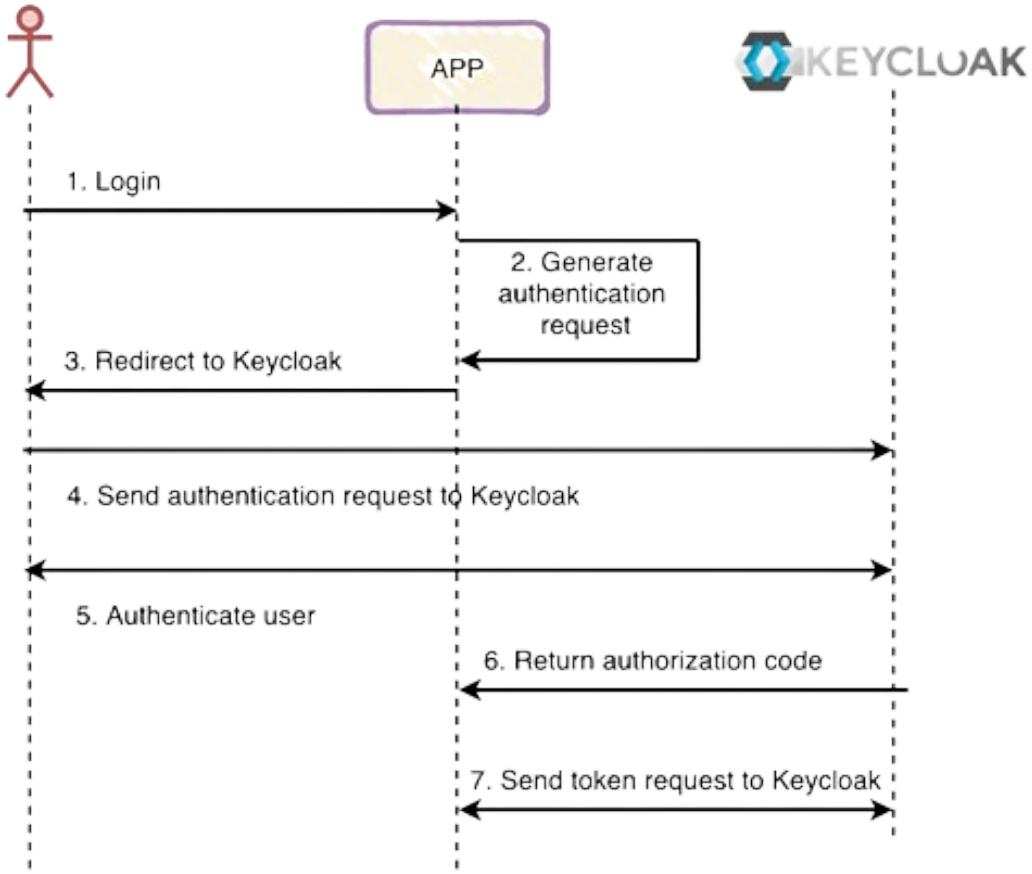


Fonte: Do Autor (2023).

4.5.5.2 Integração do Keycloak com os componentes

O processo de autenticação geralmente começa quando um usuário tenta acessar um recurso protegido. O Keycloak intervém neste ponto, garantindo que o usuário seja autenticado e autorizado antes de conceder acesso.

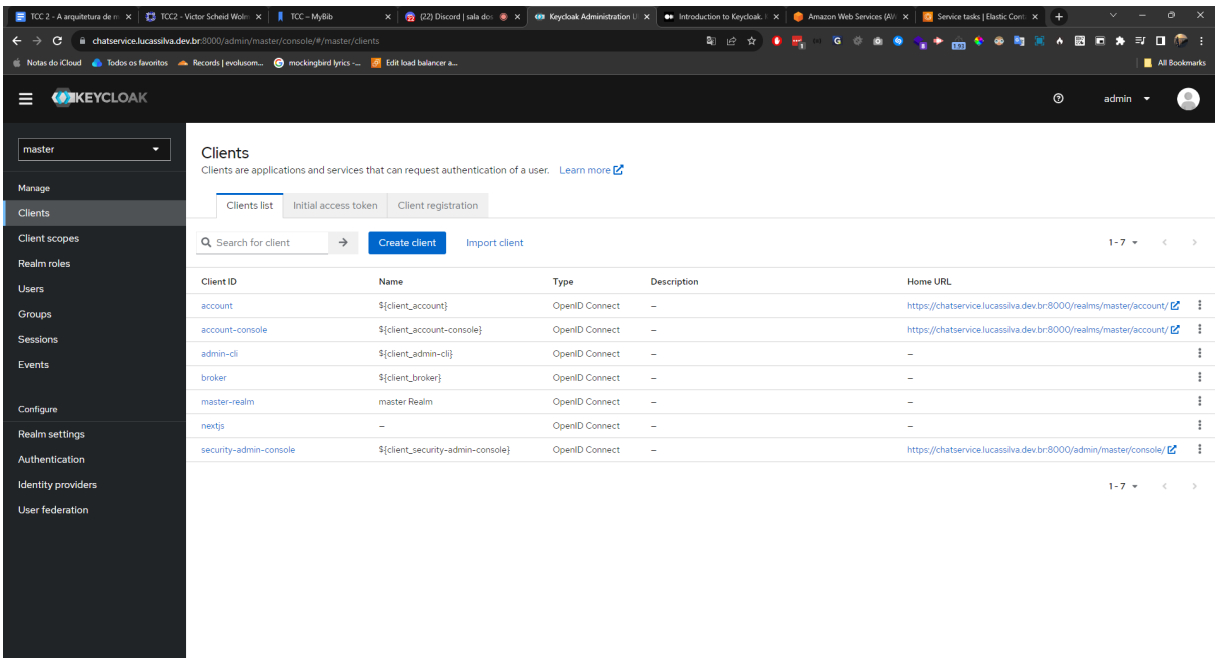
Figura 24 - Fluxo de autenticação com o Keycloak



Fonte: İleri (2022).

A integração do KeyCloak garante que a segurança seja uma preocupação primária, com cada componente do sistema desempenhando um papel específico no processo de autenticação e autorização.

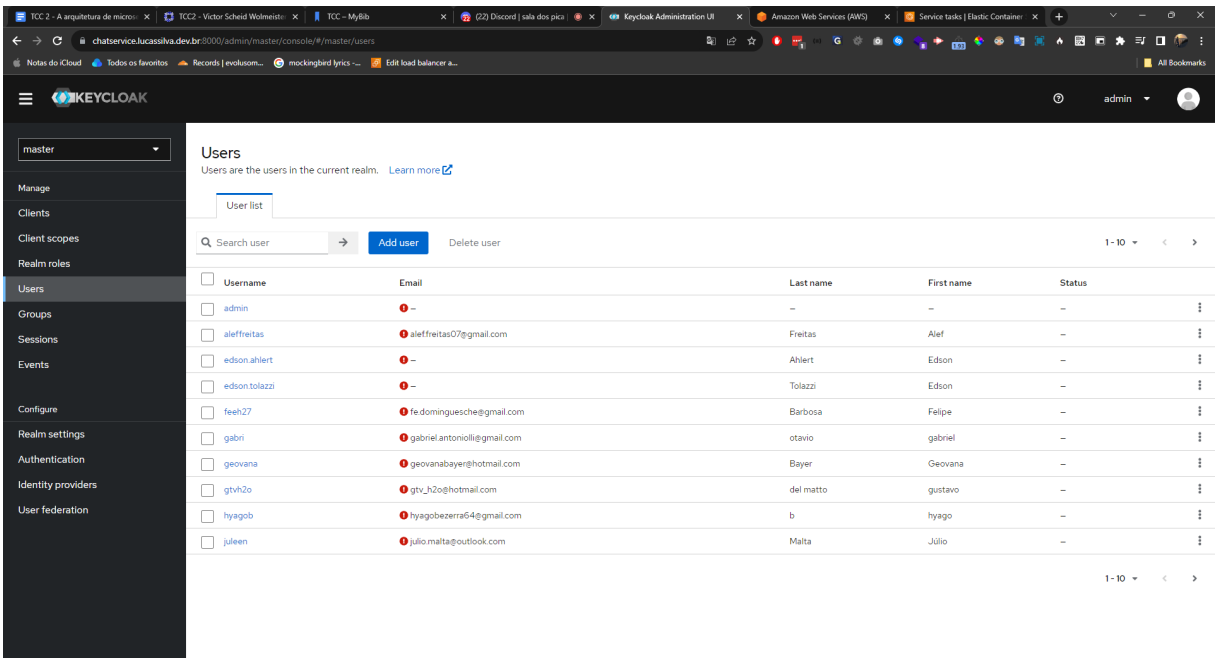
Figura 25 - Tela principal do Keycloak



Fonte: Do Autor (2023).

Uma das funcionalidades notáveis do Keycloak é sua capacidade de gerenciar de maneira centralizada o cadastro de usuários. Isso significa que, desde o momento do registro até a autenticação regular, é o Keycloak quem orquestra e valida esses processos, garantindo que sejam tão eficientes quanto seguros. Além disso, ele facilita a gestão de permissões, permitindo aos administradores definir com precisão quais recursos e informações um usuário ou grupo de usuários podem acessar.

Figura 26 - Tela de gerenciamento de usuários



Fonte: Do Autor (2023).

Em resumo, o Keycloak não é apenas um sistema de autenticação: é uma ferramenta poderosa que centraliza e simplifica a gestão de usuários e permissões, desempenhando um papel fundamental na arquitetura do sistema e garantindo uma experiência de usuário segura e fluida.

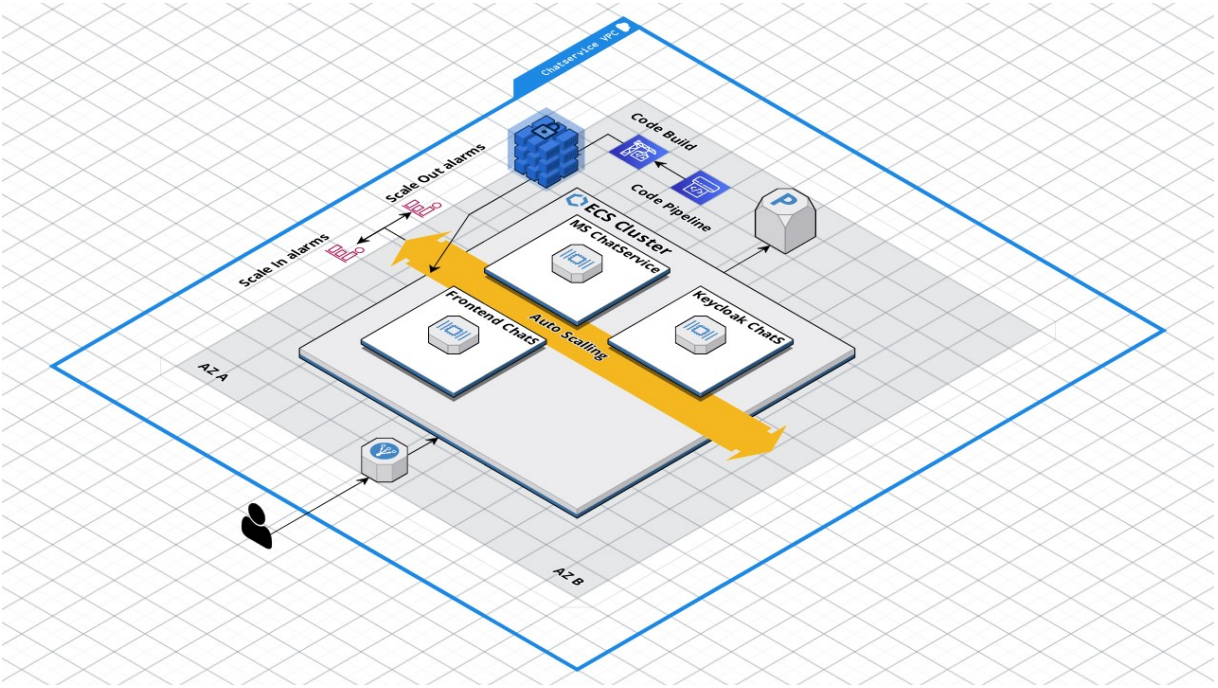
4.5.6 Implantação na AWS e automação com Terraform

A utilização de uma infraestrutura robusta e bem gerenciada é vital para a execução eficiente de qualquer aplicação moderna. A AWS oferece um leque de serviços que, aliados ao poder de automação do Terraform, garantem um ambiente otimizado, seguro e escalável.

4.5.6.1 Visão geral da infraestrutura na AWS

A AWS proporciona uma infraestrutura em nuvem resiliente, com um vasto portfólio de serviços que suportam diversas necessidades de um projeto.

Figura 27 - Visão geral infraestrutura



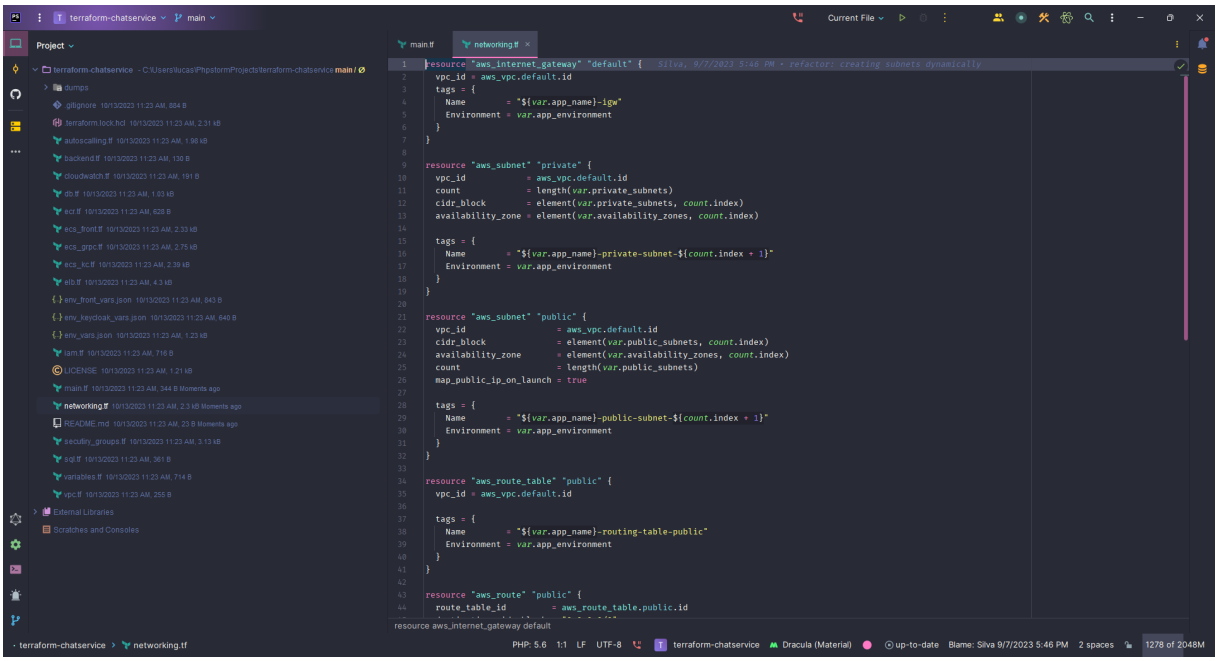
Fonte: Do Autor (2023).

A escolha por estes serviços foi motivada por suas capacidades individuais, que, quando integradas, resultam em um sistema harmonioso e altamente disponível.

4.5.6.2 Codificação da Infraestrutura com Terraform

Terraform, uma ferramenta de codificação de infraestrutura como código (IaC), foi utilizada para automatizar e gerenciar a implantação dos recursos na AWS.

Figura 28 - Criação da rede com Terraform



Fonte: Do Autor (2023).

Os recursos provisionados por meio do Terraform incluem:

Quadro 6 - Serviços de Armazenamento e Processamento

Tipo de Recurso	Especificações	Propósito
RDS MySQL	Versão 8.0.33	Armazenar cache do BFF, histórico de chats do MS Chat e autenticação do Keycloak.
ECS Cluster Serverless	0.25 vCPU, 256 MB RAM por task	Rodar os serviços: Frontend, MS Chat e Keycloak.
ECR	Não especificado	Armazenar imagens para os serviços.
ELB	Padrão	Distribuição de carga entre todas as services.

Fonte: Do Autor (2023).

Quadro 7 - Networking

Tipo de Recurso	Especificações	Propósito
Subnet Pública	Com IGW (Internet Gateway)	Conectar os recursos à Internet.
Subnet Privada	Com NAT Gateway	Permitir que recursos internos acessem a Internet.

Fonte: Do Autor (2023).

Quadro 8 - Políticas de Escalonamento

Tipo de Recurso	Especificações	Propósito
Auto Scaling Policy (Task)	Não especificado	Escalar cada task para cima e para baixo conforme a necessidade.

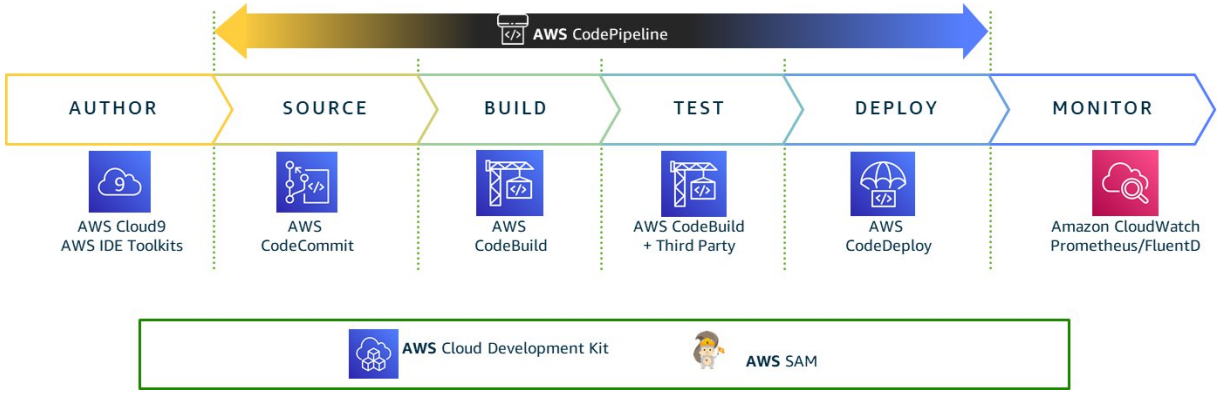
Fonte: Do Autor (2023).

Esta abordagem IaC garante consistência, reduzindo o risco de erros humanos e permitindo a replicação da infraestrutura em diferentes ambientes ou regiões.

4.5.6.3 Pipelines de CI/CD no Github

As pipelines de CI/CD facilitam a integração contínua e a entrega contínua, permitindo um fluxo de trabalho de desenvolvimento mais ágil e robusto.

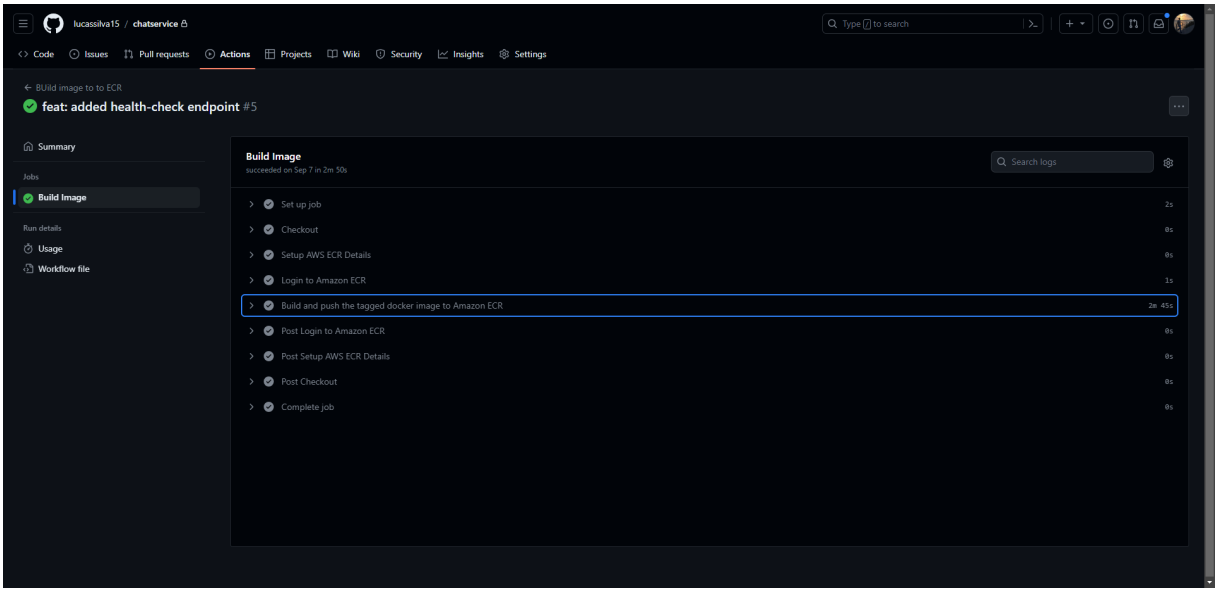
Figura 29 - Fluxo de CI/CD com ferramentas da AWS



Fonte: Do Autor (2023).

Os pipelines foram configurados no Github, um dos VCSs mais populares. Para uma visualização mais clara, conforme Figura 30.

Figura 30 - Steps da pipeline do Chatservice



Fonte: Do Autor (2023).

Através de pipelines de CI/CD estabelecidas, cada vez que uma alteração é feita no código-fonte, uma série de operações automatizadas são acionadas. Estas operações incluem testes, construção, e finalmente, o deploy da aplicação. O AWS CodeBuild é utilizado para compilar o código-fonte, executar testes e criar pacotes prontos para deploy, enquanto o AWS CodeDeploy é empregado para automatizar as implantações no ambiente, assegurando uma transição suave com mínimas interrupções.

Após a fase de construção, as imagens Docker geradas são armazenadas no Elastic Container Registry (ECR) da AWS. O ECR oferece um local seguro e escalável para armazenar e gerenciar estas imagens, garantindo que elas estejam prontamente disponíveis para implantação e que sejam mantidas de forma organizada e versionada.

Este fluxo garante que cada push ou pull request passe por verificações rigorosas antes de ser integrado ou implantado, garantindo qualidade e confiabilidade.

4.5.7 Conclusão do Desenvolvimento

O projeto enfrentou desafios e aproveitou oportunidades para criar uma solução eficaz e adaptável, cujos destaques são resumidos a seguir.

1. Ambiente de Teste e Desenvolvimento: Estabelecemos uma estrutura eficaz, utilizando Docker, para garantir consistência e reprodutibilidade em todos os ambientes. Isso permitiu uma transição suave entre desenvolvimento, teste e produção.

2. Definição do Microsserviço: A arquitetura foi meticulosamente planejada para aproveitar as vantagens do microsserviço, facilitando a escalabilidade e manutenção. Os casos de uso detalhados e os requisitos estabelecidos formaram a base para as etapas subsequentes de desenvolvimento.

3. Escolhas Tecnológicas: A adoção do Go para o microsserviço e do NextJS para o BFF e Frontend não foi aleatória. Essas tecnologias foram selecionadas por suas capacidades inerentes que se alinham com as necessidades do projeto.

4. Autenticação com KeyCloak: A segurança foi uma prioridade, e KeyCloak provou ser uma ferramenta valiosa para garantir que os usuários tenham uma experiência de autenticação segura e intuitiva.

5. Implantação na AWS e Automação com Terraform: A infraestrutura foi projetada para ser resiliente e escalável. Utilizando Terraform, conseguimos garantir uma provisão consistente de recursos na AWS e, com pipelines de CI/CD no Github, estabelecemos um fluxo de trabalho ágil e confiável.

Em conclusão, o processo de desenvolvimento foi uma jornada repleta de aprendizados e decisões importantes. Cada escolha, desde a seleção da tecnologia até a configuração da infraestrutura, foi feita para garantir que o projeto não apenas atendesse, mas superasse as expectativas. A seção de desenvolvimento ilustra a dedicação, a pesquisa e a inovação empregadas, e o resultado final é uma solução pronta para enfrentar os desafios do mundo real.

5 RESULTADO E DISCUSSÕES

Neste capítulo, apresentamos os resultados alcançados, seguidos de uma discussão. A análise consistiu na comparação dos resultados com estudos anteriores e na interpretação detalhada dos índices de desempenho obtidos. Também levamos em conta as respostas dos questionários aplicados. Além disso, discutimos os desafios encontrados durante a implementação da arquitetura de microsserviços.

5.7 Coleta de Métricas e Análise de Desempenho

Como parte da abordagem quantitativa para avaliar a eficácia e eficiência do sistema, optamos por ativar o cluster durante um período de três dias. Durante este tempo, foram monitoradas ativamente várias métricas-chave para entender o comportamento do sistema sob condições reais de operação.

Figura 31 - Visão geral de uso do cluster

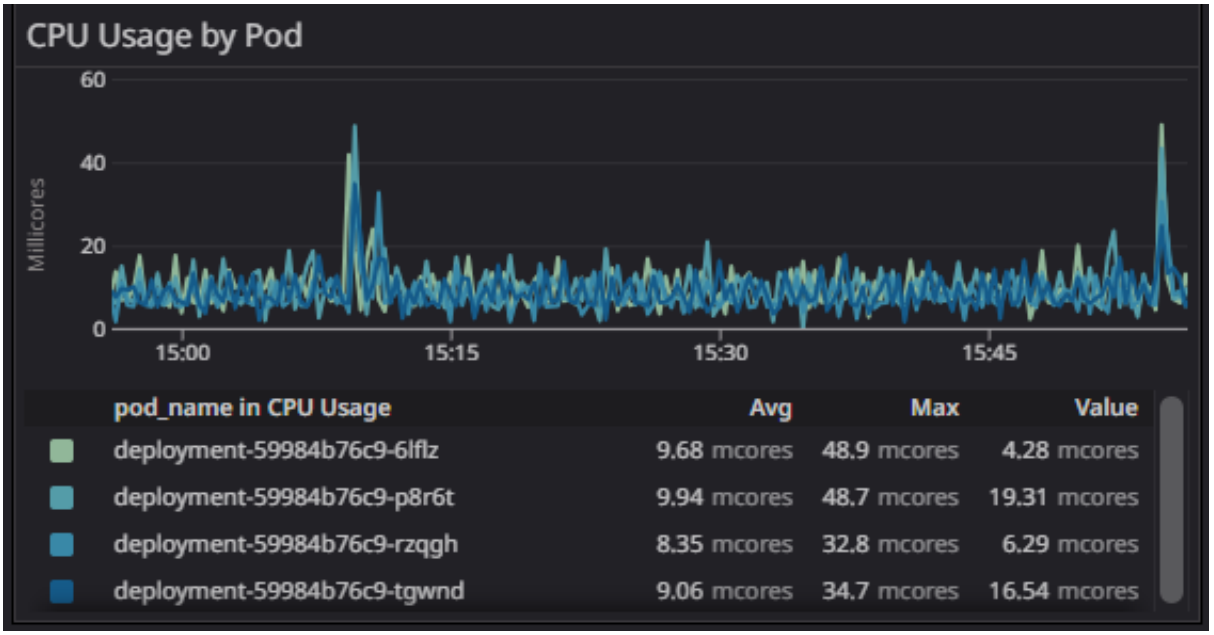
Fonte: Do Autor (2023).

É importante destacar que cada task foi configurada com 0.25vCPU e 256MB de RAM. Essa configuração foi escolhida para proporcionar um equilíbrio entre desempenho e eficiência de recursos, e as métricas observadas refletem a performance do sistema dentro desses parâmetros.

5.7.1 Uso da CPU por Serviço

Durante o período de monitoramento, observamos diferentes padrões de uso da CPU para cada um dos três serviços principais. O uso da CPU para o serviço chatservice-frontend, que consiste no Frontend em Next e BFF, mostrou-se relativamente baixo em comparação com os outros dois. Durante o processo de deploy, atingiu picos de 20% de utilização. No entanto, quando em funcionamento sob condições normais de uso, sua utilização oscilou entre 3% e 10%. Esse comportamento sugere uma estabilidade e eficiência notável para as operações frontend e BFF.

Figura 32 - Consumo de CPU do Frontend

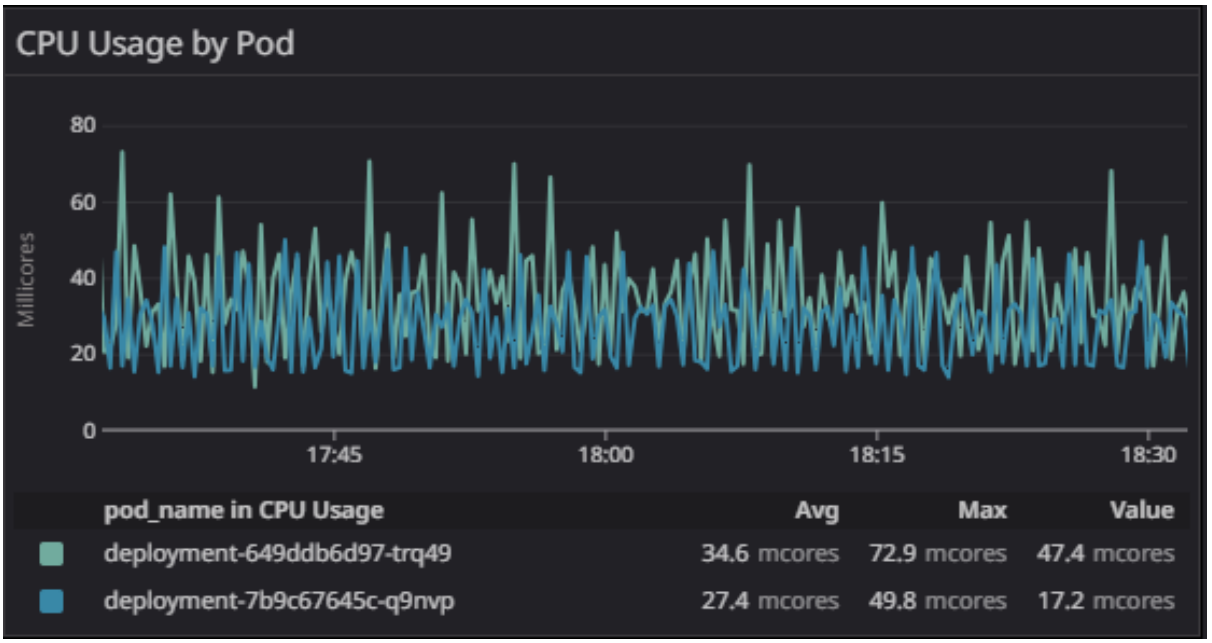


Fonte: Do Autor (2023).

Por outro lado, o serviço chatservice-keycloak, dedicado à autenticação, mostrou uma utilização inicial de CPU consideravelmente alta, chegando a picos de 80% durante certos processos. No entanto, uma vez completadas essas operações intensivas, a utilização da CPU se estabilizou em um intervalo de 20% a 45%. Tal comportamento pode indicar operações iniciais intensas, possivelmente

relacionadas à inicialização ou autenticação em massa, mas uma performance estável posteriormente.

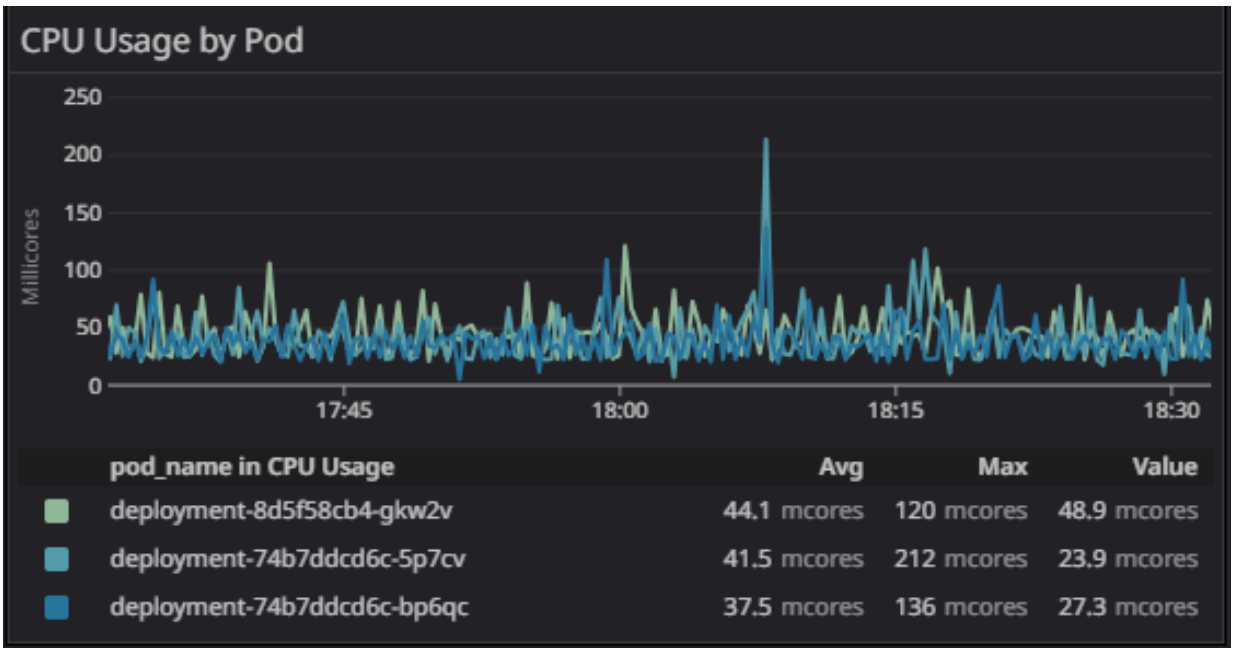
Figura 33 - Consumo de CPU do Keycloak



Fonte: Do Autor (2023).

Já o serviço chatservice, que é o MS Chat escrito em GO, também apresentou características semelhantes ao serviço de autenticação. Ele alcançou picos de 99% de utilização da CPU durante determinadas operações, particularmente durante o deploy. Após essa fase intensiva, a utilização se estabilizou em um intervalo mais moderado de 7% a 10%. Isso pode refletir a eficiência do código GO e a rapidez com que o serviço se adapta após inicializações ou mudanças.

Figura 34 - Consumo de CPU do MS Chat



Fonte: Do Autor (2023).

5.7.2. Escalonamento (Autoscaling)

Nosso sistema foi configurado com grupos de autoescalonamento para ajustar-se dinamicamente às demandas. Ao longo do período de observação, o sistema escalonou 3 vezes, aumentando a capacidade quando a demanda estava alta e reduzindo-a durante períodos de baixa atividade. Esse comportamento demonstra a capacidade do sistema de responder adequadamente às variações de carga.

Figura 35 - Aumento de Instâncias sobre uso

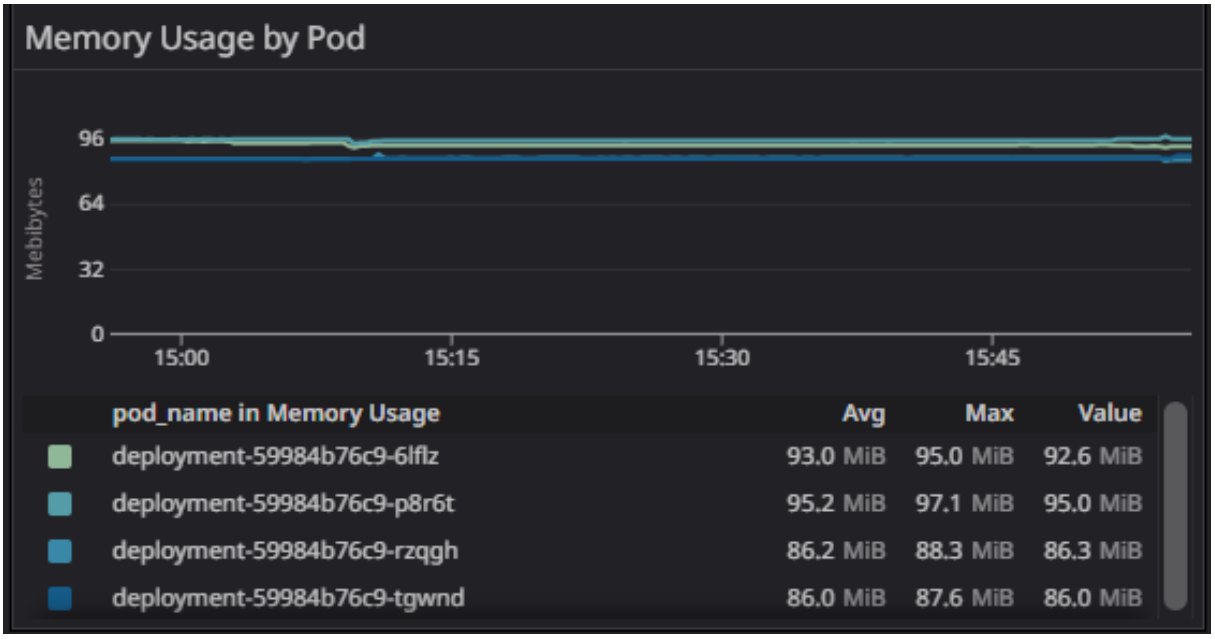
Fonte: Do Autor (2023).

5.7.3. Uso da Memória por Serviço

A gestão eficiente da memória é crucial para a operação estável e otimizada de qualquer aplicação. Ao monitorar o uso da memória de nossos três serviços centrais, identificamos padrões distintos para cada um deles.

O serviço chatservice-frontend, que atua como frontend em Next e BFF, durante o processo de deploy, apresentou picos de utilização de memória de 97%. Posteriormente, sob condições regulares de uso, estabilizou-se em uma média de 86%. O uso consistente e próximo ao pico pode indicar a demanda constante de recursos desse serviço, possivelmente devido à natureza de suas operações como interface frontend e BFF.

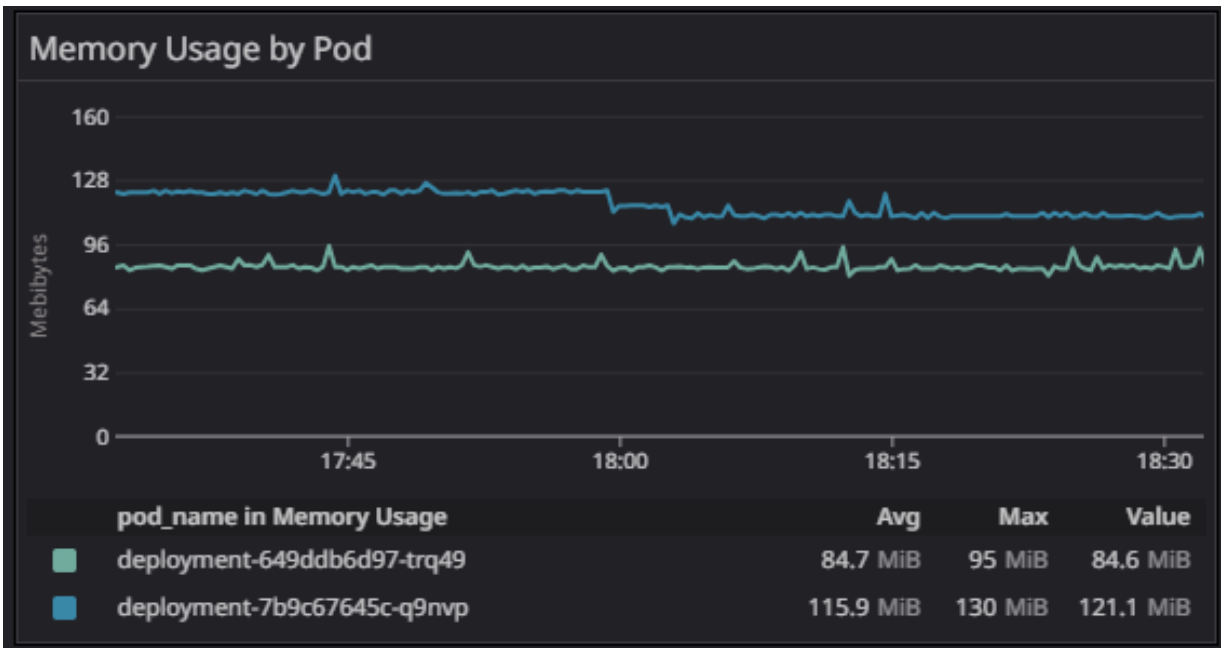
Figura 36 - Consumo de memória do Frontend



Fonte: Do Autor (2023).

Em contrapartida, o serviço chatservice-keycloak, responsável pela autenticação, registrou um pico de uso de memória de 50%. Essa cifra significativa pode refletir operações intensas, talvez relacionadas à autenticação de múltiplos usuários ou tarefas de inicialização. Após esse pico, o consumo estabilizou-se em uma média de 42%, sinalizando uma demanda de memória relativamente alta, mas consistente, para suas operações.

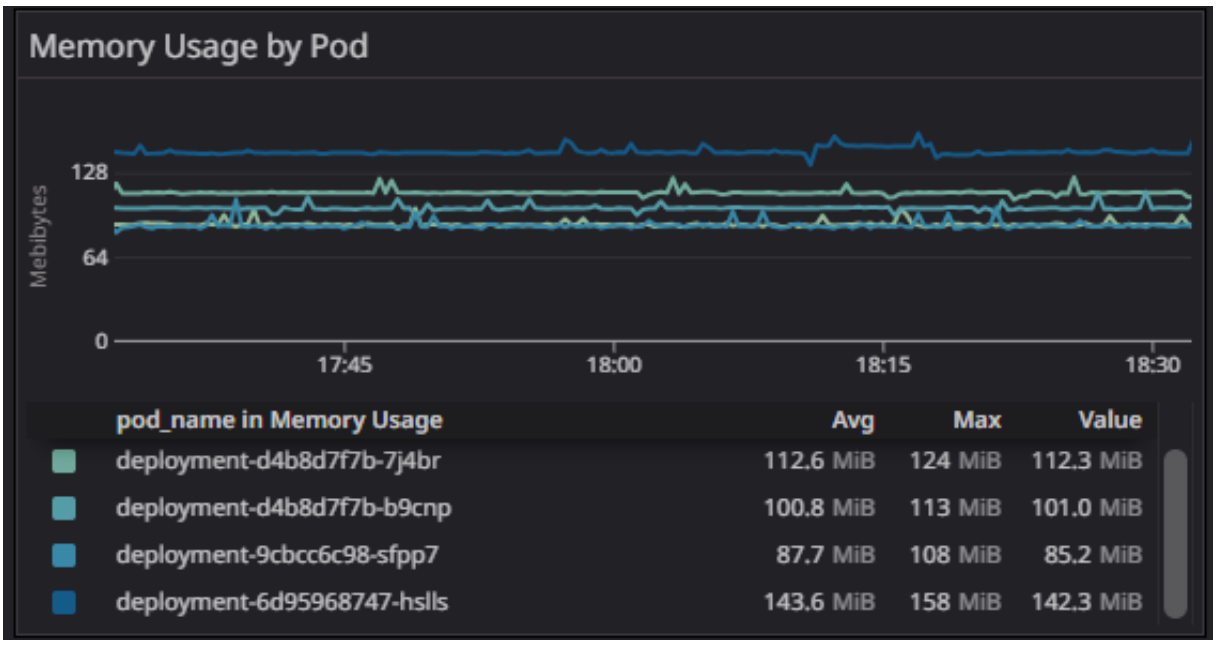
Figura 37 - Consumo de memória do Keycloak



Fonte: Do Autor (2023).

Por fim, o serviço chatservice, um MS Chat escrito em GO, apresentou características distintas dos outros dois serviços. De forma contrastante, demonstrou um uso de memória mais moderado. Durante o deploy, o pico alcançou 62%, e após a fase inicial, estabilizou-se em uma média de 55%. A eficiência na utilização de memória neste serviço pode ser atribuída, em parte, às características intrínsecas da linguagem GO, que é amplamente reconhecida por sua gestão otimizada de recursos.

Figura 38 - Consumo de memória do MS Chat



Fonte: Do Autor (2023).

Estes resultados fornecem um insight valioso sobre o comportamento do sistema em um ambiente de produção. Eles confirmam a eficiência de nossa configuração e otimização, e também nos fornecem informações para futuras melhorias. Ao combinar essas métricas quantitativas com os insights qualitativos coletados, estamos em posição de fazer avaliações bem-informadas e decisões estratégicas sobre o futuro desenvolvimento e escalonamento do sistema.

5.9 Desafios e Soluções na Implementação

A arquitetura de microsserviços promete modularidade, flexibilidade e escalabilidade, mas a sua implementação traz desafios intrínsecos. Aqui, examinamos os principais obstáculos enfrentados e as soluções adotadas para superá-los.

5.9.1 Autenticação Compartilhada entre Serviços

Desafio: Garantir uma autenticação segura e unificada entre diversos serviços é uma tarefa complexa em um ambiente de microsserviços. O tráfego de

informações sensíveis e a validação de credenciais entre serviços interdependentes tornam-se pontos de atenção.

Solução: A adoção do Keycloak emergiu como a solução ideal. Utilizando o protocolo OpenID, o Keycloak passou a gerenciar integralmente a autenticação dos usuários. Isso não apenas simplificou o processo de autenticação entre os microsserviços, mas também potencializou a segurança, centralizando a gestão de credenciais e sessões.

5.9.2. Persistência de Dados

Desafio: Em uma arquitetura de microsserviços, garantir a consistência e integridade dos dados distribuídos entre diferentes serviços é fundamental, sobretudo em cenários de falha ou alta demanda.

Solução: Optamos pelo Aurora da AWS, uma solução serverless de banco de dados. Além de oferecer alta disponibilidade e confiabilidade, o Aurora proporciona uma escalabilidade vertical automática, ajustando-se dinamicamente às necessidades da aplicação e garantindo a integridade dos dados em todos os momentos.

5.9.3. Escalabilidade Horizontal Independente:

Desafio: Cada serviço, com suas características e demandas únicas, precisa ser capaz de escalar horizontalmente de maneira autônoma, adaptando-se dinamicamente às variações de carga e tráfego.

Solução: A decisão foi usar o ECS da AWS. Esse serviço, projetado para gerenciar contêineres, provou ser uma solução simplificada, mas robusta. Criamos um cluster de "chatservice" no ECS, composto por três serviços (Frontend e BFF, MS Chat, e Keycloak), permitindo uma escalabilidade vertical individual. Esta estrutura permitiu que cada serviço respondesse de forma autônoma às demandas, escalando conforme necessário e garantindo performance otimizada.

5.10 Análise Qualitativa

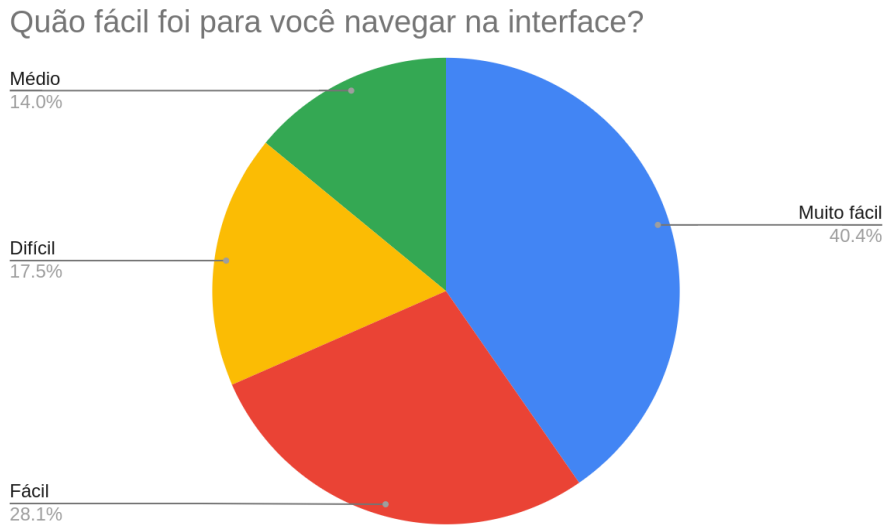
Para avaliar a eficácia e a usabilidade do serviço proposto, foi conduzida uma pesquisa qualitativa utilizando um formulário do Google Forms, contendo seis questões focadas em aspectos críticos da experiência do usuário: facilidade de navegação, incidência de bugs, tempo de carregamento da página, utilidade das respostas do ChatGPT, probabilidade de recomendação do serviço, e um espaço aberto para feedbacks adicionais. Essas questões foram essenciais para coletar insights sobre a performance do sistema e as percepções dos usuários, orientando futuras melhorias e ajustes.

A pesquisa foi realizada no período entre 21 e 23 de setembro de 2023. Ao invés de uma seleção específica de participantes, foi optado por uma abordagem mais aberta: um comunicado foi divulgado na rede interna da empresa onde este autor trabalha, onde, junto ao link de acesso do chat, havia também o link da pesquisa. Dessa forma, qualquer membro da empresa que utilizou o serviço teve a oportunidade de compartilhar suas opiniões.

O total de respondentes foi de 58 pessoas. Esse grupo forneceu feedbacks, opiniões e insights. Também todas as respostas foram coletadas de forma anônima, assegurando a privacidade e a sinceridade dos participantes. Acredito que, ao preservar a confidencialidade, conseguimos feedbacks mais genuínos, o que enriquece nossa análise.

Em resposta à pergunta sobre a facilidade de navegação na interface, a maioria dos usuários descreveu-a como "Muito fácil" (40,4%) ou "Fácil" (28,1%). Isso indica que a interface é intuitiva e amigável para a maioria dos usuários. No entanto, 14% das respostas classificaram a navegação como "Média" e 17,5% como "Difícil". Isso sugere que há espaço para melhorias em certos elementos da interface, com o objetivo de otimizar ainda mais a experiência do usuário.

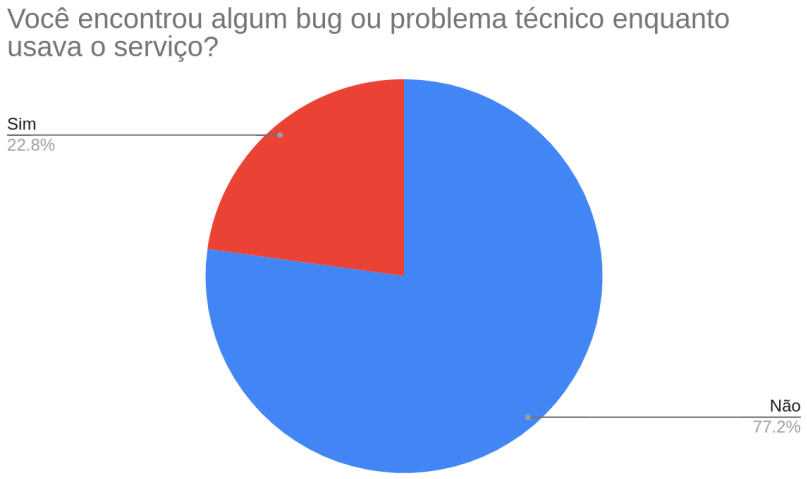
Figura 39 - Resultado questionário de navegação na interface



Fonte: Do Autor (2023).

Quando questionados se encontraram algum erro, 77,2% dos usuários responderam que "Não" encontraram bugs ou problemas técnicos. No entanto, 22,8% dos respondentes disseram que "Sim", se depararam com algumas falhas.

Figura 40 - Resultado questionário de bugs ou problemas técnico

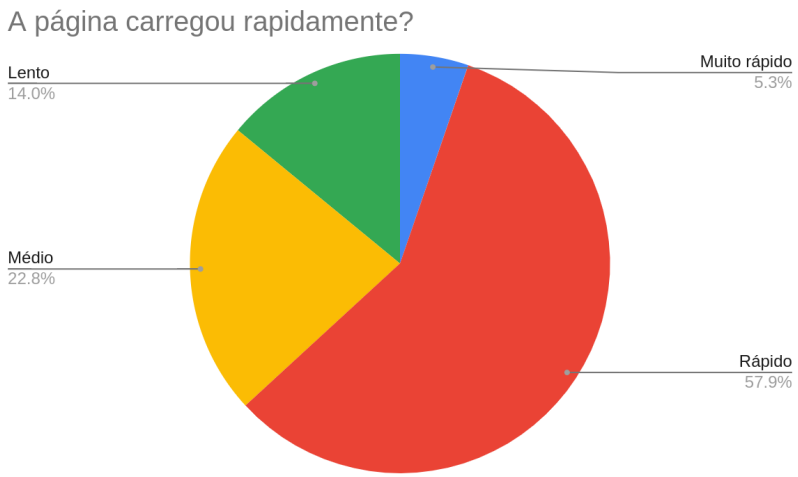


Fonte: Do Autor (2023).

Em resposta à questão sobre a velocidade de carregamento, 57,9% dos usuários classificaram o carregamento da página como "Rápido" e 5,3% como "Muito rápido". Isso sugere que os servidores estão bem otimizados e que os recursos utilizados na página são eficientes e leves. No entanto, 22,8% dos

respondentes descreveram o carregamento como "Médio" e 14% como "Lento". Estes feedbacks indicam a existência de possíveis pontos de estrangulamento ou problemas intermitentes que podem necessitar de atenção e ajustes para melhorar ainda mais a experiência dos usuários.

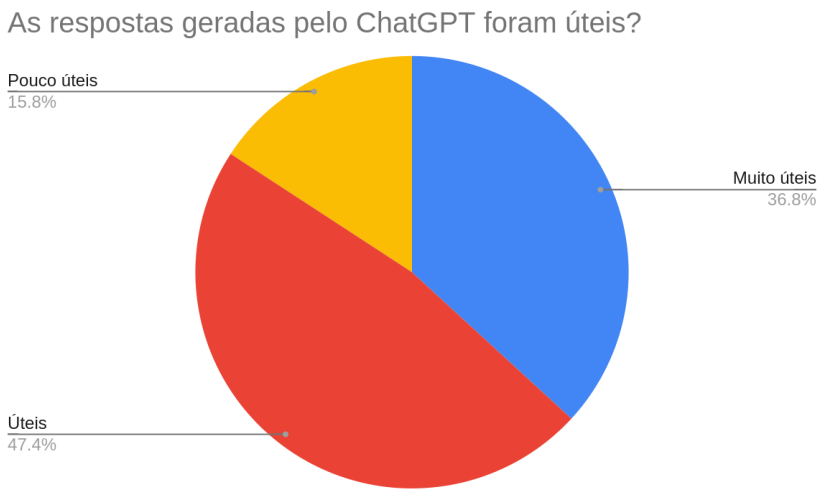
Figura 41 - Resultado questionário de velocidade do carregamento da página



Fonte: Do Autor (2023).

Também, 36,8% dos usuários classificaram as respostas do ChatGPT como "Muito úteis", enquanto 47,4% consideraram-nas "Úteis". Isso totaliza uma significativa maioria que percebe o modelo como eficiente na entrega de informações precisas e relevantes. No entanto, 15,8% dos respondentes acharam as respostas "Pouco úteis". Essa porcentagem, embora minoritária, é um lembrete da necessidade de contínuo aperfeiçoamento e adaptação do modelo para atender às diversas expectativas e necessidades dos usuários.

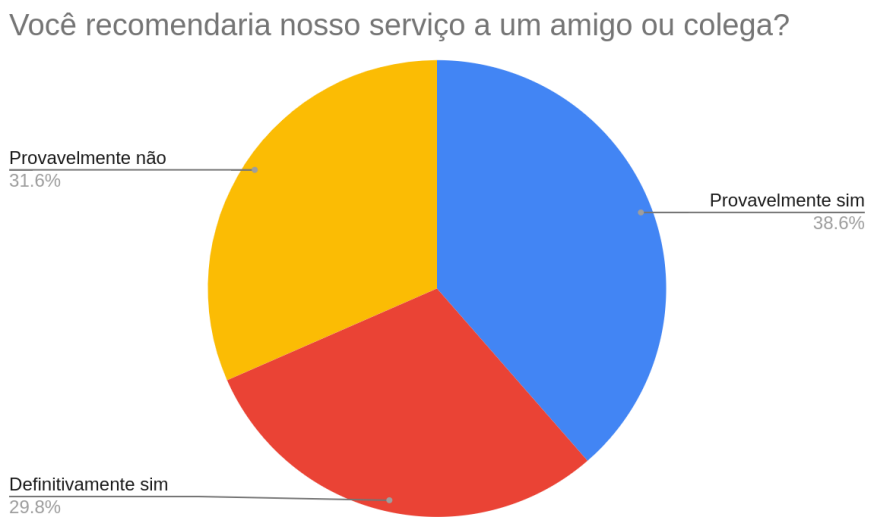
Figura 42 - Resultado questionário de utilidade das respostas



Fonte: Do Autor (2023).

38,6% dos respondentes disseram que "Provavelmente" recomendariam o serviço, enquanto 29,8% afirmaram que "Definitivamente" fariam uma recomendação. Juntos, esses percentuais demonstram uma satisfação significativa e uma percepção positiva geral do serviço. No entanto, não podemos ignorar que 31,6% dos usuários expressaram que "Provavelmente não" recomendariam. Este dado reforça a importância de entender e abordar as áreas de preocupação ou insatisfação para aqueles que têm reservas sobre a recomendação do serviço. A análise contínua e a resposta a esses feedbacks são cruciais para melhorar ainda mais a experiência geral do usuário.

Figura 43 - Resultado questionário de recomendação do serviço



Fonte: Do Autor (2023).

Os comentários extras apontam que precisamos deixar a interface mais fácil de usar e consertar os problemas técnicos que aparecem direto. Todos concordam em melhorar a aparência, algumas funcionalidades e arrumar bugs, como travamentos e dificuldades de entrar no sistema, vai fazer a experiência do usuário ficar melhor. Também é importante trazer algo novo que faça o produto se destacar dos outros que já existem.

5.11 Diretrizes Práticas para Implementação Eficaz de Microsserviços

Para atender ao objetivo central deste trabalho, que é estabelecer diretrizes práticas e recomendações para uma implementação eficaz de arquiteturas de microsserviços, consolidamos aqui um resumo das melhores práticas e recomendações estratégicas que emergiram ao longo da pesquisa. Estas orientações, embora dispersas no decorrer do texto, são o cerne para alcançar uma infraestrutura de microsserviços robusta e adaptável:

Figura 44 - Diretrizes para práticas para implementação

Fonte: Do Autor (2023).

Modularidade e Definição de Serviços: Cada microsserviço deve ser projetado com uma única responsabilidade e função de negócio, garantindo eficiência e clareza na operação.

Interoperabilidade e Integração Contínua: A utilização de APIs bem definidas é essencial para a interoperabilidade. Práticas de CI/CD são fundamentais para integração e entrega eficientes e rápidas.

Resiliência e Tolerância a Falhas: Implementar mecanismos como circuit breakers e fallbacks é crucial para manter a estabilidade do sistema diante de falhas, assegurando a continuidade operacional.

Estratégias de Escalabilidade: Técnicas de escalonamento horizontal e balanceamento de carga são necessárias para gerenciar variações de demanda, permitindo a adaptação dinâmica do sistema.

Segurança e Governança de Dados: Proteger dados e serviços é fundamental, com estratégias de autenticação, autorização e monitoramento contínuo, mantendo a integridade e a confidencialidade dos dados.

Monitoramento e Telemetria: Sistemas de monitoramento para observação em tempo real e tomada de decisões informadas são vitais, melhorando o desempenho e a capacidade de resposta do sistema.

Cultura e Colaboração Interdisciplinar: Uma cultura de colaboração entre equipes, com comunicação e cooperação efetiva, é essencial para o sucesso na implementação de microsserviços.

A concretização dessas diretrizes e recomendações não só reafirma o objetivo principal deste TCC, mas também configura um legado para práticas futuras, incentivando uma abordagem iterativa e evolutiva no campo do desenvolvimento de software. Assim, este trabalho não conclui uma jornada, mas lança as bases para que outros pesquisadores e profissionais possam continuar explorando e expandindo o vasto potencial dos microsserviços.

5.12 Conclusão dos resultados e discussões

Ao encerrar a seção de resultados e discussões deste trabalho, é notável que as evidências coletadas e as análises realizadas reiteram a viabilidade da arquitetura de microsserviços como uma estratégia robusta e flexível para o desenvolvimento de sistemas distribuídos. Os resultados obtidos são promissores, delineando um cenário onde as diretrizes e recomendações propostas mostraram-se eficazes e aplicáveis.

A síntese das práticas recomendadas, destacadas ao longo do estudo, constitui o cerne para uma implementação eficaz de microsserviços, ressaltando a importância da modularidade, interoperabilidade, resiliência, escalabilidade, segurança, monitoramento e colaboração interdisciplinar. Essas áreas-chave refletem não apenas as necessidades atuais de sistemas distribuídos, mas também apontam para uma cultura de melhoria contínua e adaptabilidade frente aos desafios emergentes.

Com base nos feedbacks coletados dos usuários durante a pesquisa, reconhece-se a necessidade de realizar ajustes que aprimorem a usabilidade e a performance dos sistemas desenvolvidos. Estes ajustes serão informados pelas experiências dos usuários e pela análise contínua dos indicadores de desempenho, garantindo que as melhorias sejam focadas e eficientes.

Além disso, este trabalho reconhece a importância de incorporar inovações que possam diferenciar e agregar valor ao produto final. O caminho a seguir inclui a exploração de novas tecnologias e abordagens, visando não apenas resolver as questões técnicas existentes, mas também oferecer recursos que coloquem os sistemas desenvolvidos à frente no mercado competitivo.

Em conclusão, este estudo não representa o término, mas uma etapa significativa na trajetória de pesquisa e desenvolvimento em arquiteturas de microsserviços. As diretrizes consolidadas aqui são um legado valioso para práticas futuras e estabelecem uma base sólida sobre a qual outros pesquisadores e profissionais poderão construir, impulsionando a evolução contínua e a excelência no campo do desenvolvimento de software.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, o objetivo foi explorar os desafios e características da arquitetura de microsserviços, focando principalmente na escalabilidade e resiliência dos sistemas. A pesquisa foi conduzida com uma metodologia científica rigorosa e abrangente, que incluiu tanto abordagens qualitativas quanto quantitativas. Isso nos permitiu investigar profundamente o tema em questão.

O foco estava centrado na aplicação real, com o desenvolvimento de serviços específicos como o serviço de chat, autenticação e interfaces, forneceu percepções sobre as promessas e desafios do uso de microsserviços. Cada componente revelou seus próprios desafios, desde a questão da autenticação compartilhada até a escalabilidade vertical e horizontal, e soluções inovadoras foram empregadas em cada etapa.

A expectativa é que as descobertas deste trabalho sirvam como guia e inspiração para profissionais e acadêmicos, lançando luz sobre práticas recomendadas e estratégias eficientes na adoção de microsserviços. Os resultados preliminares, embora ainda em uma fase incipiente, apontam para a capacidade inegável da arquitetura de microsserviços em realçar a resiliência e escalabilidade dos sistemas.

Porém, este estudo não foi isento de obstáculos. Desde a coleta de dados atualizados até a navegação pelo vasto ecossistema de microsserviços, enfrentamos desafios que exigiram adaptação, perseverança e aprendizado contínuo.

Para trabalhos futuros, sugere-se a exploração de novas tecnologias emergentes na área de microsserviços, como frameworks específicos para a gestão e orquestração desses serviços. Além disso, aprimoramentos no processo de coleta e análise de dados poderão contribuir para uma compreensão mais profunda do impacto dos microsserviços na eficiência operacional e na satisfação do cliente. Investigações adicionais sobre estratégias de segurança cibernética em arquiteturas

descentralizadas também podem oferecer insights valiosos, assim como a exploração de métodos avançados de monitoramento e telemetria para otimização de desempenho.

Concluindo, estas dificuldades não foram em vão, mas sim propulsoras de um aprofundamento acadêmico e prático. Cada desafio enfrentado enriqueceu nossa compreensão, permitindo-nos não apenas entender, mas também contribuir de forma significativa para o campo da arquitetura de microsserviços.

REFERÊNCIAS

AMBLER, S. W. **The Object Primer**: Agile model-driven Development with UML 2.0. [s.l.] Cambridge University Press, 2004.

AMUNDSEN, M.; MATT, M.; IRAKLI, N.; RONNIE, M. **Microservice Architecture**: Aligning Principles, Practices, and Culture. [s.l.] O'Reilly Media, 2016.

ANGRIST, J. D.; PISCHKE. J. **Mastering metrics**: the Path from Cause to Effect. [s.l.] Princeton University Press, 2014.

ARONEN, T. **From Monolith to Microservices**. Master's Thesis-Faculty of Science University of Helsinki: [s.n.].

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. **IEEE Software**, v. 33, n. 3, p. 42–52, mai. 2016.

BEHARA, G. K.; KHANDRIKA, T. **Microservices Practitioner Guide**. [s.l.] Meghan-Kiffer Press, 2018.

BELL, M. **Service-Oriented Modeling**: Service Analysis, Design, and Architecture. [s.l.] Wiley, 2008.

BENHAROSH, J. **What Is REST API | PHPenthusiast**. Disponível em: <https://phpenthusiast.com/blog/what-is-rest-api>. Acesso em: 3 nov. 2023.

BRAY, T. **Microservices at Amazon**. Disponível em: <https://www.tbray.org/ongoing/When/201x/2015/11/09/Microservices-at-Amazon>. Acesso em: 26 mar. 2023.

BROWN, S. **Software Architecture for developers**: Visualizing Software Architecture. [s.l.] Leanpub, 2017. v. 2

BROWN, S. **O Modelo C4 de Documentação para Arquitetura de Software**. Disponível em: <https://www.infoq.com/br/articles/C4-architecture-model/>. Acesso em: 28 mai. 2023.

CALCOTE, L.; BUTCHER, Z. **Istio**: up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe. [s.l.] O'Reilly Media, 2019.

CHACON, S.; STRAUB, B. **Pro Git**. 2. ed. Berkeley, CA: Apress, 2014.

CHONOLES, M. J.; SCHARDT, J. A. **UML 2 for Dummies**. New York: Wiley Pub, 2003.

COCKCROFT, A. **Managing Failure Modes in Microservice Architectures**. Disponível em: <https://www.infoq.com/presentations/microservices-failure-modes>. Acesso em: 16 mar. 2023.

CRESWELL, J. W. **Research Design**: Qualitative, Quantitative, and Mixed Methods Approaches. 4th. ed. [s.l.] Sage Publications, 2013.

DAYA, S.; VAN DUY, N.; EATI, K.; FERREIRA, C. M.; GLOZIC, D.; GUCER, V.; GUPTA, M.; JOSHI, S.; LAMPKIN, V.; MARTINS, M.; NARAIN, S.; VENNAM, R. **Microservices from Theory to practice**: Creating Applications in IBM Bluemix Using the Microservices Approach. [s.l.] Vervante, 2015.

DOCKER. **Enterprise Application Container Platform | Docker**. Disponível em: <https://www.docker.com/>. Acesso em: 12 out. 2023.

DRAGONI, N.; GIALLORENZO, S.; LLUCH LAFUENTE, A.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. **Microservices**: Yesterday, Today, and Tomorrow. In: MAZZARA, M.; MEYER, B. (Eds.). **Present and Ulterior Software Engineering**. Cham: Springer International Publishing, 2017. p. 195–216.

EVANS, E. **Domain-driven design**: Atacando as Complexidades no Coração do Software. 3ª. ed. [s.l.] Alta Books, 2016.

FOWLER, M. **Patterns of Enterprise Application Architecture**. 1ª. ed. [s.l.] Addison-Wesley Professional, 2002.

GARCIA-MOLINA, H.; SALEM, K. Sagas. **ACM SIGMOD Record**, v. 16, n. 3, p. 249–259, 1 dez. 1987.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. São Paulo: Atlas, 2022.

GRPC. **About gRPC**. Disponível em: <https://grpc.io/about>. Acesso em: 6 mai. 2023.

HARDT, D. **The OAuth 2.0 Authorization Framework**. Disponível em: <https://tools.ietf.org/html/rfc6749>. Acesso em: 7 mai. 2023.

HARRIS, D. **Talking Microservices with the Man Who Made Netflix's Cloud Famous**. Disponível em: <https://medium.com/s-c-a-l-e/1032689afed3>. Acesso em: 7 mai. 2023.

HASSAN, S.; BAHSOON, R.; BUYYA, R. Systematic Scalability Analysis for Microservices Granularity Adaptation Design Decisions. **Software: Practice and Experience**, v. 52, n. 6, 31 jan. 2022.

HENRIQUE, T. **SAGA Pattern para Microservices**. Disponível em: <https://dev.to/thiagosilva95/saga-pattern-para-microservices-2pb6>. Acesso em: 28 mai. 2023.

HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns**: designing, Building and Deploying Messaging Solutions. [s.l.] Addison-Wesley Professional, 2015.

HUMBLE, J.; FARLEY, D. **Continuous Delivery**: Reliable Software Releases through Build, Test, and Deployment Automation. 1ª. ed. [s.l.] Addison-Wesley Professional, 2011.

INDRASIRI, K.; SIRIWARDENA, P. **Microservices for the Enterprise**: Designing, Developing, and Deploying. 1st. ed. [s.l.] Apress, 2018.

İLERİ, A. **Introduction to Keycloak**. Disponível em: <https://abdulsamet-ileri.medium.com/introduction-to-keycloak-227c3902754a>. Acesso em: 25 set. 2023.

JONKERS, H.; IACOB, M.-E.; LANKHORST, M. M.; PROPER, E. (H.A.); QUARTEL, D. A.C. **ArchiMate® 2.1 Specification**. Disponível em: <https://pubs.opengroup.org/architecture/archimate2-doc>. Acesso em: 14 abr. 2023.

KIM, G. et al. **The Devops Handbook How to Create World-class Agility, Reliability, and Security in Technology Organizations.** [s.l.] IT Revolution Press, 2016.

KLEPPMANN, M. **Designing Data-Intensive Applications**: the Big Ideas behind Reliable, Scalable, and Maintainable Systems. [s.l.] O'Reilly Media, 2017.

LANKHORST, M. **Enterprise Architecture at Work**: Modelling, Communication and Analysis. 4st. ed. [s.l.] Springer, 2017.

LARMAN, C. **Applying UML and Patterns**: an Introduction to Object-Oriented Analysis and Design and Iterative Development. [s.l.] Pearson, 2004.

LEWIS, J.; FOWLER, M. **Microservices**: a Definition of This New Architectural Term. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 16 mar. 2023.

LIMA, T. **(Parte 3) Segurança Em APIs RESTful**. Disponível em: <https://thiagolima.blog.br/parte-3-seguran%C3%A7a-em-apis-restful-a780bd9f186a>. Acesso em: 28 mai. 2023.

MARQUESONE, R. **Big Data: Técnicas e Tecnologias para Extração de Valor dos Dados**. São Paulo: Casa Do Código, 2016.

MICROSOFT. **Cloud-native Data Patterns**. Disponível em: <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/distributed-data>.

MICROSOFT. **The API Gateway Pattern versus the Direct client-to-microservice Communication**. Disponível em: <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>. Acesso em: 3 nov. 2023.

NDUNGU, M. **Adoption of the Microservice Architecture**. Master of Science Thesis-Faculty of Science and Engineering ÅBO AKADEMI: [s.n.].

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

NYGARD, M. T. **Release It!**: Design and Deploy Production-Ready Software. 2nd. ed. [s.l.] Pragmatic Bookshelf, 2018.

O'HANLON, C. A Conversation with Werner Vogels. **Queue**, v. 4, n. 4, p. 14–22, maio 2006.

OMG GROUP. **About the Unified Modeling Language Specification Version 2.5**. Disponível em: <https://www.omg.org/spec/UML/2.5/>. Acesso em: 3 nov. 2023.

OPENAI. **OpenAI API**. Disponível em: <https://platform.openai.com/docs>. Acesso em: 25 set. 2023.

PRESTON-WERNER, T. **Semantic Versioning 2.0**. Disponível em: <https://semver.org>. Acesso em: 6 mai. 2023.

RAGAN, T. **Domain-Driven Design for Microservices**. Disponível em: <https://www.deployhub.com/domain-driven-design-microservices/>. Acesso em: 28 mai. 2023.

RICHARDSON, C. **Service Discovery in a Microservices Architecture**. Disponível em: <https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/>. Acesso em: 3 nov. 2023.

RICHARDSON, C. **Microservices Patterns**: with Examples in Java. 1ª. ed. Shelter Island, New York: Manning Publications, 2018.

RICHARDSON, L.; AMUNDSEN, M. **RESTful Web APIs**: Services for a Changing World. 1ª. ed. [s.l.] O'Reilly Media, 2013.

RODINA, D. **E-commerce Microservices (UML Deployment Diagram) - Software Ideas Modeler**. Disponível em: <https://www.softwareideas.net/a/1580/e-commerce-microservices-uml-deployment-diagram->. Acesso em: 28 mai. 2023.

SAYFAN, G. **Mastering Kubernetes - Second Edition**: Master the Art of Container Management by Using the Power of Kubernetes. 2nd. ed. [s.l.] Packt Publishing, 2018.

SKELTON, M.; PAIS, M. **Team topologies**: Organizing Business and Technology Teams for Fast Flow. [s.l.] IT Revolution, 2019.

SOFTWAREAG. **OpenID Authentication Use case and Workflow**. Disponível em: <https://documentation.softwareag.com/webmethods/compendiums/v10-11/C_API_Management/index.html#page/api-mgmt-comp/co-openid_usecase_workflow.html>. Acesso em: 28 mai. 2023.

STETSON, C. **MRA Part 6 – Circuit Breaker Pattern**. Disponível em: <https://www.nginx.com/blog/microservices-reference-architecture-nginx-circuit-breaker-pattern/>. Acesso em: 28 mai. 2023.

SUPERO. **Microsserviços: conceito, Vantagens E Desvantagens Dessa Arquitetura**. Disponível em: <https://www.supero.com.br/blog/microsservicos-conceito-vantagens-e-desvantagens-desse-tipo-de-arquitetura/>. Acesso em: 3 nov. 2023.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: an Empirical Investigation. **IEEE Cloud Computing**, v. 4, n. 5, p. 22–32, set. 2017.

THE OPEN GROUP. **ArchiMate® 3.0 Specification**. Disponível em: <https://pubs.opengroup.org/architecture/archimate30-doc/>. Acesso em: 7 mai. 2023.

THE OPEN GROUP. **ArchiMate® 3.1 Specification**. Disponível em: <https://pubs.opengroup.org/architecture/archimate31-doc/>. Acesso em: 28 mai. 2023.

VERNON, V. **Implementing Domain-Driven Design**. 1ª. ed. [s.l.] Addison-Wesley Professional, 2013.

VOGELS, W. Eventually Consistent. **Communications of the ACM**, v. 52, n. 1, p. 40–44, jan. 2009.

WILDE, E.; PAUTASSO, C. **REST: from Research to Practice**. [s.l.] Springer, 2011.

WOLFF, E. **Microservices**: Flexible Software Architecture. 1. ed. Boston: Addison-Wesley Professional, 2016.

WOLFF, E. **Microservices**: a Practical Guid**e**. [s.l.] Createspace Independent Publishing Platform, 2018.

WOOD, C.; ANTHONY, A.; LAURET, A.; SANDOVAL, K. **The API Economy**: Disruption and the Business of APIs. [s.l.] Nordic APIs AB, 2016.

YIN, R. K. **Case study research and applications**: Design and methods. 6. ed. Thousand Oaks, California: Sage Publications, 2017.

ZAFAR, F. **Investigating Quality Attributes and Best Practices of Microservices Architectures**. Master Thesis - Faculty of Mathematics, Computer Science, and Natural Sciences: [s.n.].