UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
CAMPUS ARAPIRACA
CIÊNCIA DA COMPUTAÇÃO - BACHARELADO

FRANKLYN ROBERTO DA SILVA

DESENVOLVIMENTO DE UMA APLICAÇÃO ESCALÁVEL EM NUVEM

ARAPIRACA
2023

Franklyn Roberto da Silva

Desenvolvimento de uma aplicação escalável em nuvem

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Alagoas, como requisito parcial à obtenção do título de Bacharelado/Licenciatura em Ciência da Computação.

Orientador: Prof. Dr. Tércio de Morais Sampaio Silva.

Arapiraca

2023

Universidade Federal de Alagoas – UFAL
Campus Arapiraca
Biblioteca Setorial *Campus* Arapiraca - BSCA

S586d Silva, Franklyn Roberto da
Desenvolvimento de uma aplicação escalável em nuvem [recurso eletrônico] / Franklyn Roberto da Silva. – Arapiraca, 2023.
59 f.: il.

Orientador: Prof. Dr. Tércio de Morais Sampaio.
Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Alagoas, *Campus* Arapiraca, Arapiraca, 2023.
Disponível em: Universidade Digital (UD) / RD- BSCA– UFAL (*Campus* Arapiraca).
Referências: f. 57-59.

1. Nuvem escalável. 2. Microsserviços. 3. Contêiner. 4. Escalabilidade. 5. Infraestrutura em nuvem. I. Sampaio, Tércio de Morais. II. Título.

CDU 004

Bibliotecário responsável: Nestor Antonio Alves Junior
CRB-4 / 1557

Franklyn Roberto da Silva

Desenvolvimento de uma aplicação escalável em nuvem

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Alagoas, como requisito parcial à obtenção do título de Bacharelado/Licenciatura em Ciência da Computação.

Data de aprovação: 25/05/2023.

Banca examinadora

Documento assinado digitalmente
gov.br
TERCIO DE MORAIS SAMPAIO SILVA
Data: 20/09/2023 10:55:19-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Tércio de Morais Sampaio
Universidade Federal de Alagoas - UFAL
Campus Arapiraca
(Orientador)

Documento assinado digitalmente
gov.br
RAQUEL DA SILVA CABRAL
Data: 21/09/2023 09:21:34-0300
Verifique em https://validar.iti.gov.br

Prof.ª Dr.ª Raquel da Silva Cabral
Universidade Federal de Alagoas - UFAL
Campus Arapiraca
(Examinadora)

Documento assinado digitalmente
gov.br
ALEXANDRE PAES DOS SANTOS
Data: 26/09/2023 14:35:45-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Alexandre Paes dos Santos,
Universidade Federal de Alagoas - UFAL
Campus Arapiraca
(Examinador)

RESUMO

O desenvolvimento de aplicações escaláveis em nuvem é um desafio para os profissionais da área de tecnologia da informação, pois exige o uso de arquiteturas e ferramentas adequadas para garantir a qualidade, a performance e a disponibilidade dos sistemas. Uma das abordagens mais utilizadas para esse fim é a de microsserviços, que consiste em dividir uma aplicação em pequenos serviços independentes e comunicáveis entre si. O presente trabalho tem como tema o desenvolvimento de uma aplicação escalável em nuvem, utilizando os microsserviços como estratégia para alcançar esse objetivo. O problema que motivou esta pesquisa foi a dificuldade de manter e evoluir aplicações monolíticas, que tendem a se tornar complexas e custosas com o tempo. O objetivo geral deste trabalho é demonstrar como os microsserviços, proporcionando benefícios como modularidade, escalabilidade, resiliência e independência para aplicações escaláveis. Visando demonstrar o desenvolvimento de uma aplicação escalável, um estudo de caso é apresentado e a partir dele uma arquitetura de sistema baseada em microsserviços foi projetada, definindo seus componentes e interações definição de modelos de comunicação, bem como a seleção de ferramentas de suporte ao ambiente de execução de microsserviços. Por fim, o sistema foi implementado e testado. Como resultado do desenvolvimento, o sistema de livraria online usufrui dos benefícios de uma arquitetura baseada em microsserviços de forma a garantir capacidade de escala mundial sem comprometer seus requisitos não funcionais de escalabilidade e resiliência.

Palavras-chave: nuvem escalável; microsserviços; contêiner; escalabilidade; infraestrutura em nuvem.

ABSTRACT

The development of scalable cloud applications is a challenge for professionals in the field of information technology, as it requires the use of appropriate architectures and tools to ensure the quality, performance, and availability of systems. One of the most utilized approaches for this purpose is microservices, which involves breaking down an application into small, independent, and communicable services. The current work focuses on the development of a scalable cloud application using microservices as a strategy to achieve this goal. The problem that motivated this research was the difficulty of maintaining and evolving monolithic applications, which tend to become complex and costly over time. The overall objective of this work is to demonstrate how microservices provide benefits such as modularity, scalability, resilience, and independence for scalable applications. In order to showcase the development of a scalable application, a case study is presented, and based on it, a microservices-based system architecture is designed. This includes defining its components and interactions, communication model definitions, as well as the selection of tools to support the microservices runtime environment. Finally, the system is implemented and tested. As a result of this development, the online bookstore system enjoys the benefits of a microservices-based architecture, ensuring global scalability without compromising its non-functional requirements of scalability and resilience.

Keywords: scalable cloud; microservices; container; scalability; cloud infrastructure.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface* (Interface de Programação de Aplicativos)
SOC	*Security Operations Center* (Centro de Operações de Segurança)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
SOA	*Service Oriented Architecture* (Arquitetura Orientada a Serviços)
TI	*Information Technology* (Tecnologia da Informação)
OOA	*Object-Oriented Architecture* (Arquitetura Orientada a Objetos)
AWS	*Amazon Web Services*
SOAP	*Simple Object Access Protocol*
JSON	*JavaScript Object Notation* (Notação de objeto JavaScript)
PDF	*Portable Document Format*
SLA	*Service Level Agreement*
DevOps	*Development Operations*
CI/CD	*Continuous Integration/Continuous Delivery*
REST	*Representational State Transfer*
IP	*Internet Protocol* (Protocolo de Internet)
AMQP	*Advanced Message Queuing Protocol* (Protocolo avançado de enfileiramento de mensagens)
CPU	*Central Processing Unit* (Unidade Central de Processamento)

SUMÁRIO

1 INTRODUÇÃO

A adoção de microsserviços é outro benefício importante da computação em nuvem. Os microsserviços são uma abordagem arquitetura em que um aplicativo é dividido em componentes menores e independentes, chamados de microsserviços. Cada microsserviço é responsável por uma função específica e pode ser desenvolvido, implantado e dimensionado de forma independente. Uma das vantagens de implementar a arquitetura de microserviços é a flexibilidade, que permite reutilizar os mesmos serviços em diferentes projetos e contextos de negócio. Outra vantagem é a diversidade tecnológica, que possibilita usar diferentes tecnologias e linguagens de programação para cada serviço. Além disso, a arquitetura de microserviços aumenta a escalabilidade, ou seja, a capacidade de acompanhar o crescimento das operações de uma empresa sem comprometer o desempenho e a qualidade. A arquitetura de microserviços também facilita a descentralização, que permite implementar, melhorar e corrigir cada serviço de maneira independente. Por fim, a arquitetura de microserviços otimiza a segurança, pois cada serviço tem seu próprio banco de dados, evitando que uma violação que ocorre em um serviço afete as outras áreas.

O problema que motivou esta pesquisa foi a dificuldade de manter e evoluir aplicações monolíticas que tendem a se tornar complexas e custosas com o tempo. Essas aplicações são caracterizadas por terem uma única unidade de implantação e execução, que contém toda a lógica de negócio e funcionalidades do sistema. Com o aumento das demandas por novas funcionalidades, correções de erros e adaptações a diferentes cenários, as aplicações monolíticas se tornam difíceis de compreender, testar, modificar e escalar. Além disso, elas apresentam baixa flexibilidade para lidar com mudanças tecnológicas, pois exigem a reimplantação de todo o sistema a cada alteração. Esses fatores impactam negativamente na qualidade e na produtividade do desenvolvimento de software.

O objetivo geral deste trabalho é demonstrar como os microsserviços podem facilitar o desenvolvimento de uma aplicação em nuvem, oferecendo benefícios como modularidade, escalabilidade, resiliência e independência. Os microsserviços são uma abordagem arquitetônica do desenvolvimento de software, na qual o software é composto por pequenos serviços independentes que se comunicam usando APIs (*Application Programming Interface*) bem definidas. Esses serviços são de responsabilidade de pequenas equipes auto suficientes que podem trabalhar de forma mais rápida e ágil. Para alcançar esse objetivo, este trabalho constrói uma aplicação que utiliza microsserviços, baseada em um estudo de caso de uma livraria online de alcance mundial. A aplicação foi dividida em componentes especializados e

autônomos que executam cada processo do negócio como um serviço. Esses serviços podem ser atualizados, implantados e escalados para atender à demanda de funções específicas da aplicação. A comunicação entre os serviços foi realizada por meio de interfaces leves usando protocolo HTTP (Protocolo de Transferência de Hipertexto), orientados a eventos ou por mensagens. A arquitetura de microsserviços possibilitou a construção de uma aplicação mais eficiente, flexível e resiliente.

O estudo adotou uma metodologia baseada em evidências dos benefícios dos microsserviços, que permitem melhorar os processos de negócios e aumentar a eficiência. Além disso, aplicou-se a pesquisa-ação, que envolve um ciclo de planejamento, implementação e avaliação de uma intervenção prática.

Os resultados alcançados com o projeto em nuvem foram extremamente satisfatórios. Os microsserviços proporcionaram uma arquitetura modular e flexível, facilitando o desenvolvimento, a manutenção e a evolução da aplicação. A principal conquista foi a escalabilidade, permitindo que a infraestrutura se adaptasse rapidamente às demandas do mercado, garantindo alta disponibilidade e desempenho. A aplicação também se mostrou mais resiliente, com a capacidade de lidar com falhas em serviços individuais sem impactar o funcionamento global. A interfaces leves e protocolos eficientes facilitaram a comunicação ágil e eficaz entre os microsserviços. Esses resultados destacam a importância dos microsserviços como uma arquitetura promissora. Este trabalho contribui tanto para a academia quanto para o mercado, oferecendo percepções valiosas e práticas para o desenvolvimento de aplicações em nuvem escaláveis e eficientes.

Este trabalho mostra como os microsserviços podem contribuir para o desenvolvimento de aplicações escaláveis em nuvem. Os microsserviços representam uma mudança de paradigma na forma de projetar, construir e gerenciar aplicações distribuídas, que são cada vez mais demandadas pelo mercado e pela sociedade. A aplicação desenvolvida neste trabalho ilustra como os microsserviços podem ser aplicados em um cenário real e complexo, envolvendo diversos requisitos funcionais e não funcionais. A aplicação também serve como um exemplo prático e didático para outros desenvolvedores e pesquisadores que queiram aprender ou aprofundar seus conhecimentos sobre essa abordagem arquitetônica. Portanto, este trabalho tem um valor significativo tanto para o campo acadêmico quanto para o campo profissional da computação em nuvem e dos microsserviços.

O restante deste trabalho está estruturado como se segue. No Capítulo 2, será apresentado os fundamentos das arquiteturas que desempenham um papel essencial em sistemas escaláveis e flexíveis. No Capítulo 3, são descritos os detalhes sobre os microsserviços. No Capítulo 4 segue o desenvolvimento de aplicações escaláveis em nuvem apresentando desafios significativos devido à natureza distribuída e dinâmica da nuvem. No Capítulo 5, será apresentado o desenvolvimento de uma aplicação escalável baseada em microsserviços levando em consideração os requisitos de alto desempenho e escalabilidade exigidos por um ambiente global. E por fim, no Capítulo 6, traremos as conclusões e as perspectivas futuras sobre o tema.

2 FUNDAMENTOS

A arquitetura de microserviços é uma forma de desenvolver softwares moderno, baseada em serviços pequenos e independentes que se comunicam por meio de interfaces padronizadas. Esses serviços são responsáveis por implementar capacidades de negócios específicas dentro de um domínio delimitado, e podem ser desenvolvidos, implantados e escalados de forma autônoma. Segundo a Microsoft (2023), a abordagem de microsserviços consiste em desenvolver um aplicativo para servidores por meio de um conjunto de serviços pequenos.

Arquiteturas que desempenham um papel essencial em sistemas escaláveis e flexíveis são aquelas que possibilitam a adaptação às mudanças de requisitos, demandas e tecnologias. Essas arquiteturas são baseadas em princípios como modularidade, baixo acoplamento, alta coesão, abstração e padrões de projeto. Esses princípios facilitam a decomposição do sistema em componentes independentes e reutilizáveis, que podem ser combinados de diferentes formas para atender a diferentes cenários. Por exemplo, a arquitetura de microsserviços consiste em dividir o sistema em pequenos serviços que se comunicam por meio de interfaces bem definidas e podem ser implantados e atualizados de forma independente. Isso permite que o sistema seja mais resiliente, escalável e fácil de manter. Além disso, essas arquiteturas favorecem a integração contínua, o teste automatizado e a entrega contínua, que são práticas que aumentam a qualidade e a confiabilidade do sistema. Algumas das arquiteturas que seguem esses princípios são a arquitetura em camadas, a arquitetura baseada em eventos e a arquitetura orientada a serviços.

2.1 Arquitetura em camadas

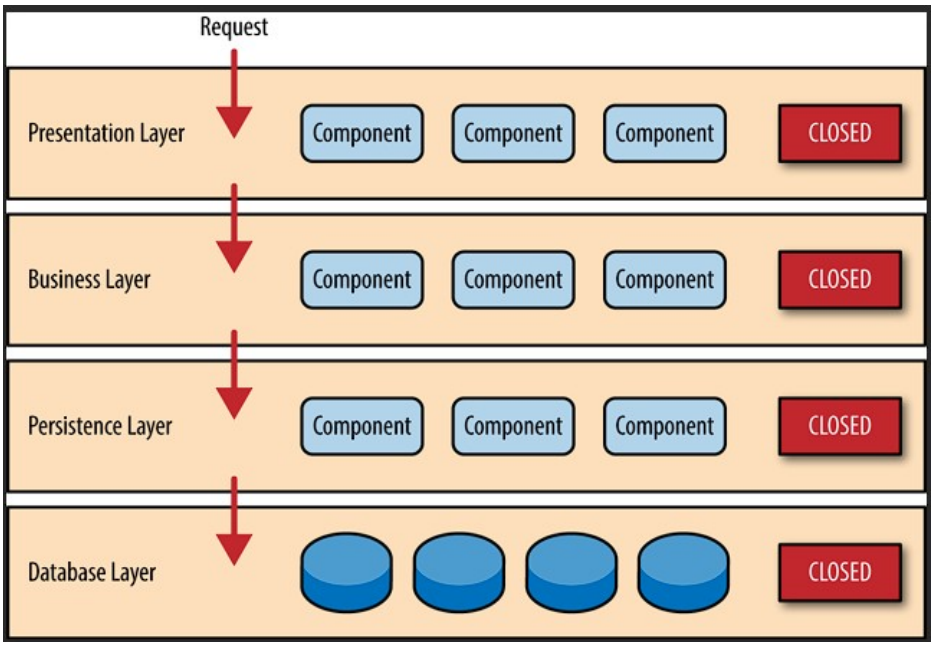
A arquitetura em camadas é um modelo de desenvolvimento de sistemas que consiste em organizar as funcionalidades em diferentes níveis de abstração, cada um com um papel específico. O propósito é simplificar a compreensão, a manutenção e a evolução do sistema, além de possibilitar a troca ou o reuso de camadas.

Um exemplo de arquitetura em camadas é a arquitetura em três camadas para aplicações web, que segmenta o sistema em três componentes principais: a camada de apresentação, a camada de negócio e a camada de persistência. A camada de apresentação é encarregada da interação com o usuário, por meio de interfaces gráficas ou serviços web. Por exemplo, uma página web que mostra os produtos de uma loja on-line faz parte da camada de apresentação. A camada de negócio é encarregada da lógica e das regras de negócio da

aplicação, como validações, cálculos e processos. Por exemplo, uma função que verifica se o usuário tem saldo suficiente para comprar um produto faz parte da camada de negócio. A camada de persistência é encarregada do acesso aos dados guardados em bancos de dados ou outros meios. Por exemplo, uma classe que insere ou consulta os dados dos produtos em um banco de dados fazem parte da camada de persistência.

Cada camada se comunica somente com as camadas adjacentes, seguindo um fluxo hierárquico. Assim, a camada de apresentação requisita serviços à camada de negócio, que por sua vez requisita operações à camada de persistência. Desse modo, cada camada esconde sua complexidade e oferece uma interface simples para as outras. Isso aumenta a modularidade, a coesão e o baixo acoplamento do sistema.

Figura 1 – Um exemplo de arquitetura em camadas



Fonte: ASKM (2019). Disponível em: http://blog.askm.com.br/2019/02/01/padroes-de-projeto-arquitetura-em-camadas/. Acesso em: 10 ago. 2023

A Figura 1, ilustra uma arquitetura em camadas com diferentes componentes em cada camada.

Camada de Apresentação (*Presentation Layer*): Essa é a camada responsável pela interação com o usuário ou com outros sistemas externos. Ela recebe as requisições (*requests*) vindas de algum cliente ou sistema e encaminha para processamento nas camadas subsequentes. Na Figura 1, tem vários componentes nessa camada. Cada um desses componentes recebe as requisições e as encaminha para a próxima camada.

Camada de Negócio (*Business Layer*): Essa camada contém a lógica de negócio do sistema. Ela recebe as requisições da camada de apresentação e processa essas requisições para realizar as operações necessárias. Assim como na camada de apresentação, existem componentes nessa camada que recebem as requisições e as encaminham para a próxima camada.

Camada de Persistência (*Persistence Layer*): Essa camada é responsável pelo acesso e manipulação dos dados armazenados em algum tipo de repositório, como um banco de dados. Ela recebe as requisições da camada de negócio e executa as operações de leitura, gravação ou atualização dos dados conforme necessário. Da mesma forma que nas camadas anteriores, existem componentes nessa camada que recebem as requisições e as encaminham para a próxima camada.

Camada de Banco de Dados (*Database Layer*): Essa é a camada final da arquitetura em camadas. Ela representa os bancos de dados onde os dados são armazenados de forma persistente. É nessa camada que as operações de leitura e escrita nos dados são realizadas de fato.

No geral, a Figura 1 ilustra a sequência de fluxo das requests através das camadas da arquitetura em camadas. Cada camada tem um conjunto de componentes que executam funções específicas e o "closed" indica que as camadas estão isoladas umas das outras, o que ajuda a manter a separação de responsabilidades e a modularidade do sistema.

Os benefícios da arquitetura em camadas são vários. Primeiro, ela facilita a divisão do trabalho entre os desenvolvedores, pois cada um pode se concentrar em uma camada específica. Segundo ela permite a reutilização de código entre diferentes sistemas que seguem o mesmo modelo. Terceiro, ela facilita a adaptação do sistema a mudanças nos requisitos ou nas tecnologias, pois basta alterar ou substituir uma camada sem afetar as outras. Quarto, ela melhora a qualidade e a confiabilidade do sistema, pois cada camada pode ser testada e depurada independentemente.

Além disso, a arquitetura em camadas possibilita a reutilização de componentes. As camadas inferiores, que fornecem serviços mais básicos, podem ser reaproveitadas por camadas superiores, que utilizam esses serviços para construir funcionalidades mais complexas. Essa separação de responsabilidades também facilita a troca de implementações. Caso seja necessário substituir uma camada por outra que ofereça a mesma interface, essa substituição pode ser feita de forma transparente para as outras camadas, minimizando o impacto nas demais partes do sistema.

2.2 Arquitetura baseada em eventos

Uma forma de projetar microserviços é usando uma arquitetura baseada em eventos. Segundo o Guia de Arquitetura do Microsoft Azure (MICROSOFT, 2023), nessa arquitetura, existem produtores de eventos que criam um fluxo de eventos e consumidores de eventos que recebem os eventos. Nesse modelo arquitetural, os microserviços se comunicam por meio de eventos assíncronos, o que aumenta a escalabilidade e a resiliência do sistema. Por exemplo, um microserviço pode publicar um evento quando um pedido é criado, e outro microserviço pode consumir esse evento para enviar uma confirmação ao cliente. Dessa forma, os microserviços não precisam se acoplar diretamente, e podem reagir a mudanças no sistema de forma flexível.

Nesta arquitetura, cada microserviço é responsável por enviar e/ou receber eventos que representam a ocorrência de uma ação ou mudança de estado. Os eventos podem ser consumidos por outros microserviços que possuem interesse em seus dados ou que precisam realizar ações a partir das informações recebidas.

A utilização de eventos assíncronos permite que os microserviços possam operar de forma independente, sem a necessidade de uma comunicação síncrona entre eles. Isso significa que o sistema pode lidar com um grande volume de eventos e escalá-los horizontalmente, sem a necessidade de aumentar a capacidade de cada microserviço individualmente.

Além disso, a arquitetura baseada em eventos torna o sistema mais resiliente, já que os eventos são armazenados em um registro de eventos (*event store*) e podem ser reproduzidos caso ocorram falhas ou indisponibilidades. Isso garante que o sistema possa se recuperar rapidamente em caso de problemas.

No entanto, é importante destacar que a utilização de eventos também apresenta alguns desafios, como a necessidade de garantir a ordem correta de processamento dos eventos, a implementação de mecanismos de controle de fluxo e a necessidade de garantir a consistência dos dados em tempo real.

Para implementar uma arquitetura baseada em eventos, é necessário definir a estrutura dos eventos e as regras de comunicação entre os microserviços. É recomendado o uso de padrões e ferramentas de mensageria, como *Kafka* e *RabbitMQ*, para facilitar a implementação e o gerenciamento dos eventos.

Outro ponto importante é a necessidade de realizar testes de carga e monitoramento para garantir a qualidade e a disponibilidade do sistema. É necessário monitorar os eventos

em tempo real para identificar possíveis problemas e ajustar as regras de comunicação caso necessário.

Por fim, a arquitetura baseada em eventos é uma abordagem eficiente para a comunicação entre os microserviços em arquiteturas de microserviços. A utilização de eventos assíncronos torna o sistema mais escalável e resiliente, permitindo lidar com grandes volumes de dados e falhas de forma mais eficiente. No entanto, é necessário estar atento aos desafios de implementação e gerenciamento para que a implementação seja bem-sucedida.

2.3 Arquiteturas Orientadas a Serviço

As arquiteturas orientadas a serviço (SOA - Service-Oriented Architecture) são uma forma de desenvolver softwares que utiliza componentes reutilizáveis chamados de serviços, que se comunicam por meio de uma linguagem comum em uma rede. Cada serviço é uma unidade independente de software que realiza uma função específica, como recuperar dados ou executar uma operação. Esses serviços podem ser acessados remotamente e modificados de forma independente. A ideia básica é desacoplar as funcionalidades do sistema em serviços independentes, que podem ser combinados de forma flexível para atender às necessidades do negócio.

Segundo a Red Hat (2020), SOA é um tipo de design de software que "torna os componentes reutilizáveis usando interfaces de serviços com uma linguagem de comunicação comum em uma rede". Por exemplo, um serviço de autenticação pode ser usado por várias aplicações que precisam verificar a identidade dos usuários. Um serviço de pagamento pode ser usado por diferentes aplicações que precisam processar transações financeiras. Um serviço de notificação pode ser usado por diferentes aplicações que precisam enviar mensagens para os usuários.

A SOA é considerada uma evolução das arquiteturas orientadas a objeto (OOA - *Object-Oriented Architecture*), que são baseadas no encapsulamento de dados e comportamentos dentro de objetos. De acordo com a AWS (2020), as arquiteturas orientadas a objeto têm limitações em relação à reutilização de código e à flexibilidade para se adaptar às mudanças nos requisitos do negócio. Por exemplo, se uma funcionalidade precisa ser adicionada ou modificada em um objeto, isso pode afetar outros objetos que dependem dele, tornando a mudança mais difícil e aumentando o risco de efeitos colaterais indesejados. Já com uma arquitetura orientada a serviço, as funcionalidades são desacopladas em serviços independentes, tornando mais fácil a reutilização de código e permitindo uma maior

flexibilidade para se adaptar às mudanças nos requisitos do negócio. Além disso, a SOA permite que os serviços sejam facilmente compartilhados e combinados para criar aplicações maiores e mais complexas, além de ser mais adaptável aos avanços da tecnologia. Por exemplo, uma aplicação de comércio eletrônico pode usar serviços de catálogo, carrinho, checkout, estoque, envio e faturamento para oferecer uma experiência completa aos clientes. Uma aplicação de saúde pode usar serviços de cadastro, agendamento, prontuário eletrônico, prescrição e cobrança para gerenciar os processos clínicos e administrativos.

Além disso, a arquitetura SOA permite criar e integrar serviços reutilizáveis em uma rede. Esses serviços são unidades ou conjuntos de funcionalidades de software independentes, que realizam tarefas específicas, como acessar dados ou executar operações. Eles se comunicam por meio de interfaces padronizadas e linguagens comuns, como SOAP, JSON, *ActiveMQ* ou *Apache Thrift*. A arquitetura SOA traz diversas vantagens para as empresas que a adotam, tais como:

- Escalabilidade: os serviços podem ser facilmente replicados ou distribuídos para atender à demanda crescente ou variável dos usuários. Por exemplo, se uma aplicação precisa consultar o saldo de uma conta bancária, ela pode usar um serviço que faz essa operação e que pode ser escalado conforme a necessidade.

- Manutenção: os serviços podem ser atualizados ou modificados de forma independente, sem afetar o funcionamento das aplicações que os consomem. Por exemplo, se um serviço que calcula o imposto de renda precisa ser alterado por causa de uma mudança na legislação, ele pode ser atualizado sem impactar as aplicações que usam esse serviço.

- Confiabilidade: os serviços podem ser monitorados e gerenciados para garantir o seu desempenho e disponibilidade. Por exemplo, se um serviço que faz a reserva de passagens aéreas falhar, ele pode ser substituído por outro serviço que oferece a mesma funcionalidade.

- Disponibilidade: os serviços podem ser acessados remotamente por diferentes aplicações e plataformas, sem depender de uma infraestrutura específica. Por exemplo, uma aplicação web pode usar um serviço que gera relatórios em PDF a partir de dados armazenados em um banco de dados na nuvem.

- Produtividade: os serviços podem ser reutilizados em diferentes aplicações e processos de negócio, evitando a duplicação de esforços e recursos. Por exemplo, um serviço que faz a validação de dados cadastrais pode ser usado por várias aplicações que precisam dessa funcionalidade.

Para implementar a arquitetura SOA, é necessário seguir um conjunto de princípios que orientam o desenvolvimento e a organização dos serviços. Esses princípios são:

- Padronização da interface: os serviços devem ter interfaces bem definidas e documentadas, que permitam a sua descoberta e consumo por outros aplicativos. Por exemplo, um serviço que faz o envio de e-mails deve ter uma interface que especifique os parâmetros necessários para essa operação, como o destinatário, o assunto e o conteúdo do e-mail.

- Autonomia: os serviços devem funcionar de forma independente uns dos outros, sem depender de estados ou contextos externos. Por exemplo, um serviço que faz a conversão de moedas deve retornar o valor convertido sem depender de outros serviços ou informações externas.

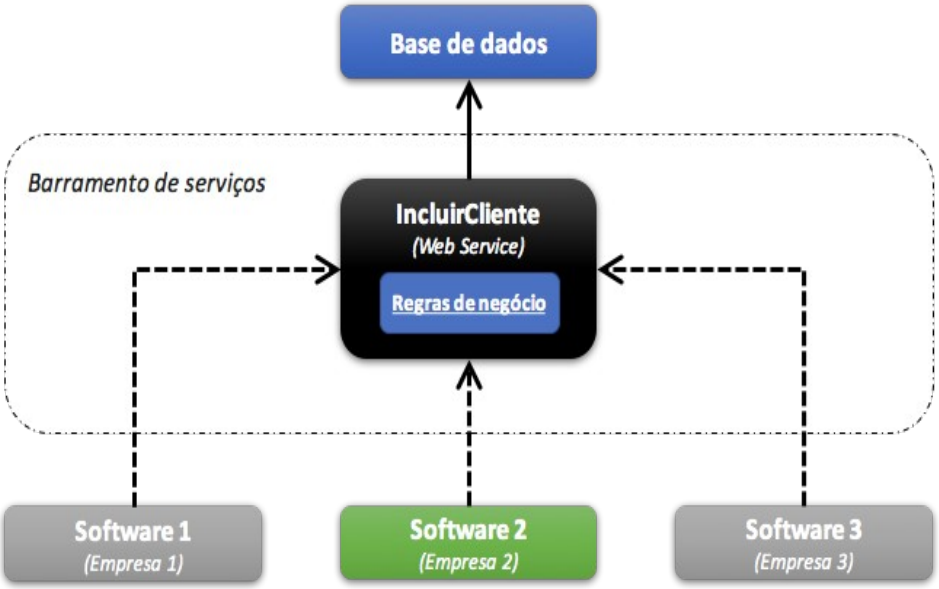
- Baixo acoplamento: os serviços devem ter o mínimo de dependências possíveis entre si, reduzindo o impacto de mudanças ou falhas em um serviço sobre os demais. Por exemplo, um serviço que faz a consulta de produtos em um catálogo deve retornar apenas os dados necessários para essa operação, sem incluir informações adicionais que possam afetar outros serviços.

- Modularidade: os serviços devem ser projetados para serem facilmente combinados em diferentes configurações, atendendo às necessidades do negócio. Por exemplo, um processo de negócio que faz a venda de produtos pode combinar vários serviços, como o serviço de consulta de produtos, o serviço de cálculo do frete, o serviço de pagamento e o serviço de emissão da nota fiscal.

Esses princípios estão diretamente relacionados com a capacidade de compartilhamento, combinação e flexibilidade oferecidos pela arquitetura SOA. Além disso, a padronização da interface dos serviços é um dos principais aspectos da arquitetura SOA, pois possibilita a interoperabilidade entre os serviços, independentemente da tecnologia ou plataforma utilizada.

Essa arquitetura é bastante utilizada para lidar com ambientes heterogêneos, pois permite que ativos de negócio conversem entre si.

Figura 2 – Exemplo de utilização da arquitetura SOA



Fonte: TREINA WEB (2023). Disponível em: https://www.treinaweb.com.br/blog/voce-sabe-o-que-e-arquitetura-orientada-a-servicos-soa/. Acesso em: 01 maio 2023.

A figura 2 é um exemplo de uma criação de uma web service chamado "IncluirCliente", que centraliza o ativo de negócio, garante a validação correta do cliente, possibilita a reutilização do serviço por outros softwares e a extensibilidade para futuras alterações. Esse serviço permite que as funcionalidades relacionadas à inclusão de um novo cliente sejam desacopladas em um serviço independente, tornando mais fácil a sua reutilização e adaptação às mudanças nos requisitos do negócio. Além disso, como o serviço segue os princípios da arquitetura SOA, ele pode ser facilmente descoberto e consumido por outros aplicativos, independentemente da tecnologia ou plataforma utilizada, o que traz ordem para o ambiente heterogêneo. O serviço "IncluirCliente" também é autônomo e modular, o que significa que ele pode ser desenvolvido, implantado e gerenciado independentemente de outros serviços, aumentando a disponibilidade e confiabilidade do aplicativo como um todo.

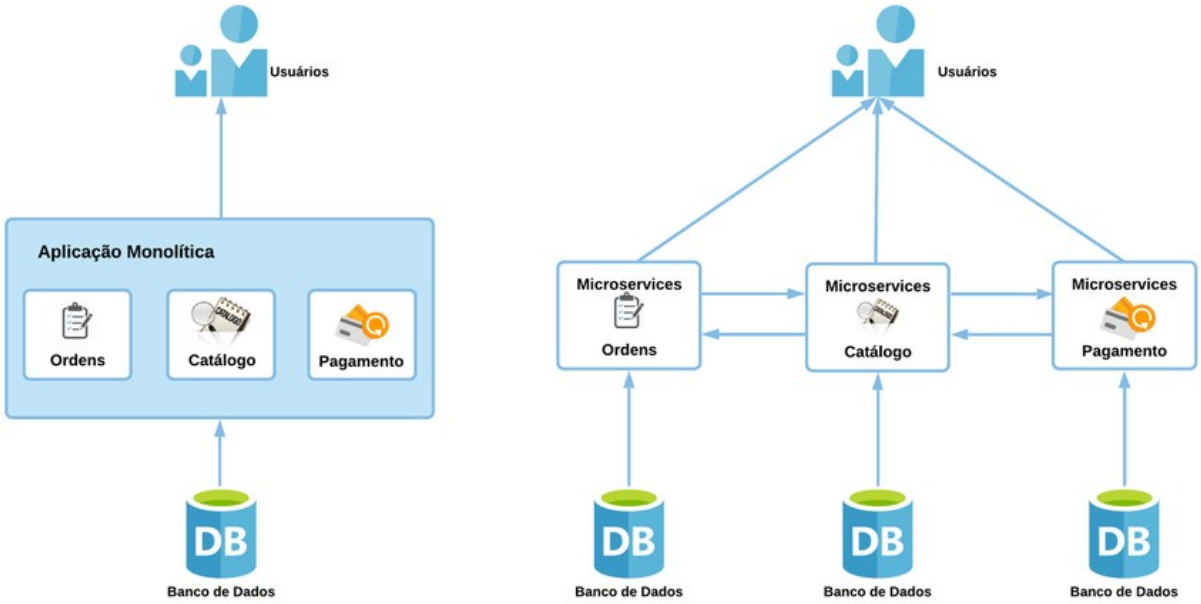
Em resumo, apresentamos os conceitos básicos do SOA, seus benefícios, princípios e desafios, bem como um exemplo de aplicação que usa essa arquitetura. Vimos que a SOA é uma forma de projetar software que facilita a reutilização e a interoperabilidade de componentes por meio de interfaces de serviço padronizadas e protocolos de comunicação comuns. A SOA também permite uma maior rapidez para entrada no mercado, uma manutenção mais eficiente e uma maior adaptabilidade às mudanças tecnológicas e aos requisitos de negócio.

3 MICROSSERVIÇOS

Com a evolução da computação em nuvem e o aumento da demanda por aplicações escaláveis, surgiu a necessidade de uma abordagem mais granular, que se refere à capacidade de dividir recursos e funcionalidades em componentes menores, independentes e flexíveis no desenvolvimento de sistemas. É neste contexto que os microsserviços surgem como uma alternativa viável para a construção de aplicações escaláveis e resilientes.

Microsserviços são estilo arquitetural que prega a divisão de um sistema em pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Estes serviços são construídos e implantados separadamente, permitindo que sejam escalados individualmente de acordo com a demanda, sem afetar o restante da aplicação. Além disso, como cada serviço é independente e pode ser desenvolvido em diferentes linguagens de programação e tecnologias, facilitando a adoção de novas tecnologias e a integração com outros sistemas.

Figura 3 – Diferença entre o modelo monolítico e a arquitetura de microsserviços



Fonte: RESEARCHGATE (2019). Disponível em: https://www.researchgate.net/figure/Figura-1-Diferenca-entre-o-modelo-monolitico-e-a-arquitetura-de-microsservicos_fig1_337636639l. Acesso em: 01 maio 2023.

Segundo Newman (2015), os microsserviços apresentam diversos benefícios, tais como escalabilidade granular, maior flexibilidade e facilidade de manutenção. Além disso, a arquitetura de microsserviços pode ser facilmente integrada com outras tecnologias, como contêineres e orquestradores de contêineres, permitindo uma gestão mais flexível e eficiente

dos serviços. É importante ressaltar que os contêineres e orquestradores de contêineres não são parte intrínseca da arquitetura de microsserviços, mas sim uma tecnologia que pode ser utilizada para oferecer suporte de infraestrutura aos serviços independentes e escaláveis.

Ao considerar a adoção de microsserviços, é importante ter em mente que essa abordagem arquitetural traz consigo desafios significativos. Por exemplo, a gestão de um grande número de serviços independentes pode ser complexa e exigir esforços adicionais de monitoramento e manutenção. Além disso, a comunicação entre esses serviços pode ser desafiadora e exigir um esforço maior na definição de interfaces claras e na adoção de práticas de comunicação assíncrona.

Por esse motivo, é fundamental que as equipes responsáveis pela adoção de microsserviços estejam cientes desses desafios e se preparem para enfrentá-los adequadamente. Isso pode incluir a adoção de ferramentas e práticas de gestão de serviços, além de investimentos em testes automatizados e outros processos de garantia de qualidade. Com a adoção de práticas recomendadas, é possível minimizar os desafios e obter os benefícios de uma arquitetura de microsserviços escalável e flexível.

Segundo Lewis e Fowler (2014), é importante adotar práticas de desenvolvimento e arquitetura que minimizem estes desafios, como o uso de interfaces bem definidas, testes automatizados e a adoção de um estilo de comunicação assíncrona.

Desta forma, o estilo arquitetural de microsserviços tem se mostrado uma abordagem altamente promissora para o desenvolvimento de aplicações escaláveis em nuvem. Essa abordagem arquitetural permite a criação de sistemas altamente flexíveis e resilientes, que podem ser escalados granularmente de acordo com a demanda do usuário. Além disso, a adoção de práticas recomendadas pode permitir que as equipes de desenvolvimento criem sistemas altamente integrados e interoperáveis.

Embora semelhante em muitos aspectos, a arquitetura de microsserviços difere da abordagem arquitetural de SOA (*Service-Oriented Architecture*). A principal diferença entre essas abordagens é que a arquitetura de microsserviços é mais focada na criação de serviços independentes e altamente escaláveis, enquanto a SOA é mais focada na integração de sistemas legados e de terceiros.

A principal vantagem da arquitetura de microsserviços é que ela permite a criação de sistemas altamente escaláveis e flexíveis, que podem ser facilmente adaptados às mudanças na demanda do usuário. Além disso, a adoção de práticas recomendadas pode permitir a criação de sistemas altamente interoperáveis e integrados, o que pode levar a uma maior eficiência e produtividade.

3.1 A importância da modularidade e da autonomia em microsserviços

Uma das principais referências no campo da arquitetura de microsserviços é Martin Fowler, que destaca a importância da modularidade para essa abordagem em seu livro "*Patterns of Enterprise Application Architecture*" (2002). Segundo Fowler, a modularidade é fundamental para permitir que os microsserviços sejam facilmente combinados e reutilizados em diferentes contextos.

Modularidade é a capacidade de dividir um sistema em partes independentes e interconectadas, chamadas de módulos, cada um com uma função específica. Esses módulos podem ser desenvolvidos, testados e implantados separadamente, o que permite que o sistema seja mais flexível, escalável e fácil de manter.

No contexto da arquitetura de microsserviços, a modularidade é especialmente importante, pois cada microsserviço é projetado para realizar uma função específica e é construído como um módulo independente. Dessa forma, os microsserviços podem ser facilmente combinados e reutilizados em diferentes contextos, permitindo a criação de sistemas complexos a partir de pequenas peças independentes.

A modularidade também pode ajudar a reduzir a complexidade geral do sistema, tornando mais fácil entender e modificar cada módulo individualmente, sem afetar outros módulos. Além disso, a modularidade permite que cada módulo seja atualizado separadamente, sem a necessidade de atualizar o sistema inteiro, o que pode ser muito útil em sistemas grandes e complexos.

Além disso, Sam Newman, em seu livro "*Building Microservices: Designing Fine-Grained Systems*" (2015), destaca a importância da autonomia dos serviços para a arquitetura de microsserviços. Segundo Newman, os serviços devem ser projetados de forma autônoma, com suas próprias bases de dados e interfaces bem definidas, de forma a minimizar o acoplamento entre os serviços e permitir que cada um possa evoluir de forma independente.

Autonomia é a capacidade de um serviço operar de forma independente, sem depender de outros serviços ou componentes do sistema para funcionar. Na arquitetura de microsserviços, a autonomia é uma característica muito importante, pois permite que os serviços sejam projetados e desenvolvidos como unidades independentes e autônomas.

Cada serviço deve ter sua própria base de dados e interface bem definida, de forma a minimizar o acoplamento entre os serviços e permitir que cada um possa evoluir de forma independente. Isso significa que cada serviço pode ser atualizado, testado e implantado sem

afetar outros serviços, o que aumenta a flexibilidade e a escalabilidade do sistema como um todo.

Além disso, a autonomia também permite que os serviços sejam distribuídos geograficamente, o que pode ajudar a reduzir a latência e melhorar a disponibilidade do sistema em diferentes regiões do mundo. Os serviços autônomos também podem ser escalados horizontalmente, o que significa que mais instâncias do serviço podem ser adicionadas à medida que a demanda aumenta, sem afetar outros serviços.

Em resumo, a autonomia é uma característica fundamental da arquitetura de microsserviços, permitindo que cada serviço seja projetado e desenvolvido de forma independente, escalável e flexível.

Dessa forma, a modularidade e a autonomia são elementos fundamentais para a arquitetura de microsserviços, como destacado por esses autores e referências importantes na área. A modularidade permite que os serviços sejam facilmente combinados, enquanto a autonomia permite que cada serviço possa evoluir de forma independente, sem comprometer a integridade da aplicação como um todo.

Para alcançar a modularidade e a autonomia, é importante que os serviços tenham uma interface bem definida, com uma documentação clara e uma versão específica. Isso permitirá que outros serviços possam se comunicar com ele de forma confiável e segura, mesmo que sejam desenvolvidos em tecnologias diferentes.

Outro aspecto importante da modularidade e da autonomia é a capacidade de versionar serviços independentemente uns dos outros. Com a modularidade, um serviço pode ser atualizado sem afetar o restante da aplicação. Isso permite que os serviços sejam evoluídos independentemente, sem precisar mudar toda a aplicação para acompanhar as mudanças.

No entanto, a modularidade e a autonomia trazem consigo novos desafios. Por exemplo, é importante garantir que os serviços sejam facilmente descobertos e que a comunicação entre eles seja confiável e segura. Além disso, a gestão de serviços individuais pode ser complexa e exigir esforços adicionais de monitoramento e manutenção.

Com base nas ideias de Sam Newman, em seu livro "*Building Microservices: Designing Fine-Grained Systems*", é possível identificar que uma das principais vantagens dos microsserviços é a possibilidade de atualizar e implantar serviços individualmente, sem afetar outros serviços em execução. Isso permite que as equipes responsáveis pelos serviços trabalhem de forma independente, sem a necessidade de coordenação com outras equipes para lançar uma nova versão da aplicação.

Outro autor que aborda o tema é Martin Fowler, em seu artigo "*Microservices*", onde ele destaca a importância da comunicação entre os serviços, sugerindo a utilização de uma abordagem de comunicação assíncrona, como a utilização de filas de mensagens. Essa abordagem ajuda a reduzir a dependência entre os serviços e permite que eles sejam escalados individualmente, de acordo com a demanda de cada serviço.

Com essas práticas recomendadas, é possível minimizar os desafios e obter os benefícios de uma arquitetura de microsserviços altamente modular e autônoma. No entanto, é importante destacar que a adoção de microsserviços não é uma solução para todos os casos de uso, e é necessário avaliar cuidadosamente as necessidades da aplicação e os requisitos de escalabilidade e manutenção antes de decidir pela adoção dessa arquitetura.

3.2 Desafios na adoção de microsserviços

A adoção de microsserviços é uma abordagem de arquitetura que tem ganhado destaque em muitas organizações atualmente. Os microsserviços consistem em um modelo de desenvolvimento em que um aplicativo é dividido em vários serviços independentes, que se comunicam por meio de APIs bem definidas. No entanto, apesar de suas vantagens, a adoção de microsserviços também apresenta desafios significativos que precisam ser considerados pelos profissionais de TI e líderes de negócios.

Complexidade da arquitetura: Ao adotar microsserviços, as organizações precisam lidar com uma arquitetura distribuída, o que pode aumentar a complexidade do sistema como um todo. Isso inclui desafios relacionados à comunicação entre os serviços, gerenciamento de dependências e orquestração dos diversos componentes.

Gerenciamento de dados: A divisão de um aplicativo em microsserviços independentes pode levar a questões de gerenciamento de dados. Cada serviço pode ter sua própria base de dados, o que torna a manutenção e consistência dos dados mais desafiadora, especialmente em cenários de transações distribuídas.

Monitoramento e depuração: Com a arquitetura distribuída dos microsserviços, o monitoramento e depuração de problemas podem se tornar mais complexos. É necessário implementar ferramentas e práticas adequadas para rastrear problemas em diferentes serviços e garantir a alta disponibilidade do sistema.

Requisitos de infraestrutura: A adoção de microsserviços pode exigir uma infraestrutura de TI mais robusta e escalável, especialmente em comparação com abordagens

monolíticas. Isso pode resultar em custos adicionais com hardware, rede e capacidade de processamento.

Desafios culturais e organizacionais: A mudança para microsserviços muitas vezes implica em uma transformação cultural e organizacional. As equipes precisam estar alinhadas e colaborar efetivamente para que a estratégia de adoção seja bem-sucedida.

Segurança e conformidade: A segurança é uma preocupação essencial em qualquer arquitetura de TI, e com os microsserviços não é diferente. É importante garantir que a comunicação entre os serviços seja segura e que os dados estejam protegidos. Além disso, a adoção de microsserviços pode ter implicações em conformidade com regulamentações específicas, como a proteção de dados.

Aprendizado e capacitação: A equipe de desenvolvimento precisa adquirir habilidades específicas para projetar, desenvolver e manter microsserviços. Investir em treinamentos e capacitação é essencial para o sucesso da adoção dessa arquitetura.

Performance e latência: Embora a arquitetura de microsserviços possa trazer benefícios em termos de escalabilidade, ela também pode introduzir latência em algumas operações devido à comunicação entre os serviços. É necessário otimizar o desempenho e garantir que a latência não afete negativamente a experiência do usuário.

Testes e garantia de qualidade: Com a divisão do aplicativo em serviços menores e independentes, os testes se tornam mais complexos. Garantir a qualidade e a integração adequada entre os serviços é um desafio a ser superado.

Versionamento e gerenciamento de mudanças: A evolução e o versionamento dos microsserviços precisam ser bem planejados e executados para evitar problemas de compatibilidade entre diferentes versões.

Em suma, a adoção de microsserviços pode trazer benefícios significativos em termos de escalabilidade, agilidade e flexibilidade, mas também implica em desafios técnicos, culturais e organizacionais que devem ser considerados cuidadosamente para garantir o sucesso da transição.

3.3 Vantagens da arquitetura de microsserviços

Embora a arquitetura de microsserviços apresenta desafios, ela também oferece várias vantagens significativas para as empresas que a adotam. Uma das principais vantagens é a escalabilidade, que permite que as empresas aumentem ou diminuam a capacidade dos serviços de forma independente. Isso significa que, se um serviço estiver sobrecarregado, ele

pode ser escalonado rapidamente para lidar com a demanda adicional, sem afetar os demais serviços.

A arquitetura de microsserviços também oferece vantagens em termos de qualidade do software. Como cada serviço é independente, é mais fácil testar e depurar cada módulo separadamente, o que reduz o risco de erros e falhas no software final. Além disso, a modularidade da arquitetura permite que os desenvolvedores trabalhem em paralelo em diferentes módulos, o que acelera o processo de desenvolvimento e reduz o tempo de entrega do software.

Além disso, a arquitetura de microsserviços permite uma maior flexibilidade no desenvolvimento e implantação de novos recursos e funcionalidades. Como cada serviço é autônomo, os desenvolvedores podem trabalhar em novas funcionalidades sem afetar o restante do sistema. Isso permite uma maior agilidade e eficiência no desenvolvimento e implantação de software.

Outra vantagem é a resiliência da arquitetura de microsserviços. Como cada serviço é autônomo, se um serviço falhar, isso não afetará os demais serviços. Além disso, os microsserviços podem ser facilmente reimplantados em outro ambiente caso ocorra uma falha, o que aumenta a disponibilidade dos serviços.

A arquitetura de microsserviços também promove uma maior especialização das equipes de desenvolvimento. Como cada serviço é desenvolvido e implantado de forma autônoma, as equipes podem se especializar em áreas específicas do sistema, o que leva a uma maior eficiência e qualidade do software.

Por fim, a arquitetura de microsserviços é altamente compatível com a abordagem DevOps, que enfatiza a colaboração e integração contínua entre as equipes de desenvolvimento e operações. Os microsserviços podem ser facilmente integrados em pipelines de entrega contínua, permitindo que as equipes implementem, testem e implantem novos serviços com rapidez e eficiência.

A arquitetura de microsserviços também pode melhorar a modularidade do software, permitindo que diferentes serviços possam ser desenvolvidos e mantidos separadamente. Isso pode levar a uma maior reutilização de código e a uma maior facilidade na manutenção do software.

Além disso, a arquitetura de microsserviços pode levar a uma maior escalabilidade e flexibilidade em ambientes de nuvem, permitindo que as empresas se adaptem facilmente a mudanças na demanda do usuário e garantindo uma alta disponibilidade dos serviços.

Por fim, microsserviços também podem melhorar a segurança do software, permitindo uma maior separação de responsabilidades e reduzindo a superfície de ataque do sistema.

Em resumo, a adoção da arquitetura de microsserviços apresenta desafios significativos, mas também oferece várias vantagens importantes para as empresas que a adotam. Com a implementação de práticas recomendadas e o compromisso de recursos necessários, as empresas podem colher os benefícios dessa arquitetura moderna e se manterem competitivas no mercado.

3.4 Desvantagens da arquitetura de microsserviços

Embora a arquitetura de microsserviços apresenta diversas vantagens, é importante destacar que ela também tem suas desvantagens e limitações.

Uma das principais desvantagens é a complexidade da infraestrutura necessária para suportar microsserviços em grande escala. Isso inclui a necessidade de gerenciar bancos de dados distribuídos, a configuração de balanceadores de carga e a implementação de protocolos de comunicação complexos, como mencionado anteriormente. Essa complexidade pode tornar a adoção de microsserviços um desafio para empresas com recursos limitados ou equipes de desenvolvimento inexperientes.

Outro desafio é a necessidade de uma coordenação mais estreita entre as equipes responsáveis pelos diferentes serviços. Embora a autonomia dos serviços seja uma vantagem, também pode levar a uma falta de alinhamento entre as equipes e uma dificuldade em manter interfaces consistentes entre os serviços. Isso pode resultar em conflitos e atrasos na implantação de novos recursos e funcionalidades.

A adoção de microsserviços também pode levar a um aumento na complexidade da segurança. Como cada serviço é autônomo, é necessário garantir que a comunicação entre eles seja segura e que a autenticação e autorização sejam implementadas corretamente. Além disso, a necessidade de gerenciar várias instâncias do mesmo serviço pode tornar a implementação de políticas de segurança mais desafiadoras.

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

Outra desvantagem é o aumento da latência devido à comunicação entre serviços. Embora a escalabilidade dos microsserviços permita que os serviços sejam escalados independentemente, a comunicação entre os serviços pode levar a um aumento da latência, especialmente em ambientes distribuídos. Isso pode afetar negativamente a experiência do usuário e a eficiência do sistema.

Por fim, é importante mencionar que a adoção de microsserviços requer um investimento significativo em termos de recursos e mudanças culturais. A mudança para uma arquitetura de microsserviços pode exigir uma mudança significativa na cultura organizacional, com uma maior ênfase na colaboração e na integração contínua. Além disso, a adoção de microsserviços pode exigir a implementação de ferramentas de monitoramento e gerenciamento adicionais, aumentando ainda mais os custos e a complexidade.

Em suma, é importante considerar as desvantagens e limitações da arquitetura de microsserviços antes de adotá-la. Como destacado pelos autores mencionados, a complexidade da infraestrutura, a coordenação entre equipes, a complexidade da segurança, o aumento da latência e a falta de padronização e complexidade do desenvolvimento são alguns dos desafios a serem enfrentados. Portanto, é essencial avaliar cuidadosamente as necessidades e recursos da empresa antes de fazer a transição para uma arquitetura de microsserviços.

3.5 Contêineres

Os contêineres são um tipo de virtualização em que cada aplicação é executada em um ambiente isolado e independente, garantindo maior segurança, portabilidade e flexibilidade. Eles são uma solução para resolver o problema de dependências de bibliotecas e ambientes de execução que podem afetar o desempenho e a escalabilidade da aplicação.

Os contêineres são executados em cima de um sistema operacional hospedeiro e compartilham o kernel do sistema, tornando-os mais leves e eficientes em relação à virtualização tradicional. Com isso, é possível empacotar uma aplicação e suas dependências em um contêiner e movê-lo facilmente entre diferentes ambientes, desde o desenvolvimento até a produção.

A utilização de contêineres possibilita o desenvolvimento de aplicações escaláveis e altamente disponíveis na nuvem. Além disso, essa tecnologia oferece vantagens como simplificação do processo de implantação e gerenciamento de aplicações, maior flexibilidade

e portabilidade, além de reforçar a segurança e o isolamento de recursos, resultando na redução de custos de infraestrutura e manutenção.

Outra vantagem dos contêineres é a capacidade de orquestrar e gerenciar múltiplos contêineres em larga escala. Por meio de ferramentas como Kubernetes, Docker e outras, torna-se viável automatizar o ciclo de implantação, escalonamento e monitoramento em ambientes com um grande número de contêineres.

A utilização conjunta de contêineres e a arquitetura de microsserviços estão emergindo como uma tendência no desenvolvimento de aplicações em nuvem. Essa abordagem permite que a aplicação seja dividida em módulos independentes e altamente escaláveis, facilitando tanto a manutenção quanto o desenvolvimento contínuo.

Assim, a combinação sinérgica de contêineres e microsserviços oferece uma solução viável para a criação de aplicações escaláveis e altamente disponíveis na nuvem. Isso proporciona maior flexibilidade, portabilidade e eficiência no gerenciamento de recursos.

3.6 Orquestração de contêineres

A orquestração de contêineres é uma das principais soluções para gerenciar aplicativos em ambientes distribuídos. Como os aplicativos modernos se tornaram cada vez mais complexos e distribuídos, o gerenciamento manual de contêineres tornou-se inviável e ineficiente.

A orquestração de contêineres oferece uma maneira mais eficiente e automatizada de gerenciar o ciclo de vida de contêineres em um ambiente distribuído. Ela permite que os desenvolvedores e operadores gerenciem aplicativos de forma mais fácil e eficiente, realizando tarefas como implantação, atualização, escalonamento e exclusão de contêineres conforme necessário.

Ao lidar com a complexidade de aplicativos distribuídos em escala, a orquestração de contêineres permite que as equipes de desenvolvimento e operações trabalhem juntas de forma mais eficaz. Ela permite que os desenvolvedores criem aplicativos com mais flexibilidade, permitindo que sejam implantados em um ambiente distribuído com mais facilidade e eficiência.

Além disso, a orquestração de contêineres é particularmente útil para aplicativos que precisam ser escalados rapidamente ou reduzidos em escala, conforme necessário. Isso permite que as equipes de operações respondam às mudanças nas demandas de aplicativos, garantindo que os usuários finais recebam o melhor desempenho possível.

Em resumo, a orquestração de contêineres é uma tecnologia essencial para gerenciar aplicativos distribuídos em escala. Ela permite que as equipes de desenvolvimento e operações trabalhem juntas de forma mais eficaz, permitindo que os aplicativos sejam implantados, atualizados e escalados com mais facilidade e eficiência.

A orquestração de contêineres emergiu como o método convencional para implementar aplicações em ambientes distribuídos, assegurando tanto eficiência quanto escalabilidade. Mediante a automatização e a adaptabilidade inerentes à orquestração, times de desenvolvimento e operações podem agora cooperar de maneira otimizada, garantindo a gestão simplificada e eficaz de aplicações em um contexto global cada vez mais intrincado e descentralizado.

4 DESAFIOS PARA O DESENVOLVIMENTO DE APLICAÇÕES ESCALÁVEIS EM NUVEM

O desenvolvimento de aplicações escaláveis em nuvem apresenta desafios significativos devido à natureza distribuída e dinâmica da nuvem, bem como às crescentes expectativas dos usuários em relação à escalabilidade e confiabilidade das aplicações. Como destaca a Microsoft Azure, é crucial garantir que os aplicativos possam lidar com fatores como escalabilidade e disponibilidade, proporcionando um serviço consistente a vários clientes, fornecendo dados em tempo real globalmente e incorporando inteligência artificial nos aplicativos.

Um dos desafios principais é projetar a arquitetura da aplicação de forma que possa escalar horizontalmente, ou seja, adicionar recursos de forma dinâmica à medida que a demanda aumenta. Isso requer o uso de técnicas como a divisão da aplicação em componentes independentes e escaláveis, o uso de serviços de armazenamento e processamento distribuídos e a implementação de mecanismos de balanceamento de carga para distribuir eficientemente as solicitações entre os recursos disponíveis.

Além disso, a natureza distribuída da nuvem introduz desafios adicionais na comunicação e coordenação entre os diferentes componentes da aplicação. É necessário garantir que os dados sejam consistentes e que as operações possam ser executadas em paralelo de maneira eficiente. Isso pode ser alcançado por meio do uso de técnicas como o armazenamento distribuído e a adoção de padrões de comunicação assíncrona.

Outro desafio importante é lidar com a escalabilidade elástica, ou seja, a capacidade da aplicação de se adaptar rapidamente às variações de carga. Isso requer o uso de mecanismos automatizados para detectar alterações na demanda e ajustar dinamicamente os recursos alocados, como a escalabilidade automática de instâncias de máquinas virtuais ou o provisionamento de contêineres.

Além disso, a confiabilidade da aplicação também é fundamental. Falhas de componentes individuais ou interrupções na infraestrutura da nuvem podem ocorrer a qualquer momento, e é necessário projetar a aplicação para ser tolerante a falhas e capaz de se recuperar automaticamente. Isso pode envolver a replicação de dados e serviços em diferentes regiões da nuvem, o monitoramento constante da saúde dos componentes da aplicação e a implementação de estratégias de recuperação rápida.

Em suma, o desenvolvimento de aplicações escaláveis em nuvem exige uma abordagem cuidadosa para lidar com os desafios impostos pela natureza distribuída e

dinâmica da nuvem. É necessário projetar arquiteturas escaláveis e distribuídas, implementar mecanismos de balanceamento de carga e escalabilidade elástica, garantir a consistência dos dados e a tolerância a falhas, e adotar práticas de monitoramento e recuperação para manter a confiabilidade e o desempenho das aplicações em nuvem.

Alguns dos principais desafios encontrados no desenvolvimento de aplicações escaláveis em nuvem são discutidos a seguir.

4.1 Gerenciamento de dados

O gerenciamento de dados em nuvem requer grandes mudanças nos sistemas de gerenciamento de dados, pois estes sistemas precisam ser escaláveis, disponíveis, ter bom desempenho e serem acessíveis em termos de custo. Isso ocorre porque as infraestruturas, plataformas e software são oferecidos como serviços em ambientes de computação em nuvem, onde empresas e usuários podem alugar capacidade de computação e armazenamento de forma transparente e sob demanda.

O gerenciamento de dados em nuvem apresenta desafios únicos em relação aos sistemas tradicionais de gerenciamento de dados. Por exemplo, a escalabilidade é um requisito fundamental em ambientes de nuvem, onde a quantidade de dados pode crescer rapidamente e de forma imprevisível. Além disso, a disponibilidade e o desempenho são críticos para garantir que os usuários possam acessar os dados de forma rápida e confiável. Por fim, o custo é um fator importante a ser considerado, já que o armazenamento e processamento de dados em nuvem pode ser caro, dependendo do volume de dados e do tipo de serviço utilizado.

Para enfrentar esses desafios, os sistemas de gerenciamento de dados em nuvem utilizam uma variedade de técnicas e tecnologias, como particionamento de dados, replicação, balanceamento de carga, cache e indexação. Além disso, os provedores de serviços em nuvem oferecem uma ampla gama de serviços de armazenamento e processamento de dados, como bancos de dados relacionais e não relacionais, sistemas de arquivos distribuídos, serviços de análise de dados e ferramentas de gerenciamento de dados. Com essas tecnologias e práticas, é possível gerenciar grandes volumes de dados em ambientes de nuvem de forma eficiente e escalável.

À medida que a tecnologia continua a evoluir, é esperado que o gerenciamento de dados em nuvem continue a se aprimorar, abrindo novas possibilidades para a análise de dados em tempo real, aprendizado de máquina e inteligência artificial. Com a convergência de

dados provenientes de várias fontes e a capacidade de processamento cada vez maior, as empresas estão se capacitando para tomar decisões mais informadas e estratégicas. No entanto, os desafios relacionados à segurança cibernética, privacidade dos dados e otimização de custos permanecerão como áreas de foco contínuo. Ao enfrentar essas questões de maneira proativa e colaborativa, a comunidade de TI está moldando um futuro onde o gerenciamento de dados em nuvem se tornará não apenas uma prática padrão, mas também uma pedra angular da transformação digital em todos os setores da sociedade.

4.2 Gerenciamento de recursos

O gerenciamento de recursos em nuvem é um conjunto de técnicas e práticas que permitem alocar e gerenciar recursos de computação, como CPU, memória, armazenamento e rede, de forma eficiente e escalável em ambientes de nuvem. A administração de recursos em ambientes de nuvem constitui-se como uma das principais complexidades, uma vez que a atribuição de recursos necessita ocorrer de maneira automática e dinâmica, alinhada à demanda dos utilizadores.

Para enfrentar esse desafio, os provedores de serviços em nuvem utilizam uma variedade de técnicas e tecnologias, como virtualização, balanceamento de carga, escalonamento automático e gerenciamento de energia. A virtualização permite que os recursos físicos sejam divididos em recursos virtuais, que podem ser alocados e desalocados de forma dinâmica, de acordo com a demanda dos usuários. O balanceamento de carga permite que os recursos sejam distribuídos de forma equilibrada entre os usuários, para evitar sobrecarga em alguns recursos e ociosidade em outros. O escalonamento automático permite que os recursos sejam alocados e desalocados automaticamente, de acordo com a demanda dos usuários. O gerenciamento de energia permite que os recursos sejam desligados ou colocados em modo de espera quando não estão sendo utilizados, para economizar energia.

O gerenciamento de recursos em nuvem é um dos principais desafios em ambientes de nuvem, já que a alocação de recursos deve ser feita de forma dinâmica e automática, de acordo com a demanda dos usuários. Para enfrentar esse desafio, os provedores de serviços em nuvem utilizam uma variedade de técnicas e tecnologias, como virtualização, balanceamento de carga, escalonamento automático e gerenciamento de energia. Com essas técnicas e tecnologias, é possível gerenciar recursos de computação em ambientes de nuvem de forma eficiente e escalável, garantindo que os usuários tenham acesso aos recursos de que

precisam, quando precisam, e que os recursos sejam utilizados de forma otimizada, sem desperdício de recursos ou energia.

Com essas técnicas e tecnologias, é possível gerenciar recursos de computação em ambientes de nuvem de forma eficiente e escalável, garantindo que os usuários tenham acesso aos recursos de que precisam, quando precisam, e que os recursos sejam utilizados de forma otimizada, sem desperdício de recursos ou energia.

4.3 Garantia de qualidade

A garantia de qualidade em ambientes de computação em nuvem é uma característica fundamental para garantir que os usuários tenham acesso a serviços de alta qualidade e confiabilidade. A qualidade do serviço em nuvem é definida entre o provedor e o usuário e expressa por meio de SLA (*Service Level Agreement*). Para garantir a qualidade do serviço em nuvem, é necessário utilizar técnicas e tecnologias que permitam monitorar e gerenciar o desempenho e a disponibilidade dos serviços em tempo real. Além disso, é importante utilizar técnicas adaptativas e dinâmicas para tornar os SGBDs em nuvem viáveis, e desenvolver ferramentas e processos que automatizem tarefas, como a alocação de recursos, para garantir que os recursos sejam utilizados de forma otimizada e eficiente, sem desperdício de recursos ou energia.

A garantia de qualidade em aplicações escaláveis em nuvem envolve testes funcionais, de integração e de carga. Testes funcionais são realizados para verificar se a aplicação está de acordo com os requisitos especificados. Testes de integração são realizados para verificar se os diferentes componentes da aplicação estão funcionando corretamente juntos. Testes de carga são realizados para verificar se a aplicação pode lidar com a carga esperada.

No entanto, realizar testes de qualidade em aplicações escaláveis em nuvem pode ser um desafio devido à natureza distribuída da arquitetura. Os testes precisam ser realizados em diferentes componentes distribuídos em diferentes nós da nuvem, o que torna o processo mais complexo e difícil de gerenciar.

Além disso, a garantia de qualidade deve ser contínua ao longo do ciclo de vida da aplicação. Com a adoção de metodologias ágeis e DevOps, as aplicações são atualizadas com frequência, o que exige que a garantia de qualidade seja realizada constantemente.

Para superar esses desafios, é necessário adotar ferramentas e práticas que facilitem a garantia de qualidade em aplicações escaláveis em nuvem. Ferramentas de automação de testes, como GitHub Actions, podem ser utilizadas para simplificar a execução de testes

funcionais e de carga. Práticas como integração contínua e entrega contínua (CI/CD) também podem ajudar a garantir a qualidade da aplicação ao longo de todo o ciclo de vida.

Outra prática importante é a monitoração da aplicação em tempo real. Com a adoção de arquiteturas distribuídas, é essencial monitorar o desempenho e a disponibilidade dos diferentes componentes da aplicação. Isso pode ser feito utilizando ferramentas de monitoração, como o Prometheus e o Grafana.

A garantia de qualidade é um desafio constante no desenvolvimento de aplicações escaláveis em nuvem. No entanto, adotar as práticas e ferramentas corretas pode ajudar a superar esses desafios e garantir que a aplicação seja confiável e de alta qualidade.

5 MÉTODOS E FERRAMENTAS

No contexto atual de desenvolvimento de software, a arquitetura de microsserviços tem se mostrado uma abordagem eficiente para lidar com demandas complexas e de escala global. A abordagem de microsserviços permite a criação de sistemas distribuídos compostos por pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Esses serviços podem ser implantados e escalados de forma independente, proporcionando uma maior agilidade no desenvolvimento, manutenção e gerenciamento de sistemas complexos.

Neste capítulo, apresentaremos o desenvolvimento de uma aplicação para venda de livros online de alcance mundial, explorando o uso de microsserviços. O desenvolvimento de uma aplicação escalável baseada em microsserviços leva em consideração os requisitos de alto desempenho e escalabilidade exigidos por um ambiente global.

Durante o desenvolvimento deste capítulo, serão explorados métodos, tecnologias e práticas necessárias para a implementação dos microsserviços, como linguagem de programação, frameworks, comunicação entre os serviços, arquitetura e gerenciamento de dados.

5.1 Caso de uso

No contexto dos microsserviços, o estudo de caso selecionado para este projeto envolve o desenvolvimento de um sistema para uma livraria online que possui alcance mundial. Essa livraria trabalha com a venda de livros para vários países e possui um sistema de câmbio para converter para a moeda local. Alto desempenho e escalabilidade são demandas cruciais para lidar com um cenário em escala mundial.

A fim de atender a essas necessidades, é adotado o modelo arquitetural de microsserviços. Tal modelo permite a criação de um sistema modular, no qual cada funcionalidade é implementada como um serviço independente, o que facilita o gerenciamento e a escalabilidade. Além disso, os microsserviços podem ser desenvolvidos e implantados de forma autônoma, o que agiliza o processo de atualização e manutenção do sistema como um todo.

Com essa abordagem, a livraria online poderá lidar com o grande volume de transações e demandas de diferentes países, garantindo uma experiência de compra eficiente e adaptada às preferências locais dos clientes. O modelo de microsserviços proporciona

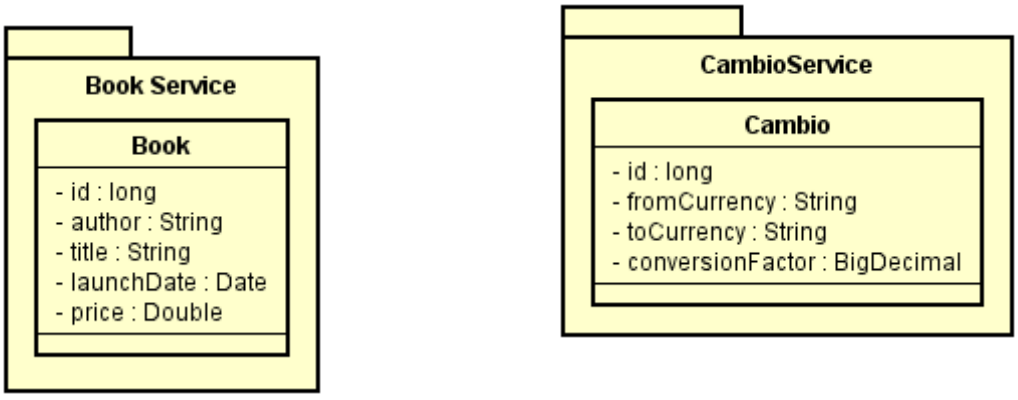
flexibilidade e agilidade no desenvolvimento e permite que a livraria se mantenha competitiva em um mercado global cada vez mais exigente.

5.2 Arquitetura do sistema

A arquitetura do sistema da livraria online é baseada em microsserviços, o que permite uma estrutura modular e escalável para lidar com as demandas de desempenho e escalabilidade em um ambiente de alcance mundial.

Dentro dessa arquitetura, o sistema da livraria é composto por dois microsserviços independentes que desempenham funções específicas. Para fins de demonstração dos requisitos de desempenho e escalabilidade, dois serviços foram implementados. O "Book Service" é responsável por fornecer informações detalhadas sobre os livros como autor, título, data de lançamento do livro e preço, enquanto o "Câmbio Service" lida com as operações de câmbio como a moeda de origem, moeda de destino, fator de conversão e convertendo o preço dos livros para a moeda local do cliente.

Figura 4 - Diagrama UML: Estrutura do serviço Book Service está localizado à esquerda e o Câmbio Service à direita

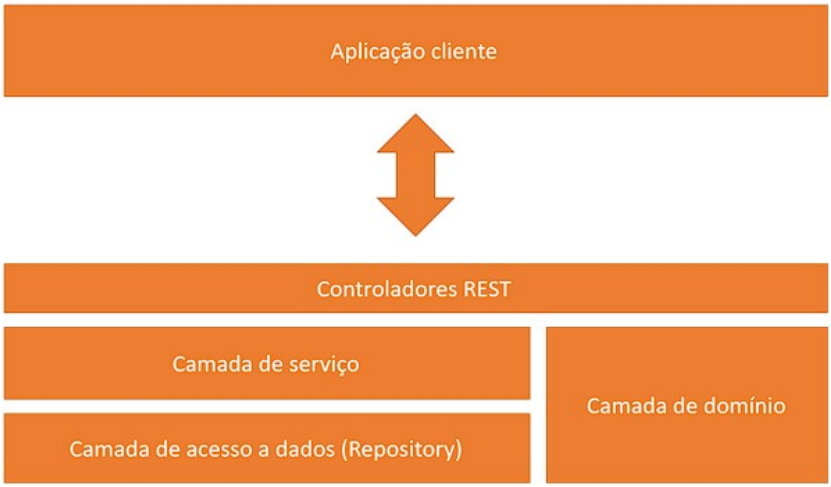


Fonte: O autor (2023).

A adoção de microsserviços também se justifica pelo ganho de flexibilidade no sistema que se refere à capacidade de desenvolver e gerenciar cada micro serviço independentemente. Isso significa que os microsserviços podem ser implantados, dimensionados e atualizados separadamente, facilitando a manutenção e evolução da

plataforma como um todo continuamente. Dessa forma, é possível adicionar novos serviços ou modificar os existentes sem impactar todo o sistema.

Figura 5 – Diagrama MVC: Estrutura da aplicação dentro do Book Service e Cambio Service



Fonte: O autor (2023).

A Figura 5 mostra a arquitetura do sistema Book Service e Cambio Service, dividido em várias camadas. Cada camada desempenha um papel específico no processamento e gerenciamento dos dados.

Controladores REST: Essa camada representa a interface de programação de aplicativos (API) REST, que permite a comunicação e interação com o aplicativo por meio de solicitações HTTP. Os controladores REST recebem as solicitações dos clientes, processam-nas e enviam as respostas apropriadas.

Camada de Serviços: A camada de serviços contém a lógica de negócios e é responsável por orquestrar as operações e a lógica do aplicativo. Ela recebe as solicitações dos controladores REST e coordena as chamadas à camada de domínio para realizar as operações necessárias.

Camada de Domínio: Essa camada representa a parte central do aplicativo, onde reside a lógica de negócios principal. Um exemplo, a camada de domínio é representada pelo objeto "book" (livro). Ela contém as regras de negócio relacionadas a livros, como validações, cálculos e manipulações dos dados. A camada de domínio não deve depender de outras camadas e deve ser independente da tecnologia.

Camada de Acesso a Dados (*Repository*): Essa camada é responsável pelo acesso e manipulação dos dados persistentes, geralmente em um banco de dados. Ela fornece uma interface para recuperar, inserir, atualizar e excluir dados do sistema. No contexto da Figura 5,

o repositório (ou camada de acesso a dados) permite o armazenamento e recuperação de informações sobre livros.

Essa arquitetura em camadas segue o princípio de separação de papeis/responsabilidades (SoC - *Separation of Concerns*), onde cada camada possui uma responsabilidade claramente definida e não se sobrepõe. Isso permite uma melhor organização e manutenção do código, além de facilitar a escalabilidade e a reutilização de componentes.

Em resumo, a adoção da arquitetura de microsserviços permite a flexibilidade, escalabilidade e evolução do sistema para enfrentar os desafios de aplicações escaláveis em nuvens.

Ao decidir pela adoção de microsserviços, é fundamental levar em consideração os requisitos e características específicas do projeto, como a demanda por alto desempenho e escalabilidade mencionada anteriormente. Avaliar os benefícios em relação à complexidade e ao custo de desenvolvimento e manutenção é essencial.

Em alguns casos, pode ser mais eficiente implementar sistemas seguindo o modelo monolítico, especialmente em aplicações menores que não exigem escalabilidade. Essa abordagem simplificada evita a introdução de complexidades desnecessárias e reduz a sobrecarga de gerenciamento.

No entanto, na situação apresentada do estudo de caso da livraria online, a abordagem de microsserviços é adequada. A necessidade de atender a uma base de clientes global e oferecer informações detalhadas sobre os livros, juntamente com a conversão de preços em moedas locais, justifica a adoção dessa arquitetura.

Cada caso deve ser analisado individualmente, ponderando os benefícios em relação à complexidade e ao custo, a fim de determinar se a abordagem de microsserviços trará vantagens significativas.

5.2.1 Componentes do sistema

A arquitetura do sistema é composta pelos seguintes componentes:

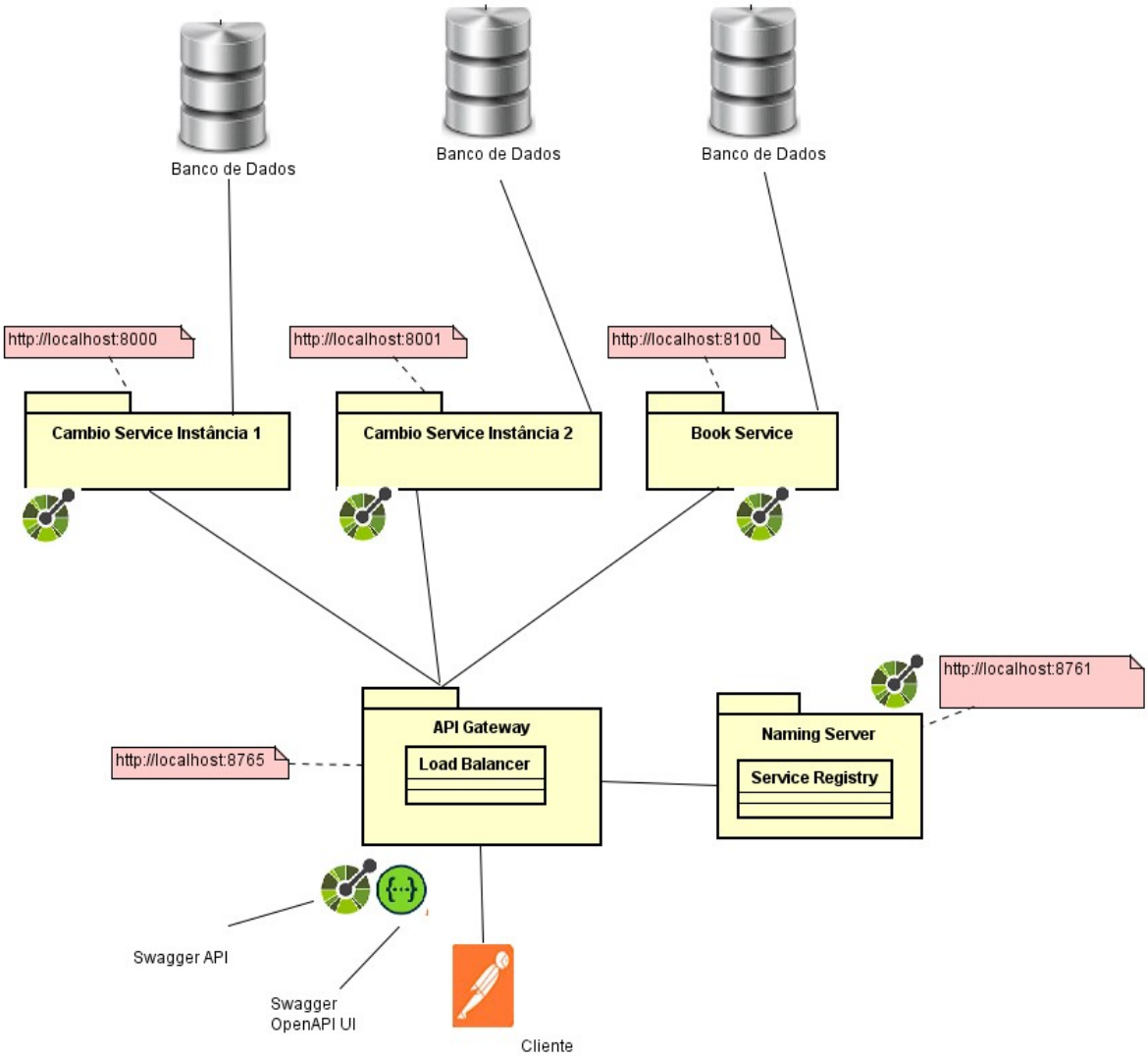
- **Microsserviço Book Service**: Este microsserviço é responsável por fornecer informações detalhadas sobre os livros disponíveis na livraria. Ele recebe solicitações para informações sobre autores, títulos, datas de lançamento e preços dos livros.
- **Microsserviço Câmbio Service**: O Microsserviço Câmbio Service lida com as operações de câmbio para converter os preços dos livros para a moeda local do cliente.

Ele recebe as solicitações contendo a moeda de origem, moeda de destino e fator de conversão, e retorna o preço convertido.

- **Banco de Dados**: O banco de dados é responsável pelo armazenamento dos dados dos livros, incluindo informações sobre autores, títulos, datas de lançamento e preços. Ele também armazena as taxas de câmbio atualizadas para permitir as conversões de moeda.
- **Gateway**: O Gateway é o ponto de entrada da aplicação e atua como um roteador para as solicitações dos clientes. Ele redireciona as requisições para os microsserviços apropriados com base nos endpoints solicitados.
- **Naming Server**: O Naming Server é responsável por permitir a descoberta dinâmica dos microsserviços dentro da arquitetura. Ele atua como um registro centralizado onde os microsserviços se registram e fornecem suas informações de localização. Isso permite que os microsserviços sejam escalados horizontalmente e sejam adicionados ou removidos dinamicamente sem afetar a funcionalidade do sistema.
- **Load Balancer**: O Load Balancer é um componente que distribui as solicitações de entrada entre várias instâncias de microsserviços em execução. Ele ajuda a balancear a carga de trabalho e garantir que cada instância do microsserviço receba uma quantidade equilibrada de solicitações, melhorando o desempenho e a escalabilidade do sistema.
- **Service Registry**: O Service Registry é um repositório no Naming Server que mantém o registro das informações de localização de cada microsserviço. Ele permite que os microsserviços se registrem e atualizem suas informações, como endereço IP e porta, para que outros componentes, como o Load Balancer, possam descobrir e rotear as solicitações corretamente.

Esses componentes adicionais, o Naming Server, Load Balancer e Service Registry, são cruciais para a efetiva implementação da arquitetura de microsserviços, fornecendo a capacidade de descoberta dinâmica, balanceamento de carga e gerenciamento de localização dos microsserviços na livraria online.

Figura 6 – Arquitetura do sistema da livraria online baseada em microsserviços.



Fonte: O autor (2023).

A Figura 6 fornecida ilustra a livraria online com dois microsserviços independentes ("Book Service" e "Câmbio Service"). Caso seja necessário escalar, uma nova instância é criada, como mostrado na Figura 6, o "Câmbio Service Instância 2". Além de um API Gateway com load balancer e um Naming Server com service registry, a comunicação entre os componentes pode ser organizada da seguinte forma:

1. Microsserviços e API Gateway:

- O "Book Service" e o "Câmbio Service" seriam implementados como microsserviços independentes, cada um com sua própria lógica de negócio e banco de dados.

- O "Book Service" seria responsável por fornecer informações detalhadas sobre os livros, como autor, título, data de lançamento e preço. Ele poderia expor endpoints RESTful para operações como consultar informações de um livro específico ou listar todos os livros disponíveis.
- O "Câmbio Service" seria responsável pelas operações de câmbio, convertendo o preço dos livros para a moeda local do cliente. Ele também poderia expor endpoints RESTful para conversões de moeda.
- O API Gateway atuaria como um ponto de entrada único para os clientes da livraria online. Ele receberia as solicitações dos clientes e rotearia as solicitações para os microsserviços correspondentes. O API Gateway também poderia lidar com autenticação, autorização, cache de dados e outras funcionalidades comuns.

2. Naming Server e Service Registry:

- O Naming Server, também conhecido como Service Registry, é responsável por registrar e rastrear os microsserviços disponíveis no sistema. Ele permite que os microsserviços se registrem quando iniciam e se retirem quando param.
- Quando um microsserviço é iniciado, ele registraria suas informações (por exemplo, seu nome, endereço IP, porta e endpoints) no Naming Server.
- O API Gateway consultaria o Naming Server para descobrir as instâncias disponíveis de cada microsserviço. Isso permite que o API Gateway encaminhe as solicitações de forma dinâmica e balanceie a carga entre as instâncias disponíveis.

3. Comunicação entre os componentes:

- Para a comunicação entre o API Gateway, o "Book Service" e o "Câmbio Service", pode-se utilizar chamadas de API RESTful. O API Gateway encaminharia as solicitações dos clientes para os microsserviços correspondentes, utilizando o serviço de balanceamento de carga para distribuir as solicitações entre as instâncias disponíveis.
- O API Gateway pode adicionar informações extras nas solicitações, como autenticação e informações de contexto, antes de encaminhá-las para os microsserviços. Os microsserviços, por sua vez, processariam as solicitações e retornariam as respostas adequadas ao API Gateway, que as encaminharia de volta aos clientes.

Em resumo, o API Gateway com load balancer e o Naming Server com service registry ajudam a orquestrar e direcionar as solicitações entre os microsserviços, proporcionando escalabilidade, desempenho e facilitando a descoberta de serviços no sistema distribuído da livraria online.

5.2.2 Modelo de Comunicação

A comunicação entre os componentes é assíncrona. Cada nó envia mensagens para outros nós sem esperar por uma resposta imediata, o que é útil quando a comunicação não precisa ser síncrona e quando a tolerância a falhas é necessária.

Um elemento crucial nesse modelo é a troca de mensagens assíncronas em microsserviços. Essa abordagem permite que os microsserviços se comuniquem diretamente, enviando e recebendo mensagens relevantes para suas operações. Dessa forma, cada microsserviço pode se comunicar com outros microsserviços interessados, sem depender de intermediários específicos, como o Apache Kafka ou o RabbitMQ.

Na arquitetura de microsserviços, os microsserviços se comunicam trocando eventos assíncronos. Um evento representa uma ação ou ocorrência significativa em um contexto específico. Os microsserviços podem publicar eventos quando algo acontece em seu domínio ou se inscrever em eventos específicos para serem notificados quando algo relevante ocorre em outros microsserviços.

No contexto da livraria, temos dois microsserviços independentes: o "Book Service" e o "Câmbio Service". Esses microsserviços desempenham funções específicas e são responsáveis por fornecer informações sobre os livros e gerenciar as operações de câmbio, respectivamente.

Quando um cliente solicita informações detalhadas sobre um livro, por exemplo, o processo de comunicação é iniciado. O cliente envia uma mensagem para o "Book Service", solicitando os dados do livro desejado. Essa mensagem contém os parâmetros necessários, como o ID do livro.

O "Book Service" recebe a mensagem e processa a solicitação, consultando seu banco de dados ou sistema de armazenamento para recuperar as informações detalhadas sobre o livro. Assim que essas informações são obtidas, o "Book Service" envia uma mensagem de resposta ao cliente, contendo os dados solicitados, como autor, título, data de lançamento e preço.

Suponhamos agora que o cliente queira visualizar o preço do livro em sua moeda local. Nesse caso, ele envia uma nova mensagem para o "Câmbio Service", contendo o preço do livro em uma moeda de origem e a moeda de destino desejada.

O "Câmbio Service" recebe essa mensagem e executa a operação de conversão de moeda, utilizando o fator de conversão correspondente. Assim que o preço convertido é calculado, o "Câmbio Service" envia uma mensagem de resposta ao cliente, contendo o preço do livro convertido para a moeda local.

Em ambos os casos, a comunicação entre os microsserviços é assíncrona, ou seja, o cliente não precisa esperar pela resposta imediata. Isso permite que os microsserviços executem suas tarefas de forma independente e otimizada, sem depender de uma resposta imediata de outros componentes.

A arquitetura orientada a eventos desempenha um papel importante nesse modelo de comunicação. Os microsserviços podem publicar eventos quando algo relevante acontece em seus domínios ou se inscrever em eventos específicos para serem notificados quando algo importante ocorre em outros microsserviços.

Por exemplo, o "Book Service" pode publicar um evento informando que um novo livro foi adicionado ao seu catálogo. Outros microsserviços, como o "Câmbio Service" ou até mesmo um serviço de notificação para clientes, podem se inscrever neste evento para serem notificados sobre a adição do novo livro.

Essa abordagem de comunicação baseada em eventos e microsserviços traz benefícios significativos, como a capacidade de escalar cada microsserviço independentemente, aprimorar a resiliência do sistema como um todo e facilitar a integração de novos serviços no futuro. Além disso, a comunicação assíncrona permite uma maior flexibilidade e modularidade, tornando o sistema mais adaptável às necessidades em constante evolução.

5.2.3 Gerenciamento de Dados

No contexto dos microsserviços, o gerenciamento de dados é descentralizado, o que significa que cada serviço é responsável por seu próprio armazenamento e gerenciamento de dados. Isso permite que cada serviço escolha o tipo de banco de dados ou sistema de armazenamento mais adequado para suas necessidades específicas.

No caso de uso de uma livraria online que possui dois microserviços, o gerenciamento de dados foi implementado da seguinte maneira:

1. Microsserviço "Book":

O microsserviço "Book" é responsável por lidar com solicitações relacionadas a informações sobre autores, títulos, datas de lançamento e preços dos livros.

- Armazenamento de Dados: foi utilizado um banco de dados adequado para armazenar as informações dos livros, como autor, título, data de lançamento e preço. Exemplos de bancos de dados populares são MySQL, PostgreSQL ou MongoDB. O banco de dados foi projetado para atender às necessidades do microsserviço "Book" e permitir consultas eficientes para recuperar as informações necessárias.
- APIs: O microsserviço "Book" deve expor APIs para receber solicitações de informações dos clientes. Essas APIs podem ser implementadas usando um framework adequado, como o Flask (Python), Spring Boot (Java) ou Node.js. As APIs foram projetadas para lidar com solicitações para informações de autores, títulos, datas de lançamento e preços dos livros. Por exemplo, uma rota GET /books/{id} para obter informações sobre um livro específico.
- Comunicação entre Microsserviços: A comunicação entre microsserviços, foi utilizada um mecanismo de troca de mensagens assíncronas, como um sistema de filas (por exemplo, RabbitMQ ou Apache Kafka). Isso permitirá que o microsserviço "Book" envie mensagens relevantes para outros microsserviços ou consuma mensagens enviadas por outros microsserviços.
- Além da comunicação, o uso de sistemas de monitoramento e rastreamento pode fornecer percepções valiosas sobre o desempenho e o comportamento do sistema de gerenciamento de dados. Isso permite identificar possíveis problemas e otimizar a infraestrutura conforme necessário (por exemplo, Distributed Tracing ou Prometheus).

2. Microsserviço "Cambio":

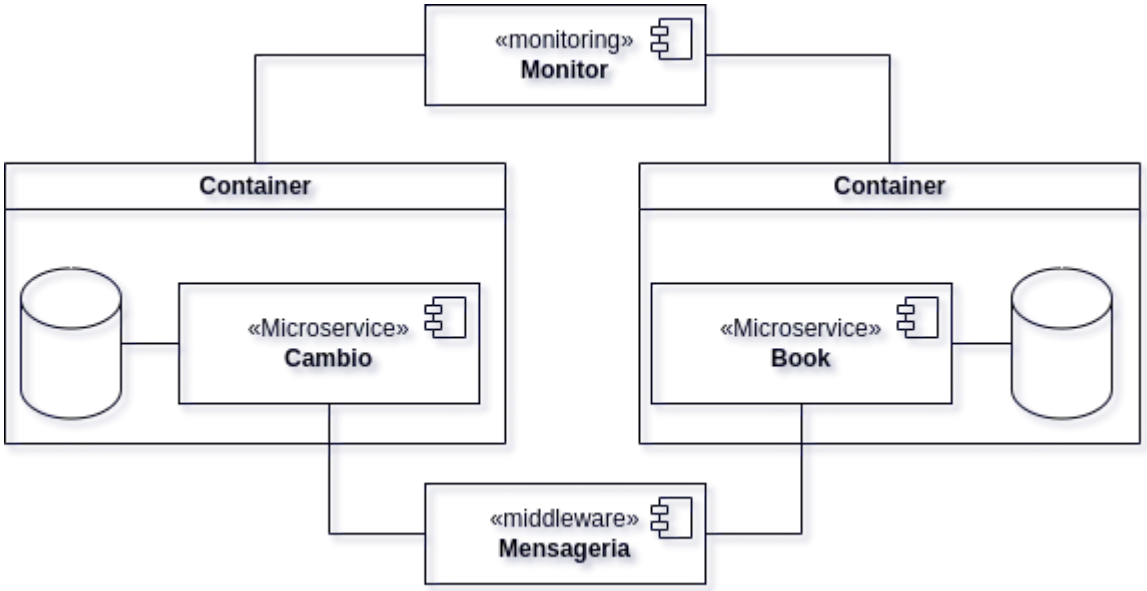
O microsserviço "Cambio" lida com solicitações relacionadas à conversão de moedas. Para gerenciar essas solicitações foi utilizado essa abordagem de gerenciamento de dados:

- Armazenamento de Dados: armazenar informações sobre as taxas de câmbio em um banco de dados. Isso permitirá que o microsserviço "Cambio" consulte o fator de conversão necessário para realizar a conversão de moeda. Pode-se utilizar um banco de dados como o MySQL ou PostgreSQL para armazenar essas informações.

- APIs: O microsserviço "Cambio" expor APIs para receber solicitações de conversão de moedas dos clientes. As APIs foram implementadas usando um framework adequado, assim como mencionado anteriormente. Por exemplo, você pode ter uma rota POST /convert para receber as informações da moeda de origem, moeda de destino e fator de conversão.
- Comunicação entre Microsserviços: O microsserviço "Cambio" pode se comunicar com outros microsserviços para obter informações adicionais. Isso pode ser feito utilizando o mecanismo de troca de mensagens mencionado anteriormente.
- O uso de sistemas de monitoramento e rastreamento foi utilizado também.

É importante lembrar que a implementação exata do gerenciamento de dados em microsserviços pode variar dependendo das tecnologias e ferramentas escolhidas, bem como das necessidades específicas do sistema.

Figura 7 – Disposição dos bancos de dados nos microsserviços.



Fonte: O autor (2023).

A Figura 7 mostra a arquitetura de microsserviços, onde estão representados o serviço de câmbio (Cambio Service), o serviço de livros (Book Service). Esses microsserviços estão conectados ao sistema de mensageria, que permite a comunicação assíncrona entre eles.

O sistema de mensageria facilita a troca de informações e eventos relevantes entre os microsserviços. Isso significa que o Cambio Service e o Book Service podem enviar mensagens para o sistema de mensageria, que encaminha essas mensagens para o destinatário

apropriado. Essa comunicação assíncrona ajuda a garantir a integração eficiente e o processamento adequado dos dados entre os diferentes serviços.

Além disso, a Figura 7 mostra que o sistema de mensageria está conectado ao sistema de monitoramento e rastreamento. Isso permite que o sistema de monitoramento e rastreamento receba informações sobre as mensagens trocadas entre os microsserviços. Essa integração possibilita o acompanhamento e a análise do fluxo de dados, bem como o monitoramento do desempenho e do comportamento do sistema como um todo.

Por fim, tanto o Cambio Service quanto o Book Service possuem seus respectivos bancos de dados. Esses bancos de dados estão conectados ao sistema de monitoramento e rastreamento, o que significa que o sistema de monitoramento e rastreamento pode acessar e analisar os dados armazenados nesses bancos.

O gerenciamento de dados envolve não apenas a escolha adequada do sistema de banco de dados, mas também a utilização de práticas de controle de migração, sistemas de monitoramento e rastreamento, bem como sistemas de mensageria. Esses elementos são essenciais para garantir a confiabilidade, a consistência e a escalabilidade do sistema de gerenciamento de dados.

5.3 Tecnologias de microsserviços

Para a implementação dos microsserviços da livraria online, foram utilizadas as seguintes tecnologias:

- **Java 16**: O Java 16 é uma versão da linguagem de programação Java, que oferece recursos e melhorias mais recentes. É amplamente utilizado para o desenvolvimento de aplicativos de alto desempenho e escaláveis.
- **Spring Tool Suite 4 IDE**: O Spring Tool Suite 4 IDE é um ambiente de desenvolvimento integrado (IDE) baseado no Eclipse, projetado especificamente para o desenvolvimento com o framework Spring. Ele oferece recursos avançados de desenvolvimento e depuração para facilitar a criação de aplicativos Spring.
- **Apache Maven**: O Apache Maven é uma ferramenta de automação de compilação e gerenciamento de dependências. Ele permite gerenciar eficientemente as bibliotecas e dependências do projeto, simplificando o processo de construção e implantação da aplicação.

- **MySQL**: O MySQL é um sistema de gerenciamento de banco de dados relacional amplamente utilizado. Ele fornece um ambiente confiável e eficiente para armazenar, gerenciar e acessar os dados da aplicação.
- **HeidiSQL**: O HeidiSQL é uma ferramenta de administração de banco de dados MySQL, que permite visualizar e gerenciar facilmente os bancos de dados, tabelas, consultas e outros objetos relacionados.
- **Docker Desktop**: O Docker Desktop é uma plataforma que permite criar e executar aplicações em contêineres. Ele facilita o processo de empacotar a aplicação e suas dependências em um ambiente isolado e portátil. O Docker Compose é uma ferramenta que permite executar ambientes de aplicações com vários contêineres do Docker baseados em definições contidas em um arquivo YAML (um formato de serialização de dados). Ele usa as definições de serviço para criar ambientes personalizados com contêineres que podem compartilhar redes e volumes de dados. O Docker Compose simplifica o gerenciamento de processos de aplicações em contêineres que dependem de vários serviços.
- **Postman**: O Postman é uma ferramenta de desenvolvimento de API que facilita a criação, teste e documentação de APIs. Ele permite enviar solicitações HTTP personalizadas, visualizar as respostas e automatizar testes.
- **Git e GitHub**: Git é um sistema de controle de versão distribuído amplamente utilizado, enquanto o GitHub é uma plataforma de hospedagem de repositórios Git. Juntos, eles fornecem um ambiente para controle de versão e colaboração no desenvolvimento de software.
- **GitHub Actions:** é uma ferramenta fornecida pela plataforma GitHub que permite a automação de fluxos de trabalho no desenvolvimento de software (integração contínua). Integrado ao controle de versão Git, o GitHub Actions oferece recursos avançados para automatizar tarefas, como compilação, teste e implantação de aplicativos.

Essas ferramentas são essenciais para o processo de criação e execução da aplicação escalável na nuvem, proporcionando um ambiente de desenvolvimento eficiente e facilitando o gerenciamento de código, dependências, banco de dados.

Para gerenciar os microsserviços, a aplicação utiliza o Spring Boot, que é um framework popular para construir aplicativos em Java. Ele fornece uma configuração padrão que pode ser facilmente personalizada para atender às necessidades do aplicativo. Com o

Spring Boot, os desenvolvedores podem construir rapidamente aplicativos prontos para produção, sem precisar se preocupar com a complexidade da configuração.

Além do Spring Boot, a aplicação utiliza várias outras bibliotecas do ecossistema do Spring para gerenciar microsserviços. Algumas delas são:

- **Spring Cloud**: É um conjunto de bibliotecas que ajudam a criar sistemas distribuídos baseados em microsserviços. Ele fornece soluções para problemas comuns, como balanceamento de carga, descoberta de serviços, configuração, monitoramento e muito mais.
- **Spring Cloud LoadBalancer**: É uma biblioteca que fornece uma abstração para balanceamento de carga em sistemas distribuídos. Ele pode ser usado para implementar balanceadores de carga em nível de serviço ou em nível de aplicação.
- **Spring Cloud Gateway**: É uma biblioteca que fornece uma solução para a criação de gateways de API. Ele permite que os desenvolvedores criem proxies de API personalizados para seus serviços, onde podem gerenciar o tráfego e aplicar políticas de segurança e autorização.
- **Netflix Eureka**: É uma biblioteca de registro e descoberta de serviços. Ele permite que os microsserviços se registrem em um registro centralizado, onde podem ser descobertos e consumidos por outros serviços.
- **Resilience4j**: É uma biblioteca que fornece recursos para tornar aplicativos resilientes e tolerantes a falhas, por meio da implementação de padrões de design, como Circuit Breaker, Retry, Rate Limiter, Bulkhead e TimeLimiter.
- **Feign**: É uma biblioteca que permite aos desenvolvedores criar clientes HTTP declarativamente em Java. É uma solução eficaz para chamar serviços RESTful e pode ser usado em conjunto com outras bibliotecas do ecossistema Spring.
- **Spring Boot Actuator**: É uma biblioteca que fornece recursos para monitorar e gerenciar a aplicação em tempo de execução. Ele expõe endpoints HTTP que fornecem informações sobre a aplicação, como saúde, métricas, estado do thread e muito mais. Os desenvolvedores podem usar esses endpoints para monitorar e gerenciar a aplicação em tempo real. O Spring Boot Actuator é muito útil em ambientes de produção, pois fornece informações valiosas sobre

o estado da aplicação e ajuda os desenvolvedores a diagnosticar problemas em tempo hábil.

- **Spring Rabbit**: É uma biblioteca que fornece uma abstração para o uso do protocolo AMQP (*Advanced Message Queuing Protocol*) em aplicativos Spring. O RabbitMQ é um sistema de mensageria que facilita a comunicação assíncrona entre os componentes do sistema. Ele atua como uma fila de mensagens para enviar e receber mensagens entre o Gateway, o Microsserviço Book Service e o Microsserviço Câmbio Service. Isso permite uma comunicação eficiente e escalável entre esses componentes.
- **Spring Cloud Starter Sleuth**: É uma biblioteca que fornece recursos de rastreamento distribuído para microsserviços. Ele gera um ID de rastreamento exclusivo para cada solicitação de serviço e permite que os desenvolvedores vejam todo o caminho de execução da solicitação em todos os serviços envolvidos.
- **OpenAPI UI**: É uma biblioteca que gera documentação interativa para APIs RESTful, usando a especificação OpenAPI (anteriormente conhecida como Swagger, que é uma biblioteca que fornece uma estrutura para criar documentação e clientes para APIs RESTful. Ele permite que os desenvolvedores criem documentação clara e legível para suas APIs e gere clientes para várias plataformas). Basicamente ele ajuda os desenvolvedores a criar documentação clara e legível para suas APIs.
- **MySQL Connector/J**: É uma biblioteca que fornece um driver JDBC para o banco de dados MySQL. Ele permite que os desenvolvedores se conectem ao banco de dados MySQL e executem consultas e operações.
- **Flyway**: É uma biblioteca de migração de banco de dados que permite que os desenvolvedores gerenciem a evolução do esquema do banco de dados de forma programática. Ele automatiza o processo de aplicação de alterações de esquema em um banco de dados e fornece um controle de versão dos scripts de migração. Com o Flyway, os desenvolvedores podem aplicar e reverter facilmente alterações de esquema em um banco de dados sem se preocupar com a sincronização manual de vários bancos de dados. Ele também é compatível com vários bancos de dados, o que o torna uma ferramenta muito útil para equipes que trabalham com vários bancos de dados diferentes.

- **Zipkin**: Com a utilização da biblioteca Zipkin, torna-se mais fácil detectar gargalos em sistemas distribuídos, pois é possível identificar quais serviços estão consumindo mais recursos ou causando lentidão na transação. Além disso, a biblioteca permite a identificação de erros em um ambiente distribuído, tornando mais fácil a depuração e resolução de problemas. O Zipkin é uma biblioteca que fornece recursos de rastreamento distribuído usando o Zipkin, um sistema de rastreamento distribuído de código aberto. Ele permite que os desenvolvedores vejam o caminho de execução de uma solicitação em todos os serviços envolvidos em uma interface visual. O Distributed Tracing Server é responsável pelo monitoramento e rastreamento das solicitações que percorrem os diferentes microsserviços. Ele recebe informações de rastreamento dos componentes, como o Gateway, o Microsserviço Book Service e o Microsserviço Câmbio Service, por meio do RabbitMQ. Essas informações de rastreamento são registradas e analisadas para obter insights sobre o desempenho e a latência das solicitações.

5.4 Implementação da Aplicação Escalável

Antes de iniciar a implementação da aplicação escalável na nuvem, é imprescindível configurar um ambiente de desenvolvimento apropriado para esta finalidade. A configuração do ambiente foi realizada em um sistema operacional Windows 10 e envolveu a instalação das ferramentas de desenvolvimento, mencionadas no subtítulo 5.3 Tecnologias de microsserviços.

Depois de configurar o ambiente de desenvolvimento, a próxima etapa é a implementação da aplicação escalável. A implementação da aplicação foi realizada adotando as melhores práticas de desenvolvimento em nuvem para este caso de uso específico. A escolha da arquitetura de microsserviços e a utilização de contêineres Docker para empacotar e implantar cada serviço foram decisões estratégicas que contribuíram para a eficiência e escalabilidade do sistema. A arquitetura de microsserviços e o uso de contêineres Docker são considerados excelentes práticas para o desenvolvimento em nuvem. Essas abordagens permitem a criação de sistemas modulares, facilitando o desenvolvimento, o gerenciamento e a escalabilidade de serviços independentes. No entanto, é importante ressaltar que a adequação dessas práticas depende das necessidades e requisitos específicos de cada projeto.

A implementação de uma aplicação escalável requer práticas eficientes para garantir uma implantação rápida e confiável das atualizações de software. Uma forma de implementar a entrega contínua é através do uso de ferramentas como o GitHub Actions, que oferece recursos para a automação de fluxos de trabalho e integração com o controle de versão. Com o GitHub Actions, os desenvolvedores podem definir etapas específicas para construir, testar e implantar a aplicação, permitindo um ciclo de desenvolvimento ágil e eficiente.

Nesse sentido, o Github Actions é um serviço de automação de fluxo de trabalho que oferece recursos poderosos para criar pipelines personalizadas. Ele permite aos desenvolvedores automatizar tarefas como testes de unidade, integração contínua e implantação, além de criar fluxos de trabalho para executar automaticamente a cada push no repositório do Github.

O Docker Hub, por sua vez, é um registro de imagens Docker que permite aos desenvolvedores armazenar e distribuir imagens de contêineres Docker para diferentes versões da aplicação. Com o Docker Hub, é possível gerenciar as dependências e configurações necessárias para executar a aplicação em diferentes ambientes, como produção, *staging(*encenação*)* é um ambiente de pré-produção onde a aplicação é testada antes de ser implantada em produção. Nesse ambiente, a equipe de desenvolvimento pode testar a aplicação em um ambiente controlado, simulando as condições do ambiente de produção e identificando possíveis problemas antes de implantá-la em produção. No contexto do Docker Hub, é possível gerenciar imagens de contêineres específicas para cada ambiente, incluindo o ambiente de staging, para garantir que as configurações e dependências estejam corretas antes de implantar a aplicação em produção e desenvolvimento.

A combinação do Github Actions e Docker Hub possibilita aos desenvolvedores implementar uma pipeline de entrega contínua automatizada para sua aplicação escalável em nuvem. Isso permite automatizar todo o processo de compilação, teste, empacotamento e implantação da aplicação em contêineres Docker, garantindo uma implantação rápida e confiável das atualizações de software.

Assim, a entrega contínua com Github Actions e Docker Hub é uma prática eficiente e confiável para implementar atualizações de software em uma aplicação escalável. Com essa abordagem, os desenvolvedores podem aumentar a velocidade de lançamento de novas funcionalidades, garantindo a estabilidade e confiabilidade da aplicação em todos os ambientes.

Para desenvolver uma aplicação utilizando as ferramentas mencionadas, é necessário seguir uma estratégia de desenvolvimento e integração gradativa dos serviços. Inicialmente,

criamos o microsserviço de câmbio, que é responsável por realizar as conversões de moeda. Para estruturá-lo, começamos com a criação de um mock que imita o comportamento do serviço. Em seguida, integramos o serviço ao banco de dados, configurando o JPA e adicionando o Flyway para gerenciar as migrações de banco de dados.

Em seguida, criamos o segundo microsserviço, o book-service, e o configuramos para consumir o serviço de câmbio. Para orquestrar a comunicação entre esses dois serviços, precisamos de um Service Registry, que é responsável por rastrear os serviços disponíveis. Para isso, usamos o Eureka e configuramos nossos microsserviços para se conectarem a ele.

Em seguida, precisamos implementar o Load Balancer para garantir a escalabilidade e resiliência dos nossos serviços. Para isso, usamos o Eureka em conjunto com o Feign e o Spring Cloud LoadBalancer para implementar um Load Balancer básico.

No entanto, para garantir a resiliência dos nossos serviços, precisamos adicionar o Resilience4j à nossa arquitetura. O Resilience4j é um framework que fornece vários padrões de resiliência, como Circuit Breaker, Retry e Rate Limiter, que ajudam a manter a estabilidade da aplicação, mesmo em caso de falhas.

Além disso, precisamos adicionar um API Gateway à nossa arquitetura para simplificar a comunicação entre nossos serviços e os clientes externos. Para isso, usamos o Spring Cloud Gateway, que nos permite configurar rotas e políticas de segurança para nossos serviços.

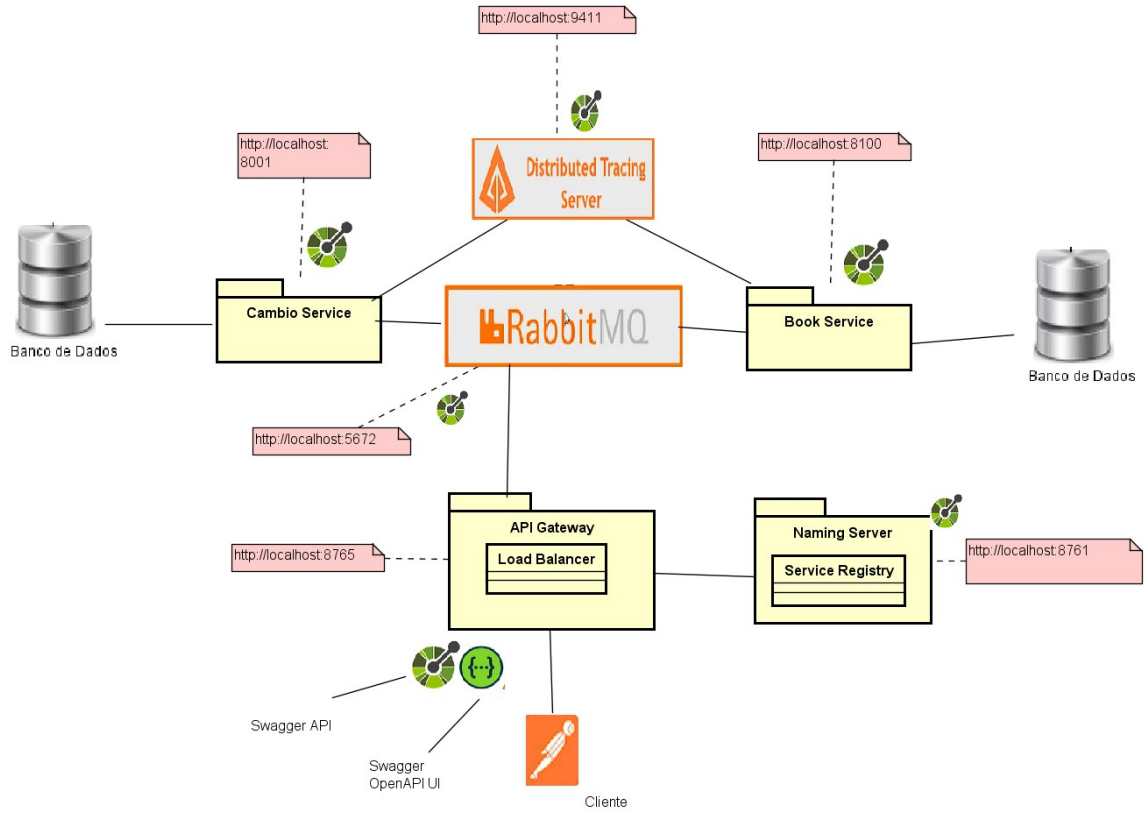
Também adicionamos o OpenAPI aos nossos microsserviços, o que nos permite documentar nossos endpoints e fornecer uma interface de usuário para testá-los. Para ajustar as rotas, configuramos o arquivo application.yml e ajustamos as configurações do OpenAPI UI em nosso API Gateway.

Por fim, para garantir a consistência de configuração em todos os nossos microsserviços, adicionamos o Spring Cloud Config Server à nossa arquitetura. O Config Server nos permite centralizar a configuração de nossos microsserviços e fornecê-la a eles de forma consistente e confiável.

Em resumo, a implementação de uma arquitetura de microsserviços pode ser complexa, mas o uso de ferramentas e frameworks como Java, Spring Boot, Spring Cloud, Feign, Netflix Eureka, Spring Cloud LoadBalancer, Spring Cloud Gateway, Resilience4j, SQL, banco de dados relacional, Docker e Github podem tornar o processo mais eficiente e organizado. Essas ferramentas facilitam o desenvolvimento, a implantação e o gerenciamento de serviços independentes e escaláveis que se comunicam por meio de APIs bem definidas. Além disso, a arquitetura de microsserviços permite a agilidade e a inovação, pois cada

serviço pode ser atualizado sem afetar o restante do sistema. No entanto, para obter os benefícios dos microsserviços, é importante seguir boas práticas e padrões de arquitetura, como usar um gateway de API, definir um contexto limitado para cada serviço, aplicar políticas prontas para uso e usar uma tecnologia de orquestração.

Figura 8 - Diagrama Completo da Aplicação: A imagem ilustra todos componentes da arquitetura microserviço da aplicação



Fonte: O autor (2023).

A Figura 8 representa uma aplicação completa, construída com o uso de microsserviços e uma variedade de tecnologias. Na ilustração, podemos observar os seguintes componentes: Book Service, Cambio Service, API Gateway, Naming Server, RabbitMQ, Distributed Tracing e um banco de dados. Para acessar a aplicação, o cliente utiliza o API Gateway, onde está disponível uma interface OpenAPI UI, que exibe todos os endpoints da aplicação. Toda essa aplicação está disponível no github.

É importante ressaltar que o desenvolvimento de microsserviços deve seguir boas práticas e padrões de arquitetura, a fim de garantir a escalabilidade, flexibilidade e segurança dos sistemas.

Em conclusão, com todas essas ferramentas e frameworks, desenvolver uma arquitetura de microsserviços eficiente e escalável é possível. No entanto e ressaltando, é importante destacar que a implementação de uma arquitetura de microsserviços não é adequada para todos os projetos e é necessário avaliar cuidadosamente as necessidades e requisitos do projeto antes de decidir pela adoção dessa abordagem.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, apresentamos uma abordagem para o desenvolvimento de aplicações escaláveis em nuvem utilizando a arquitetura de microsserviços, contêineres e orquestração de contêineres.

Destacamos a importância da modularidade e autonomia em microsserviços, discutindo os desafios enfrentados na adoção dessa abordagem, o gerenciamento de dados, recursos e garantia de qualidade. Exploramos diversos aspectos relacionados ao desenvolvimento de aplicações em nuvem, apresentando os conceitos fundamentais. Em seguida, discutimos a arquitetura de microsserviços como uma abordagem eficiente para criar aplicações escaláveis, flexíveis.

A aplicação desenvolvida foi uma livraria online de alcance mundial, demandando alto desempenho e capacidade de escalabilidade para lidar com um grande número de usuários. Adotamos a arquitetura de microsserviços, em que a livraria é composta por microsserviços independentes. Como o propósito de demonstração, foram desenvolvidos dois serviços: o "Book Service" que fornece informações detalhadas sobre os livros, enquanto o "Câmbio Service" lida com operações de câmbio, convertendo os preços para a moeda local do cliente.

Utilizamos uma variedade de ferramentas e tecnologias para construir uma aplicação escalável, possibilitando a comunicação entre os microsserviços, balanceamento de carga, resiliência e documentação dos pontos finais (*endpoints)*. Essas ferramentas desempenharam um papel importante na viabilização da implementação.

Durante todo o processo, identificamos pontos fortes e fracos da abordagem adotada. Além disso, certas bibliotecas utilizadas na implementação prática, como o OpenZipkin, Flyway, Eureka, Spring Cloud LoadBalancer, Rabbit AMQP e Actuator, apresentaram limitações e problemas potenciais em relação a desempenho, integração, escalabilidade e configuração avançada.

Apesar desses pontos fracos, o método adotado mostrou-se eficaz na criação de uma aplicação escalável em nuvem, alcançando resultados relevantes. Isso comprova a viabilidade e os benefícios da arquitetura de microsserviços para aplicações escaláveis. No entanto, é importante ressaltar que cada projeto possui necessidades e requisitos específicos, exigindo uma avaliação cuidadosa antes de adotar essa abordagem.

No contexto da computação em nuvem e aplicações escaláveis, existem várias direções que podem ser exploradas em futuros trabalhos. A realização de estudos

comparativos entre diferentes abordagens de desenvolvimento de aplicações escaláveis e a análise de desempenho e escalabilidade em cenários de alta demanda.

Este trabalho contribuiu para o avanço da propagação do conhecimento referente ao desenvolvimento de aplicações escaláveis em nuvem. A abordagem adotada, juntamente com as arquiteturas de microsserviços discutidas, demonstraram a eficiência na construção de uma aplicação escalável e disponível. Espera-se que este trabalho sirva como base para futuras pesquisas e desenvolvimentos práticos nesse campo, auxiliando na criação de aplicações cada vez mais robustas e eficientes na nuvem.

REFERÊNCIAS

AMAZON WEB SERVICES. Microservices. **Amazon Web Services**, 2023. Disponível em: https://aws.amazon.com/pt/microservices/. Acesso em: 27 ago. 2023.

AMAZON WEB SERVICES. O que é arquitetura orientada a serviços. **Amazon Web Services**, 2023. Disponível em: https://aws.amazon.com/pt/what-is/service-oriented-architecture/. Acesso em: 27 ago. 2023.

APACHE SOFTWARE FOUNDATION. Apache Maven. **Apache Software Foundation**, 2023. Disponível em: https://maven.apache.org. Acesso em: 27 ago. 2023.

BALDISSERA, Olívia. Quais são os tipos de arquitetura de software e como escolher o melhor para seu projeto. **PUCPR Digital**, 24 mar. 2021. Disponível em: https://posdigital.pucpr.br/blog/tipos-de-arquitetura-de-software/. Acesso em: 27 ago. 2023.

CONCEIÇÃO, Melissa Tidori da; PINTO, Giuliano Scombatti. Arquitetura de microsserviços: microservice architecture. **Interface Tecnológica**, Taquaritinga, v. 18, n. 2, p. 1-10, 2023. DOI: 10.31510/infa. v18i2.1186. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/1186/669. Acesso em: 27 ago. 2023.

ERL, Thomas. **Service-oriented architecture**: concepts, technology and design. Upper Saddle River, NJ: Prentice Hall, 2005. Disponível em: https://www.arcitura.com/wp-content/uploads/2017/09/Erl_SOABook2_Ch07-2.pdf Acesso em: 27 ago. 2023.

FOWLER, Martin. Microservices: a definition of this new architectural term. **martinfowler.com**, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 27 ago. 2023.

FOWLER, Martin. **Patterns of enterprise application architecture**. São Paulo: Pearson Education, 2002. Disponível em: https://www.amazon.com.br/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420. Acesso em: 27 ago. 2023.

GITHUB. GitHub Actions. **GitHub**, 2023. Disponível em: https://github.com/features/actions. Acesso em: 27 ago. 2023.

HADESFRANKLYN. Desenvolvimento de aplicação em nuvem escalável. **GitHub**, 2023. Disponível em: https://github.com/hadesfranklyn/Desenvolvimento-de-aplicacao-em-nuvem-escalavel/. Acesso em: 27 ago. 2023.

HADESFRANKLYN. Perfil no Docker Hub. **Docker Hub**, 2023. Disponível em: https://hub.docker.com/u/hadesfranklyn/. Acesso em: 27 ago. 2023.

HEIDISQL. HeidiSQL: a MySQL and PostgreSQL Client. **HeidiSQL**, 2023. Disponível em: https://www.heidisql.com/. Acesso em: 27 ago. 2023.

IBM. Microservices: o que são microsserviços? **IBM Brasil**, 2023. Disponível em: https://www.ibm.com/br-pt/topics/microservices. Acesso em: 27 ago. 2023.

MARTIN, Robert C. Clean architecture: a craftsman's guide to software structure and design. London: Pearson, 2017. Disponível em: https://www.amazon.com.br/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164. Acesso em: 27 ago. 2023.

MICROSOFT AZURE. O que é computação em nuvem? **Microsoft Azure**, 2023. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-cloud-computing . Acesso em: 27 ago. 2023.

MICROSOFT. Arquitetura de microsserviços. **Microsoft**, 2023. Disponível em: https://learn.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture. Acesso em: 27 ago. 2023.

MICROSOFT. Estilo de arquitetura controlada por evento, **Microsoft**, 2023. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/event-driven. Acesso em: 27 ago. 2023.

MICROSOFT. O que são microservices. **Microsoft**, 2023. Disponível em: https://learn.microsoft.com/pt-br/devops/deliver/what-are-microservices. Acesso em: 27 ago. 2023.

MySQL. MySQL Documentation. **My SQL**, 2023. Disponível em: https://dev.mysql.com/doc/. Acesso em: 27 ago. 2023.

NEWMAN, Sam. **Building microservices**: designing fine-grained systems. 2nd. ed. Sebastopol, CA, EUA: O'Reilly Media, Inc., 2021. ISBN: 9781492034025. Disponível em https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/. Acesso em: 27 ago. 2023.

ORACLE CORPORATION. Java Platform, Standard Edition Documentation. **Oracle Corporation**. 2023. Disponível em: https://docs.oracle.com/en/java/. Acesso em: 27 maio 2023.

ORACLE CORPORATION. MySQL Connector/J8.2.0. **Oracle Corporation**, 2023. Disponível em: https://dev.mysql.com/downloads/connector/j/. Acesso em: 27 ago. 2023.

PARSONS, Rebecca. Adotar microsserviços? **Thoughtworks**, 15 may 2018. Disponível em: https://www.thoughtworks.com/pt-br/insights/blog/microservices-adopt. Acesso em: 27 ago. 2023.

RED HAT. Microservices. **Red Hat,** 25 jan. 2023. Disponível em: https://www.redhat.com/pt-br/topics/microservices. Acesso em: 27 ago. 2023.

RED HAT. O que é arquitetura orientada a serviços. **Red Hat**, 7 ago. 2023. Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-service-oriented-architecture. Acesso em: 27 ago. 2023.

RED HAT. O que são microservices. **Red Hat**, 1 maio 2023. Disponível em: https://www.redhat.com/pt-br/topics/microservices/what-are-microservices. Acesso em: 27 ago. 2023.

RESILIENCE4J. Resilience4j Documentation. **Resilience4j,** 2023. Disponível em: https://resilience4j.readme.io/docs. Acesso em: 27 ago. 2023.

SPRING.IO Spring Boot Reference Documentation. **Spring,** 2023 Disponível em: https://docs.spring.io/spring-boot/docs/current/reference/html/index.html. Acesso em: 27 ago. 2023.

SPRING.IO. Spring AMQP. **Spring**, 2023. Disponível em: https://spring.io/projects/spring-amqp. Acesso em: 27 ago. 2023.

SPRING.IO. Spring Cloud Gateway. **Spring**, 2023. Disponível em: https://spring.io/projects/spring-cloud-gateway. Acesso em: 27 ago. 2023.

SPRING.IO. Spring Cloud LoadBalancer. **Spring**, 2023. Disponível em: https://spring.io/projects/spring-cloud-loadbalancer. Acesso em: 27 ago. 2023.

SPRING.IO. Spring Cloud Sleuth. **Spring**, 2023. Disponível em: https://spring.io/projects/spring-cloud-sleuth. Acesso em: 27 ago. 2023.

SPRING.IO. Spring Cloud. **Spring**, 2023. Disponível em: https://spring.io/projects/spring-cloud. Acesso em: 27 ago. 2023.

SPRING.IO. Spring Tools. Disponível em: https://spring.io/tools. **Spring**, 2023. Acesso em: 27 ago. 2023.

SWAGGER. Swagger UI. **Swagger**, [202?]. Disponível em: https://swagger.io/tools/swagger-ui/. Acesso em: 27 ago. 2023.

VILLACA, Luis Henrique Neves; PIMENTA JUNIOR, Antônio Francisco; AZEVEDO, Leonardo Guerreiro. Construindo aplicações distribuídas com microsserviços. **Research Gate**, 2018. Disponível em: https://www.researchgate.net/profile/Luis-Villaca/publication/338989226_Construindo_Aplicacoes_Distribuidas_com_Microsservicos/links/5e3f7a5f458515072d8a9eab/Construindo-Aplicacoes-Distribuidas-com-Microsservicos.pdf. Acesso em: 27 ago. 2023.

ZIPKIN. Zipkin Distributed Tracing System. **Zipkin**, 2023. Disponível em: https://zipkin.io/. Acesso em: 27 ago. 2023.