UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM REDES DE COMPUTADORES

MARIA ISADORA GONÇALVES MARTINS DE OLIVEIRA

ANÁLISE DE DESEMPENHO DE *WEB SERVERS* UTILIZANDO DIFERENTES *BACKENDS* DA PLATAFORMA DOCKER

QUIXADÁ

2023

MARIA ISADORA GONÇALVES MARTINS DE OLIVEIRA

ANÁLISE DE DESEMPENHO DE *WEB SERVERS* UTILIZANDO DIFERENTES *BACKENDS* DA PLATAFORMA DOCKER

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Redes de Computadores.

Orientador: Prof. Dr. Michel Sales Bonfim.

QUIXADÁ

2023

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)

O48a Oliveira, Maria Isadora Gonçalves Martins de.
Análise de desempenho de web servers utilizando diferentes backends da plataforma docker / Maria Isadora Gonçalves Martins de Oliveira. – 2023.
53 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2023.
Orientação: Prof. Dr. Michel Sales Bonfim.

1. Análise de desempenho. 2. Web servers. 3. Conteinerização. I. Título.

CDD 004.6

MARIA ISADORA GONÇALVES MARTINS DE OLIVEIRA

ANÁLISE DE DESEMPENHO DE *WEB SERVERS* UTILIZANDO DIFERENTES *BACKENDS* DA PLATAFORMA DOCKER

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Redes de Computadores.

Aprovada em: / / .

BANCA EXAMINADORA

Prof. Dr. Michel Sales Bonfim (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Alisson Barbosa de Souza
Universidade Federal do Ceará (UFC)

Prof. Me. Francisco Luciano Castro Martins Júnior
Instituto Federal de educação, ciência e tecnologia do Ceará (IFCE)

Dedico este trabalho à minha família e amigos, que sempre acreditaram no meu potencial e nunca mediram esforços para proporcionar-me uma educação de qualidade.

AGRADECIMENTOS

Agradeço profundamente aos meus pais, Marco Antônio e Francisca Irisleide, pela dedicação incansável em proporcionar uma educação de qualidade para mim e minha irmã, Isis Maria. Sua constante disponibilidade e apoio foram fundamentais para enfrentar os desafios de viver em outra cidade.

Expresso meu agradecimento ao meu namorado, aos meus amigos e aos meus familiares pelo apoio e pela paciência demonstrada nos momentos mais estressantes. Suas presenças foram pilares essenciais durante essa jornada.

Também estendo meus agradecimentos a todos os professores que fizeram parte da minha trajetória educacional, destacando o professor Michel Sales, pela parceria e excelente orientação.

"Não há saber mais ou saber menos. Há saberes diferentes." (Paulo Freire, 1987)

RESUMO

No contexto da crescente demanda por aplicações web modernas, a pesquisa apresentada neste trabalho surge em resposta ao contínuo desenvolvimento de tecnologias destinadas a aprimorar o desempenho e eficiência dos *web servers*. Os *web servers*, fundamentais para executar e armazenar aplicativos acessados por meio de redes, desempenham um papel vital na experiência do usuário. Nesse cenário, a escolha e otimização desses servidores tornam-se cruciais para garantir respostas rápidas e eficazes. O advento da conteinerização, exemplificado pelo Docker, trouxe uma abordagem inovadora à implantação de aplicativos, permitindo execução isolada e escalabilidade eficiente. Este estudo concentra-se na análise de desempenho de *web servers* em ambientes Docker, reconhecendo a relevância de selecionar configurações ideais para obter o máximo rendimento e uma boa experiência ao usuário final. Para a realização deste trabalho, foram estabelecidos seis cenários que incorporam diferentes *backends* (runC, gVisor e o componente Windows Subsystem for Linux (WSL)). Além da variação nos *backends*, foram delineados dois *web servers* distintos, o Apache e o Nginx. Adicionalmente, optou-se por utilizar dois sistemas operacionais diferentes, o Linux e o Windows. À medida que os cenários foram implementados, empregou-se a ferramenta de *benchmarking* ApacheBench. Essa ferramenta possibilitou a coleta de dados de métricas cruciais para análise, incluindo solicitações com falha, solicitações por segundo e tempo por solicitações. Testes estatísticos, como *Shapiro-Wilk* e *Bootstrap*, foram empregados para fundamentar e validar as conclusões derivadas da análise de desempenho. Após análise comparativa, verificou-se que o cenário que apresentou o melhor desempenho foi o primeiro cenário avaliado. Neste cenário, utilizou-se o *web server* Apache, hospedado no Docker com o *backend* conteinerd e runC, implementados no sistema operacional Linux. Essa constatação oferece respostas valiosas para profissionais da Tecnologia da Informação que desempenham papéis cruciais no projeto e desenvolvimento de aplicações *web*, proporcionando-lhes informações que os capacitarão a oferecer experiências ainda melhores para os usuários.

Palavras-chave: análise de desempenho; *web servers*; conteinerização.

ABSTRACT

In the context of the growing demand for modern web applications, the research presented in this work arises in response to the continuous development of technologies aimed at improving the performance and efficiency of web servers. Web servers, essential for running and storing applications accessed over networks, play a vital role in the user experience. In this scenario, the choice and optimization of these servers become crucial to guarantee fast and effective responses. The advent of containerization, exemplified by Docker, has brought an innovative approach to application deployment, enabling isolated execution and efficient scalability. This study focuses on the performance analysis of web servers in Docker environments, recognizing the relevance of selecting ideal configurations to obtain maximum performance and a good end-user experience. To carry out this work, six scenarios were established that incorporate different backends (runC, gVisor and the WSL component). In addition to the variation in backends, two distinct web servers were outlined, Apache and Nginx. Additionally, it was decided to use two different operating systems, Linux and Windows. As the scenarios were implemented, the benchmarking ApacheBench tool was used. This tool made it possible to collect data on crucial metrics for analysis, including failed requests, requests per second, and time per requests. Statistical tests, such as Shapiro-Wilk and Bootstrap, were employed to substantiate and validate the conclusions derived from the performance analysis. After comparative analysis, it was found that the scenario that presented the best performance was the first scenario evaluated. In this scenario, we used web server Apache, hosted on Docker with backend container and runC, implemented on the Linux operating system. This finding offers valuable answers for Information Technology professionals who play crucial roles in the design and development of web applications, providing them with information that will enable them to offer even better experiences for users.

Keywords: performance analysis; web servers; containerization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CPU	Unidade Central de Processamento
HCS	Host Compute Service
HTTP	Hyper Text Transfer Protocol
OCI	Open Container Iniciative
TI	Tecnologia da Informação
W3Techs	World Wide Web Technology Surveys
WSL	Windows Subsystem for Linux

SUMÁRIO

1 INTRODUÇÃO

Nos últimos tempos, a crescente demanda por aplicações web modernas tem impulsionado o contínuo desenvolvimento de tecnologias e ferramentas destinadas a aprimorar o desempenho e eficiência dos servidores web. Um *web server* é um *software* responsável por executar e armazenar aplicativos de *software* que são acessados e utilizados a partir de uma rede. Os *web server* mais comuns no mercado da tecnologia da informação são o Apache e o Nginx. Esses servidores podem ser executados por meio da conteinerização (KITHULWATTA *et al.*, 2022).

A conteinerização (SERVICES, 2023b) consiste na utilização de uma arquitetura baseada em *containers* para a implantação de aplicativos e serviços, o que implica na execução desses elementos de forma isolada e escalável.

Atualmente, uma das tecnologias de *containers* mais populares é o Docker. Embora tenha sido inicialmente criado para o sistema operacional Linux, ele se expandiu e está disponível para uso em outros sistemas operacionais, como o Windows. Essa ampla compatibilidade contribuiu para o crescimento e popularidade do Docker (SERGEEV *et al.*, 2022). A plataforma Docker suporta diferentes *backends*, um *backend* é um *software* responsável por executar *containers* e gerenciar imagens de *containers* em um ambiente de implantação (ESPE *et al.*, 2020).

É importante que *web servers* sejam implementados em ambientes que permitam uma boa otimização do seu desempenho, pois isso pode reduzir a sua carga e até mesmo acelerar o carregamento de páginas. Na literatura, foram encontrados trabalhos relacionados que abordam três escopos: comparação de *web servers* distintos hospedados no Docker, comparação de diferentes *backends* e comparação do uso do Docker no Windows e no Linux. Contudo, o fato de haver diferentes *backends* e diferentes sistemas operacionais suportados pelo Docker torna necessário a realização de uma análise de desempenho mais detalhada para comparar os *web servers* em cenários variados. Sendo assim, o foco deste trabalho é implementar diferentes cenários a fim de realizar essa análise de desempenho.

Ao final deste estudo, será possível obter conclusões sobre o desempenho dos *web servers* nos diferentes cenários propostos. Serão analisadas métricas como tempo de resposta e taxa de transferência, com o intuito de identificar em qual cenário se tem melhor desempenho. Essas conclusões serão fundamentais para orientar a escolha do melhor cenário e maximizar o desempenho em futuras implementações.

1.1 Objetivos

1.1.1 Objetivo Geral

Avaliar o desempenho dos *web servers*, Apache e Nginx, na plataforma Docker, utilizando diferentes *backends* e sistemas operacionais hospedeiros.

1.1.2 Objetivos Específicos

1. Selecionar e configurar os *backends* apropriados para utilização com o Docker em ambientes Windows e Linux.

2. Desenvolver um plano de experimentos abrangente, incluindo a definição de métricas de desempenho e cenários de teste.

3. Realizar a coleta de dados e análise dos resultados obtidos a fim de avaliar o desempenho dos *web servers* Apache e Nginx em diferentes configurações de *backends* e sistemas operacionais hospedeiros.

A estrutura restante do trabalho foi organizada da seguinte maneira: no Capítulo 2 são apresentados os principais conceitos que direcionam este trabalho. No Capítulo 3 está inserida a descrição dos trabalhos relacionas e as comparações com o trabalho que está sendo proposto. Já no Capítulo 4 será realizado a explicação de como foi configurado o ambiente para execução dos cenários e execução dos testes. No Capítulo 5 é exposta à análise dos resultados e a discussão a partir da análise de desempenho empregada no trabalho. Por fim, no Capítulo 6 é apresentado as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão apresentados os conceitos fundamentais para a compreensão e orientação deste trabalho. Serão discutidos a virtualização baseada em containers, os *web servers*, como o Apache e Nginx, e os *benchmarks*, que são ferramentas que geram tráfego e calculam métricas utilizadas para medir o desempenho dos sistemas e aplicativos. O entendimento desses conceitos é essencial para compreender a relevância deste estudo e realizar uma análise crítica dos resultados obtidos.

2.1 Virtualização baseada em *containers*

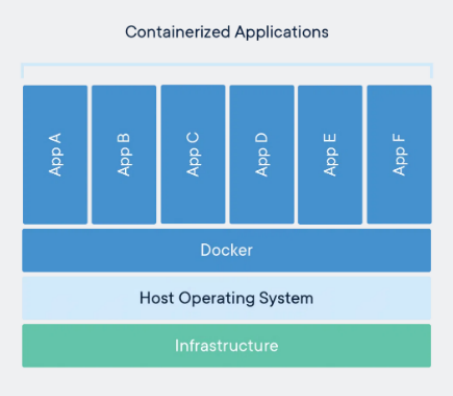
A virtualização baseada em *containers* é popularmente chamada de virtualização de nível de sistema operacional, ou seja, em que as instâncias compartilham um único kernel. Sendo assim, pode-se dizer que a virtualização assistida por *containers* é uma solução mais leve que a virtualização baseada em hipervisores, já que esta trabalha no nível de abstração do hardware (XAVIER *et al.*, 2013).

De acordo com COSTA e SANTOS (2021), neste tipo de virtualização, as instâncias são criadas com base em imagens reutilizáveis contendo todos os recursos necessários para a execução de uma aplicação encapsulada na imagem, como, por exemplo, código-fonte, bibliotecas e configurações, desta forma, produzindo um pacote de *software* leve. Consequentemente, o pacote de *software* pode ser executado rapidamente e com confiabilidade em qualquer ambiente de computação, desde que o mesmo possua compatibilidade com a tecnologia utilizada.

Na Figura 1, podemos analisar a arquitetura da virtualização baseada em *containers*. Inicialmente, temos uma infraestrutura, que pode ser tanto um base física quanto uma virtual, responsável por hospedar toda a tecnologia. Em seguida, temos o sistema operacional hospedeiro, que serve como ambiente principal para que os *containers* sejam executados. No ambiente de execução, utilizamos a tecnologia baseada em *containers*, que no caso, é o Docker. As aplicações são hospedadas dentro do Docker.

Dois recursos presentes no Linux são utilizados para a construção de *containers*, sendo eles: *cgroups* e *namespace*. *Cgroups* ou grupos de controle, é uma implementação usada para limitar e isolar diferentes tipos de recursos, como, por exemplo, Unidade Central de Processamento (CPU) e memória. Já os *namespaces* visam o isolamento de grupos de processos fazendo com que outros grupos de processos não sejam visualizados (ROMERO, 2016).

Figura 1 – Virtualização baseada em *containers*



Fonte: (DOCKER, 2023a).

Na execução deste trabalho, optamos por utilizar a plataforma Docker, já que é uma solução de virtualização baseada em *containers.* Na próxima Seção será detalhado o Docker.

2.1.1 Docker

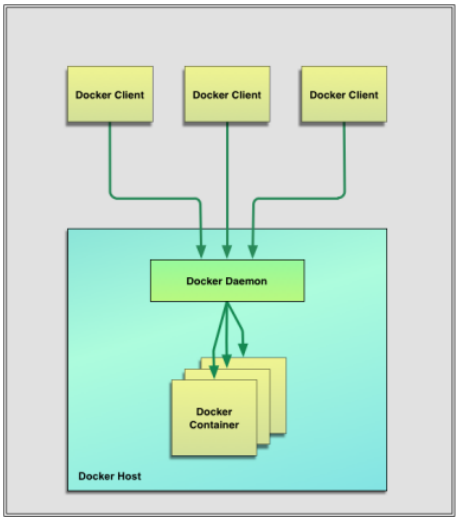
A história do Docker iniciou-se em 2008, quando a empresa dotCloud foi fundada por Solomon Hykes. Em março de 2013, a empresa decidiu deixar o núcleo da plataforma *open source*, ou seja, qualquer pessoa poderia analisar o código e contribuir com as melhorias do Docker. Após todo o sucesso da plataforma, a empresa dotCloud passou a se chamar Docker (VITALINO; CASTRO, 2018).

O *Docker* é uma plataforma de virtualização de *containers* que possui código aberto. Ela é capaz de facilitar o desenvolvimento, envio e execução de aplicativos por meio dos chamados *containers* (DOCKER, 2023b).

Como mencionado, os *containers* são a base para a execução da arquitetura do *Docker*. Eles são unidades de *software* leves que são capazes de encapsular aplicações e as suas dependências. Os *containers* são isolados do sistema operacional e do ambiente em que o *host* é executado, garantindo a segurança do ambiente de desenvolvimento e produção (DOCKER, 2023b).

O *Docker* usa uma arquitetura cliente-servidor, como ilustrado de forma simplificada na Figura 2. Nessa arquitetura, o *Docker Client*, que é a interface que os usuários usam para interagir com o Docker, se comunica com o *Docker Daemon*, o principal serviço que gerencia os *containers* em um sistema. Essa comunicação ocorre através do uso de uma *API REST*, por meio de soquetes UNIX ou mesmo uma interface de rede. O *Docker Daemon* é quem se responsabiliza

Figura 2 – Arquitetura do *Docker*



Fonte: (TURNBULL, 2017).

pelo trabalho de construção, execução e distribuições dos *Docker Containers*. O *Docker client* e o *Docker daemon* podem ser executados em um mesmo *host* ou conectados mediante um *Docker Daemon* remoto. O *Docker Host* é o sistema operacional em que o *docker daemon* está instalado e é essencial para o funcionamento do Docker na sua totalidade (TURNBULL, 2017).

2.1.1.1 Backends no ambiente Linux

Na seção anterior, abordamos o Docker. No entanto, para uma compreensão mais aprofundada do Docker no ambiente Linux, dedicaremos esta seção aos *backends*, os quais desempenham um papel fundamental na execução de *containers* Docker no Linux.

Os *backends* são responsáveis por executar e gerenciar os componentes necessários para a execução de um *container*. Por consequência, os *backend* exercem uma função crucial no gerenciamento de *containers*, proporcionando uma execução segura e eficiente (KAISER *et al.*, 2022).

Segundo Kaiser *et al.* (2022), pode-se agrupar os *backends* em duas categorias, são elas: *backends* de nível superior e *backends* de nível inferior. O *backend* de nível superior é aplicado em aspectos como o formato, desempacotamento, gerenciamento e compartilhamento de imagens de *backends*. Por outro lado, os *rbackends* de nível inferior estão relacionados a execução dos *containers*, envolvendo a configuração de *namespaces* e *Cgroups* para os mesmos.

A *Open Container Iniciative (OCI)* Iniciative (2023), é um projeto de governança leve e aberta, instituído sob a supervisão da Linux Fundation, tendo como objetivo o desenvolvimento de padrões abertos para formatos de imagem de *containers* e tempos de execução de *containers*.

Empresas líderes no setor de *containers*, incluindo Docker e CoreOS, foram responsáveis por lançar a OCI em junho de 2015.

Atualmente, a OCI consiste em três componentes-chave: a *runtime-spec*[1], a *image-spec*[2] e a *distribuition-spec*[3]. A *runtime-spec* é responsável por definir as diretrizes para a configuração, por definir o ambiente de execução e por definir o ciclo de vida de um *container*. A *image-spec* tem como objetivo a viabilização de criação de ferramentas capazes de operar a construção, o transporte e a preparação de imagens de *containers* para execução. Já a *distribuition-spec* é responsável por definir um protocolo *Application Programming Interface (API)* que facilite e padronize a distribuição de um conteúdo (INICIATIVE, 2019).

A seguir serão apresentados os *backends* que serão utilizados no trabalho, discutindo alguns dos *backends* existentes.

2.1.1.1.1 Containerd

O containerd (CONTAINERD, 2023) é um dos principais *softwares* para a execução de *containers* adotados na indústria, isso se deve à sua simplicidade, robustez e capacidade de funcionar em diversas plataformas. Ele funciona como um *daemon* em sistemas operacionais Linux e Windows, oferecendo inúmeros recursos para gerenciar todas as fases do ciclo de vida dos *containers* em sistemas hospedeiros. Essas funcionalidades incluem o gerenciamento de imagens, transferência de dados, execução e monitoramento de *containers*, além de fornecer suporte para recursos de armazenamento e rede de baixo nível.

2.1.1.1.2 RunC

O runC (CONTAINERS, 2023) é uma ferramenta de linha de comando projetada para gerar e executar *containers* no sistema operacional Linux, aderindo às especificações da OCI. Vale ressaltar que o runC é uma ferramenta de baixo nível que não foi criada com o objetivo de atender diretamente aos usuários finais. Sua principal função é ser utilizada por *softwares* de *backends* de nível superior, como o containerd, que fazem uso do runC como componente essencial para a base de suas funcionalidades. Para utilizar o runC, é necessário que se tenha um pacote OCI contendo um sistema de arquivos raiz e um arquivo de especificação denominado config.json.

[1] https://github.com/opencontainers/runtime-spec
[2] https://github.com/opencontainers/image-spec
[3] https://github.com/opencontainers/distribution-spec

2.1.1.1.3 gVisor

Conforme a documentação oficial (GVisor, 2023) o gVisor é um kernel de aplicativo, projetado para implementar uma considerável parte da interface de chamada do sistema operacional Linux. Seu principal objetivo é fornecer uma camada adicional de isolamento entre os aplicativos que estão sendo executados e o sistema operacional hospedeiro. Uma das características essenciais do gVisor é que o mesmo inclui um *backend* chamado runsc, que é totalmente compatível com a OCI. Essa integração permite que o gVisor seja facilmente utilizado em conjunto com várias ferramentas e plataformas de *containers*.

2.1.1.2 Virtualização no Windows

Nesta seção, abordaremos componentes relacionados à virtualização e à execução de *containers* no ambiente Windows. Vale ressaltar alguns dos componentes citados não foram usados nos cenários propostos.

2.1.1.2.1 Windows Subsystem for Linux

Conforme apontado por (MICROSOFT, 2023c), o WSL possibilita a execução de um ambiente Linux diretamente no Windows, sem a necessidade de modificar ou de sobrecarregar o sistema com uso de máquina virtual, ou instalação dualboot. Para utilizar o WSL como *backend* para o funcionamento do Docker, é necessário instalar o Docker Desktop e habilitar o mecanismo baseado em WSL. Isso permitirá a execução de *containers* tanto Linux quanto Windows no Docker Desktop, tudo a partir do mesmo computador. (MICROSOFT, 2023d)

2.1.1.2.2 runHCS

O runHCS (MICROSOFT, 2023a) é uma derivação do runC. Assim como o runC, o runHCS segue as diretrizes estabelecidas pela OCI. Porém, este foi desenvolvido com foco na plataforma Windows, sendo projetado para operar nesse sistema operacional. Uma das principais funcionalidades do runHCS é a comunicação com o *Host Compute Service (HCS)* para criar e gerenciar *containers*. Basicamente, o runHCS é uma ferramenta de linha de comando para executar *containers* no Windows, este é capaz de aproveitar o isolamento do Hyper-V[4] que pode fazer com que várias instâncias de *containers* sejam executadas de forma simultânea em um

[4] https://learn.microsoft.com/pt-br/virtualization/windowscontainers/manage-containers/hyperv-container

sistema hospedeiro.

2.1.1.2.3 Hyper-V

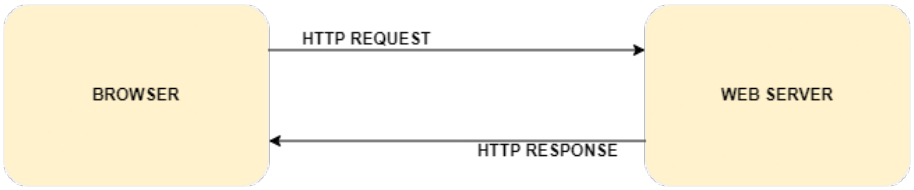
Segundo Microsoft (2023b) o Hyper-V é produto de virtualização de hardware da Microsoft, que possibilita a criação e execução de máquinas virtuais, que, por sua vez, são versões de software de computadores completos. O Hyper-V opera cada máquina virtual em seu próprio ambiente isolado, permitindo a execução simultânea de várias máquinas virtuais em um único hardware. O Hyper-V pode ser empregado como *backend* no Docker, no entanto, apenas em algumas versões do Windows, como o Windows 10 Pro e Enterprise.

2.2 *Web Servers*

Segundo Putro e Supono (2022), um *web server* é um *software* que recebe solicitações enviadas por meio de um navegador e retorna páginas de um site ou um documento HTML. O servidor web atua como centro de controle, processando as solicitações recebidas dos navegadores e gerenciando toda a comunicação entre o navegador e o servidor para processamento de um site. Em síntese, o servidor web desempenha um papel fundamental na gestão da interação entre o navegador e o servidor para ser disponibilizado um site ao usuário.

A Figura 3 oferece uma representação visual do funcionamento de um servidor web. Em que primeiramente o *browser* faz uma requisição Hyper Text Transfer Protocol (HTTP) ao *web server* e este retorna uma página web por meio de uma resposta HTTP.

Figura 3 – Web Servers



Fonte: Elaborado pela autora

Tanto na indústria de tecnologia da informação quanto no meio acadêmico, o Apache e o Nginx são os servidores web mais utilizados e conhecidos (KITHULWATTA *et al.*, 2022). De acordo com pesquisa da *World Wide Web Technology Surveys (W3Techs)*, o servidor web Nginx é o mais utilizado em todas as aplicações voltadas para a Internet, com 34,4%. Já o Apache foi usado por 31,4% das aplicações voltadas para a Internet. A Figura 4 apresenta uma comparação do uso de servidores web com base nos dados estatísticos da pesquisa realizada pela W3Techs

(W3TECHS, 2023).

Figura 4 – Servidores Web mais populares

© W3Techs.com	usage	change since 1 April 2023
1. Nginx	34.4%	-0.1%
2. Apache	32.0%	-0.2%
3. Cloudflare Server	20.5%	+0.3%
4. LiteSpeed	11.9%	+0.1%
5. Microsoft-IIS	5.5%	-0.1%

percentages of sites

Fonte: W3Techs

2.2.1 Nginx

Criado por Igor Sysoev, o Ngnix é um servidor web de código aberto, que possui recursos adicionais como o proxy reverso, o cache HTTP, o balanceamento de carga e tem também suporte a IPv6. Uma das principais vantagens do Nginx em comparação com outros servidores web é o seu baixo consumo de memória (CHANDRA, 2019).

2.2.2 Apache

O Apache é um servidor web de código aberto e baseado em UNIX. Ele foi incialmente desenvolvido a partir do código do NCSA HTTPD 1.3, que era um servidor. O Apache oferece recursos que garantem alta performance, funcionalidade avançada, eficiência e velocidade. Por essas razões, o Apache se tornou um dos servidores web mais populares (CHANDRA, 2019).

2.3 Análise de Desempenho

De acordo com Jain (1991), a avaliação de desempenho é uma forma de arte, pois exige um profundo entendimento do sistema que será avaliado e uma cuidadosa escolha da metodologia para a conduzir a análise. Além disso, cada analista pode possuir um estilo único ao realizar uma avaliação, utilizando diferentes métricas de desempenho e metodologias distintas para a análise.

Para conduzir a análise proposta, torna-se necessário empregar uma ferramenta de *benchmarking*. O *benchmarking* é o processo de medição e comparação, destinado a alcançar

resultados aprimorados (ALBERTIN *et al.*, 2016).

2.3.1 ApacheBench

A ferramenta a ser empregada é o ApacheBench[5]. Trata-se de uma ferramenta desenvolvida pela organização Apache que possibilita a análise de protocolo HTTP. O ApacheBench é uma ferramenta de código aberto, de fácil utilização por linha de comando, e fornece resultados que podem ser posteriormente analisados (CHANDRA, 2019). Alguns dos valores retornados incluem solicitações com falha, solicitações por segundo, tempo por solicitação, tempo por solicitações concorrentes e taxa de transferência. Essas métricas serão utilizadas para avaliar o desempenho dos cenários. As métricas que foram utilizadas estão na Tabela 1:

Tabela 1 – Métricas de Desempenho do *Web Server*

Métrica	**O que é**
Solicitações com falha	É o número de solicitações que não obtiveram êxito.
Solicitações por segundo	É a taxa média de solicitações que o servidor é capaz de atender por segundo.
Tempo por solicitação	É o tempo médio que o servidor leva para processar uma solicitação.
Tempo por solicitação concorrente	É o tempo médio gasto pelo servidor para processar solicitações simultâneas.
Taxa de transferência	É a quantidade de dados transferidos entre o servidor e o cliente.

Fonte: Elaborado pela autora.

2.4 Testes estatísticos

A estatística é uma área que utiliza métodos de coleta, apresentação e análise de dados visando embasar decisões, solucionar problemas e aprimorar o planejamento em produtos e sistemas. Isso é especialmente relevante em situações de variabilidade, onde as observações podem resultar em diferentes conclusões (MONTGOMERY; RUNGER, 2016). Segundo Fernandes (1999), o principal objetivo de uma análise estatística é realizar inferências sobre uma população, utilizando como base as informações obtidas por meio de uma amostra representativa retirada dessa população. Ferramentas comuns para essa análise são os chamados testes de hipóteses e intervalo de confiança.

[5] https://httpd.apache.org/docs/2.4/programs/ab.html

Segundo Hirakata *et al.* (2019), os testes de hipóteses são um método empregado para avaliar a veracidade de uma afirmação. Em uma abordagem simplificada, esses testes configuram uma estratégia de decisão que possibilita a aceitação ou rejeição de uma hipótese, fundamentada nas informações obtidas a partir de uma amostra.

Um teste de hipótese só pode incluir duas hipóteses sobre uma determinada população, uma referida como Hipótese Nula e a outra designada como Hipótese Alternativa, a hipótese nula é uma afirmação considerada verdadeira até que seja comprovado o contrário, ao passo que a hipótese alternativa é uma proposição que se opõe à hipótese nula. Além disso, quando investigamos uma hipótese, só existem duas possibilidades, ou ela realmente é verdadeira, ou ela é falsa, o que implica a existência de dois tipos de erro: Tipo I e Tipo II (HIRAKATA *et al.*, 2019).

O erro Tipo I se configura quando rejeitamos erroneamente uma Hipótese Nula verdadeira, levando a uma conclusão incorreta de que existe uma diferença, quando, na verdade, não há. Por contraste, o erro Tipo II ocorre quando não rejeitamos a Hipótese Nula, que é falsa, resultando na não detecção de uma diferença ou efeito real que de fato está presente. Dessa forma, o erro Tipo I implica uma interpretação equivocada da existência de uma diferença ou efeito, enquanto o erro Tipo II implica a não identificação de uma diferença ou efeito real que está presente (HIRAKATA *et al.*, 2019).

Para decidir se a hipótese nula deve ser rejeitada, estabelece-se uma região crítica, composta por valores que indicam a rejeição da hipótese nula. Durante o teste de hipótese, compara-se um valor obtido com um valor crítico, dependente do nível de significância e da distribuição dos dados. Além disso, o teste pode gerar um parâmetro chamado *P-value*, indicando o menor nível de significância para o qual a rejeição da hipótese nula é possível (MONTGOMERY; RUNGER, 2016).

O teste de normalidade utilizado para verificar se uma distribuição é normal ou não, foi o teste de *Shapiro Wilk*, em que quando o *P-value* é maior que 0,05, se caracteriza uma distribuição normal (ROYSTON, 1992).

Por fim, o intervalo de confiança é uma estimativa que fornece um intervalo para um parâmetro específico de uma população, sendo construído de maneira a proporcionar alta confiança em relação a esse parâmetro desconhecido da população. A determinação desse intervalo envolve a utilização de limites, os quais representam uma faixa de valores plausíveis para o referido parâmetro.(MONTGOMERY; RUNGER, 2016).

A escolha do método não paramétrico foi fundamentada na dispensa da necessidade de dados com distribuições normais. Além disso, métodos não paramétricos tendem a oferecer estimativas mais robustas diante da presença de valores atípicos, os chamados *outliers*, vale ressaltar que os *outliers* podem causar anomalias nos resultados que forem obtidos. Vale ressaltar que em poucos cenários os dados apresentaram distribuições normais. Diante desse contexto, optou-se pelo emprego do método *Bootstrap* para a obtenção de intervalos de confiança estimados para parâmetros populacionais, dada sua natureza não paramétrica. O *Bootstrap* é um método de reamostragem que se baseia em seleções aleatórias com reposição dentro da amostra original. Esse método oferece uma robustez, mas pode proporcionar estimativas estatísticas confiáveis.(DICICCIO; EFRON, 1996).

3 TRABALHOS RELACIONADOS

Nesta seção, são abordados alguns estudos relacionados ao projeto proposto neste trabalho, onde são destacados pontos em comum e as diferenças em relação à presente pesquisa.

3.1 Performance Evoluation of Docker-based Apache and Nginx Web Server

Em Kithulwatta *et al.* (2022), os autores realizam uma análise de desempenho dos servidores web Apache e Nginx quando estão hospedados no Docker. Foram analisadas métricas de desempenho como disponibilidade, tempo de resposta e requisições com falha. No estudo experimental, utilizou-se somente o Linux como sistema operacional. Além disso, para analisar as métricas, foram utilizados os seguintes *benchmarks*: ApacheBench e Siege. Com o experimento foi possível perceber que ambos servidores web têm bom desempenho e que estes são mais adequados para uso de hospedagem de aplicativos centrados na rede ao nível empresarial.

Semelhante ao estudo de Kithulwatta *et al.* (2022), o presente trabalho busca analisar o desempenho do Apache e do Nginx ao serem executados em *containers* Docker. No entanto, ao contrário do trabalho de Kithulwatta *et al.* (2022), este estudo será realizado com o Docker utilizando diferentes *backends* e também executado em sistemas operacionais hospedeiros diferentes, abrangendo tanto o Windows quanto o Linux.

3.2 Docker Container Performance Comparison on Windows and Linux Operating Systems

No estudo realizado por Sergeev *et al.* (2022), foi conduzido uma análise comparativa do Docker no Windows e no Linux, empregando sistemas operacionais hospedeiros distintos. Os pesquisadores desenvolveram uma ferramenta de teste para avaliar o desempenho aritmético e o desempenho da memória. Os resultados do estudo indicaram que o Docker no ambiente Linux possui um melhor desempenho que o Docker no ambiente Windows.

O trabalho proposto, diferentemente do estudo realizado por Sergeev *et al.* (2022), irá focar em outros componentes para análise. O trabalho apresentado anteriormente foca na análise do Docker em si, em diferentes sistemas operacionais hospedeiros, já o estudo proposto visa a análise do desempenho de diferentes *web servers*, considerando diferentes *backends* e sistemas operacionais.

3.3 Performance Evoluation of Container Runtimes

O trabalho de Espe *et al.* (2020) analisa o desempenho de diferentes *backends*. Foram utilizados dois *backends* amplamente empregados, o containerd e o CRI-O, juntamente com outros dois *backends* de baixo nível, o runC e o gVisor. Foi desenvolvida uma ferramenta para avaliar os seguintes aspectos: escalabilidade, desempenho dos *containers* que estão em execução e o desempenho das operações dos tempos de execução dos *containers*. O estudo concluiu que o containerd tem melhor desempenho em relação ao uso de CPU, a latência e a escalabilidade, enquanto o CRI-O apresenta melhor desempenho em termos de operações do sistema de arquivo.

Em contrapartida, ao estudo do Espe *et al.* (2020) o trabalho proposto não se concetra especificamente em realizar uma análise de desempenho dos *backends*. No entanto, esses *backends* desempenham um papel crucial na condução da análise de desempenho proposta, uma vez que o estudo busca explorar diferentes cenários utilizando *backends* distintos.

3.4 Análise Comparativa

Tal como feito em Kithulwatta *et al.* (2022), este trabalho tem como finalidade a realização de uma análise de desempenho de *web servers* no Docker. Contudo, Kithulwatta *et al.* (2022) desenvolveu o estudo somente no sistema operacional Linux, já este irá fazer a análise tanto no sistema operacional Windows como no sistema operacional Linux. Além disso, no trabalho proposto avaliaremos diferentes cenários com diferentes *backends* da plataforma Docker, ao contrário do estudo de Kithulwatta *et al.* (2022) que considerou apenas um cenário do Docker com dois *containers*.

Analisando o trabalho proposto e o trabalho de Sergeev *et al.* (2022), pode-se perceber que no trabalho mencionado a análise de performance é feita no uso do Docker em diferentes sistemas operacionais. Por outro lado, o estudo sugerido realizará análise de desempenho de *web servers* no Docker com cenários distintos e em diferentes sistemas operacionais hospedeiros, incluindo Linux e Windows. Uma semelhança entre o trabalho realizado por Sergeev *et al.* (2022) e o proposto é o uso de diferentes *backends*, no estudo de Sergeev *et al.* (2022) o Hyper-V é usado como *backend* para execução do Docker no Windows, enquanto o Docker Engine[1] é usado no Linux. Já em nosso projeto, será utilizado diferentes *backends*. No Linux, será usado o containerd como*backend* de alto nível, e runC e gVisor como *backend* de baixo nível. No

[1] https://docs.docker.com/engine/

Windows será feito uso do WSL2[2] do Docker.

Por fim, ao analisar o trabalho de Espe *et al.* (2020) pode-se perceber que ele difere do trabalho proposto no sentido de que o primeiro realiza análise de desempenho de *backends*, enquanto o último propõe uma análise de desempenho do Apache e o do Nginx no Docker com diferentes *backends*. O trabalho de Espe *et al.* (2020) utilizou tempos de execução como: CRI-O, containerd, runC e gVisor para realizar o estudo experimental, além disso, usou o Linux como sistema operacional hospedeiro. De maneira semelhante a este trabalho, no estudo proposto serão utilizados containerd, gVisor e runC.

A Tabela 2 apresenta um resumo das características presentes nos trabalhos relacionados que foram mencionados e a comparação dessas características com as do trabalho proposto.

Tabela 2 – Análise comparativa entre os trabalhos relacionados e o presente trabalho

Trabalho	***Web Server***	**Sistema Operacional**	**Diferentes *Backends***
(KITHULWATTA *et al.*, 2022)	Apache e Nginx	Linux	Não
(SERGEEV *et al.*, 2022)	X	Windows e Linux	Sim
(ESPE *et al.*, 2020)	X	Linux	Sim
Trabalho Proposto	Apache e Nginx	Windows e Linux	Sim

Fonte: Elaborado pela autora.

[2] https://docs.docker.com/desktop/windows/wsl/

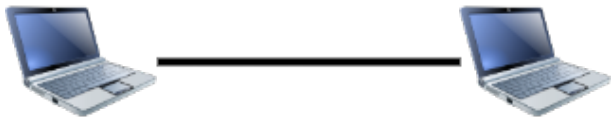
4 EXPERIMENTOS

Para que a análise fosse realizada de maneira eficaz, foi essencial configurar os cenários com os servidores web e os *backends* do Docker, bem como preparar o ambiente para a execução dos testes. O objetivo principal da análise era identificar qual desses cenários proporcionaria o melhor desempenho, levando em consideração as métricas escolhidas. A seguir, será explicado como foi realizada a configuração do ambiente.

4.1 Ambiente

Todos os testes foram realizados em uma topologia de rede ponto a ponto, o que significa que a conexão envolveu dois computadores. A representação dessa configuração pode ser encontrada na Figura 5. A organização desta estrutura foi fundamental, pois em um dos notebooks, criamos um ambiente Docker que incluía os servidores web e a aplicação, enquanto o outro notebook foi dedicado à execução dos testes utilizando a ferramenta de *benchmarking* escolhida.

Figura 5 – Topologia



Fonte: Elaborada pela autora

O notebook que hospedava o ambiente Docker era um Asus Vivobook X515DA, equipado com um processador Ryzen 7, um SSD de 256GB e 8GB de memória RAM. Este dispositivo operava em dual boot, executando os sistemas operacionais, Ubuntu 22.04.3 LTS e Windows 11 Home.

Por outro lado, o notebook utilizado para os testes era um Lenovo Ideapad 145, equipado com um processador Intel Core i5 de 10ª geração, além de possuir um SSD de 256GB, um HDD de 500GB e 8GB de memória RAM. Este notebook também operava em dual boot, com os sistemas operacionais, Ubuntu 22.04.3 LTS e Windows 10 Home.

Os próximos tópicos irão apresentar os ambientes configurados para a análise de desempenho. Na Seção 4.1.1, explicaremos como o ambiente de testes foi configurado, e na Seção 4.1.2, descreveremos como configuramos os cenários a serem analisados.

4.1.1 Ambiente de teste

Na máquina de testes, instalamos o ApacheBench, uma ferramenta que utilizamos para gerar os dados que posteriormente foram analisados. Para otimizar o processo de coleta de dados, criamos um script[1] contendo a linha de comando para executar o ApacheBench. Além disso, utilizamos o awk[2] para processar os resultados e criar um arquivo .csv[3] contendo apenas as métricas que seriam posteriormente analisadas, essas métricas foram especificadas na Tabela 1.

A linha de comando para executar o ApacheBench foi "ab -t 300 -n 10000 -c 100", desta forma, criamos assim um teste que faz simulação realista e com uma carga moderada. Torna necessário ressaltar que, para uma garantia de resultados confiáveis e representativos na análise, foram conduzidos 30 testes para cada cenário. Nos tópicos a seguir, está detalhado os parâmetros escolhidos:

- ab: É esse o comando que irá chamar a ferramenta de *benchmarking*, o ApacheBench.
- -t 300: Neste parâmetro, especificamos um limite de tempo para execução do teste em segundos, equivalente a 5 minutos.
- -n 10000: Especificamos o número total de solicitações que devem ser executadas durante o teste.
- -c 100: Este parâmetro define o número de solicitações concorrentes que devem ser mantidas simultaneamente durante o teste.

A ferramenta gera os dados e estes foram usados para a análise, as métricas utilizadas foram:

- **Solicitações com falha:** esta métrica está relacionada ao número de solicitações que não obtiveram êxito durante a execução dos testes. Solicitações com falha indicam que o servidor web está sobrecarregado e não possui capacidade suficiente para atender às requisições com sucesso (KITHULWATTA *et al.*, 2022).
- **Solicitações por segundo:** métrica de desempenho que indica quantas solicitações o *web server* é capaz de atender durante um segundo. Um valor mais alto sugere uma maior capacidade do servidor em atender à quantidade de solicitações. Além disso, essa métrica pode identificar gargalos de desempenho, indicando a necessidade de otimizações no desempenho ou de recursos de hardware adicionais. Em resumo, trata-se de uma métrica com

[1] https://github.com/mariaisadora-github/TCC-IsadoradeOliveira/tree/main/Scripts
[2] https://guialinux.uniriotec.br/awk/
[3] https://support.google.com/google-ads/answer/9004364?hl=pt-BR

um impacto direto na qualidade do serviço oferecido pelo *web server* (KITHULWATTA *et al.*, 2022).

- **Tempo por solicitação:** o Tempo por solicitação[4] é uma métrica que indica o tempo médio, em milissegundos, que uma solicitação leva para ser respondida pelo *web server*. Essa métrica avalia a eficiência do *web server* em lidar com cada requisição individual, impactando diretamente na experiência do usuário em relação ao *web server*. Um tempo por solicitação mais baixo geralmente resulta em respostas mais rápidas e contribui para uma experiência do usuário mais satisfatória.
- **Tempo por solicitações concorrentes[5]:** indica o tempo médio que um *web server* leva para processar solicitações quando estas são feitas de forma concorrente, ou seja, quando várias solicitações são enviadas simultaneamente. Essa métrica reflete a eficiência na capacidade do servidor de lidar com múltiplas solicitações concorrentes, demonstrando também a capacidade de escalabilidade do servidor. Calcular essa métrica traz uma abordagem mais próxima da realidade, uma vez que, em ambientes reais, projetos precisam enfrentar várias requisições sendo feitas ao mesmo tempo, sem que comprometa o tempo por solicitação.
- **Taxa de transferência:** métrica que evidencia a quantidade de dados que o *web server* é capaz de transferir para os usuários em um determinado período, geralmente expressa em kbytes/sec. Essa métrica reflete a eficiência do servidor na entrega das respostas das requisições, proporcionando uma experiência fluida aos usuários. Ao analisar essa métrica, é possível identificar eventuais gargalos na rede e limitações de largura de banda (SERVICES, 2023a).

No próximo tópico serão apresentados as configurações dos cenários que foram feitos para realizarmos a análise.

4.1.2 Ambiente dos cenários

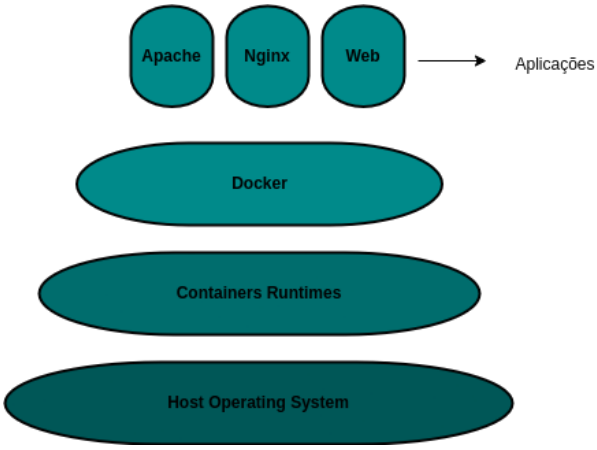
Para que fosse realizado uma análise relevante, foram escolhidos cenários que incluíam *web servers*, diferentes *backends* do Docker, além de dois sistemas operacionais.

Uma visão geral dos cenários incluídos na máquina com o ambiente Docker está representado na Figura 6. Na imagem podemos ver que teremos um sistema operacional hospedeiro, um *backend*, o Docker e hospedado no docker irão estar o Nginx, o Apache e a

[4] https://httpd.apache.org/docs/2.4/programs/ab.html
[5] https://httpd.apache.org/docs/2.4/programs/ab.html

aplicação web. É importante ressaltar que a aplicação web[6] utilizada foi desenvolvida como projeto final de uma disciplina da graduação. No âmbito desse projeto, a camada de *front-end* foi desenvolvida utilizando *Vue.js*, enquanto o *back-end* foi implementado por meio da plataforma *Strapi*.

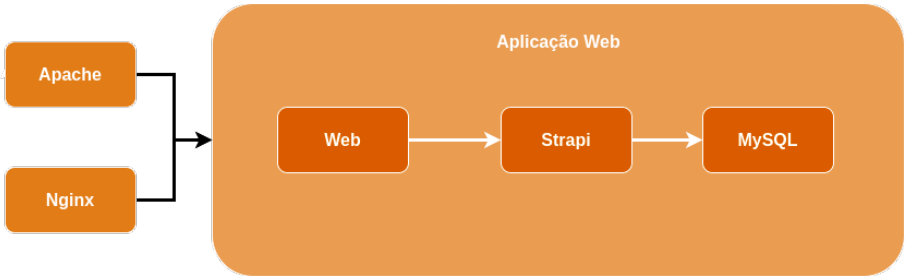
Figura 6 – Visão geral dos cenários



Fonte: Elaborado pela autora

Na Figura 7 é possível identificar os *containers* que estão sendo usados na aplicação web. Entretanto, tornou-se necessário aprimorar o arquivo docker-compose.yml, que anteriormente incluía *containers* para web, strapi e mysql. Para atender aos requisitos do projeto, precisamos adicionar novos *containers* , especificamente o Apache e o Nginx. Dessa forma, os *web servers* redirecionam para o *front-end* da aplicação.

Figura 7 – Visão dos Containers Implementados



Fonte: Elaborado pela autora

É importante destacar que a página web testada é um ambiente no qual são feitas solicitações *GET*, e o seu *path* é definido como */home*. Quando essas solicitações são executadas, o retorno total é de 479 *bytes*.

No Tabela 3 podemos observar de forma mais detalhada os cenários executados para a realização do que foi proposto.

[6] https://github.com/vitorreiel/DeliFood

Tabela 3 – Cenários

Cenários	***Web Server***	***Backend***	**Sistema Operacional**
Cenário 1	Apache	Containerd/RunC	Linux
Cenário 2	Nginx	Containerd/RunC	Linux
Cenário 3	Apache	Containerd/gVisor	Linux
Cenário 4	Nginx	Containerd/gVisor	Linux
Cenário 5	Apache	WSL	Windows
Cenário 6	Nginx	WSL	Windows

Fonte: Elaborado pela autora.

4.1.2.1 Cenário 1 e Cenário 2

Para criar o primeiro ambiente de execução, instalamos o Docker na versão 24.0.6, juntamente com o containerd na versão 1.6.24 e o runC na versão 1.1.9. É importante destacar que, ao instalar o Docker, a versão do runC que acompanhou foi a anteriormente mencionada, 1.1.9. Em seguida, no sistema Ubuntu, provisionamos os *containers* para realizar os testes. A orquestração dos *containers* foi realizada com o uso do docker-compose[7], que incluiu as imagens necessárias. No cenário 1 a versão do Apache utilizada nesse ambiente foi a 2.4.58, já no cenário 2 a versão do Nginx foi 1.25.2. Com o ambiente funcionando, conectamos as máquinas conforme mostrado na Figura 5 e com isso, partirmos para o teste, o ambiente de teste foi explicado na seção 4.1.1.

Na Figura 8, é possível observar que no Docker estão instalados e configurados os *backends* containerd e runC. É importante mencionar que o runC é o *backend* padrão do docker, eliminando a necessidade de configurações adicionais. No momento da captura, o *backend* de execução configurado é o runC.

Figura 8 – *Backend* de execução: runC

```
isadora@isamartins:~$ docker info | grep -i "Runtime"
 Runtimes: io.containerd.runc.v2 runc
 Default Runtime: runc
```

Fonte: Elaborado pela autora

4.1.2.2 Cenário 3 e Cenário 4

Para a configuração dos ambientes subsequentes de execução, tiramos proveito da instalação já existente do Docker na versão 24.0.6 e do containerd na versão 1.1.9. A única etapa adicional necessária foi a instalação do gVisor e a subsequente configuração para que o Docker o

[7] https://github.com/mariaisadora-github/TCC-IsadoradeOliveira/tree/main/Docker-compose-Backends/RunC

adotasse como *backend* em substituição ao runC. No momento em que instalamos o gVisor, a *branch* era release-20230911.0, sendo assim, esta foi a implementada no cenário.

A fim de realizar a configuração do uso do gVisor, procedemos da seguinte maneira: inicialmente, instalamos o gVisor conforme as instruções detalhadas na documentação oficial[8]. Em seguida, acessamos o ambiente do Docker e navegamos até o diretório /etc/docker, onde criamos um arquivo denominado 'daemon.json' com o seguinte conteúdo:

Figura 9 – Configuração do arquivo daemon.json para o gVisor

```
{
  "runtimes": {
    "gvisor": {
      "path": "/usr/local/bin/runsc"
    }
  },
  "default-runtime": "gvisor"
}
```

Fonte: Elaborado pela autora.

Após a criação deste arquivo de configuração, reiniciamos o serviço Docker, o que resultou na sua capacidade de utilizar o gVisor. A Figura 10 a seguir ilustra que no Docker há instalado e configurado os *backends* containerd, runC e gVisor, mas que o *backend* utilizado naquele momento é o gVisor.

Figura 10 – *Backend* de execução: gVisor

```
isadora@isamartins:~$ docker info | grep -i "Runtime"
 Runtimes: gvisor io.containerd.runc.v2 runc
 Default Runtime: gvisor
```

Fonte: Elaborado pela autora

Com a configuração do Docker concluída, avançamos para a preparação do restante do ambiente, iniciando os *containers* conforme especificado em um arquivo docker-compose.yml[9]. Esta configuração diferiu daquela utilizada nos cenários 1 e 2, uma vez que neste arquivo era necessário indicar o *backend* que estava sendo empregado. Nos cenários 1 e 2, a rede foi especificada no próprio arquivo docker-compose.yml. Contudo, nos cenários 3 e 4, não foi viável adotar a mesma abordagem. Nesses casos, optamos por remover a configuração específica da rede, recorrendo à rede padrão do Docker para facilitar a integração. Após o ambiente estar devidamente operacional, prosseguimos com a conexão da topologia e a execução do teste.

[8] https://gvisor.dev/docs/user$_guide/install/$

[9] https://github.com/mariaisadora-github/TCC-IsadoradeOliveira/tree/main/Docker-compose-Backends/gVisor

4.1.2.3 Cenário 5 e Cenário 6

Os próximos cenários foram configurados no sistema operacional Windows, como indicado na Tabela 3. Nos cenários do Windows, o componente WSL foi utilizado como *backend*.

Primeiramente, para configurar o ambiente Docker no Windows, seguimos os seguintes passos:

1. **Instalação do Windows Subsystem for Linux (WSL):**
   - Iniciei o processo instalando o WSL conforme as instruções disponíveis no site oficial da Microsoft[10].
2. **Instalação do Docker Desktop para Windows:**
   - Em seguida, realizei a instalação do Docker Desktop para Windows, seguindo as orientações fornecidas na documentação oficial do Docker[11].
3. **Configuração do Docker com o WSL:**
   - Após a instalação, acessei o Docker Desktop e configure o mecanismo de *backend* para utilizar o WSL como parte do processo, usando uma distribuição do Ubuntu.

Esses passos viabilizaram a configuração do ambiente Docker no sistema operacional Windows. Uma vez que o ambiente estava pronto, procedemos à inicialização dos *containers* por meio do arquivo docker-compose.yml[12] e estabelecemos a conexão do notebook com o ambiente de testes. Isso nos permitiu realizar os testes e coletar os dados necessários.

No próximo capítulo, apresentaremos uma análise detalhada dos dados que foram coletados durante o processo. Nesta etapa, exploraremos as informações e as métricas obtidas, fornecendo uma análise aprofundada dos resultados e conclusões decorrentes da nossa pesquisa.

[10] https://learn.microsoft.com/pt-br/windows/wsl/install
[11] https://docs.docker.com/desktop/install/windows-install/
[12] https://github.com/mariaisadora-github/TCC-IsadoradeOliveira/tree/main/Docker-compose-Backends/WSL

5 ANÁLISE DOS RESULTADOS

A análise dos dados foi conduzida no Google Colab[1], onde realizamos o *download* dos arquivos .csv contendo os dados coletados. Utilizamos o ambiente do Google Colab para processar e analisar esses dados com base nas métricas previamente estabelecidas, aproveitando as bibliotecas Python disponíveis para essa finalidade, como: pandas, seaborn, matplotlib, numpy e scipy.

Na etapa de análise dos dados, optamos por empregar o teste de normalidade, mais precisamente o teste de *Shapiro-Wilk*. Este teste desempenha um papel crucial ao determinar se uma amostra segue uma distribuição normal ou não, com a hipótese nula indicando que a amostra provavelmente possui uma distribuição normal, enquanto a hipótese alternativa sugere que a amostra não segue uma distribuição normal.

Uma vez obtidos os resultados do teste de *Shapiro-Wilk*, direcionamos nossa atenção para o método *Bootstrap*. Por meio desse método, exploramos os intervalos de confiança associados a cada amostra, com um grau de confiança de 95%. O Bootstrap é uma abordagem de reamostragem que nos permite calcular estimativas mais robustas e avaliar a variabilidade dos resultados.

Em síntese, a integração dessas etapas nos proporciona uma melhor compreensão do campo estatístico dos nossos dados, desde a avaliação de sua normalidade até a quantificação da incerteza por meio dos intervalos de confiança derivados do método *Bootstrap*. Essa abordagem integrada contribui para uma análise mais abrangente e confiável de nossos conjuntos de dados.

5.1 Análise exploratória dos dados

Prosseguiremos agora com uma análise exploratória dos dados referente a cada métrica, empregando o teste de *Shapiro-Wilk*. Isso proporciona uma avaliação mais precisa da distribuição dos dados em nossa análise exploratória.

[1] https://colab.research.google.com/

5.1.1 Solicitações com falha

5.1.1.1 Teste de Shapiro Wilk

Na Tabela 4 apresentada abaixo, destaca-se a observação crucial de que exclusivamente nos cenários que utilizam o Nginx como *web server* (cenários 2, 4 e 6), a distribuição demonstra-se como normal. Essa constatação é estabelecida pelo *P-value* maior que 0,05, que no caso dos cenários 2, 4 e 6 atingiu 1. Esse padrão de normalidade nos cenários específicos sugere uma estabilidade estatística consistente, fornecendo percepções valiosas para a compreensão das características dessas distribuições específicas.

Tabela 4 – Resultado do Teste de Shapiro-Wilk - Solicitações com falha

Cenário	P-value	Distribuição Normal
Cenário 1	0,00928	Não
Cenário 2	1,0	Sim
Cenário 3	2,51e-05	Não
Cenário 4	1,0	Sim
Cenário 5	0,00228	Não
Cenário 6	1,0	Sim

Fonte: Elaborado pela autora.

Na Figura 11 apresentada abaixo, a análise por meio de boxplot revela de maneira evidente a presença de *outliers* nos cenários desprovidos de distribuições normais. É pertinente ressaltar que tais *outliers* representam dados que se destacam significativamente em relação aos demais, indicando variações consideráveis. Esta observação reforça a heterogeneidade e a presença de pontos atípicos em tais cenários.

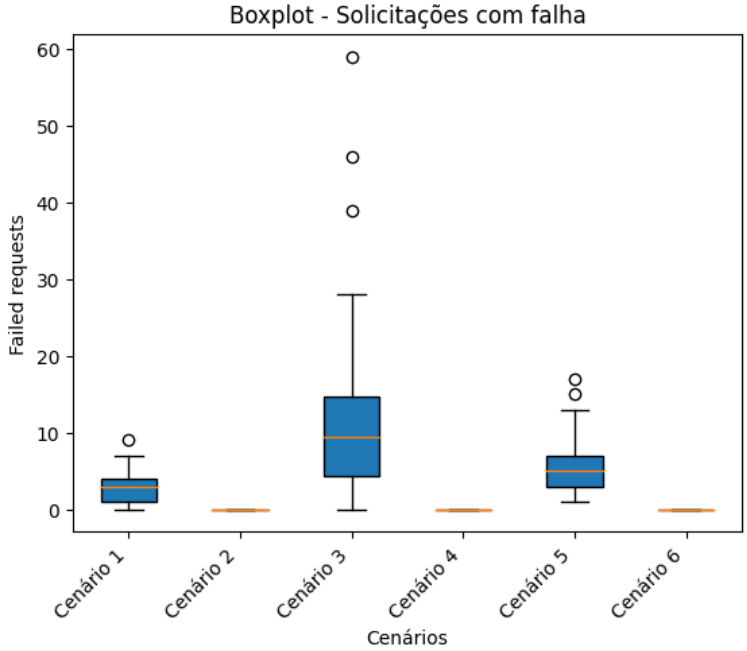
5.1.2 Solicitações por segundo

5.1.2.1 Teste de Shapiro Wilk

Na Tabela 5 abaixo, é possível analisar a presença de uma distribuição normal nos cenários, considerando a métrica de solicitações por segundo, por meio do teste de *Shapiro-Wilk*. Ressalta-se que solicitações por segundo se refere ao número de solicitações que podem ser atendidas em um determinado intervalo de tempo.

Pode-se inferir que, com base nos resultados do teste de *Shapiro-Wilk* aplicados a métrica solicitações por segundo, apenas os cenários 2 e 6 exibem uma distribuição normal,

Figura 11 – Boxplot - Solicitações com Falha



Fonte: Elaborado pela autora

Tabela 5 – Resultado do Teste de Shapiro-Wilk para Solicitações por Segundo

Cenário	P-value	Distribuição Normal
Cenário 1	1,80496e-07	Não
Cenário 2	0,09690	Sim
Cenário 3	0,00203	Não
Cenário 4	3,24889e-09	Não
Cenário 5	9,42493e-07	Não
Cenário 6	0,29933	Sim

Fonte: Elaborado pela autora.

destacando-se como exceções em relação aos demais cenários.

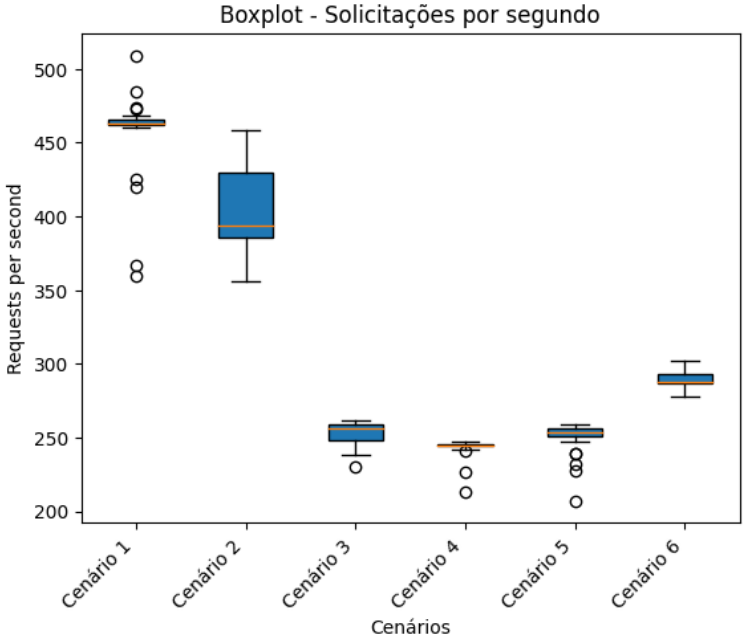
Na Figura 12 a seguir, a análise mediante o boxplot evidencia claramente a presença de *outliers* nos cenários que não apresentam distribuições normais. Notavelmente, os cenários 1 e 5 se destacam significativamente em relação aos demais, sugerindo a ocorrência predominante de que nos dados das amostras há pontos atípicos.

5.1.3 Tempo por solicitação

5.1.3.1 Teste de Shapiro Wilk

Na Tabela 6 a seguir, destaca-se claramente que os cenários 2 e 6 são os únicos a exibir uma distribuição normal, evidenciado pelo *P-value* superior a 0,05. Também podemos observar que existe uma ausência de normalidade nos dados quando o gVisor é utilizado como

Figura 12 – Boxplot - Solicitações por Segundo



Fonte: Elaborado pela autora

backend do Docker.

Tabela 6 – Resultado do Teste de Shapiro-Wilk para Tempo por Solicitação

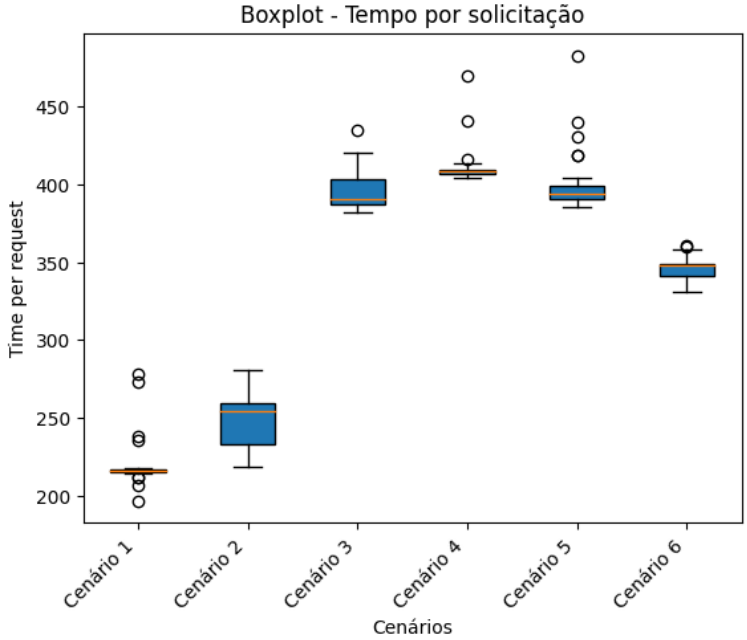
Cenário	**P-value**	**Distribuição Normal**
Cenário 1	3,55e-08	Não
Cenário 2	0,10291	Sim
Cenário 3	0,00111	Não
Cenário 4	1,78e-09	Não
Cenário 5	2,23e-07	Não
Cenário 6	0,27816	Sim

Fonte: Elaborado pela autora.

Na Figura 13 abaixo, a análise mais detalhada do comportamento dos dados é facilitada por meio do boxplot. É notório perceber que, apesar de haver uma distribuição normal nos cenários 2 e 6, há a presença de um *outlier* no cenário 6. Por outro lado, no cenário 2, não existem *outliers*. Esse padrão indica uma maior consistência nos dados, com valores mais concentrados ao redor da média.

Em contrapartida, nos cenários em que a distribuição dos dados não é normal, a presença de *outliers* se destaca, sobretudo no primeiro cenário, caracterizado pelo uso do *web server* Apache e da tecnologia runC como *backend* do Docker.

Figura 13 – Boxplot - Tempo por Solicitação



Fonte: Elaborado pela autora

5.1.4 Tempo por solicitações concorrentes

5.1.4.1 Teste de Shapiro Wilk

Na Tabela 7 a seguir, é possível verificar quais cenários apresentaram uma distribuição normal ou não em relação à métrica de tempo por solicitações concorrentes, com base nos resultados do Teste de *Shapiro-Wilk*. Vale ressaltar que, ao contrário da métrica analisada anteriormente, neste caso, estamos examinando o tempo médio que um sistema leva para processar uma solicitação quando várias solicitações concorrentes ocorrem de forma simultânea.

Tabela 7 – Resultados do Teste de Shapiro-Wilk para Tempo por solicitações concorrentes

Cenário	**P-value**	**Distribuição Normal**
Cenário 1	3,54e-08	Não
Cenário 2	0,10	Sim
Cenário 3	0,0011	Não
Cenário 4	1,79e-09	Não
Cenário 5	2,22e-07	Não
Cenário 6	0,28	Sim

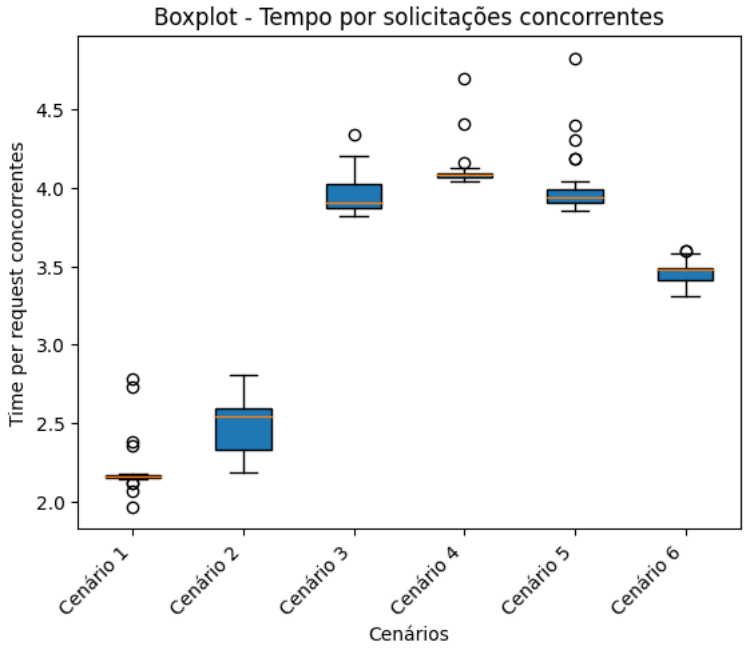
Fonte: Elaborado pela autora.

Pode-se observar que os valores do *p-value* dessa métrica assemelham-se aos valores do *p-value* da métrica tempo por solicitação. Isso pode ter ocorrido devido ao fato de ambas terem comportamentos estatísticos semelhantes. Nota-se também que, mais uma vez, os cenários

que apresentam uma distribuição normal são os cenários 2 e 6.

Na Figura 14 a seguir, é exibido um boxplot com uma análise mais aprofundada sobre o comportamento dos dados. Pode-se observar que, novamente, principalmente nos cenários que não possuem uma distribuição normal, há uma maior incidência de *outliers*. Nos cenários com distribuição normal, nota-se a presença de *outliers* apenas no cenário 6.

Figura 14 – Boxplot - Tempo por Solicitações Concorrentes



Fonte: Elaborado pela autora

5.1.5 Taxa de transferência

5.1.5.1 Teste de Shapiro Wilk

Na Tabela 8 abaixo, encontram-se os resultados do teste implementado na métrica de taxa de transferência, destinado a analisar os cenários em relação a essa métrica.

Conforme evidenciado na tabela, mais uma vez, os únicos cenários que exibem uma distribuição normal são os cenários 2 e 6. Por outro lado, os cenários 1, 3, 4 e 5 não apresentam normalidade em relação aos seus dados.

Na Figura 15 a seguir, a análise da distribuição é facilitada por meio do boxplot. Observa-se que nos cenários com distribuições normais, não há presença de *outliers*. Por outro lado, nos cenários com distribuições não normais, destaca-se a presença de *outliers*, sendo mais evidente nos cenários 1 e 5. Vale ressaltar novamente que esses *outliers* indicam valores

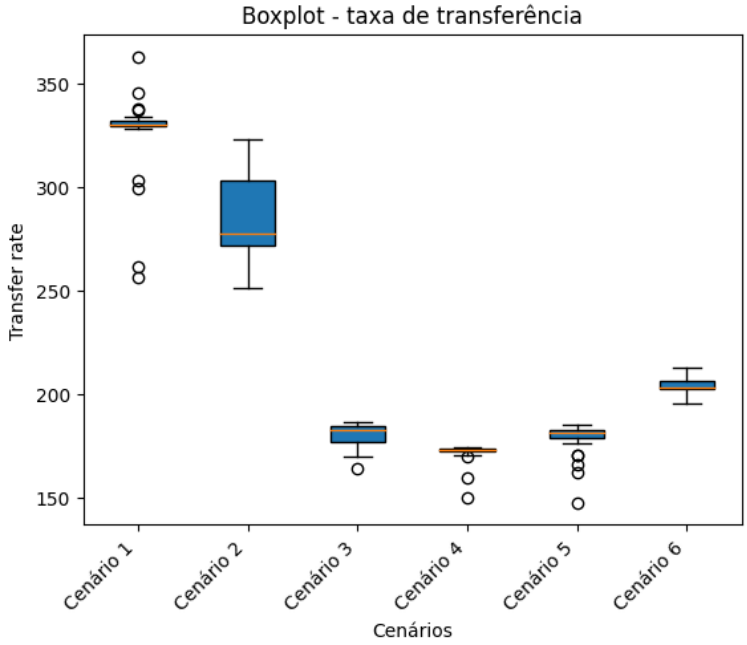
Tabela 8 – Resultados do Teste de Shapiro-Wilk para Taxa de Transferência

Cenário	P-value	Distribuição Normal
Cenário 1	1,81e-08	Não
Cenário 2	0,10	Sim
Cenário 3	0,0011	Não
Cenário 4	1,79e-09	Não
Cenário 5	2,22e-07	Não
Cenário 6	0,28	Sim

Fonte: Elaborado pela autora.

atípicos que se afastam do padrão geral da distribuição, não se aproximando tanto da média. Essa observação contribui para uma compreensão mais aprofundada da variabilidade nos tempos de solicitação entre os cenários.

Figura 15 – Boxplot - Taxa de Transferência



Fonte: Elaborado pela autora

5.2 Comparação dos Resultados

Baseado nos resultados da seção anterior, é possível concluir que a maioria das nossas amostras não seguem uma distribuição normal. E que, mesmo o tamanho delas sendo suficientemente grande (maior ou igual a 30), optamos por um teste não paramétrico, usando como estatística a média. Para tanto, decidimos pelo uso do *Bootstrap* devido à sua robustez em lidar com a ausência de normalidade nos dados, o método escolhido apresenta-se como uma abordagem flexível e confiável, especialmente quando as suposições sobre a distribuição

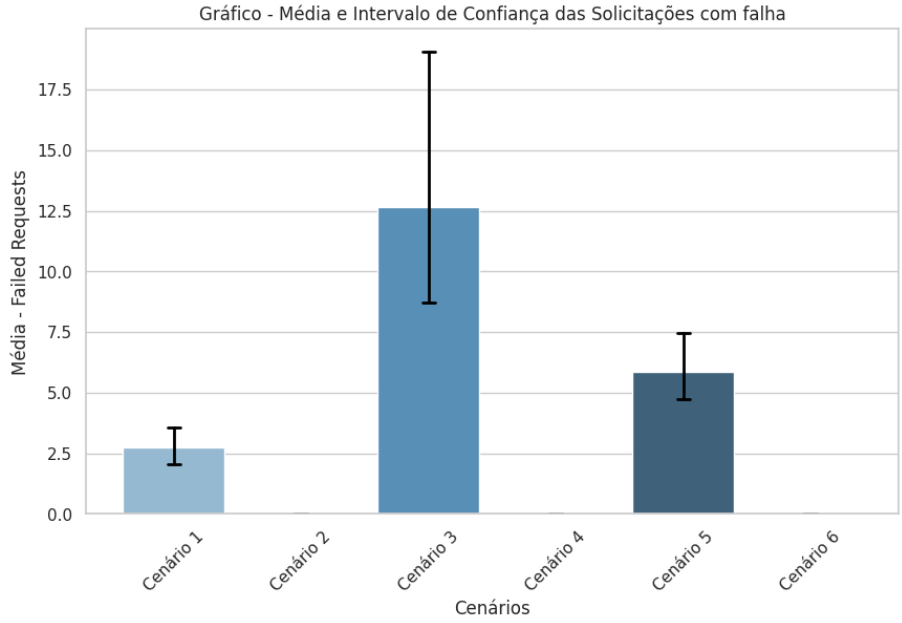
dos dados não são plenamente atendidas. Isso possibilita uma inferência estatística eficaz em diversos contextos.

5.2.1 Solicitações com falha

Devido à ausência de distribuições normais em alguns cenários, optou-se por empregar o método *Bootstrap* para calcular o intervalo de confiança, estabelecendo um nível de confiança de 95%. Essa escolha visa superar as limitações decorrentes da falta de normalidade dos dados, garantindo uma análise mais sólida e abrangente dos resultados.

Na Figura 16 subsequente, observa-se que o intervalo de confiança para os cenários 2, 4 e 6, que tem distribuição normal, é 0. Notavelmente, destaca-se que o cenário 1 se difere por ter o menor intervalo de confiança, sugerindo uma maior precisão nas estimativas.

Figura 16 – Bootstrap - Solicitações com Falha



Fonte: Elaborado pela autora

Na Tabela 9, ao analisar detalhadamente os valores dos limites do intervalo de confiança e a média amostral, destaca-se que os cenários que utilizam o Nginx como *web server* exibem uma incidência de falhas consideravelmente menor. Esta observação sugere a possibilidade de que o Apache, ao contrário do Nginx, talvez não esteja gerenciando de forma eficiente o volume de solicitações requisitadas. A disparidade nos resultados ressalta diferenças substanciais no desempenho entre os *web servers* avaliados, indicando uma inclinação favorável à escolha ou preferência pelo Nginx em cenários onde a minimização de falhas é crucial.

Tabela 9 – Resultados Bootstrap - Solicitações com Falha

Cenário	Limite Inferior	Média	Limite Superior
Cenário 1	2,1	2,733	3,6
Cenário 2	0,0	0,0	0,0
Cenário 3	8,833	12,633	19,133
Cenário 4	0,0	0,0	0,0
Cenário 5	4,7	5,867	7,433
Cenário 6	0,0	0,0	0,0

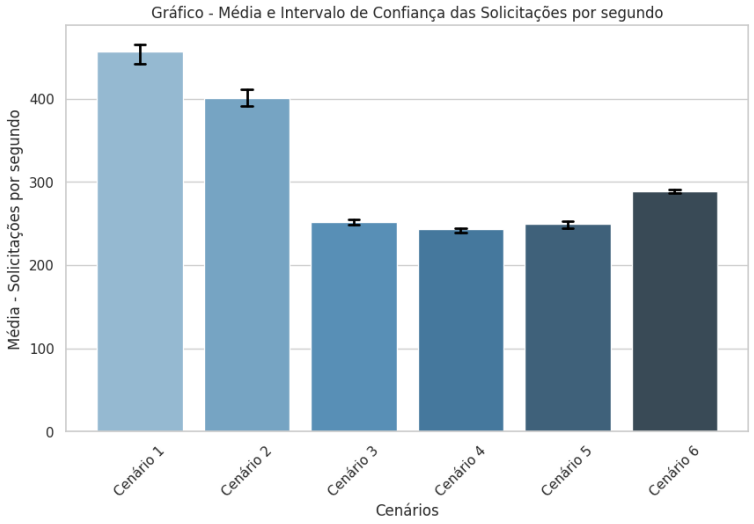
Fonte: Elaborado pela autora.

5.2.2 Solicitações por segundo

O método foi utilizado para calcular o intervalo de confiança da métrica de solicitações por segundo, considerando a média dos dados das amostras.

Na Figura 17 é visível que o intervalo de confiança mais amplo pertence ao cenário 1, enquanto o menor é observado no cenário 6. Além disso, ao notar e analisar que nos cenários 3, 4 e 5 os intervalos de confiança se sobrepõem, essa sobreposição sugere a inexistência de diferenças estatisticamente significativas entre esses cenários.

Figura 17 – Bootstrap - Solicitações por Segundo



Fonte: Elaborado pela autora

Na Tabela 10, é possível realizar uma análise mais detalhada dos valores dos limites do intervalo de confiança, tanto o inferior quanto o superior, além da média. Nota-se que o cenário que demonstra um desempenho superior é o cenário 1, sendo capaz de atender um maior número de solicitações por segundo. Isso sugere a possibilidade de que nos outros cenários possam existir gargalos que afetam o desempenho, principalmente o cenário 4, que é o que tem o pior desempenho na referida métrica.

Tabela 10 – Resultados Bootstrap - Solicitações por Segundo

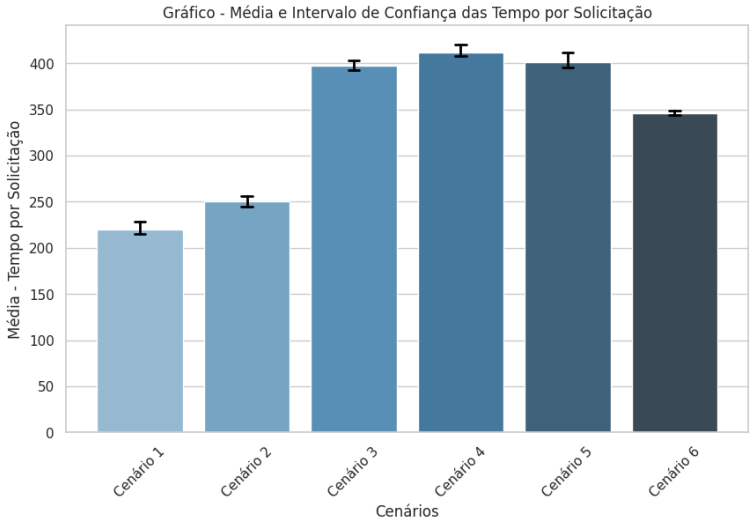
Cenário	Limite Inferior	Média	Limite Superior
Cenário 1	438,86	468,65	483,91
Cenário 2	394,26	404,88	417,64
Cenário 3	248,77	252,11	254,70
Cenário 4	239,01	243,27	244,92
Cenário 5	244,47	250,07	253,00
Cenário 6	286,81	288,87	290,98

Fonte: Elaborado pela autora.

5.2.3 Tempo por solicitação

Ao empregar este método na avaliação da métrica de tempo por solicitação, é possível realizar uma análise mais aprofundada dos intervalos de confiança na média associados a cada cenário proposto. Ao examinar a Figura 18 subsequente, destaca-se que o cenário 6 se distingue pelo menor intervalo de confiança, enquanto o cenário 1 exibe um intervalo mais amplo. Vale ressaltar que os cenários 3, 4 e 5 não se diferem estatisticamente, devido à sobreposição de seus intervalos.

Figura 18 – Bootstrap - Tempo por Solicitação



Fonte: Elaborado pela autora

Na Tabela 11 apresentada a seguir, torna-se viável realizar uma análise mais detalhada dos valores do limite inferior e superior de cada intervalo de confiança, calculados para cada cenário. Além disso, observa-se que a menor média da métrica de tempo por solicitação é associada ao cenário 1, ou seja, ao cenário que utiliza o *web server* Apache e o runC como *backend* do Docker. Dessa forma, podemos concluir que este cenário se destaca como aquele em que a métrica de tempo por solicitação atinge o menor valor médio, impactando de forma

direta na experiência do usuário em relação ao *web server*, já que um tempo por solicitação mais baixo geralmente resulta em respostas mais rápidas e, portanto, contribui para uma experiência do usuário mais satisfatória.

Na análise dessa métrica, destacam-se os cenários 4 e 5 como os de desempenho menos satisfatório, evidenciado pelo notável aumento no tempo médio por solicitação.

Tabela 11 – Resultados - Tempo por solicitação

Cenário	Limite Inferior	Média	Limite Superior
Cenário 1	207,30	214,60	232,27
Cenário 2	240,58	248,01	254,38
Cenário 3	392,94	397,08	402,53
Cenário 4	408,38	411,42	419,37
Cenário 5	395,71	400,76	411,54
Cenário 6	343,81	346,32	348,95

Fonte: Elaborado pela autora.

5.2.4 Tempo por solicitações concorrentes

Na análise representada pela Figura 19 a seguir, examinam-se os resultados obtidos por meio da aplicação do método *Bootstrap*. Esse método foi empregado com o propósito de avaliar os intervalos de confiança dos cenários em relação à métrica de tempo por solicitações concorrentes.

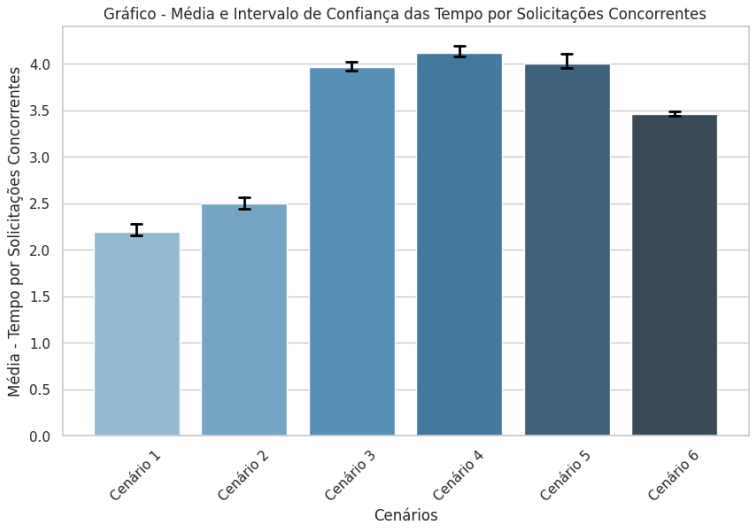
É notável que o cenário 1 exibe o maior intervalo de confiança. No entanto, em relação à métrica em questão, este cenário apresentou um desempenho superior em comparação aos outros cenários. Por outro lado, o terceiro cenário proposto se destaca pelo menor intervalo de confiança, sugerindo uma maior consistência nos resultados, apesar de ser um dos cenários que possui um tempo médio mais elevado por solicitações concorrentes.

Ademais, é relevante destacar que os cenários 3, 4 e 5 têm seus intervalos de confiança se sobrepondo entre si. Portanto, esses cenários não contribuem de maneira substancial para a análise comparativa, pois não há diferenças estatisticamente significativas entre eles.

Na Tabela 12 apresentada a seguir, é possível analisar os valores dos limites do intervalo de confiança, bem como observar a média do tempo por solicitações concorrentes.

Percebe-se que a menor média está presente no primeiro cenário proposto, onde se utiliza o servidor web Apache, o *backend* Containerd e o *backend* Runc em um ambiente Linux. Além disso, pode-se notar que o cenário que obteve um pior desempenho foi o quarto cenário

Figura 19 – Bootstrap - Tempo por Solicitações Concorrentes



Fonte: Elaborado pela autora

analisado, o mesmo pode star sem capacidade para ser escalável e suportar a quantidade de requisições feitas, sendo assim, possuem uma capacidade inferior para responder às solicitações, o que influencia negativamente na experiência do usuário.

Essa análise detalhada fornece evidências valiosas sobre a variabilidade e consistência dos dados, permitindo uma compreensão mais aprofundada dos cenários propostos, especialmente daquele que se destaca. Nesse cenário específico, a combinação de um servidor web Apache, *backend* Containerd e *backend* Runc em um ambiente Linux demonstra notável eficiência no processamento de solicitações concorrentes.

Tabela 12 – Resultados - Tempo por Solicitações Concorrentes

Cenário	Limite Inferior	Média	Limite Superior
Cenário 1	2,0724	2,1462	2,3128
Cenário 2	2,4066	2,48	2,5419
Cenário 3	3,9302	3,9709	4,0299
Cenário 4	4,0836	4,1142	4,1919
Cenário 5	3,9563	4,0076	4,113
Cenário 6	3,4377	3,4633	3,4888

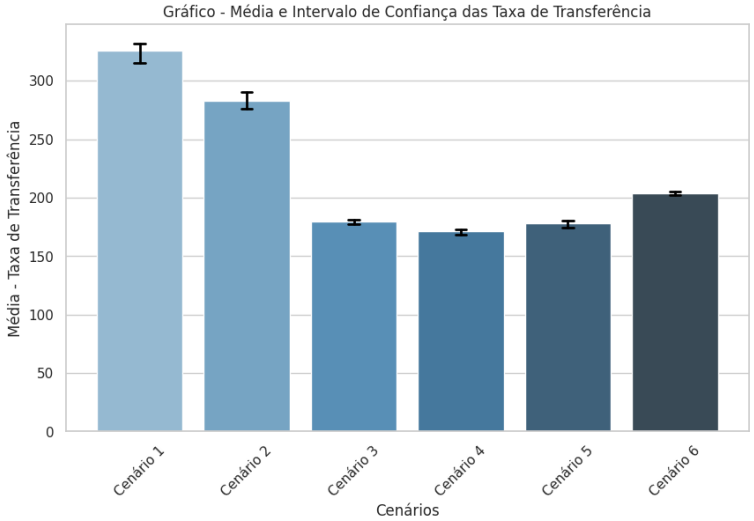
Fonte: Elaborado pela autora.

5.2.5 Taxa de transferência

Na Figura, é apresentada graficamente a análise dos intervalos de confiança calculados a partir do método proposto. Nota-se que o primeiro cenário apresentou um intervalo de confiança mais amplo, enquanto o último cenário exibiu um intervalo de confiança menor.

É relevante observar que alguns intervalos de confiança se sobrepõem, sendo essa sobreposição evidente nos cenários 3, 4 e 5.

Figura 20 – Bootstrap - Taxa de transferência



Fonte: Elaborado pela autora

Na Tabela 13 a seguir, é possível realizar uma análise mais detalhada dos valores do intervalo de confiança, juntamente com as médias associadas.

Tabela 13 – Resultados - Taxa de Transferência

Cenário	**Limite Inferior**	**Média**	**Limite Superior**
Cenário 1	313,36	334,08	345,12
Cenário 2	278,19	285,47	294,95
Cenário 3	177,19	179,66	181,45
Cenário 4	168,58	171,52	172,68
Cenário 5	174,13	178,24	180,26
Cenário 6	202,20	203,68	205,19

Fonte: Elaborado pela autora.

Pode-se inferir que o cenário que apresenta um intervalo de confiança mais amplo também é aquele que possui um desempenho superior em relação à métrica de taxa de transferência. Em outras palavras, é possível sugerir que os demais cenários podem enfrentar problemas de gargalos que interferem no desempenho do *web server*, sendo assim, não entregam uma boa experiência ao usuário.

Quando se considera a média dos dados dos diferentes cenários, destaca-se de maneira positiva o cenário 1, em contraste com o cenário 4, que se revela como o de desempenho mais desfavorável nessa última métrica examinada.

5.3 Discussão

Após análise detalhada dos resultados, torna-se evidente que, no que tange à normalidade dos dados, os cenários mais notáveis são o 2 e o 6. No cenário 2, que emprega o Nginx com *backend* Containerd e RunC no ambiente Linux, e no cenário 6, que utiliza o componente WSL no sistema operacional Windows, observamos uma notável conformidade com a distribuição normal em seus conjuntos de dados. Esses cenários destacam-se por apresentar uma maior aderência a padrões estatísticos regulares, contribuindo para uma interpretação mais confiável dos dados coletados.

Nos demais cenários, nos quais o teste de *Shapiro-Wilk* indica a não normalidade, observa-se uma presença de *outliers*. Essa notável quantidade de pontos discrepantes aponta para a existência de valores atípicos nesses cenários, indicando variações nos dados que se afastam do comportamento padrão. Com a presença de *outliers*, os resultados obtidos podem conter anomalias devido à dispersão desses valores atípicos, o que pode afetar negativamente os resultados.

Outro ponto relevante para análise é a precisão dos dados nos diferentes cenários, considerando especialmente os intervalos de confiança calculados por meio do método *Bootstrap*. O cenário que se destaca com um intervalo de confiança mais amplo, nas métricas analisadas, é o cenário 1. Em contrapartida, o cenário que se destaca com um intervalo de confiança mais estreito nas métricas é o cenário 6. Essa variação na amplitude dos intervalos de confiança entre os cenários pode influenciar na interpretação da consistência e confiabilidade dos resultados em cada contexto específico.

Com base nos resultados em relação às métricas, podemos identificar em qual aspecto cada cenário apresentou um desempenho superior. Essa análise é refletida na Tabela 14 a seguir.

Tabela 14 – Resultado do Desempenho dos Cenários por Métrica

Métrica	**Cenários com melhor desempenho**
Solicitações com Falha	Cenários 2, 4 e 6
Solicitações por Segundo	Cenário 1
Tempo por Solicitação	Cenário 1
Tempo por Solicitações Concorrentes	Cenário 1
Taxa de Transferência	Cenário 1

Fonte: Elaborado pela autora.

Os cenários 2, 4 e 6 destacaram-se positivamente na métrica de "solicitações com falha", pois não apresentaram qualquer falha no processamento das solicitações. Essa eficiência

pode ser atribuída ao uso do Nginx, conhecido por ser um ambiente altamente eficiente em situações de alto tráfego. A robustez do Nginx em lidar com um grande volume de solicitações simultâneas contribui para a ausência de falhas nessas condições, o que é crucial para garantir a estabilidade e confiabilidade do *web server*.

Nas demais métricas avaliadas, o cenário que se destacou pelo desempenho superior foi o cenário 1. A eficácia do cenário 1, no qual o *web server* Apache interage com o containerd e o runC no Docker em um ambiente Linux, pode ser atribuída à integração eficiente entre o *web server* e o ambiente de execução.

A escolha do *web server*, do *backend* e do sistema operacional em conjunto desempenhou um papel crucial nos resultados favoráveis alcançados. Vale ressaltar que, como o Docker foi inicialmente desenvolvido para uso no Linux, o desempenho pode não ser tão otimizado em ambiente Windows. Sendo assim, a afinidade entre o Docker e o Linux pode ter impulsionado a eficiência operacional, resultando em tempos por solicitações mais rápidos, bons resultados quando existem solicitações simultâneas, alto número de solicitações atendidas por determinado tempo e melhor taxa de transferência.

Pode-se concluir que a escolha do *backend* teve influência nos resultados obtidos. Levando em consideração que o runC é conhecido por sua leveza e simplicidade, podendo ter se alinhado de maneira mais eficaz com as otimizações do *web server* Apache para ambiente Linux. Enquanto isso, o *backend* gVisor, apesar de oferecer camadas adicionais de segurança, pode ter introduzido uma complexidade desnecessária para cenários específicos, resultando em menor eficiência de desempenho.

Desta forma, a escolha do runC como *backend* pode ter contribuído para a eficiência operacional e o alto desempenho observados no cenário 1.

6 CONCLUSÃO

Este estudo empregou uma análise de desempenho de *web servers* em distintos ambientes de execução. Tal avaliação se torna crucial para extrair conclusões fundamentais que não apenas orientam a seleção do cenário mais adequado, mas também visam otimizar o desempenho em implementações futuras.

Para alcançar o objetivo proposto, foram realizadas diversas etapas, incluindo a seleção cuidadosa de tecnologias e ferramentas, bem como a definição dos cenários e métricas a serem empregados para obter resultados significativos. Além disso, para conduzir uma análise de desempenho eficiente com os dados coletados, tornou-se indispensável a realização de testes estatísticos, especialmente o teste de Shapiro-Wilk e a aplicação do método Bootstrap. Essas abordagens metodológicas foram essenciais para fundamentar e validar as conclusões derivadas da avaliação de desempenho.

Houve algumas dificuldades em relação à montagem do ambiente de execução dos cenários, como a configuração do gVisor para uso no Docker, ainda estando com o *backend* runC instalado. Além disso, surgiram problemas com o Hyper-V no Windows, uma vez que o sistema operacional utilizado, Windows 11 Home, não é compatível com essa tecnologia. Vale ressaltar que o Docker já vem instruído para ser usado com o WSL, tornando desnecessário um cenário com o Windows utilizando o Hyper-V como *backend*, isso pelo fato de que o WSL permite a execução do Docker sem a necessidade de uma máquina virtual, resultando em uma solução mais leve e eficiente, sendo assim, esse cenário foi descartado.

A análise de desempenho revelou que o cenário 1, que utiliza o *web server* Apache com containerd e runC como *backend* do Docker, se destacou positivamente em quatro das cinco métricas analisadas. Contudo, é importante ressaltar que esse cenário apresentou uma quantidade significativa de solicitações com falha em uma das métricas analisadas.

Em perspectiva de trabalhos futuros, uma direção promissora de pesquisa seria a avaliação do desempenho de *web servers* e *backends* do Docker em diferentes provedores de nuvem, como a *Amazon Web Services* e a *Microsoft Azure*. Esta análise em ambientes de nuvem permitiria compreender como as características distintas e infraestruturas de cada provedor podem influenciar no comportamento e na eficiência das soluções baseadas em *containers*. A análise poderia abranger adaptações de desempenho dos *web servers* às peculiaridades de cada provedor, eficácia dos *backends* do Docker em ambientes de nuvem, além de comparações de desempenho entre os provedores. Uma pesquisa dessa natureza traria resultados fundamentais

para profissionais de Tecnologia da Informação (TI) e desenvolvedores que buscam otimizar suas aplicações na nuvem.

REFERÊNCIAS

ALBERTIN, M. R.; KOHL, H.; ELIAS, S. J. B. **Manual do Benchmarking**. [S.l]: Imprensa Universitária, 2016.

CHANDRA, A. Y. Analisis performansi antara apache & nginx web server dalam menangani client request. **Jurnal Sistem dan Informatika (JSI)**, v. 14, n. 1, p. 48–56, 2019.

CONTAINERD. **Containerd**. 2023. Disponível em: https://github.com/containerd/containerd. Acesso em 18 de maio de 2023.

CONTAINERS, O. **runC - GitHub Repository**. 2023. Disponível em: https://github.com/opencontainers/runc. Acesso em 18 de maio de 2023.

COSTA, G. F.; SANTOS, C. D. **Utilização de containers: um contexto histórico**. **Revista Científica e-Locução**, v. 1, n. 20, p. 23, 2021. Disponível em: https://periodicos.faex.edu.br/index.php/e-Locucao/article/view/388. Acesso em: Acesso em 04 maio 2023.

DICICCIO, T. J.; EFRON, B. **Bootstrap confidence intervals**. **Statistical science**, Institute of Mathematical Statistics, v. 11, n. 3, p. 189–228, 1996.

DOCKER. **Use containers to Build, Share and Run your applications**. 2023. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 08 maio 2023.

DOCKER. **Visão geral do Docker**. 2023. Disponível em: https://docs.docker.com/get-started/overview/. Acesso em: 11 maio 2023.

ESPE, L.; JINDAL, A.; PODOLSKIY, V.; GERNDT, M. **Performance evaluation of container runtimes.** [S.l: s.n], 2020. p. 273-281.

FERNANDES, E. M. d. G. **Estatística aplicada**. Braga: American Mathematical Society, 1999.

GVisor. **gVisor Documentation**. 2023. Disponível em: https://gvisor.dev/docs/. Acesso em 18 de maio de 2023.

HIRAKATA, V. N.; MANCUSO, A. C. B.; CASTRO, S. M. de J. Teste de hipóteses: perguntas que você sempre quis fazer, mas nunca teve coragem. **Clinical and Biomedical Research**, v. 39, n. 2, 2019.

INICIATIVE, O. C. **OpenContainers Specifications**. 2019. Disponível em: https://specs.opencontainers.org/. Acesso em: 17 maio 2023.

INICIATIVE, O. C. **Sobre a Iniciativa de Contêineres Abertos**. 2023. Disponível em: https://opencontainers.org/about/overview/. Acesso em: 17 maio 2023.

JAIN, R. **The Art of Computer Systems Performance Analysis**. New York, NY, USA: John Wiley & Sons, 1991. ISBN 978-0471503361.

KAISER, S.; HAQ, M. S.; TOSUN, A. ; KORKMAZ, T. Container technologies for arm architecture: A comprehensive survey of the state-of-the-art. **IEEE Access**, v. 10, p. 84853–84881, 2022.

KITHULWATTA, W.; JAYASENA, K.; KUMARA, B.; RATHNAYAKA, R. **Performance evaluation of docker-based apache and nginx web server**. [S.l], 2022. p. 1-6.

MICROSOFT. **Ferramentas de Plataforma de Contêiner no Windows**. 2023. Disponível em: https://learn.microsoft.com/pt-br/virtualization/windowscontainers/deploy-containers/containerd. Acesso em: 18 maio 2023.

MICROSOFT. **Visão geral da tecnologia Hyper-V**. 2023. Disponível em: https://learn.microsoft.com/pt-br/windows-server/virtualization/hyper-v/hyper-v-technology-overview. Acesso em 06 de novembro de 2023.

MICROSOFT. **Windows Subsystem for Linux (WSL)**. 2023. Disponível em: https://learn.microsoft.com/pt-br/windows/wsl/about. Acesso em 06 de novembro de 2023.

MICROSOFT. **WSL containers in Windows Subsystem for Linux**. 2023. Disponível em: https://learn.microsoft.com/pt-br/windows/wsl/tutorials/wsl-containers. Acesso em 06 de novembro de 2023.

MONTGOMERY, D. C.; RUNGER, G. C. **Estatística Aplicada e Probabilidade para Engenheiros**. 6. ed. Rio de Janeiro: LTC, 2016.

PUTRO, Z. P.; SUPONO, R. Comparison analysis of apache and nginx webserver load balancing on proxmox ve in supporting server performance. **International Research Journal of Advanced Engineering and Science**, v. 7, n. 3, p. 144–151, 2022.

ROMERO, D. **Containers com Docker do desenvolvimento à produção**. [*S. l.*]: Casa do Código, 2016.

ROYSTON, P. Approximating the shapiro-wilk w-test for non-normality. **Statistics and computing**, Springer, v. 2, p. 117–119, 1992.

SERGEEV, A.; REZEDINOVA, E.; KHAKHINA, A. **Docker container performance comparison on Windows and Linux operating systems**. [S.l: s.n], 2022. p. 1-4.

SERVICES, A. W. **Latency and Throughput**. 2023. Disponível em: https://aws.amazon.com/pt/what-is/latency/. Acesso em 13 de novembro de 2023.

SERVICES, A. W. **What is Containerization?** 2023. Disponível em: https://aws.amazon.com/pt/what-is/containerization/. Acesso em 24 de maio de 2023.

TURNBULL, J. **The Docker Book**. [*S. l.*: *s. n.*], 2017.

VITALINO, J. F. N.; CASTRO, M. A. N. **Descomplicando o Docker**. [*S. l.*]: Brasport, 2018.

W3TECHS. **W3Techs - extensive and reliable web technology surveys**. 2023. Disponível em: https://w3techs.com/. Acesso em: 18 maio 2023.

XAVIER, M. G.; NEVES, M. V.; ROSSI, F. D.; FERRETO, T. C.; LANGE, T.; ROSE, C. A. F. D. **Performance evaluation of container-based virtualization for high performance computing environments**. [S.l: s.n], 2013. p. 233-240.