RAFAEL PORTO VIEIRA DE MOURA

ARQUITETURA DE MICROSSERVIÇOS:

ENDEREÇANDO REQUISITOS NÃO-FUNCIONAIS COM SERVIÇOS AWS

LAVRAS - MG

2025

RAFAEL PORTO VIEIRA DE MOURA

ARQUITETURA DE MICROSSERVIÇOS:
ENDEREÇANDO REQUISITOS NÃO-FUNCIONAIS COM SERVIÇOS AWS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para a obtenção do título de Bacharel.

Prof. DSc. Ricardo Terra
Orientador

LAVRAS - MG

2025

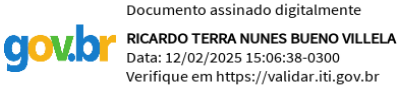
RAFAEL PORTO VIEIRA DE MOURA

ARQUITETURA DE MICROSSERVIÇOS: ENDEREÇANDO REQUISITOS NÃO-FUNCIONAIS COM SERVIÇOS AWS
MICROSERVICES ARCHITECTURE: ADDRESSING NON-FUNCTIONAL REQUIREMENTS WITH AWS SERVICES

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para a obtenção do título de Bacharel.

APROVADA em 07 de Fevereiro de 2025.

Prof. Dra. Renata Teles Moreira	UFLA
Prof. Dr. Maurício Ronny de Almeida Souza	UFLA



Prof. DSc. Ricardo Terra
Orientador

LAVRAS - MG
2025

Dedico este trabalho a todos aqueles cujas contribuições e sabedoria foram fundamentais para o meu aprendizado.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Wilson e Maria, o amor, paciência e incentivo em todos os momentos da minha jornada.

Ao meu orientador, Ricardo Terra, a transmissão de conhecimento e contribuições que me guiaram ao longo de todo o desenvolvimento deste trabalho.

Aos colegas de curso e amigos, a parceria, troca de ideias e momentos de descontração que tornaram essa caminhada mais leve e enriquecedora.

À instituição de ensino, por oferecer um ambiente que favorece o aprendizado e por proporcionar os recursos necessários para a realização deste trabalho.

Por fim, a todas as pessoas que, de alguma forma, estiveram presentes durante esta trajetória, meu mais sincero agradecimento. Este trabalho é um reflexo do apoio e colaboração de todos vocês.

Muito obrigado!

RESUMO

A crescente demanda por sistemas de comércio eletrônico com alta disponibilidade, escalabilidade e segurança tem imposto desafios significativos aos desenvolvedores, especialmente na manutenção de desempenho consistente durante picos de acesso e no gerenciamento seguro de informações sensíveis. Esses requisitos não-funcionais são críticos para assegurar a continuidade dos serviços e a satisfação dos usuários. Nesse contexto, a computação em nuvem com sua promessa de elasticidade e resiliência somada à arquitetura de microsserviços que permite a modularidade e flexibilidade, é uma solução capaz de satisfazer os requisitos críticos e assegurar a satisfação dos usuários. Neste trabalho, é projetado e implementado o `CompraVirtual`, uma aplicação de comércio eletrônico que adota a arquitetura de microsserviços para segmentar funcionalidades como autenticação, gestão de produtos, gestão de estoque, pedidos e notificações. Mais importante, a arquitetura do sistema adota 25 tecnologias AWS, as quais foram metodologicamente selecionadas a partir de 18 requisitos não-funcionais derivados dos requisitos do sistema. Logo, cada um dos cinco microsserviços foram projetados e implementados por meio das tecnologias AWS que os auxiliam no atendimento dos seus requisitos não-funcionais específicos. Como contribuição, este trabalho afirma que a combinação estruturada de tecnologias AWS pode satisfazer requisitos complexos de sistemas distribuídos, oferecendo um modelo replicável para futuros projetos de pesquisa e desenvolvimento na área de arquitetura de microsserviços. A análise realizada revelou que a satisfação de requisitos não-funcionais críticos exige múltiplas tecnologias em sinergia, enquanto alguns requisitos específicos podem ser atendidos por uma única solução. Além disso, foi observado que nenhuma tecnologia utilizada conseguiu atender a mais de três requisitos simultaneamente, o que reforça a importância de um planejamento arquitetural bem fundamentado.

Palavras-chave: Microsserviços. Computação em Nuvem. AWS. Arquitetura de Software. Sistemas Distribuídos. Engenharia de Software. Requisitos Não-Funcionais.

ABSTRACT

The growing demand for e-commerce systems with high availability, scalability, and security has posed significant challenges to developers, particularly in maintaining consistent performance during access peaks and securely managing sensitive information. These non-functional requirements are critical to ensuring service continuity and user satisfaction. In this context, cloud computing, with its promise of elasticity and resilience, combined with the microservices architecture that enables modularity and flexibility, provides a solution capable of addressing these critical requirements and ensuring user satisfaction. This study designs and implements `CompraVirtual`, an e-commerce application that adopts the microservices architecture to segment functionalities such as authentication, product management, inventory management, order processing, and notifications. More importantly, the system's architecture incorporates 25 AWS technologies, methodically selected based on 18 non-functional requirements derived from system specifications. Consequently, we designed and implemented each of the five microservices using AWS technologies that support addressing their specific non-functional requirements. As a contribution, this work demonstrates that the structured combination of AWS technologies can handle complex distributed system requirements, providing a replicable model for future research and development projects in the field of microservices architecture. The analysis revealed that addressing critical non-functional requirements requires multiple technologies working in synergy, while some specific requirements can be met by a single solution. Moreover, it was observed that none of the used technologies could satisfy more than three requirements simultaneously, reinforcing the importance of well-founded architectural planning.

Keywords: Microservices Architecture. Cloud Computing. AWS. Software Architecture. Distributed Systems. Software Engineering. Non-Functional Requirements.

LISTA DE FIGURAS

Figura 2.1 – Limites de serviço reforçados por limites de equipe 15
Figura 2.2 – Banco de dados único vs banco de dados da aplicação 17
Figura 4.1 – Processo de integração do código do sistema `CompraVirtual` 26
Figura 4.2 – Processo de implantação contínua do sistema `CompraVirtual` 28
Figura 4.3 – Arquitetura da rede virtual 30
Figura 4.4 – Arquitetura da resolução de domínios e emissão de certificados 31
Figura 4.5 – Arquitetura de monitoramento 32
Figura 4.6 – Microsserviços x Tecnologias 34
Figura 4.7 – Tecnologias por categorias de RNFs 38
Figura 4.8 – Grafo de relacionamento entre RNFs e Tecnologias 40

LISTA DE TABELAS

Tabela 3.1 – Requisitos não-funcionais do `CompraVirtual` 22
Tabela 4.1 – Tecnologias x Requisitos não-funcionais do `CompraVirtual` 36

SUMÁRIO

1 Introdução 9
2 *Background* 11
2.1 Computação em Nuvem 11
2.1.1 Características essenciais 11
2.1.2 Modelos de serviços 12
2.2 Microsserviços 14
2.2.1 Características comuns 14
2.2.2 Tecnologias 18
3 *Sistema* 20
3.1 Descrição 20
3.2 Requisitos não-funcionais 21
4 Projeto e Implementação 25
4.1 Arquitetura do sistema 25
4.2 Arquitetura dos microsserviços 33
4.3 Análise e discussão 35
5 CONCLUSÃO 41
REFERÊNCIAS 43

1 INTRODUÇÃO

O avanço da tecnologia e o aumento exponencial do uso de sistemas digitais exigem que as aplicações modernas ofereçam desempenho e escalabilidade elevados, especialmente em setores como o comércio eletrônico (HASSELBRING; STEINACKER, 2017; NETO, 2009). Esses sistemas enfrentam o desafio de gerenciar uma alta demanda de usuários simultâneos, o que exige uma infraestrutura robusta e escalável, capaz de manter a consistência e o desempenho nos processos mesmo sob carga intensa. O impacto direto de falhas em sistemas de comércio eletrônico pode ser medido em perdas financeiras e na reputação, tornando-se essencial a implementação de arquiteturas que garantam a disponibilidade e a resiliência do serviço para os usuários finais (NETO, 2009; SUGIANTORO; ANSHARI; SUDRAJAT, 2020).

Nesse contexto, a arquitetura de microsserviços pode ser adotada devido à sua capacidade de dividir aplicações em componentes independentes, facilitando a escalabilidade e a manutenção do sistema. Cada microsserviço é responsável por uma funcionalidade específica, podendo ser desenvolvido, testado e implantado de forma isolada, o que proporciona maior flexibilidade e modularidade (FOWLER; LEWIS, 2014; NEWMAN, 2022; FOWLER, 2017). Além disso, a computação em nuvem, com sua promessa de elasticidade e resiliência, permite que sistemas construídos com essa arquitetura se adaptem dinamicamente à variação de carga, garantindo o desempenho e a continuidade do serviço sem a necessidade de infraestrutura física dedicada (MELL; GRANCE, 2011; NEWMAN, 2022).

Este trabalho apresenta o desenvolvimento do `CompraVirtual`, uma aplicação de comércio eletrônico orientada à nuvem *Amazon Web Services* (AWS), utilizando a arquitetura de microsserviços. O objetivo é projetar e implementar um sistema que atenda aos requisitos não-funcionais essenciais. Para isso, foram selecionadas e configuradas tecnologias da AWS que atendem a cada um desses requisitos. A implementação de práticas como infraestrutura como código, permitiu automação e flexibilidade no gerenciamento de mudanças, enquanto serviços de monitoramento, escalabilidade e disponibilidade asseguraram a qualidade e a confiabilidade do sistema.

Como contribuição, este trabalho argumenta que a combinação estruturada de tecnologias AWS pode satisfazer requisitos complexos de sistemas distribuídos, oferecendo um modelo replicável para futuros projetos de pesquisa e desenvolvimento na área de arquitetura de microsserviços. A análise realizada evidenciou que requisitos como disponibilidade, desempenho, escalabilidade e segurança demandam a integração de múltiplas tecnologias, enquanto requisitos como monitoramento e evolucionabilidade podem ser atendi-

dos com uma única solução específica. Adicionalmente, nenhuma tecnologia foi capaz de atender mais de três requisitos não-funcionais simultaneamente, reforçando a importância de uma abordagem arquitetural planejada.

Este trabalho está organizado da seguinte forma. A Seção 2 introduz a computação em nuvem e a arquitetura de microsserviços. Em seguida, a Seção 3 descreve o sistema `CompraVirtual` e define os requisitos não-funcionais. A Seção 4 descreve o processo de projetar e implementar o sistema e relaciona as tecnologias utilizadas e os requisitos não-funcionais. Por fim, a Seção 5 relata os resultados obtidos e apresenta trabalhos futuros.

2 *BACKGROUND*

Este capítulo apresenta os conceitos fundamentais que sustentam a arquitetura proposta para o sistema `CompraVirtual`. A Seção 2.1 aborda a Computação em Nuvem, destacando suas características essenciais e modelos de serviço. Em seguida, a Seção 2.2 discute a Arquitetura de Microsserviços, explorando suas principais características e as tecnologias que possibilitam sua implementação. Esses conceitos fornecem a base teórica necessária para embasar as decisões arquiteturais adotadas neste trabalho e sua relação com os requisitos não-funcionais definidos.

2.1 Computação em Nuvem

A computação em nuvem, de acordo com a definição do Instituto Nacional de Padrões e Tecnologia (NIST), é um modelo que oferece acesso fácil e instantâneo a recursos de computação compartilhados, como redes, servidores, armazenamento, aplicativos e serviços. Esses recursos podem ser configurados de acordo com as necessidades e são provisionados e liberados de forma rápida, sem exigir esforço significativo de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2011).

Nesta seção, são abordados os aspectos do modelo de computação em nuvem. A Seção 2.1.1 aborda as características essenciais desse modelo e a Seção 2.1.2 aborda os modelos de serviços ofertados. É importante compreender esses aspectos da computação em nuvem para alinhar as expectativas do que se pode obter desse modelo e tomar a decisão mais apropriada à necessidade.

2.1.1 Características essenciais

O modelo de computação em nuvem proposta pelo NIST possui cinco características essenciais (MELL; GRANCE, 2011):

- **Auto-atendimento sobre demanda**: O consumidor pode provisionar capacidades de computação conforme necessário, sem exigir interação humana com o provedor de serviço.
- **Acesso amplo à rede**: As capacidades estão disponíveis pela rede e são acessadas por meio de mecanismos padrão que promovem o uso por plataformas de cliente heterogêneas (por exemplo, telefones celulares, *tablets*, *laptops* e estações de trabalho).

- **Agrupamento de recursos**: O provedor agrupa recursos de computação para atender a vários consumidores em um modelo de vários inquilinos. Os recursos físicos e virtuais são atribuídos e reatribuídos dinamicamente de acordo com a demanda do consumidor. O cliente geralmente não tem controle sobre a localização exata dos recursos, mas pode especificar em um nível mais alto de abstração. Os recursos incluem armazenamento, processamento, memória e largura de banda de rede.

- **Elasticidade rápida**: As capacidades podem ser elasticamente provisionadas e liberadas de forma rápida e automática, de acordo com a demanda. Para o consumidor, essas capacidades disponíveis para provisionamento aparentam ser ilimitadas e podem ser utilizadas em qualquer quantidade e momento desejado.

- **Serviço mensurável**: Em sistemas em nuvem, o uso de recursos é automaticamente controlado e otimizado por meio de medição adequada ao tipo de serviço (como armazenamento, processamento, largura de banda e contas de usuários ativas). Esse uso pode ser monitorado, controlado e relatado, garantindo transparência tanto para o provedor quanto para o consumidor do serviço.

Monteiro (2022) argumenta que essas não são as cinco características somente. Erl, Puttini e Mahmood (2013) também adicionam a característica resiliência, pois é uma capacidade que a maior parte das plataformas de computação em nuvem possuem. Essa característica refere-se a capacidade da solução de ser tolerante a falhas. Ainda, Zhang, Cheng e Boutaba (2010) incluem a característica de orientação para serviços já que são disponibilizadas como um conjunto de serviços, sendo assim, esses serviços têm que cumprir com *Service Level Agreement* (SLA), que define quais condições de serviço o provedor tem que garantir (MONTEIRO, 2022).

Neste trabalho, foram consideradas tanto as características definidas por Mell e Grance (2011) quanto a característica de resiliência destacada por Erl, Puttini e Mahmood (2013) e o conceito de *Service Level Agreement* (SLA) apresentado por Zhang, Cheng e Boutaba (2010).

2.1.2 Modelos de serviços

O modelo de computação em nuvem proposta pelo NIST possui três modelos de serviços:

- **Software como Serviço (SaaS)**: É um modelo de computação em nuvem em que o consumidor utiliza as aplicações do provedor hospedadas em uma infraestrutura de nuvem, acessíveis por meio de

interfaces web ou programas. O consumidor não controla a infraestrutura subjacente, mas desfruta de características como escalabilidade, compartilhamento da aplicação e suporte fácil, incluindo atualizações e correções de software (MELL; GRANCE, 2011; ALMUBADDEL; ELMOGY, 2016). Exemplos de serviços SaaS incluem Google Workspace[1] e o Amazon WorkSpaces[2].

- **Plataforma como Serviço (PaaS)**: É um modelo de computação em nuvem que permite ao consumidor implantar aplicativos na infraestrutura de nuvem usando ferramentas e recursos fornecidos pelo provedor, sem se preocupar com a gestão da infraestrutura subjacente (MELL; GRANCE, 2011). O PaaS possui características como arquitetura multi-inquilino, controle preciso sobre segurança e compartilhamento, capacidades avançadas de fluxo de trabalho, escalabilidade incorporada, interface do usuário personalizável, personalização do banco de dados, integração habilitada por serviços, funcionalidades de autoatendimento e *middleware* pré-construído (ALMUBADDEL; ELMOGY, 2016). Alguns exemplos de serviços PaaS incluem AWS Elastic Beanstalk[3] e AWS Lambda[4].

- **Infraestrutura como Serviço (IaaS)**: O consumidor tem a capacidade de provisionar recursos essenciais de computação, como processamento, armazenamento e redes, e implantar e executar software personalizado, incluindo sistemas operacionais e aplicativos (MELL; GRANCE, 2011), com recursos fornecidos como serviço, escalabilidade dinâmica e autoatendimento (ALMUBADDEL; ELMOGY, 2016). Exemplos de serviços IaaS incluem Amazon EC2[5] e Amazon VPC[6].

Segundo Monteiro (2022), a definição do NIST sobre os modelos de serviço tem sido alvo de questionamentos devido ao avanço tecnológico que tornou a distinção menos clara e possibilitou o surgimento de novos serviços não abrangidos pela definição original. Como resultado, o conceito de *Anything as a Service* (XaaS) foi introduzido, englobando qualquer tecnologia que possa ser disponibilizada pela Internet, mesmo que antes fosse exclusivamente oferecida localmente (*on-premises*) (MIYACHI, 2018).

[1] https://workspace.google.com/
[2] https://clients.amazonworkspaces.com/
[3] https://aws.amazon.com/elasticbeanstalk/
[4] https://aws.amazon.com/lambda/
[5] https://aws.amazon.com/ec2/
[6] https://aws.amazon.com/vpc/

2.2 Microsserviços

A arquitetura de microsserviços é uma arquitetura nativa da nuvem (BALALAIE; HEYDARNOORI; JAMSHIDI, 2016), em que uma aplicação é dividida em vários serviços independentes, cada um executando em seu próprio processo e se comunicando por meio de mecanismos leves (FOWLER; LEWIS, 2014). Cada serviço pode ser desenvolvido, implantado e escalado independentemente, proporcionando maior flexibilidade e modularidade ao sistema (FOWLER; LEWIS, 2014; NEWMAN, 2022). Ao contrário do estilo monolítico, em que uma aplicação é construída como uma única unidade, com todas as suas funcionalidades, lógica e componentes executando em um único processo (PONCE; MÁRQUEZ; ASTUDILLO, 2019), os microsserviços permitem flexibilidade na escolha de linguagens de programação e tecnologias de armazenamento de dados (FOWLER; LEWIS, 2014; FOWLER, 2017; NEWMAN, 2022).

Segundo Newman (2022) e Zimmermann (2017), os microsserviços são uma maneira de fazer Arquitetura Orientada a Serviços (SOA) da maneira "certa", com foco em implantações independentes, enquanto Fowler e Lewis (2014) afirmam que, na verdade, são arquiteturas distintas em termos de compartilhamento de componentes, comunicação de serviços, mediação de serviços e acesso remoto a serviços.

Apesar das vantagens, a arquitetura de microsserviços traz desafios. Esses incluem lidar com a comunicação complexa entre serviços, assegurar a consistência dos dados em ambientes distribuídos, gerenciar a orquestração e coreografia entre serviços e monitorar a saúde e o desempenho de múltiplos serviços (PATHAK; SINGH, 2023).

Nesta seção, são abordados os aspectos da arquitetura de microsserviços. A Seção 2.2.1 aborda as características comuns e a Seção 2.2.2 aborda as tecnologias que a viabilizam. É importante compreender esses aspectos para compreender as limitações, vantagens e necessidades dessa arquitetura e tomar a decisão mais apropriada para cada produto.

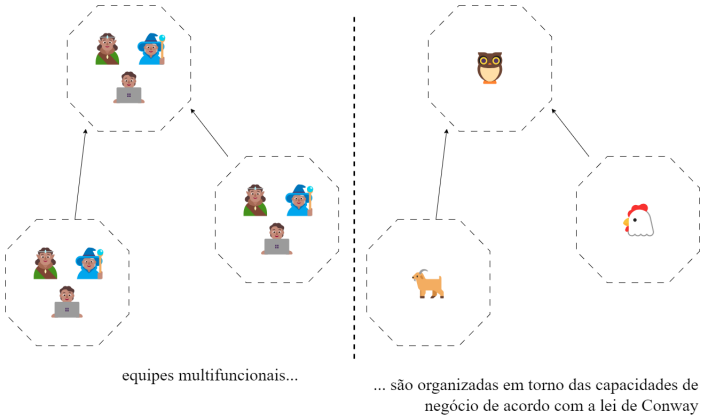
2.2.1 Características comuns

Embora não haja uma definição precisa desse estilo arquitetônico, é importante ressaltar que características comuns podem ser observadas (FOWLER; LEWIS, 2014). Para fornecer uma melhor compreensão, Fowler e Lewis (2014) estabeleceram algumas principais características do estilo arquitetônico de microsserviços:

- **Componentização via serviços**: Consiste em dividir o software em serviços independentes que podem ser implantados e atualizados de forma separada. Essa abordagem permite maior flexibilidade e modularidade, minimizando o impacto de alterações em componentes individuais (FOWLER; LEWIS, 2014). De acordo com Zhou et al. (2023), uma das principais motivações para a componentização por meio de serviços é a implementação contínua e a adoção do *DevOps*. Essas práticas permitem um desenvolvimento ágil e contínuo, além de promover a colaboração entre equipes de desenvolvimento e operações.

- **Organizado em torno das capacidades de negócio**: Influenciada pela Lei de Conway – a qual afirma que a estrutura de um sistema projetado por uma organização será uma cópia da estrutura de comunicação dessa organização (CONWAY, 1968) – a abordagem de microsserviços divide a aplicação em serviços organizados em torno das capacidades de negócios, com equipes multifuncionais responsáveis pelo desenvolvimento completo, desde a experiência do usuário até o gerenciamento do projeto (NEWMAN, 2022). Conforme ilustra a Figura 2.1, isso traz vantagens em termos de modularidade e clareza das fronteiras entre as equipes (FOWLER; LEWIS, 2014).

Figura 2.1 – Limites de serviço reforçados por limites de equipe



Fonte: Adaptado de Fowler e Lewis (2014)

- **Produtos, não Projetos**: As equipes que adotam microsserviços preferem um modelo em que as equipes possuam um produto ao longo de todo o seu ciclo de vida, em vez de seguir o tradicional modelo de projeto, onde o software é considerado concluído e entregue a uma equipe de manutenção (FO-

WLER; LEWIS, 2014). Essa abordagem é inspirada pela mentalidade da *Amazon*[7] de "*build it / run it*", em que os desenvolvedores têm responsabilidade total pelo software em produção. Isso permite que eles estejam em contato direto com o desempenho do software e tenham um envolvimento mais próximo com os usuários, assumindo parte do suporte (O'HANLON, 2006; KILLALEA, 2020).

- ***Endpoints* inteligentes e *pipes* burros**: Na abordagem de microsserviços, os *endpoints* são projetados para serem inteligentes, enquanto os canais de comunicação são mantidos de forma simples (FOWLER; LEWIS, 2014). Isso significa que cada microsserviço possui sua própria lógica de domínio e atua como um filtro que recebe solicitações, aplica a lógica apropriada e produz respostas, utilizando protocolos simples (FOWLER; LEWIS, 2014). Segundo Zhou et al. (2023), os principais mecanismos de comunicação mencionados por profissionais de software/IT, Banco e Finanças, são tanto o protocolo HTTP(s) RESTful API (FIELDING, 2000; FOWLER, 2010) quanto o RPC (BIRRELL; NELSON, 1984). Ainda segundo Zhou et al. (2023), os *frameworks* de mensagens leves como RabbitMQ5[8] e Kafka6[9] também são opções adotadas.

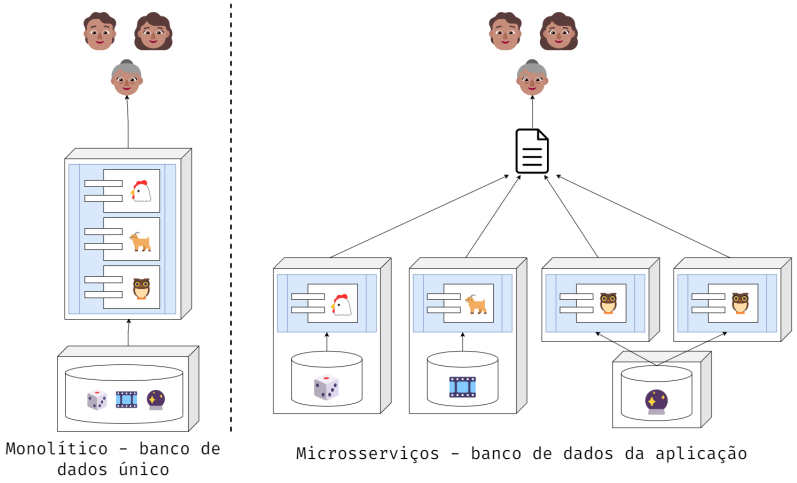
- **Governança descentralizada**: A governança centralizada tende a padronizar em plataformas de tecnologia única, o que limita as opções de desenvolvimento. No entanto, o estilo arquitetônico de microsserviços permite utilizar a ferramenta certa para cada tarefa (FOWLER; LEWIS, 2014). Essas práticas de governança descentralizada e abordagem flexível contribuem para a inovação e agilidade das equipes de microsserviços (FOWLER; LEWIS, 2014), pois, como argumenta Maslow (1966), "nem todo problema é um prego e nem toda solução é um martelo". No entanto, Fowler (2017) defende que essa decisão tem que ser tomada com cautela, pois a padronização tende a facilitar o desenvolvimento.

- **Governança de dados descentralizada**: A abordagem de microsserviços enfatiza a descentralização das decisões de armazenamento de dados, permitindo que cada serviço gerencie seu próprio banco de dados, conforme ilustra a Figura 2.2. Isso é conhecido como Persistência Poliglota (SADALAGE; FOWLER, 2012) e permite a utilização de diferentes tecnologias de banco de dados por cada serviço. Essa descentralização dos dados proporciona maior autonomia aos serviços e permite a adaptação

[7] https://www.amazon.com
[8] https://www.rabbitmq.com/
[9] https://kafka.apache.org/

às necessidades específicas de cada contexto, contribuindo para a flexibilidade e escalabilidade do sistema (FOWLER; LEWIS, 2014).

Figura 2.2 – Banco de dados único vs banco de dados da aplicação



Fonte: Adaptado de Fowler e Lewis (2014)

- **Automação de Infraestrutura**: Envolve incorporar práticas de automatização nos processos de testes e implantação, por exemplo, o *DevOps* que permite o desenvolvimento, implantação e gerenciamento de microsserviços. A principal vantagem do *DevOps* é a capacidade de realizar ciclos rápidos de entrega, integrando desenvolvimento e operações (EBERT et al., 2016). É recomendado que *DevOps* e os microsserviços sejam baseados em nuvem, principalmente quando o objetivo é fornecer serviços de forma eficiente (FOWLER; LEWIS, 2014; EBERT et al., 2016; PATHAK; SINGH, 2023).

- **Projetado para falha**: De acordo com Fowler e Lewis (2014), a utilização de serviços como componentes em uma arquitetura de microsserviços traz a necessidade de lidar com a possibilidade de falhas nos serviços. É importante detectar e responder rapidamente a essas falhas. O fato de um microsserviço estar inativo pode ser a causa de falhas em cascata de outros microsserviços em todo sistema (ZHOU et al., 2023). Por isso, as equipes de microsserviços dão grande ênfase ao monitoramento em tempo real da aplicação, verificando tanto aspectos arquiteturais quanto métricas relevantes para o negócio (ZHOU et al., 2023).

- ***Design* Evolutivo**: É uma abordagem de desenvolvimento de software que busca criar sistemas que possam se adaptar e evoluir com o tempo (FOWLER; LEWIS, 2014). Em vez de tentar prever todos os

requisitos e implementar uma solução final, o *design* evolutivo prioriza a flexibilidade e a capacidade de responder às mudanças à medida em que surgem (FOWLER; LEWIS, 2014). Isso permite que o software seja continuamente aprimorado e atualizado de acordo com as necessidades em constante evolução do usuário e do mercado (FOWLER; LEWIS, 2014). Um exemplo de *design* evolutivo é o S3[10] da AWS como relata Killalea (2020).

2.2.2 Tecnologias

Newman (2022) argumenta que o conhecimento das tecnologias disponíveis é essencial para que a implementação de microsserviços seja um sucesso. Porém, deve-se agir com cautela em adotar muitas tecnologias novas ao iniciar o trabalho com microsserviços, já que pode ter um impacto contraproducente. Ainda, Newman (2022) traz algumas dessas tecnologias que tornam os microsserviços viáveis, as quais estão descritas abaixo:

- **Agregação de *logs* e *tracing* distribuído**: Para implementar a arquitetura de microsserviços, é crucial a criação de um sistema de agregação de *logs*. Essa ferramenta viabiliza a coleta e análise centralizada dos *logs* de todos os serviços, integrando-se a um sistema de alertas ativo. A adição de identificadores de correlação aprimora a eficácia dessas ferramentas, facilitando o isolamento dos *logs* relacionados a um fluxo específico de chamadas e, consequentemente, simplificando a resolução de problemas. Com o aumento da complexidade do sistema, é vital explorar tecnologias que permitam a análise de *traces* entre serviços, identificando gargalos e viabilizando questionamentos sobre o sistema. Nesse cenário, a AWS oferece o serviço *Amazon CloudWatch*[11] que coleta, visualiza e monitora *logs*, métricas e eventos em tempo real por meio de painéis automatizados.

- **Contêineres e *Kubernetes***[12]: É altamente recomendável executar instâncias de microsserviços de forma isolada. Os contêineres, nesse cenário, surgem como a solução ideal, oferecendo inicialização ágil e eficiência econômica. Para uma gestão eficaz desses contêineres, é aconselhável o uso de plataformas de orquestração, pois permitem a distribuição eficiente das instâncias de contêineres para garantir a robustez e o *throughput* necessários para o serviço, além de otimizar o uso das máquinas

[10] https://aws.amazon.com/s3/
[11] https://aws.amazon.com/pt/cloudwatch/
[12] https://kubernetes.io/

subjacentes. Nesse contexto, o *Kubernetes*, uma solução *open source*, representa uma escolha viável. Além disso, a AWS oferece o *Amazon Elastic Container Service*[13], um serviço de gerenciamento de contêineres escalável.

- ***Streaming***: Os dados precisam ser compartilhados entre os serviços, demandando o uso de tecnologias que possibilitem a transmissão e o processamento contínuo dos dados. Nesse contexto, o Apache Kafka, uma solução *open source*, representa uma escolha viável. Além disso, a AWS oferece o Amazon Kinesis[14], um serviço que simplifica a captura, o processamento e a análise de dados de *streaming* em tempo real.

- **Nuvem pública e produtos *serverless***: Conforme a arquitetura se expande, a demanda operacional também cresce. Uma abordagem eficaz é transferir parte dessa carga para provedores de nuvem pública, os quais oferecem uma ampla gama de serviços gerenciados. Entre esses, destacam-se os serviços de nuvem rotulados como *serverless*, que permitem operar em um nível mais alto de abstração. Um exemplo notável é a *AWS Lambda*[15], uma variação da Plataforma como Serviço (PaaS), denominada Função como Serviço (FaaS), na qual é necessário apenas implantar o código, pois a inicialização de instâncias é gerida pelo provedor.

[13] https://docs.aws.amazon.com/ecs/
[14] https://aws.amazon.com/pt/kinesis/
[15] https://aws.amazon.com/pt/lambda/

3 *SISTEMA*

O autor deste trabalho, projetou e implementou o sistema de comércio eletrônico `CompraVirtual`. Sistemas de comércio eletrônico são amplamente reconhecidos por sua alta demanda de usuários. Embora inerente ao setor de comércio eletrônico, essa alta demanda impõe desafios significativos (HASSELBRING; STEINACKER, 2017; NETO, 2009). Portanto, o sistema deve ser projetado e implementado de tal forma que possa ser escalado facilmente para acomodar um grande número de usuários simultâneos, mantendo um desempenho consistente e rápido. Além disso, a disponibilidade contínua é essencial, uma vez que qualquer interrupção no sistema pode resultar em perda de vendas e consequentemente receitas, além da insatisfação dos clientes.

Na Seção 3.1, é descrito o sistema. Já na Seção 3.2, são apresentados os requisitos não-funcionais do sistema. Esses requisitos devem orientar o desenvolvimento e a implementação, garantindo que o sistema atenda às necessidades do negócio e aos padrões de qualidade estabelecidos.

3.1 Descrição

O sistema `CompraVirtual` segue uma arquitetura de microsserviços projetada para atender as funcionalidades de um comércio eletrônico com foco no gerenciamento de estoque.[1] De forma sucinta, essas funcionalidades foram divididas nos seguintes cinco microsserviços:

- **Autenticação e Autorização**: Gerencia o acesso de usuário;
- **Produtos**: Gerencia informações e imagens de produtos;
- **Pedidos**: Gerencia a criação, finalização e cancelamento de pedidos;
- **Estoque**: Gerencia o estoque; e
- **Notificações**: Oferece um sistema flexível de notificação por e-mail.

[1] Uma breve descrição de cada funcionalidade é elaborada, porém a documentação técnica detalhada das funcionalidades não foi elaborada pois não se encontra dentro do escopo deste trabalho. Em compensação, o código fonte de cada microsserviço está publicamente disponível em: <https://github.com/rafaelportomoura/ufla-tcc>.

3.2 Requisitos não-funcionais

Os Requisitos não-funcionais (RNFs) se concentram na qualidade geral de um sistema, em oposição às funcionalidades específicas que descrevem o que o software deve fazer. Em outras palavras, RNFs explicam como essas funcionalidades devem ser executadas, desempenhando um papel vital no sucesso do sistema (VERONEZ, 2022).

A Tabela 3.1 reporta os requisitos não-funcionais do sistema, juntamente com suas descrições, tendo sido obtidos a partir do catálogo estabelecido no trabalho de Veronez (2022). Os requisitos apresentados estão descritos de forma abstrata, uma vez que este trabalho não tem como objetivo testar empiricamente os requisitos não-funcionais. Assim, cada descrição deve ser interpretada como um direcionamento para a definição de requisitos mais específicos e mensuráveis. Além disso, as correlações entre os requisitos indicam influências diretas na estratégia de implementação, mas não necessariamente configuram relações de dependência ou precondições, por exemplo:

- A disponibilidade do sistema (RNF-DISP-1) está diretamente relacionada à escalabilidade automática (RNF-ESC-1), garantindo que os serviços permaneçam acessíveis sob alta demanda;
- A baixa latência na obtenção de imagens (RNF-DES-2) está associada à alta disponibilidade desses arquivos (RNF-DISP-3), uma vez que tempos de resposta reduzidos dependem da acessibilidade contínua dos recursos; e
- O uso de contêineres para escalabilidade (RNF-ESC-2) impacta diretamente a independência dos microsserviços (RNF-IND-3) e a segurança da plataforma (RNF-SEG-1), pois o isolamento dos serviços influencia tanto a proteção quanto a modularidade do sistema.

É importante mencionar que não houve testes empíricos, pois este estudo confia nas especificações e garantias providas pela AWS para cada serviço utilizado, assumindo que essas informações são suficientes para atender aos requisitos não-funcionais propostos. Além disso, a validação rigorosa desses requisitos exigiria uma infraestrutura dedicada e metodologias específicas, o que extrapolaria o escopo deste trabalho.

Tabela 3.1 – Requisitos não-funcionais do `CompraVirtual`

Categoria	Identificador	Descrição	Correlação
Escalabilidade	RNF-ESC-1	A escalabilidade dos microsserviços deve ser automatizada.	RNF-DISP-1 RNF-IMPL-2
	RNF-ESC-2	Utilizar contêineres para facilitar a escalabilidade.	RNF-IND-3 RNF-SEG-1
Disponibilidade	RNF-DISP-1	Garantir que os sistemas de bancos de dados e microsserviços estejam sempre disponíveis.	RNF-ESC-1
	RNF-DISP-2	Redirecionar requisições de forma balanceada para evitar sobrecarga.	n/a
	RNF-DISP-3	A taxa de falha ao obter imagens deve ser extremamente baixa.	RNF-DES-2
Evolucionalidade	RNF-EVOL-1	Garantir flexibilidade e a capacidade de responder à mudanças.	n/a
Desempenho	RNF-DES-1	As respostas das requisições devem ser rápidas e não causar percepção de lentidão ao usuário.	n/a
	RNF-DES-2	Arquivos de imagens devem ser obtidos com baixa latência.	RNF-DISP-3
Gerenciabilidade	RNF-GER-1	Garantir a configuração de infraestrutura como código.	RNF-IMPL-1
	RNF-GER-2	Utilizar ferramentas para execução de testes automatizados.	n/a
Implantação	RNF-IMPL-1	Automatizar a implantação de microsserviços.	RNF-GER-1
	RNF-IMPL-2	Utilizar ferramentas para o gerenciamento e orquestração das implantações.	RNF-ESC-1
Independência	RNF-IND-1	Microsserviços não devem compartilhar código fonte.	RNF-SEG-1
	RNF-IND-2	Não compartilhar bancos de dados entre microsserviços.	RNF-SEG-1
Monitoramento	RNF-MON-1	Monitoramento e rastreamento da comunicação interna e externa dos microsserviços.	n/a
	RNF-MON-2	Monitoramento da utilização dos recursos de hardware.	n/a
Segurança	RNF-SEG-1	A plataforma na qual um microsserviço é executado deve ser independente.	RNF-ESC-2 RNF-IND-1 RNF-IND-2
	RNF-SEG-2	Manter a comunicação segura entre microsserviços e clientes.	n/a

A disponibilidade do sistema é de extrema importância no contexto de comércios eletrônicos, pois indisponibilidades podem gerar impactos financeiros e na reputação da marca (NETO, 2009; SUGIANTORO; ANSHARI; SUDRAJAT, 2020). Nesse contexto, o requisito RNF-DISP-1 garante que os microsserviços e banco de dados se recuperem rápido de falhas e permaneçam o menor tempo possível indisponíveis, e o requisito RNF-DISP-2 garante que as requisições sejam redirecionadas de forma balanceada evitando a sobrecarga. Porém, em momentos de picos de tráfego, para manter a disponibilidade é indispensável a escalabilidade do sistema, com isso o requisito RNF-ESC-2 permite que novas instâncias de microsserviços sejam geradas rapidamente e os requisitos RNF-ESC-1 e RNF-IMPL-2 garantem que a escalabilidade seja feita de forma eficaz e eficiente, ajudando na otimização dos custos de infraestrutura, pois permitem aumentar e reduzir o número de instâncias conforme necessário.

No processo de compra, a capacidade de obter rapidamente as imagens dos produtos desempenha um papel crucial. Qualquer atraso nesse processo pode resultar na frustração dos usuários, levando-os a abandonar suas compras (NETO, 2009; SUGIANTORO; ANSHARI; SUDRAJAT, 2020). Visando evitar tal cenário, os requisitos RNF-DISP-3 e RNF-DES-2 garantem uma resposta ágil aos usuários, para proporcionar uma experiência de compra fluida e satisfatória.

O requisito RNF-DES-1 visa aprimorar o desempenho do sistema, reduzindo o tempo de resposta em requisições que não dependem estritamente de execução síncrona, proporcionando um desempenho mais ágil.

Quando se trata do suporte e do tratamento de falhas, é fundamental ter uma visão abrangente do sistema como um todo. Com essa finalidade, os requisitos RNF-MON-1 e RNF-MON-2 garantem que o técnico de suporte tenha facilidade em identificar a origem das falhas e corrigi-las de forma eficiente.

A segurança é um fator crítico para o sistema, pois brechas podem resultar em prejuízos significativos (NETO, 2009; SUGIANTORO; ANSHARI; SUDRAJAT, 2020). Nesse contexto, foi definido o requisito RNF-SEG-1 que garante a independência da plataforma em que os microsserviços operam, assegurando sua capacidade de funcionar de forma autônoma e resiliente. Esse requisito está intimamente ligado ao requisito RNF-ESC-2 que garante execuções em plataformas isoladas e aos requisitos RNF-IND-1 e RNF-IND-2 que garantem uma clara separação entre os microsserviços. Além disso, o requisito RNF-SEG-2 tem como objetivo manter a comunicação segura entre microsserviços e clientes para proteger os dados sensíveis e a confidencialidade das informações transmitidas.

A etapa de integração e implantação do software desempenha um papel essencial para tornar a gestão da arquitetura de microsserviços mais fácil (EBERT et al., 2016). Portanto, é crucial aderir aos requisitos RNF-GER-1, RNF-GER-2 e RNF-IMPL-1. A automação de testes proporciona confiabilidade ao código desenvolvido, enquanto a configuração da infraestrutura como código assegura uma implantação automatizada. Isso não apenas reduz os riscos humanos, mas também aumenta a agilidade do processo, contribuindo para uma implementação mais eficiente e menos propensa a erros.

O sistema deve ser flexível e estar preparado para mudanças que surgirem (FOWLER; LEWIS, 2014), logo é importante o requisito RNF-EVOL-1, para garantir que o sistema esteja em constante evolução.

4 PROJETO E IMPLEMENTAÇÃO

Newman (2022) argumenta que conhecer as tecnologias é essencial para que a implementação de microsserviços seja um sucesso. Dito isso, o autor deste trabalho projetou e implementou o sistema `CompraVirtual` orientado à nuvem AWS, a qual oferece mais de 200 serviços (AWS, 2024z), o que torna difícil selecionar a tecnologia correta para cada necessidade. Neste capítulo, são apresentadas as arquiteturas do sistema e as tecnologias utilizadas.

O tempo de projeto e implementação do sistema deste trabalho não foi mensurado pelo autor por meio de uma metodologia específica. No entanto, o período entre o início do desenvolvimento e a entrega da última funcionalidade totalizou cinco meses, com uma média de quatro horas semanais dedicadas ao projeto.

Na Seção 4.1, são apresentadas as arquiteturas do sistema e suas tecnologias. Na Seção 4.2, são apresentadas as arquiteturas de cada microsserviço e suas tecnologias. Por fim, na Seção 4.3, são relacionadas as arquiteturas e tecnologias utilizadas com os requisitos não-funcionais.

4.1 Arquitetura do sistema

Nesta seção, são apresentadas as arquiteturas dos serviços cruciais para o ecossistema dos microsserviços. Para facilitar a compreensão, as tecnologias e arquiteturas foram agrupadas pelas funções que exercem:

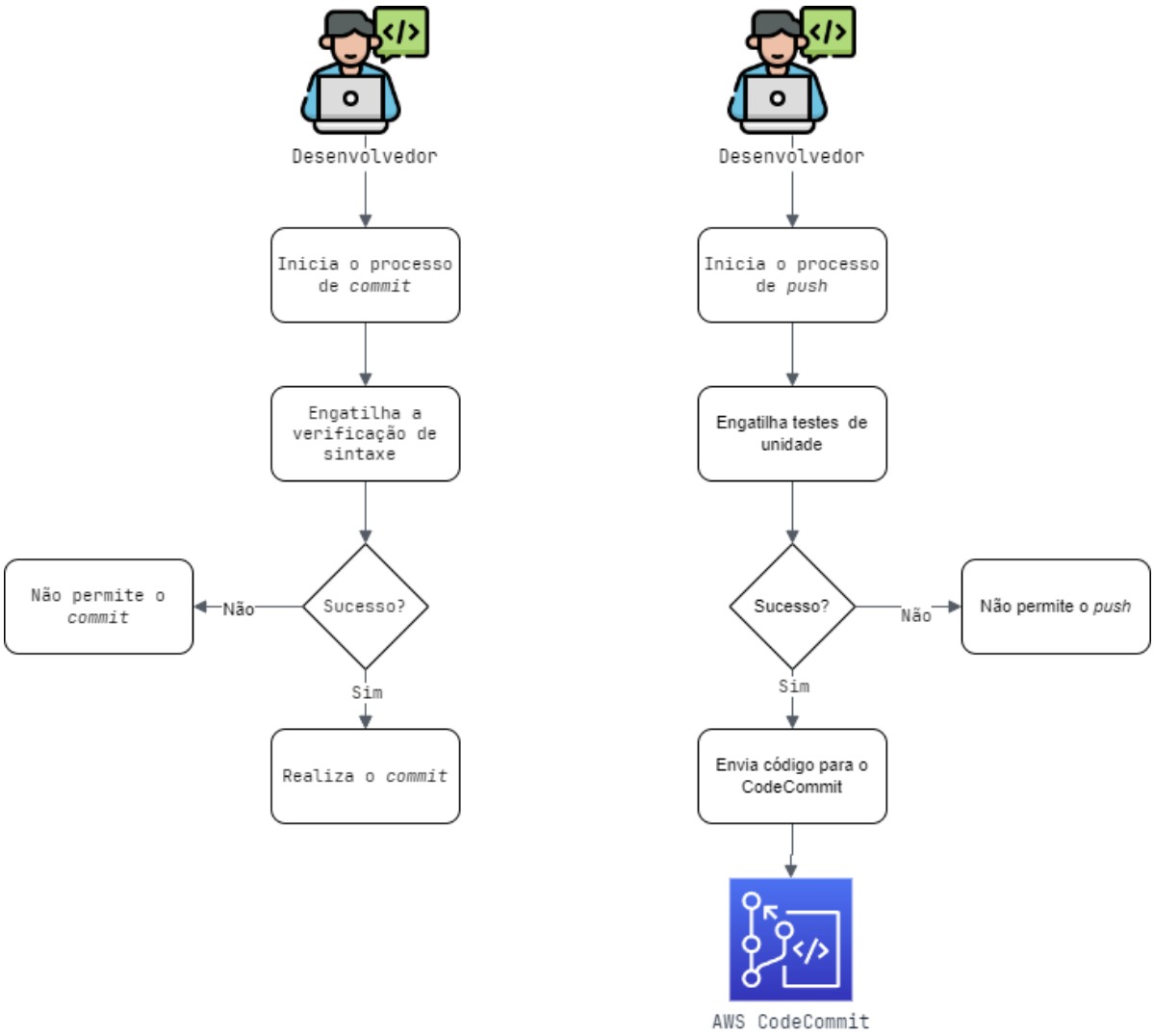
- **Desenvolvimento**: Devido à governança descentralizada, o desenvolvimento de microsserviços ocorre de maneira mais rápida em comparação com sistemas monolíticos (NEWMAN, 2022). No entanto, conforme argumentado por Fowler (2017), a falta de padronização pode gerar atrasos no desenvolvimento e dificultar a gestão do sistema. No sistema `CompraVirtual`, o autor implementou a padronização da nomenclatura dos serviços. Além disso, foram padronizadas as *tags* para melhorar a gestão de custos na Nuvem (*FinOps*). A linguagem TypeScript[1] foi escolhida como padrão de desenvolvimento devido à maior afinidade do autor com essa tecnologia.

  O processo de integração também foi padronizado para todos os microsserviços, utilizando o sistema de versionamento Git[2] em conjunto com o servidor AWS CodeCommit (AWS, 2024l) e o gerenciador de gatilhos Husky[3]. Conforme detalhado na Figura 4.1, o Husky permite a execução de *scripts* ao acionar comandos do Git, criando fluxos automáticos de integração.

  - Processo de *Commit*:
    - O desenvolvedor inicia o processo de *commit*, que aciona a verificação de sintaxe através do ESLint[4].

[1] https://www.typescriptlang.org/
[2] https://www.git-scm.com/
[3] https://typicode.github.io/husky/
[4] https://eslint.org/

Figura 4.1 – Processo de integração do código do sistema `CompraVirtual`



Fonte: Autor

- Caso a verificação de sintaxe seja bem-sucedida, o *commit* é realizado.
- Se a verificação falhar, o *commit* não é permitido, o que garante que apenas código com sintaxe correta seja integrado.

– Processo de *Push*:

- O desenvolvedor inicia o processo de *push*, que aciona a execução de testes de unidade através do Mocha[5].

[5] https://mochajs.org/

    - Se todos os testes forem bem-sucedidos, o código é enviado para o AWS CodeCommit.
    - Caso algum teste falhe, o *push* não é permitido, o que garante que apenas código funcional e testado seja integrado ao repositório.

  Esse fluxo garante a integridade e a qualidade do código, padronizando as etapas de desenvolvimento e facilita a gestão do ecossistema de microsserviços no `CompraVirtual`.

- **Implantação Contínua**: De acordo com Newman (2022), um fator crucial para o sucesso da implementação é a infraestrutura estar definida como código. Para isso, foi utilizado o serviço AWS CloudFormation (AWS, 2024i) que permite declarar os serviços da AWS como código. Isso facilita a implantação com diferentes parâmetros dos mesmos serviços.

  O software Jenkins[6] é utilizado para a criação de *pipelines* que são acionadas após alterações no ramo do AWS CodeCommit, garantindo um processo de implantação contínua eficiente e automatizado. Esse fluxo assegura a integridade e a qualidade do código, padroniza as etapas de desenvolvimento e facilita a gestão do ecossistema de microsserviços. Para hospedar o Jenkins, é utilizada uma instância EC2 (AWS, 2024n) com as permissões necessárias para criação dos recursos na conta AWS.

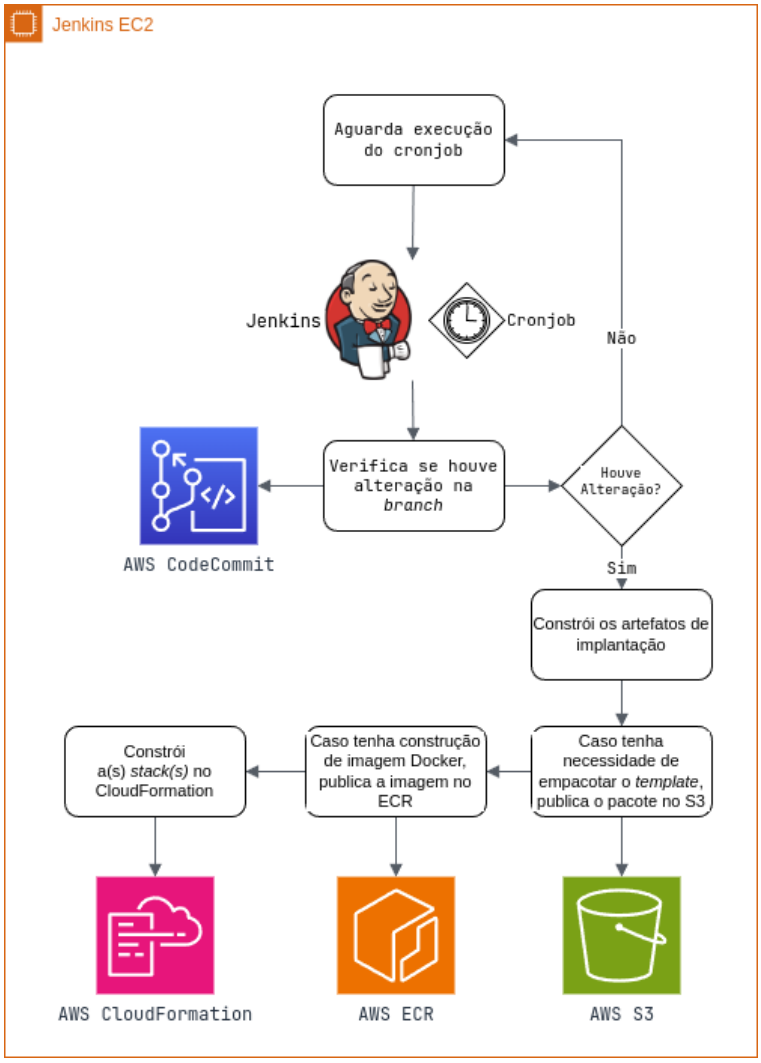
  A Figura 4.2 ilustra processo de implantação dos serviços que envolve as seguintes etapas:

    1. Execução do *cronjob*: Em um determinado intervalo de tempo, cada *pipeline* verifica se há alteração no ramo (*branch*) do repositório no AWS CodeCommit.
    2. Construção de artefatos de implantação: Se houver alterações, o Jenkins constrói os artefatos de implantação necessários.
    3. Publicação de artefatos no S3: Caso tenha a necessidade de empacotar o *template* do CloudFormation, o pacote é publicado no AWS S3 (AWS, 2024c).
    4. Publicação da imagem Docker: Se houver construção de imagem Docker, a imagem é publicada no AWS ECR (AWS, 2024o).
    5. Construção de *stacks* no CloudFormation: Os recursos AWS são criados no formato de pilha (*Stack*[7]) no AWS CloudFormation facilitando a gestão dos recursos.

- ***Virtual Private Cloud***: A *Virtual Private Cloud* (VPC) é a rede virtual privada oferecida pela AWS, que proporciona posicionamento de recursos, conectividade e segurança (AWS, 2024y). A VPC configurada nesse projeto está dividida em duas sub-redes (*subnets*): uma privada e outra pública.

[6] https://www.jenkins.io/
[7] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html

Figura 4.2 – Processo de implantação contínua do sistema `CompraVirtual`



Fonte: Autor

Na sub-rede privada, os serviços computacionais e de dados são hospedados sem IP público, garantindo maior segurança. Por outro lado, a sub-rede pública hospeda serviços responsáveis pela comunicação externa, como o *NAT Gateway*, que realiza a tradução de IP privado para IP público. A comunicação externa com alguns serviços da AWS ocorre através do roteamento das requisições para o serviço de *Endpoints* e a comunicação com a Internet é realizada por meio do roteamento para o *InternetGateway*.

Os serviços ACL (*Access Control List*) e *SecurityGroup* reforçam a segurança da rede. O ACL permite ou bloqueia requisições com base na porta para cada sub-rede, enquanto o *SecurityGroup* controla o acesso aos recursos permitindo apenas determinadas faixas de IPs e portas.

A Figura 4.3 contém relacionamentos que representam o fluxo de comunicação e roteamento dentro da VPC:

  - Relacionamento entre o *InternetGateway* e o *Router*: indica o fluxo de dados entre a Internet e a VPC por meio do *InternetGateway*, permitindo a comunicação de entrada e saída com recursos na sub-rede pública.
  - Relacionamento entre o *Router* e as Sub-redes (Privada e Pública): mostra o roteamento interno dentro da VPC, conectando a sub-rede privada e a sub-rede pública por meio do roteador, que gerencia o tráfego entre as sub-redes.
  - Relacionamento entre a Sub-rede Pública e o *NAT Gateway*: representa a comunicação da sub-rede privada com a Internet por meio do *NAT Gateway* localizado na sub-rede pública. O *NAT Gateway* permite que instâncias na sub-rede privada acessem a Internet sem expor seus IPs privados.
  - Relacionamento entre o *Router* e os *Endpoints*: indica o roteamento das requisições de serviços específicos da AWS diretamente por meio de *endpoints*, facilitando o acesso seguro e eficiente a esses serviços sem sair da rede AWS.

- **Resolução de domínios e emissão de certificados**: O serviço Route53 (AWS, 2024t) é responsável pela resolução de domínio, enquanto o serviço AWS Certificate Manager (AWS, 2024g) emite os certificados de segurança necessários para a comunicação via HTTPS[8]. Esses dois serviços operam em conjunto com o AWS API Gateway (AWS, 2024d), o qual gerencia a comunicação entre o cliente e o ecossistema de microsserviços. O API Gateway permite a utilização de funções AWS Lambda (AWS, 2024s) para autenticação do usuário, além de disponibilizar *logs* para melhor monitoramento das *APIs*.

  Para a comunicação interna dos microsserviços, o API Gateway e a emissão de certificados não são utilizados. Em vez disso, o serviço AWS Cloud Map (AWS, 2024h) facilita a descoberta de serviços e a malha de serviço AWS App Mesh (AWS, 2024e) gerencia a comunicação entre eles. O AWS Fargate (AWS, 2024q) é utilizado para hospedar os servidores, o qual é detalhado na Seção 4.2.

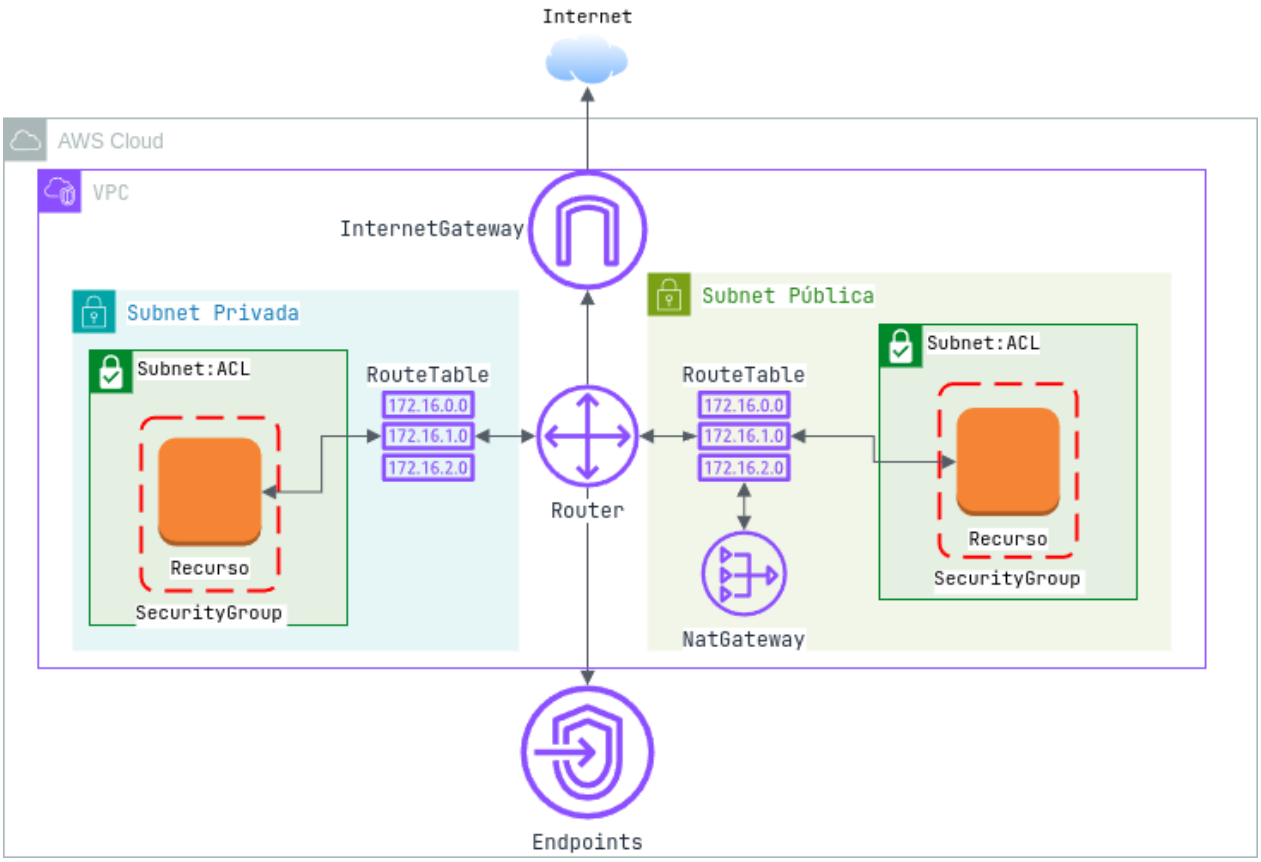
  A Figura 4.4 ilustra a arquitetura utilizada para resolução de domínios e emissão de certificados.

  Cliente externo:

  1. Cliente para AWS Route53: O cliente solicita a resolução de domínio ao AWS Route53, que resolve o domínio, direcionando ao API Gateway.
  2. Cliente para API Gateway: O cliente solicita uma conexão HTTPS ao API Gateway que utiliza o certificado do AWS Certificate Manager para iniciar o protocolo.

[8] O HTTPS é uma forma segura de enviar dados entre um servidor web e um navegador web (CLOUDFARE, 2024).

Figura 4.3 – Arquitetura da rede virtual



Fonte: Autor

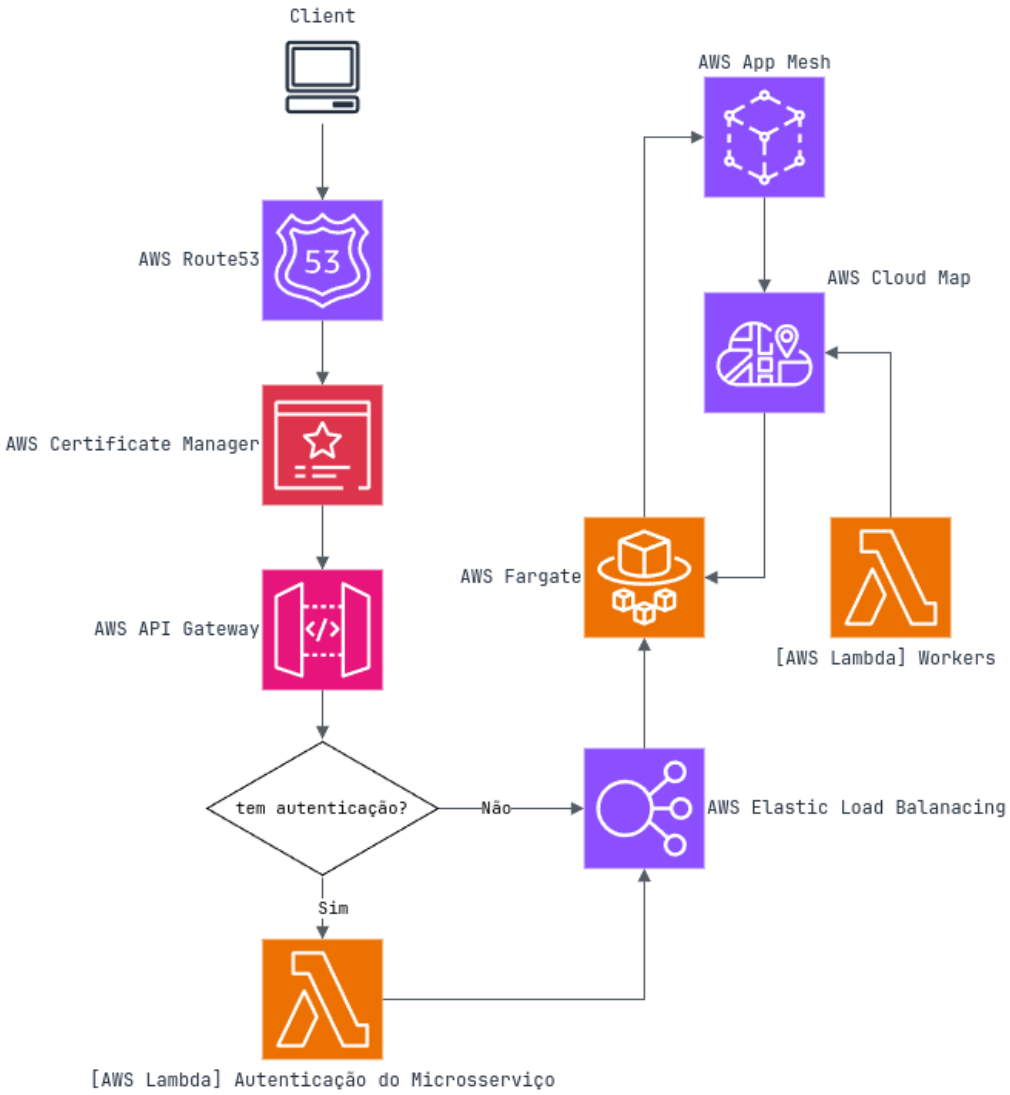
3. API Gateway verifica se há autenticação: Caso tenha autenticação, uma função AWS Lambda é responsável por verificar se o *token* de acesso contém permissão para acessar o recurso. Caso não tenha permissão, é retornado um erro de autorização ao usuário. Entretanto, se não houver autenticação ou se o cliente estiver autorizado a acessar o recurso, o API Gateway encaminha a solicitação ao AWS Elastic Load Balancing (ELB) (AWS, 2024p).

4. AWS ELB para AWS Fargate: O ELB encaminha a solicitação ao contêiner do microsserviço hospedado no AWS Fargate que envia a resposta ao usuário encerrando o processo.

Cliente interno:

1. Lambda para AWS CloudMap: O cliente solicita a resolução de domínio ao AWS CloudMap, que resolve o domínio, direcionando ao AWS Fargate.

Figura 4.4 – Arquitetura da resolução de domínios e emissão de certificados



Fonte: Autor

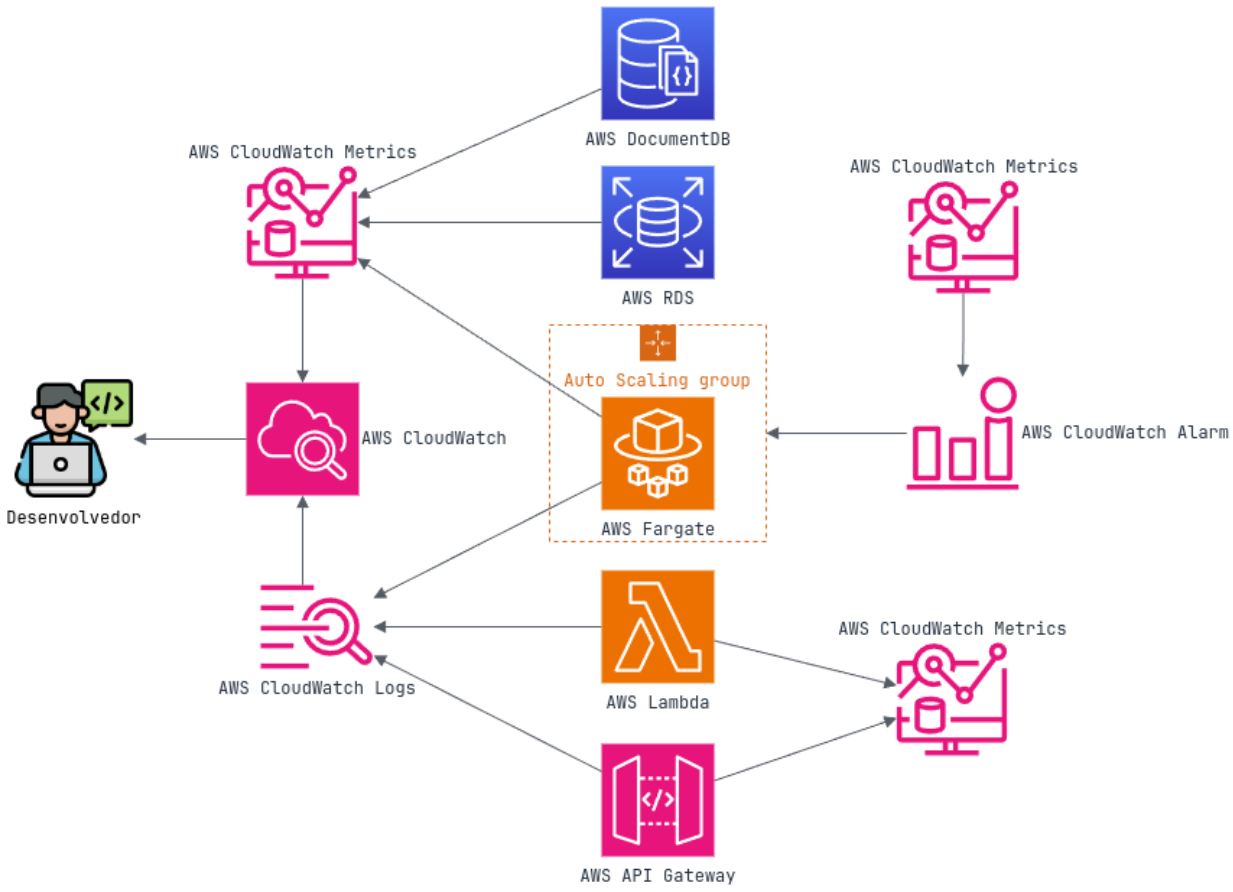
2. AWS Fargate ao App Mesh: O contêiner do microsserviço faz a requisição e o *side-car* do Envoy[9] encaminha a requisição ao AWS App Mesh que realiza a descoberta do domínio com o AWS Cloud Map e efetua a requisição no microsserviço.

- **Monitoramento**: O monitoramento das aplicações e serviços é uma parte crucial da manutenção de um sistema de microsserviços confiável (ZHOU et al., 2023). O Amazon CloudWatch desempenha um papel fundamental nesse aspecto ao fornecer uma plataforma robusta para monitorar e gerenciar *logs*, métricas e eventos de diversos

[9] https://www.envoyproxy.io/

serviços AWS. Por meio do CloudWatch, é possível coletar e rastrear métricas em tempo real, definir alarmes, visualizar gráficos e obter percepções detalhados sobre o desempenho e a saúde dos sistemas. Além disso, o CloudWatch Logs permite a agregação, monitoramento e análise dos *logs* das aplicações, facilitando a detecção e resolução de problemas (AWS, 2024k).

Figura 4.5 – Arquitetura de monitoramento



Fonte: Autor

A Figura 4.5 ilustra a arquitetura de monitoramento utilizando o CloudWatch. A imagem demonstra como o Amazon CloudWatch coleta métricas e *logs* de diferentes serviços da AWS para fornecer uma visão abrangente do desempenho e da saúde dos sistemas. Uma explicação detalhada dos componentes e seus fluxos se encontra a seguir:

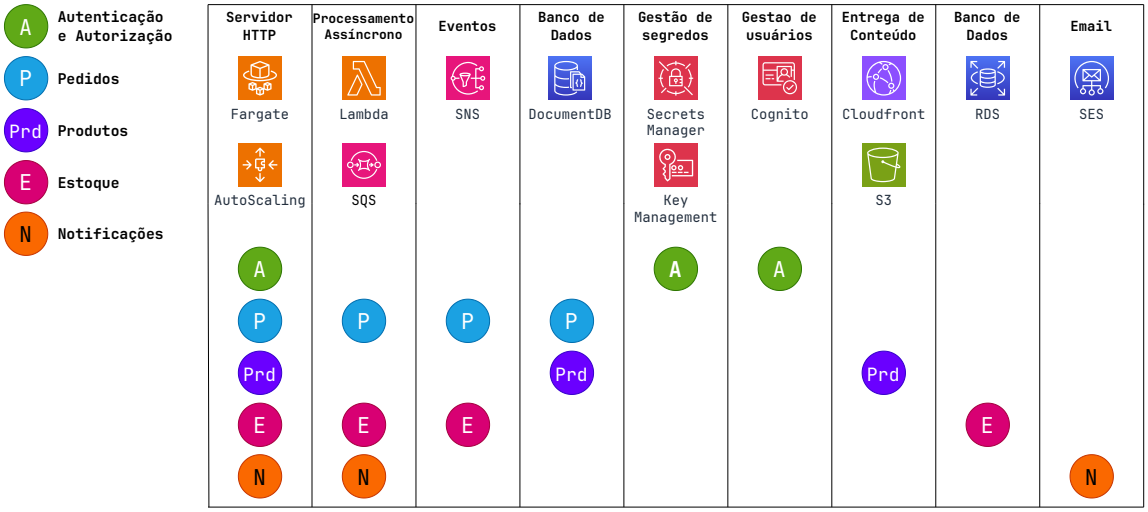
1. **AWS CloudWatch Metrics**: serviço de monitoramento que coleta e rastreia métricas dos recursos da AWS, permitindo a visualização em tempo real, definição de alarmes e resposta automática a mudanças. No projeto foi implementado para monitorar os seguintes recursos:
   - **AWS DocumentDB** (AWS, 2024a): Coleta métricas de desempenho e utilização do AWS DocumentDB, permitindo monitorar o banco de dados orientado a documentos.
   - **AWS RDS** (AWS, 2024b): Coleta métricas de bancos de dados relacionais hospedados no AWS RDS, proporcionando percepções sobre consultas, uso de CPU e outros parâmetros críticos.
   - **AWS Fargate**: Coleta métricas de contêineres gerenciados pelo AWS Fargate, monitorando sua utilização de recursos e desempenho.
   - **AWS Lambda**: Coleta métricas de execução das funções AWS Lambda, incluindo duração das execuções e número de invocações.
   - **AWS API Gateway**: Coleta métricas do AWS API Gateway, como latência de resposta e contagem de solicitações.
2. **AWS CloudWatch Logs**: Os *logs* gerados pelas APIs do API Gateway, pelos contêineres do AWS Fargate e pelas funções do AWS Lambda são agregados no CloudWatch Logs, facilitando a análise e o diagnóstico de problemas.
3. **AWS CloudWatch Alarm**: Baseado nas métricas coletadas, o CloudWatch pode acionar alarmes que desencadeiam ações automáticas, como escalonamento feito pelo AWS Auto Scaling, que gerencia a escalabilidade dos contêineres no AWS Fargate, garantindo que a capacidade da aplicação ajuste-se automaticamente conforme a demanda.

4.2 Arquitetura dos microsserviços

A arquitetura de microsserviços do sistema `CompraVirtual` é composta por serviços independentes, cada um responsável por uma funcionalidade específica e utilizando tecnologias da AWS para garantir escalabilidade, segurança e eficiência. A Figura 4.6 apresenta graficamente a relação entre microsserviços e as tecnologias usadas na arquitetura.
Microsserviço Autenticação e Autorização:

- **Servidor HTTP**: Utiliza AWS Fargate (AWS, 2024q) que facilita a gestão de contêineres sem a necessidade de provisionamento de servidores. Esse serviço juntamente com o AWS Auto Scaling (AWS, 2024f) permite que as operações de escalabilidade e gerenciamento sejam automatizadas, proporcionando alta disponibilidade e eficiência na alocação de recursos.

Figura 4.6 – Microsserviços x Tecnologias



- **Gestão de Segredos**: AWS Secrets Manager (AWS, 2024u) e AWS Key Management Service (KMS) (AWS, 2024r) são utilizados para armazenar e gerenciar segredos, como credenciais e chaves criptográficas. O uso dessas tecnologias garante a segurança e a proteção dos dados sensíveis armazenados e acessados pelo sistema.
- **Gestão de Usuários**: Implementado com AWS Cognito (AWS, 2024m) que fornece funcionalidades para cadastro, autenticação e autorização de usuários. AWS Cognito facilita o gerenciamento de identidades de usuários, assegurando que somente usuários autorizados possam acessar os recursos apropriados do sistema.

Microsserviço Pedidos:

- **Servidor HTTP**: Semelhante ao microsserviço de Autenticação e Autorização.
- **Eventos**: AWS SNS (*Simple Notification Service*) (AWS, 2024w) e AWS SQS (*Simple Queue Service*) (AWS, 2024x) são utilizados para comunicação assíncrona entre os componentes do sistema, enquanto funções AWS Lambda processam os eventos relacionados aos pedidos. Essa combinação permite uma arquitetura desacoplada, onde a comunicação entre serviços é robusta e eficiente.
- **Banco de dados**: AWS DocumentDB (AWS, 2024a) é usado para armazenar informações sobre os pedidos. DocumentDB é compatível com MongoDB[10], oferecendo uma solução de banco de dados gerenciado que é altamente escalável e flexível para dados semi-estruturados.

Microsserviço Produtos:

- **Servidor HTTP**: Semelhante ao microsserviço de Autenticação e Autorização.

[10] https://www.mongodb.com/pt-br

- **Banco de dados**: Semelhante ao microsserviço de Pedidos.
- **Entrega de conteúdo**: AWS S3 (AWS, 2024c) é utilizado para armazenar imagens de produtos e outros conteúdos estáticos, enquanto AWS CloudFront serve como CDN[11] para distribuir esses recursos globalmente com baixa latência.

Microsserviço Estoque:

- **Servidor HTTP**: Semelhante ao microsserviço de Autenticação e Autorização.
- **Eventos**: Semelhante ao microsserviço de Pedidos.
- **Banco de dados**: AWS RDS (AWS, 2024b) é utilizado para armazenar dados de estoque. O RDS proporciona uma solução de banco de dados relacional gerenciado, com suporte para *backup* automatizado e alta disponibilidade.

Microsserviço Notificações:

- **Servidor HTTP**: Semelhante ao microsserviço de Autenticação e Autorização.
- **Processamento Assíncrono**: AWS Lambda (AWS, 2024s) e AWS SQS (AWS, 2024x) são utilizados para processamento assíncrono de mensagens, permitindo a escalabilidade e resposta rápida às atividades de notificação. Diferente do microsserviço de Pedidos, pois nesse caso o envio da mensagem é direto para a fila SQS, já no microsserviço de Pedidos, a fila SQS é inscrita em um tópico SNS e recebe apenas mensagens que correspondem ao filtro da inscrição.
- **Envio de e-mail**: AWS *Simple Email Service* (SES) (AWS, 2024v) é utilizado para enviar e-mails, garantindo alta entrega e segurança na comunicação com os usuários.

4.3 Análise e discussão

Nesta seção, é apresentada uma análise detalhada das tecnologias AWS utilizadas no projeto `CompraVirtual`, correlacionando-as com os requisitos não-funcionais previamente estabelecidos na Seção 3.2. A Tabela 4.1 relaciona cada tecnologia com os requisitos não-funcionais relacionados, explicando de forma objetiva como cada tecnologia contribui para atender às necessidades de disponibilidade, desempenho, segurança, monitoramento, gerenciamento, escalabilidade, evolução e implantação do sistema.

A tabela serve como um guia para entender como a arquitetura de microsserviços do `CompraVirtual` utiliza as ferramentas e serviços da AWS para garantir que os objetivos de qualidade e eficiência sejam alcançados. Esse

[11] Uma CDN (*Content Delivery Network*) é uma infraestrutura composta por diversos servidores, projetada para facilitar o acesso rápido e seguro aos conteúdos online, distribuindo-os eficientemente pela rede (HOSTINGER, 2024).

mapeamento detalhado é essencial para assegurar que todas as áreas críticas do sistema sejam cobertas por soluções robustas e bem integradas. Dessa forma, é possível garantir que os requisitos de qualidade do sistema sejam atendidos, proporcionando um funcionamento estável e seguro do `CompraVirtual`.

Tabela 4.1 – Tecnologias x Requisitos não-funcionais do `CompraVirtual`

Tecnologia	**Descrição**	**RNF**
AWS Fargate	Permite escalabilidade automática de contêineres, ajustando a capacidade com base na demanda, garantindo alta disponibilidade e eficiência.	RNF-DISP-1, RNF-ESC-1
AWS Auto Scaling	Ajusta automaticamente a capacidade dos serviços, assegurando que os recursos estejam disponíveis conforme necessário para manter desempenho e disponibilidade.	RNF-DISP-1, RNF-ESC-1
AWS RDS	Fornece um banco de dados gerenciado com *backups* automáticos e recuperação de falhas, garantindo disponibilidade e durabilidade dos dados.	RNF-DISP-1, RNF-IND-2
AWS DocumentDB	Provê um banco de dados de documentos gerenciado, compatível com MongoDB, que oferece alta disponibilidade, escalabilidade e durabilidade	RNF-DISP-1, RNF-IND-2.
AWS ELB	Redireciona requisições de forma balanceada, evitando sobrecarga e assegurando a distribuição eficiente de tráfego entre os recursos.	RNF-DISP-2, RNF-ESC-1
AWS API Gateway	Gerencia requisições externas, fornecendo roteamento eficiente, autenticação e autorização, e melhorando a segurança e o desempenho das APIs.	RNF-DES-1, RNF-SEG-2
AWS Lambda	Processa eventos e execuções em segundo plano, escalando automaticamente com a demanda e garantindo resiliência através de funções *Serverless*.	RNF-DES-1, RNF-ESC-1
AWS CloudFront	Acelera a entrega de conteúdo ao usar uma rede de distribuição global, melhorando o desempenho e a disponibilidade de arquivos estáticos.	RNF-DES-2, RNF-DISP-3
AWS Cognito	Gerencia a autenticação e autorização dos usuários, assegurando que somente usuários verificados possam acessar determinados serviços e recursos.	RNF-SEG-1
AWS KMS	Proporciona criptografia de dados, garantindo a segurança em trânsito e em repouso.	RNF-SEG-1, RNF-SEG-2
AWS Secrets Manager	Armazena e gerencia segredos como credenciais e chaves de API de forma segura.	RNF-SEG-1
AWS CloudWatch	Monitora a saúde e o desempenho dos serviços, envia alertas em tempo real e agrega *logs* para análise detalhada.	RNF-MON-1, RNF-MON-2
AWS CloudFormation	Permite definir e provisionar infraestrutura como código, facilitando a gestão e atualização de recursos e automatizando a recuperação.	RNF-GER-1, RNF-IMPL-1, RNF-IMPL-2
AWS App Mesh	Gerencia o tráfego entre microsserviços, garantindo roteamento eficiente, segurança e visibilidade de ponta a ponta.	RNF-ESC-1, RNF-SEG-2
AWS Cloud Map	Permite a descoberta dinâmica de serviços , ajudando na escalabilidade e na flexibilidade do sistema.	RNF-ESC-1
AWS SQS	Garante a comunicação assíncrona entre serviços, armazenando e entregando mensagens de forma confiável.	RNF-DES-1
AWS SNS	Facilita a publicação de eventos desacoplados, permitindo uma comunicação eficiente e escalável.	RNF-DES-1
AWS S3	Armazena imagens de produtos e outros conteúdos estáticos com alta disponibilidade e baixa latência.	RNF-DISP-3, RNF-DES-2
AWS SES	Envia e-mails de forma eficiente, garantindo alta taxa de entrega e segurança na comunicação com os usuários.	RNF-DES-1
AWS CodeCommit	Consite em um repositório de código gerenciado que facilita a colaboração em projetos de desenvolvimento de software, permitindo versionamento e controle de mudanças.	RNF-EVOL-1, RNF-GER-1, RNF-IND-1
Jenkins	Consiste em uma ferramenta de integração contínua e entrega contínua (CI/CD) que automatiza o processo de construção, teste e implantação de aplicações, garantindo uma *pipeline* eficiente e consistente.	RNF-IMPL-1, RNF-IMPL-2, RNF-GER-2
AWS EC2	Consiste em um serviço de computação em nuvem utilizado para hospedar o Jenkins, proporcionando um ambiente escalável e confiável para a automação de CI/CD.	RNF-IMPL-1, RNF-IMPL-2
AWS ECR	Consiste em um repositório gerenciado de contêineres Docker que facilita o armazenamento, gerenciamento e implantação de imagens de contêineres de forma segura e escalável.	RNF-IMPL-2, RNF-ESC-2
AWS VPC	Proporciona isolamento de rede, permitindo a definição de sub-redes, tabelas de roteamento e *gateways* de rede para controlar o tráfego de entrada e saída.	RNF-SEG-2, RNF-SEG-3

Tecnologia	Descrição	RNF
AWS Route 53	Consiste em um serviço de DNS escalável e de alta disponibilidade que roteia os usuários finais para aplicações da Internet, com baixa latência e alta confiabilidade.	RNF-DISP-2, RNF-DISP-3
AWS ACM	Gerencia certificados SSL/TLS para proteger a comunicação entre os serviços, proporcionando segurança em trânsito.	RNF-SEG-1, RNF-SEG-2

A partir da análise realizada na Tabela 4.1, é possível identificar como as tecnologias da AWS foram cuidadosamente selecionadas e configuradas para atender aos requisitos não-funcionais do sistema `CompraVirtual`. Cada tecnologia desempenha um papel fundamental na garantia de aspectos cruciais como disponibilidade, desempenho, segurança, escalabilidade e manutenção do sistema.

A escolha de tecnologias como AWS Fargate, AWS Auto Scaling e AWS CloudWatch destaca a preocupação em manter uma infraestrutura que se ajuste automaticamente às demandas variáveis do sistema, garantindo que os recursos estejam sempre disponíveis quando necessário, sem comprometer o desempenho. Além disso, a utilização de serviços como AWS Cognito, AWS KMS e AWS ACM reforça o compromisso com a segurança, assegurando que os dados e as comunicações sejam protegidos tanto em trânsito quanto em repouso.

As tecnologias de comunicação desempenham um papel essencial na arquitetura de microsserviços do sistema `CompraVirtual`. O AWS API Gateway gerencia o tráfego de requisições externas, enquanto o AWS App Mesh assegura a comunicação interna entre microsserviços com visibilidade e controle aprimorados. Já o AWS SQS permite a comunicação assíncrona, desacoplando os serviços e proporcionando maior resiliência e escalabilidade à arquitetura. Esses componentes de comunicação são fundamentais para manter a integridade e o desempenho do sistema.

O AWS CloudFront, por sua vez, desempenha um papel crucial na entrega rápida e segura de conteúdos estáticos, como imagens e arquivos de mídia, para os usuários do sistema `CompraVirtual`. Esse serviço de *Content Delivery Network* (CDN) distribui o conteúdo de forma global, reduzindo a latência e melhorando a experiência do usuário final. Ao utilizar pontos de presença (PoPs) distribuídos geograficamente, o CloudFront garante que o conteúdo seja servido a partir do local mais próximo ao usuário, aumentando a eficiência e a rapidez do sistema, além de proporcionar maior resiliência contra falhas (AWS, 2024j).

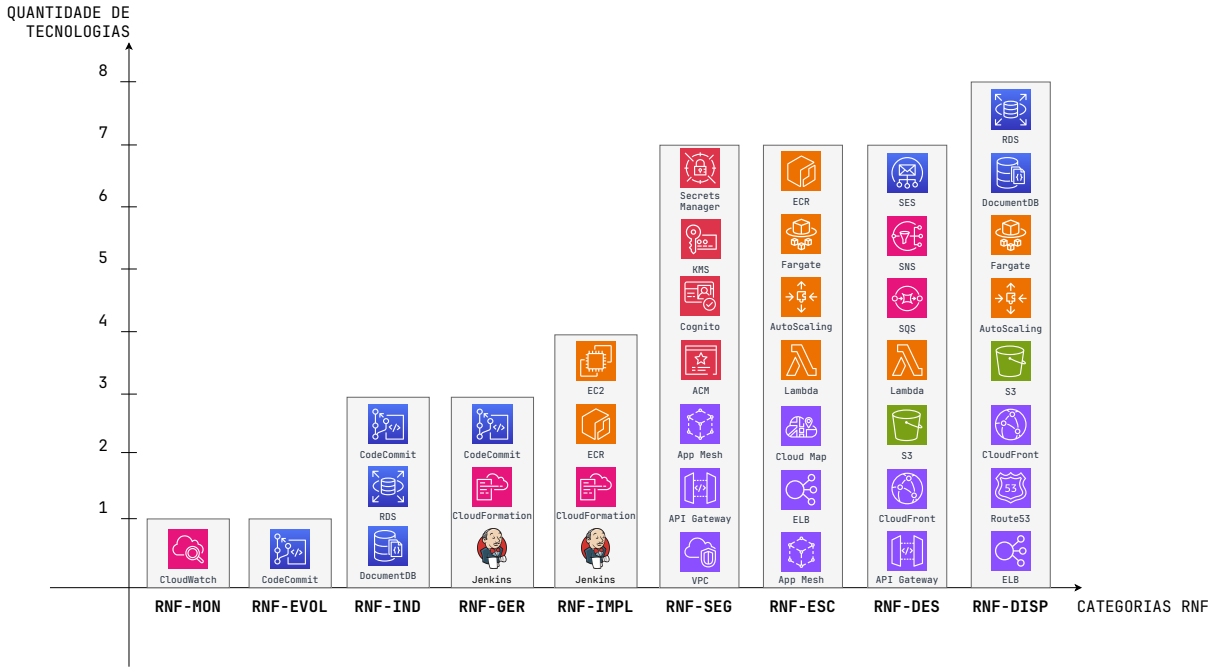
As tecnologias escolhidas também mostram uma preocupação com a flexibilidade e a evolução contínua do sistema. Ferramentas como AWS CloudFormation e AWS CodeCommit, em conjunto com práticas de infraestrutura como código (IaC), permitem que o sistema seja facilmente mantido e atualizado, minimizando o risco de erros humanos e garantindo a consistência nas operações.

A implementação dessas tecnologias garante o cumprimento integral de todos os requisitos não-funcionais estabelecidos. Neste trabalho, a intenção do autor é abordar o maior número de tecnologias AWS, porém é bom ressaltar que adotar muitas tecnologias novas ao iniciar o trabalho com microsserviços pode ter um impacto contraproducente (NEWMAN, 2022).

Ao analisar a Figura 4.7, observa-se que os requisitos de evolucionabilidade e monitoramento possuem apenas uma tecnologia associada cada. No caso de evolucionabilidade, a tecnologia utilizada foi AWS CodeCommit, enquanto para monitoramento foi adotada AWS CloudWatch. Esse resultado sugere que a satisfação desses requisitos requerem menos dependência de ferramentas específicas.

Por outro lado, os requisitos de disponibilidade, desempenho, escalabilidade e segurança, que demandam um conjunto mais diversificado de tecnologias para garantir sua implementação apresentam a maior quantidade de tecnologias associadas, totalizando oito soluções para a categoria disponibilidade e sete soluções para desempenho, escalabilidade e segurança.

Figura 4.7 – Tecnologias por categorias de RNFs



Fonte: Autor

A Figura 4.8 apresenta um grafo composto por dois tipos de vértices: nas extremidades, encontram-se as tecnologias empregadas no sistema `CompraVirtual`, enquanto no centro estão os requisitos não funcionais estabelecidos na Seção 3.2.

Ao analisar a Figura 4.8, é possível observar uma distribuição variada no atendimento dos requisitos não-funcionais pelas tecnologias empregadas no sistema `CompraVirtual`. O requisito RNF-ESC-1 se destaca por ser amplamente suportado por múltiplas tecnologias, indicando a prioridade dada à escalabilidade do sistema, que é fundamental para acomodar variações de demanda e assegurar desempenho consistente.

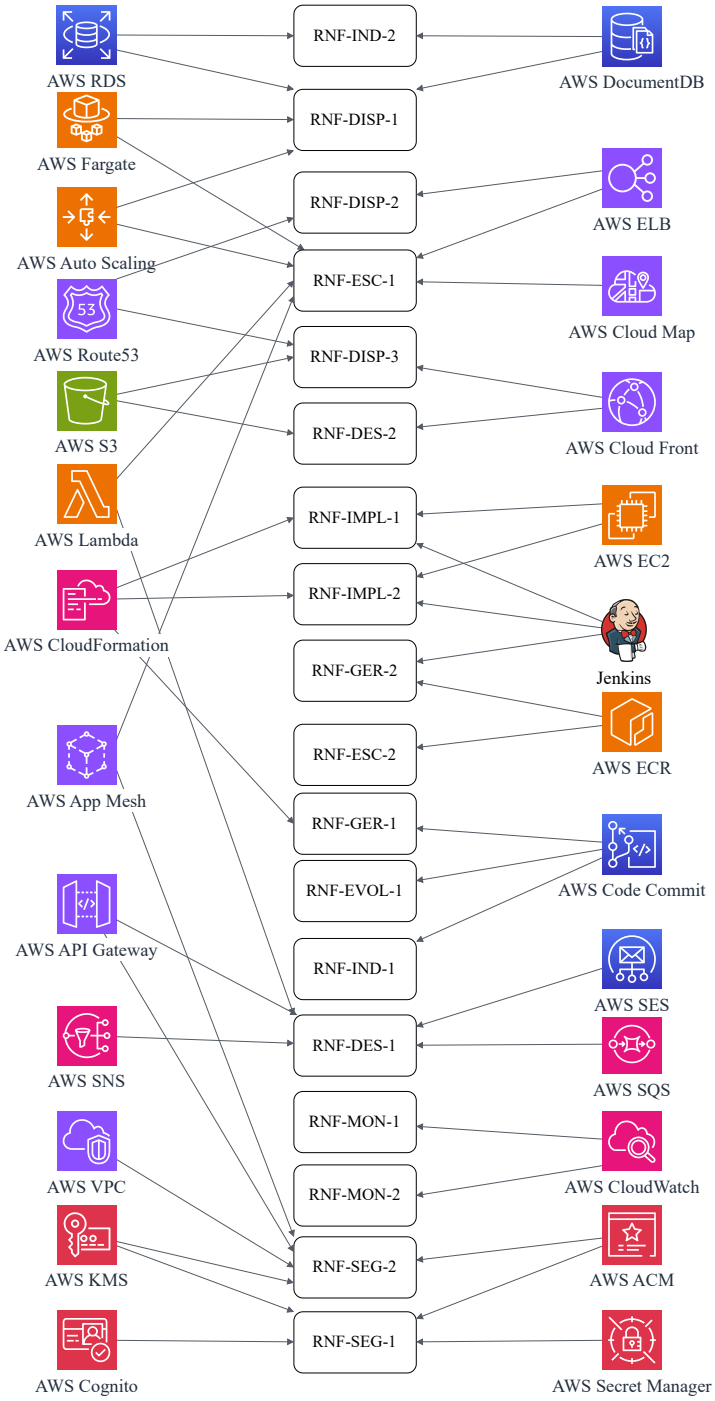
Por outro lado, os requisitos RNF-ESC-2, RNF-EVOL-1, RNF-IND-1 e RNF-MON-2 são cobertos por apenas uma tecnologia cada, o que sugere que esses aspectos específicos são atendidos de forma mais direcionada e concentrada. Isso pode indicar uma abordagem otimizada para esses requisitos, onde tecnologias específicas foram cuidadosamente selecionadas para lidar com necessidades pontuais, sem comprometer a complexidade do sistema.

Também é possível observar que as tecnologias AWS CodeCommit, AWS CloudFormation e Jenkins possuem a maior cobertura de requisitos não-funcionais, totalizando três cada. Por outro lado, as tecnologias AWS Cognito, AWS Secrets Manager, AWS Cloud Map, AWS SES, AWS SQS e AWS SNS atendem apenas a um requisito não-funcional.

Isso indica que algumas tecnologias desempenham um papel mais abrangente na arquitetura do sistema, contribuindo para múltiplos aspectos, como gerenciamento de infraestrutura, automação de implantação e controle de versionamento. Em contraste, outras tecnologias têm um escopo mais específico e direcionado para funções pontuais, como autenticação de usuários (AWS Cognito), armazenamento seguro de credenciais (AWS Secrets Manager), descoberta de serviços (AWS Cloud Map), envio de notificações por e-mail (AWS SES), gerenciamento de eventos (AWS SNS) e processamento assíncrono (AWS SQS).

Essa diferenciação reflete a natureza modular da arquitetura baseada em microsserviços, na qual algumas tecnologias são essenciais para o funcionamento geral do sistema, enquanto outras são empregadas para atender a necessidades mais especializadas.

Figura 4.8 – Grafo de relacionamento entre RNFs e Tecnologias



Fonte: Autor

5 CONCLUSÃO

A implementação de uma arquitetura de microsserviços na nuvem AWS foi motivada pela necessidade de criar uma infraestrutura escalável, segura e de alto desempenho para um sistema de comércio eletrônico. O objetivo principal foi construir uma solução moderna, robusta e flexível, que atendesse aos requisitos não funcionais críticos e proporcionasse uma base sólida para futuras expansões.

O `CompraVirtual` é uma aplicação de comércio eletrônico baseada em microsserviços, com foco na gestão de estoque e operações de venda. Ele possui serviços distintos para autenticação, gerenciamento de produtos, pedidos e notificações, o que permite um *design* modular e escalável.

Foram definidos requisitos não funcionais essenciais que garantem sua robustez e qualidade. A *disponibilidade* é um aspecto crítico, assegurando que o sistema esteja sempre acessível aos usuários, mesmo em situações de alta demanda. *Desempenho* é outro requisito essencial, com foco em assegurar tempos de resposta rápidos e uma experiência ágil para os usuários. A *escalabilidade* permite que o sistema se ajuste dinamicamente a variações de demanda, permitindo um grande número de usuários simultâneos sem comprometer o serviço. *Segurança* também é um requisito fundamental, protegendo dados sensíveis e assegurando que a comunicação seja segura e confiável. Além disso, a *flexibilidade* e a *evolucionabilidade* garantem que o sistema possa ser adaptado e atualizado continuamente, acompanhando as necessidades e mudanças no mercado. Esses requisitos juntos formam a base para um sistema robusto, eficiente e preparado para crescimento e mudanças futuras.

O sistema foi implementado utilizando práticas de Infraestrutura como Código (IaC) que promovem a automação e a flexibilidade por meio do AWS CloudFormation. Isso facilita o gerenciamento de mudanças e adaptações ao longo do tempo, além de permitir um *design* evolutivo que responde às necessidades de negócio e às inovações tecnológicas. O projeto e infraestrutura do sistema, incluindo a implementação dos microsserviços, pode ser encontrada em:

https://github.com/rafaelportomoura/ufla-tcc

A arquitetura foi projetada para atender aos requisitos da Seção 3.2. Para atender ao requisito de *disponibilidade*, foram utilizados o AWS S3 em conjunto com o AWS CloudFront que garantem a distribuição rápida e contínua de conteúdo, assegurando que o sistema esteja sempre acessível, mesmo sob alta demanda. Para atender ao requisito de *desempenho*, foram utilizados o AWS Lambda, AWS SNS e AWS SQS que, por meio do processamento assíncrono e comunicação desacoplada entre serviços, reduz o tempo de resposta do sistema. Para atender o requisito de *segurança*, foram utilizados o AWS KMS para criptografia de dados e AWS Cognito para a gestão de identidade, protegendo informações sensíveis e controlando o acesso. Para atender o requisito de *escalabilidade*, foram utilizados o AWS Fargate e AWS Auto Scaling, que ajustam automaticamente a capacidade do sistema conforme a demanda. Para atender o requisito de *monitoramento*, foi utilizado o AWS CloudWatch que coleta e analisa métricas em tempo real, proporcionando

visibilidade sobre a saúde e o desempenho do sistema, facilitando a detecção e resolução de problemas. E, para atender aos requisitos de *flexibilidade* e *evolucionabilidade*, foram utilizados o CloudFormation que provê a infraestrutura como código e o AWS CodeCommit que funciona como um servidor *Git* na AWS.

Este trabalho contribui para o campo da engenharia de software ao demonstrar como a combinação estruturada de serviços AWS pode atender a requisitos rigorosos dentro de uma arquitetura de microsserviços. A análise evidenciou que requisitos mais exigentes, como disponibilidade e segurança, demandam múltiplas tecnologias interconectadas, enquanto outros requisitos como monitoramento e evolucionabilidade podem ser satisfeitos por soluções isoladas. Além disso, verificou-se que nenhuma tecnologia utilizada foi capaz de atender mais de três requisitos não-funcionais simultaneamente, reforçando a necessidade de um planejamento arquitetural detalhado. A abordagem proposta pode servir como referência para futuras implementações e aprimoramentos em sistemas distribuídos baseados em microsserviços.

Para os trabalhos futuros, algumas direções de pesquisa podem ser exploradas. Primeiramente, uma comparação entre diferentes provedores de nuvem, em relação à AWS, seria relevante para avaliar vantagens e desvantagens de cada provedor nos requisitos levantados na Seção 3.2. Além disso, é importante considerar a viabilidade de implementar o sistema em uma infraestrutura física própria. Esse estudo permitiria uma análise comparativa com a nuvem, especialmente em termos de custos de manutenção, requisitos de hardware e complexidade no gerenciamento.

Outro aspecto a ser investigado é a otimização de custos em nuvem. Nesse contexto, explorar práticas de *FinOps* pode ajudar a identificar estratégias para reduzir os custos operacionais, utilizando ferramentas de monitoramento e automação de escalabilidade para maximizar a eficiência financeira. Por fim, avaliar a possibilidade de expandir o sistema para múltiplas regiões de nuvem poderia proporcionar uma análise sobre o impacto na latência, disponibilidade e custos associados ao tráfego inter-regional, promovendo uma infraestrutura ainda mais resiliente e ágil.

REFERÊNCIAS

ALMUBADDEL, M.; ELMOGY, A. M. Cloud computing antecedents, challenges, and directions. In: **16th Internet of things and Cloud Computing (ICC)**. [S.l.: s.n.], 2016. p. 1–5.

AWS. **Amazon DocumentDB**. 2024. Disponível em: <https://aws.amazon.com/documentdb/>. Acesso em: 05 jul. 2024.

AWS. **Amazon RDS**. 2024. Disponível em: <https://aws.amazon.com/rds/>. Acesso em: 05 jul. 2024.

AWS. **Amazon S3**. 2024. Disponível em: <https://aws.amazon.com/s3/>. Acesso em: 05 jul. 2024.

AWS. **AWS Api Gateway**. 2024. Disponível em: <https://aws.amazon.com/api-gateway>. Acesso em: 05 jul. 2024.

AWS. **AWS App Mesh**. 2024. Disponível em: <https://aws.amazon.com/app-mesh>. Acesso em: 05 jul. 2024.

AWS. **AWS Auto Scaling**. 2024. Disponível em: <https://aws.amazon.com/autoscaling>. Acesso em: 05 jul. 2024.

AWS. **AWS CertificateManager**. 2024. Disponível em: <https://aws.amazon.com/certificate-manager>. Acesso em: 05 jul. 2024.

AWS. **AWS Cloud Map**. 2024. Disponível em: <https://aws.amazon.com/cloud-map>. Acesso em: 05 jul. 2024.

AWS. **AWS Cloudformation**. 2024. Disponível em: <https://aws.amazon.com/cloudformation>. Acesso em: 05 jul. 2024.

AWS. **AWS Cloudfront**. 2024. Disponível em: <https://aws.amazon.com/cloudfront>. Acesso em: 05 jul. 2024.

AWS. **AWS Cloudwatch**. 2024. Disponível em: <https://aws.amazon.com/cloudwatch>. Acesso em: 05 jul. 2024.

AWS. **AWS CodeCommit**. 2024. Disponível em: <https://aws.amazon.com/codecommit>. Acesso em: 05 jul. 2024.

AWS. **AWS Cognito**. 2024. Disponível em: <https://aws.amazon.com/cognito>. Acesso em: 05 jul. 2024.

AWS. **AWS EC2**. 2024. Disponível em: <https://aws.amazon.com/ec2>. Acesso em: 05 jul. 2024.

AWS. **AWS Elastic Container Registry**. 2024. Disponível em: <https://aws.amazon.com/ecr>. Acesso em: 05 jul. 2024.

AWS. **AWS Elastic Load Balancing**. 2024. Disponível em: <https://aws.amazon.com/elasticloadbalancing>. Acesso em: 05 jul. 2024.

AWS. **AWS Fargate**. 2024. Disponível em: <https://aws.amazon.com/fargate>. Acesso em: 05 jul. 2024.

AWS. **AWS Key Management Service**. 2024. Disponível em: <https://aws.amazon.com/kms>. Acesso em: 05 jul. 2024.

AWS. **AWS Lambda**. 2024. Disponível em: <https://aws.amazon.com/lambda>. Acesso em: 05 jul. 2024.

AWS. **AWS Route53**. 2024. Disponível em: <https://aws.amazon.com/route53>. Acesso em: 05 jul. 2024.

AWS. **AWS Secrets Manager**. 2024. Disponível em: <https://aws.amazon.com/secrets-manager>. Acesso em: 05 jul. 2024.

AWS. **AWS Simple Email Service**. 2024. Disponível em: <https://aws.amazon.com/ses>. Acesso em: 05 jul. 2024.

AWS. **AWS Simple Notification Service**. 2024. Disponível em: <https://aws.amazon.com/sns>. Acesso em: 05 jul. 2024.

AWS. **AWS Simple Queue Service**. 2024. Disponível em: <https://aws.amazon.com/sqs>. Acesso em: 05 jul. 2024.

AWS. **AWS Virtual Private Cloud**. 2024. Disponível em: <https://aws.amazon.com/vpc>. Acesso em: 05 jul. 2024.

AWS. **O que é AWS?** 2024. Disponível em: <https://aws.amazon.com/pt/what-is-aws>. Acesso em: 05 jul. 2024.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables DevOps: Migration to a cloud-native architecture. **IEEE Software**, v. 33, n. 3, p. 42–52, 2016.

BIRRELL, A. D.; NELSON, B. J. Implementing remote procedure calls. **ACM Transactions on Computer Systems**, v. 2, n. 1, p. 39–59, 1984.

CLOUDFARE. **O que é HTTPS?** 2024. Disponível em: <https://www.cloudflare.com/pt-br/learning/ssl/what-is-https/>. Acesso em: 15 jul. 2024.

CONWAY, M. E. How do committees invent? **Datamation**, v. 14, n. 4, p. 28–31, 1968.

EBERT, C. et al. DevOps. **IEEE Software**, v. 33, n. 3, p. 94–100, 2016.

ERL, T.; PUTTINI, R.; MAHMOOD, Z. **Cloud Computing: Concepts, Technology & Architecture**. Upper Saddle River: Pearson Education, 2013.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, 2000.

FOWLER, M. **Richardson maturity model, steps toward the glory of REST**. 2010. Disponível em: <https://www.martinfowler.com/articles/richardsonMaturityModel.html>. Acesso em: 12 nov. 2023.

FOWLER, M.; LEWIS, J. **Microservices, a definition of this new architectural term**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 12 nov. 2023.

FOWLER, S. J. **Microsserviços prontos para a produção: Construindo sistemas padronizados em uma organização de engenharia de software**. São Paulo: Novatec, 2017.

HASSELBRING, W.; STEINACKER, G. Microservice architectures for scalability, agility and reliability in e-commerce. In: **1st International Conference on Software Architecture Workshops (ICSAW)**. [S.l.: s.n.], 2017. p. 243–246.

HOSTINGER. **Aprenda o que é CDN**. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-cdn>. Acesso em: 01 ago. 2024.

KILLALEA, T. A second conversation with Werner Vogels: The amazon CTO sits with Tom Killalea to discuss designing for evolution at scale. **Queue**, v. 18, n. 5, p. 67–92, 2020.

MASLOW, A. H. **The Psychology of Science**. Nova York: Harper & Row, 1966.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. National Institute of Standards and Technology, 2011. Disponível em: <https://www.nist.gov/publications/nist-definition-cloud-computing>. Acesso em: 12 nov. 2023.

MIYACHI, C. What is “cloud”? it is time to update the NIST definition? **IEEE Cloud computing**, v. 5, n. 3, p. 6–11, 2018.

MONTEIRO, E. C. **Metodologia multicritério de apoio à seleção de plataformas de cloud computing**. Dissertação (Mestrado) — Universidade do Porto, 2022.

NETO, O. C. de V. **Análise de disponibilidade em sistemas de software na Web.** Dissertação (Mestrado) — Escola Politécnica, Universidade de São Paulo, 2009.

NEWMAN, S. **Criando Microsserviços: Projetando Sistemas com Componentes Menores e Mais Especializados**. São Paulo: Novatec, 2022.

O’HANLON, C. A conversation with Werner Vogels: Learning from the Amazon technology platform: Many think of Amazon as ’that hugely successful online bookstore.’ you would expect Amazon CTO Werner Vogels to embrace this distinction, but in fact it causes him some concern. **Queue**, v. 4, n. 4, p. 14–22, 2006.

PATHAK, G.; SINGH, M. A review of cloud microservices architecture for modern applications. In: **14th World Conference on Communication & Computing (WCONF)**. [S.l.: s.n.], 2023. p. 1–7.

PONCE, F.; MÁRQUEZ, G.; ASTUDILLO, H. Migrating from monolithic architecture to microservices: A rapid review. In: **38th International Conference of the Chilean Computer Science Society (SCCC)**. [S.l.: s.n.], 2019. p. 1–7.

SADALAGE, P. J.; FOWLER, M. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. Boston: Addison-Wesley, 2012.

SUGIANTORO, B.; ANSHARI, M.; SUDRAJAT, D. Developing framework for web based e-commerce: Secure-SDLC. **Journal of Physics: Conference Series**, v. 1566, n. 1, p. 012020, 2020.

VERONEZ, M. **Um Catálogo de Requisitos Não-Funcionais e Padrões Arquiteturais para Microsserviços**. Dissertação (Mestrado) — Universidade Estadual do Oeste do Paraná, 2022.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, v. 1, n. 1, p. 7–18, 2010.

ZHOU, X. et al. Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. **Journal of Systems and Software**, v. 195, p. 111521, 2023.

ZIMMERMANN, O. Microservices tenets: Agile approach to service development and deployment. **Computer Science-Research and Development**, v. 32, n. 3, p. 301–310, 2017.