Estudo Qualitativo sobre Ferramentas de Testes de Carga e Geração de Tráfego HTTP

Brito C. Gurgel; Oliveira L. Pereira {IFPB, Campus João Pessoa}

E-mails: brito.cleilson@academico.ifpb.edu.br, luciana.oliveira@ifpb.edu.br.

Palavras-Chave: microserviços; kubernetes; reprodutabilidade; escalabilidade.

1 Introdução

A reprodutibilidade é um dos pilares fundamentais da pesquisa científica, especialmente em áreas como a computação, onde os resultados precisam ser validados em condições controladas e replicáveis (LIU et al., 2021). Ferramentas de teste de carga e geração de tráfego são essenciais para estudos como arquiteturas de microsserviços, especialmente quando implantadas em orquestradores como o *Kubernetes*. Sua função é simular o comportamento realista de usuários para avaliar rigorosamente o desempenho, a escalabilidade e a resiliência da arquitetura.

Apesar de muitos trabalhos utilizarem ferramentas de geração de tráfego e teste de carga em suas metodologias, a escolha da ferramenta é frequentemente empírica. Nota-se uma ausência de análise crítica sobre as capacidades e particularidades que justifiquem qual seria a mais apropriada para os objetivos do projeto.

O trabalho de (SCHNEIDER, 2023) realiza uma avaliação comparativa dos principais *service meshes* em ambientes Kubernetes, integrando análise quantitativa (segurança, observabilidade) e *benchmarks* de desempenho práticos no *Google Kubernetes Engine* (GKE). Para garantir reprodutibilidade e avaliação rigorosa, o estudo incorporou ferramentas de geração de tráfego, permitindo a simulação controlada de cargas e a medição de métricas como latência e *throughput*. Contudo, o trabalho se insere no contexto onde raramente fornecem uma justificativa detalhada para a seleção de uma tecnologia em detrimento de outra.

Já o trabalho de (GARCIA; MENDEZ; MENDEZ, 2024) utilizou-se também de uma ferramenta de geração de tráfego para reproduzir um ambiente simulando acessos com o objetivo de atribuir pesos a certos *endpoints* para permitir analizar as interações entre os PODs para aplicar algoritmos de *deep learning* para predição da cargas de trabalho de CPU e memória.

Este artigo realiza uma análise comparativa qualitativa das ferramentas *Locust, JMeter, K6 e Artillary*.O estudo avalia a aplicação dessas ferramentas em microsserviços e ambientes conteinerizados com base em critérios essenciais, como suporte a HTTP/HTTPS, geração de carga realista, parametrização dinâmica, qualidade dos relatórios e escalabilidade distribuída. Inclui ainda análise de integração CI/CD (Integração e distribuição contínua), documentação, visando eficiência operacional em cenários práticos.

2 Materiais e Métodos

Este estudo empregou uma metodologia de análise comparativa qualitativa para avaliar ferramentas de geração de tráfego HTTP em ambientes de microsserviços.

2.1 Estudo de Caso

Para a validação e coleta de dados, foi utilizada a aplicação *"Online Boutique Microservices"* desenvolvida pela Google, disponível em (Google Cloud Platform, 2025),e que ilustrata uma arquitetura nativa da nuvem com 11 microsserviços interconectados. Esta aplicação foi implantada em um *cluster Kubernetes* utilizando a ferramenta *Docker desktop*, com o *service mesh Istio* e *Prometheus* para a coleta de dados. Para validar características não observadas na prática, consultamos a documentação oficial de cada ferramenta, utilizando um Modelo de Linguagem Amplo (LLM), o GPT-4.1, para otimizar a busca por funcionalidades específicas.

2.2 Ferramentas de Geração de Tráfego

As ferramentas selecionadas para análise foram:

- Locust: Uma ferramenta de teste de carga *open-source* baseada em Python, possui flexibilidade na criação de scripts e capacidade de distribuição. Doc. oficial ***https://docs.locust.io/en/stable/***

- JMeter (Apache JMeter): Uma aplicação o*pen-source* baseada em Java para teste de carga e desempenho de aplicações e serviços. Doc. oficial ***https://jmeter.apache.org/***

- k6: Uma ferramenta de teste de carga *open-source* e de alto desempenho, que permite escrever testes em JavaScript. Doc. oficial ***https://k6.io/docs/***

- Artillery: Uma ferramenta moderna e de código aberto para testes de carga e desempenho, com cenários definidos usualmente em YAML. Doc. oficial ***https://www.artillery.io/docs***

2.3 Critérios de Seleção das Ferramentas

A avaliação das ferramentas foi pautada por critérios técnicos essenciais para o contexto de microsserviços e contêineres:

- Suporte a HTTP/HTTPS Avançado: Capacidade de criar scripts personalizados, manipular cabeçalhos e cookies, e lidar com diversos métodos HTTP.

- Geração de Carga Realista: Flexibilidade para simular o comportamento do usuário com probabilidades distintas por *endpoint* e fluxos de usuário complexos.

- Parametrização e Variáveis: Facilidade para usar dados dinâmicos e encadear requisições.

- Relatórios e Métricas Detalhadas: Qualidade das métricas e dos relatórios para análise.

- Escalabilidade e Distribuição: Capacidade de gerar carga a partir de múltiplos nós ou em ambientes distribuídos.

- Integração com Service Mesh: Capacidade de operar e ser configurada em ambientes com *service mesh* (ex: Istio), compreendendo e respeitando as políticas de tráfego.

- Integração com CI/CD: Aptidão para ser incorporada em *pipelines* de Integração Contínua e Entrega Contínua (CI/CD).

- Comunidade e Suporte: Atividade da comunidade, qualidade da documentação e disponibilidade de suporte técnico.

2.4 Procedimento para Análise

Para avaliar a aderência de cada ferramenta aos critérios, foi realizada uma análise qualitativa atribuindo-se uma pontuação de 0 a 5. As notas foram justificadas com base na documentação oficial de cada ferramenta, considerando suas funcionalidades, arquitetura e flexibilidade no contexto de microsserviços e orquestração.

3 Resultados e Discussão

A análise comparativa revelou que Locust e k6 são as ferramentas mais eficazes para testes de carga em microsserviços. Ambas se destacam pela abordagem de 'testes como código' (Python para Locust, JavaScript para k6), que proporciona controle avançado sobre as requisições, a geração de carga e a parametrização dinâmica, essenciais para simular cenários realistas em aplicações como a *Online Boutique*. Ambas são escaláveis e integram-se facilmente com pipelines CI/CD, com vantagem para o k6 em relatórios e detalhamento de métricas. Em contrapartida, o JMeter, embora versátil, demonstrou limitações na flexibilidade e automação de cenários dinâmicos. Já o Artillery, apesar de ser uma ferramenta leve, apresentou restrições na geração de relatórios detalhados e na capacidade de distribuição de carga. A integração eficiente com *service mesh*, como Istio, depende da capacidade das ferramentas de manipular cabeçalhos HTTP e da observabilidade do

mesh. Assim, a alta programabilidade de Locust e k6 os torna ideais para testes de carga reprodutíveis em arquiteturas de microsserviços. A Figura 1 apresenta a pontuação atribuída a cada ferramenta com base nos critérios de avaliação. Para facilitar a comparação visual, foi plotado esses valores em um gráfico de radar, no qual a maior aderência a um critério é indicada pela proximidade à borda externa. Para total transparência, a justificativa detalhada para cada nota pode ser consultada em nosso repositório do projeto no GitHub: https://github.com/britocleilson/SIMPIF.

Análise Comparativa de Ferramentas de Geração de Tráfego

Pontuação das Ferramentas por Critério

	Locust	k6	JMeter	Artillery
Suporte a HTTP/HTTPS Avançado	5	5	4	4
Geração de Carga Realista	5	5	4	4
Parametrização e Variáveis	5	5	3	4
Relatórios e Métricas Detalhadas	3	5	4	3
Escalabilidade e Distribuição	5	5	4	3
Integração com Service Mesh	4	4	3	3
Integração com CI/CD	5	5	3	4
Comunidade e Suporte	4	5	5	3

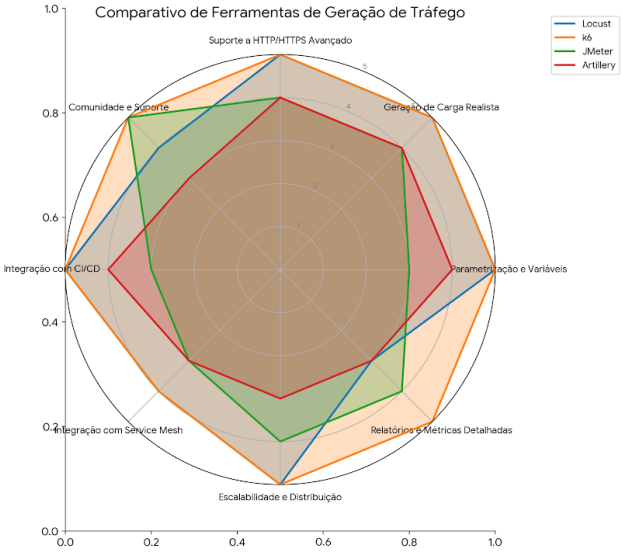

Figura 1: Comparativo das Ferramentas

4 Considerações Finais

O desempenho equivalente entre Locust e k6 indica que a escolha ideal depende do contexto do projeto e da equipe técnica. O Locust se sobressai em cenários complexos que se beneficiam da integração com o ecossistema Python. Por outro lado, o k6 é a melhor opção quando a prioridade é a integração com *pipelines* CI/CD, relatórios avançados e monitoramento em tempo real. A decisão deve equilibrar aspectos técnicos e a familiaridade da equipe com cada ferramenta. Observa-se ainda que, embora o uso dessas ferramentas seja frequente em pesquisas, raramente há justificativas detalhadas para a escolha de uma ou outra.

Referências

GARCIA, J. L. C.; MENDEZ, R. M.; MENDEZ, R. M. Deep learning-based resource utilization prediction for multi-pod applications in kubernetes. In: RIVERA, M. F. M.; ZAGAL-FLORES, R.; BARRIA-HUIDOBRO, C. (Ed.). *Telematics and Computing*. Cham: Springer Nature Switzerland, 2024. p. 202–217. ISBN 978-3-031-77293-1.

Google Cloud Platform. *Google Online Boutique Microservices Demo*. 2025. <https://github.com/GoogleCloudPlatform/microservices-demo>. Accessed: 29 may 2025.

LIU, C. et al. On the reproducibility and replicability of deep learning in software engineering. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 31, n. 1, out. 2021. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3477535>.

SCHNEIDER, M. *Performance Benchmarking of Open-Source Service Meshes Using Load Testing*. Tese (Doutorado) — Master's thesis, University of Applied Sciences Campus Vienna, Vienna, Austria, 2023.