UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUIZ HENRIQUE OLIVEIRA SILVA

ANÁLISE DE AMBIENTAÇÃO DE APLICAÇÕES *PHP* E *.NET* EM NUVEM: UM ESTUDO BASEADO EM DESEMPENHO

QUIXADÁ

2025

LUIZ HENRIQUE OLIVEIRA SILVA

ANÁLISE DE AMBIENTAÇÃO DE APLICAÇÕES *PHP* E *.NET* EM NUVEM: UM ESTUDO BASEADO EM DESEMPENHO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Arthur de Castro Callado.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581a Silva, Luiz Henrique Oliveira.
Análise de ambientação de aplicações PHP e .NET em nuvem : um estudo baseado em desempenho / Luiz Henrique Oliveira Silva. – 2025.
101 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2025.
Orientação: Prof. Dr. Arthur de Castro Callado.

1. Computação em nuvem. 2. PHP. 3. .NET. 4. AWS. 5. AZURE. I. Título.

CDD 004

LUIZ HENRIQUE OLIVEIRA SILVA

ANÁLISE DE AMBIENTAÇÃO DE APLICAÇÕES *PHP* E *.NET* EM NUVEM: UM ESTUDO BASEADO EM DESEMPENHO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 29/07/2025

BANCA EXAMINADORA

Prof. Dr. Arthur de Castro Callado (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Marcelo Uchôa de Alencar
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Rafael Braga
Universidade Federal do Ceará (UFC)

A Deus, por me permitir chegar até aqui. Àquela voz interior, que sempre me deu forças para continuar, mesmo quando tudo parecia incerto.

RESUMO

As linguagens de programação, em seus aspectos gerais, compartilham o mesmo objetivo principal: resolver problemas computacionais, apesar de apresentarem características distintas. Desse modo, o presente estudo realiza uma comparação entre o PHP, uma linguagem com tipagem dinâmica e interpretada em tempo de execução, e o C#, linguagem da plataforma .NET, fortemente tipada e compilada previamente para código intermediário. A utilização de duas linguagens que operam de formas distintas é interessante para analisar comportamentos que podem se estender a outras linguagens semelhantes. Dessa forma, a comparação ocorre em ambientes de nuvem, especificamente na AWS e no Azure. A escolha desses ambientes baseou-se em critérios como desempenho, popularidade, escalabilidade e aceitação por desenvolvedores. Para testar essas aplicações, foram selecionadas as ferramentas de teste de carga *JMeter* e *K6*, escolhidas após uma análise prévia das principais ferramentas *open source*. Os testes foram realizados utilizando métodos de leitura e gravação nas aplicações, permitindo identificar diferenças significativas no comportamento dos ambientes analisados. A AWS apresentou resultados superiores para operações de leitura, enquanto o Azure destacou-se nas operações de escrita, especialmente com aplicações .NET. Os resultados também evidenciaram diferenças importantes em relação aos custos operacionais dos ambientes avaliados. Assim, este estudo oferece informações importantes na escolha criteriosa de provedores de nuvem, considerando performance, tipo de operação predominante e orçamento disponível.

Palavras-chave: Computação em nuvem; PHP; .NET; C#; AWS; Azure

ABSTRACT

Programming languages, in their general aspects, share the primary objective of solving computational problems, despite having distinct characteristics. Therefore, this study provides a comparative analysis between PHP, a dynamically-typed language interpreted at runtime, and C#, a .NET platform language which is strongly-typed and pre-compiled into intermediate code. Using two languages with fundamentally different operational modes is useful for understanding behaviors that can be generalized to other similar languages. The comparison is conducted within cloud environments, specifically AWS and Azure, selected based on performance, popularity, scalability, and developer adoption criteria. For load-testing these applications, the tools *JMeter* and *K6* were selected after an initial analysis of leading open-source performance testing tools. Tests involving read and write operations were executed, enabling the identification of significant differences in behavior between the analyzed environments. AWS presented superior results for read operations, while Azure stood out in write operations, particularly with .NET applications. The findings also highlighted significant differences in operational costs between the evaluated environments. Thus, this study provides valuable insights to assist in the careful selection of cloud providers, considering performance, predominant application operations, and available budget.

Keywords: Cloud Computing; PHP; .NET; C#; AWS; Azure

LISTA DE FIGURAS

Figura 1 – Aspectos da evolução da *web* 17
Figura 2 – Demonstração da arquitetura cliente-servidor proposta por Tim Berners-Lee. 18
Figura 3 – Demonstração simples do design de Application Programming Interface (API) Representational State Transfer (REST). 19
Figura 4 – Demonstração de uma arquitetura de computação em nuvem 23
Figura 5 – Virtualização com suporte em *hardware* 24
Figura 6 – Serviços ofertados na computação em nuvem 26
Figura 7 – Demonstração entre nuvem privada, pública e privada 27
Figura 8 – Etapas metodológicas do presente trabalho 33
Figura 9 – Linguagem utilizada na criação da ferramenta Hey 37
Figura 10 – Popularidade da ferramenta Hey no GitHub 38
Figura 11 – Versão da linguagem Go 38
Figura 12 – Instalação da ferramenta Hey 39
Figura 13 – Demonstração de teste na ferramenta Hey 41
Figura 14 – Linguagens utilizadas na ferramenta Wrk 42
Figura 15 – Popularidade da ferramenta Wrk 42
Figura 16 – Detalhes da máquina virtual criada na Google Cloud 43
Figura 17 – Instalação e teste na ferramenta Wrk 44
Figura 18 – Linguagens utilizadas na ferramenta JMeter 45
Figura 19 – Popularidade da ferramenta JMeter 45
Figura 20 – Configuração de usuários virtuais no *Apache* JMeter 46
Figura 21 – Resultados do teste na ferramenta *Apache* JMeter 47
Figura 22 – Linguagens utilizadas na ferramenta K6 48
Figura 23 – Popularidade da ferramenta K6 48
Figura 24 – Instalação Chocolatey 49
Figura 25 – Instalação da ferramenta K6 49
Figura 26 – Resultados do teste na ferramenta K6 50
Figura 27 – Comparação entre os ambientes AWS, Azure e GCP 54
Figura 28 – Receita no mercado de infraestrutura em nuvem. 55
Figura 29 – Utilização de nuvem por parte de desenvolvedores profissionais. 56
Figura 30 – Utilização de nuvem por parte de usuários aprendendo a programar. 56

Figura 31 – Receita da Microsoft em serviços. 59
Figura 32 – Performance de *e-commerces* da Tailândia. 60
Figura 33 – Ferramentas de testes de cargas mais ativas no ano de 2020. 60
Figura 34 – Interesse na ferramenta Hey nos últimos 5 anos. 61
Figura 35 – Interesse na ferramenta Wrk nos últimos 5 anos. 62
Figura 36 – Interesse nas ferramentas Hey vs Wrk nos últimos 5 anos. 63
Figura 37 – Interesse na ferramenta JMeter nos últimos 5 anos. 64
Figura 38 – Interesse nas ferramentas JMeter vs Hey vs Wrk nos últimos 5 anos 65
Figura 39 – Interesse na ferramenta K6 nos últimos 5 anos. 66
Figura 40 – Interesse nas ferramentas K6 vs JMeter vs Hey vs Wrk nos últimos 5 anos. . 67
Figura 41 – Diagrama de classes da aplicação elaborado no *software* Astah 69
Figura 42 – Plano utilizado na *Azure Application Service* 70
Figura 43 – Demonstração do *deploy* PHP através do *Github Actions* 71
Figura 44 – Configuração do *Nginx* na Azure 71
Figura 45 – Demonstração do *deploy* .NET através do *Visual Studio* 72
Figura 46 – Plano utilizado no *Elastic Beanstalk* 73
Figura 47 – Configuração do *Nginx* na AWS 74
Figura 48 – Configuração da ferramenta *JMeter* para o método GET 75
Figura 49 – Configuração da ferramenta *JMeter* para o método POST 76
Figura 50 – *Summary Report* do método GET na aplicação .NET na Azure. 76
Figura 51 – *Graph Result* do método GET na aplicação .NET na Azure. 77
Figura 52 – *Summary Report* do método GET na aplicação .NET na AWS. 77
Figura 53 – *Graph Result* do método GET na aplicação .NET na AWS. 78
Figura 54 – *Summary Report* do método POST na aplicação .NET na Azure. 78
Figura 55 – *Graph Result* do método POST na aplicação .NET na Azure. 79
Figura 56 – *Summary Report* do método POST na aplicação .NET na AWS. 79
Figura 57 – *Graph Result* do método POST na aplicação .NET na AWS. 80
Figura 58 – *Summary Report* do método GET na aplicação PHP na Azure. 80
Figura 59 – *Graph Result* do método GET na aplicação PHP na Azure. 81
Figura 60 – *Summary Report* do método GET na aplicação PHP na AWS. 81
Figura 61 – *Graph Result* do método GET na aplicação PHP na AWS. 82
Figura 62 – *Summary Report* do método POST na aplicação PHP na Azure. 82

Figura 63 – *Graph Result* do método POST na aplicação PHP na Azure. 83
Figura 64 – *Summary Report* do método POST na aplicação PHP na AWS. 83
Figura 65 – *Graph Result* do método POST na aplicação PHP na AWS. 84
Figura 66 – Resultado do método GET na aplicação .NET no ambiente da Azure 85
Figura 67 – Resultado do método POST na aplicação .NET no ambiente da Azure . . . 86
Figura 68 – Resultado do método GET na aplicação .NET no ambiente da AWS 86
Figura 69 – Resultado do método POST na aplicação .NET no ambiente da AWS 87
Figura 70 – Resultado do método GET na aplicação PHP no ambiente da Azure 88
Figura 71 – Resultado do método POST na aplicação PHP no ambiente da Azure 88
Figura 72 – Resultado do método GET na aplicação PHP no ambiente da AWS 89
Figura 73 – Resultado do método POST na aplicação PHP no ambiente da AWS 90

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASP	Application Service Provisioning
AWS	Amazon Web Services
BaaS	Backup as a Service
CD	Continuous Deployment
CERN	European Council for Nuclear Research
CI	Continuous Integration
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
EC2	Elastic Compute Cloud
FTP	File Transfer Protocol
GB	Gigabyte
GCP	Google Cloud Platform
GFLOPS	Giga Floating-Point Operations Per Second
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IOPS	Storage I/O
IoT	*Internet das Coisas*
JSON	JavaScript Object Notation
PaaS	Platform as a Service
PDC	Professional Developers Conference
REST	Representational State Transfer
S3	Simple Storage Service
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
TI	Tecnologia da Informação
URL	Uniform Resource Locator
VM	Virtual Machine

VU	Virtual User
WWW	Word Wide Web
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 13

1.1 Objetivos 14

1.1.1 Objetivo Geral 15

1.1.2 Objetivos Específicos 15

2 FUNDAMENTAÇÃO TEÓRICA 16

2.1 Desenvolvimento *web* 16

2.2 Computação em nuvem 20

2.3 Camadas da arquitetura de computação em nuvem orientada a serviços 24

3 TRABALHOS RELACIONADOS 29

3.1 Desvendando as Nuvens: Uma análise comparativa de desempenho de aplicações *Serverless* e Containers em provedores de computação Em nuvem 29

3.2 Computação em nuvem: Uma análise comparativa das plataformas disponibilizadas por Amazon, Google e Microsoft 29

3.3 An Empirical Study on the Impact of Programming Languages on the Performance of Open-source Serverless Platforms 30

3.4 Comparação entre os trabalhos 31

4 METODOLOGIA 33

4.1 Definir critérios para seleção de nuvem 33

4.2 Selecionar ambientes de nuvem 35

4.3 Definir critérios de ferramentas de teste de carga 35

4.4 Definição, configuração e aplicação prática 36

4.5 Avaliar aplicação em cada ambiente escolhido 36

4.6 Síntese dos resultados 36

5 RESULTADOS PRELIMINARES 37

5.1 Instalação das ferramentas de testes de cargas 37

5.1.1 Instalação e teste da ferramenta Hey 37

5.1.2 Instalação e teste da ferramenta Wrk 41

5.1.3 Instalação e teste da ferramenta JMeter 44

5.1.4 Instalação e teste da ferramenta K6 48

6 RESULTADOS . . . 51

6.1 Critérios para seleção de nuvem . . . 51

6.2 Seleção de ambientes de nuvem . . . 55

6.2.1 Visão geral dos provedores selecionados . . . 57

6.2.1.1 Amazon Web Services . . . *57*

6.2.1.2 Microsoft Azure . . . *57*

6.3 Critérios de ferramentas de teste de carga . . . 59

6.3.1 Hey . . . 61

6.3.2 Wrk . . . 62

6.3.3 JMeter . . . 63

6.3.4 K6 . . . 65

6.3.5 Comparativo e seleção das ferramentas . . . 67

6.4 Configuração e aplicação prática . . . 68

6.4.1 Deploy das aplicações no ambiente Azure . . . 70

6.4.2 Deploy das aplicações no ambiente AWS . . . 72

6.5 Avaliação das aplicações nos ambientes . . . 74

6.5.1 Testes de carga com a ferramenta JMeter na aplicação .NET . . . 75

6.5.1.1 Testes de cargas com método GET . . . *76*

6.5.1.2 Testes de cargas com método POST . . . *78*

6.5.2 Testes de carga com a ferramenta JMeter na aplicação PHP . . . 80

6.5.2.1 Testes de cargas com método GET . . . *80*

6.5.2.2 Testes de cargas com método POST . . . *82*

6.5.3 Testes de carga com a ferramenta K6 na aplicação .NET . . . 84

6.5.4 Testes de carga com a ferramenta K6 na aplicação PHP . . . 87

6.6 Síntese dos resultados . . . 90

7 CONCLUSÃO . . . 94

REFERÊNCIAS . . . 96

1 INTRODUÇÃO

No início, a Internet era basicamente um armazenamento de informações acessível universalmente e tinha pouco efeito nos sistemas de software. Esses sistemas executavam em computadores locais e eram acessíveis apenas dentro da organização. Por volta do ano 2000, a Internet começou a evoluir, e mais e mais recursos passaram a ser adicionados aos navegadores (Sommerville, 2011).

Nesse sentido, para Schwab (2016) as tecnologias digitais, fundamentadas no computador, software e redes, não são novas, mas estão causando rupturas à terceira revolução industrial; estão se tornando mais sofisticadas e integradas e, consequentemente, transformando a sociedade e a economia global.

De posse desse conceito, é necessário explorar as formas de infraestrutura de serviços que surgiram ou foram adaptadas para serem utilizadas na Internet. Entre elas, está a *Cloud Computing*, que utiliza a internet como meio de acesso a recursos computacionais. Para Tsai *et al.* (2010), as nuvens surgiram como uma infraestrutura de computação que possibilita a entrega rápida de recursos computacionais como um serviço, de forma escalável e virtualizada.

Conforme abordado por Qian *et al.* (2009) a computação em nuvem é uma técnica computacional onde serviços de TI são fornecidos por um grande número de unidades computacionais de baixo custo conectadas por redes IP. Sendo capaz de fornecer uma estrutura mais barata em um cenário de alocação de servidores.

A computação em nuvem, o sonho de longa data de computação como uma utilidade, tem o potencial de transformar uma grande parte da indústria de TI, tornando o software ainda mais atraente como serviço (Buyya *et al.*, 2009). Nesse sentido, a disponibilização de recurso encontrada nos serviços de nuvem é vista como uma maneira mais fácil de manter um software.

No entanto, embora a computação em nuvem esteja na vanguarda da transformação digital do século XXI, não podemos isentá-la de preocupações nos quesitos de segurança e desempenho. Conforme abordado por Gunukula (2024), ao explorarmos o futuro da computação em nuvem, também devemos abordar essas preocupações e investigar como as tecnologias emergentes e as melhores práticas podem mitigar riscos ao mesmo tempo em que maximizam os benefícios da nuvem.

Em vista deste contexto, aplicações responsivas alocadas em nuvem e que operem em situações de estresse ofertando um bom desempenho é de suma importância. Os serviços web desempenham um papel crucial na economia atual, permitindo que empresas e usuários

acessem aplicações e dados remotamente, facilitando a comunicação e a automação de processos. No entanto, esses serviços podem apresentar variações significativas de desempenho quando executados em diferentes provedores de nuvem, devido às particularidades da infraestrutura e das otimizações oferecidas por cada um (Almeida; Filho, 2024).

Segundo Buyya *et al.* (2009), algumas das aplicações baseadas em nuvem, tanto tradicionais quanto emergentes, incluem redes sociais, hospedagem de sites, entrega de conteúdo e processamento de dados instrumentados em tempo real. Cada um desses tipos de aplicação possui diferentes requisitos de composição, configuração e implantação. Com isso, para a IBM (2025), obter insights sobre como melhorar o desempenho de seus aplicativos e a disponibilidade para os usuários pode ter impactos significativos em seus resultados e na sustentabilidade geral de seus produtos e serviços.

Diante desse cenário, notando-se as diversas ofertas de serviços em nuvens por parte de grandes empresas, este trabalho visa analisar o comportamento e desempenho de aplicações *PHP* e *.NET* hospedadas em nuvem, considerando critérios de desempenho em suas respectivas operacionalidades. A ideia é orientar desenvolvedores na escolha de um ambiente em nuvem que entregue uma melhor facilidade na hospedagem da aplicação com um desempenho favorável, além de fornecer uma orientação teórica e prática sobre esses serviços.

A organização deste trabalho está estruturada da seguinte maneira: o Capítulo 2 explora os fundamentos teóricos que sustentam os principais conceitos utilizados ao longo da pesquisa. No Capítulo 3, são discutidos os estudos relacionados, oferecendo um panorama do cenário no qual este trabalho está inserido. O Capítulo 4 descreve em detalhes a metodologia empregada. No Capítulo 5, são demonstrado os resultados preliminares das ferramentas de cargas. No Cápitulo 6 são apresentados e analisados os resultados obtidos. Finalmente, o Capítulo 7 traz as considerações finais do estudo. Essa estrutura foi pensada para garantir uma apresentação clara, lógica e progressiva do conteúdo desenvolvido.

1.1 Objetivos

Nesta seção serão apresentados os objetivos gerais e específicos ao trabalho proposto.

1.1.1 Objetivo Geral

O objetivo geral desse trabalho é apresentar uma análise de aplicações *PHP* e *.NET* alocadas em nuvens selecionadas baseadas em critérios de desempenho, escalabilidade e utilização.

1.1.2 Objetivos Específicos

- Identificar as diferenças entre os provedores em serviços de nuvem com a maior popularidade;
- Analisar em cada provedor identificado, em vista de suas diferenças, dois que melhor ofertem em critérios de desempenho, escalabilidade, baixo custo e utilização;
- Diferenciar os serviços baseado nos critérios definidos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tratará do embasamento dos conceitos utilizados no decorrer do estudo. Desse modo, estuda-se conceitos sobre o desenvolvimento *web*, computação em nuvem e camadas da arquitetura de computação em nuvem orientada a serviços.

2.1 Desenvolvimento *web*

A internet, do modo que era conhecida até a década de 1990, tratava-se apenas de uma rede para troca de informações entre universidades e centros de pesquisas. De modo que, os protocolos para a comunicação eram bastante técnicos, o que dificultava o seu uso por parte de usuários comuns. Uma nova aplicação, a Word Wide Web (WWW), mudou essa realidade e atraiu para a rede milhares de novos usuários, sem a menor pretensão acadêmica (Tanenbaum, 2020).

Criada por Tim Berners-Lee, a *web* 1.0 foi a primeira versão da internet que podia ser utilizada por usuários comuns. Na *web* 1.0, os sites não tinham muita interatividade. Era possível ler as coisas que outras pessoas publicavam, mas muito pouco além disso. Basicamente, era como os jornais e revistas digitais, mas com a seção de comentários desativada (Brave, 2022).

Ao final dos anos 1990, inicia-se a transição da *web* 1.0 para a *web* 2.0. Essa transição ocorreu à medida que a infraestrutura e as ferramentas de desenvolvimento da internet se tornaram mais avançadas e que mais pessoas começaram a participar (Brave, 2022). De certa forma, isso caracterizou o desenvolvimento *web*, visto que essa transição mudou significamente a disponibilidade de conteúdos na *web*.

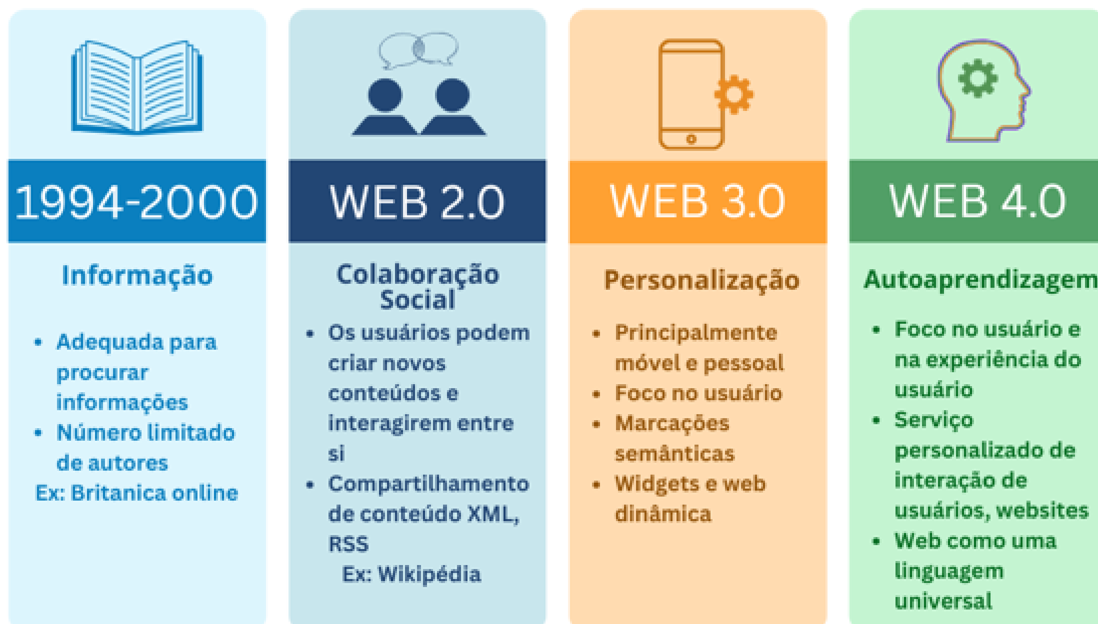
Para O'Reilly (2005), a *web* 2.0, surge em meio a observação de uma explosão de novos sites e novos aplicativos . Dessa forma, a *web* 2.0 tratou-se de uma segunda geração de serviços com tecnologias e ideias que permite a colaboração e troca de informações, possibilitando participação ativa do usuário na produção e disseminação de conteúdo, tornando a comunicação na *Internet* bidirecional com mais recursos dinâmicos.

Ademais, com o impulsionamento da tecnologia e novas formas de desenvolvimento, surge a *Web* 3.0, reconhecida como *Web* semântica, tornou-se operativa em 2006, propondo-se a dar sentido aos dados espalhados na rede. Começa a aparecer uma modulação do ambiente *web*, que passa a incorporar a capacidade de interpretar seus próprios conteúdos, e ofertar personalização e otimização da experiência online dos usuários, buscando uma navegação mais

dirigida (Guimaraes; Rocha, 2021).

Nesse sentido, a *Web* 4.0 surge com o reaproveitamento dos recursos da *Web* 3.0 e ampliação das tecnologias com uma forte utilização de inteligência artificial, *Internet das Coisas* (IoT) e integração de serviços (HUBSPOT, 2023). Segundo Guimaraes e Rocha (2021), a *web* 4.0, tem como foco o comportamento a partir de informações e dados do usuário, com o objetivo de fornecer ao usuário uma melhor experiência baseada em suas buscas. Ademais, a utilização de tecnologias de *Machine Learning* e *Deep Learning* são fundamentais para a assertividade no processamento dessas informações. A Figura 1 ilustra a evolução da *web* e alguns dos seus principais aspectos ao longo dos anos.

Figura 1 – Aspectos da evolução da *web*



Fonte: Adaptado de (Webnode, 2023)

Não obstante, o passo dado por Tim Berners-Lee na criação da *web* 1.0 foi importante para todo o desenvolvimento *web* que surgiria a seguir. Dessa forma, segundo CERN (2025), a proposta do protocolo HyperText Transfer Protocol (HTTP), um protocolo baseado em requisição e resposta que serviu como um divisor de águas para a transferência de dados na *web*, facilitando a interoperabilidade entre as plataformas.

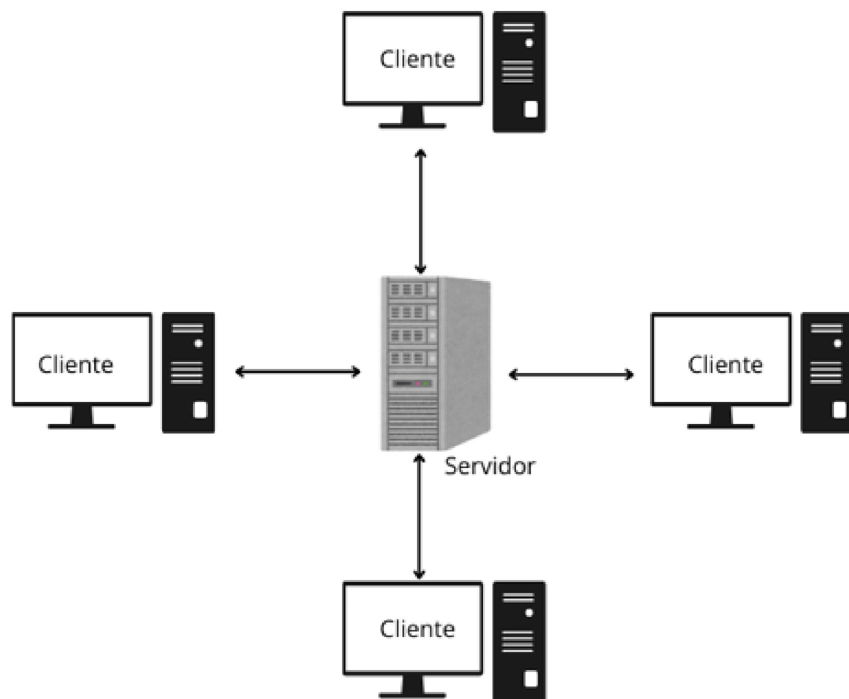
Ademais, Tim Berners-Lee estruturou o inicio da *web* 1.0 criando o HyperText Markup Language (HTML). Criada no European Council for Nuclear Research (CERN) e lançada em 1991, o HTML é uma linguagem de marcação utilizada para criar e exibir conteúdo na *web*. Sua primeira versão incluía apenas tags básicas para formatar textos, incluir *links* e

imagens em preto e branco (Vieira, 2023).

Para sustentar o HTTP em seu modelo de requisição e resposta, Berners-Lee também propôs uma arquitetura baseada em cliente-servidor. Desse modo, quando bem definido desde o início, o sistema pode ser mais escalável, flexível e duradouro, onde diferentes tecnologias e fontes de dados podem coexistir e evoluir sem perder compatibilidade (Berners-Lee, 1989).

A Figura 2 representa o modelo cliente-servidor proposto por Berners-Lee (1989) no início da WWW. Na representação há um servidor responsável por processar as requisições dos quatros clientes conectados. Os clientes são basicamente navegadores que solicitam recursos ao servidor, como por exemplo, páginas HTML. Essa solicitação de recursos é feita mediante o protocolo HTTP, no qual, o cliente emite uma *request*, e o servidor devolve uma *response*.

Figura 2 – Demonstração da arquitetura cliente-servidor proposta por Tim Berners-Lee.



Fonte: Adaptado de (Berners-Lee, 1989)

Dessa forma, o HTTP e o HTML serviram como alicerce para essa *web* primitiva que logo cresceu de maneira exponencial. Segundo Fielding *et al.* (2017), em 1993, o número de servidores *web* públicos cresceu a uma taxa exponencial, dobrando a cada três meses, e continuou nesse ritmo acelerado por mais de três anos. Entre o crescimento do interesse comercial pela *web* e a velocidade com que extensões estavam sendo introduzidas, o sucesso estava despedaçando a

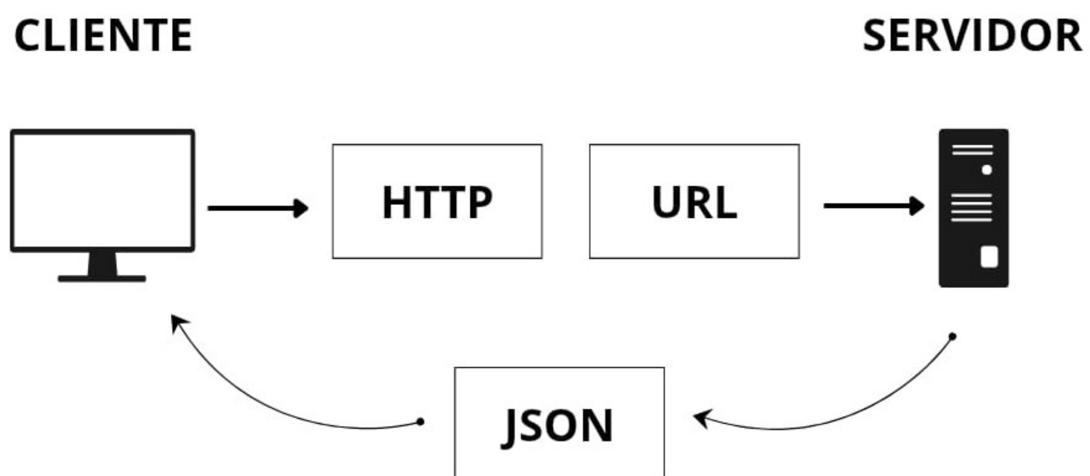
comunidade de desenvolvimento da *web* devido ao caos organizacional.

Baseando-se no que foi construindo por Tim Berners-Lee, surge o REST, um estilo arquitetural para a construção de serviços. Segundo a IBM (2024b), o REST surge no ano 2000, idealizado pelo cientista da computação Roy Fielding em sua tese de doutorado, e oferece um nível relativamente alto de flexibilidade, escalabilidade e eficiência para desenvolvedores.

De modo geral, o REST utiliza o protocolo HTTP para comunicar ao servidor o que deve ser feito com o recurso recebido. Nesse sentido, a API define um conjunto de regras que deve ser seguido por um desenvolvedor para estabelecer uma comunicação entre aplicações (Amazon Web Services, 2024). Dessa forma, a API REST se tornou bastante utilizada devido ao fato de oferecer uma forma simples, padronizada e eficiente para a comunicação entre serviços.

A Figura 3 apresenta uma demonstração do modelo REST proposto por Roy Fielding. Na Figura, há um cliente responsável por enviar requisições HTTP por meio de uma Uniform Resource Locator (URL). Do outro lado, há o recebimento dessa requisição por parte do servidor, onde ocorre o processamento e o envio de uma resposta. Essa resposta pode ser em diversos formatos, porém, o formato mais utilizado é o JavaScript Object Notation (JSON). Segundo a IBM (2024b), o JSON é popular porque pode ser lido por humanos e máquinas – e é independente da linguagem de programação.

Figura 3 – Demonstração simples do design de API REST.



Fonte: Adaptado de (Microsoft Learn, 2024)

A escolha das linguagens de programação utilizadas neste trabalho — *PHP* e *.NET* — se deu por alinhamento técnico com a evolução dos padrões da *web*. O surgimento do protocolo

HTTP e, posteriormente, da arquitetura REST, redefiniu a forma como aplicações se comunicam pela internet, promovendo um modelo cliente-servidor padronizado. Essa padronização impulsionou o crescimento de linguagens orientadas ao desenvolvimento *web*, como o *PHP*, que segundo PHP (2024), nasceu e evoluiu diretamente para responder às demandas do servidor HTTP.

Por sua vez, a plataforma *.NET*, com o C# como linguagem principal, incorporou nativamente o suporte a API REST com estruturas modernas como o *ASP.NET Core*, aliando desempenho, tipagem estática e compilação antecipada.

Ambas as tecnologias são profundamente influenciadas pelas exigências do modelo HTTP/REST: o *PHP*, com sua leveza e simplicidade para responder a requisições *web*, e o *.NET*, com sua robustez e eficiência para aplicações corporativas. Assim, a escolha dessas linguagens reflete não apenas dois paradigmas distintos — tipagem dinâmica versus estática — mas também uma afinidade operacional com outras linguagens: como *Java* e *Go*, no caso do C#, e *JavaScript* e *Python*, no caso do PHP.

Nesse sentido, é clara a importância da fundamentação da *web* por Tim Berners-Lee, e o seguimento por parte de outros protagonistas como Roy Fielding. Os esforços e descobertas realizadas por esses entusiastas da área tecnológica são os pilares para o funcionamento da *web* ainda nos tempos modernos.

2.2 Computação em nuvem

Com o crescente avanço da disponibilidade de serviços *web*, a alocação de recursos gerou grandes custos para as empresas. Nesse sentido, alternativas surgiram para mitigar esses impactos. Uma dessas alternativas foi a computação em nuvem. Segundo o Google Cloud (2024), a nuvem oferece mais flexibilidade e confiabilidade, desempenho e eficiência melhorados e ajuda a reduzir os custos.

Entretanto, o termo de computação em nuvem na forma que conhecemos hoje foi um aprimoramento de técnicas e conceitos ao longo dos anos. Segundo a RNP (2023), o americano John McCarthy discutiu na decáda de 1950 o uso compartilhado do computador, de forma simultânea, por dois ou mais usuários. O conceito foi chamado por ele de *Utility Computing*.

Nesse sentido, outro conceito importante foi o *time sharing*. Segundo O'Sullivan (1967), o *time sharing*, conhecido como tempo compartilhado, surge ao final da década de 1960 e envolve a alocação de recursos computacionais compartilhados entre usuários em terminais

remotos. No *time sharing* o tempo de processamento da Central Processing Unit (CPU) é dividido entre os usuários, onde as atividades de cada usuário são processadas de maneira alternada, dando a impressão ao usuário de um sistema só para si.

Conforme os estudos e aplicações do compartilhamento de recursos avançaram, empresas de telecomunicações começaram a basear-se nesses conceitos e começaram a utilizar a mesma estrutura física para permitir o acesso compartilhado por diferentes usuários (RNP, 2023).

Nesse contexto, a computação em nuvem utilizou conceitos anteriores para se formalizar da forma que conhecemos atualmente. Segundo Mishra (2014), o termo "computação em nuvem" foi utilizado por funcionários da Compaq Computer em 1996, e em 1997 o professor da Universidade do Texas, Ramnath Chellappa, utilizou o termo em uma de sua palestra intitulada "Intermediários na Computação em Nuvem: Um Novo Paradigma de Computação".

Computação em nuvem é um termo para descrever um ambiente de computação baseado em uma imensa rede de servidores, sejam estes virtuais ou físicos, oferecendo um conjunto de recursos como capacidade de processamento, armazenamento e conectividade, sendo então um estágio mais evoluído do conceito de virtualização (Taurion, 2009). Para Fano e Gershman (2002), uma definição para a virtualização consiste da execução de softwares em servidores, com disponibilidade de acesso externo. Portanto, pode-se afirmar que a computação em nuvem é o sucessor da virtualização, com novas estratégias em sua aplicação.

Complementarmente, a computação em nuvem é um grande reservatório de recursos virtualizados facilmente utilizáveis e acessíveis. Esses recursos podem ser dinamicamente reconfigurados para ajustar a carga (escala) variável do sistema, permitindo também um uso ótimo dos recursos. Esse reservatório de recursos é geralmente explorado por um modelo *pay-per-use* (pagar para usar) no qual as garantias são oferecidas por um Provedor de Infraestrutura e Serviço (Vaquero *et al.*, 2008).

Segundo Taurion (2009), no início da década de 2000, empresas como Google e Amazon, de forma independente, criaram imensos parques computacionais para oferecerem comercialmente os serviços em nuvem. Podendo ser adquirida de forma comercial, a computação em nuvem teve seu uso alavancado em diversos setores, especialmente no corporativo.

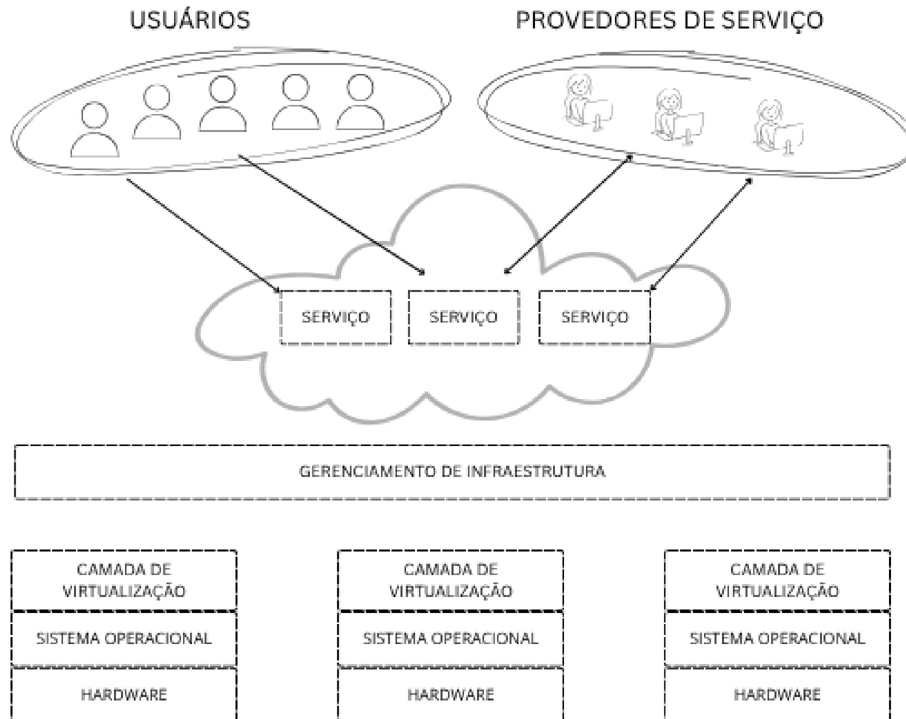
Para Fano e Gershman (2002), as empresas que utilizam computação em nuvem acabam adquirindo uma redução de custos associados ao consumo de energia elétrica, redução ou aumento de recursos computacionais sem necessitar de expansão física e diminuição dos

custos operacionais da Tecnologia da Informação (TI). Nesse sentido, há grandes benefícios na adoção da computação em nuvem por parte de empresas que visam reduzir custos operacionais e computacionais.

Nesse sentido, de forma generalizada a computação em nuvem pode ser definida como a entrega de recursos de TI por meio da Internet. Segundo a AWS (2025), o armazenamento em nuvem é fornecido por um provedor de serviços em nuvem, que possui e opera o armazenamento de dados em grandes datacenters em diversas localidades do mundo. Os provedores de armazenamento em nuvem gerenciam os espaços para a alocação de datacenters, a capacidade e a segurança, além de manter os dados acessíveis pela Internet em um modelo de pagamento conforme o uso.

A Figura 4 demonstra a arquitetura da computação em nuvem. Ao lado esquerdo, há os clientes finais que acessam os serviços da nuvem. Ao lado direito, há os provedores de serviços, que são as empresas que criam e gerenciam os serviços em nuvem. No centro, há a nuvem e seus serviços como hospedagem de *site* e bancos de dados. As setas indicam que os usuários consomem, enquanto os provedores gerenciam e fornecem. Finalmente, temos a camada física responsável por gerenciar os recursos como o *hardware*, o sistema operacional e a camada de virtualização.

Figura 4 – Demonstração de uma arquitetura de computação em nuvem



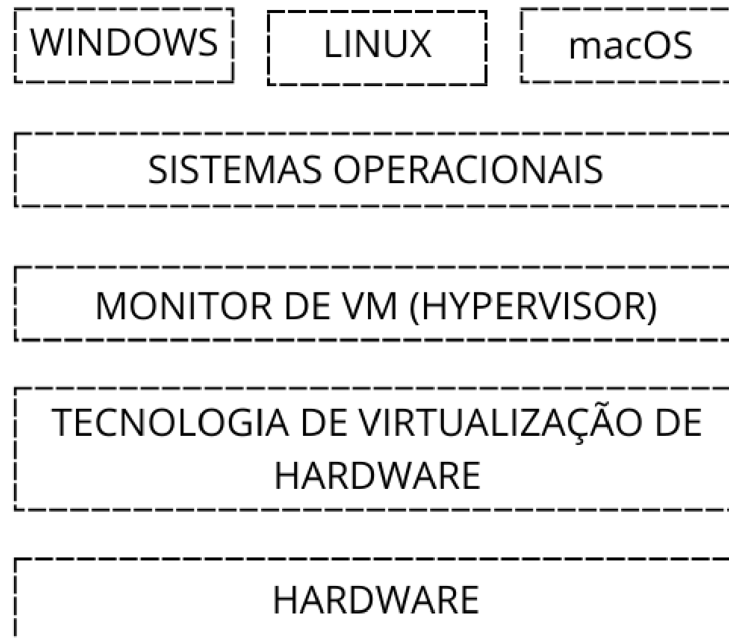
Fonte: Adaptado de (Vaquero *et al.*, 2008)

De modo geral, a camada física é o alicerce da infraestrutura da computação em nuvem. Ela é composta por servidores, discos rígidos, processadores, memória, placas de rede e outros dispositivos físicos. Sem essa base, nenhum outro nível da nuvem existiria (IBM, 2024a). Outro grande fundamento da computação em nuvem é a virtualização, segundo Kreutz *et al.* (2009), tecnologias de virtualização existem tanto em nível de *software* quanto em nível de *hardware*. Na virtualização em nível de *hardware*, há a possibilidade de gerenciar simultaneamente diferentes tipos de sistemas operacionais.

Nesse sentido, a virtualização trabalha criando uma camada de abstração sobre o *hardware* de um computador. De modo que, um único recurso de *hardware* possa ser utilizado em vários computadores virtuais, também chamado de Virtual Machine (VM), onde cada VM se comporta de maneira independente em seus respectivos modos de execução (IBM, 2024a). A Figura 5 demonstra uma virtualização em *hardware*, onde o *hardware* é a base física, e a tecnologia de virtualização em *hardware* são os recursos embutidos que suportam a virtualização como o Intel VT-x e o AMD-v, posteriormente, o *Hypervisor* é o responsável por gerenciar os recursos físicos como processamento e memória. Por fim, cada VM possui seu sistema

operacional independente.

Figura 5 – Virtualização com suporte em *hardware*



Fonte: Adaptado de (Kreutz *et al.*, 2009)

Dessa forma, com a utilização de computação em nuvem, uma empresa pode se abstrair de uma camada de complexidade demandada pela infraestrutura computacional e se concentrar em outros aspectos para expansão de serviços (Taurion, 2009). Dessa forma, é evidente que desde o estudo de John McCarthy sobre computação compartilhada até a década atual, a computação em nuvem contribuiu de forma crescente para a expansão da tecnologia.

2.3 Camadas da arquitetura de computação em nuvem orientada a serviços

Através do conceito de disponibilização de software pela *web*, também conhecido como Application Service Provisioning (ASP), a computação em nuvem se tornou bastante utilizada, visto que permitiu aos desenvolvedores construir aplicações sem a necessidade de se preocupar com a infraestrutura para hospedagem. Segundo Iyer e Henderson (2012), no modelo ASP, provedores de serviços implantam, gerenciam e hospedam remotamente aplicações de software empacotadas por meio de servidores centralizados, entregando esses serviços às empresas sob um contrato de aluguel ou assinatura. As empresas utilizam apenas o que precisam e pagam apenas pelo que usam. O objetivo é reduzir a complexidade, minimizar os custos e

melhorar a agilidade organizacional.

Para Mell e Grance (2011), a computação em nuvem é um modelo que possibilita o acesso conveniente e sob demanda via rede a um conjunto compartilhado de recursos. Ainda segundo os autores, o modelo de computação em nuvem é composto por cinco características essenciais, sendo elas: autoatendimento sob demanada, acesso amplo à rede, agrupamento de recursos, elasticidade rápida e serviço mensurável.

Além do mais, para Mell e Grance (2011), a computação em nuvem é definida por três modelos de serviço e quatro modelos de implantação. Os três modelos de serviços são Software as a Service (SaaS), Platform as a Service (PaaS) e Infrastructure as a Service (IaaS). Ademais, para Taurion (2009), há também o Backup as a Service (BaaS) que trata-se de uma forma de SaaS. De modo geral, os quatro modelos trabalham de formas distintas mas com o mesmo objetivo de reduzir a complexidade sobre a infraestrutura.

O modelo SaaS entrega um software pronto para uso. Segundo Mell e Grance (2011), a capacidade fornecida ao consumidor é utilizar as aplicações do provedor, no qual, as aplicações são acessíveis através da internet partir de diversos dispositivos como navegador. A aplicação SaaS é ideal para usuários finais, que exigem uma maior praticidade.

Diferentemente do SaaS, no modelo PaaS o consumidor implanta aplicações criadas por ele próprio, através da utilização de linguagens de programação e ferramentas suportadas pelo provedor. O consumidor não controla a infraestrutura de nuvem, como servidores e redes, mas possui total controle sobre as aplicações implantadas (Badger *et al.*, 2011). Esse modelo é mais utilizado por desenvolvedores, visto que é possível estabelecer configurações das aplicações hospedadas.

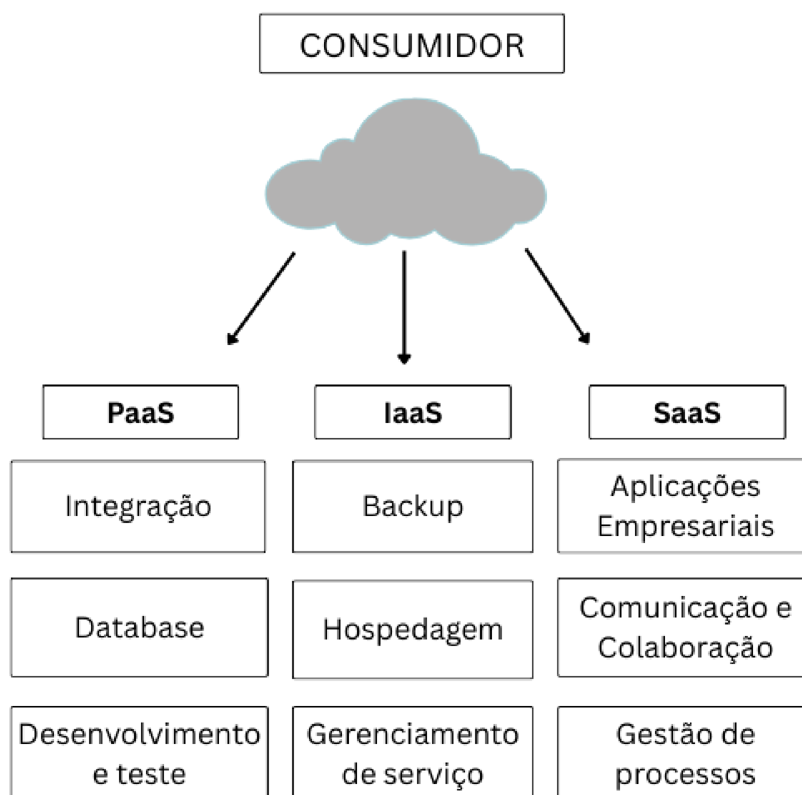
Por outro lado, para Badger *et al.* (2011) o IaaS entrega recursos de infraestrutura física e virtual via internet através do provedor. De forma que, no lugar de comprar servidores ou *hardware*, o consumidor aluga recursos conforme sua necessidade. Nesse modelo, há uma maior utilização do *pay-per-use*, no qual o consumidor paga apenas pelos recursos que consome, especialmente em recursos de processamento e armazenamento.

Visto como um enorme problema para grande parte das empresas, o armazenamento para *backup* apesar de importante, muitas vezes é negligenciado. Segundo o NetApp (2023), 60% das empresas que perdem dados críticos fecham dentro de 6 meses após a perda. Uma solução é o serviço de nuvem BaaS, que é um método de armazenamento externo dos dados em nuvem de forma segura. Dessa forma, a oferta do serviço BaaS é vista como uma ótima ferramenta para

esse gerenciamento. Segundo Taurion (2009), empresas contratam o BaaS devido ao fato do *backup* ser um processo complexo, intensivo e necessário.

A Figura 6 representa os serviços da computação em nuvem. No topo da Figura está o consumidor, responsável por utilizar os serviços da nuvem. No centro, há a nuvem representando toda a infraestrutura da computação em nuvem, posteriormente, há alguns dos principais serviços ofertados pelo PaaS, IaaS e SaaS. Desse modo, cada serviço em nuvem é capaz de oferecer um nível diferente de abstração e responsabilidade do consumidor.

Figura 6 – Serviços ofertados na computação em nuvem



Fonte: Adaptado de (Badger *et al.*, 2011)

Além dos modelos de serviços apresentados anteriormente, a nuvem possui seus modelos de implantação. Segundo Mishra (2014), há vários modelos de implantação de nuvem, mas quatro são os mais utilizados, sendo eles: nuvem pública, privada, híbrida e comunitária. Esses modelos visam garantir a qualidade do serviço.

Segundo Badger *et al.* (2011), a nuvem privada é operada e controlada por uma organização que pode consistir em várias unidades de negócio. Nesse cenário, uma única organização possui toda uma infraestrutara de *datacenters* dedicados.

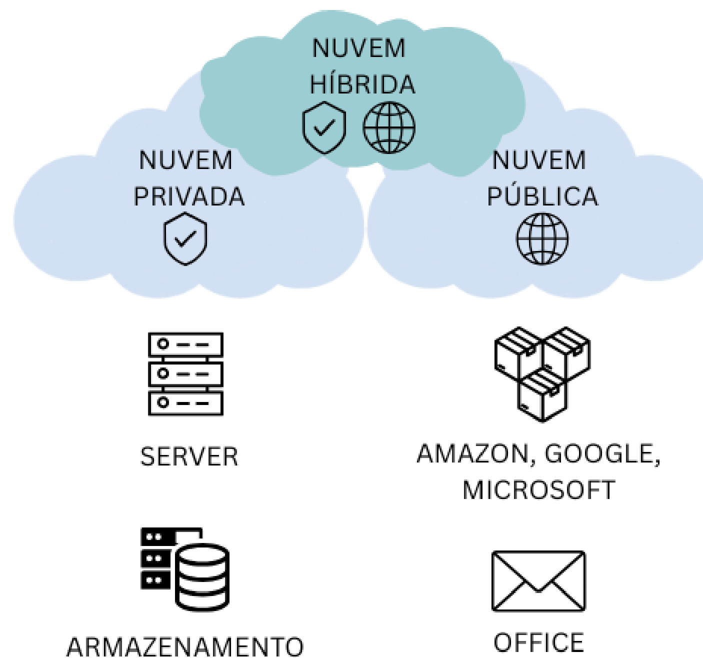
Por outro lado, para Badger *et al.* (2011), na nuvem pública toda a infraestrutura

da nuvem é disponibilizada para uso aberto pelo público. Isso é feito de modo que possa ser gerenciada por uma empresa, instituição ou uma combinação de organizações. Dessa forma, todos os recursos são fornecidos por terceiros.

Finalmente, definida por Badger *et al.* (2011), a nuvem híbrida é uma combinação entre a pública e privada, na qual dados podem ser compartilhados entre elas, enquanto a nuvem comunitária é uma nuvem compartilhada entre organizações com interesses em comum.

A Figura 7 representa os três principais modelos de implantação da computação em nuvem: nuvem privada, pública e híbrida. A nuvem privada oferece maior controle e segurança sobre os dados. A nuvem pública é fornecida por empresas como Amazon, sendo acessível ao público em geral, com serviços como armazenamento e aplicativos *online* como por exemplo o *Office*. Por outro lado, a nuvem híbrida combina os dois modelos anteriores, permitindo maior flexibilidade ao integrar recursos locais com serviços da nuvem pública.

Figura 7 – Demonstração entre nuvem privada, pública e privada



Fonte: Adaptado de (Badger *et al.*, 2011)

O presente estudo utiliza, na camada de Plataforma como Serviço (*PaaS*), dois dos serviços mais consolidados no mercado: o *Azure App Service*, da Microsoft, com maior foco em aplicações *web* e API, e o *AWS Elastic Beanstalk*, da Amazon, com suporte mais amplo e flexível. Ambos os serviços permitiram a implantação de aplicações sem a necessidade de gerenciar a infraestrutura subjacente, facilitando o processo de desenvolvimento, escalabilidade

e monitoramento. A utilização de diferentes plataformas teve como objetivo comparar a experiência de implantação e os recursos oferecidos por cada provedor, considerando aspectos como desempenho, facilidade de uso e implantação dos serviços, além dos custos em cada ambiente.

3 TRABALHOS RELACIONADOS

Nesta seção, serão abordados os trabalhos relacionados que se assemelham com a ideia de pesquisa deste trabalho.

3.1 Desvendando as Nuvens: Uma análise comparativa de desempenho de aplicações *Serverless* e Containers em provedores de computação Em nuvem

Em Filho (2024), é realizada uma análise comparativa entre aplicações implantadas nas arquiteturas *serverless* e em *containers*, utilizando provedores de computação em nuvem. O autor destaca o crescimento da adoção dessas tecnologias no mercado e aponta a necessidade de compreender os impactos de desempenho e custo que cada abordagem pode trazer para o desenvolvimento de aplicações modernas.

Para a condução da análise, foram desenvolvidas aplicações equivalentes em ambas as arquiteturas, com o objetivo de medir métricas como tempo de execução e custo financeiro em diferentes provedores de nuvem. O estudo utilizou testes de desempenho nos ambientes de nuvens selecionados.

Como conclusão, o autor observa que, os custos como AWS *lambda* é mais econômico em comparação com Azure *Function* e Google *Function*.

Dessa forma, o trabalho de Filho (2024), se assemelha ao presente trabalho por também se tratar de uma análise comparativa baseada em métricas de desempenho em ambientes de nuvem. No entanto, o presente trabalho se diferencia por focar na comparação entre linguagens de programação *PHP* e *.NET* em vez de verificar apenas custos, buscando avaliar qual delas apresenta melhor desempenho em termos de tempo de resposta e consumo de recursos em diferentes cenários de ambientação em nuvem.

3.2 Computação em nuvem: Uma análise comparativa das plataformas disponibilizadas por Amazon, Google e Microsoft

No trabalho de Araujo (2019), é realizada uma análise comparativa nos ambientes da Amazon Web Services (AWS), *Google Cloud Platform* e Azure. O autor destaca o crescimento na adoção desses serviços em diversos setores, ressaltando a importância da avaliação de desempenho para selecionar corretamente uma plataforma em contextos reais de uso.

Para executar essa análise, o autor realizou análises nos três principais ambientes

de nuvens, sendo o Azure, Google e Amazon, e as métricas utilizadas nas comparações estão associadas a custos, infraestrutura e tratamento de dados.

Como resultado, o autor identificou que todas as plataformas possuem bom desempenho geral, mas com particularidades em contextos específicos. A AWS se destaca pela infraestrutura, enquanto o *Google Cloud Platform* se destaca por conexões de baixa latência, destacando-se pela relação custo-benefício adequada para aplicações de pequeno e médio porte. Segundo o autor, a Azure se destaca pela quantidade de serviços ofertados e pelo acervo de tutoriais disponíveis.

Dessa forma, o trabalho de Araujo (2019) assemelha-se ao presente estudo ao buscar avaliar ambientes de computação em nuvem. Contudo, difere metodologicamente, já que o presente trabalho propõe a construção e análise de aplicações específicas em PHP e .NET, avaliando o desempenho dessas linguagens em diferentes plataformas de nuvem, com objetivo de determinar qual linguagem oferece melhor desempenho em termos de tempo de resposta e consumo eficiente de recursos.

3.3 *An Empirical Study on the Impact of Programming Languages on the Performance of Open-source Serverless Platforms*

No trabalho de Ataie *et al.* (2024), é conduzido um estudo empírico sobre o impacto das linguagens de programação no desempenho de plataformas *serverless* de código aberto. Os autores investigam três plataformas populares — *OpenFaaS, Nuclio e Fission* — implantadas em clusters Kubernetes, com o objetivo de avaliar como as linguagens Python e Node.js influenciam métricas de desempenho como tempo de resposta, *throughput* e volume de dados transferidos.

Para realizar a análise, foram desenvolvidas funções equivalentes em *Python* e *Node.js*, executadas sob diferentes níveis de concorrência utilizando as ferramentas *Hey* e *Wrk*, ambas tendo o propósito de gerar tráfego HTTP com o objetivo de medir o desempenho e a capacidade de resposta de servidores sob diferentes cargas, isto é, funcionando como *Load Generators*. Os experimentos foram conduzidos na infraestrutura do *CloudLab*, uma ferramenta projetada para experimentos em sistemas de computação em nuvem, na qual permite uma avaliação controlada e reprodutível do desempenho das plataformas em diferentes cenários de uso.

Os resultados indicam que, na plataforma *OpenFaaS*, funções escritas em *Python* apresentam melhor desempenho em termos de tempo de resposta e *throughput*. Por outro

lado, na plataforma *Fission*, o *Node.js* supera o *Python*, especialmente em cenários com alta concorrência. Além disso, a plataforma *Nuclio* demonstrou desempenho superior de forma geral, independentemente da linguagem utilizada, destacando-se como a mais eficiente entre as avaliadas.

Este estudo se assemelha ao presente trabalho ao investigar o impacto das linguagens de programação no desempenho de aplicações em ambientes de computação em nuvem. No entanto, diferencia-se ao focar em plataformas *serverless* de código aberto e nas linguagens *Python* e *Node.js*, enquanto o presente trabalho propõe a análise de aplicações desenvolvidas em *PHP* e *.NET*. Desse modo, o estudo é de suma importância por entregar noções de métricas para avaliações de desempenho nos ambientes de nuvem.

3.4 Comparação entre os trabalhos

O presente trabalho visa analisar o desempenho de aplicações escritas em *.NET* e *PHP* em ambientes de nuvem. Para isso, foram selecionados três trabalhos que correspondem ao objetivo do presente trabalho. No Quadro 1, é possível observar todos os trabalhos relacionados nesta seção, e a diferença e relação entre eles com base nos seguintes parâmetros: Ambiente de Nuvem, Linguagem, Métricas e Tipo de Aplicação.

Quadro 1 – Comparação entre os trabalhos.

Trabalho	**Ambiente de Nuvem**	**Linguagem(s)**	**Métricas**	**Tipo de Aplicação**
(Filho, 2024)	AWS, Azure e GCP	–	Custo	Aplicação web em *Javascript*
(Araujo, 2019)	AWS, Azure e GCP	–	Infraestrutura, presença por região e serviços ofertados	-
(Ataie *et al.*, 2024)	Kubernetes no CloudLab	Python, Node.js	Tempo de resposta, *throughput*, volume de dados transferido	Funções serverless utilizando OpenFaaS, Nuclio e Fission
Presente trabalho	AWS, Azure	PHP, .NET	Tempo de resposta, *throughtput*, média e mediana de respostas	APIs REST desenvolvidas em PHP e .NET

Fonte: Elaborado pelo autor.

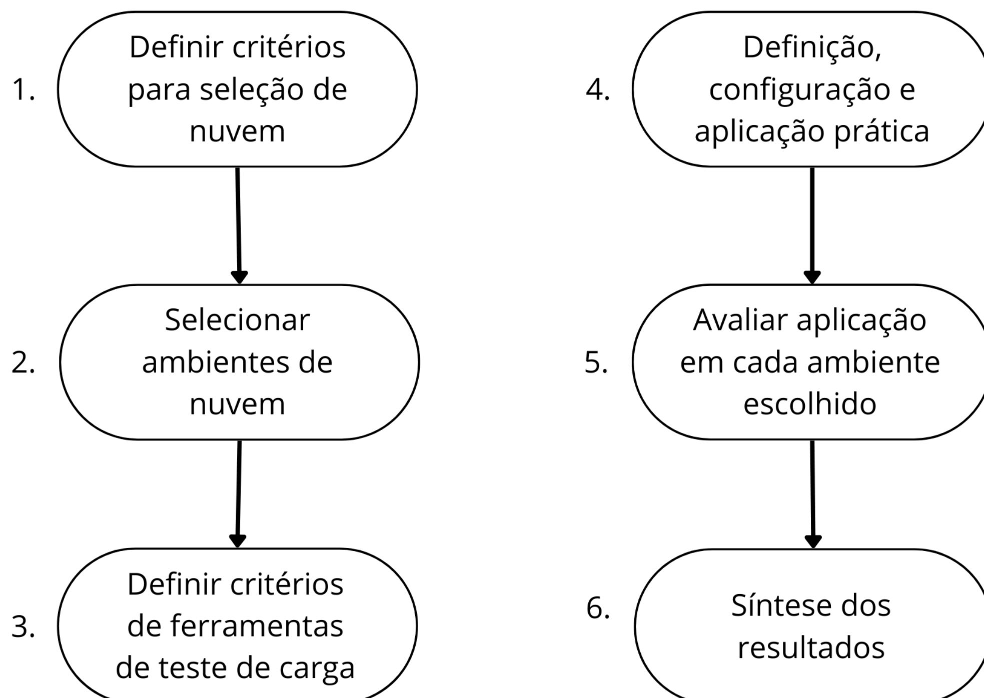
O primeiro critério avalia os ambientes de nuvem utilizados em cada respectivo trabalho. O segundo critério analisa se houve uma linguagem utilizada no contexto da avaliação das nuvens. O terceiro critério avalia as métricas extraídas nos estudos, e o quarto e último critério avalia se houve algum tipo de aplicação desenvolvida para reforçar a análise das métricas.

4 METODOLOGIA

Neste capítulo, serão pontuados os procedimentos metodológicos que serão executados no decorrer do presente trabalho para se chegar ao objetivo principal. A metodologia utilizada no presente estudo é uma forma resumida da metodologia abordada por (Jain, 1991).

Dessa forma, os procedimentos metodológicos do presente estudo servem como um guia de orientação sobre o passo a passo da realização da pesquisa, sendo eles: definir critérios para seleção de nuvem, selecionar ambientes de nuvem, definir critérios de ferramentas de teste de carga, definição, configuração e aplicação prática, avaliar aplicação em cada ambiente escolhido e síntese dos resultados. Todas essas etapas são ilustradas na Figura 8 abaixo e detalhadas posteriormente.

Figura 8 – Etapas metodológicas do presente trabalho



Fonte: Elaborado pelo autor

4.1 Definir critérios para seleção de nuvem

Essa é a etapa inicial desse estudo e ela consiste em definir os critérios para seleção das nuvens que serão estudadas no presente trabalho. A revisão bibliográfica realizada nesta

etapa também busca identificar as ferramentas de cargas utilizadas por trabalhos acadêmicos que analisam o desempenho de nuvens. Esses critérios foram definidos baseados em uma revisão bibliográfica dos trabalhos relacionados e outros trabalhos que coincidem com o tema desse estudo.

O levantamento bibliográfico foi realizado por meio de três principais cadeias de buscas em língua inglesa, aplicadas nas bases *IEEE Xplore* e *Google Scholar*, no período de 2014 a 2025: *("cloud providers" OR "cloud platform") AND ("benchmark" OR "performance" OR "Comparative analysis")* e *("resource consumption" OR "CPU usage" OR "memory usage") AND ("AWS" OR "Azure" OR "cloud")* e *("Cloud computing" OR "cloud providers") AND ("language comparison" OR "programming languages") AND ("performance" OR "benchmark").*

Em cada base, inicialmente foram retornados, aproximadamente, 16.900 artigos no *IEEE Xplore*, 17.200 no *Google Scholar* para essas buscas combinadas. Realizada as buscas, foram aplicados os critérios de inclusão para revisão, tais como: revisão de alta performance, cargas de trabalhos escaláveis e importância de *benchmarking* em nuvem.

A partir do total de artigos retornados em cada base, foi realizado um processo de triagem em múltiplas etapas. Inicialmente, foram eliminados trabalhos duplicados entre as bases e artigos que não apresentavam aderência ao tema, com base na leitura dos títulos e resumos. Em seguida, foram aplicados critérios de disponibilidade de acesso ao texto completo e aderência aos temas de interesse — como *benchmarking*, desempenho em nuvem e comparação entre arquiteturas ou linguagens. Após essa filtragem, restaram um número reduzido de artigos para revisão completa. O Quadro 2 traz a quantidade de artigos revisados para determinada cadeia de busca. Ademais, o Quadro também traz a quantidade de artigos selecionados e que se relacionam com a temática do presente estudo.

Quadro 2 – Chaves de busca e artigos revisados e selecionados baseado em critérios.

Chaves de busca	Art. Revisados	Art. Selecionados
"cloud providers" OR "cloud platform") AND ("benchmark" OR "performance" OR "Comparative analysis")	9	3
("resource consumption" OR "CPU usage" OR "memory usage") AND ("AWS" OR "Azure" OR "cloud")	8	2
("Cloud computing" OR "cloud providers") AND ("language comparison" OR "programming languages") AND ("performance" OR "benchmark")	5	2

Fonte: Elaborado pelo autor

4.2 Selecionar ambientes de nuvem

Nesta seção, é utilizada a análise da seção anterior para a realização da seleção dos ambientes de nuvem em que as aplicações serão hospedadas e posteriormente testadas.

Nesse sentido, conforme utilizado por Araujo (2019), também foi definida a utilização do website *Stack Overflow*, um website americano criado em 2008 com o objetivo de fornecer meios para usuários, especialmente do universo da tecnologia, retirarem dúvidas através de perguntas e respostas na plataforma. Dessa forma, baseando-se nas questões dos usuários, podemos analisar a popularidade de alguns ambientes de nuvem com base em dados de entrevistas realizadas pela própria plataforma *Stack Overflow* sobre as preferências de cada entrevistado.

4.3 Definir critérios de ferramentas de teste de carga

Nesta etapa, serão definidos os critérios para a seleção das ferramentas de cargas utilizadas para avaliar as aplicações. Serão considerados quatro critérios, sendo eles: Protocolos suportados, métricas coletadas, capacidade de escalabilidade e comunidade. Nesse sentido, serão avaliadas quatro principais ferramentas: *Wrk*, *Hey*, *JMeter* e *K6*. Na qual as duas primeiras são ferramentas leves e menos populares, utilizadas a base de linha de comando, e as duas últimas são ferramentas mais completas, com o *Apache* JMeter possuindo Interface Gráfica (GUI) e o *K6* baseando-se em *scripts* em JavaScript com a utilização da biblioteca da ferramenta para a elaboração de testes de cargas altamente configurados. Dessa forma, avaliando-se critérios de popularidade, compatibilidade e métricas, apenas duas serão selecionadas para análise de cargas

das aplicações definidas neste trabalho.

4.4 Definição, configuração e aplicação prática

Nesta etapa foi feita a definição da aplicação prática, como o diagrama de classes, a escrita de código em ambas linguagens PHP e .NET, e *deploy* e configuração das aplicações nos ambientes de nuvem selecionados.

4.5 Avaliar aplicação em cada ambiente escolhido

Nesta etapa, o objetivo foi realizar uma avaliação das aplicações nos ambientes definidos, observando-se critérios de configuração das aplicações e custos, além de métricas, tais como: média, mediana e *throughput*. Nesse sentido, as aplicações foram hospedadas nos ambientes, observando-se facilidade de configuração, *deploy* e custos, e foram sujeitas a testes de cargas para avaliação de desempenho, observando-se as métricas.

4.6 Síntese dos resultados

Finalmente, nesta etapa visa foi feita uma análise e uma síntese dos resultados baseadas na elaboração dos testes e métricas extraídas nas seções 4.4 e 4.5.

Essa etapa permitiu identificar o comportamento de cada aplicação nos diferentes ambientes de nuvem e suas respectivas diferenças. A identificação desses comportamentos e extração dessas informações foram fundamentais para a conclusão deste trabalho.

5 RESULTADOS PRELIMINARES

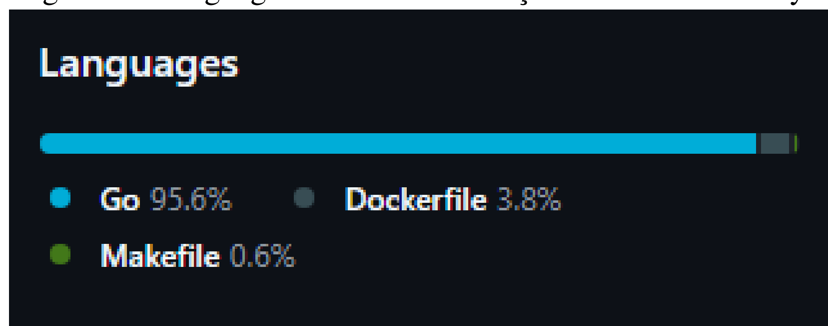
Esse capítulo tem como objetivo demonstrar os resultados preliminares elaborados pelo autor, contemplando a instalação e testes das ferramentas de testes de cargas, expondo análises e o passo a passo.

5.1 Instalação das ferramentas de testes de cargas

5.1.1 Instalação e teste da ferramenta Hey

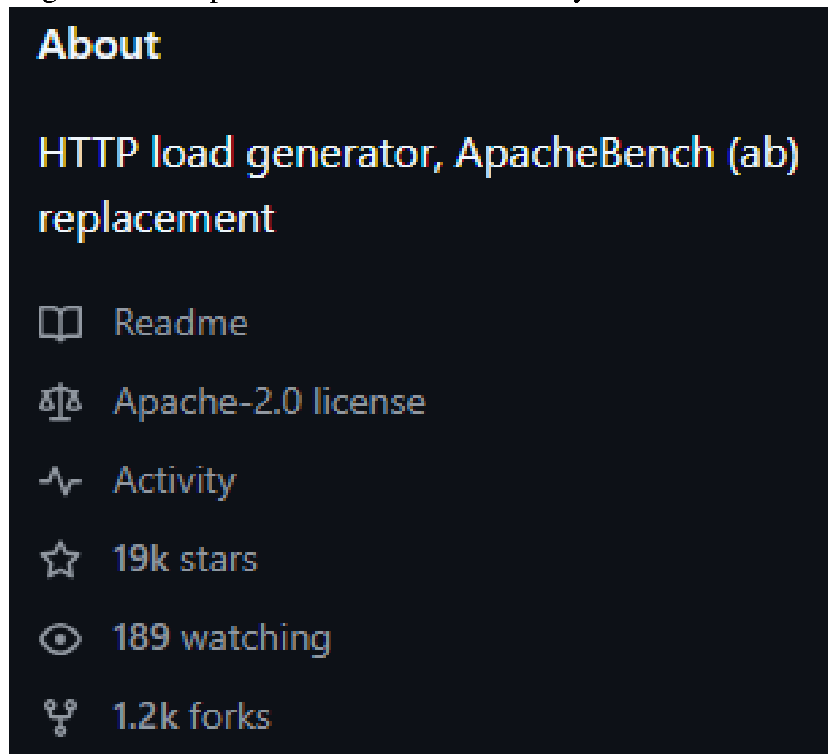
A documentação contida no repositório da ferramenta Hey, disponibilizada por (Dogan, 2025), embora contenha poucas instruções sobre a instalação, o código fonte revela a escrita da ferramenta na linguagem de programação Go. A Figura 9 demonstra a porcentagem da linguagem utilizada na criação da ferramenta conforme o repositório no GitHub. Por outro lado, a Figura 10 demonstra a popularidade da ferramenta Hey com cerca de 19.000 estrelas.

Figura 9 – Linguagem utilizada na criação da ferramenta Hey



Fonte: Adaptado de (Dogan, 2025)

Figura 10 – Popularidade da ferramenta Hey no GitHub



Fonte: Adaptado de (Dogan, 2025)

Nesse cenário, para efeito de compilação e instalação da ferramenta Hey, a instalação da linguagem de programação Go faz-se necessária. Nesse sentido, a documentação oficial disponibilizada por (Google, 2025b) foi utilizada para o *download* do *.msi*. Após a instalação a Figura 11 valida a versão da linguagem instalada.

Figura 11 – Versão da linguagem Go

```
luizh@LAPTOP-6AAL6SGL MINGW64 ~
$ go version
go version go1.24.4 windows/amd64
```

Fonte: Elaborado pelo autor

Dessa forma, após a instalação da linguagem de programação, a instalação da ferramenta Hey pode ser feita apontando para o repositório (Dogan, 2025). A Figura 12 demonstra o comando utilizado para a instalação via terminal.

Figura 12 – Instalação da ferramenta Hey

```
luizh@LAPTOP-6AAL6SGL MINGW64 ~
$ go install github.com/rakyll/hey@latest
go: downloading github.com/rakyll/hey v0.1.4
go: downloading golang.org/x/net v0.0.0-20181017193950-04a2e542c03f
go: downloading golang.org/x/text v0.3.0
```

Fonte: Elaborado pelo autor

Finalmente, após a instalação da ferramenta, os comandos podem ser utilizados para testes de cargas em APIs com protocolos HTTP e HTTPS. Dessa forma, o Quadro 3 aponta os parâmetros que podem acompanhar o comando para testes de cargas na ferramenta Hey.

Quadro 3 – Principais parâmetros de configuração da ferramenta Hey.

Parâmetro	**Descrição**
-n	Número total de requisições a serem enviadas. Padrão: 200.
-c	Número de usuários a serem executados simultaneamente. O total de requisições não pode ser menor que o nível de concorrência. Padrão: 50.
-q	Limite de taxa (QPS - consultas por segundo) por *workers*. Padrão: sem limite.
-z	Duração total do teste. Ao atingir o tempo especificado, o teste é encerrado. Exemplo: -z 10s ou -z 3m.
-o	Tipo de saída dos resultados. Padrão: resumo no terminal. Alternativa: “csv” para exportação de métricas em formato CSV.
-m	Método HTTP (GET, POST, PUT, DELETE, HEAD, OPTIONS).
-H	Cabeçalho HTTP personalizado. Pode ser repetido para múltiplos cabeçalhos. Exemplo: -H “Accept: text/html”.
-t	Timeout de cada requisição em segundos. Padrão: 20 segundos.
-A	Cabeçalho HTTP Accept.
-d	Corpo da requisição HTTP.
-D	Corpo da requisição HTTP a partir de um arquivo. Exemplo: ./file.txt
-T	Content-Type da requisição. Padrão: “text/html”.
-a	Autenticação básica no formato usuário:senha.
-x	Endereço de Proxy HTTP no formato host:port.
-h2	Habilita o protocolo HTTP/2.
-host	Define o cabeçalho HTTP Host.
-disable-compression	Desativa a compressão HTTP.
-disable-keepalive	Desativa o Keep-Alive, impedindo a reutilização de conexões TCP entre requisições.
-disable-redirects	Impede o seguimento automático de redirecionamentos HTTP.
-cpus	Número de núcleos de CPU a serem utilizados. Padrão: número de núcleos disponíveis na máquina.

Fonte: Adaptado de (Dogan, 2025)

Para efeito de testes na ferramenta, um simples teste foi efetuado utilizando os parâmetros *-n* para indicar a quantidade de requisições e *-c* para indicar a quantidade de conexões. A Figura 13 demonstra o teste realizado utilizando a ferramenta Hey, com *-n* 500 e *-c* 50. Os resultados retornados pela ferramenta apontam dados interessantes como o sumário, indicando o tempo total para conclusão das requisições (2,34 segundos) e o tempo médio para processar cada requisição (0,1434 segundos). Dados de latência e quantidade de requisições com tempos

semelhantes também são apresentados no *Response Time*.

Figura 13 – Demonstração de teste na ferramenta Hey

```
luizh@LAPTOP-6AAL6SGL MINGW64 ~
$ hey -n 500 -c 50 https://jsonplaceholder.typicode.com/posts

Summary:
  Total:        2.3418 secs
  Slowest:      0.7170 secs
  Fastest:      0.0336 secs
  Average:      0.1434 secs
  Requests/sec: 213.5117


Response time histogram:
  0.034 [1]     |
  0.102 [199]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.170 [135]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.239 [113]   |■■■■■■■■■■■■■■■■■■■■■■■
  0.307 [28]    |■■■■■■
  0.375 [18]    |■■■■
  0.444 [1]     |
  0.512 [2]     |
  0.580 [1]     |
  0.649 [0]     |
  0.717 [2]     |


Latency distribution:
  10% in 0.0457 secs
  25% in 0.0623 secs
  50% in 0.1402 secs
  75% in 0.1949 secs
  90% in 0.2697 secs
  95% in 0.3005 secs
  99% in 0.5043 secs
```

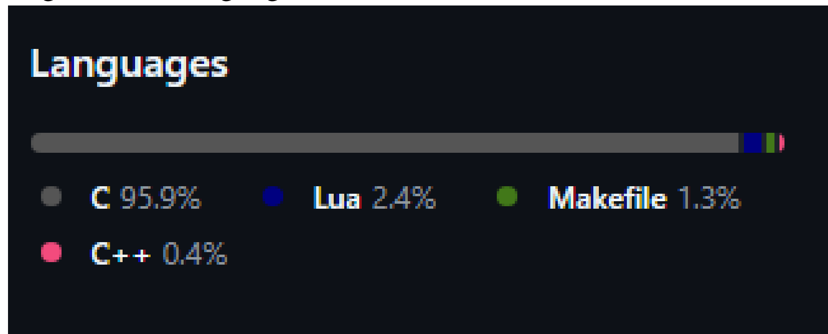
Fonte: Elaborado pelo autor

Dessa forma, mediante a instalação e configuração da ferramenta Hey, foi possível realizar um teste de carga inicial utilizando uma API pública de exemplo. O processo demonstrou a simplicidade e a eficiência da ferramenta para a geração de requisições simultâneas, possibilitando a coleta de métricas relevantes como tempo médio de resposta, taxa de requisições por segundo e distribuição de latência.

5.1.2 *Instalação e teste da ferramenta Wrk*

A documentação do Wrk pode ser encontrada no repositório no GitHub disponibilizada por (Glozer, 2025). A Figura 14 representa as linguagens utilizadas para a criação da ferramenta Wrk e os respectivos percentuais. Conforme a Figura 14, a linguagem C é a mais abrangente, em segundo lugar há a linguagem Lua que foi utilizada pelos criadores da ferramenta para disponibilizar *scripts* prontos para automatização.

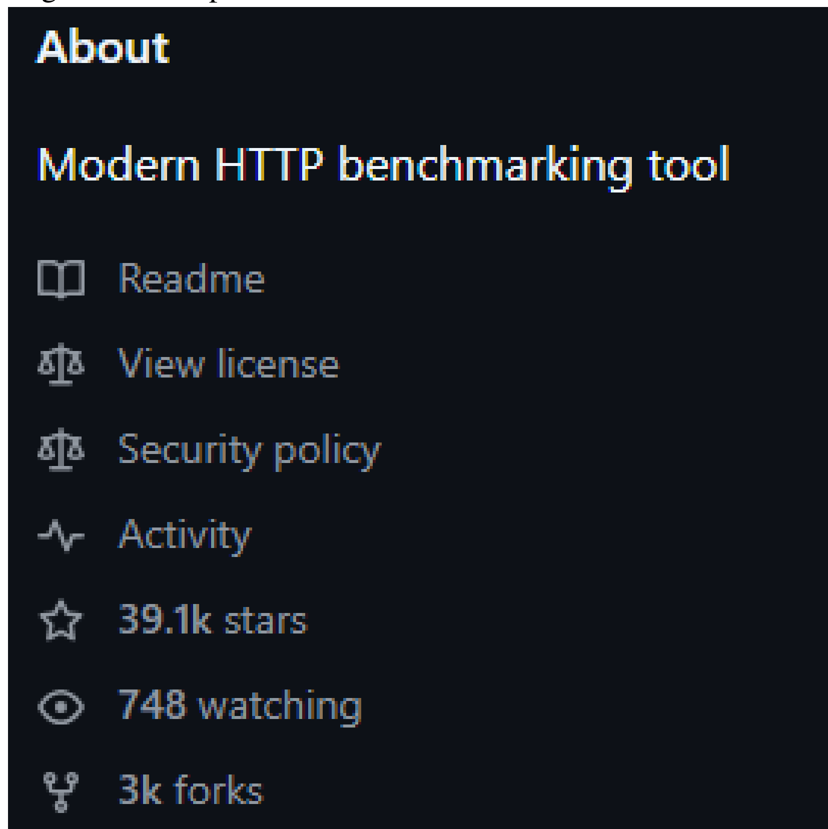
Figura 14 – Linguagens utilizadas na ferramenta Wrk



Fonte: Adaptado de (Glozer, 2025)

Em quesito de popularidade, o Wrk apresentou uma comunidade maior que a comunidade vista no Hey, com cerca de 39.100 estrelas concedidas e 3.000 *forks* no repositório oficial da ferramenta, demonstrando uma maior adesão por parte dos usuários. A Figura 15 demonstra as informações extraídas do repositório.

Figura 15 – Popularidade da ferramenta Wrk



Fonte: Adaptado de (Glozer, 2025)

Nesse sentido, para efeito de instalação e de testes, uma máquina virtual foi criada na Google Cloud para a execução do Wrk. A Figura 16 demonstra os detalhes da máquina virtual, uma *e2-medium*, utilizada para instalação e execução de um simples teste.

Figura 16 – Detalhes da máquina virtual criada na Google Cloud

Informações básicas

Nome	instance-20250625-010258
ID da instância	5966763802706244637
Descrição	Nenhum
Tipo	Instância
Status	Em execução

Configuração da máquina

Tipo de máquina	e2-medium (2 vCPUs, 4 GB de memória)
Plataforma de CPU	Intel Broadwell
Plataforma mínima de CPU	Nenhum
Arquitetura	x86/64

Fonte: Elaborado pelo autor

A criação da máquina virtual com um sistema de distribuição Linux foi necessária, visto que, a ferramenta Wrk não possui suporte para o sistema Windows. Nesse sentido, a instalação do Wrk no terminal Linux pode ser realizada através do comando *sudo apt install wrk*.

A utilização da ferramenta é realizada totalmente via linha de comando, nesse sentido, a documentação do Wrk disponibiliza alguns parâmetros essenciais que podem ser utilizados no momento de um teste. Os parâmetros estão demonstrados no Quadro 4.

Quadro 4 – Principais parâmetros da ferramenta Wrk.

Parâmetro	**Descrição**
`-c, -connections`	Número total de conexões HTTP a serem mantidas abertas. Cada *thread* manipula N = conexões/threads.
`-d, -duration`	Duração total do teste (ex: `2s`, `2m`, `2h`).
`-t, -threads`	Número total de *threads* a serem utilizadas.
`-s, -script`	Caminho para um script em LuaJIT para personalizar requisições (headers, corpo, etc.).
`-H, -header`	Header HTTP adicional para adicionar à requisição, ex: `"User-Agent: wrk"`.
`-latency`	Exibe estatísticas detalhadas de latência por *thread*.
`-timeout`	Define o tempo máximo de espera por uma resposta antes de registrar um *timeout*.

Fonte: Adaptado de (Glozer, 2025)

Embora o Wrk não forneça suporte para requisições HTTPs, se limitando apenas ao protocolo HTTP, a instalação e os testes foram efetuados para garantir o desempenho da ferramenta. Dessa forma, a Figura 17 demonstra a instalação da ferramenta e um teste realizado. Para realização do teste os parâmetros -t4, -c100 e -d15s utilizados no comando indicam, respectivamente, o uso de 4 threads, 100 conexões simultâneas e duração total de 15 segundos para o teste de carga.

Figura 17 – Instalação e teste na ferramenta Wrk

```
luizviphenrique@instance-20250625-010258:~$ sudo apt install wrk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wrk is already the newest version (4.1.0-3+b2).
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
luizviphenrique@instance-20250625-010258:~$ wrk -t4 -c100 -d15s http://httpbin.org/get
Running 15s test @ http://httpbin.org/get
  4 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   221.30ms  287.80ms   1.99s    85.88%
    Req/Sec   200.36     70.41   520.00     72.50%
  11984 requests in 15.02s, 4.87MB read
  Socket errors: connect 0, read 0, write 0, timeout 6
  Non-2xx or 3xx responses: 4
Requests/sec:    797.98
Transfer/sec:    331.95KB
luizviphenrique@instance-20250625-010258:~$
```

Fonte: Elaborado pelo autor

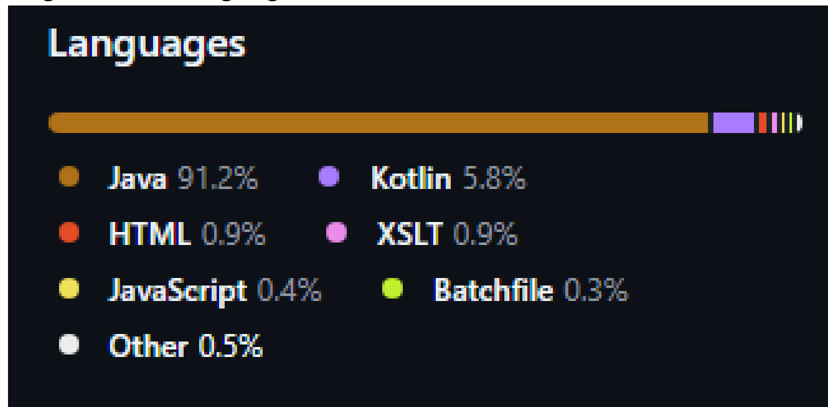
As métricas retornadas pelo Wrk, conforme a Figura 17 demonstra, apontam simplicidade porém uma poderosa capacidade de sobrecarga. De acordo com as métricas, em 15 segundos foram realizadas 11.984 requisições bem-sucedidas, com aproximadamente 800 requisições por segundo, e com cerca de 4,87 *megabytes* retornados pelo *endpoint* de teste. Ademais, 6 requisições tiveram erro de *timeout* e houveram 4 respostas com erro.

5.1.3 Instalação e teste da ferramenta JMeter

O *website* oficial do JMeter (JMETER, 2024) possui a documentação oficial da ferramenta. No repositório do GitHub da ferramenta notou-se que há uma vasta atividade de desenvolvedores e contribuintes, entrando de acordo com o que foi apresentado por (Lonn, 2020) na Figura 33 sobre a atividade das principais ferramentas de teste de carga.

A Figura 18 demonstra as principais linguagens utilizadas no desenvolvimento da ferramenta JMeter. Nesse sentido, a linguagem dominante é Java, no entanto, outras linguagens como Kotlin e Javascript são utilizadas para a criação de *plugins*.

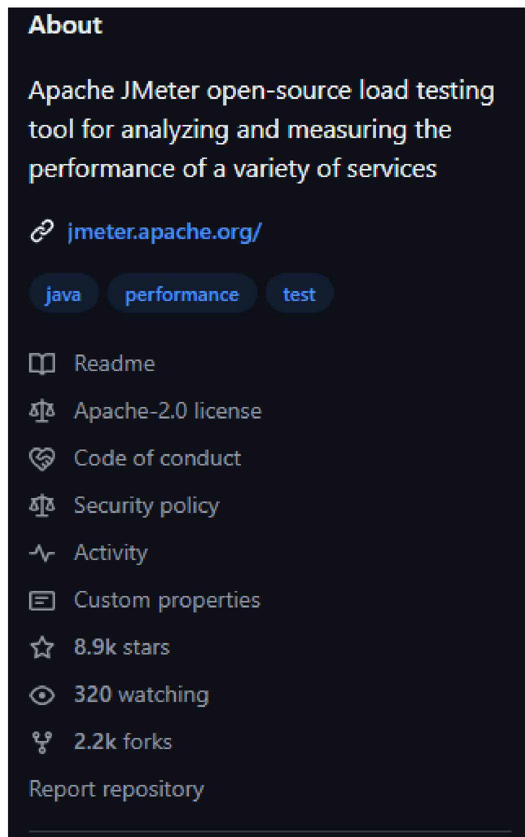
Figura 18 – Linguagens utilizadas na ferramenta JMeter



Fonte: Adaptado de (JMeter, 2024)

Sobre a popularidade do JMeter, o *website* oficial da ferramenta não disponibiliza a quantidade de downloads na ferramenta, no entanto, o repositório da ferramenta no GitHub indicou 81 contribuintes e cerca de 8.900 estrelas e 2.200 *forks*. A Figura 19 demonstra algumas das informações obtidas no repositório.

Figura 19 – Popularidade da ferramenta JMeter



Fonte: Adaptado de (JMeter, 2024)

Conforme visto anteriormente, a aplicação JMeter é escrita em Java e portanto, é

uma ferramenta multiplataforma. Dessa forma, pode ser utilizada sem problemas em sistemas Windows, Linux e macOS. A instalação é simples e pode ser realizada através do site oficial (JMETER, 2024), obtendo-se um arquivo do JMeter em *.zip* ou *.tgz*. O Quadro 5 elenca os principais requisitos para o *download* e execução da ferramenta.

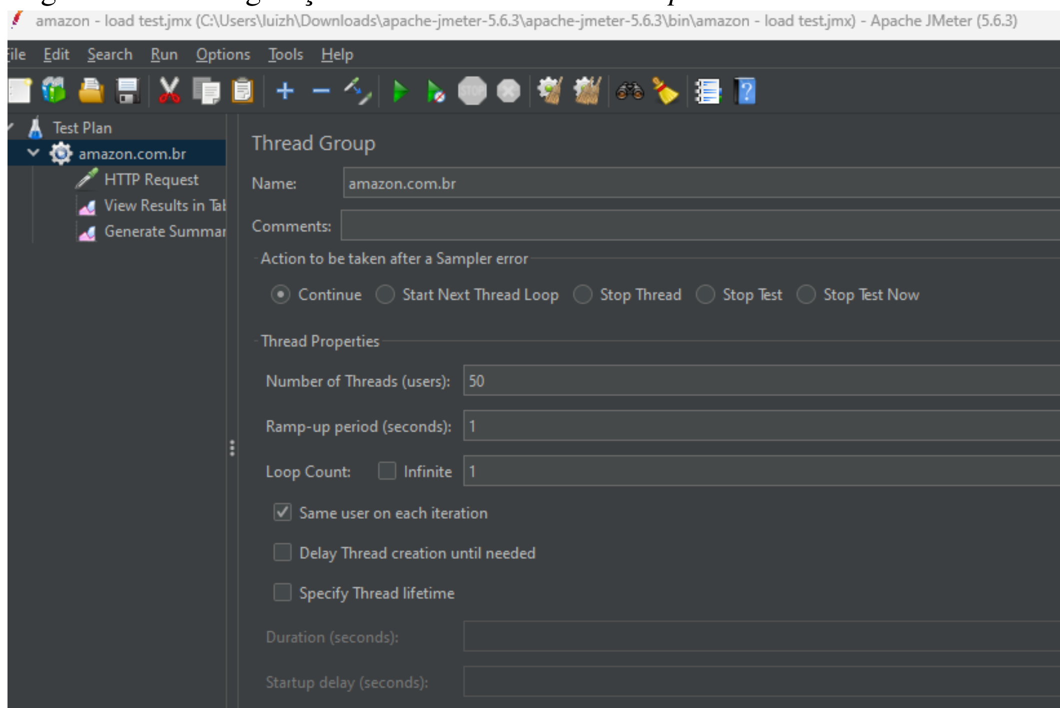
Quadro 5 – Requisitos para instalação do *Apache JMeter*.

Requisito	**Descrição**
Java (JDK ou JRE)	Necessário ter o Java instalado. Versões recomendadas: Java 8, 11 ou 17. Verificar com `java -version`.
Sistema Operacional	Compatível com Windows, Linux e macOS, por ser uma aplicação multiplataforma baseada em Java.
Interface Gráfica (GUI)	Recomendado para uso completo. Necessário um ambiente gráfico em sistemas Linux para abrir a interface.
Espaço em Disco	É necessário espaço suficiente para extrair o pacote (cerca de 150MB) e para armazenar resultados dos testes.

Fonte: Adaptado de (JMETER, 2024)

O teste no JMeter é algo mais simples visto que a ferramenta disponibiliza uma interface gráfica intuitiva. Para efeito de testes, uma simples configuração foi realizada na ferramenta para a execução de um teste de carga na URL da *amazon.com.br*. A Figura 20 ilustra a configuração de 50 usuários para a realização do teste de carga.

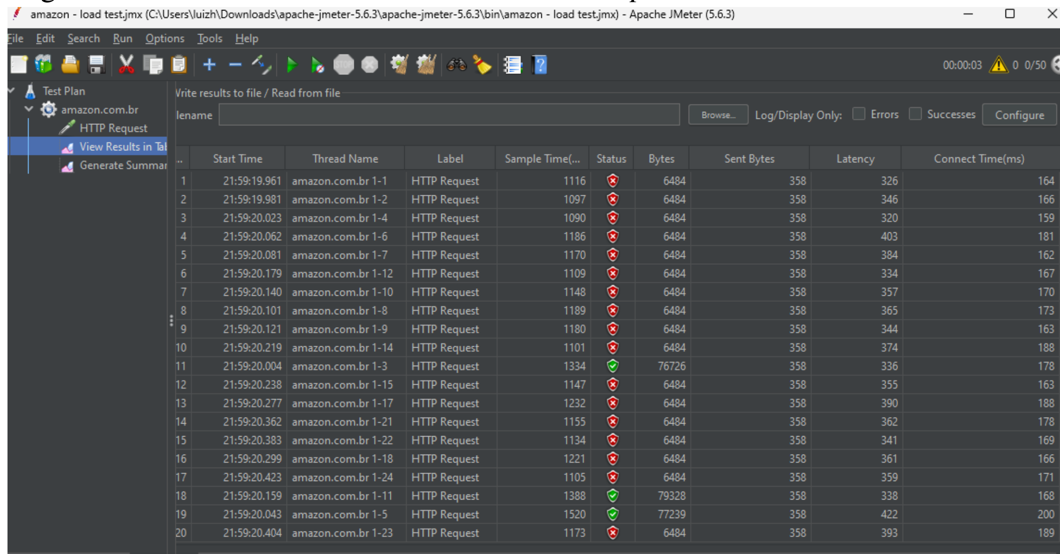
Figura 20 – Configuração de usuários virtuais no *Apache* JMeter



Fonte: Elaborado pelo autor

Os resultados para cada teste de carga pode ser obtido de diversas formas, sendo assim, a ferramenta JMeter demonstrou-se bastante poderosa na exibição dos resultados e disponibilização das métricas. A Figura 21 ilustra os resultados obtidos no teste conforme a configuração exibida anteriormente. Na coleta dos resultados do teste, a opção escolhida para visualização foi tabela de resultados.

Figura 21 – Resultados do teste na ferramenta *Apache* JMeter



..	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	21:59:19.961	amazon.com.br 1-1	HTTP Request	1116		6484	358	326	164
2	21:59:19.981	amazon.com.br 1-2	HTTP Request	1097		6484	358	346	166
3	21:59:20.023	amazon.com.br 1-4	HTTP Request	1090		6484	358	320	159
4	21:59:20.062	amazon.com.br 1-6	HTTP Request	1186		6484	358	403	181
5	21:59:20.081	amazon.com.br 1-7	HTTP Request	1170		6484	358	384	162
6	21:59:20.179	amazon.com.br 1-12	HTTP Request	1109		6484	358	334	167
7	21:59:20.140	amazon.com.br 1-10	HTTP Request	1148		6484	358	357	170
8	21:59:20.101	amazon.com.br 1-8	HTTP Request	1189		6484	358	365	173
9	21:59:20.121	amazon.com.br 1-9	HTTP Request	1180		6484	358	344	163
10	21:59:20.219	amazon.com.br 1-14	HTTP Request	1101		6484	358	374	188
11	21:59:20.004	amazon.com.br 1-3	HTTP Request	1334		76726	358	336	178
12	21:59:20.238	amazon.com.br 1-15	HTTP Request	1147		6484	358	355	163
13	21:59:20.277	amazon.com.br 1-17	HTTP Request	1232		6484	358	390	188
14	21:59:20.362	amazon.com.br 1-21	HTTP Request	1155		6484	358	362	178
15	21:59:20.383	amazon.com.br 1-22	HTTP Request	1134		6484	358	341	169
16	21:59:20.299	amazon.com.br 1-18	HTTP Request	1221		6484	358	361	166
17	21:59:20.423	amazon.com.br 1-24	HTTP Request	1105		6484	358	359	171
18	21:59:20.159	amazon.com.br 1-11	HTTP Request	1388		79328	358	338	168
19	21:59:20.043	amazon.com.br 1-5	HTTP Request	1520		77239	358	422	200
20	21:59:20.404	amazon.com.br 1-23	HTTP Request	1173		6484	358	393	189

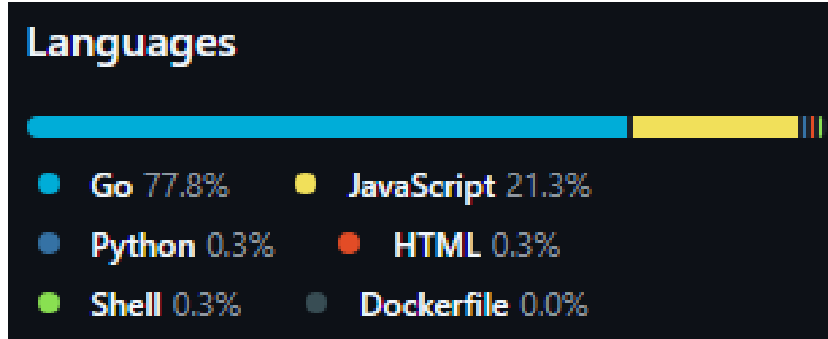
Fonte: Elaborado pelo autor

Conforme os resultados obtidos, a *amazon.com.br* possui um mecanismo de bloqueio de requisições para testes automatizados. No entanto, o teste foi útil para coletar informações, dessa forma, conforme a Figura 21 foram observadas falhas em várias requisições, com exceção de 3. O tempo médio de resposta variou entre aproximadamente 1090 *ms* e 1520 *ms*, com latência média oscilando entre 320 *ms* e 393 *ms*. O tempo de conexão ficou entre 164 *ms* e 189 *ms*, enquanto o tamanho das respostas recebidas (*Bytes*) manteve-se constante, em torno de 6484 *bytes* para requisições que falharam, sugerindo mensagens padronizadas de erro.

5.1.4 Instalação e teste da ferramenta K6

O repositório da ferramenta K6 indica uma forte atividade. A Figura 22 demonstra as principais linguagens utilizadas no desenvolvimento da ferramenta. Cerca de 77,8% do *core* da ferramenta é escrita em Go, enquanto 21,3% é escrita em JavaScript para *scripts* de testes.

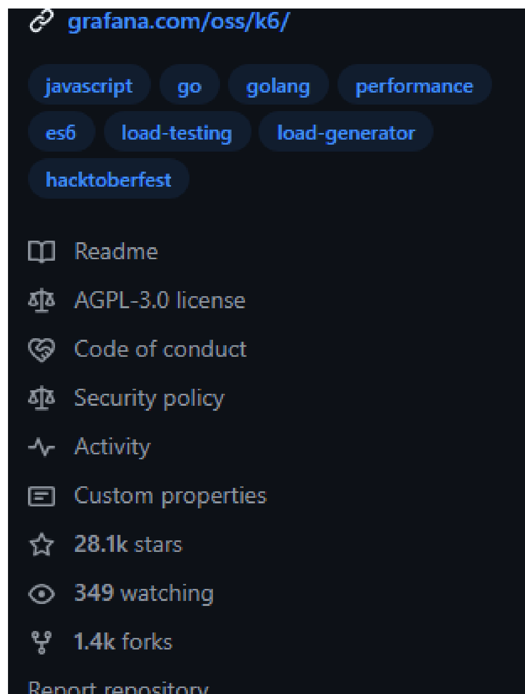
Figura 22 – Linguagens utilizadas na ferramenta K6



Fonte: Adaptado de (Grafana, 2025)

Em matéria de popularidade, acompanhando o repositório da ferramenta, há uma forte atividade por parte dos desenvolvedores. Atualmente, a ferramenta conta com cerca de 28.100 estrelas e 208 contribuintes. A Figura 23 demonstra alguma das informações contidas no repositório.

Figura 23 – Popularidade da ferramenta K6



Fonte: Adaptado de (Grafana, 2025)

A instalação da ferramenta K6, conforme a documentação oficial, pode ser realizada de diversas maneiras. Uma das maneiras é através da obtenção do binário *.zip* obtido na documentação oficial. Outra maneira, que é a maneira que foi seguida neste estudo, é uma instalação via *powershell* através de um gerenciador de pacotes denominado *Chocolatey*.

Inicialmente, é necessária a instalação do gerenciador de pacotes *Chocolatey*. Dessa forma, conforme a documentação oficial do *Chocolatey*, a instalação do gerenciador deve ser realizada via *powershell* (Chocolatey Software, 2025). A Figura 24 demonstra o comando utilizado e a versão do gerenciador de pacotes após a instalação.

Figura 24 – Instalação Chocolatey

```
Administrador: Windows PowerShell
PS C:\WINDOWS\system32> Set-ExecutionPolicy Bypass -Scope Process -Force; `
>> [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
>> iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
AVISO: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.
AVISO: An existing Chocolatey installation was detected. Installation will not continue. This script will not overwrite
 existing installations.
If there is no Chocolatey installation at 'C:\ProgramData\chocolatey', delete the folder and attempt the installation
again.

Please use choco upgrade chocolatey to handle upgrades of Chocolatey itself.
If the existing installation is not functional or a prior installation did not complete, follow these steps:
 - Backup the files at the path listed above so you can restore your previous installation if needed.
 - Remove the existing installation manually.
 - Rerun this installation script.
 - Reinstall any packages previously installed, if needed (refer to the lib folder in the backup).

Once installation is completed, the backup folder is no longer needed and can be deleted.
PS C:\WINDOWS\system32> choco -v
2.4.3
PS C:\WINDOWS\system32>
```

Fonte: Elaborado pelo autor

Com o gerenciador de pacotes instalado, a instalação do K6 pode ser efetuada através do comando `choco install k6`. A Figura 25 demonstra a instalação da ferramenta e posteriormente a execução do comando `k6 version`, validando a instalação.

Figura 25 – Instalação da ferramenta K6

```
Administrador: Windows PowerShell
PS C:\WINDOWS\system32> choco install k6
Chocolatey v2.4.3
Installing the following packages:
k6
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading k6 1.0.0... 100%

k6 v1.0.0 [Approved]
k6 package files install completed. Performing other installation steps.
The package k6 wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): y

Extracting 64-bit C:\ProgramData\chocolatey\lib\k6\tools\k6-v1.0.0-windows-amd64.zip to C:\ProgramData\chocolatey\lib\k6
\tools...
C:\ProgramData\chocolatey\lib\k6\tools
 ShimGen has successfully created a shim for k6.exe
 The install of k6 was successful.
  Deployed to 'C:\ProgramData\chocolatey\lib\k6\tools'

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
 https://chocolatey.org/compare
PS C:\WINDOWS\system32> k6 version
k6.exe v1.0.0 (commit/41b4984b75, go1.24.2, windows/amd64)
PS C:\WINDOWS\system32>
```

Fonte: Elaborado pelo autor

Finalmente, após a instalação, os testes na ferramenta podem ser efetuados totalmente

via *script* em JavaScript. O script[1] foi utilizado para o teste de carga na ferramenta K6. A execução do comando: `k6 run k6_script.js` executa o teste de carga e, posteriormente, entrega os resultados. A Figura 26 demonstra os resultados obtidos nos testes.

Figura 26 – Resultados do teste na ferramenta K6

```
█ THRESHOLDS

  http_req_duration
  ✓ 'p(99) < 3000' p(99)=74.46ms


█ TOTAL RESULTS

  checks_total.......................: 300     9.961133/s
  checks_succeeded...................: 100.00% 300 out of 300
  checks_failed......................: 0.00%   0 out of 300

  ✓ status foi 200

  HTTP
  http_req_duration......................................................: avg=38.87ms min=25.7ms med=36.01ms max=308.95ms
p(90)=46.83ms p(95)=56.85ms
    { expected_response:true }...........................................: avg=38.87ms min=25.7ms med=36.01ms max=308.95ms
p(90)=46.83ms p(95)=56.85ms
  http_req_failed........................................................: 0.00%  0 out of 300
  http_reqs..............................................................: 300    9.961133/s

  EXECUTION
  iteration_duration.....................................................: avg=1.04s   min=1.02s  med=1.03s   max=1.3s
p(90)=1.05s   p(95)=1.09s
  iterations.............................................................: 300    9.961133/s
  vus....................................................................: 1      min=1        max=15
  vus_max................................................................: 15     min=15       max=15

  NETWORK
  data_received..........................................................: 1.9 MB 62 kB/s
```

Fonte: Elaborado pelo autor

Os resultados da Figura 26 demonstram algumas das métricas que podem ser extraídas após um teste de carga com a ferramenta *K6*. Nos resultados, é possível verificar que as requisições realizadas ficaram abaixo dos 3000 ms visto que, o P(99), ou seja, 99% das requisições foram concluídas em até 74,46 ms. Outras métricas importantes estão relacionadas à quantidade de requisições feitas, como *checks_total*, e requisições realizadas com sucesso, como *checks_succeeded*. Os resultados apontam total completude nessas métricas, o que significa que 100% passaram com status 200. Ademais, há diversas outras métricas relevantes, como as de rede, indicando o tamanho total dos dados recebidos (*data_received*) e o tamanho total dos dados enviados (*data_send*).

[1] Disponível em: https://github.com/luizenrike/load_test_01_k6. Acesso em: 29 jun. 2025.

6 RESULTADOS

Neste capítulo, são apresentados os resultados obtidos baseando-se nos procedimentos metodológicos descritos neste trabalho.

6.1 Critérios para seleção de nuvem

Definida as chaves de busca para a seleção de artigos, uma revisão bibliográfica foi realizada nos artigos selecionados, buscando entender quais os critérios utilizados pelos respectivos autores para analisar os ambientes de nuvem. Na revisão, notou-se que os três principais critérios estão relacionados a desempenho ou escalabilidade, custo e conjunto de funcionalidades e serviços. Alguns outros estudos também consideram o suporte ao cliente como critério de seleção.

No trabalho de (Huang; Fang, 2024), os três principais critérios citados são analisados. Os autores também dão ênfase ao quarto critério de suporte ao cliente. Para Huang e Fang (2024), a *AWS* é a plataforma mais antiga, fornecendo uma gama de serviços consolidados no mercado, possuindo clientes como *Netflix* e *Adobe*. Os autores também enfatizam a importância da *Azure* e *Google Cloud Platform*, sendo a *Azure* a maior concorrente da *AWS*, por fornecer fácil integração com serviços da Microsoft, principalmente para empresas que já utilizam tecnologias da Microsoft, como *BMW* e *Verizon*.

A metodologia presente no estudo de (Huang; Fang, 2024), apresentou testes de *benchmark* para avaliar desempenho e simulações de custo. Os critérios de funcionalidades e suporte ao cliente foram analisados baseados nos serviços disponibilizados pelas plataformas e nos números de avaliações e de chamados resolvidos.

O Quadro 6 traz as análises de desempenho realizadas por (Huang; Fang, 2024). O Quadro possui quatro colunas, sendo três principais. A segunda coluna, mede o Giga Floating-Point Operations Per Second (GFLOPS), ou seja, a quantidade (em bilhões) de operações de ponto flutuante por segundo. A terceira coluna, mede o Storage I/O (IOPS), ou seja, quantas operações de leitura ou escrita um sistema de armazenamento consegue executar por segundo. A quarta coluna, mede a latência na rede, ou seja, o tempo que um pacote de dados leva para ir de um ponto A a um ponto B e retornar.

Quadro 6 – Análise dos resultados dos *Benchmarks* de desempenho.

Provedor	Compute Performance (GFLOPS)	Storage I/O (IOPS)	Network Latency (ms)
AWS	250	2000	2.0
Azure	230	1800	2.5
GCP	240	1900	2.2

Fonte: Adaptado de (Huang; Fang, 2024)

Analisando o Quadro 6, notou-se que a AWS liderou os resultados, tendo um melhor desempenho em computação, uma melhor capacidade de leitura e escrita e uma latência menor. A segunda posição ficou com a Google Cloud Platform (GCP), que teve um desempenho, capacidade de leitura e escrita e latência levemente inferior quando comparada com a AWS. O terceiro lugar ficou com a Azure, onde os resultados ficaram levemente abaixo da GCP.

O Quadro 7, traz a análise dos custos mensais por cenários de uso de cada ambiente. Resumidamente, há três cenários, sendo eles de pequena empresa, média empresa e grande empresa. Em suma, os autores concluem que, para quem prioriza minimização de custos, principalmente em operações menores, a AWS leva vantagem nos custos mensais. Em segundo lugar fica a GCP, e a última posição novamente ficou com a Azure.

Quadro 7 – Custos mensais por cenário de uso.

Cenário de Uso	AWS ($/mês)	Azure ($/mês)	GCP ($/mês)
Pequena Empresa (1 VM, 1 TB de armazenamento)	50	55	53
Empresa Média (10 VMs, 10 TB de armazenamento)	500	520	510
Grande Empresa (100 VMs, 100 TB de armazenamento)	5000	5100	5050

Fonte: Adaptado de (Huang; Fang, 2024)

O Quadro 8 contém as análises relacionadas a cada ambiente de nuvem. Nas avaliações, a Azure se destaca em questão de capacidades de nuvem híbrida. Entretanto, em questão de ferramentas para desenvolvedores a AWS se destaca melhor, embora todas as três apresentem “Extensivo” nesse quesito, a AWS se destaca por fornecer melhores ferramentas, principalmente API de integração, o que enriquece a comunidade com exemplos. A Azure fica em segundo lugar nesse quesito, e a terceira posição fica com a GCP.

Quadro 8 – Disponibilidade de funcionalidades.

Funcionalidade	AWS	Azure	GCP
Serviços de Computação	Extensivo	Extensivo	Extensivo
Soluções de Armazenamento	Extensivo	Extensivo	Extensivo
Ferramentas de IA e Aprendizado de Máquina	Avançado	Avançado	Avançado
Big Data e Analytics	Avançado	Avançado	Avançado
Serviços de IoT	Extensivo	Extensivo	Extensivo
Capacidades de Nuvem Híbrida	Moderado	Extensivo	Moderado
Computação Serverless	Sim	Sim	Sim
Ferramentas para Desenvolvedores	Extensivo	Extensivo	Extensivo

Fonte: Adaptado de (Huang; Fang, 2024)

O Quadro 9 traz as análises em relação ao suporte ao cliente que é fornecido por cada plataforma. Diante das análises, notou-se que o suporte da AWS possui um tempo de resposta menor e uma satisfação ao cliente superior. A GCP assume a segunda posição, com um tempo médio de resposta e satisfação do cliente levemente inferior ao da AWS. O terceiro lugar novamente fica com a Azure com um tempo e satisfação levemente inferior ao da GCP.

Quadro 9 – Avaliação do suporte ao cliente.

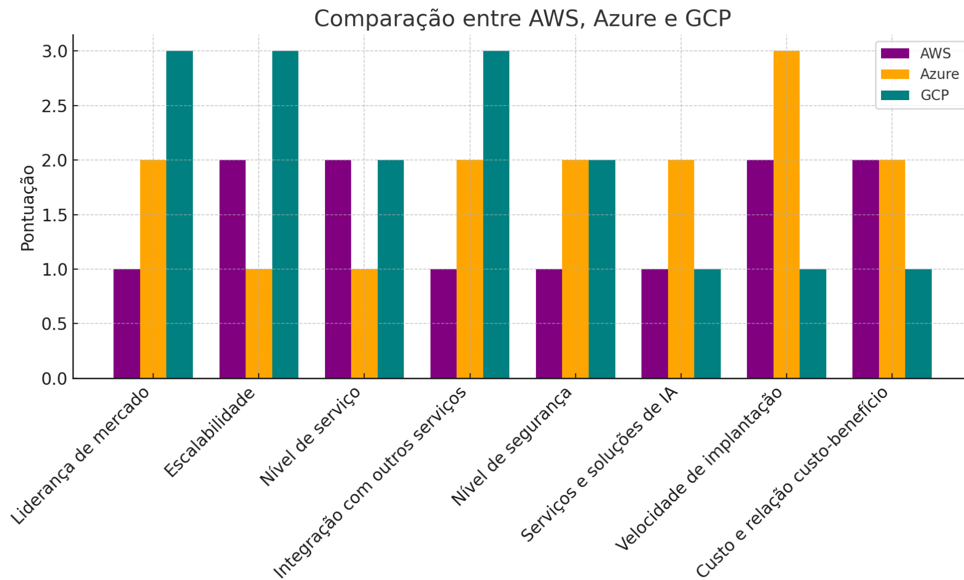
Provedor	Tempo médio de resposta (horas)	Satisfação do cliente (1–5)
AWS	1,5	4,5
Azure	2,0	4,2
GCP	1,8	4,3

Fonte: Adaptado de (Huang; Fang, 2024)

Segundo as análises retiradas do estudo de (Huang; Fang, 2024), a AWS assume total liderança com uma melhor precificação, maior desempenho, boas funcionalidades e um ótimo suporte ao cliente. A GCP fica em segundo lugar, obedecendo aos mesmos quesitos, e o terceiro lugar fica com a Azure, que se destaca melhor nas funcionalidades com nuvem híbrida. Nesse sentido, a escolha de um ambiente de nuvem pode ser decidida por alguns desses critérios, mas também pode ser levada em conta a capacidade distinta de cada um, principalmente em integração de serviços.

No trabalho de (Nevludov; Sotnik, 2023), há um estudo sobre os três ambientes analisados anteriormente. No estudo, os autores se debruçam sobre alguns outros critérios como escalabilidade, custo, serviço e alguns outros. A Figura 27 permite uma melhor visualização dos resultados obtidos pelos autores.

Figura 27 – Comparação entre os ambientes AWS, Azure e GCP



Fonte: Adaptado de (Nevludov; Sotnik, 2023)

De acordo com os resultados de (Nevludov; Sotnik, 2023), não há uma definição clara de um ambiente vencedor, entretanto, é observada uma eficiência melhor de alguns ambientes em um critério específico. Conforme observa-se na Figura 27, em quesito de escalabilidade, a GCP leva o primeiro lugar, seguida da AWS e posteriormente da Azure, todavia, em quesito de velocidade de implantação, a Azure obtém o melhor resultado, seguida da AWS posteriormente da GCP.

No estudo de (Saraswat; Tripathi, 2020), há uma definição de alguns critérios fundamentais ao escolher uma nuvem, sendo alguns deles: serviços de infraestrutura e computação, serviços de tecnologias de rede, tecnologias de armazenamento, suporte a bancos de dados, serviços de backup e ferramentas essenciais. Para Saraswat e Tripathi (2020), a AWS pode ser a melhor opção, visto que é uma plataforma que possui um portfólio mais amplo e com maior estabilidade com um custo razoável. A Azure pode ser a melhor opção quando a empresa utiliza tecnologias da Microsoft e há necessidade de integração entre essas tecnologias. A GCP se torna mais ideal para empresas que lidam com uma maior quantidade de dados e buscam melhor flexibilidade nos recursos.

Nesse sentido, conforme analisado nos estudo de (Huang; Fang, 2024), (Saraswat; Tripathi, 2020) e (Nevludov; Sotnik, 2023), os critérios que mais foram discutidos estão relacionados a desempenho, custo e serviços, entretanto, neste presente estudo também será levado em consideração o critério de suporte ao cliente. Os critérios servem como parâmetros para analisar o ambiente, entretanto, cada ambiente pode se sair melhor em um determinado critério e

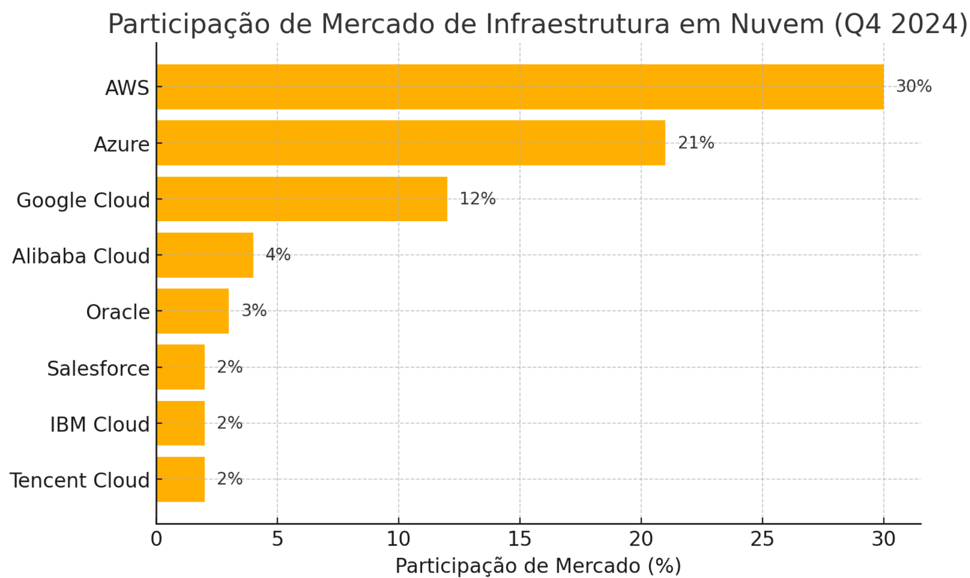
a escolha pode depender das necessidades e requisitos de cada aplicação ou negócio.

6.2 Seleção de ambientes de nuvem

Conforme abordado na metodologia deste trabalho, esta seção utilizará como base os critérios abordados na seção 6.1, todavia, com o objetivo de incrementar ainda mais o estudo, também utilizaremos outros dois critérios: utilização por parte de usuários da tecnologia e arrecadação das principais plataformas.

O *website* STATISTA (2024), elencou em 2024 as maiores receitas dos provedores de computação em nuvem. Nesse sentido, conforme a Figura 28, a AWS lidera com 30% de participação, sendo então o provedor que mais captura receita no mercado de infraestrutura em nuvem, indicando liderança em volume de negócios. A Azure assume o segundo lugar com 21%, e a GCP o terceiro lugar com 12%.

Figura 28 – Receita no mercado de infraestrutura em nuvem.



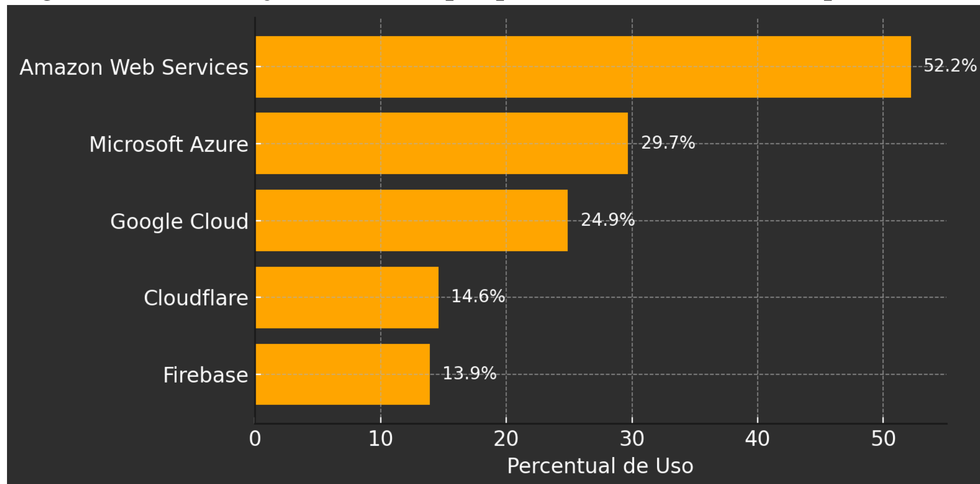
Fonte: Adaptado de (STATISTA, 2024)

No critério de utilização, conforme abordado no estudo de (Araujo, 2019), foi utilizada a plataforma *StackOverflow*, uma plataforma utilizada por diversos desenvolvedores e aprendizes de tecnologia para compartilhar conhecimentos através de diversas ferramentas.

O *website* STACKOVERFLOW (2024), realizou uma entrevista entre seus usuários para identificar a porcentagem da utilização de plataformas de computação em nuvem. A Figura 29 representa a porcentagem de utilização dos ambientes em nuvem por parte de desenvolvedores profissionais. Na figura, a AWS lidera a pesquisa com 52,2%, a Azure aparece em segundo lugar

com 29,2% e a GCP em terceiro com 24,9%.

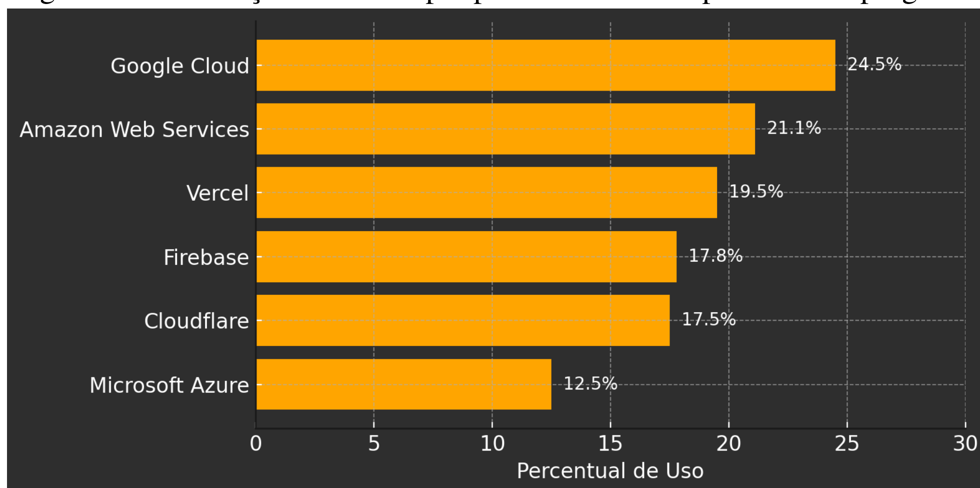
Figura 29 – Utilização de nuvem por parte de desenvolvedores profissionais.



Fonte: Adaptado de (STACKOVERFLOW, 2024)

A entrevista realizada pelo *website* (STACKOVERFLOW, 2024) também englobou usuários que estão aprendendo a programar. Nesse cenário, conforme a Figura 30, a GCP lidera a pesquisa com 24,5%, a AWS aparece em segundo lugar com 21,1% enquanto a Azure aparece em sexto lugar com 12,5%, ficando atrás de outros ambientes menos populares. O fato da GCP aparecer em primeiro lugar na escolha de usuários que estão iniciando no universo da programação está interligada a diversos fatores, alguns deles são: créditos iniciais generosos, foco em materiais didáticos e facilidade no aprendizado.

Figura 30 – Utilização de nuvem por parte de usuários aprendendo a programar.



Fonte: Adaptado de (STACKOVERFLOW, 2024)

Nesse sentido, analisando os critérios de desempenho, custos e serviços, atrelados aos critérios de infraestrutura e utilização por parte dos usuários, os três principais ambientes

(AWS, Azure e GCP) apresentaram métricas próximas, entretanto os ambientes da AWS e Azure foram selecionados baseando-se em critérios de desempate. Ambos ambientes se destacaram nos critérios de velocidade de implantação e custo-benefício apresentados na Figura 27, também se destacaram pela receita em infraestrutura conforme abordado na Figura 28.

6.2.1 Visão geral dos provedores selecionados

6.2.1.1 Amazon Web Services

A AWS foi lançada oficialmente em 2006, quando a equipe da Amazon percebeu que sua própria experiência com o dimensionamento de infraestrutura do site de e-commerce poderia beneficiar terceiros. Inicialmente, foram disponibilizados o serviço de armazenamento em nuvem Simple Storage Service (S3) em março de 2006 e, em agosto do mesmo ano, o Elastic Compute Cloud (EC2), permitindo que qualquer pessoa, de startups a grandes empresas, acessasse recursos de computação sob demanda pela Internet (AWS, 2024b).

Desde seu lançamento, a AWS expandiu rapidamente seu portfólio de serviços e conquistou mercado por oferecer um modelo de pagamento *pay-as-you-go*, que substitui grandes investimentos iniciais em hardware por custos operacionais variáveis, escaláveis conforme a demanda. Em 2010, já eram dezenas de serviços, desde bancos de dados gerenciados a filas de mensagens. Em 2025, ultrapassam 200 produtos e funcionalidades, contemplando desde o *core* de computação até inteligência artificial e Internet das Coisas (AWS, 2024b).

Para sustentar essa variedade de serviços, a AWS construiu uma rede mundial de regiões e zonas de disponibilidade. Cada região AWS é composta por pelo menos três zonas de disponibilidade fisicamente isoladas, com exceção da Austrália e Nova Zelândia, interligadas por links de fibra de alta velocidade. Essa arquitetura garante alta disponibilidade e baixa latência para clientes de todos os continentes. Além disso, a AWS mantém *data centers* em locais estratégicos, cobrindo cerca de 30 regiões geográficas e mais de 80 zonas de disponibilidade ao redor do mundo (AWS, 2024a).

6.2.1.2 Microsoft Azure

O lançamento do Azure ocorreu formalmente na *Professional Developers Conference (PDC)* em outubro de 2008, quando Ray Ozzie, então arquiteto-chefe de software da Microsoft, anunciou ao público a primeira versão da plataforma, originalmente denominada Windows Azure.

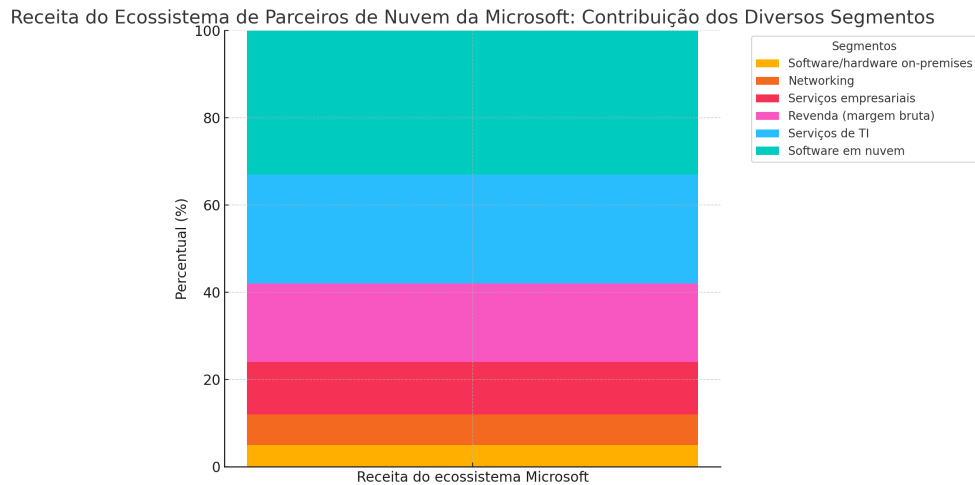
Nessa época inicial, o Azure tinha como principal objetivo oferecer serviços de computação em nuvem com foco em aplicações baseadas no sistema operacional Windows, permitindo aos desenvolvedores construírem e hospedarem suas aplicações em *data centers* da Microsoft, acessíveis através da Internet (Microsoft, 2022).

Em 2010, após quase dois anos desde seu anúncio inicial, o Azure foi oficialmente disponibilizado para uso comercial, proporcionando serviços de hospedagem de aplicativos, armazenamento de dados e computação escalável sob demanda. No início de 2014, o Windows Azure foi renomeado para Microsoft Azure, refletindo sua expansão para além do ambiente Windows e destacando sua capacidade multiplataforma. Desde então, a Azure evoluiu rapidamente, expandindo suas ofertas para incluir mais de 200 produtos e serviços que abrangem inteligência artificial, análise de dados, *blockchain*, IoT e diversos outros domínios tecnológicos (Microsoft, 2022).

Ao longo dos anos, a Azure construiu uma extensa infraestrutura global, com mais de 60 regiões distribuídas em 140 países (Microsoft, 2024). Essa infraestrutura robusta permite aos clientes implementarem seus aplicativos próximos aos usuários finais, proporcionando baixa latência, alta disponibilidade e conformidade com requisitos regulatórios específicos por região. Além disso, cada região do Azure é organizada em múltiplas zonas de disponibilidade fisicamente separadas, oferecendo maior redundância e segurança aos dados e serviços hospedados na nuvem.

Desde o seu lançamento, a Microsoft Azure experimentou uma significativa ascensão em termos de adoção e participação no mercado global de infraestrutura em nuvem. O rápido crescimento da Azure pode ser atribuído, entre outros fatores, à sua capacidade de integração nativa com outras soluções Microsoft, como Microsoft 365, Dynamics 365 e a ampla adoção de ferramentas de desenvolvimento como Visual Studio e GitHub, criando um ecossistema altamente atrativo para empresas e desenvolvedores de diversos segmentos. A Figura 31, extraída do relatório da (IDC, 2023), representa os produtos mais lucrativos da Microsoft em 2023 com os serviços de nuvem liderando a maior fatia da receita.

Figura 31 – Receita da Microsoft em serviços.



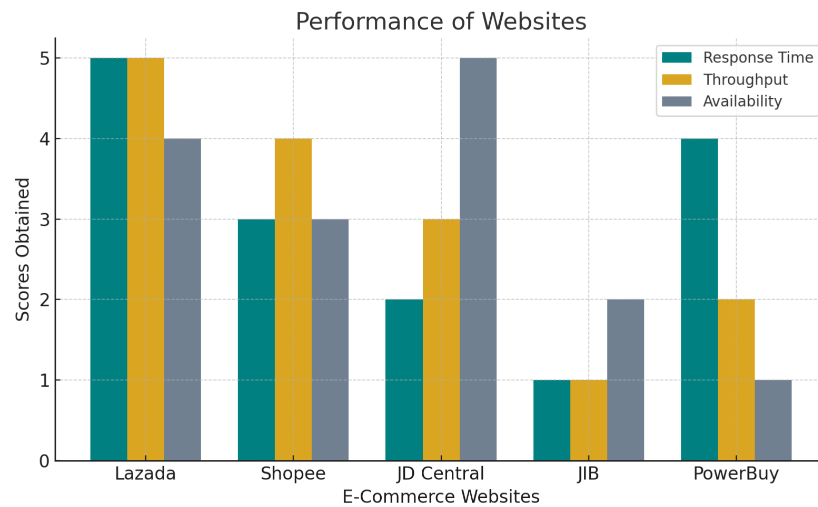
Fonte: Adaptado de (IDC, 2023)

6.3 Critérios de ferramentas de teste de carga

Conforme abordado na etapa metodológica deste estudo, esta seção contempla uma análise relacionada as ferramentas de cargas *open source* disponíveis. A análise contempla alguns critérios, especialmente de protocolos suportados, métricas coletadas e comunidade.

O estudo publicado por Shafana *et al.* (2021) efetua testes de cargas em cinco grandes sites de *e-commerce* da Tailândia. No estudo, a ferramenta JMeter foi utilizada para capturar os resultados. Nesse sentido, para (Shafana *et al.*, 2021) os testes de carga são uma fase importante durante o desenvolvimento de sistemas em larga escala e com base nas métricas coletadas durante os testes, os desenvolvedores podem corrigir problemas que surgem quando diversos usuários acessam a mesma URL simultaneamente. A Figura 32 demonstra as principais métricas utilizadas no estudo, sendo *throughput*, tempo de resposta e disponibilidade.

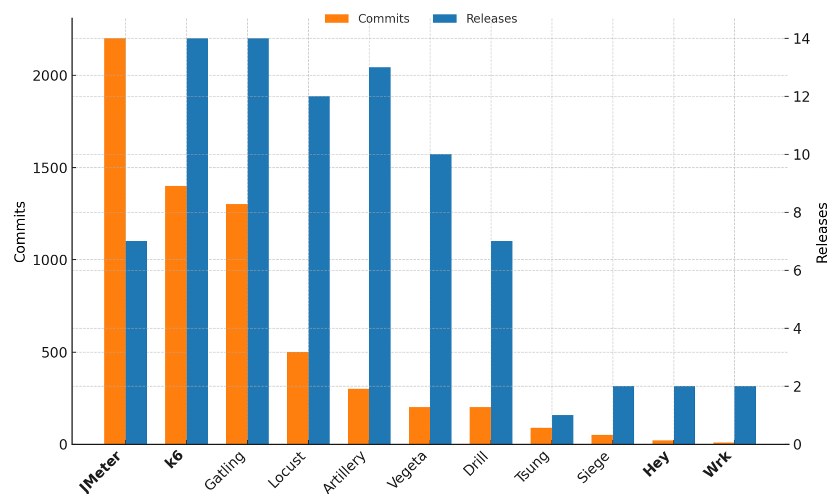
Figura 32 – Performance de *e-commerces* da Tailândia.



Fonte: Adaptado de (Shafana *et al.*, 2021)

Uma revisão realizada por (Lonn, 2020) sobre as principais ferramentas de cargas *open source* identificou as principais ferramentas mais ativas no ano de 2020 em quesito de *commits* e *releases* em seus respectivos repositórios. A Figura 33 demonstra a atividade das principais ferramentas, com destaque para as ferramentas JMeter, K6, Hey e Wrk.

Figura 33 – Ferramentas de testes de cargas mais ativas no ano de 2020.



Fonte: Adaptado de (Lonn, 2020)

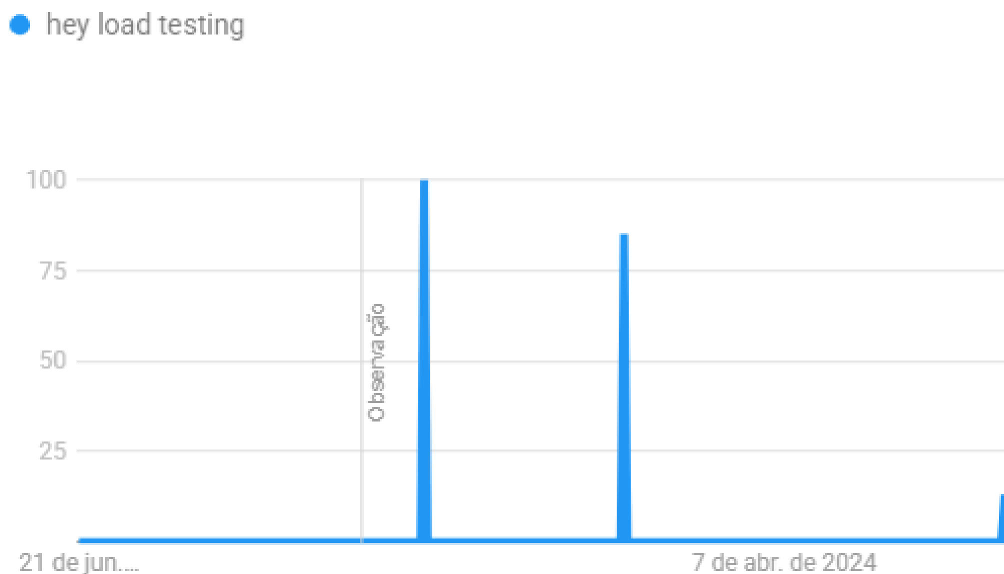
Diante desse cenário, uma análise mais aprofundada das quatro ferramentas de teste de cargas elencadas na etapa metodológica deste estudo viabiliza a seleção de duas ferramentas que irão efetuar testes nas aplicações elaboradas no decorrer do estudo.

6.3.1 Hey

Hey é uma ferramenta moderna e leve de teste de carga HTTP, desenvolvida com o objetivo de simplificar e agilizar avaliações de desempenho de aplicações *web* e APIs. É uma ferramenta que foi desenvolvida em linguagem Go, e o objetivo geral da sua criação foi para fornecer alta performance e facilidade de uso, possibilitando que usuários realizem rapidamente testes básicos e avançados em servidores web e serviços REST. Apesar de leve, a ferramenta Hey é capaz de gerar um grande número de requisições simultâneas mesmo com recursos computacionais modestos, o que facilita a realização de *benchmarks* rápidos e eficientes.

Embora destaque-se pela simplicidade em sua utilização, oferecendo uma interface por linha de comando, a ferramenta Hey nos últimos anos apresentou uma baixa taxa de atualização conforme a Figura 33 e uma baixa procura por parte dos desenvolvedores. A Figura 34 demonstra o interesse nos últimos cinco anos.

Figura 34 – Interesse na ferramenta Hey nos últimos 5 anos.



Fonte: Adaptado de (Google, 2025d)

Embora a ferramenta tenha perdido espaço nos últimos anos, ela ainda se mantém como uma das principais ferramentas devido a facilidade com que os usuários podem configurar parâmetros essenciais, como por exemplo o número total de requisições, concorrência (quantidade simultânea de conexões abertas) e métodos HTTP personalizados, permitindo adaptações rápidas às necessidades específicas de cada cenário de teste. A possibilidade de execução de requisições HTTP simples e complexas torna o Hey uma alternativa ágil.

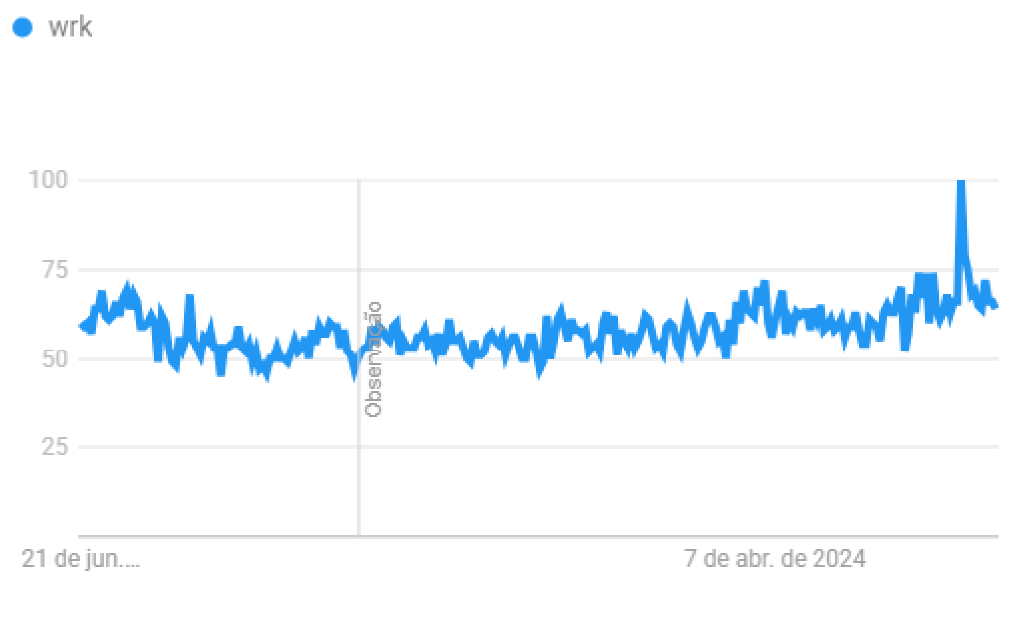
6.3.2 *Wrk*

Semelhantemente a ferramenta Hey vista na seção 6.3.1, o Wrk também é uma ferramenta de código aberto, leve e sem interface gráfica que permite aos usuários testes de cargas através de linhas de comando. Segundo API7 (2022), o Wrk é uma ferramenta sólida capaz de gerar estresse significativo o suficiente para sobrecarregar o programa do lado do servidor.

Nesse sentido, de acordo com (Glozer, 2025), a ferramenta Wrk foi escrita na linguagem C e sua estrutura *multi-threaded* permite que múltiplos núcleos de CPU sejam utilizados simultaneamente durante o teste, aumentando a capacidade de geração de tráfego de forma controlada, demonstrando o poder computacional da ferramenta.

Embora nos últimos anos a ferramenta tenha perdido espaço para outras gigantes, o Wrk ainda se destaca pela sua simplicidade. A Figura 35 demonstra o interesse em busca da ferramenta Wrk no Google nos últimos 5 anos.

Figura 35 – Interesse na ferramenta Wrk nos últimos 5 anos.



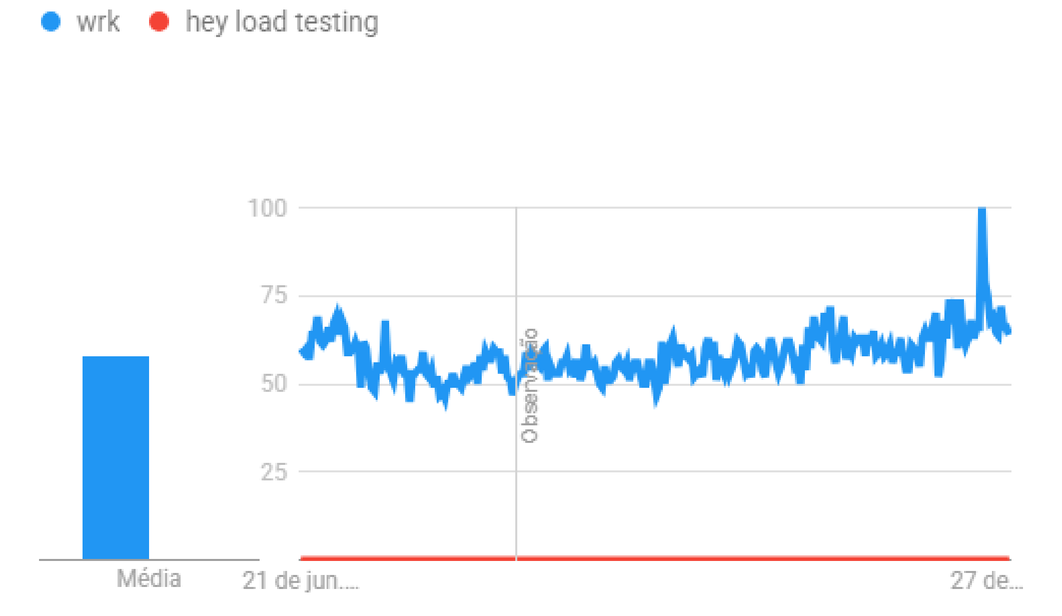
Mundo. Nos últimos 5 anos. Pesquisa Google na Web.

Fonte: Adaptado de (Google, 2025e)

Em uma comparação realizada no Google Trends para analisar o volume de buscas nas ferramentas Hey e Wrk, enquanto o interesse na ferramenta Hey ficou próximo de zero, notou-se que o Wrk manteve um interesse estável nos últimos anos, e apontou um pico de interesse maior do que o Hey no ínicio de 2025. A Figura 36 demonstra a diferença de interesse entre

ambas as ferramentas, o que indica uma maior popularidade e uma melhor adesão principalmente dos desenvolvedores na ferramenta Wrk.

Figura 36 – Interesse nas ferramentas Hey vs Wrk nos últimos 5 anos.



Fonte: Adaptado de (Google, 2025a)

Conforme observado nas subseções 5.1.1 e 5.1.2, embora a popularidade da ferramenta Wrk seja maior do que a popularidade do Hey, a ferramenta Hey demonstrou maior facilidade na instalação e mais recursos para serem utilizados no momento do teste de carga. Ademais, a ausência do suporte ao protocolo HTTPs é um ponto negativo no Wrk. Em relação aos resultados retornados após um teste de carga, os resultados da ferramenta Hey demonstraram maior completude quando comparados com a ferramenta Wrk.

6.3.3 JMeter

O Apache JMeter é uma ferramenta de código aberto, desenvolvida pela *Apache Software Foundation*, projetada para testar e medir o desempenho de aplicações web e serviços diversos. Lançado no início dos anos 2000, o JMeter evoluiu de uma simples ferramenta de teste de carga em protocolo HTTP para um oficina de testes que suporta múltiplos protocolos.

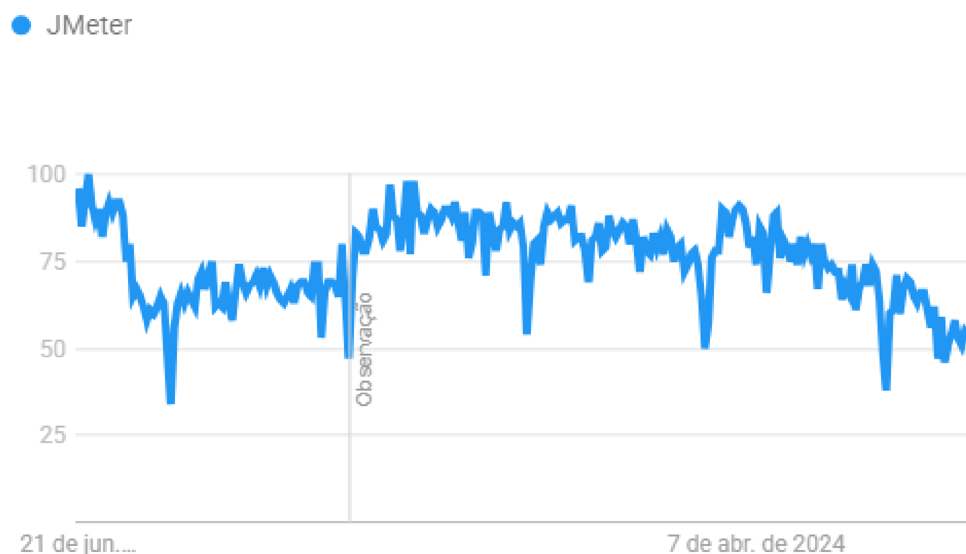
De acordo com (JMETER, 2024), a ferramenta atualmente suporta diversos protocolos e serviços como HTTP, HTTPs, File Transfer Protocol (FTP), Simple Object Access Protocol (SOAP), REST e diversos outros. Escrito em Java, o JMeter é um *software* multiplataforma

e com alta portabilidade. Ademais, os resultados dos testes podem ser obtidos em diversos formatos como JSON, HTML e eXtensible Markup Language (XML).

Uma caracteristica principal do JMeter é uma interface gráfica, que facilita a criação e a configuração de testes de cargas, ademais, a ferramenta possui uma extensa variedade de *plugins* capazes de gerar relatórios em tempo real e de realizar integração com ferramentas de CI/CD. Esse conjunto de características reforçam o poder do JMeter mediante as outras ferramentas.

Em matéria de interesse, a Figura 33 demonstrou que o JMeter assumiu a liderança. Nesse sentido, com o objetivo de reforçar as buscas pela ferramenta, uma pesquisa no Google Trends indicou o forte interesse dos usuários na ferramenta. A Figura 37 demonstra o interesse nos últimos 5 anos.

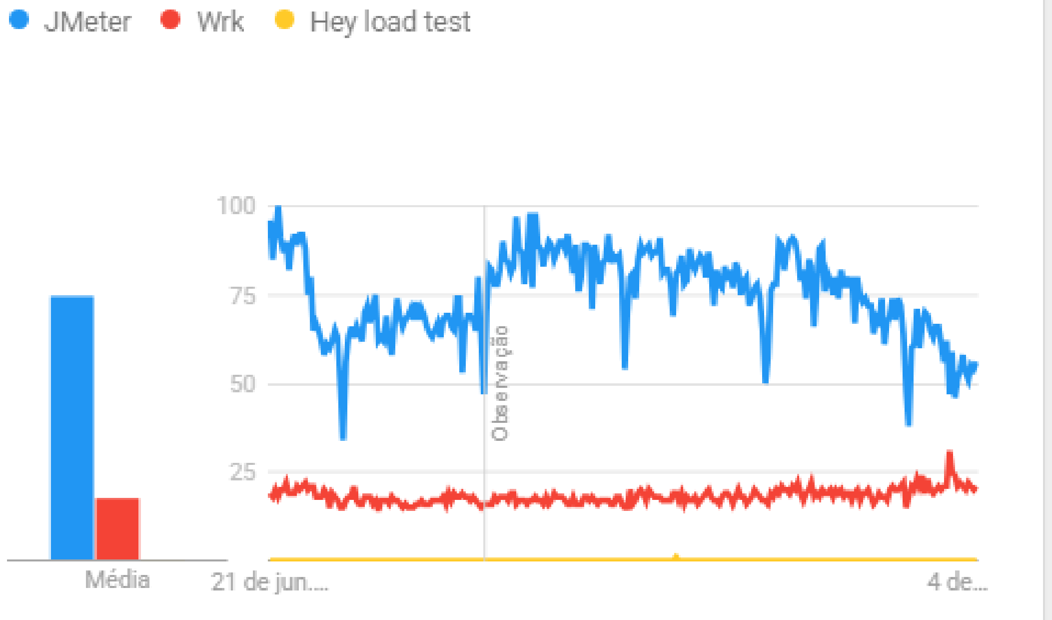
Figura 37 – Interesse na ferramenta JMeter nos últimos 5 anos.



Fonte: Adaptado de (Google, 2025f)

Em uma comparação entre as três ferramentas vistas, o JMeter se mantém na liderança em primeiro lugar, seguido pelo Wrk em segundo e a ferramenta Hey permanece em última próxima a zero. A Figura 38 demonstra os resultados. Ademais, conforme visto na subseção 5.1.3, o JMeter possui um enorme potencial, além de diversas métricas que podem ser obtidas com diferentes *Listeners*.

Figura 38 – Interesse nas ferramentas JMeter vs Hey vs Wrk nos últimos 5 anos



Fonte: Adaptado de (Google, 2025c)

6.3.4 K6

A ferramenta K6, inicialmente conhecida como *Load Impact*, escrita em sua grande parte na linguagem `Go`, é uma ferramenta de testes de cargas de código aberto. Segundo Gustafsson (2022), em 2021 a Grafana Labs, uma empresa renomada em ferramentas de observabilidade, adquiriu o K6. Dessa forma, atualmente a ferramenta é mantida pela Grafana Labs, porém, ainda é uma ferramenta de código aberto.

Atualmente o K6 é totalmente gratuito, entretanto, ele possui uma versão paga, comercial, denominada K6 *Cloud* na qual é mantida e desenvolvida pela Grafana Labs. Resumidamente, trata-se de uma versão que possui interface gráfica e as execuções dos testes não são executadas localmente, mas sim em um servidores de nuvem da Grafana, visando facilitar a execução de testes de cargas.

A documentação oficial do K6 (Grafana, 2024), elenca os principais protocolos suportados pela ferramenta. Dessa forma, o Quadro 10 demonstra os principais protocolos suportados pela ferramenta, sendo alguns nativos e outros suportados somente através de extensões.

Quadro 10 – Protocolos suportados pela ferramenta K6.

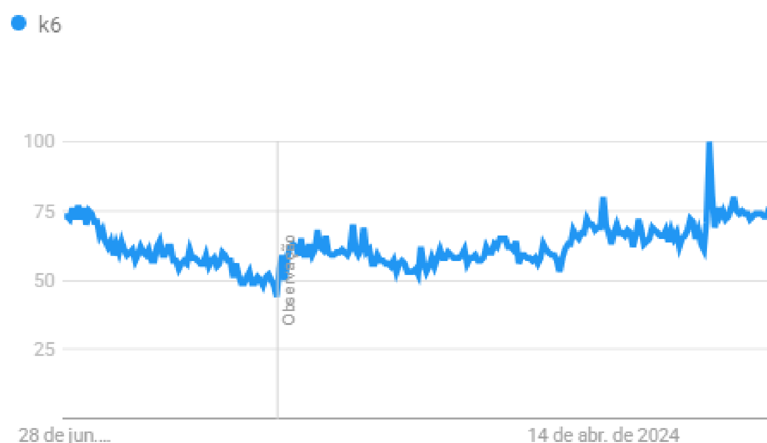
Protocolo	**Descrição**	**Suporte**
HTTP/1.1	Suporte completo a requisições HTTP tradicionais, incluindo cabeçalhos, corpo e autenticação.	Nativo
HTTP/2	Suporte a conexões multiplexadas, mais eficientes que HTTP/1.1.	Nativo
WebSocket	Permite testar conexões em tempo real com comunicação bidirecional.	Nativo
gRPC	Suporte a chamadas gRPC usando a extensão `xk6-grpc`.	Extensão
MQTT	Suporte ao protocolo de mensagens MQTT, usado em IoT, via extensão `xk6-mqtt`.	Extensão

Fonte: Adaptado de (Grafana, 2024)

Os resultados dos testes de cargas efetuados na ferramenta K6 podem ser obtidos em diversos formatos de saída. Os principais formatos suportados são: `text`, `json` e`csv`. Ademais, a ferramenta possui outros diversos formatos de saída que são compatíveis com servidores e nuvem.

Conforme a Figura 33, o K6 ficou atrás somente do *Apache* JMeter. Nesse sentido, o interesse na ferramenta é constante e com o objetivo de reforçar o interesse pela ferramenta, uma pesquisa no Google Trends na ferramenta K6 também indicou uma forte busca por parte dos usuários. A Figura 39 demonstra as buscas pelo K6 nos últimos 5 anos.

Figura 39 – Interesse na ferramenta K6 nos últimos 5 anos.

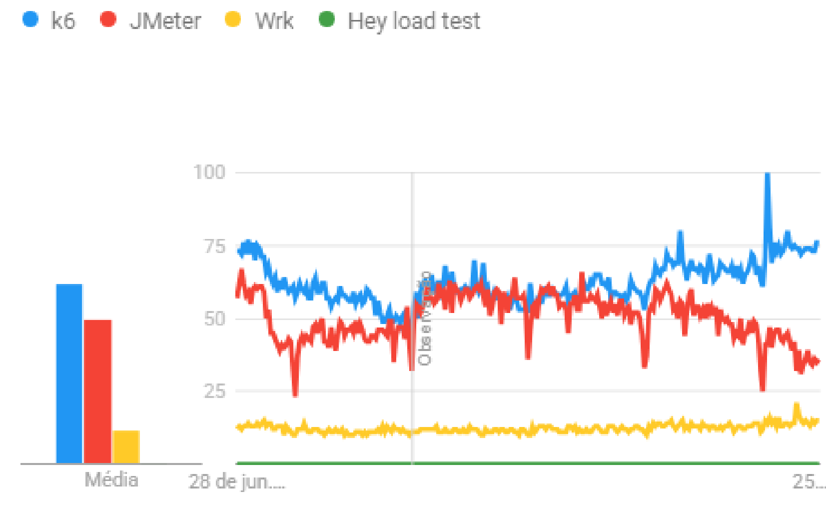


Fonte: Adaptado de (Google, 2025h)

Comparando as quatro ferramentas vistas até o momento, em matéria de interesse e

buscas no Google, o K6 assumiu a liderança sendo a ferramenta mais buscada. O JMeter ficou em segundo lugar, o Wrk em terceiro e a ferramenta Hey fica em último. A Figura 40 demonstra os resultados.

Figura 40 – Interesse nas ferramentas K6 vs JMeter vs Hey vs Wrk nos últimos 5 anos.



Mundo. Nos últimos 5 anos. Pesquisa Google na Web.

Fonte: Adaptado de (Google, 2025g)

6.3.5 *Comparativo e seleção das ferramentas*

Conforme as instalações e testes realizados no capítulo 5, e conforme a metodologia deste estudo, apenas duas ferramentas foram selecionadas. As ferramentas selecionadas serão utilizadas para testar as aplicações desenvolvidas e hospedadas nos ambientes de nuvens também já selecionados.

Os critérios de seleção mais relevantes foram métricas, portabilidade e popularidade. O Quadro 11 realiza um comparativo entre as ferramentas e seleciona as duas que atenderam os critérios.

Quadro 11 – Comparativo entre ferramentas de teste de carga avaliadas.

Ferramenta	Selecionada	Critérios Atendidos	Justificativa da Seleção ou Exclusão
Hey	Não	Portabilidade, Métricas	Não selecionada devido à *baixa popularidade* e menor suporte da comunidade.
Wrk	Não	Métricas, Popularidade	Não selecionada por *não ser multiplataforma* (somente Unix) e por *não suportar HTTPS*, o que inviabiliza testes com APIs na nuvem.
JMeter	Sim	Todos	Selecionada por apresentar ampla *portabilidade*, *suporte a múltiplos protocolos*, métricas detalhadas e grande popularidade no mercado.
K6	Sim	Todos	Selecionada por ser moderna, multiplataforma, com suporte robusto a *HTTP/HTTPS*, métricas programáveis e fácil integração com pipelines de Continuous Integration (CI)/Continuous Deployment (CD).

Fonte: Elaborado pelo autor

Nesse sentido, de acordo com os critérios atendidos em cada ferramenta, verificou-se que o *JMeter* e o *K6* foram selecionadas para serem utilizadas nos testes das aplicações desenvolvidas na seção 6.4.

6.4 Configuração e aplicação prática

Com o objetivo de realizar as análises de teste de carga em ambientes de nuvem, uma aplicação prática necessita ser desenvolvida. Nesse sentido, a aplicação que será utilizada no estudo consistirá em uma API REST, desenvolvida em duas linguagens distintas: *PHP* e *C#*. Para ambas as implementações, serão utilizados *frameworks* específicos: no *PHP*, será adotado o *Laravel*, e no *C#*, o *ASP.NET Core*, pertencente à plataforma *.NET*.

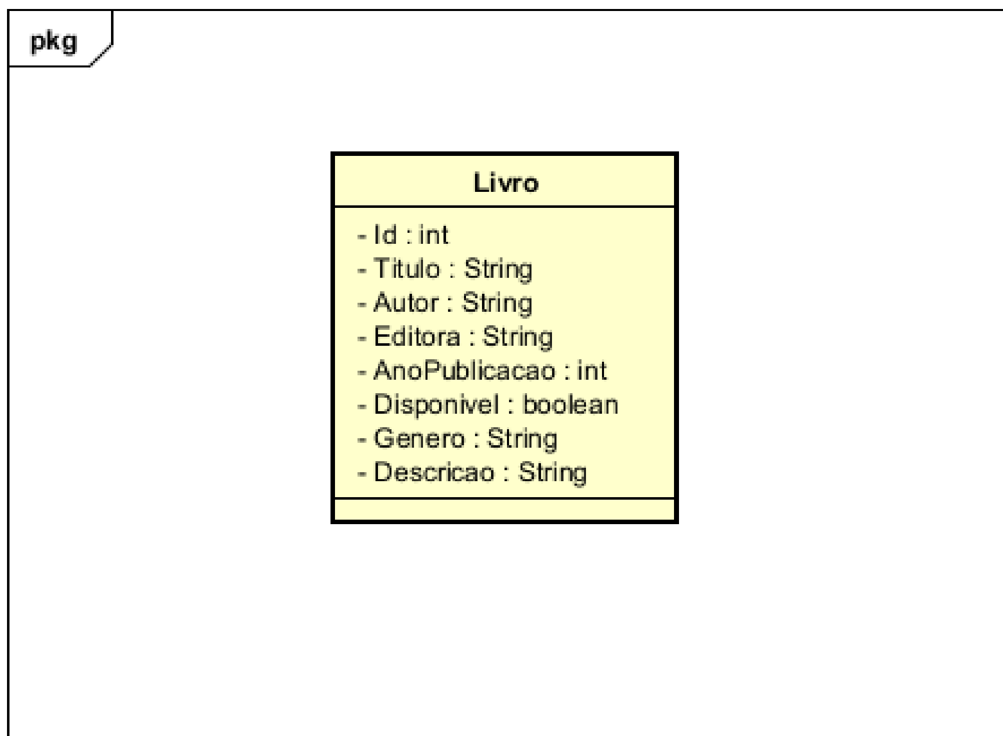
Antes da inicialização da aplicação prática, faz-se necessário entender conceitos importantes que serão vistos nesta seção. O termo *framework* pode ser definido como um conjunto de bibliotecas que facilitam o desenvolvimento de aplicações. De maneira complementar, um *framework* de software é uma plataforma concreta ou conceitual onde códigos comuns com

funcionalidades genéricas podem ser seletivamente especializados ou sobrescritos por desenvolvedores ou usuários. Os frameworks assumem a forma de bibliotecas, nas quais uma interface de programação de aplicações (API) bem definida pode ser reutilizada em qualquer parte do software em desenvolvimento (Mwendi, 2014).

Outro termo importante e que se faz necessário entender, é o Create, Read, Update and Delete (CRUD). Segundo Jadhav *et al.* (2025), CRUD é um acrônimo para Create (Criar), Read (Ler), Update (Atualizar) e Delete (Excluir), sendo essencial para a implementação de qualquer aplicação robusta com banco de dados relacional.

A aplicação prática do presente estudo consiste em uma API REST, implementando operações CRUD. A API utilizará um banco de dados relacional *MySQL* para o armazenamento dos dados. A Figura 41 apresenta a estrutura da classe *Livro*, que será utilizada na aplicação prática. Como o objetivo da aplicação é apenas servir de meio para a realização dos testes de carga, uma única classe é suficiente para atender às necessidades do estudo.

Figura 41 – Diagrama de classes da aplicação elaborado no *software* Astah



Fonte: Elaborado pelo autor

As aplicações desenvolvidas estão disponíveis em repositórios públicos. A aplicação *PHP*[1] e *.NET*[2] encontram-se nos respectivos repositórios para análises e estudos.

[1] Disponível em: https://github.com/luizenrike/deploy-php-api. Acesso em: 15 jul. 2025.
[2] Disponível em: https://github.com/luizenrike/livro-api. Acesso em: 15 jul. 2025.

6.4.1 *Deploy das aplicações no ambiente Azure*

As aplicações desenvolvidas foram hospedadas inicialmente no ambiente da Microsoft. Iniciando-se pela aplicação em PHP e posteriormente a aplicação em .NET. Para a realização do *deploy*, a configuração do ambiente da Azure foi por meio da *Azure Application Service*, um modelo PaaS disponibilizado pela Microsoft. Dessa forma, toda a camada de infraestrutura foi abstraída durante o *deploy*.

Na *Azure Application Service* foi utilizado o plano B2, um plano que visa aplicações que necessitam de uma capacidade mediana de processamento, ofertando 2 núcleos dedicados e 3,5 Gigabyte (GB) de memória . A Figura 42 entrega mais detalhes do plano e os custos.

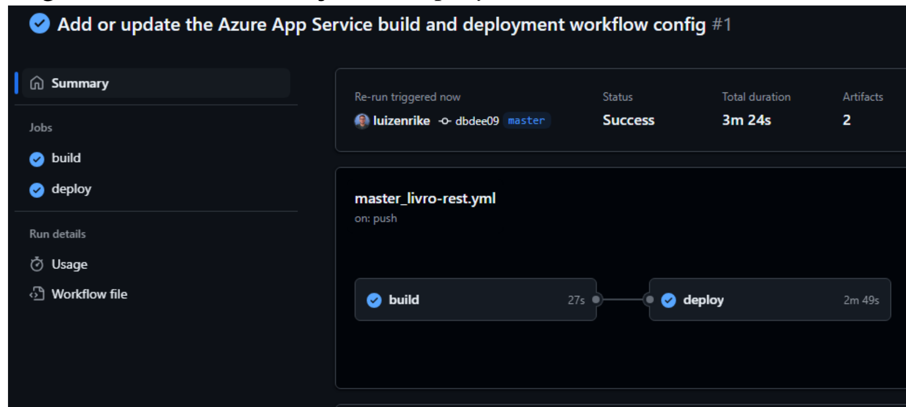
Figura 42 – Plano utilizado na *Azure Application Service*

	Nome	ACU/vCPU	vCPU	Memória (GB)	Armazenamento Remoto (GB)	Custo por hora (instância)	Custo por mês (instância)
	Desenvolvimento/Teste (Para cargas de trabalho com menos demanda)						
	Gratuito F1	60 minutos/dia de	N/A	1	1	**Gratuito**	**Gratuito**
	Compartilhado D1	240 minutos/dia de	N/A	1	1	**0,013 USD**	**9,49 USD**
	Básico B1	100	1	1.75	10	**0,075 USD**	**54,75 USD**
✓	Básico B2	100	2	3.5	10	**0,15 USD**	**109,50 USD**
	Básico B3	100	4	7	10	**0,30 USD**	**219,00 USD**

Fonte: Elaborado pelo autor

Após a seleção do plano e a escolha da pilha de *runtime* na primeira máquina, a aplicação PHP foi hospedada no *Github*. Nesse cenário, por meio do Centro de Implantação disponibilizado no *Azure Application Service*, é possível realizar a ligação entre o ambiente da Azure e o Github. Através dessa ligação, é possível realizar o *deploy* da aplicação por meio de pipelines do *Github Actions*, orquestradas automaticamente, que permitem o CI/CD. A Figura 43 demonstra o *deploy* da aplicação PHP.

Figura 43 – Demonstração do *deploy* PHP através do *Github Actions*



Fonte: Elaborado pelo autor

A pilha de *runtime* para hospedagem de aplicações PHP no *Azure Application Service* utiliza o *Linux* como sistema operacional e o *Nginx* como *proxy* reverso, recebendo as requisições e servindo os conteúdos estáticos. Entretanto, a configuração do `Nginx` por padrão necessita de alguns ajustes. A Figura 44 aponta as principais alterações, sendo especificamente no caminho do *root*, *server name* e *location*.

Figura 44 – Configuração do *Nginx* na Azure

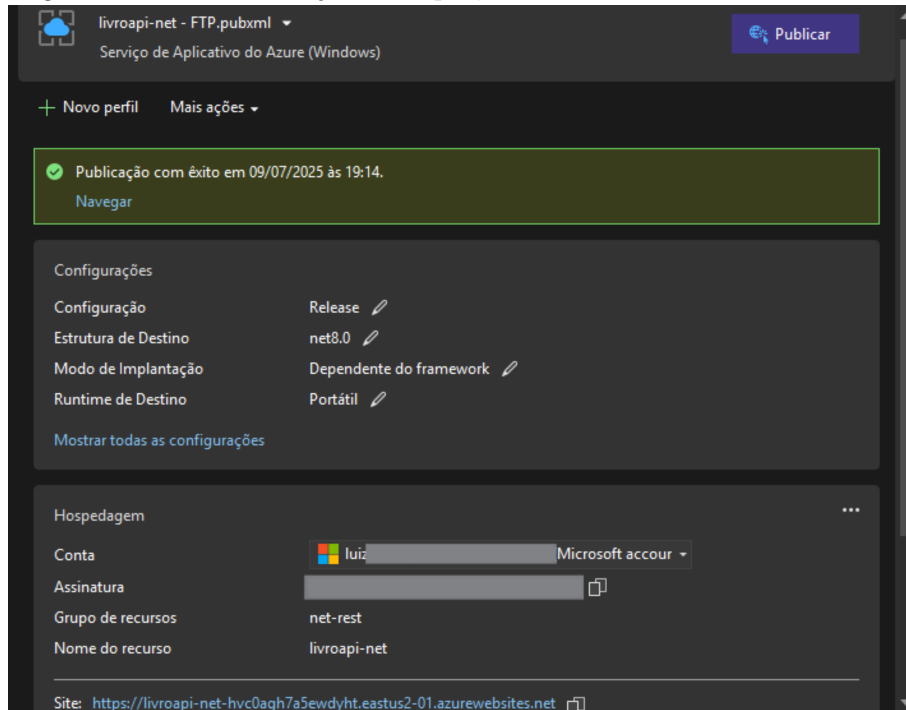
```
server {
    listen 8080;
    listen [::]:8080;
    root /home/site/wwwroot/public;
    index  index.php index.html index.htm;
    server_name livro-rest-hqbzcrdmcgdjh3fp.eastus2-01.azurewebsites.net;
    port_in_redirect off;

    location / {
        index  index.php index.html index.htm hostingstart.html;
        try_files $uri $uri/ /index.php?$query_string;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /html/;
    }

    location ~ /\.git {
        deny all;
        access_log off;
        log_not_found off;
    }

    location ~* \.php$ {

        fastcgi_pass 127.0.0.1:9000;
        include fastcgi_params;

        fastcgi_param HTTP_PROXY "";
        fastcgi_param SCRIPT_FILENAME $request_filename;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param QUERY_STRING $query_string;
        fastcgi_intercept_errors on;
        fastcgi_connect_timeout 300;
≡ Menu | ssh://root@169.254.130.3:2222 | SSH CONNECTION ESTABLISHED
```

Fonte: Elaborado pelo autor

A Microsoft disponibiliza uma maneira mais rápida para o *deploy* de aplicações es-

critas em *.NET*, permitindo que tal ação seja realizada através da própria *Integrated Development Environment (IDE)* do *Visual Studio* por meio da opção *Publish*, na qual realiza a construção e empacotamento e publicação da aplicação no perfil do ambiente da *Azure Application Service*. A Figura 45 demonstra o *deploy* da aplicação .NET através do *Visual Studio* pelo método de publicação.

Figura 45 – Demonstração do *deploy* .NET através do *Visual Studio*



Fonte: Elaborado pelo autor

A pilha de *runtime* escolhida para execução da aplicação .NET foi a .NET 8.0, com utilização do sistema operacional Windows. Após o *deploy* , nenhuma configuração extra foi necessária, de forma que, a publicação por meio do *Visual Studio* preparou todo o ambiente de forma automatizada.

6.4.2 *Deploy das aplicações no ambiente AWS*

No ambiente da AWS, o *deploy* foi realizado utilizando o *Elastic Beanstalk*, um modelo disponibilizado pela AWS, dessa forma, semelhantemente ao *Azure Application Service*, o PaaS da AWS abstrai toda a infraestrutura durante o *deploy*.

No *Elastic Beanstalk* foi utilizado o plano `t3.small`, plano este que padroniza a *baseline* de processamento dos núcleos em 20%. Todavia, utiliza-se créditos que podem acumular quando o processamento da máquina é menor ou igual a *baseline*. De modo que,

em um cenário onde uma aplicação necessita de mais capacidade de processamento devido a oscilações constantes, os créditos obtidos são utilizados. Uma vez que há o esgotamento desses créditos, a aplicação só poderá contar com a capacidade de *baseline* de processamento de núcleo da máquina.

A Figura 46 demonstra o plano utilizado no *Elastic Beanstalk*, plano que contém 2 núcleos de processamento, com *baseline*, e 2 GB de memória.

Figura 46 – Plano utilizado no *Elastic Beanstalk*

Nome	vCPUs	Memória (GiB)	Performance de linha de base por vCPU	Créditos de CPU obtidos por hora	Largura de banda para expansão de rede (Gbps)	Largura de banda para expansão do EBS (Mbps)	Preço sob demanda por hora*	Instância reservada por 1 ano – por hora*	Instância reservada por 3 anos efetiva por hora*
t3.nano	2	0,5	5%	6	5	Até 2.085	USD 0,0052	USD 0,003	USD 0,002
t3.micro	2	1,0	10%	12	5	Até 2.085	USD 0,0104	USD 0,006	USD 0,005
t3.small	2	2,0	20%	24	5	Até 2.085	USD 0,0209	USD 0,012	USD 0,008
t3.medium	2	4,0	20%	24	5	Até 2.085	USD 0,0418	USD 0,025	USD 0,017
t3.large	2	8,0	30%	36	5	Até 2.780	USD 0,0835	USD 0,05	USD 0,036
t3.xlarge	4	16,0	40%	96	5	Até 2.780	USD 0,1670	USD 0,099	USD 0,067
t3.2xlarge	8	32,0	40%	192	5	Até 2.780	USD 0,3341	USD 0,199	USD 0,133

Fonte: Elaborado pelo autor

Após a seleção do plano, houve a escolha da pilha de *runtime* para aplicações PHP no ambiente da AWS. Após a configuração do ambiente, o *deploy* é realizado de maneira mais otimizada, permitindo o *upload* da aplicação compactada no próprio ambiente, de modo que, toda a configuração posterior é orquestrada de forma automatizada pelo *Elastic Beanstalk*.

A pilha de *runtime* para aplicações PHP no *Elastic Beanstalk* utiliza o sistema operacional *Linux* e o *Nginx* como *proxy* reverso. Todavia, semelhantemente ao problema encontrado na *Azure Application Service*, o *Nginx* do *Elastic Beanstalk* não vem configurado corretamente, gerando problemas para acessar as rotas da aplicação, dessa forma, uma configuração manual através do console *ssh* é necessária para manter a aplicação com as rotas acessíveis. A Figura 47 aponta as principais alterações necessárias no arquivo `nginx.conf` da máquina EC2 na AWS.

Figura 47 – Configuração do *Nginx* na AWS

```
include /etc/nginx/conf.d/*.conf;

server {
    listen       80;
    server_name  localhost;

    root   /var/app/current/public;
    index  index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include        fastcgi_params;
        fastcgi_pass   unix:/var/run/php-fpm/www.sock;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }

       location ~ /\.(env|git) {
       deny all;
    }

    client_max_body_size 50M;
    client_body_timeout 120s;
  }
}
-- INSERT --
```

Fonte: Elaborado pelo autor

A pilha de *runtime* utilizada no ambiente para hospedagem da aplicação .NET utiliza o sistema operacional *Linux* também utilizando o *Nginx* como *proxy* reverso. O *deploy* da aplicação .NET no ambiente AWS foi realizado de forma otimizada, apenas com o *upload* do arquivo compactado contendo toda a aplicação. Entretanto, a utilização do comando `dotnet publish -c Release -o ./publish` para a construção e empacotamento da aplicação em um diretório é necessária. Dessa forma, após a construção da aplicação, o arquivo foi enviado para o ambiente, sem a necessidade de configurações adicionais.

6.5 Avaliação das aplicações nos ambientes

Nesta seção, as ferramentas *JMeter* e *K6* foram utilizadas para a confecção e execução dos testes de cargas nas aplicações hospedadas nos ambientes da AWS e Azure de forma que as métricas extraídas são essenciais para a síntese dos resultados.

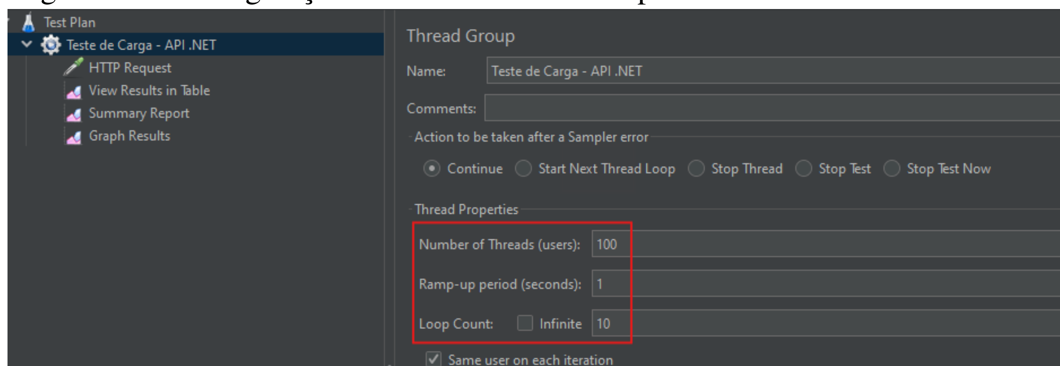
Inicialmente, as métricas mais importantes e que são os pilares dos testes de cargas do presente estudo são: *throughput*, tal métrica tem como objetivo medir a capacidade bruta do serviço medindo a vazão, ou seja, quantas requisições um serviço pode executar por segundo.

Mediana e latência média, indicando o tempo médio das respostas. Percentis de latência, tais como P(95) e P(99), indicam como se comportam as piores 5% ou 1% das requisições, e são essenciais para identificar atrasos. Dessa forma, com a obtenção de tais métricas é possível estimar os ambientes que melhor apresentam resultados baseado nos testes de cargas.

6.5.1 Testes de carga com a ferramenta JMeter na aplicação .NET

Por se tratar de uma aplicação CRUD, os testes de cargas foram efetuados nos formatos apenas de leitura e escrita, utilizando os métodos `GET` e `POST` com o objetivo de extrair resultados mais consolidados. A Figura 48 demonstra os parâmetros utilizados na ferramenta *JMeter* para os testes de cargas tanto no ambiente da Azure quanto na AWS.

Figura 48 – Configuração da ferramenta *JMeter* para o método GET

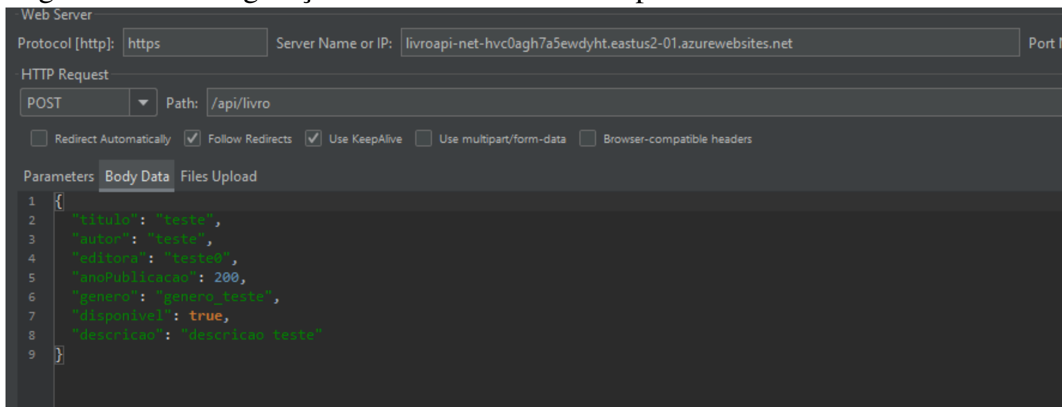


Fonte: Elaborado pelo autor

Conforme o que é apresentado na Figura 48, a primeira linha indica 100 usuários concorrentes disparando requisições ao mesmo tempo. A segunda linha representa a inicialização das *threads*, onde ao longo de 1 segundo ocorre a inicialização das 100 *threads*. Por fim, a terceira linha indica a repetição do *script* 10 vezes, totalizando `100 x 10 = 1.000` requisições. A Figura também demonstra os *listeners*, responsáveis por capturar os resultados durante a execução dos testes de cargas, sendo eles: *Summary Report* (Relatório Resumido), *Graph Results* (Resultados em Gráfico) e *View Results in Table* (Visualizar Resultados em Tabela). Entretanto, para este estudo analisaremos apenas os dois primeiros *listeners*.

Para o método POST foi utilizado um JSON. Entretanto, o número de usuários concorrentes foi reduzido para 50, e o número de repetições foi reduzido para 5, totalizando `50 x 5 = 250` requisições de escrita na aplicação. A Figura 49 apresenta o JSON enviado durante as requisições.

Figura 49 – Configuração da ferramenta *JMeter* para o método POST



Fonte: Elaborado pelo autor

6.5.1.1 *Testes de cargas com método GET*

O primeiro teste de carga foi através do método GET, realizando a listagem de aproximadamente 250 linhas do banco de dados *MySQL* na aplicação hospedada no ambiente da *Azure Application Service*. A Figura 50 apresenta o *Summary Report* dos resultados obtidos após o estresse da aplicação.

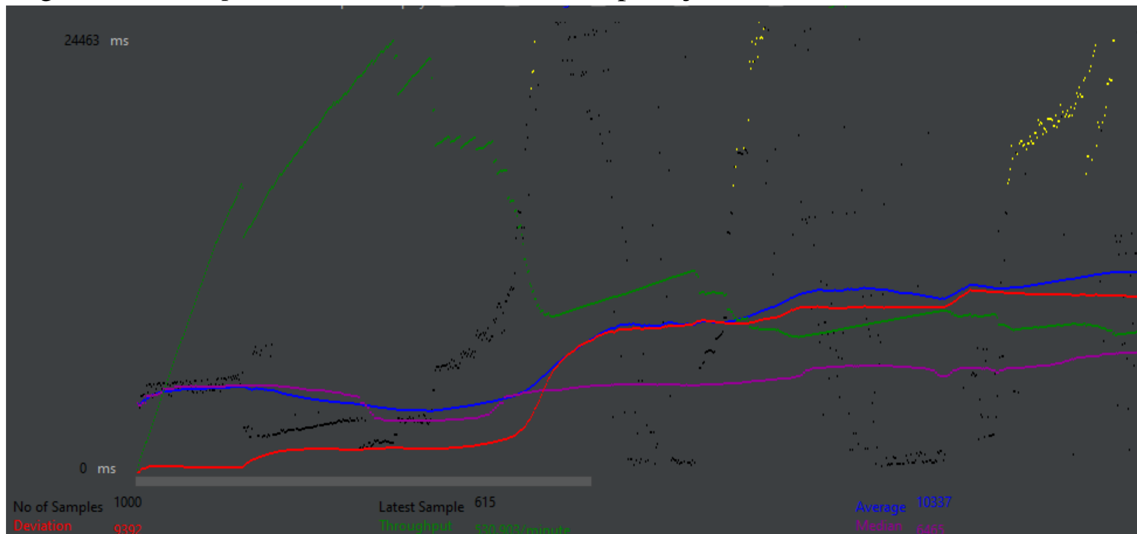
Figura 50 – *Summary Report* do método GET na aplicação .NET na Azure.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	10337	169	37993	9392.81	18.90%	8.8/sec	277.50	1.49	32114.7
TOTAL	1000	10337	169	37993	9392.81	18.90%	8.8/sec	277.50	1.49	32114.7

Fonte: Elaborado pelo autor

A Figura 51 aponta o gráfico obtido no momento da execução do teste de carga. O gráfico aponta a mediana, uma métrica importante representada pela linha roxa, que ficou em cerca de `6.465 ms`, ou seja, 50% das requisições tiveram tempo menor ou igual a esse.

Figura 51 – *Graph Result* do método GET na aplicação .NET na Azure.



Fonte: Elaborado pelo autor

O segundo teste de carga foi realizado no ambiente da AWS, utilizando os mesmos parâmetros de configuração apresentados na Figura 48. A Figura 52 apresenta o *Summary Report* contendo as métricas obtidas no ambiente AWS.

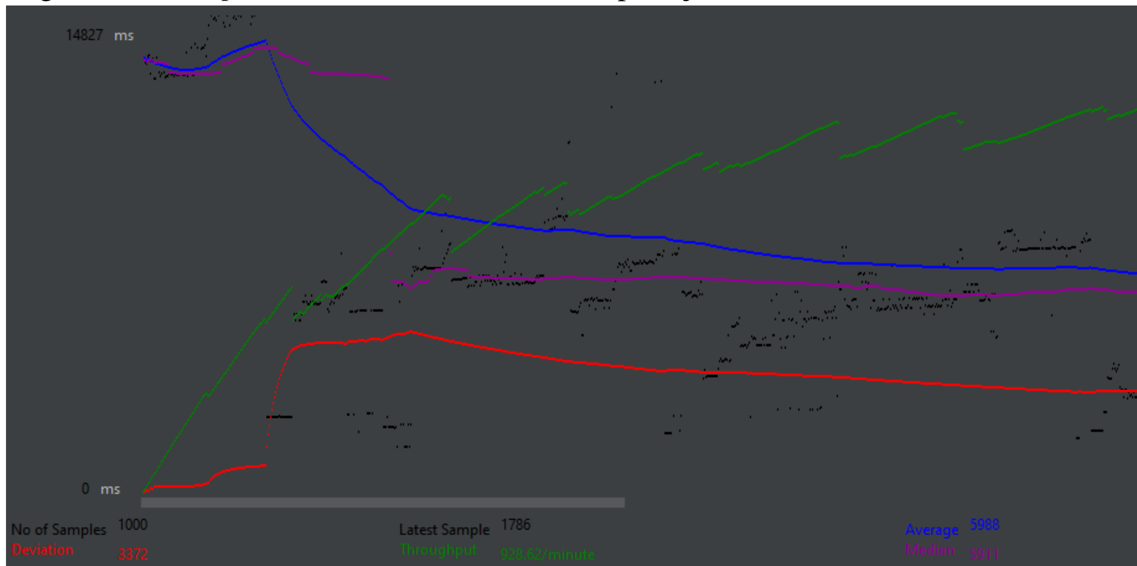
Figura 52 – *Summary Report* do método GET na aplicação .NET na AWS.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	5988	1420	15700	3372.54	0.00%	15.5/sec	587.82	2.37	38891.5
TOTAL	1000	5988	1420	15700	3372.54	0.00%	15.5/sec	587.82	2.37	38891.5

Fonte: Elaborado pelo autor

A Figura 53 apresenta o *Graph Result* obtido durante a execução dos testes na AWS. A mediana ficou em torno de `5.911 ms`.

Figura 53 – *Graph Result* do método GET na aplicação .NET na AWS.



Fonte: Elaborado pelo autor

6.5.1.2 *Testes de cargas com método POST*

Conforme abordado no início da seção, no quesito de escrita, foi utilizado o método `POST`. No ambiente da Azure, a Figura 54 apresenta o *Summary Report* após a execução dos testes na aplicação .NET na Azure.

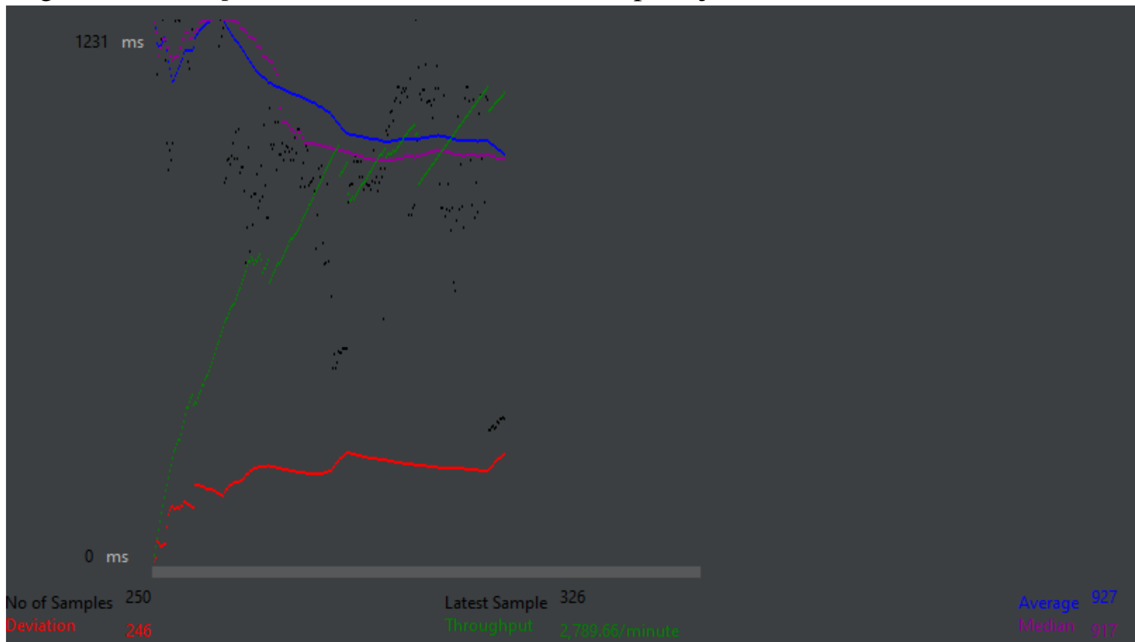
Figura 54 – *Summary Report* do método POST na aplicação .NET na Azure.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	250	927	301	1884	246.31	0.00%	46.5/sec	39.96	18.57	880.0
TOTAL	250	927	301	1884	246.31	0.00%	46.5/sec	39.96	18.57	880.0

Fonte: Elaborado pelo autor

A Figura 55 apresenta o *Graph Result* do momento da execução do teste. A mediana ficou em torno de `917 ms`.

Figura 55 – *Graph Result* do método POST na aplicação .NET na Azure.



Fonte: Elaborado pelo autor

O segundo teste de carga foi realizado no ambiente da AWS, com os mesmos parâmetros de configuração apresentado no início da seção. A Figura 56 presenta o Summary Report após a execução dos testes na aplicação .NET na AWS.

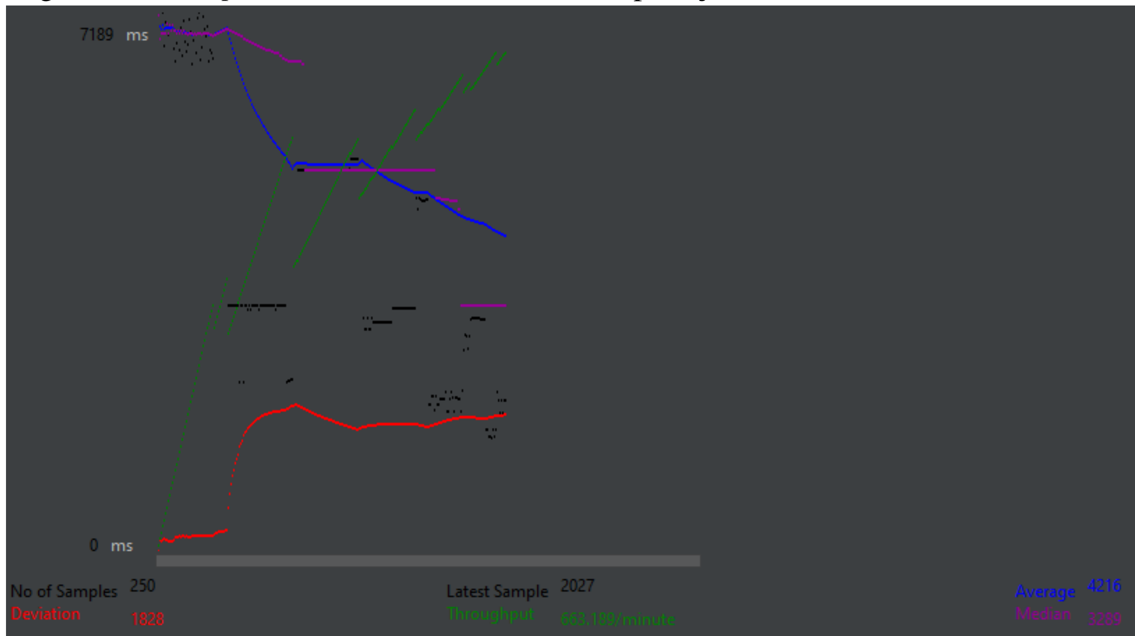
Figura 56 – *Summary Report* do método POST na aplicação .NET na AWS.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	250	4216	1521	7970	1828.95	0.00%	11.1/sec	3.86	4.25	358.0
TOTAL	250	4216	1521	7970	1828.95	0.00%	11.1/sec	3.86	4.25	358.0

Fonte: Elaborado pelo autor

A Figura 57 apresenta o *Graph Result* do momento da execução do teste. A mediana ficou em torno de `3.289 ms`. O fato da mediana ter ficado mais alta pode ter relação com a *baseline* da AWS, que pode ter limitado a utilização de CPU.

Figura 57 – *Graph Result* do método POST na aplicação .NET na AWS.



Fonte: Elaborado pelo autor

6.5.2 Testes de carga com a ferramenta JMeter na aplicação PHP

Nos testes para a a aplicação PHP, foram utilizados os mesmos parâmetros apresentados no início da seção para a aplicação .NET.

6.5.2.1 Testes de cargas com método GET

A Figura 58 apresenta o *Summary Report* obtido após a execução dos testes de cargas na aplicação PHP no ambiente da Azure.

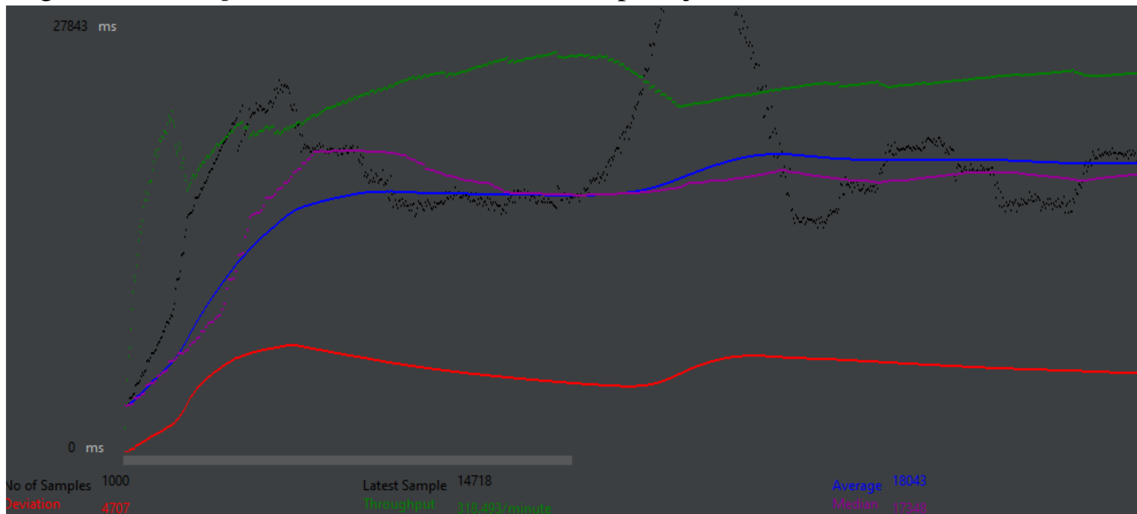
Figura 58 – *Summary Report* do método GET na aplicação PHP na Azure.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	18043	2920	31359	4707.87	0.00%	5.3/sec	371.28	1.77	71622.2
TOTAL	1000	18043	2920	31359	4707.87	0.00%	5.3/sec	371.28	1.77	71622.2

Fonte: Elaborado pelo autor

A Figura 59 apresenta o *Graph Result* obtido durante a execução da aplicação PHP no ambiente da Azure. A mediana ficou em torno de `17.348 ms`

Figura 59 – *Graph Result* do método GET na aplicação PHP na Azure.



Fonte: Elaborado pelo autor

A Figura 64 apresenta o *Summary Report* obtido após a execução dos testes de cargas na aplicação PHP no ambiente da AWS.

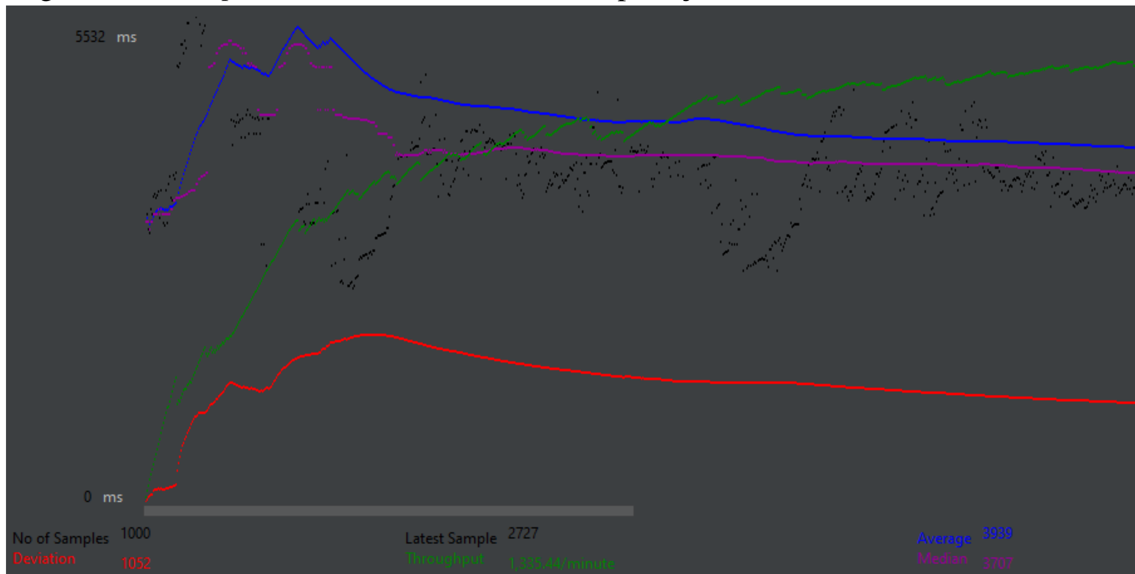
Figura 60 – *Summary Report* do método GET na aplicação PHP na AWS.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	3939	2443	9177	1052.29	0.00%	22.3/sec	1551.54	3.85	71382.0
TOTAL	1000	3939	2443	9177	1052.29	0.00%	22.3/sec	1551.54	3.85	71382.0

Fonte: Elaborado pelo autor

A Figura 61 apresenta o *Graph Result* obtido durante a execução da aplicação PHP no ambiente da Azure. A mediana ficou em torno de `3.707 ms`, bem menor do que observado no ambiente da Azure.

Figura 61 – *Graph Result* do método GET na aplicação PHP na AWS.



Fonte: Elaborado pelo autor

6.5.2.2 *Testes de cargas com método POST*

Os parâmetros para o teste de carga com o método POST foram os mesmos apresentados no inicio da Seção 6.5.1. Iniciando-se pelos resultados obtidos no ambiente da Azure, a Figura 62 apresenta o *Summary Report* obtido ao final do teste de carga.

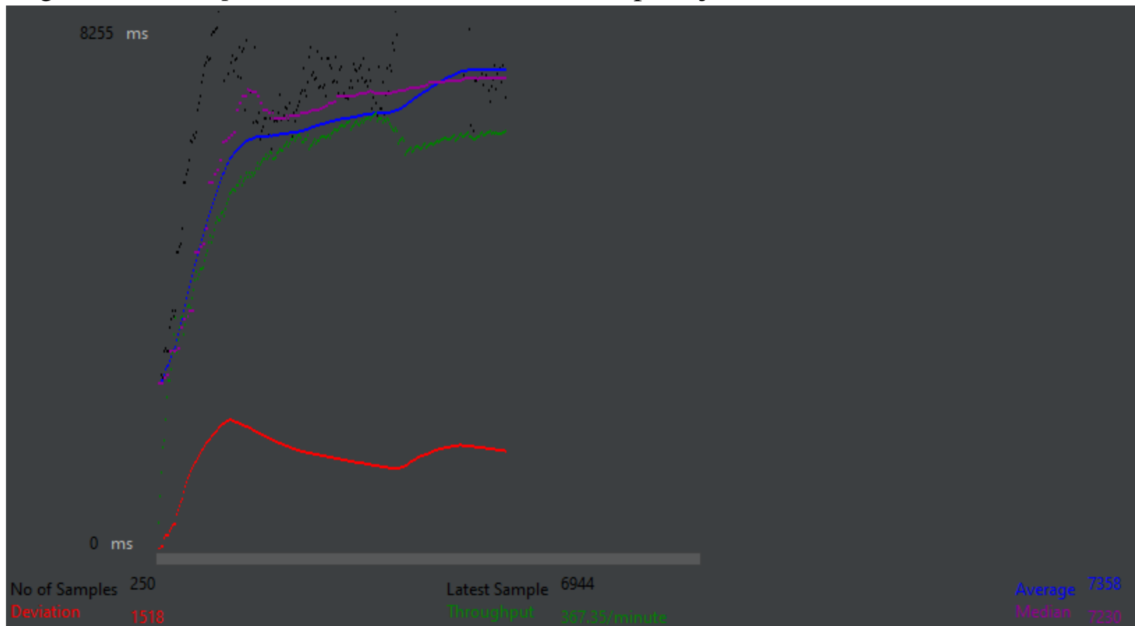
Figura 62 – *Summary Report* do método POST na aplicação PHP na Azure.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	250	7358	2545	10185	1518.42	0.00%	6.1/sec	428.21	3.53	71618.5
TOTAL	250	7358	2545	10185	1518.42	0.00%	6.1/sec	428.21	3.53	71618.5

Fonte: Elaborado pelo autor

A Figura 63 apresenta o *Graph Result* obtido durante a execução da aplicação PHP com o método POST no ambiente da Azure. A mediana ficou em torno de `7.230 ms`.

Figura 63 – *Graph Result* do método POST na aplicação PHP na Azure.



Fonte: Elaborado pelo autor

Também para efeito de comparação, os mesmos testes de cargas foram efetuados na AWS. A Figura 64 apresenta o *Summary Report* obtido após a execução dos testes.

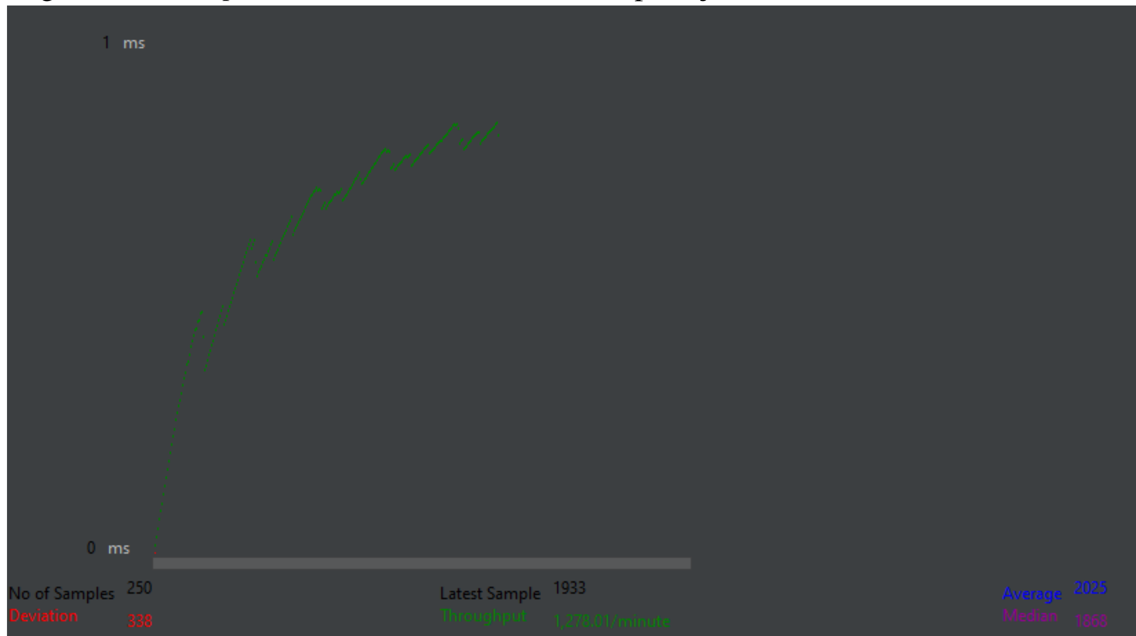
Figura 64 – *Summary Report* do método POST na aplicação PHP na AWS.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	3939	2443	9177	1052.29	0.00%	22.3/sec	1551.54	3.85	71382.0
TOTAL	1000	3939	2443	9177	1052.29	0.00%	22.3/sec	1551.54	3.85	71382.0

Fonte: Elaborado pelo autor

A Figura 65 demonstra o *Graph Result* obtido durante as requisições de escrita no ambiente da AWS. A mediana ficou em torno de `1.868 ms`, abaixo do obtido na Azure.

Figura 65 – *Graph Result* do método POST na aplicação PHP na AWS.



Fonte: Elaborado pelo autor

6.5.3 Testes de carga com a ferramenta K6 na aplicação .NET

Os testes elaborados com a ferramenta *K6* agem de forma diferente dos testes efetuados no *JMeter*. De modo que, enquanto no *JMeter* 100 *threads* são inicializadas de forma instântanea e a distribuição de *loops* para cada *thread* é exatamente 10, no *script*[3] do GET do *K6* definimos `vus:100`, que são corrotinas da linguagem *Go*, e `iterations:1000`, de modo que 1.000 iterações são compartilhadas entre os 100 Virtual User (VU)s, resultando emVUs com contagens de loops variáveis (por exemplo, 8, 10, 12 iterações).

A importância de testar iterações compartilhadas é interessante para determinar qual ambiente consegue lidar melhor com cargas diferentes, ou seja, qual ambiente lida melhor com cenários onde a padronização não é um requisito.

Por outro lado, o *script*[4] utilizado no teste de carga para o método POST atua de forma diferente, inicialmente, há a utilização de 30 VUs ao longo dos 10 primeiros segundos, posteriormente, essa quantidade de VUs é reduzida a 20 durante 10 segundos e por último é reduzida a 5 VUs nos 10 segundos finais. Essa subida e redução de VUs é interessante para simular um comportamento normal de usuários ao longo do dia em uma aplicação.

A Figura 66 demonstra os resultados obtidos após a execução dos testes na aplicação

[3] Disponível em: https://github.com/luizenrike/k6_scripts/blob/master/k6_get_script.js. Acesso em: 15 jul. 2025.
[4] Disponível em: https://github.com/luizenrike/k6_scripts/blob/master/k6_post_script.js. Acesso em: 15 jul. 2025.

.NET no ambiente da Azure. No teste, 79,8% (798 de 1000) das requisições foram realizadas com sucesso, enquanto 20,2% (202 de 1000) falharam. A mediana foi de 8,69 segundos.

Figura 66 – Resultado do método GET na aplicação .NET no ambiente da Azure

```
     scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1000 iterations shared among 100 VUs (maxDuration: 10m0s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=32.28s


  █ TOTAL RESULTS

    checks_total.......................: 1000    7.265279/s
    checks_succeeded...................: 79.80% 798 out of 1000
    checks_failed......................: 20.20% 202 out of 1000

    ✗ status foi 200
      ↳  79% — ✓ 798 / ✗ 202

    HTTP
    http_req_duration.......................................................: avg=12.53s min=311.86ms med=8.69s max=42.9s  p(90)=25.81s p(95)=28.64
s
      { expected_response:true }............................................: avg=9.78s  min=311.86ms med=6.56s max=32.57s p(90)=24.82s p(95)=26.85
s
    http_req_failed.........................................................: 20.20% 202 out of 1000
    http_reqs...............................................................: 1000   7.265279/s

    EXECUTION
    iteration_duration......................................................: avg=12.69s min=312.38ms med=8.69s max=42.9s  p(90)=25.81s p(95)=28.64
s
    iterations..............................................................: 1000   7.265279/s
    vus.....................................................................: 2      min=2          max=100
    vus_max.................................................................: 100    min=100        max=100

    NETWORK
    data_received...........................................................: 33 MB  237 kB/s
    data_sent...............................................................: 226 kB 1.6 kB/s



running (02m17.6s), 000/100 VUs, 1000 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  02m17.6s/10m0s  1000/1000 shared iters
ERRO[0137] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

A Figura 67 apresenta os resultados do teste da ferramenta *K6* obtidos na Azure realizando operações de escrita através do método POST. Foram realizadas cerca de 332 requisições. O tempo médio de resposta foi `684 ms`, enquanto a mediana foi de `205,22 ms`, entretanto o p(99), isto é, o 99o percentil, foi de `4,94` segundos.

Figura 67 – Resultado do método POST na aplicação .NET no ambiente da Azure

```
     scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
              * default: Up to 30 looping VUs for 30s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=4.94s


  █ TOTAL RESULTS

    checks_total.......................: 664     21.573912/s
    checks_succeeded...................: 100.00% 664 out of 664
    checks_failed......................: 0.00%   0 out of 664

    ✓ status foi 200 ou 201
    ✓ retornou corpo

    HTTP
    http_req_duration.......................................................: avg=684.49ms min=164.15ms med=205.22ms max=7.7s p(90)=2.47s p(95)=4.2
7s
      { expected_response:true }............................................: avg=684.49ms min=164.15ms med=205.22ms max=7.7s p(90)=2.47s p(95)=4.2
7s
    http_req_failed.........................................................: 0.00%  0 out of 332
    http_reqs...............................................................: 332    10.786956/s

    EXECUTION
    iteration_duration......................................................: avg=1.72s    min=1.16s    med=1.21s    max=8.7s p(90)=3.47s p(95)=5.2
7s
    iterations..............................................................: 332    10.786956/s
    vus.....................................................................: 7      min=3         max=30
    vus_max.................................................................: 30     min=30        max=30

    NETWORK
    data_received...........................................................: 514 kB 17 kB/s
    data_sent...............................................................: 152 kB 4.9 kB/s




running (0m30.8s), 00/30 VUs, 332 complete and 0 interrupted iterations
default ✓ [======================================] 00/30 VUs  30s
ERRO[0031] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

Na AWS o cenário foi diferente. A Figura 68 apresenta os resultados obtidos nos testes. Conforme os resultados, 100% das requisições foram respondidas com sucesso, a média foi de `4,96` segundos e a mediana foi de `4,61` segundos.

Figura 68 – Resultado do método GET na aplicação .NET no ambiente da AWS

```
     scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1000 iterations shared among 100 VUs (maxDuration: 10m0s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=13.72s


  █ TOTAL RESULTS

    checks_total.......................: 1000    19.644407/s
    checks_succeeded...................: 100.00% 1000 out of 1000
    checks_failed......................: 0.00%   0 out of 1000

    ✓ status foi 200

    HTTP
    http_req_duration.......................................................: avg=4.96s min=1.32s med=4.61s max=15.05s p(90)=9.63s p(95)=11.79s
      { expected_response:true }............................................: avg=4.96s min=1.32s med=4.61s max=15.05s p(90)=9.63s p(95)=11.79s
    http_req_failed.........................................................: 0.00%  0 out of 1000
    http_reqs...............................................................: 1000   19.644407/s

    EXECUTION
    iteration_duration......................................................: avg=5s    min=1.32s med=4.61s max=15.49s p(90)=9.67s p(95)=12.22s
    iterations..............................................................: 1000   19.644407/s
    vus.....................................................................: 49     min=49        max=100
    vus_max.................................................................: 100    min=100       max=100

    NETWORK
    data_received...........................................................: 39 MB  764 kB/s
    data_sent...............................................................: 108 kB 2.1 kB/s




running (00m50.9s), 000/100 VUs, 1000 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  00m50.9s/10m0s  1000/1000 shared iters
ERRO[0051] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

A Figura 69 demonstra os resultados obtidos na AWS através do método POST. A média foi de 2,67 segundos e a mediana de 2,5 segundos, entretanto, apenas 156 requisições foram realizadas.

Figura 69 – Resultado do método POST na aplicação .NET no ambiente da AWS

```
     scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
              * default: Up to 30 looping VUs for 30s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=5.77s


  █ TOTAL RESULTS

    checks_total.......................: 312     9.832195/s
    checks_succeeded...................: 100.00% 312 out of 312
    checks_failed......................: 0.00%   0 out of 312

    ✓ status foi 200 ou 201
    ✓ retornou corpo

    HTTP
    http_req_duration.......................................................: avg=2.67s min=1.38s med=2.5s  max=5.83s p(90)=4.48s p(95)=5.29s
      { expected_response:true }............................................: avg=2.67s min=1.38s med=2.5s  max=5.83s p(90)=4.48s p(95)=5.29s
    http_req_failed.........................................................: 0.00% 0 out of 156
    http_reqs...............................................................: 156   4.916097/s

    EXECUTION
    iteration_duration......................................................: avg=3.74s min=2.38s med=3.52s max=6.83s p(90)=5.59s p(95)=6.43s
    iterations..............................................................: 156   4.916097/s
    vus.....................................................................: 6     min=3        max=30
    vus_max.................................................................: 30    min=30       max=30

    NETWORK
    data_received...........................................................: 62 kB 1.9 kB/s
    data_sent...............................................................: 54 kB 1.7 kB/s




running (0m31.7s), 00/30 VUs, 156 complete and 0 interrupted iterations
default ✓ [======================================] 00/30 VUs  30s
ERRO[0032] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

6.5.4 *Testes de carga com a ferramenta K6 na aplicação PHP*

Os parâmetros e *script* utilizados foram os mesmos utilizados para a aplicação .NET. A Figura 70 apresenta os resultados obtidos durante o teste no método GET. A média foi de 20,14 segundos e a mediana foi de 20,26 segundos.

Figura 70 – Resultado do método GET na aplicação PHP no ambiente da Azure

```
     scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1000 iterations shared among 100 VUs (maxDuration: 10m0s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=25.76s


  █ TOTAL RESULTS

    checks_total.......................: 1000    4.726378/s
    checks_succeeded...................: 100.00% 1000 out of 1000
    checks_failed......................: 0.00%   0 out of 1000

    ✓ status foi 200

    HTTP
    http_req_duration.......................................................: avg=20.14s min=1.91s med=20.26s max=29.96s p(90)=23.46s p(95)=24.57s
      { expected_response:true }............................................: avg=20.14s min=1.91s med=20.26s max=29.96s p(90)=23.46s p(95)=24.57s
    http_req_failed.........................................................: 0.00%  0 out of 1000
    http_reqs...............................................................: 1000   4.726378/s

    EXECUTION
    iteration_duration......................................................: avg=20.26s min=2.7s  med=20.29s max=29.96s p(90)=23.56s p(95)=24.98s
    iterations..............................................................: 1000   4.726378/s
    vus.....................................................................: 5      min=5         max=100
    vus_max.................................................................: 100    min=100       max=100

    NETWORK
    data_received...........................................................: 97 MB  456 kB/s
    data_sent...............................................................: 225 kB 1.1 kB/s




running (03m31.6s), 000/100 VUs, 1000 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  03m31.6s/10m0s  1000/1000 shared iters
ERRO[0212] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

A Figura 71 apresenta os resultados obtidos após os testes no método POST. A média ficou em 649 ms e a mediana em 466 ms.

Figura 71 – Resultado do método POST na aplicação PHP no ambiente da Azure

```
     scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
              * default: Up to 30 looping VUs for 30s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✓ 'p(99) < 3000' p(99)=2.45s


  █ TOTAL RESULTS

    checks_total.......................: 660     21.018429/s
    checks_succeeded...................: 100.00% 660 out of 660
    checks_failed......................: 0.00%   0 out of 660

    ✓ status foi 200 ou 201
    ✓ retornou corpo

    HTTP
    http_req_duration.......................................................: avg=649.02ms min=199.59ms med=466.33ms max=2.57s p(90)=1.26s p(95)=1.
85s
      { expected_response:true }............................................: avg=649.02ms min=199.59ms med=466.33ms max=2.57s p(90)=1.26s p(95)=1.
85s
    http_req_failed.........................................................: 0.00%  0 out of 330
    http_reqs...............................................................: 330    10.509214/s

    EXECUTION
    iteration_duration......................................................: avg=1.69s    min=1.2s     med=1.5s     max=4.01s p(90)=2.38s p(95)=3.
07s
    iterations..............................................................: 330    10.509214/s
    vus.....................................................................: 5      min=3         max=30
    vus_max.................................................................: 30     min=30        max=30

     NETWORK
     data_received..........................................................: 535 kB 17 kB/s
     data_sent..............................................................: 151 kB 4.8 kB/s




running (0m31.4s), 00/30 VUs, 330 complete and 0 interrupted iterations
default ✓ [======================================] 00/30 VUs  30s

luizh@LAPTOP-6AAL6SGL MINGW64 ~/Desktop/figuras tcc/codigo_tcc (master)
$
```

Fonte: Elaborado pelo autor

No ambiente da AWS, para o teste no método GET, a média foi de 4,21 segundos, e a mediana foi de 3,94 segundos, demonstrando um melhor desempenho quando comparado com os resultados da Azure. A Figura 73 apresenta os resultados.

Figura 72 – Resultado do método GET na aplicação PHP no ambiente da AWS

```
     scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1000 iterations shared among 100 VUs (maxDuration: 10m0s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✗ 'p(99) < 3000' p(99)=10.46s


  █ TOTAL RESULTS

    checks_total.......................: 1000    21.107226/s
    checks_succeeded...................: 100.00% 1000 out of 1000
    checks_failed......................: 0.00%   0 out of 1000

    ✓ status foi 200

    HTTP
    http_req_duration.......................................................: avg=4.21s min=2.2s med=3.94s max=12.17s p(90)=5.41s p(95)=6.76s
      { expected_response:true }............................................: avg=4.21s min=2.2s med=3.94s max=12.17s p(90)=5.41s p(95)=6.76s
    http_req_failed.........................................................: 0.00%  0 out of 1000
    http_reqs...............................................................: 1000   21.107226/s

    EXECUTION
    iteration_duration......................................................: avg=4.26s min=2.2s med=3.95s max=12.17s p(90)=5.74s p(95)=7.12s
    iterations..............................................................: 1000   21.107226/s
    vus.....................................................................: 1      min=1          max=100
    vus_max.................................................................: 100    min=100        max=100

    NETWORK
    data_received...........................................................: 71 MB  1.5 MB/s
    data_sent...............................................................: 128 kB 2.7 kB/s




running (00m47.4s), 000/100 VUs, 1000 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  00m47.4s/10m0s  1000/1000 shared iters
ERRO[0048] thresholds on metrics 'http_req_duration' have been crossed
```

Fonte: Elaborado pelo autor

A Figura 73 apresenta os resultados obtidos no teste do método POST. A média foi de 1,85 segundos e a mediana foi de 1,84 segundos.

Figura 73 – Resultado do método POST na aplicação PHP no ambiente da AWS

```
    scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
             * default: Up to 30 looping VUs for 30s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  █ THRESHOLDS

    http_req_duration
    ✓ 'p(99) < 3000' p(99)=2.05s


  █ TOTAL RESULTS

    checks_total.......................: 402     12.212736/s
    checks_succeeded...................: 100.00% 402 out of 402
    checks_failed......................: 0.00%   0 out of 402

    ✓ status foi 200 ou 201
    ✓ retornou corpo

    HTTP
    http_req_duration.......................................................: avg=1.85s min=1.78s med=1.84s max=2.15s p(90)=1.88s p(95)=1.91s
      { expected_response:true }............................................: avg=1.85s min=1.78s med=1.84s max=2.15s p(90)=1.88s p(95)=1.91s
    http_req_failed.........................................................: 0.00%  0 out of 201
    http_reqs...............................................................: 201    6.106368/s

    EXECUTION
    iteration_duration......................................................: avg=2.89s min=2.79s med=2.85s max=3.56s p(90)=3.09s p(95)=3.11s
    iterations..............................................................: 201    6.106368/s
    vus.....................................................................: 4      min=3        max=30
    vus_max.................................................................: 30     min=30       max=30

    NETWORK
    data_received...........................................................: 198 kB 6.0 kB/s
    data_sent...............................................................: 74 kB  2.2 kB/s



running (0m32.9s), 00/30 VUs, 201 complete and 0 interrupted iterations
default ✓ [======================================] 00/30 VUs  30s
```

Fonte: Elaborado pelo autor

6.6 Síntese dos resultados

Nesta seção, a síntese dos resultados é realizada utilizando os resultados obtidos nos testes da Seção 6.5, portanto a definição do melhor ambiente é baseada nas métricas.

O Quadro 12 reúne as principais métricas obtidas nos testes. Para requisições de listagem com método GET em aplicações .NET, o ambiente da AWS apresentou melhores resultados.

Quadro 12 – Comparação de desempenho do método *GET* na API .NET entre Azure B2 e AWS t3.small usando *JMeter*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg (ms)	10 337	5 988	AWS melhor
Mediana (ms)	6 465	5 911	AWS melhor
Throughput (req/s)	8,8	15,5	AWS $\approx$1,8× mais alto
% de sucesso	81,1%	100%	AWS mais confiável

Fonte: Elaborado pelo autor

O Quadro 13 reúne as métricas em ambos os ambientes para o método de escrita, POST. Conforme as métricas obtidas, a Azure obteve os melhores resultados, e isso pode ser caracterizado por alguns fatores como a CPU inteiramente dedicada do plano B2 da Azure.

Quadro 13 – Comparação de desempenho do método *POST* na API .NET entre Azure B2 e AWS t3.small usando *JMeter*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg (ms)	927	4216	Azure melhor
Mediana (ms)	917	3289	Azure melhor
Throughput (req/s)	46,5	11,1	Azure ≈4,2× mais alto
% de sucesso	100%	100%	empate (sem falhas)

Fonte: Elaborado pelo autor

Comparando os Quadros 14 e 15 o cenário é o mesmo, a AWS apresenta melhores resultados em operações de leitura, enquanto a Azure apresenta melhores resultados em operações de escrita.

Quadro 14 – Comparação de desempenho do método *GET* na API PHP entre Azure B2 e AWS t3.small usando *JMeter*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg (ms)	18 043	3 939	AWS melhor
Mediana (ms)	17 348	3 707	AWS melhor
p(99) (ms)	31 359	9 177	AWS melhor
Throughput (req/s)	5,3	22,3	AWS ≈4× mais alto
% de sucesso	100%	100%	empate (sem falhas)

Fonte: Elaborado pelo autor

Quadro 15 – Comparação de desempenho do método *POST* na API PHP entre Azure B2 e AWS t3.small usando *JMeter*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg (ms)	7 358	2 025	Azure melhor
Mediana (ms)	7 230	1 868	Azure melhor
Throughput (req/s)	6,1	21,3	AWS ≈3,5× mais alto
% de sucesso	100%	100%	empate (sem falhas)

Fonte: Elaborado pelo autor

Os Quadros 16 e 17 apresentam os resultados obtidos nas operações de leitura e escrita da aplicação .NET com a ferramenta K6. O cenário é semelhante aos resultados do JMeter, onde as operações de leitura obtiveram melhores desempenhos na AWS enquanto as de escrita na Azure.

Quadro 16 – Comparação de desempenho do método *GET* da API .NET entre Azure B2 e AWS t3.small usando *K6*.

Métrica	**Azure B2 (GET)**	**AWS t3.small (GET)**	**Conclusão**
Avg *http_req_duration*	12,53s	4,96s	AWS melhor
Mediana (med)	8,69s	4,61s	AWS melhor
p(99)	32,28 s	13,72 s	AWS melhor
Throughput (req/s)	7,27	19,64	AWS ≈3× mais alto
% de sucesso	79,8%	100%	AWS mais confiável

Fonte: Elaborado pelo autor

Quadro 17 – Comparação de desempenho do método *POST* da API .NET entre Azure B2 e AWS t3.small usando *K6*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg *http_req_duration*	684 ms	2 670 ms	Azure melhor
Mediana (med)	205 ms	2 500 ms	Azure melhor
p(99)	4,94 s	5,77 s	Azure melhor
Throughput (req/s)	10,79	4,92	Azure ≈ 2,2× mais alto
% de sucesso	100 %	100 %	empate (sem falhas)

Fonte: Elaborado pelo autor

Os Quadros 18 e 19 apresentam os resultados e comparação dos testes na aplicação PHP em ambos ambientes. O cenário é semelhante, a AWS apresenta-se melhor nas operações de leitura enquanto a Azure apresenta melhores resultados nas operações de escrita.

Quadro 18 – Comparação de desempenho do método *GET* na API PHP entre Azure B2 e AWS t3.small usando *K6*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg *http_req_duration*	20,14s	4,21s	AWS melhor
Mediana (50º perc.)	20,26s	3,94s	AWS melhor
p(99)	25,76s	10,46s	AWS melhor
Throughput (req/s)	4,73	21,11	AWS ≈4,5× mais alto
% de sucesso	100%	100%	empate (ambos sem falhas)

Fonte: Elaborado pelo autor

Quadro 19 – Comparação de desempenho do método *POST* na API PHP entre Azure B2 e AWS t3.small usando *K6*.

Métrica	**Azure B2**	**AWS t3.small**	**Conclusão**
Avg *http_req_duration*	649ms	1,85s	Azure melhor
Mediana (50º perc.)	466ms	1,84s	Azure melhor
p(99)	2,05s	2,05s	empate
Throughput (req/s)	10,51	6,10	Azure 1,7× mais alto
% de sucesso	100%	100%	empate (sem falhas)

Fonte: Elaborado pelo autor

Nesse sentido, para os ambientes selecionados, cada um se destaca melhor em uma situação, não há campeão, apenas um ambiente que possui uma infraestrutura melhor para um determinado tipo de operação. Entretanto, a AWS se destaca por ofertar a mesma quantidade de núcleos, embora a memória seja mais baixa, por um preço bem menor do que a Azure, o Quadro 20 traz os custos de cada instância.

Quadro 20 – Comparação de custos entre *Azure App Service* B2 e *Elastic Beanstalk* t3.small.

Plano	**vCPUs**	**RAM**	**Custo/hora (US$/h)**	**Custo mensal (US$/mês)**
Azure App Service B2	2	3,50 GB	$\approx 0{,}15$	109,50
AWS EC2 t3.small	2	2 GB	0,0208	15,18

Fonte: Adaptado de (AWS, 2025) e (Microsoft, 2025)

Nesse sentido, ao considerar o uso das instâncias por um período de 12 dias para validar todos os testes, observa-se uma disparidade significativa nos custos entre as plataformas. Enquanto a Azure entrega mais memória no plano B2, seu custo é consideravelmente mais alto. A AWS, por sua vez, oferece o mesmo número de vCPUs por um preço muito inferior, ainda que com menor quantidade de RAM. O Quadro 21 apresenta os valores proporcionais para esse período, evidenciando a vantagem da AWS em termos de custo-benefício nesse cenário específico. Para efeitos de cálculo foi utilizado o válor do dólar em R$ 5,50[5].

Quadro 21 – Comparação de custos em 12 dias entre *Azure App Service* B2 e *Elastic Beanstalk* t3.small.

Plano	**vCPUs**	**RAM**	**Custo (US$ - 12 dias)**	**Custo (R$ - 12 dias)**
Azure App Service B2	2	3,50 GB	43,80	R$ 240,90
AWS EC2 t3.small	2	2 GB	6,07	R$ 33,39

Fonte: Elaborado pelo autor, considerando dólar a R$ 5,50

[5] Valor do dólar consultado em https://br.investing.com/currencies/usd-brl, acesso em 04 ago. 2025.

7 CONCLUSÃO

Neste trabalho, o objetivo principal consistiu em avaliar o comportamento de aplicações desenvolvidas nas linguagens PHP e .NET hospedadas em ambientes de nuvem selecionados com base nos critérios de desempenho, popularidade, escalabilidade e utilização por parte de desenvolvedores. A seleção desses ambientes foi realizada de forma criteriosa e analítica. Inicialmente, uma revisão bibliográfica foi realizada utilizando chaves de buscas. Os artigos selecionados elencaram critérios para a escolha de ambientes de nuvens, alguns desses critérios foram utilizados no presente trabalho.

Ademais, após a escolha dos ambientes de nuvens, também houve a escolha das ferramentas de teste de cargas. Inicialmente, foram utilizados artigos para elencar as ferramentas de testes de carga *open source* mais populares. Após a listagem das ferramentas, apenas quatro foram escolhidas baseadas em critérios de popularidade e completude. As quatro ferramentas foram instaladas e expostas a testes iniciais, onde apenas duas foram selecionadas baseando-se novamente em critério de completude e utilização. O objetivo dessa seleção ajuda desenvolvedores no momento de escolher as ferramentas de testes de carga que melhor apresentam métricas.

Os resultados obtidos nos ambientes não visam definir um vencedor, entretanto, o objetivo é ajudar a escolher o melhor ambiente baseando-se no desempenho, tipo de aplicação e orçamento. Nesse sentido, o presente estudo entrega uma reflexão mais aprofundada sobre esses principais pontos destacados.

Em relação aos ambientes testados, observou-se que as diferenças de desempenho dependem diretamente do tipo de operação executada pelas aplicações. No cenário analisado, as aplicações na AWS apresentaram melhor desempenho em operações de leitura, tanto para aplicações PHP quanto para .NET, obtendo latências mais baixas e maior vazão. Por outro lado, a plataforma Azure obteve melhores resultados em operações de gravação, especialmente com a aplicação .NET, oferecendo menores tempos médios e medianos e maior estabilidade na latência das respostas.

Outro aspecto importante evidenciado neste estudo é a influência das características específicas de cada ambiente na performance das aplicações. Enquanto a *Azure App Service* oferece recursos dedicados de processamento, eliminando a preocupação com créditos de CPU e garantindo um desempenho mais estável sob cargas constantes, as instâncias EC2 do *Elastic Beanstalk* do tipo t3 da AWS utilizam um modelo *burstable* com uma *baseline* que favorece

cargas variáveis e pode resultar em economia financeira, porém com desempenho variável dependendo do acúmulo e utilização dos créditos.

Os resultados também destacaram diferenças significativas em relação aos custos operacionais dos ambientes analisados. Apesar do desempenho superior em apenas cenários de gravação, o *Azure App Service* B2 apresentou um custo consideravelmente mais elevado comparado ao AWS t3.small. Portanto, o presente estudo serve como um guia, reforçando que é essencial que desenvolvedores e empresas ponderem cuidadosamente a relação custo-benefício ao escolher uma plataforma de hospedagem, considerando o tipo predominante de operações, previsibilidade de carga, orçamento disponível e necessidades específicas da aplicação.

O presente estudo abre caminhos para pesquisas futuras, de modo que, derivados das conclusões deste estudo, escopos de pesquisas mais aprofundadas incluem:

1. Ampliar a análise para outras linguagens mais populares como Java e Go, com o objetivo de proporcionar uma visão mais abrangente.
2. Ampliar a análise para outros provedores de nuvem e diferentes tipos de instâncias ou planos, com intuito de verificar como variações na infraestrutura afetam o desempenho das aplicações hospedadas.
3. Investigar o impacto da localização geográfica dos recursos (como banco de dados e armazenamento de arquivos) no desempenho das aplicações hospedadas em ambientes de nuvem, considerando diferentes regiões e zonas de disponibilidade.
4. Avaliar o comportamento e desempenho das aplicações utilizando diferentes estratégias de autoescalabilidade e gerenciamento automático de recursos, incluindo mecanismos de cache e balanceamento de carga.
5. Explorar outros critérios além de desempenho, como segurança e facilidade de integração com ferramentas de CI/CD buscando proporcionar uma análise ainda mais completa e abrangente das soluções em nuvem.

REFERÊNCIAS

ALMEIDA, T. A. de; FILHO, N. P. R. Análise de desempenho de um servidor web em diferentes cenários em provedores de cloud computing. **Revista Fatec Tiradentes de Tecnologia**, v. 28, n. 139, p. 35–47, 10 2024. Disponível em: https://revistaft.com.br/analise-de-desempenho-de-um-servidor-web-em-diferentes-cenarios-em-provedores-de-cloud-computing/. Acesso em: 1 mar. 2025.

Amazon Web Services. **O que é uma API RESTful?** 2024. Disponível em: https://aws.amazon.com/pt/what-is/restful-api/. Acesso em: 6 maio 2025.

API7. **Performance Testing Tool**: wrk. 2022. Disponível em: https://api7.ai/learning-center/openresty/performance-testing-tool-wrk. Acesso em: 21 jun. 2025.

ARAUJO, M. N. de. **Computação em Nuvem**: Uma análise comparativa das plataformas disponibilizadas por amazon, google e microsoft. 2019. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal do Rio Grande do Norte (IFRN). Disponível em: https://memoria.ifrn.edu.br/handle/1044/2524. Acesso em: 9 fev. 2025.

ATAIE, E.; POOSHANI, M.; AQASIZADE, H. An empirical study on the impact of programming languages on the performance of open-source serverless platforms. **International Journal of Engineering**, v. 38, p. 424–435, 05 2024.

AWS. **Global Infrastructure**: Regions & availability zones. 2024. Disponível em: https://aws.amazon.com/pt/about-aws/global-infrastructure/regions_az/. Acesso em: 21 jun. 2025.

AWS. **Our Origins**. 2024. Disponível em: https://aws.amazon.com/pt/about-aws/our-origins/. Acesso em: 21 jun. 2025.

AWS. **Amazon EC2 T3 Instances**. 2025. Disponível em: https://aws.amazon.com/pt/ec2/instance-types/t3/. Acesso em: 19 jul. 2025.

AWS. **O que é o armazenamento em nuvem?** 2025. Disponível em: https://aws.amazon.com/pt/what-is/cloud-storage/. Acesso em: 11 mai. 2025.

BADGER, L.; GRANCE, T.; PATT-CORNER, R.; VOAS, J. **NIST Cloud Computing Reference Architecture**. [*S. l.*], 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf. Acesso em: 18 mai. 2025.

BERNERS-LEE, T. **Information Management**: A proposal. 1989. Disponível em: https://www.w3.org/History/1989/proposal.html. Acesso em: 1 mai. 2025.

BRAVE. **Web3**: Qual a diferença entre web 1.0, 2.0 e 3.0? 2022. Disponível em: https://brave.com/pt-br/web3/versus-web1-and-web2/. Acesso em: 17 abr. 2025.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. **Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit**: Challenges and opportunities. 2009. Disponível em: https://arxiv.org/abs/0907.4878. Acesso em: 14 abr. 2025.

CERN. **A Short History of the Web**. 2025. Disponível em: https://home.cern/science/computing/birth-web/short-history-web. Acesso em: 1 mai. 2025.

Chocolatey Software. **Chocolatey - The package manager for Windows**. 2025. Disponível em: https://chocolatey.org/. Acesso em: 28 jun. 2025.

DOGAN, J. **Hey**: Http load generator, apachebench (ab) replacement, formerly known as rakyll/boom. 2025. Disponível em: https://github.com/rakyll/hey. Acesso em: 21 jun. 2025.

FANO, A.; GERSHMAN, A. The future of business services in the age of ubiquitous computing. **Communications of the ACM**, ACM, v. 45, n. 12, p. 83–87, 2002.

FIELDING, R. T.; TAYLOR, R. N.; ERENKRANTZ, J. R.; GORLICK, M. M.; WHITEHEAD, J.; KHARE, R.; OREIZY, P. Reflections on the rest architectural style and "principled design of the modern web architecture"(impact paper award). In: **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2017. (ESEC/FSE 2017), p. 4–14. ISBN 9781450351058. Disponível em: https://doi.org/10.1145/3106237.3121282. Acesso em: 6 abr. 2025.

FILHO, V. C. **Desvendando As Nuvens**: Uma análise comparativa de desempenho de aplicações serverless e containers em provedores de computação em nuvem. Dissertação (Dissertação de Mestrado) – Universidade Estadual Paulista (UNESP), Instituto de Biociências, Letras e Ciências Exatas, Campus de São José do Rio Preto, São José do Rio Preto, SP, Brasil, 2024. Orientadora: Prof.ª Dr.ª Renata Spolon Lobato. Disponível em: https://repositorio.unesp.br/server/api/core/bitstreams/450f5973-2f53-40b3-bde0-44f0e57d5e10/content. Acesso em: 1 fev. 2025.

GLOZER, W. **wrk**: Modern http benchmarking tool. 2025. Disponível em: https://github.com/wg/wrk. Acesso em: 21 jun. 2025.

GOOGLE. **Comparação do interesse de pesquisa entre "wrk"e "hey load testing"nos últimos 5 anos**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=wrk,hey%20load%20testing&hl=pt. Acesso em: 21 jun. 2025.

GOOGLE. **Go Programming Language - Downloads**. 2025. Disponível em: https://go.dev/dl/. Acesso em: 21 jun. 2025.

GOOGLE. **Google Trends: JMeter, Wrk, Hey Load Test**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=JMeter,Wrk,Hey%20load%20test&hl=pt. Acesso em: 25 jun. 2025.

GOOGLE. **Interesse de pesquisa por "Hey Load Testing"nos últimos 5 anos**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=hey%20load%20testing&hl=pt. Acesso em: 21 jun. 2025.

GOOGLE. **Interesse de pesquisa por "wrk"nos últimos 5 anos**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=wrk&hl=pt. Acesso em: 21 jun. 2025.

GOOGLE. **JMeter**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=JMeter&hl=pt. Acesso em: 25 jun. 2025.

GOOGLE. **Tendência de buscas por k6, JMeter, Wrk e Hey load testing nos últimos 5 anos**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=k6,JMeter,Wrk,Hey%20load%20testing&hl=pt. Acesso em: 28 jun. 2025.

GOOGLE. **Tendência de buscas por "k6"nos últimos 5 anos**. 2025. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=k6&hl=pt. Acesso em: 28 jun. 2025.

Google Cloud. **Quais são os benefícios da computação em nuvem?** 2024. Disponível em: https://cloud.google.com/learn/advantages-of-cloud-computing?hl=pt-BR. Acesso em: 7 maio 2025.

GRAFANA. **Protocols supported by k6**. 2024. Disponível em: https://grafana.com/docs/k6/latest/using-k6/protocols/. Acesso em: 28 jun. 2025.

GRAFANA. **k6 - A modern load testing tool**. 2025. Disponível em: https://github.com/grafana/k6. Acesso em: 28 jun. 2025.

GUIMARAES, L. J. B. L. S.; ROCHA, E. C. de F. Práticas informacionais e design thinking: Abordando usuários 3.0 na ciência da informação. **RDBCI: Revista Digital de Biblioteconomia e Ciência da Informação**, v. 19, p. e021029, 2021. Disponível em: https://www.scielo.br/j/rdbci/a/QcgVHjdGRGmNXCN37SLmwzN/. Acesso em: 23 abr. 2025.

GUNUKULA, S. The future of cloud computing: Key trends and predictions for the next decade. **International Journal of Research in Computer Applications and Information Technology (IJRCAIT)**, IAEME Publication, v. 7, n. 2, p. 528–538, 2024. Disponível em: https://iaeme.com/MasterAdmin/Journal_uploads/IJRCAIT/VOLUME_7_ISSUE_2/IJRCAIT_07_02_041.pdf. Acesso em: 21 abr. 2025.

GUSTAFSSON, R. **Grafana k6, one year later**: Lessons learned after an acquisition. 2022. Disponível em: https://grafana.com/blog/2022/10/06/grafana-k6-one-year-later-lessons-learned-after-an-acquisition/. Acesso em: 28 jun. 2025.

HUANG, R.; FANG, S. Comparative analysis of cloud service providers. **International Journal of Cloud Computing and Database Management**, v. 5, n. 1, p. 13–16, 2024. Disponível em: https://www.computersciencejournals.com/ijccdm/article/55/5-1-4-950.pdf. Acesso em: 11 abr. 2025.

HUBSPOT. **Da web 1.0 à 4.0: conheça a evolução e entenda as diferenças**. 2023. Disponível em: https://br.hubspot.com/blog/marketing/evolucao-web. Acesso em: 23 abr. 2025.

IBM. **O que é virtualização?** 2024. Disponível em: https://www.ibm.com/br-pt/topics/virtualization. Acesso em: 13 maio 2025.

IBM. **REST APIs - IBM**. 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/rest-apis. Acesso em: 6 maio 2025.

IBM. **O que é monitoramento de cloud?** 2025. Disponível em: https://www.ibm.com/br-pt/topics/cloud-monitoring. Acesso em: 15 abr. 2025.

IDC. **The Business Opportunity for Partners in the Microsoft Cloud Ecosystem**. 2023. Disponível em: https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-product-and-services/azure/documents/IDC-whitepaper.pdf. Acesso em: 21 jun. 2025.

IYER, B.; HENDERSON, J. Preparing for the future: Understanding the seven capabilities of cloud computing. **MIS Quarterly Executive**, v. 9, p. 117–131, 01 2012.

JADHAV, K. M.; PATIL, T. S.; SUTAR, P. S.; BHORE, H. S. Review paper on database basics for developers: Understanding crud operations. **International Journal of Research Publication and Reviews**, v. 6, n. 2, p. 2693–2697, 2025. Disponível em: https://ijrpr.com/uploads/V6ISSUE2/IJRPR38853.pdf. Acesso em: 28 jun. 2025.

JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling, NY: Wiley**. [*S. l.*: *s. n.*], 1991. ISBN 0471503361.

JMETER. **Apache JMeter**. 2024. Disponível em: https://jmeter.apache.org/index.html. Acesso em: 25 jun. 2025.

JMETER. **Apache JMeter**. 2024. Disponível em: https://github.com/apache/jmeter. Acesso em: 27 jun. 2025.

KREUTZ, D.; MACEDO, D.; ARBIZA, L. **Virtualização**: Conceitos, aplicações, mercado e prática. 2009. 50 p. Disponível em: https://www.researchgate.net/publication/280091108_Virtualizacao_Conceitos_Aplicacoes_Mercado_e_Pratica. Acesso em: 5 abr. 2025.

LONN, R. **Open Source Load Testing Tool Review**. 2020. Disponível em: https://grafana.com/blog/2020/03/03/open-source-load-testing-tool-review. Acesso em: 21 jun. 2025.

MELL, P.; GRANCE, T. The nist definition of cloud computing. Gaithersburg, MD, n. NIST Special Publication 800-145, 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Acesso em: 18 maio 2025.

MICROSOFT. **The History of Microsoft Azure**. 2022. Disponível em: https://techcommunity.microsoft.com/blog/educatordeveloperblog/the-history-of-microsoft-azure/3574204. Acesso em: 21 jun. 2025.

MICROSOFT. **Microsoft Physical Security**. 2024. Disponível em: https://learn.microsoft.com/pt-br/azure/security/fundamentals/physical-security. Acesso em: 21 jun. 2025.

MICROSOFT. **Azure App Service on Windows Pricing**. 2025. Disponível em: https://azure.microsoft.com/en-us/pricing/details/app-service/windows. Acesso em: 19 jul. 2025.

Microsoft Learn. **Design de API em microsserviços - Azure Architecture Center**. 2024. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/microservices/design/api-design. Acesso em: 6 maio 2025.

MISHRA, D. Cloud computing: The era of virtual world. **International Journal of Computer Science Engineering**, VOLUME 3, p. 204–209, 07 2014.

MWENDI, E. Software frameworks, architectural and design patterns. **Journal of Software Engineering and Applications**, v. 07, p. 670–678, 01 2014.

NetApp. **What is Backup as a Service (BaaS)?** 2023. Disponível em: https://www.netapp.com/cloud-services/what-is-backup-as-a-services-baas/. Acesso em: 19 maio 2025.

NEVLUDOV, I.; SOTNIK, S. Cloud giants: Aws, azure and gcp. In: IEEE, 2023, Ivano-Frankivsk, Ukraine. **Proceedings of the 2nd International Conference on Innovative Solutions in Software Engineering (ICISE)**. Ivano-Frankivsk, Ukraine, 2023. p. 18–23. Disponível em: https://openarchive.nure.ua/server/api/core/bitstreams/4bf853f3-608f-483d-8f66-63079dde99bf/content. Acesso em: 1 fev. 2025.

O'REILLY, T. **What Is Web 2.0**: Design patterns and business models for the next generation of software. 2005. Disponível em: https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html. Acesso em: 23 abr. 2025.

O'SULLIVAN, T. C. Exploiting the time-sharing environment. In: RAYTHEON COMPANY, SUDBURY, MASSACHUSETTS, 1967, New York, NY, USA. **Proceedings of the 1967 22nd National Conference**. New York, NY, USA: Association for Computing Machinery, 1967. p. 169–175. Disponível em: https://dl.acm.org/doi/pdf/10.1145/800196.805986. Acesso em: 25 fev. 2025.

PHP. **PHP Manual - History of PHP**. 2024. Disponível em: https://www.php.net/manual/en/history.php.php. Acesso em: 25 jul. 2025.

QIAN, L.; LUO, Z.; DU, Y.; GUO, L. Cloud computing: An overview. In: JAATUN, M. G.; ZHAO, G.; RONG, C. (Ed.). **Cloud Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 626–631. ISBN 978-3-642-10665-1.

RNP. **O que é computação em nuvem?** 2023. Disponível em: https://esr.rnp.br/computacao-em-nuvem/o-que-e-computacao-em-nuvem/. Acesso em: 7 maio 2025.

SARASWAT, M.; TRIPATHI, R. Cloud computing: Comparison and analysis of cloud service providers-aws, microsoft and google. In: **2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)**. [*S. l.*: *s. n.*], 2020. p. 281–285.

SCHWAB, K. **A Quarta Revolução Industrial**. São Paulo: Edipro, 2016. Disponível em: https://www.amazon.com/Fourth-Industrial-Revolution-Klaus-Schwab-ebook/dp/B01JEMROIU/. Acesso em: 2 fev. 2025.

SHAFANA, A.; AMEER, F. M.; MUSTHAFA, N. Assessing the e-commerce websites for performance using automated testing tools. In: . [*S. l.*: *s. n.*], 2021.

SOMMERVILLE, I. **Engenharia de Software**. 10ª. ed. São Paulo: Addison-Wesley, 2011.

STACKOVERFLOW. **Developer Survey 2024 - Technology:**: Cloud platforms. 2024. Disponível em: https://survey.stackoverflow.co/2024/technology#1-cloud-platforms. Acesso em: 21 jun. 2025.

STATISTA. **Participação mundial de mercado dos principais provedores de infraestrutura em nuvem**: (q4 2024). 2024. Disponível em: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. Acesso em: 21 jun. 2025.

TANENBAUM, A. S. **Rede de Computadores**. 6. ed. São Paulo: Pearson Education, 2020.

TAURION, C. **Cloud Computing**: Computação em nuvem. Rio de Janeiro, Brasil: Brasport, 2009. ISBN 9788574524238. Disponível em: https://books.google.com.br/books?id=mvir2X-A2mcC. Acesso em: 3 maio 2025.

TSAI, W.-T.; SUN, X.; BALASOORIYA, J. Service-oriented cloud computing architecture. In: **2010 Seventh International Conference on Information Technology: New Generations**. [*S. l.*: *s. n.*], 2010. p. 684–689.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition. **ACM SIGCOMM Computer Communication Review**, ACM, v. 39, n. 1, p. 50–55, 2008.

VIEIRA, D. **Conheça o HTML, uma das linguagens mais usadas na web**. 2023. Disponível em: https://www.hostgator.com.br/blog/conheca-o-html/. Acesso em: 01 mai. 2025.

Webnode. **What Is Good Web Design?** 2023. Disponível em: https://www.webnode.com/blog/what-is-good-web-design/. Acesso em: 1 maio 2025.