INTERFACE TECNOLÓGICA

A EVOLUÇÃO DA ARQUITETURA DE SOFTWARE: do Monólito aos Microsserviços

THE EVOLUTION OF SOFTWARE ARCHITECTURE: from Monolith to Microservices

Gabriele Carolina Rodrigues Ciriano – gabriele.ciriano@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

Giuliano Scombatti Pinto - giuliano.pinto@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

DOI: 10.31510/infa.v22i2.2269
Data de submissão: 21/09/2025
Data do aceite: 03/12/2025
Data da publicação: 20/12/2025

RESUMO

A arquitetura de software evoluiu significativamente nas últimas décadas, impulsionada pelas crescentes demandas por escalabilidade, flexibilidade e agilidade no desenvolvimento de sistemas. Este artigo explora a transição da arquitetura monolítica tradicional, caracterizada por um único bloco coeso de código, para a arquitetura de microsserviços, que fragmenta aplicações em serviços independentes e especializados. Inicialmente, os sistemas eram desenvolvidos como monólitos, favorecendo a simplicidade de implementação e *deploy*. No entanto, à medida que os sistemas cresciam, surgiam limitações relacionadas à manutenção, escalabilidade e tempo de resposta às mudanças. O artigo também discute os principais desafios dessa abordagem, como complexidade de gerenciamento, segurança distribuída e necessidade de observabilidade avançada. O objetivo deste trabalho foi analisar a evolução das arquiteturas de software, destacando a transição do modelo monolítico para a arquitetura de microsserviços, seus impactos no desenvolvimento de sistemas, e os principais benefícios e desafios associados a essa transformação. Este estudo caracteriza-se como uma pesquisa qualitativa de natureza exploratória, com abordagem bibliográfica e descritiva. Conclui-se que a escolha da arquitetura mais adequada deve considerar fatores técnicos e organizacionais, destacando-se que, embora os microsserviços tragam inúmeros benefícios, sua adoção requer maturidade e planejamento estruturado.

Palavras-chave: Arquitetura de Software, Monólito, Microsserviços, SOA, Escalabilidade, Sistemas Distribuídos.

INTERFACE TECNOLÓGICA

ABSTRACT

Software architecture has evolved significantly in recent decades, driven by the increasing demands for scalability, flexibility, and agility in systems development. This paper explores the transition from the traditional monolithic architecture, characterized by a single cohesive block of code, to the microservices architecture, which fragments applications into independent and specialized services. Initially, systems were developed as monoliths, favoring simplicity of implementation and deployment. However, as systems grew, limitations emerged related to maintainability, scalability, and response time to changes. The paper also discusses the main challenges of this approach, such as management complexity, distributed security, and the need for advanced observability. The objective of this work was to analyze the evolution of software architectures, highlighting the transition from the monolithic model to the microservices architecture, its impacts on systems development, and the main benefits and challenges associated with this transformation. This study is characterized as a qualitative research of an exploratory nature, with a bibliographic and descriptive approach. It is concluded that the choice of the most appropriate architecture must consider technical and organizational factors, highlighting that, although microservices bring numerous benefits, their adoption requires maturity and structured planning.
Keywords: Software Architecture, Monolith, Microservices, SOA, Scalability, Distributed Systems.

1 INTRODUÇÃO

Com o avanço das tecnologias da informação e a crescente demanda por sistemas mais escaláveis, ágeis e resilientes, a arquitetura de software passou por transformações significativas ao longo das últimas décadas (Katal et al., 2025).

Tradicionalmente, as aplicações eram desenvolvidas utilizando uma estrutura monolítica, através da integração de todos os componentes do sistema em uma única base de código e processo de execução. Embora eficaz em propostas de menor complexidade, tal tecnologia mostrou-se limitada perante a necessidade de crescimento, manutenção contínua e inovação rápida (Kaloudis, 2024).

Nesse contexto, arquiteturas mais flexíveis começaram a emergir, sendo a Arquitetura Orientada a Serviços (SOA) um marco importante na transição para modelos mais desacoplados. No entanto, a consolidação da arquitetura de microsserviços possibilitou alcançar um patamar superior de modularização, permitindo que sistemas fossem compostos por serviços independentes, pequenos e especializados, que se funcionam de forma coordenada, porém isolada (Mooghala, 2023).

O surgimento de metodologias ágeis, a popularização da computação em nuvem, o amadurecimento de práticas DevOps e o avanço de ferramentas de automação,

observabilidade e segurança em ambientes distribuídos estimularam esta tecnologia, porém ainda com desafios, pressupondo alterações não só técnicas, mas igualmente culturais e organizacionais (Katal et al., 2025).

A constante evolução dos ambientes de desenvolvimento de software, motivada pela demanda por sistemas mais ágeis, escaláveis e resilientes, consolidou a arquitetura como tópico central no campo da tecnologia da informação. Desta forma, compreender a transição da arquitetura monolítica para a arquitetura de microsserviços é essencial para organizações que buscam inovação, competitividade e sustentabilidade técnica a longo prazo (Eli; Olsen, 2021).

Este artigo tem como objetivo analisar essa trajetória evolutiva, discutindo os principais aspectos que motivaram a transição do monólito para os microsserviços, bem como os benefícios, limitações e implicações dessa mudança para o desenvolvimento de sistemas modernos.

A metodologia implementada foi de uma pesquisa qualitativa de natureza exploratória, com abordagem bibliográfica e descritiva. Os resultados foram organizados de forma a apresentar uma linha evolutiva das arquiteturas de software, ressaltando aspectos técnicos, organizacionais e operacionais.

2 ARQUITETURA DE SOFTWARE

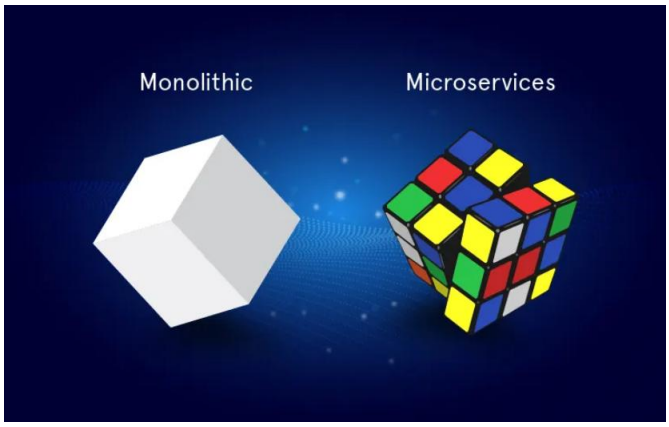
A arquitetura de software é entendida como a estrutura fundamental de um sistema de software, composta pelos seus componentes principais, suas inter-relações e pelos princípios e diretrizes que orientam seu design e evolução (Trowbridge et al., 2016). Segundo Bass et al. (2015), a arquitetura é crucial para a definição das qualidades do sistema, incluindo atributos como desempenho, escalabilidade, segurança e manutenibilidade.

Os componentes arquiteturais representam unidades funcionais autônomas que encapsulam lógica e dados, enquanto as conexões entre eles estabelecem os mecanismos de comunicação e coordenação. De acordo com Clements et al. (2020), estilos arquiteturais como monolítico, orientado a serviços (SOA), baseado em eventos e microsserviços, fornecem padrões reutilizáveis para organizar esses componentes e conexões de maneira eficiente e sustentável.

O estilo arquitetural monolítico, caracterizado pela integração de todas as funcionalidades em um único sistema, apresenta simplicidade inicial, mas tende a se tornar rígido e difícil de escalar conforme a aplicação cresce (Trowbridge et al., 2016).

Por outro lado, a arquitetura de microsserviços propõe a decomposição do sistema em pequenos serviços independentes, cada um responsável por uma funcionalidade específica, comunicando-se através de APIs bem definidas. Essa abordagem facilita a escalabilidade, o desenvolvimento paralelo e a resiliência do sistema, embora introduza complexidade adicional na gestão e operação. A figura 1 exemplifica a comparação entre os sistemas monolítico e de microserviços (Newman, 2019).

Figura 1 - Imagem comparativa entre os sistemas monolítico e de microsserviços



Fonte: Baddula, 2023.

A escolha da arquitetura adequada é um fator determinante para o sucesso do projeto, influenciando diretamente a capacidade do sistema em atender requisitos funcionais e não funcionais (Krutch, 2017). Além disso, uma boa arquitetura deve prever a evolução do sistema, permitindo adaptações às mudanças tecnológicas e de mercado, sem comprometer sua integridade (Bass et al., 2015).

2.1. Arquitetura Monolítica: Características e Limitações

A arquitetura monolítica consiste em um único sistema indivisível onde todos os componentes e funcionalidades são desenvolvidos, testados e implantados como uma única aplicação (Chu, 2020). Essa abordagem consolidada reúne as camadas de interface, lógica de negócio e acesso a dados em um único artefato executável, o que facilita a construção inicial e o gerenciamento simplificado do sistema (Richardson, 2018).

Segundo Newman (2019), a principal vantagem do monólito está na sua simplicidade estrutural, que permite ciclos de desenvolvimento mais rápidos em projetos de pequeno ou

médio porte, além de facilitar a comunicação entre módulos devido à proximidade no código-fonte. Em contrapartida, essa centralização provoca um forte acoplamento entre os componentes, o que torna a manutenção e a evolução do sistema progressivamente mais complexas à medida que o código cresce (Chu, 2020).

A escalabilidade da arquitetura monolítica é outro ponto crítico. Conforme destacado por Bass et al. (2015), o modelo obriga a replicação do sistema inteiro, mesmo quando apenas algumas funcionalidades exigem maior capacidade, resultando em uso ineficiente de recursos. Além disso, a implantação contínua enfrenta desafios significativos, pois uma alteração mínima requer a recompilação e o *redeploy* do sistema completo, elevando os riscos de indisponibilidade e erros em produção (Richardson, 2018).

Dessa forma, apesar de sua simplicidade inicial e adequação para sistemas menores, a arquitetura monolítica apresenta sérias limitações em ambientes que demandam alta escalabilidade, flexibilidade e agilidade, fatores que motivaram a busca por alternativas mais modulares, como a arquitetura de microsserviços (Newan, 2019; Richardson, 2018).

2.2 Arquitetura de Microsserviços

A arquitetura de microsserviços é um estilo arquitetural emergente que propõe a construção de sistemas a partir da composição de serviços pequenos, autônomos e independentes, cada um responsável por uma funcionalidade específica do domínio da aplicação. Essa abordagem tem se destacado nos últimos anos como uma alternativa mais flexível e escalável frente às arquiteturas monolíticas e orientadas a serviços tradicionais (Newman, 2019).

Diferentemente do modelo monolítico, em que todas as funcionalidades do sistema estão acopladas em uma única aplicação, os microsserviços são desacoplados e implantáveis independentemente, o que proporciona maior liberdade de escolha tecnológica, escalabilidade horizontal sob demanda e melhor isolamento de falhas (Richards; Ford, 2020).

Segundo Newman (2015), um microsserviço deve ser coeso, pequeno o suficiente para ser completamente compreendido por uma única equipe de desenvolvimento, e suficientemente autônomo para que possa ser desenvolvido, testado e implantado sem interferência nos demais serviços do sistema. Essa independência promove entregas contínuas, integração com práticas de DevOps e maior agilidade na resposta a mudanças nos requisitos do negócio.

Os microsserviços geralmente se comunicam por meio de protocolos leves, como REST/HTTP, gRPC ou mensageria assíncrona (por exemplo, RabbitMQ ou Apache Kafka), favorecendo a flexibilidade e a resiliência. Além disso, os dados são frequentemente privados por serviço, ou seja, cada serviço gerencia seu próprio banco de dados, o que reforça o encapsulamento e evita dependências indesejadas entre componentes (Guillermo, 2018).

Esse estilo arquitetural também se baseia em princípios como descentralização, resiliência, observabilidade e automação de operações. Em um ambiente com dezenas ou centenas de microsserviços, é essencial dispor de ferramentas para monitoramento distribuído, rastreamento de chamadas, balanceamento de carga e tolerância a falhas (Heredia et al., 2020).

Embora ofereça inúmeras vantagens, a arquitetura de microsserviços também introduz desafios adicionais, como a complexidade de orquestração, a gestão de consistência distribuída e a necessidade de uma cultura organizacional madura para lidar com a fragmentação da aplicação. Assim, sua adoção deve ser cuidadosamente planejada, levando em consideração os requisitos técnicos e estratégicos do sistema em desenvolvimento (Eli; Olsen, 2021).

A adoção da arquitetura de microsserviços exige um ecossistema tecnológico robusto e especializado, capaz de lidar com os desafios de distribuição, automação, escalabilidade, segurança e comunicação entre serviços. Ao contrário de arquiteturas monolíticas, onde grande parte da lógica e da infraestrutura reside em uma única aplicação, os microsserviços distribuem essas responsabilidades entre diversos componentes independentes. Nesse cenário, a utilização de ferramentas modernas e plataformas específicas torna-se essencial para garantir o bom funcionamento da aplicação como um todo (Newman, 2019).

Entre as tecnologias mais utilizadas em ambientes de microsserviços, destacam-se:

- Conteinerização e Orquestração, que por meio de ferramentas como o Docker, permite empacotar microsserviços com todas as suas dependências em unidades leves e portáveis. Cada contêiner executa um serviço de forma isolada, garantindo consistência entre os ambientes de desenvolvimento, teste e produção (Burns et al., 2016).
- Comunicação entre Serviços: os microsserviços podem se comunicar de forma síncrona, geralmente usando RESTful APIs ou gRPC, ou de forma assíncrona, utilizando sistemas de mensageria como RabbitMQ, Apache Kafka ou Amazon SQS. A comunicação

assíncrona é recomendada em cenários que exigem alta disponibilidade, resiliência e baixo acoplamento (Omar; Khalid, 2021).

- *Service Discovery* e API *Gateway*: em ambientes distribuídos, o descobrimento de serviços (*Service Discovery*) é essencial para localizar dinamicamente os serviços disponíveis. Tecnologias como Eureka (Netflix OSS), Consul (HashiCorp) ou Zookeeper são frequentemente usadas. O API *Gateway* atua como um ponto central de entrada para as chamadas externas ao sistema, agregando funcionalidades como roteamento de requisições, limitação de taxa (*rate limiting*), autenticação, e registro de logs (Barletta et al., 2019).
- Observabilidade e Monitoramento: entre as tecnologias mais utilizadas estão Prometheus e Grafana: monitoramento e visualização de métricas; ELK Stack (Elasticsearch, Logstash, Kibana): centralização e análise de logs; Jaeger ou Zipkin: rastreamento distribuído (*tracing*) de requisições entre serviços (Faseeha et al., 2025).
- Segurança e Autenticação: centralizada via OAuth 2.0, OpenID Connect ou sistemas de identidade como Keycloak. Para proteger a comunicação entre serviços, é comum utilizar TLS, tokens JWT e políticas de segurança implementadas via *service mesh*, como Istio (Dodanduwa; Kaluthanthri, 2018).
- Integração Contínua e Entrega Contínua (CI/CD): ferramentas como Jenkins, GitLab CI, Argo CD, Tekton e GitHub Actions são amplamente utilizadas para automatizar a construção, teste, versionamento e implantação contínua de microsserviços, permitindo ciclos de entrega mais ágeis e confiáveis (Rostami et al., 2023).

3 PROCEDIMENTOS METODOLÓGICOS

A presente pesquisa caracteriza-se como um estudo qualitativo, de natureza exploratória, conduzido por meio de uma revisão de literatura narrativa sobre a evolução das arquiteturas de software, com ênfase na transição de modelos monolíticos para arquiteturas baseadas em microsserviços.

Para isso, foi realizada uma busca sistemática de fontes bibliográficas em bases acadêmicas reconhecidas, como IEEE Xplore, ACM Digital Library, Scopus, ScienceDirect e Google Scholar, bem como em livros técnicos e publicações atualizadas da área de Engenharia de Software.

A pesquisa conduzida por meio de um levantamento bibliográfico em bases de dados científicas reconhecidas possibilitou a identificação de publicações pertinentes ao tema. A

aplicação dos filtros resultou em 142 artigos inicialmente identificados nas diversas bases consultadas. Após a remoção de duplicatas, permaneceram 103 estudos únicos para triagem. Após a aplicação dos critérios de inclusão e exclusão, 25 artigos foram considerados adequados para compor o trabalho, representando o corpo final de evidências utilizado para fundamentar a análise apresentada.

Os critérios de inclusão consideraram publicações entre os anos de 2010 e 2025, com relevância para os conceitos fundamentais da arquitetura de software, abordagens arquiteturais emergentes e os impactos dessas transformações sobre escalabilidade, manutenibilidade, testes e ciclo de vida do software.

Foram utilizados como descritores os termos: "*software architecture evolution*", "*monolithic architecture*", "*microservices*", "*service-oriented architecture*" e "*cloud-native software*".

Após a coleta, os materiais foram organizados, classificados e analisados de forma crítica, com o objetivo de sintetizar os principais conceitos, identificar tendências, destacar vantagens e desvantagens dos modelos comparados e apontar lacunas ou desafios presentes na literatura. Os resultados da análise são apresentados em tópicos temáticos ao longo do trabalho, apoiando-se em uma estrutura cronológica e conceitual da evolução arquitetural.

4 RESULTADOS E DISCUSSÃO

A comparação entre arquitetura monolítica e arquitetura de microsserviços envolve a análise de uma série de vantagens e desvantagens que impactam diretamente o desenvolvimento, a manutenção, a escalabilidade e a operação de sistemas de software (Al-Debagy; Martinek, 2019).

A arquitetura monolítica apresenta diversas vantagens, especialmente nas fases iniciais de um projeto. Sua principal força está na simplicidade inicial, sendo mais fácil de desenvolver, testar e implantar (Blinowski et al., 2022).

Como toda a aplicação está concentrada em um único bloco, há menor sobrecarga operacional, o que facilita o monitoramento e a gestão em ambientes menores. A integração entre os componentes ocorre de forma nativa, já que eles compartilham o mesmo espaço de memória e banco de dados, reduzindo a complexidade da comunicação interna. Além disso, a curva de aprendizado é mais suave, tornando essa abordagem mais acessível para equipes pequenas ou menos experientes em arquiteturas distribuídas (Chavan, 2021).

No entanto, a arquitetura monolítica também apresenta desvantagens importantes. Sua escalabilidade é limitada, já que não é possível escalar partes específicas da aplicação isoladamente. Com o crescimento do sistema, a manutenção torna-se complexa, dificultando a compreensão e a evolução do código. As implantações são mais arriscadas, pois qualquer alteração exige o *redeploy* completo da aplicação, elevando o risco de falhas. Além disso, há uma rigidez tecnológica, dificultando a adoção de diferentes tecnologias em partes distintas do sistema (Al-Debagy; Martinek, 2019).

Por outro lado, a arquitetura de microsserviços traz vantagens relevantes para aplicações em larga escala. Uma de suas principais qualidades é a escalabilidade granular, permitindo que cada serviço seja escalado de forma independente, otimizando recursos (Cruz; Saito, 2021).

A divisão em serviços pequenos e bem definidos também facilita a manutenção e evolução contínua do sistema, além de proporcionar maior resiliência, uma vez que falhas em um serviço não comprometem todo o sistema. Outro ponto forte é a flexibilidade tecnológica, que permite o uso de diferentes linguagens, bancos de dados e *frameworks* conforme a necessidade de cada serviço. Isso também favorece entregas contínuas mais seguras, já que é possível atualizar serviços individualmente, sem afetar o funcionamento global da aplicação (Rostami et al., 2023).

Em contrapartida, essa abordagem traz complexidade operacional elevada, exigindo uma infraestrutura robusta para lidar com orquestração, monitoramento, segurança, balanceamento de carga e logs distribuídos. O gerenciamento de dados torna-se mais desafiador, pois os dados estão distribuídos entre vários serviços, exigindo estratégias como consistência eventual ou padrões como Sagas. Além disso, a comunicação intensiva entre serviços, feita por HTTP/gRPC ou mensageria, pode gerar latência e falhas, e a curva de aprendizado para implementar e manter essa arquitetura é significativamente maior, exigindo conhecimentos técnicos avançados em áreas como DevOps, redes e sistemas distribuídos (Blinowski et al., 2022).

Assim, a escolha entre monólito e microsserviços não deve ser baseada em modismos, mas sim nas necessidades do projeto, maturidade da equipe, escala da aplicação e infraestrutura disponível. Enquanto o monólito pode ser mais vantajoso em fases iniciais ou projetos menores, os microsserviços se mostram superiores em cenários complexos, com alto volume de usuários e demanda por evolução contínua (Cruz; Saito, 2021).

A Tabela 1 a seguir apresenta uma síntese comparativa entre monólito e microsserviços.

Tabela 1 - Resumo comparativo entre monólito e microsserviços.

Critério	Monólito	Microsserviços
Complexidade inicial	Baixa	Alta
Escalabilidade	Limitada	Alta e independente
Facilidade de implantação	Simples	Complexa (orquestração, CI/CD)
Tolerância a falhas	Baixa (falha afeta tudo)	Alta (falhas isoladas)
Manutenção	Difícil em sistemas grandes	Mais fácil por serviço modular
Flexibilidade tecnológica	Limitada	Alta
Reuso de código	Centralizado	Compartilhamento por APIs
Gerenciamento de dados	Simples (banco único)	Complexo (dados distribuídos)

Fonte: Cruz e Saito (2021).

5 CONCLUSÃO

A avanço da arquitetura de software, a partir do modelo monolítico aos microsserviços, demonstra o aumento da demanda por sistemas mais adaptáveis, dinâmicos e resilientes. As arquiteturas monolíticas retratam vantagens em projetos de menor escala, em razão de sua simplicidade e menor demanda de infraestrutura. Já os microsserviços se destacam como a solução mais adequada para sistemas desafiadores, sendo possível estabelecer uma maior independência de serviços e flexibilidade no desenvolvimento. A transição entre esses modelos requer alterações técnicas culturais e organizacionais e eventualmente a aplicação de novas práticas de gestão, DevOps e automação. Desta forma, é essencial a compreensão de tais transformações para que profissionais e organizações possam desfrutar de inovação, eficiência e sustentabilidade técnica, permitindo decisões conscientes sobre a arquitetura mais apropriada ao contexto e às necessidades do software.

REFERÊNCIAS

AL-DEBAGY, O.; MARTINEK, P. A Comparative Review of Microservices and Monolithic Architectures. 2019. Disponível em: https://arxiv.org/abs/1905.07997 Acesso em: 27 jun. 2025.

BADDULA, P. **The Evolution of Software Architecture: Monolithic to Microservices**. Medium, 28 jul. 2023.

BARLETTA, V. S.; CAIVANO, D.; NANNAVECCHIA, A.; SCALERA, M. A Spell Checking Web Service API for Smart City Communication Platforms. **Open Journal of Applied Sciences.** V.9 n.1. 2019

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3rd ed. Addison-Wesley Professional, 2015.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYLEK, A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. **IEEE Access**, v. 10, p. 1–1, 2022.

BURNS, B.; GRANT, B.; OPPENHEIMER, D. Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. **Communications of the ACM**, v. 59, n. 5, p. 50–57, 2016.

CHAVAN, A. Exploring event-driven architecture in microservices – patterns, pitfalls and best practices. **International Journal of Science and Research Archive**, v. 4, n. 1, p. 229–249, 2021.

CHU, B. Monolithic vs Microservices Architectures. **Journal of Software Engineering**, v. 14, n. 2, p. 34-45, 2020.

CLEMENTS, P. **Documenting Software Architectures: Views and Beyond**. 2nd ed. Addison-Wesley, 2020.

CRUZ, R., SAITO, E. Aplicando Arquitetura de Microsserviços no Desenvolvimento de Software. **Revista de Ubiquidade**, 4(2), 58-84. 2021.

DODANDUWA, K.; KALUTHANTHRI, I. Role of Trust in OAuth 2.0 and OpenID Connect. **2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS),** Colombo, Sri Lanka, pp. 1-4, 2018.

ELI, H., OLSEN, D. Creating a sustainable digital infrastructure: The role of service-oriented architecture, **Procedia Computer Science**, v. 181, p.597-604. 2021.

FASEEHA, U.; JAMIL, S, H.; SAMAD, F. S.; AHMED, H. Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms. **IEEE Access**, vol. 13, pp. 72011-72039, 2025.

GUILLERMO, R., DÍAZ-PACE, A., SORIA, A. A case-based reasoning approach to reuse quality-driven designs in service-oriented architectures, Information Systems, v. 77, p.167-189. 2018.

HEREDIA, J. S., FLORES-GARCÍA, E., SOLANO, A. R. Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST and GRAPHQL. **Applied Technologies**, vol.1193, pp.286, 2020.

KALOUDIS, M. Evolving Software Architectures from Monolithic Systems to Resilient Microservices: Best Practices, Challenges and Future Trends. **International Journal of Advanced Computer Science and Applications**. v. 15, n. 9, 2024.

KATAL, A.; PRASANNA, P.; BIRLA, R.; KUNAL. Evolution from Monolithic to Microservices Architecture: A New Era in Software Architecture. In: ROSSIT, D.; TORRES-AGUILAR, C. E.; TONCOVICH, A. A. (eds). Advancements in Optimization and Nature-Inspired Computing for Solutions in Contemporary Engineering Challenges. Singapore: Springer, 2025. (Springer Tracts in Nature-Inspired Computing).

KRUTCH, J. The Importance of Software Architecture. **IEEE Software**, v. 34, n. 5, p. 12–15, 2017.

MOOGHALA, S. A Comprehensive Study of the Transition from Monolithic to Micro services-Based Software Architectures. **Journal of Technology and Systems**. v. 5, 27-40. 2023.

NEWMAN, S. **Building Microservices**. O'Reilly Media, 2019.

OMAR, H. A., KHALID, S. A. Performance Analysis the Hybrid MQTT/UMA and Restful IoT. **Security Model Advances in Internet of Things** Vol.11 No.1. 2021.

RICHARDS, M.; FORD, N. **Fundamentals of Software Architecture: An Engineering Approach**. Boston: O'Reilly Media, 2020.

RICHARDSON, C. **Microservices Patterns: With Examples in Java**. Manning Publications, 2018.

ROSTAMI, M. P., MENS, T., GOLZADEH, M., DECAN, A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. **Empirical Software Engineering**, 28(2), 52. 2023.

TROWBRIDGE, T.; MAIER, M.; STUMPF, M. **Software Architecture Foundations, Theory, and Practice**. Wiley, 2016.