UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Gabriel Dillenburg Martins

Análise Comparativa de REST e GraphQL em Sistemas de Microserviços: Avaliando Desempenho

Porto Alegre

2023

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Gabriel Dillenburg Martins

Análise Comparativa de REST e GraphQL em Sistemas de Microserviços: Avaliando Desempenho

Trabalho apresentado à disciplina Engenharia de Software Aplicada, pelo Curso de Especialização em Engenharia de Software da Universidade do Vale do Rio dos Sinos - UNISINOS, ministrada pelo(a) professor(a) Josiane Brietzke Porto

Porto Alegre

2023

Análise comparative de REST e GraphQL em Sistemas de Microserviços: Avaliando Desempenho

Gabriel Dillenburg Martins

[1]Universidade do Vale do Rio dos Sinos - UNISINOS
Porto Alegre – RS – Brasil

gabrielmartins5@edu.unisinos.br

Abstract. *This paper presents a comparative performance analysis between REST and GraphQL architectures within a microservices context. By employing an empirical approach, the study evaluates the impact of each architecture on latency, throughput, and resource consumption in a Dockerized environment. Leveraging Gatling for load testing, the research simulates up to 80,000 requests distributed over a one-minute span, providing insights into the scalability and efficiency of REST and GraphQL. The findings offer valuable considerations for developers and architects in choosing the appropriate API architecture for diverse application scenarios.*

Resumo. *Este artigo realiza uma análise comparativa de desempenho entre as arquiteturas de API REST e GraphQL em um contexto de microserviços. Através de uma metodologia empírica, avalia-se o impacto de cada arquitetura em termos de latência, throughput e consumo de recursos em um ambiente Dockerizado. Utilizando testes de carga com o Gatling, o estudo simula até 80.000 requisições distribuídas ao longo de um minuto, fornecendo insights sobre a escalabilidade e eficiência de REST e GraphQL. Os resultados apresentam considerações valiosas para desenvolvedores e arquitetos na escolha da arquitetura de API mais adequada para diferentes cenários de aplicação.*

1. Introdução

A crescente complexidade dos sistemas de software e a demanda por soluções escaláveis e eficientes em aplicações web modernas têm impulsionado o desenvolvimento e a adoção de diversas arquiteturas de API. Entre elas, as arquiteturas REST (Representational State Transfer) e GraphQL ganham destaque, oferecendo abordagens distintas para o gerenciamento e a transferência de dados entre sistemas cliente-servidor. Este estudo se propõe a realizar uma análise comparativa do desempenho entre REST e GraphQL, duas das arquiteturas de API

mais influentes na atualidade, no contexto de microserviços. No cenário tecnológico atual, a escalabilidade e eficiência dos sistemas de software tornaram-se focos primários de pesquisa e desenvolvimento. As arquiteturas de API, especialmente em aplicações web, são cruciais para a gestão e transferência eficaz de dados entre clientes e servidores. Duas das principais arquiteturas de API que se destacam são REST e GraphQL, cada uma apresentando características únicas e vantagens específicas [Fielding 2000; Hartig e Pérez 2017].

Este estudo foca na comparação de desempenho entre REST e GraphQL no contexto de microserviços, analisando métricas como tempo de resposta dos serviços, quantidade de respostas com e sem erro e quantidade de requisições simultâneas realizadas em cada rota de serviço. O estudo procura entender como essas arquiteturas respondem sob alta carga, uma situação comum em aplicações web modernas [Newman 2015].

Escolher a arquitetura de API apropriada é crucial para a escalabilidade e eficiência das aplicações web. Diante de múltiplas opções, os desenvolvedores enfrentam o desafio de optar pela arquitetura mais adequada às suas necessidades. O artigo busca responder à pergunta: "Como as arquiteturas REST e GraphQL se comparam em termos de desempenho em ambientes de microserviços sob alta carga?".

1.1. Objetivos

Este estudo tem como objetivo geral avaliar comparativamente o desempenho das arquiteturas REST e GraphQL em um ambiente de microserviços. E sobre os objetivos específicos da pesquisa, é possível listar abaixo:

- Analisar o tempo de resposta de REST e GraphQL sob altas cargas de trabalho.
- Medir o número de requisições com sucesso e erro alcançadas por REST e GraphQL em cenários de alta demanda.
- Avaliar a eficiência em cada rota disponível nos serviços.

A escolha deste tema deriva da importância crescente das arquiteturas de API no desenvolvimento de software. REST e GraphQL são amplamente utilizados na indústria, mas ainda há uma lacuna na literatura em relação à sua comparação direta de desempenho em cenários práticos [Posta 2018; Vazquez-Ingelmo et al. 2020]. Este estudo visa preencher essa lacuna, oferecendo análises baseadas em dados empíricos.

1.2. Estrutura do Artigo

O artigo é estruturado da seguinte maneira: Após a introdução, a Seção 2 aborda o embasamento teórico de REST e GraphQL. A metodologia utilizada é

descrita na Seção 3. Os resultados são apresentados na Seção 4, seguidos pela análise na Seção 5. A Seção 6 conclui o trabalho, resumindo as descobertas e sugerindo direções para pesquisas futuras.

2. Fundamentação teórica

2.1. Arquitetura Orientada a Serviços

A arquitetura orientada a serviços (SOA) é um paradigma para organizar e utilizar capacidades distribuídas que podem estar sob o controle de diferentes domínios de propriedade [Erl 2005]. Essa abordagem oferece uma maneira de integrar diversos serviços de software, com um foco em maior agilidade e flexibilidade nos negócios. Em SOA, os serviços são disponibilizados para outras entidades de software (clientes ou outros serviços) por meio de uma rede, geralmente a Internet.

2.2. Microserviços

Microserviços representam uma abordagem arquitetural que estrutura uma aplicação como uma coleção de serviços leves e independentes, executados em processos distintos [Newman 2015]. Cada microserviço é focado em uma única funcionalidade ou negócio e pode ser desenvolvido, implantado e escalado de forma independente. Esta abordagem promove a modularidade, facilita a manutenção, e é ideal para ambientes de desenvolvimento e entrega contínua [Fowler e Lewis 2014]. Microserviços podem ser expostos via APIs REST ou GraphQL, tornando a comparação de desempenho entre estas duas arquiteturas particularmente relevante no contexto de sistemas distribuídos.

2.3. REST (Representational State Transfer)

REST, um estilo arquitetural introduzido por Fielding em sua dissertação de doutorado, descreve um conjunto de restrições para criar serviços web [Fielding 2000]. REST utiliza um modelo de comunicação sem estado, no qual as requisições do cliente contêm todas as informações necessárias para o servidor processá-las. O REST se baseia fortemente em métodos HTTP padrão (GET, POST, PUT, DELETE) e é caracterizado por sua simplicidade e escalabilidade, facilitando a integração entre sistemas em um ambiente distribuído [Richardson e Ruby 2008].

2.4. GraphQL

GraphQL, desenvolvido pelo Facebook em 2012, é uma linguagem de consulta para APIs e um tempo de execução para atender essas consultas com dados existentes [Facebook Inc. 2015]. Diferente do REST, que usa múltiplos endpoints, o GraphQL opera sobre um único endpoint e oferece aos clientes a flexibilidade para requisitar exatamente os dados que precisam. Isso pode reduzir a quantidade de

dados transferidos em uma requisição, o que é particularmente útil em ambientes com restrições de largura de banda [Hartig e Pérez 2017].

2.5. Comparação entre REST e GraphQL

Enquanto REST e GraphQL servem ao mesmo propósito fundamental de facilitar a comunicação entre cliente e servidor, eles diferem significativamente em sua abordagem e design. REST é baseado em recursos com uma representação clara e uma abordagem padronizada usando métodos HTTP. Em contraste, GraphQL oferece uma maior flexibilidade na obtenção de dados, permitindo que os clientes definam a estrutura da resposta [Vazquez-Ingelmo et al. 2020]. Essas diferenças têm implicações importantes no desempenho, especialmente em termos de latência e throughput, que são exploradas neste estudo.

2.6. Gatling (Ferramenta para os testes de carga)

Gatling é uma ferramenta de teste de carga de código aberto projetada para medir o desempenho de aplicações web [Gatling Corp 2021]. Diferente de outras ferramentas de teste de carga, o Gatling usa uma DSL (Domain-Specific Language) baseada em Scala para a definição de cenários de teste, o que facilita a simulação de comportamentos complexos de usuários em aplicações web. Gatling é capaz de gerar um alto volume de solicitações para simular a carga em um sistema, coletando métricas detalhadas como tempo de resposta, número de solicitações por segundo e taxas de sucesso e erro. Isso o torna uma ferramenta valiosa para testar a performance de APIs REST e GraphQL, especialmente em um ambiente de microserviços.

3. Metodologia

3.1. Abordagem de Pesquisa

Esta pesquisa adota uma abordagem empírica quantitativa para avaliar e comparar o desempenho das arquiteturas REST e GraphQL em um ambiente de microserviços. O estudo utiliza testes de carga para simular cenários de usuário realistas, coletando dados quantitativos para análise [Kothari 2004].

3.2. Ambiente de Teste

Os testes foram realizados em um ambiente controlado utilizando Docker, que permite a criação de containers isolados para hospedar os microserviços REST e GraphQL. Cada microserviço foi implementado seguindo as melhores práticas de desenvolvimento para as respectivas arquiteturas [Turnbull 2014]. O Docker Compose foi utilizado para gerenciar e orquestrar os containers, garantindo consistência e replicabilidade do ambiente de teste [Nadareishvili et al. 2016].

3.3. Configuração dos Microserviços

Os microserviços foram desenvolvidos em Node.js, com a API REST construída utilizando o framework Express.js e a API GraphQL utilizando Apollo Server. Ambas as APIs foram configuradas para responder a operações CRUD (Create, Read, Update, Delete).

3.4. Testes de Carga com Gatling

Para os testes de carga, utilizou-se a ferramenta Gatling, que permite simular um grande número de usuários acessando as APIs simultaneamente. A configuração do Gatling foi realizada para gerar um total de 80.000 requisições distribuídas uniformemente em um período de um minuto para cada arquitetura de API. Cada teste foi repetido três vezes para garantir a precisão dos dados [Gatling Corp 2021].

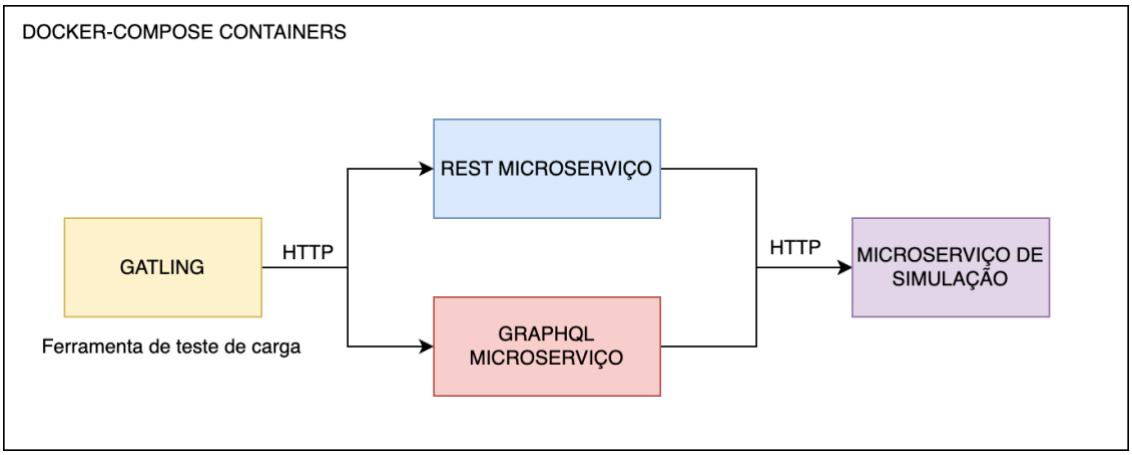

Figura 1 – Visão geral da arquitetura para a realização dos testes. Fonte: Elaborado pelo autor (2023).

3.5. Coleta e Análise de Dados

Durante os testes, métricas como latência, throughput e taxa de erro foram coletadas. Os dados foram então analisados usando métodos estatísticos para identificar diferenças significativas no desempenho entre as duas arquiteturas. A análise envolveu o uso de software estatístico para garantir rigor e objetividade nos resultados [Field 2013].

4. Resultados

Os testes de carga realizados para as arquiteturas REST e GraphQL no contexto de microserviços revelaram diferenças significativas em termos de desempenho e taxa de falhas. Ambos os testes consistiram em um total de 80.000 requisições para cada arquitetura.

4.1. Resultados REST

Métrica	Valor OK	Valor Falha
Requisições Bem-Sucedidas	53.412	-
Requisições com Falha	-	26.588
Tempo de Resposta Minimo (ms)	2	0
Tempo de Resposta Maximo (ms)	18,578	60,008
Tempo de Resposta Médio (ms)	6,625	7,310
Percentil 50 (ms)	6,595	10,001
Percentil 75 (ms)	8,277	10,003
Percentil 95 (ms)	14,728	20,015
Percentil 99 (ms)	15,425	20,086

Tabela 1 – Resultado global da arquitetura REST. Fonte: Elaborado pelo Autor (2023).

4.2. Resultados GraphQL

Métrica	Valor OK	Valor Falha
Requisições Bem-Sucedidas	46.424	-
Requisições com Falha	-	33.576
Tempo de Resposta Minimo (ms)	11	0
Tempo de Resposta Maximo (ms)	45,178	60,002
Tempo de Resposta Médio (ms)	9,143	5,663
Percentil 50 (ms)	8,716	0
Percentil 75 (ms)	11,964	20,002
Percentil 95 (ms)	17,435	20,006
Percentil 99 (ms)	28,446	20,020

Tabela 2 – Resulta global da arquitetura GraphQL. Fonte: Elaborado pelo Autor (2023).

4.3. Resultados Comparativos

Métrica	REST	GraphQl
Total de Requisições	80.000	80.000
Requisições Bem-Sucedidas	53.412 (66.8%)	46.424 (58%)
Requisições com Falha	26.588 (33.2%)	33.576 (42%)
Tempo de Resposta Mínimo (ms)	2 (ok), 0 (falha)	11 (ok), 0 (falha)
Tempo de Resposta Máximo (ms)	18,578 (ok), 60,008 (falha)	45,178 (ok), 60,002 (falha)
Tempo de Resposta Médio (ms)	6,625 (ok), 7,310 (falha)	9,143 (ok), 5,663 (falha)

Tabela 3 - Comparação entre REST e arquitetura GraphQL. Fonte: Elaborado pelo Autor (2023)

Estas tabelas oferecem uma visão clara e concisa das diferenças de desempenho entre as arquiteturas REST e GraphQL, baseada nos dados coletados durante os testes de carga. A análise desses dados na seção subsequente do artigo irá explorar as implicações desses resultados para a escolha entre REST e GraphQL em aplicações de microserviços.

5. Análise de Resultados

5.1. Interpretação geral dos resultados

Os dados coletados e apresentados nas seções anteriores fornecem uma visão clara das diferenças de desempenho entre as arquiteturas REST e GraphQL. Ao analisar esses resultados, é essencial considerar não apenas as métricas de desempenho brutas, mas também o contexto e as implicações dessas métricas no desenvolvimento e na manutenção de aplicações de microserviços.

5.2. Comparação de desempenho

A análise das taxas de sucesso das requisições revela que a arquitetura REST teve uma taxa de falha significativamente menor do que GraphQL. Este resultado pode ser atribuído a várias causas potenciais, como a maneira como cada arquitetura gerencia a carga de dados e as solicitações dos clientes. Com isso, se sugere que o GraphQL pode exigir mais recursos de processamento por solicitação [Hartig e Pérez 2017]. Os resultados mostram que o GraphQL teve uma latência média maior em comparação com o REST. Isso pode ser devido à complexidade adicional na construção de respostas personalizadas em GraphQL, o que pode aumentar o tempo de processamento [Facebook Inc. 2015]. Por outro lado, a capacidade do GraphQL de fornecer respostas precisas e específicas às solicitações do cliente pode reduzir o volume total de dados transmitidos, potencialmente compensando a latência adicional em redes com largura de banda limitada.

5.3. Conclusão da Análise

Esta análise dos resultados destaca a importância de considerar vários fatores ao escolher entre REST e GraphQL para a arquitetura de microserviços. A decisão deve ser baseada em uma avaliação cuidadosa das necessidades específicas da aplicação, das características de desempenho observadas e do contexto operacional em que a aplicação será implementada.

6. Conclusão e Direções para Pesquisas Futuras

Este estudo realizou uma comparação detalhada entre as arquiteturas REST e GraphQL no contexto de microserviços, com foco no desempenho e na escalabilidade. As principais descobertas incluem:

- Taxa de Sucesso das Requisições: REST demonstrou uma maior taxa de sucesso em comparação com GraphQL sob condições de carga intensiva.
- Latência e Tempo de Resposta: GraphQL apresentou um tempo de resposta médio maior do que REST para requisições bem-sucedidas.
- Implicações de Escalabilidade: Enquanto REST mostrou melhor desempenho sob carga, o tempo de resposta mais alto do GraphQL pode ser uma consideração crítica em ambientes altamente escaláveis.
- Considerações para Desenvolvimento: A escolha entre GraphQL e REST deve ser baseada em uma avaliação cuidadosa das necessidades específicas da

aplicação, incluindo o tipo de dados, a frequência de mudanças no esquema e a complexidade das operações.

6.1. Implicações Práticas

As descobertas deste estudo oferecem insights valiosos para desenvolvedores e arquitetos de sistemas na escolha da arquitetura de API mais adequada para suas aplicações de microserviços. Em particular, destacam a necessidade de avaliar as características de desempenho em relação aos requisitos específicos da aplicação e ao ambiente operacional.

6.2. Direções para Pesquisas Futuras

Este estudo abre várias direções para pesquisas futuras, incluindo:

- Análise de Custo-Benefício: Investigação mais profunda sobre o custo-benefício de cada arquitetura, considerando não apenas o desempenho, mas também outros fatores como manutenção, desenvolvimento e escalabilidade.
- Estudos em Diferentes Contextos de Aplicação: Realização de estudos semelhantes em diferentes contextos de aplicação para avaliar como as características específicas de cada arquitetura se adaptam a variados requisitos de negócio.
- Avaliação de Ferramentas de Teste Alternativas: Explorar o uso de outras ferramentas de teste de carga para validar e expandir as descobertas deste estudo.
- Impacto da Complexidade da Consulta no GraphQL: Pesquisas adicionais focadas em como a complexidade das consultas GraphQL afeta o desempenho geral da aplicação.

6.3. Conclusão

Este estudo contribui para uma compreensão mais aprofundada das arquiteturas REST e GraphQL, oferecendo uma base para decisões informadas no design e na implementação de microserviços. As descobertas destacam a importância de uma escolha cuidadosa da arquitetura de API, sugerindo que a decisão ideal pode variar significativamente dependendo das especificidades de cada aplicação.

Referências

[Fielding 2000] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.

[Hartig e Pérez 2017] Hartig, O., & Pérez, J. (2017). Semantics and Complexity of GraphQL. In Proceedings of the 2017 World Wide Web Conference.

[Newman 2015] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[Posta 2018] Posta, C. (2018). Microservices for Java Developers: A Hands-On Introduction to Frameworks and Containers. O'Reilly Media.

[Vazquez-Ingelmo et al. 2020] Vazquez-Ingelmo, A., García-Holgado, A., García-Peñalvo, F. J. (2020). Analyzing the usability of the W3C Web Content Accessibility Guidelines. In Universal Access in the Information Society, 19(1), 155-174.

[Erl 2005] Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall.

[Richardson e Ruby 2008] Richardson, L., & Ruby, S. (2008). RESTful Web Services. O'Reilly Media.

[Fowler e Lewis 2014] Fowler, M., & Lewis, J. (2014). Microservices a definition of this new architectural term. ThoughtWorks.

[Gatling Corp 2021] Gatling Corp. (2021). Gatling: Open-Source Load Testing.

[Kothari 2004] Kothari, C. R. (2004). Research Methodology: Methods and Techniques. New Age International.

[Turnbull 2014] Turnbull, J. (2014). The Docker Book: Containerization is the new virtualization. James Turnbull.

[Nadareishvili et al. 2016] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media.

[Field 2013] Field, A. (2013). Discovering Statistics Using IBM SPSS Statistics. Sage.

[Facebook Inc. 2015] Facebook Inc. (2015). GraphQL: A data query language.