Template de TCC-Pesquisa na Engenharia de Software: Formato da Sociedade Brasileira de Computação (SBC)

André Rodrigues de Freitas Faria[1]

[1]Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica de Minas Gerais (PUC MINAS) R. Dom José Gaspar, 500 - Coração Eucarístico, Belo Horizonte - MG, 30535-901

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escrtos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Danilo Boechat - daniloboechat@pucminas.br
Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: (A ser definido no próximo semestre)

Belo Horizonte, DIA de MÊS de ANO.

1. Introdução

A arquitetura de software é uma área fundamental da engenharia de software, responsável por definir e organizar a estrutura de um sistema, seus componentes e os relacionamentos entre eles, tanto em um nível macro quanto micro. Segundo a norma ISO/IEC/IEEE 42010:2011, arquitetura é "a organização fundamental de um sistema, incorporada em seus componentes, nos relacionamentos entre eles e com o ambiente, e nos princípios que governam seu design e evolução" [ISO/IEC/IEEE 2011] Martin Fowler complementa essa definição ao afirmar que "a arquitetura de software é o conjunto de decisões difíceis de mudar posteriormente" [Fowler 2003].

Um dos modelos mais tradicionais de desenvolvimento de software é a arquitetura monolítica, na qual todas as funcionalidades do sistema — como interface do usuário, lógica de negócios e acesso a dados — são integradas em uma única base de código e implantadas como uma unidade indivisível. Essa abordagem simplifica o desenvolvimento

e a implantação iniciais, mas pode levar a desafios significativos à medida que o sistema cresce em complexidade, dificultando a escalabilidade e a manutenção. Segundo a IBM, ”a arquitetura monolítica é um modelo tradicional de desenvolvimento de software em que uma única base de código executa várias funções de negócios”[IBM 2024]. Essa pesquisa tem como objetivo entender o real impacto de escalabilidade e desenvolvimento que um modelo de monolito distribuído possui.

Apesar da crescente adoção de arquiteturas baseadas em microsserviços, muitas empresas de grande porte ainda mantêm sistemas monolíticos em funcionamento. Organizações como GitHub, Twitter, Shopify, Basecamp, Atlassian, Airbnb, SoundCloud e Netflix continuam a operar partes significativas de suas plataformas utilizando arquiteturas monolíticas [Juhirdeen 2024], seja por razões de legado, estabilidade ou complexidade envolvida na migração para arquiteturas mais modernas. Essa realidade ressalta a importância de compreender os impactos e desafios associados aos monolitos distribuídos, especialmente para auxiliar desenvolvedores na transição para arquiteturas de microsserviços, evitando a repetição de erros comuns nesse processo .

O objetivo geral deste trabalho é **analisar o desempenho de uma arquitetura de monólito distribuído em relação a uma arquitetura de microsserviços**, com base em critérios de eficiência, escalabilidade e consumo de recursos computacionais. Como objetivos específicos, esta pesquisa propõe: (i) comparar o tempo de resposta entre aplicações construídas sob ambas as arquiteturas, utilizando testes de carga controlados; (ii) avaliar o consumo de recursos computacionais, como CPU e memória, em diferentes níveis de estresse; (iii) analisar a escalabilidade horizontal de cada arquitetura, identificando suas limitações e capacidade de adaptação à demanda crescente; e (iv) documentar a estrutura e os desafios técnicos enfrentados na implementação prática dos dois modelos arquiteturais em um ambiente padronizado.

A arquitetura de microsserviços surgiu como uma evolução natural dos monólitos, visando resolver problemas de escalabilidade e manutenção. No entanto, muitas organizações, ao tentarem adotar microsserviços sem o devido planejamento e compreensão, acabam criando o que é conhecido como ”monólitos distribuídos”. Nesses casos, o sistema mantém as desvantagens de um monólito tradicional, como alto acoplamento e dificuldade de manutenção, acrescidas da complexidade inerente a sistemas distribuídos. Essa abordagem pode resultar em maior consumo de recursos e tempos de resposta mais elevados, comprometendo a escalabilidade horizontal e a eficiência do sistema como um todo. Conforme destacado por Elemar Júnior, ”o monólito distribuído é a pior das alternativas: mantém as desvantagens do monólito tradicional e adiciona a complexidade de sistemas distribuídos”[Júnior 2023].

Durante a realização desta pesquisa, será desenvolvido o backend de uma aplicação de teste utilizando duas abordagens arquiteturais distintas: monólito distribuído e microsserviços. Ambas as versões do sistema serão projetadas para oferecer a mesma funcionalidade, permitindo que a comparação entre elas ocorra em um ambiente controlado e isento de variações funcionais. Além disso, serão cuidadosamente documentadas as estruturas internas de cada projeto, bem como a infraestrutura em nuvem utilizada para sua execução. Essa padronização é essencial para garantir uma análise precisa e objetiva dos aspectos de desempenho, consumo de recursos e escalabilidade de cada arquitetura.

2. Fundamentação Teórica

3. Trabalhos Relacionados

3.1. Sections and Paragraphs

Nas últimas décadas, arquiteturas de software vêm evoluindo com o intuito de atender às crescentes demandas por escalabilidade, manutenção e desempenho em aplicações web. Nesse contexto, diversos estudos comparativos entre monólitos e microsserviços foram conduzidos com o objetivo de identificar vantagens, limitações e cenários mais adequados para cada abordagem.

Khant Hmue et al. (2024) realizaram um estudo prático e comparativo entre arquiteturas monolíticas e baseadas em microsserviços aplicadas a um sistema de e-commerce. Ambas as versões foram desenvolvidas com a stack MERN (MongoDB, *Express, React e Node.js*) e submetidas a testes de carga utilizando o Apache JMeter. Os resultados demonstraram que a arquitetura de microsserviços apresentou menor latência (777 ms contra 1488 ms no monólito, sob 200 usuários concorrentes), menor taxa de erro (19,93% contra 40,33%) e maior throughput em cenários de alta demanda. Apesar das vantagens de desempenho, os autores reconhecem que a complexidade operacional dos microsserviços, especialmente no que diz respeito à segurança, orquestração e monitoramento, pode representar uma barreira para sua adoção em projetos menores ou com equipes reduzidas [**?**]

Em outro estudo, Hasan et al. (2023) focam especificamente na métrica de manutenibilidade de arquiteturas de software durante a migração de sistemas legados monolíticos para microsserviços. Os autores propõem um modelo baseado em métricas estruturais, como acoplamento, coesão e complexidade, para avaliar a qualidade arquitetural de sistemas que adotam microsserviços. A pesquisa é fundamentada em modelos de qualidade reconhecidos, como o ISO/IEC 25010, e propõe a utilização dessas métricas ainda na fase de projeto, como forma de reduzir a dívida técnica e evitar falhas estruturais durante ou após a migração. Embora não realizem testes empíricos, os autores sugerem que tais métricas podem ser integradas a ferramentas de análise estática para apoiar decisões arquiteturais fundamentadas [**?**]

Kamisetty et al. (2023) apresentam uma análise comparativa ampla entre monólitos e microsserviços com foco em escalabilidade e desempenho, com base em revisões sistemáticas da literatura e estudos de caso reais. O artigo destaca que, embora os monólitos ofereçam simplicidade e facilidade de desenvolvimento inicial, eles se tornam progressivamente mais difíceis de escalar e manter. Microsserviços, por outro lado, promovem isolamento de falhas, escalabilidade independente de componentes e ciclos de desenvolvimento mais ágeis. Contudo, a adoção de microsserviços impõe novos desafios relacionados à comunicação interserviços, consistência de dados e governança arquitetural. O estudo conclui que não há uma abordagem universalmente superior, e a escolha entre monólito e microsserviços deve considerar o contexto específico da aplicação e as capacidades da equipe de desenvolvimento [**?**]

Esses trabalhos evidenciam que, embora a arquitetura de microsserviços apresente benefícios claros em relação a desempenho e escalabilidade, especialmente em cenários de alta concorrência, há uma complexidade inerente que deve ser cuidadosamente avaliada. A análise comparativa proposta neste trabalho contribui com essa discussão ao

explorar uma alternativa intermediária, o monólito distribuído, que busca unir benefícios de modularidade e simplicidade.

4. Materiais e Métodos

Esta pesquisa é classificada como de natureza aplicada, com abordagem quantitativa e método experimental. O estudo visa analisar o impacto de diferentes arquiteturas de software — monólito distribuído e microsserviços — no desempenho de aplicações backend, a partir de medições empíricas obtidas em ambiente controlado. A escolha do método experimental se justifica pela necessidade de comparação objetiva entre as abordagens arquiteturais, com base em dados coletados em tempo real.

O ambiente técnico utilizado foi cuidadosamente definido para garantir a equivalência entre os testes. As aplicações foram desenvolvidas em Java 21 utilizando o framework Spring Boot 3.x, com o banco de dados PostgreSQL 15. A comunicação entre serviços, na versão de microsserviços, foi feita por meio de chamadas HTTP REST. A orquestração e implantação foram realizadas via Docker e Docker Compose, garantindo reprodutibilidade entre os ambientes. A infraestrutura cloud utilizada foi baseada em instâncias t3.medium da AWS EC2, rodando Ubuntu 22.04. Para testes de carga, utilizou-se o Apache JMeter, enquanto o Prometheus, em conjunto com Grafana, foi empregado para o monitoramento do uso de recursos (CPU, memória e throughput).

O método adotado consistiu na implementação de duas versões da mesma aplicação, com funcionalidade idêntica. A primeira versão foi construída com arquitetura de monólito distribuído, caracterizada por múltiplos módulos fortemente acoplados e executados como um único serviço. A segunda versão utilizou arquitetura de microsserviços, com componentes independentes, comunicação desacoplada e possibilidade de escalabilidade individual por serviço. Ambas as versões foram submetidas a testes de estresse e carga para avaliação comparativa.

As métricas utilizadas na avaliação foram: tempo de resposta médio (em milissegundos), taxa de erro (% de respostas 4xx/5xx), consumo de CPU e memória (coletados em tempo real), throughput (requisições por segundo) e capacidade de escalabilidade horizontal. Essas métricas foram escolhidas por serem amplamente utilizadas em benchmarks de desempenho de sistemas distribuídos, refletindo diretamente na eficiência e na robustez das arquiteturas analisadas.

Para a coleta e análise dos dados, foram utilizados os seguintes instrumentos: Apache JMeter, para simulação de carga e extração de dados de desempenho; Prometheus, para coleta de métricas de infraestrutura; e Grafana, para visualização dos resultados em tempo real. Além disso, foram desenvolvidos scripts automatizados para coleta, padronização e exportação dos dados para posterior análise estatística e construção de gráficos comparativos.

A seguir, apresenta-se o cronograma de execução do TCC II, distribuído em quinzenas, com as principais etapas da pesquisa. O cronograma foi planejado com o objetivo de garantir a divisão equilibrada das atividades e o cumprimento progressivo dos marcos estabelecidos:

1ª ago/2025: Definição final da arquitetura e configuração de ambientes
2ª ago/2025: Implementação da arquitetura de monólito distribuído

1ª set/2025: Implementação da arquitetura de microsserviços
2ª set/2025: Configuração de infraestrutura e execução dos primeiros testes
1ª out/2025: Execução de testes de carga e coleta de dados
2ª out/2025: Análise preliminar de desempenho e refinamento de testes
1ª nov/2025: Escrita dos capítulos de Resultados e Discussão
2ª nov/2025: Consolidação da documentação técnica e geração de gráficos
1ª dez/2025: Revisão geral do TCC e aplicação de ajustes

Referências

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

IBM (2024). What is monolithic architecture? https://www.ibm.com/think/topics/monolithic-architecture. Acesso em: 18 maio 2025.

ISO/IEC/IEEE (2011). ISO/IEC/IEEE 42010:2011 – Systems and software engineering — Architecture description. https://www.iso.org/standard/50508.html. Acesso em: 18 maio 2025.

Juhirdeen, S. (2024). What are the big companies still uses monolithic architecture. https://medium.com/@sadhnajuhirdeen/what-are-the-big-companies-still-uses-monolithic-architecture-2555a93 Acesso em: 18 maio 2025.

Júnior, E. (2023). A maldição dos sistemas monolíticos distribuídos. https://eximia.co/a-maldicao-dos-sistemas-monoliticos-distribuidos/. Acesso em: 18 maio 2025.

Referências

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

IBM (2024). What is monolithic architecture? https://www.ibm.com/think/topics/monolithic-architecture. Acesso em: 18 maio 2025.

ISO/IEC/IEEE (2011). ISO/IEC/IEEE 42010:2011 – Systems and software engineering — Architecture description. https://www.iso.org/standard/50508.html. Acesso em: 18 maio 2025.

Juhirdeen, S. (2024). What are the big companies still uses monolithic architecture. https://medium.com/@sadhnajuhirdeen/what-are-the-big-companies-still-uses-monolithic-architecture-2555a93 Acesso em: 18 maio 2025.

Júnior, E. (2023). A maldição dos sistemas monolíticos distribuídos. https://eximia.co/a-maldicao-dos-sistemas-monoliticos-distribuidos/. Acesso em: 18 maio 2025.

Apêndice

A. Exemplo de Seção em um Apêndice