UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO TÉCNICO INDUSTRIAL DE SANTA MARIA
REDES DE COMPUTADORES

Jardel Batista Gonçalves

# APRIMORANDO A EFICIÊNCIA: SOFTWARE DE GERENCIAMENTO PARA SERVIDORES NGINX VIRTUALIZADOS

Santa Maria, RS
2024

Jardel Batista Gonçalves

# APRIMORANDO A EFICIÊNCIA: SOFTWARE DE GERENCIAMENTO PARA SERVIDORES NGINX VIRTUALIZADOS

Trabalho de Conclusão de Curso apresentado ao Curso Superior em Redes de Computadores, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Redes de Computadores**.

Orientador: Prof. Renato Preigschadt de Azevedo

Santa Maria, RS
2024

**Jardel Batista Gonçalves**

# APRIMORANDO A EFICIÊNCIA: SOFTWARE DE GERENCIAMENTO PARA SERVIDORES NGINX VIRTUALIZADOS

Trabalho de Conclusão de Curso apresentado ao Curso Superior em Redes de Computadores, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Redes de Computadores**.

**Aprovado em 10 de dezembro de 2024:**

_____

**Renato Preigschadt de Azevedo, Dr. (UFSM)**
**(Presidente/Orientador)**

_____

**Simone Regina Ceolin, Dra. (UFSM)**

_____

**Guilherme Dhein, Dr. (UFSM)**

Santa Maria, RS
2024

## DEDICATÓRIA

Dedico esta conquista à minha mãe e aos meus irmãos, cuja crença no meu potencial e apoio incondicional foram essenciais em cada etapa desta jornada.
Dedico também aos amigos verdadeiros, que estiveram ao meu lado nos momentos de alegria e nos desafios enfrentados ao longo do curso.

## AGRADECIMENTOS

A realização deste trabalho e a conclusão do curso de Redes de Computadores foram possíveis graças ao apoio e à contribuição de muitas pessoas que desempenharam papéis fundamentais em minha vida.

Agradeço a todos que, de alguma forma, contribuíram para essa jornada e, de forma especial:

- À minha mãe, Jussara de Freitas Batista, que me ensinou o valor da perseverança e da excelência em tudo o que fazemos. Sua dedicação e apoio incondicional não só durante o curso, mas ao longo de toda a minha vida, foram essenciais para esta conquista.

- Ao meu irmão, Jordan Nunes Gonçalves, minha fonte de inspiração desde a infância. Mais do que um irmão e melhor amigo, foi um guia, sempre me ajudando em todos os aspectos e me orientando com o cuidado de um pai.

- Ao meu irmão, Jhonatan Batista de Freitas, que esteve ao meu lado em todas as etapas, demonstrando preocupação genuína com meu desempenho e fazendo sempre o possível para me ajudar.

- Aos meus amigos, em especial ao Gabriel Alexandre Folleto e ao Vinícios Glaser Ramos, que são companheiros em todos os momentos, tanto nos desafios acadêmicos quanto fora deles, demonstrando amizade verdadeira e apoio constante.

- Aos professores, em especial à professora Simone Regina Ceolin e ao professor Renato Preigschadt De Azevedo, que me acompanharam desde o início do curso, fornecendo suporte, conhecimento e incentivo em cada etapa.

Por fim, meu agradecimento a todos que, de alguma forma, fizeram ou fazem parte da minha vida e que contribuíram para minha formação. Cada gesto de apoio e cada palavra de incentivo foram fundamentais para que eu chegasse até aqui.

Sorte é o que acontece quando a preparação encontra a oportunidade.

(Séneca)

# RESUMO

## APRIMORANDO A EFICIÊNCIA: SOFTWARE DE GERENCIAMENTO PARA SERVIDORES NGINX VIRTUALIZADOS

AUTOR: Jardel Batista Gonçalves
Orientador: Renato Preigschadt de Azevedo

Com o avanço contínuo da tecnologia da informação, novas soluções e tecnologias emergem, trazem consigo desafios crescentes para os administradores de rede. Este cenário exige uma atualização constante dos conhecimentos, já que a evolução tecnológica é constante. Embora existam diversas ferramentas disponíveis no mercado para o gerenciamento de redes e servidores, muitas delas são de custo elevado para aproveitar completamente seus recursos. O *software* desenvolvido chamado Lynx integra diversas tecnologias, utilizando linguagens de programação, bibliotecas Python e um banco de dados que centraliza informações de monitoramento e automação. O sistema foi projetado para monitorar métricas essenciais de servidores Nginx, como uso de CPU, memória RAM, taxa de requisições por segundo (RPS), latência de resposta e outras, analisando essas métricas para tomar decisões dinâmicas e escaláveis, como o provisionamento automático de novos contêineres e ajustes no balanceamento de carga. A solução desenvolvida destaca-se por sua escalabilidade e flexibilidade, permitindo a incorporação de novos contêineres Linux (LXC) de maneira dinâmica, sem comprometer o desempenho geral do ambiente. Com isso, o Lynx se apresenta como uma alternativa acessível e eficaz para o gerenciamento de servidores web que utilizam o Nginx e contêineres LXC.

**Palavras-chave:** *Software*. Tecnologias. Gerenciamento. *Nginx*. Escalável.

# ABSTRACT

## ENHANCING EFFICIENCY: MANAGEMENT SOFTWARE FOR VIRTUALIZED NGINX SERVERS

AUTHOR: Jardel Batista Gonçalves
ADVISOR: Renato Preigschadt de Azevedo

With the continuous advancement of information technology, new solutions and technologies emerge, bringing with them increasing challenges for network administrators. This scenario demands constant updates in knowledge, as technological evolution is relentless. Although there are various tools available in the market for managing networks and servers, many of them are expensive to fully utilize their features. Lynx integrates a variety of technologies, leveraging programming languages, Python libraries, and a database to centralize monitoring and automation information. The system was designed to monitor essential metrics of Nginx servers, such as CPU usage, RAM usage, requests per second (RPS), response latency, and others. It analyzes these metrics to make dynamic and scalable decisions, such as the automatic provisioning of new containers or adjustments to load balancing. The developed solution stands out for its scalability and flexibility, allowing the dynamic incorporation of new Linux containers (LXC) without compromising the overall performance of the environment. With this, Lynx presents itself as an accessible and efficient alternative for managing web servers that utilize Nginx and LXC containers.

**Keywords:** Software. Technologies. Management. Nginx. Scalable.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| AWS | *Amazon Web Services* |
| CSRF | *Cross-site request forgery* |
| CSS | *Cascading Style Sheets* |
| CSV | *Comma-separated values* |
| DDoS | *Distributed Denial of Service* |
| DNS | *Domain Name System* |
| IPv4 | *Internet Protocol version 4* |
| HTML | *HyperText Markup Language* |
| HTTP | *Hypertext Transfer Protocol* |
| LXC | *Linux Containers* |
| LXD | *Linux Containers Daemon* |
| MTV | *Model-Template-View* |
| ORM | *Object-Relational Mapping* |
| RAM | *Random Access Memory* |
| RPS | Requisições por segundo |
| SCP | *Secure Copy Protocol* |
| SNMP | *Simple Network Management Protocol* |
| SLAAC | *Stateless Address Autoconfiguration* |
| SSL | *Secure Sockets Layer* |
| SO | Sistema Operacional |
| SSH | *Secure Shell* |
| TI | Tecnologia da Informação |
| TLS | *Transport Layer Security* |
| URL | *Uniform Resource Locator* |
| VM | Máquina Virtual |
| VMM | *Virtual Machine Monitor* |
| WoL | *Wake-on-Lan* |
| XSS | *cross-site scripting* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Com a expansão e crescente demanda da Internet, a gestão da infraestrutura de Tecnologia da Informação (TI) tornou-se essencial. Um eficaz gerenciamento requer o estabelecimento de um fluxo eficiente no tráfego de informações, garantindo a utilização adequada e não sobrecarga dos recursos, além do transporte confiável e seguro dos dados (NETO, 2021).

Dentro da infraestrutura de Tecnologia da Informação (TI), destacam-se os servidores web, cujo propósito fundamental é fornecer o conteúdo requisitado por usuários finais através da internet. Esses servidores desempenham funções críticas ao armazenar, processar e entregar o conteúdo web solicitado por usuários que acessam sites ou aplicações online. Esses ambientes são notoriamente complexos, demandando um controle rigoroso, e frequentemente são implementados em ambientes de contêineres, sendo o *Docker* uma escolha comum para essa finalidade (GOMES, 2019).

Ao incorporar servidores web por meio de virtualização, é evidente que o administrador de sistemas de TI enfrentará novos desafios em suas responsabilidades diárias. Nesse contexto, os administradores buscam soluções que simplifiquem suas tarefas, e uma ferramenta popular para essa finalidade é o Portainer.io[1]. Essa ferramenta é empregada no gerenciamento, monitoramento e automação de tarefas em ambientes de contêineres *Docker*, oferecendo uma abordagem mais eficiente e facilitando a administração desses sistemas complexos (BATISTA, 2022).

Neste projeto, será elaborado um software web destinado ao gerenciamento de servidores web que fazem uso da virtualização por meio do LXC. A implementação desse *software* será realizada utilizando a linguagem de programação Python, juntamente do *framework* Flask.

Serão empregadas ferramentas para simulação de requisições web, visando testar a aplicação desenvolvida e reproduzir cenários simulados realistas. O objetivo final consiste em proporcionar ao gestor de Tecnologia da Informação uma ferramenta que ofereça amplas facilidades, possibilitando o eficiente gerenciamento, monitoramento e automação dos servidores web.

## 1.1 JUSTIFICATIVA

A maioria das aplicações desenvolvidas para gerenciamento e automação de servidores web está direcionada aos usuários que adotam a tecnologia *Docker*. Adicionalmente, essas ferramentas geralmente requerem uma assinatura anual para o acesso total aos re-

[1] Acessível em: https://www.portainer.io/

cursos, tornando-se custosas para instituições que não dispõem de orçamento adequado para essa prioridade.

Diante desse cenário, torna-se evidente a necessidade de criar uma aplicação web voltada para o gerenciamento de servidores web que façam uso da tecnologia LXC/LXD para virtualização, utilizando o *Nginx* como servidor *Hypertext Transfer Protocol* (HTTP).

O projeto a ser desenvolvido consiste em uma solução de controle e automação de servidores web, completamente de código aberto[2], com o objetivo de proporcionar uma simplificação para o administrador de rede.

## 1.2 OBJETIVO GERAL

O objetivo geral deste trabalho consiste em desenvolver uma ferramenta abrangente para gerenciar, monitorar e automatizar servidores web virtualizados que empregam o *Nginx* como escalonador. O objetivo é facilitar a interação com o escalonador e automatizar o lançamento de contêineres conforme necessário, ao mesmo tempo em que fornece dados cruciais para os administradores. Esta proposta surge em resposta à crescente demanda por soluções de virtualização eficientes, visando simplificar a gestão e melhorar o controle dos recursos de *hardware*.

## 1.3 OBJETIVOS ESPECÍFICOS

- Prover uma ferramenta de *software* de código aberto, que visa o gerenciamento de servidores web;
- Utilizar LXC para hospedar os servidores web, que utilizarão *Nginx* tanto para ser um servidor HTTP como para realizar o balanceamento de carga entre os contêineres;
- Desenvolver o máximo do programa utilizando a linguagem de programação Python com o *framework* definido;
- Utilizar ferramentas como o Siege para realizar testes de desempenho em toda a arquitetura, abrangendo desde o gerenciador até o balanceador de carga e os próprios servidores web;
- Documentar e validar o *software* desenvolvido, com o objetivo de facilitar sua utilização.

[2]Software de código aberto é aquele cujo código-fonte é acessível e pode ser modificado e distribuído livremente.

## 1.4 ESTRUTURA

O restante deste trabalho segue a seguinte estrutura: no Capítulo 2, é apresentado o referencial teórico, que compreende o material utilizado no desenvolvimento desta pesquisa. O Capítulo 3 apresenta a metodologia proposta, detalhando o caminho que será seguido para alcançar os objetivos estabelecidos. O Capítulo 4 apresenta o cronograma das atividades propostas. Por fim, o Capítulo 5 abrange a conclusão deste trabalho.

# 2 REFERENCIAL TEÓRICO

Neste capítulo, serão destacados os principais estudos que fundamentaram o desenvolvimento do trabalho proposto. Será abordada a base tecnológica, visando proporcionar uma compreensão mais aprofundada desses elementos. Serão apresentados conceitos, tecnologias e informações essenciais que foram empregados no decorrer do desenvolvimento deste projeto.

## 2.1 SERVIDORES WEB

Há mais de 30 anos, iniciou-se a utilização de servidores web, responsáveis por armazenar os arquivos que compõem os sites, como documentos HTML, imagens, folhas de estilo e arquivos JavaScript, entregando-os aos dispositivos dos usuários finais. Esses servidores são configurados para gerenciar os componentes aos quais cada usuário tem acesso, controlando, por exemplo, os arquivos acessados por eles (MDN, 2023b).

Esses servidores adotam uma arquitetura cliente-servidor. Nesse modelo, quando o usuário pretende carregar o conteúdo do site, ele utiliza seu navegador para solicitar o acesso via internet. Esse procedimento é conhecido como uma requisição HTTP. O navegador busca o endereço IP do site solicitado, traduzindo a URL das páginas web por meio do *Domain Name System* (DNS). Esse processo tem como finalidade localizar o servidor (WEBER, 2023).

Após o servidor ser localizado, ele recebe a requisição HTTP do cliente e processa através do seu servidor HTTP. Assim que o servidor aceitar a requisição, ele procura pelos dados relevantes em seus arquivos. Em seguida, são retornados para o navegador que enviou a requisição os arquivos do site, e então, o usuário web consegue ver o conteúdo do site (WEBER, 2023).

Além disso, é importante destacar que um servidor web pode ser classificado como estático ou dinâmico. Um servidor web estático consiste em um computador e um software HTTP, os quais enviam os arquivos de um site de volta para o navegador sem realizar alterações. Por outro lado, um servidor web dinâmico realiza atualizações nos arquivos hospedados antes de entregá-los por meio de um servidor HTTP. Essa abordagem possibilita a geração e envio de conteúdo dinâmico para o navegador (WEBER, 2023).

Na construção de um servidor web, uma das peças fundamentais que deve-se escolher é um servidor HTTP, que é um *software* responsável por atuar como um intermediário entre os clientes da web e os recursos que estão sendo solicitados, facilitando a comunicação e a entrega do conteúdo (MDN, 2023b).

A Figura 1 apresenta de forma simplificada como funciona uma requisição para o

servidor web.

Figura 1 – Requisição web simplificada


Fonte: Adaptado de MDN (2023b).

Diversos *softwares* atuam como servidores HTTP, destacando-se o *Nginx* e o Apache. A Seção 2.2 abordará discussões sobre a escolha do *Nginx* em vez do Apache para o desenvolvimento do trabalho.

## 2.2 NGINX VS APACHE

*Nginx* e Apache são duas principais opções de *software* de servidor HTTP. O Apache usa uma arquitetura orientada a processos, enquanto o *Nginx* adota uma arquitetura assíncrona orientada a eventos (REESE, 2008).

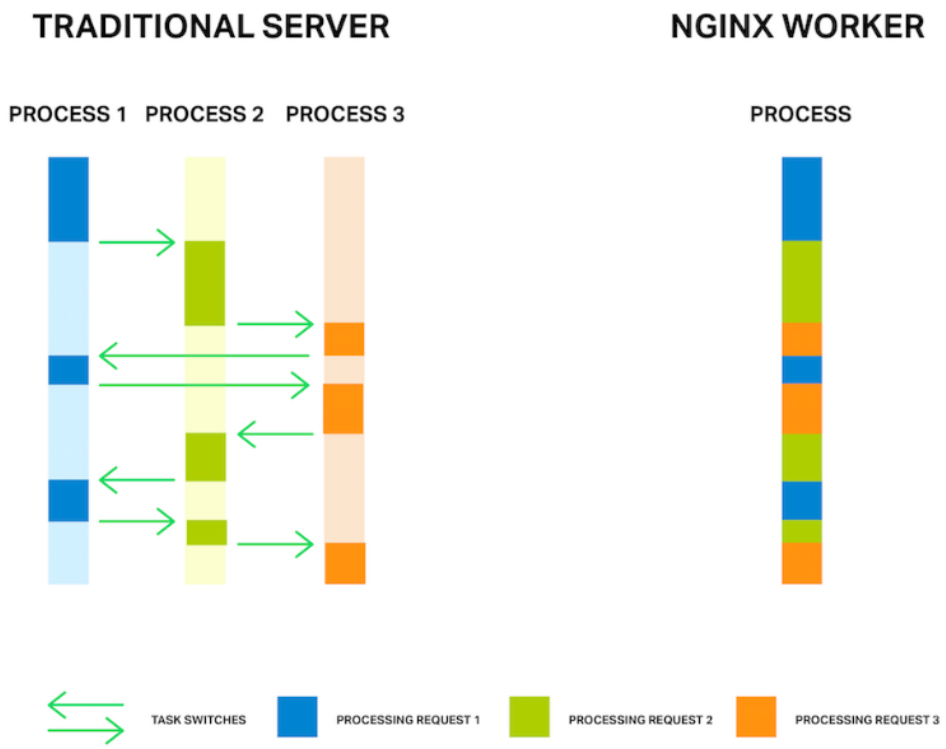
Em uma arquitetura orientada a processos, cada solicitação de conexão é tratada por um processo separado, e assim cada processo tem seu próprio espaço de memória e *thread* dedicado para lidar com a requisição. Embora isso proporcione isolamento entre as solicitações, também pode resultar em uma sobrecarga significativa, pois a criação de um novo processo para cada solicitação pode consumir recursos consideráveis, especialmente em cenários de alto tráfego (OSMAN, 2023).

Já em uma arquitetura assíncrona orientada a eventos, em vez de criar um novo processo ou *thread* para cada solicitação, utiliza-se um número limitado de processos ou *threads* para gerenciar diversas conexões simultaneamente. Por exemplo, para atender a 1000 usuários, o *Nginx* pode operar com apenas 10 processos, enquanto o Apache requer 1000 processos, um para cada conexão. Cada processo adicional consome memória e tempo de CPU, além de aumentar a sobrecarga causada pela alternância entre processos (*context switching*), o que pode degradar o desempenho. Esse modelo assíncrono, ao otimizar o uso de recursos do sistema, é particularmente eficiente em cenários de alto tráfego, oferecendo maior escalabilidade e capacidade de atender múltiplas conexões de forma simultânea (OSMAN, 2023).

O *Nginx* não apenas desempenha o papel de servidor HTTP, mas também atua como um versátil proxy reverso e balanceador de carga, conferindo-lhe recursos adicio-

nais em comparação com o Apache. Sua notável eficiência computacional resulta em um consumo reduzido de recursos, enquanto seu suporte eficaz para *Secure Sockets Layer* (SSL) e *Transport Layer Security* (TLS) possibilita a configuração de conexões seguras. Além disso, o *Nginx* oferece recursos integrados para aprimorar a segurança, incluindo controle de acesso, limitação de taxa e proteção contra ataques *Distributed Denial of Service* (DDoS) (LONGEN, 2023). A Figura 2 apresenta uma comparação entre os tipos de arquiteturas.

Figura 2 – Tipos de arquitetura para lidar com requisições




Fonte: Bartenev (2015).

Essas características fazem do *Nginx* uma escolha destacada para ambientes web que demandam desempenho otimizado e robustez na gestão de tráfego, ou seja, é o servidor HTTP a ser utilizado nesse projeto.

## 2.3 VIRTUALIZAÇÃO

A virtualização é um tópico sempre relevante no campo da computação, embora não seja uma novidade. A ideia de máquina virtual (VM) foi introduzida em torno do início dos anos 70 com a chegada da linguagem de programação Java. O uso dessas VMs tornou-se popular devido à prática comum da época, em que cada computador, mesmo de um único fabricante, possuía seu próprio SO. A utilização de VMs possibilitava que *soft-*

*ware* legado fosse executado em computadores mais modernos, independentemente do SO presente. Essa abordagem proporcionava uma maneira eficaz de preservar e prolongar a funcionalidade de *softwares* em ambientes tecnologicamente evoluídos (CARISSIMI, 2008).

Um SO é uma camada de *software* posicionada entre o *hardware* e as aplicações, facilitando tarefas para os usuários com o objetivo de aprimorar a eficiência e a conveniência no uso do computador. Com o passar dos anos, o desenvolvimento de *hardwares* mais poderosos, repletos de recursos computacionais, tornou-se evidente. Através da criação de VMs, tornou-se possível aproveitar de maneira mais eficiente esses recursos. Isso é viabilizado pela capacidade de executar diversos sistemas operacionais dentro do mesmo *hardware*, permitindo a otimização do aproveitamento dos recursos disponíveis em cada um deles (COSTA; SANTOS, 2021).

Para viabilizar a execução de diversos sistemas operacionais em um único *host* físico, é essencial contar com um componente de *software* conhecido como hypervisor, também denominado *Virtual Machine Monitor* (VMM). Sua função primordial é estabelecer uma camada de virtualização entre o *hardware* físico do computador e as VMs em execução. O hypervisor desempenha um papel crucial ao facilitar o compartilhamento eficiente dos recursos do *host* com as VMs, ao mesmo tempo em que assegura uma separação completa entre elas. Além disso, fornece interfaces de *hardware* virtuais essenciais para o funcionamento adequado das VMs. Essa abordagem possibilita a utilização simultânea de diferentes sistemas operacionais em um único ambiente físico, sendo uma camada adicional, conforme apresenta a Figura 3 (SILVA; SANTOS, 2016).

Figura 3 – Ambiente com utilização de máquina virtual


Fonte: Adaptado de COSTA e SANTOS (2021).

### 2.3.1 Contêineres e sua utilização

Contêineres são uma tecnologia de virtualização leve que permite encapsular aplicações e suas dependências em um único pacote executável. Diferente de máquinas virtuais (VMs), os contêineres compartilham o kernel do sistema operacional da máquina hospedeira, mas mantêm isolamento suficiente para executar aplicativos de forma independente. Isso significa que cada contêiner possui seu próprio sistema de arquivos, bibliotecas e configurações, garantindo que as aplicações sejam executadas de maneira consistente, independentemente do ambiente subjacente (LINUXCONTAINERS.ORG, 2023).

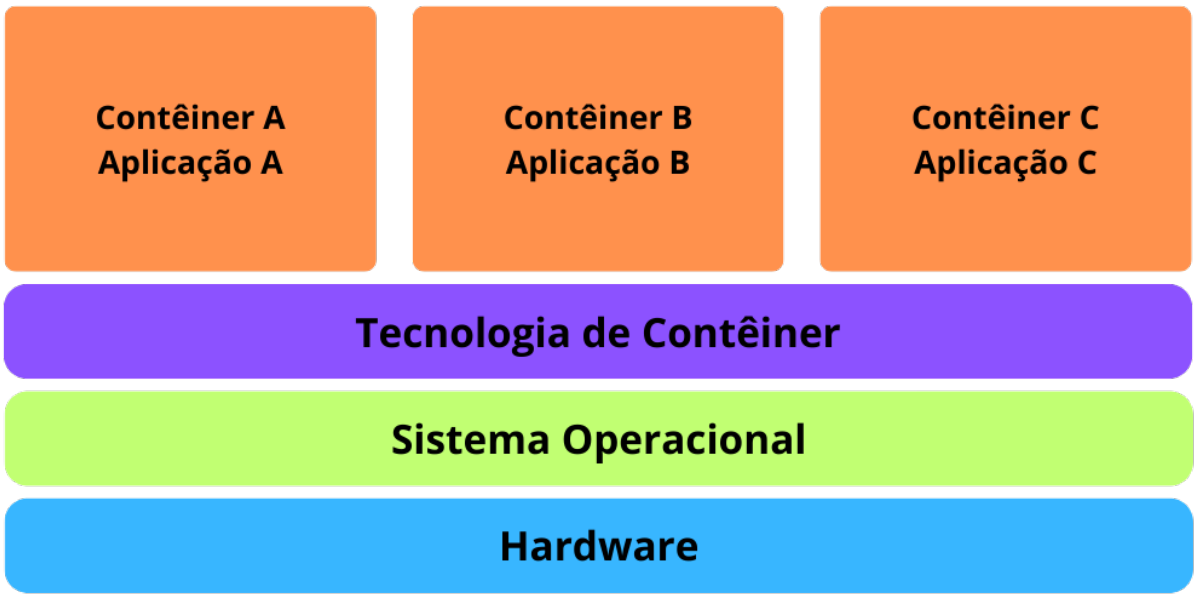
Conforme mencionado por Docker (2023), embora contêineres e VMs ofereçam benefícios semelhantes em termos de isolamento e alocação de recursos, eles operam de maneira distinta. Enquanto as VMs virtualizam tanto o sistema operacional quanto o *hardware* dos hosts, os contêineres virtualizam apenas o sistema operacional, resultando em uma eficiência significativa. Essas diferenças fundamentais na abordagem de virtualização contribuem para as características distintas de desempenho e eficiência de cada tecnologia.

As instâncias de contêineres são construídas a partir de imagens reutilizáveis que contêm todos os elementos necessários para executar a aplicação encapsulada na imagem específica. Isso geralmente inclui códigos-fonte, configurações, bibliotecas e outras dependências essenciais. O resultado é um pacote de software leve e autônomo, capaz de ser executado de maneira rápida e confiável em qualquer ambiente de computação que suporte a tecnologia (DOCKER, 2023).

Além disso, os contêineres oferecem não apenas eficiência e rapidez na execução de aplicativos, mas também promovem a portabilidade e a consistência do ambiente de desenvolvimento. Ao encapsular todas as dependências de uma aplicação em uma imagem, os desenvolvedores podem garantir que o software funcione de maneira consistente em diferentes ambientes, desde o ambiente de desenvolvimento até a produção.

Além disso, os contêineres facilitam a implantação e o dimensionamento de aplicativos, permitindo que novas instâncias sejam iniciadas rapidamente conforme necessário, o que é crucial em ambientes de microsserviços, por exemplo (LINUXCONTAINERS.ORG, 2023). A Figura 4 ilustra a arquitetura utilizada nas tecnologias de virtualização baseadas em contêineres.

Figura 4 – Ambiente com utilização de contêiner




Fonte: Adaptado de Docker (2023).

### 2.3.2 LXC/LXD

O LXC é uma tecnologia de virtualização de contêineres de nível de sistema operacional, que aproveita recursos de virtualização do kernel do Linux, como *namespaces* e *cgroups*, para isolar processos e recursos em ambientes independentes. Ele oferece uma abordagem leve e direta para criar e executar contêineres, permitindo que os usuários criem múltiplos sistemas Linux isolados em um único host Linux (LINUXCONTAINERS.ORG, 2023).

O LXD é uma extensão do LXC que fornece uma camada de gerenciamento de contêineres mais avançada e abrangente. Ele simplifica o processo de criação, implantação e gerenciamento de contêineres, oferecendo recursos como *snapshots*, migração de contêineres entre *hosts* e *clusters*, e gerenciamento de redes. Além disso, o LXD oferece uma interface de linha de comando para interagir com os contêineres de forma fácil e intuitiva (CASALICCHIO; IANNUCCI, 2020).

Uma das principais vantagens do LXC/LXD é seu alto desempenho e eficiência, pois os contêineres compartilham o mesmo kernel do host, evitando o *overhead* associado à virtualização de *hardware* tradicional. Isso permite que os usuários criem e executem contêineres com baixo impacto no desempenho do sistema e com uma sobrecarga mínima de recursos (ALEKSIC, 2022).

Além disso, o LXC/LXD é altamente escalável e flexível, permitindo que os usuários

criem e gerenciem contêineres em escala, desde ambientes de desenvolvimento locais até *clusters* de servidores em nuvem. Sua abordagem modular e extensível o torna uma escolha popular para uma variedade de casos de uso, incluindo desenvolvimento de aplicativos, testes de *software*, virtualização de servidores e implantação de aplicativos em ambientes de produção (ALEKSIC, 2022).

De acordo com Mendes e Duarte (2019), as ferramentas de virtualização LXC/LXD destacam-se por seu desempenho superior, apresentando eficiência nos recursos, como no desempenho do processador e na velocidade de comunicação da memória RAM. Portanto é por esses motivos que essa tecnologia será utilizada.

## 2.4 DESENVOLVIMENTO DE *SOFTWARE*

Desenvolver um sistema computacional é uma tarefa que demanda habilidade e dedicação, pois requer a análise, compreensão e resolução de um ou mais problemas específicos. Quando se trata do desenvolvimento de *software* para a plataforma web, a complexidade aumenta ainda mais, uma vez que diversos aspectos precisam ser considerados para garantir que o sistema seja acessível de forma remota e segura através de um navegador (MILETTO; BERTAGNOLLI, 2014).

De acordo com Miletto e Bertagnolli (2014), ao realizar um projeto de desenvolvimento web, é crucial adotar um processo sistematizado. Isso começa pela identificação do objetivo geral da aplicação. Além disso, é fundamental empregar um modelo de desenvolvimento incremental, no qual o software é construído em partes, sendo que cada uma delas é testada após a conclusão do código correspondente.

Dado que o propósito final deste projeto é criar uma ferramenta web que atue como um super gerenciador para servidores baseados em contêineres LXC/LXD, é crucial estabelecer as tecnologias a serem utilizadas tanto no desenvolvimento do *front-end*[1] quanto no do *back-end*[2].

Neste projeto, as principais tecnologias empregadas incluem HTML, CSS, MD-Bootstrap, Javascript e Python, todas integradas no desenvolvimento com o *framework* Django. O armazenamento de dados foi realizado com o SQLite, banco de dados embutido no Django. As interações com o banco foram implementadas por meio do *Object-Relational Mapping* (ORM), proporcionando uma abstração eficiente das consultas SQL e facilitando o gerenciamento das informações.

[1] *Front-end* desenvolve a interface de um site com HTML, CSS, JavaScript e suas bibliotecas, permitindo interação do usuário.

[2] *Back-end* é a parte invisível de um site, responsável pela lógica e funcionalidade, processando requisições do usuário e interagindo com o banco de dados.

### 2.4.1 HTML

Conforme destacado por MDN (2023a), o *HyperText Markup Language* (HTML) é a base estrutural de qualquer página da web, responsável por organizar e descrever o conteúdo exibido no navegador. Trata-se de uma linguagem de marcação amplamente adotada que define a estrutura e os elementos de uma página, como títulos, parágrafos, imagens, tabelas e links.

HTML funciona em conjunto com outras tecnologias essenciais, como CSS e JavaScript, mas seu papel é exclusivamente estrutural, fornecendo o esqueleto sobre o qual as páginas da web são construídas. A simplicidade e a flexibilidade do HTML garantem sua relevância, tornando-o indispensável no arsenal de qualquer desenvolvedor web (MDN, 2023a).

### 2.4.2 CSS

O *Cascading Style Sheets* (CSS) é uma linguagem de estilização fundamental no desenvolvimento web, projetada para definir a aparência e o layout das páginas. Ele complementa linguagens como HTML ao fornecer controle detalhado sobre elementos visuais, incluindo cores, fontes, margens, espaçamentos e tamanhos. O CSS separa a estrutura do conteúdo da apresentação visual, promovendo maior organização no código e facilitando a manutenção e a reutilização de estilos em diferentes páginas ou projetos (MDN, 2022).

Com o passar do tempo, o CSS evoluiu para atender às demandas do design moderno, introduzindo recursos como *media queries* para layouts responsivos, gradientes, animações e até variáveis. Essas inovações permitem a criação de interfaces mais dinâmicas e adaptáveis, essenciais para atender às necessidades de dispositivos móveis e telas de diferentes tamanhos. Através de sua flexibilidade e poder, o CSS se consolidou como uma ferramenta indispensável para desenvolvedores, ajudando a criar experiências visuais ricas e consistentes na web (MDN, 2022).

### 2.4.3 MDBootstrap

O *Material Design for Bootstrap* (MDBootstrap) é uma biblioteca de componentes *front-end* que combina o *framework* Bootstrap com o design visual do Material Design, criado pelo Google. Essa combinação proporciona uma maneira rápida e eficiente de criar interfaces de usuário modernas e responsivas, mantendo a familiaridade e a flexibilidade do Bootstrap, ao mesmo tempo em que incorpora os princípios estéticos do Material Design, como sombras, animações suaves e transições (MDBOOTSTRAP, 2024).

O MDBootstrap oferece uma ampla gama de componentes prontos para uso, como botões, *cards*, formulários, navegação, entre outros, todos estilizados de acordo com as diretrizes de Material Design. Além disso, a biblioteca inclui funcionalidades adicionais, como animações e efeitos interativos, que permitem aos desenvolvedores criar interfaces dinâmicas e atraentes sem a necessidade de escrever código adicional. Sua utilização acelera o desenvolvimento de aplicações web, proporcionando uma experiência de usuário mais agradável e intuitiva (MDBOOTSTRAP, 2024).

### 2.4.4 JavaScript

Conforme destacado por Flanagan (2012), JavaScript emerge como a linguagem de programação essencial para o futuro da web, sendo amplamente adotada na grande maioria dos sites modernos. Integra-se à tríade de tecnologias fundamentais que todos os desenvolvedores web devem dominar: HTML, para definir o conteúdo das páginas; *Cascading Style Sheets* (CSS), para determinar sua apresentação visual; e JavaScript, para governar seu comportamento dinâmico e interativo.

JavaScript é uma linguagem de programação de destaque, especialmente reconhecida por sua ampla aplicação na criação de interatividade em páginas da web. Com o avanço do desenvolvimento web, surgiram *frameworks* e bibliotecas JavaScript robustas, como o React, que facilitam significativamente a construção de interfaces complexas e dinâmicas (FLANAGAN, 2012).

### 2.4.5 Python

A linguagem de programação Python foi oficialmente criada em 1991 por Guido van Rossum, na Holanda. Atualmente, a Python é desenvolvida e mantida pela Python *Software Foundation*. É uma linguagem de programação de código aberto de alto nível, amplamente utilizada e disponível para várias plataformas de computação, além de ser multiparadigma, suportando tanto programação orientada a objetos quanto programação estruturada, por exemplo (MONTEIRO, 2023).

Uma característica distintiva do Python é sua sintaxe única, que torna a leitura e a escrita de código mais intuitivas. Ao contrário de linguagens como C, C++ e Java, o Python não utiliza chaves para definir blocos de código, e o uso do ponto e vírgula como final de comando é opcional, destacando sua legibilidade e simplicidade (MONTEIRO, 2023).

No desenvolvimento web, frameworks como Django, conhecido por criar aplicativos robustos e escaláveis, e Flask, ideal para aplicações menores, são amplamente utilizados (MONTEIRO, 2023). Além disso, bibliotecas como Python3-lxc permitem gerenciar con-

têineres diretamente no código, facilitando a automação e o gerenciamento de recursos, ampliando o potencial do Python na virtualização (GRABER, 2023).

### 2.4.6 Django

Django é um *framework* de desenvolvimento web escrito em Python, projetado para facilitar a criação de aplicações web robustas, seguras e escaláveis. Ele oferece uma série de ferramentas e bibliotecas prontas para uso, permitindo que os desenvolvedores se concentrem na lógica de negócios, sem precisar reinventar funcionalidades comuns, como autenticação de usuários, gerenciamento de banco de dados e envio de *emails* (DJANGO, 2024).

Uma das características mais notáveis do Django é o ORM, que abstrai as interações com o banco de dados, permitindo que os desenvolvedores manipulem os dados usando objetos Python, sem precisar escrever comandos SQL. Isso facilita a integração com diferentes sistemas de banco de dados e torna o código mais legível e fácil de manter. Além disso, o Django adota o padrão *Model-Template-View* (MTV), que organiza a aplicação em camadas bem definidas, promovendo uma estrutura modular que facilita a escalabilidade e manutenção dos projetos (DJANGO, 2024).

Ele também é amplamente reconhecido por sua segurançaqq, proporcionando proteções automáticas contra várias ameaças comuns, como ataques CSRF, SQL *Injection* e XSS. Sua grande comunidade de desenvolvedores e a documentação detalhada contribuem para sua popularidade, tornando-o uma escolha confiável tanto para projetos pequenos quanto para grandes sistemas empresariais. A flexibilidade, aliada a uma excelente performance e escalabilidade, garante que o Django seja uma das principais ferramentas para o desenvolvimento de aplicações web modernas (DJANGO, 2024).

### 2.4.7 SQLite

O SQLite é um sistema de gerenciamento de banco de dados relacional leve e embutido, que armazena dados em um único arquivo. Dentro do Django, o SQLite é a opção padrão para o armazenamento de dados em aplicações de desenvolvimento e testes. Sua principal vantagem é a simplicidade e a configuração mínima necessária, o que a torna ideal para projetos de pequeno a médio porte, além de ser especialmente útil para protótipos e testes rápidos. Ao ser integrado ao Django, o SQLite permite que os desenvolvedores criem bancos de dados relacionais sem se preocupar com a complexidade de um sistema de banco de dados externo, como MySQL ou PostgreSQL (SQLITE, 2024).

## 2.5 TESTE DE CARGA

A ferramenta Siege é uma solução open-source amplamente empregada para realizar testes de carga em servidores web. Por meio da simulação de múltiplos usuários acessando o servidor simultaneamente, o Siege avalia a performance e a resiliência do sistema frente a altos volumes de tráfego. Essa ferramenta é essencial para entender o comportamento do servidor em condições extremas, identificando gargalos e limitações antes que problemas impactem usuários finais (ABBAS; SULTAN; BHATTI, 2017).

Durante os testes, o Siege envia milhares de solicitações HTTP ao servidor alvo em intervalos configuráveis, permitindo personalizar cenários de teste que reflitam situações reais de uso. Essa abordagem possibilita não apenas testar o desempenho do servidor, mas também analisar sua estabilidade ao longo do tempo, especialmente em casos de picos de tráfego. Com isso, a ferramenta contribui significativamente para a identificação de problemas relacionados à escalabilidade e limitações de capacidade, fundamentais para a otimização do sistema (FULMER, 2004).

Além de sua aplicabilidade em servidores web, o Siege é versátil e pode ser utilizado em diferentes tipos de aplicações conectadas à internet, incluindo serviços baseados em APIs e plataformas de comércio eletrônico. Após os testes, a ferramenta gera relatórios detalhados com métricas como número de transações, disponibilidade, tempo de resposta, taxa de transferência, transações bem-sucedidas e com falha, além de informações sobre a simultaneidade e tempos extremos das transações. Essas métricas são indispensáveis para ajustes no código, nas configurações do servidor e na arquitetura da aplicação, assegurando que o sistema suporte a carga planejada sem interrupções e garanta a disponibilidade e a escalabilidade (FULMER, 2004).

O Quadro 1 apresenta o significado detalhado de cada métrica fornecida pela ferramenta, destacando sua importância no processo de análise e otimização de desempenho.

Quadro 1 – Dados disponibilizadas pela ferramenta Siege

| Dados | Características |
|---|---|
| **Transações** | Número de acessos ao servidor, pode ocorrer do número de transações exceder o número de acessos programados. Cada servidor acessado conta como uma transação. |
| **Disponibilidade** | Valor em porcentagem de conexões tratadas com sucesso pelo servidor. |
| **Tempo decorrido** | Tempo que leva para ser realizada a execução do teste, desde a execução do comando até a última transação feita. |
| **Dados transferidos** | A quantidade em Bytes de dados transferidos, inclui as informações do cabeçalho da requisição. |
| **Tempo de resposta** | O tempo médio necessário para responder às solicitações de cada usuário simulado. |
| **Taxa de transação** | O número médio de transações que o servidor foi capaz de processar por segundo. |
| **Taxa de transferência** | O número médio de bytes transferidos a cada segundo do servidor para todos os usuários simulados. |
| **Simultaneidade** | O número médio de conexões simultâneas, um número que aumenta à medida que o desempenho do servidor diminui. |
| **Transações bem-sucedidas** | O número de vezes que o servidor respondeu com um código de retorno < 400. |
| **Transações com falha** | O número de vezes que o servidor respondeu com um código de retorno >= 400 mais a soma de todas as transações de *sockets* com falha, que inclui tempos limite de *sockets*. |
| **Transações mais longa** | A maior quantidade de tempo que uma única transação levou, entre todas as transações. |
| **Transações mais curta** | A menor quantidade de tempo que uma única transação levou, entre todas as transações. |

Fonte: Adaptado de Fulmer (2004).

## 2.6 TRABALHOS RELACIONADOS

Neste capítulo, serão apresentados trabalhos que serviram como embasamento teórico e prático para o desenvolvimento do presente estudo, destacando suas semelhanças com o trabalho em questão.

### 2.6.1 *EVALUATION TEST AND IMPROVEMENT OF LOAD BALANCING ALGORITHMS OF NGINX*

O trabalho realizado por Ma e Chi (2022) representa uma evolução no algoritmo de balanceamento de carga do *Nginx*, concentrando-se no aprimoramento do algoritmo *Ip_Hash*. Este algoritmo é empregado para encaminhar solicitações de um mesmo endereço IP para um servidor específico em um conjunto de servidores, utilizando o valor de *hash* do endereço IP do cliente como critério. Esse algoritmo ao contrário do *Round Robin* ou do *Least Connections* permite uma consistência na sessão, já que o mesmo endereço IP sempre deve ser direcionado para o mesmo servidor do *cluster*.

Os autores propuseram duas melhorias no algoritmo, sendo a primeira aplicada ao cálculo do *hash*. Inicialmente, o algoritmo calculava o *hash* utilizando apenas os três primeiros números do endereço IP, o que resultava em muitas colisões de *hash*. Como consequência, as requisições não eram distribuídas uniformemente entre os servidores *upstream*. Para aprimorar essa abordagem, eles propuseram utilizar a operação XOR e realizar deslocamento de bits para a direita. Essa modificação foi projetada para reduzir significativamente as colisões de *hash*, proporcionando um balanceamento mais eficiente das requisições entre os servidores *upstream*.

A segunda melhoria proposta pelos autores consistiu em aprimorar o método de pesquisa para determinar qual nó (servidor) deve ser selecionado para atender a requisição. No algoritmo original, essa pesquisa era feita de forma sequencial, o que em certas circunstâncias demandava um tempo considerável para encontrar o nó adequado. Para melhorar esse processo, os autores optaram por implementar uma pesquisa binária. Para viabilizar essa abordagem, foi necessário ordenar os nós, o que foi realizado utilizando o algoritmo de ordenação conhecido como *quicksort*, que tem um desempenho eficiente na grande maioria dos casos. As chaves do vetor que representam cada servidor *upstream* é representada por um *hash* feito com um valor que representa o peso do servidor e o endereço IP.

Esse trabalho acaba abrindo uma gama de possibilidades de aprimoramento no projeto desenvolvido, já que ele trás uma melhoria considerável em um algoritmo de balanceamento de carga, que pode ser utilizado no desenvolver do projeto.

### 2.6.2 RUFUS: FERRAMENTA PARA O GERENCIAMENTO DE INFRAESTRUTURA PARA A EXECUÇÃO DE APLICAÇÕES EM CONTAINERS

Neste trabalho escrito por Souza et al. (2016), é destacada a utilização da tecnologia de virtualização, onde se tem o encapsulamento das aplicações em ambientes computacionais.

Neste contexto, os autores propuseram o desenvolvimento de uma ferramenta denominada Rufus, a qual oferece *appliances* de aplicações por meio do LXC. Além disso, foi criada uma interface gráfica intuitiva para gerenciar os contêineres, tornando o processo mais acessível para usuários que não estejam familiarizados com a linha de comando.

O trabalho proposto por Souza et al. (2016) apresenta notáveis semelhanças com o projeto em desenvolvimento aqui. Por exemplo, ambos fazem uso de contêineres LXC e da linguagem Python, aproveitando bibliotecas como a pylxc, que oferece uma variedade de recursos para a manipulação desses contêineres.

Além disso, ambos os projetos têm como objetivo desenvolver uma interface web para permitir que o usuário final gerencie os contêineres. No entanto, neste trabalho proposto, o foco está no desenvolvimento dessa interface para contêineres LXC que atuam como servidores web, utilizando o *Nginx* como servidor HTTP e balanceador de carga. Portanto, além das funções de manipulação dos contêineres LXC, a interface web deste projeto apresentará outras informações relevantes.

### 2.6.3 CONSTRUÇÃO E GERENCIAMENTO DE SISTEMAS DE MONITORAMENTO FOCADOS AO AMBIENTE CLOUD

O trabalho desenvolvido por Balbino (2021) destaca a importância da utilização de ferramentas de gerenciamento de sistemas em conjunto, especialmente para ambientes de nuvem. Essas ferramentas, quando combinadas, oferecem um poder significativo de análise e manipulação de dados, não apenas para ambientes de nuvem, mas também para servidores locais.

O autor optou por realizar o monitoramento utilizando as ferramentas Zabbix, Grafana e *New Relic*, com foco no ambiente da *Amazon Web Services* (AWS). Além disso, ele empregou a linguagem de programação Python e *scripts* em Shell para automatizar a integração das ferramentas mencionadas anteriormente. Esse trabalho foi desenvolvido para o gerenciamento de SO Linux com a utilização do *Crontab* para realizar rotinas automáticas.

O projeto desenvolvido por Balbino (2021) não se trata da criação de um novo *software*, mas sim da integração de um conjunto de ferramentas existentes para gerenciar um ambiente de forma eficiente. As semelhanças com o trabalho desenvolvido incluem

o uso da linguagem Python e possivelmente o uso do *crontab*[3]. Além disso, a ferramenta Grafana, que é *open-source* e disponível no *GitHub*[4], permite a geração de gráficos e *dashboards* a partir de dados de diversos bancos de dados, o que pode ser útil na construção da interface web para o projeto desenvolvido.

[3]Utilitário do Unix usado para agendar tarefas para serem executadas em momentos específicos.
[4]Plataforma de hospedagem de código-fonte e arquivos com controle de versão.

# 3 DESENVOLVIMENTO

Este capítulo descreve as etapas realizadas durante o desenvolvimento do trabalho, abrangendo desde a escolha das tecnologias utilizadas até a apresentação da ferramenta criada, que foi nomeada como Lynx. Cada fase do processo é detalhada para fornecer uma visão clara e organizada do percurso adotado.

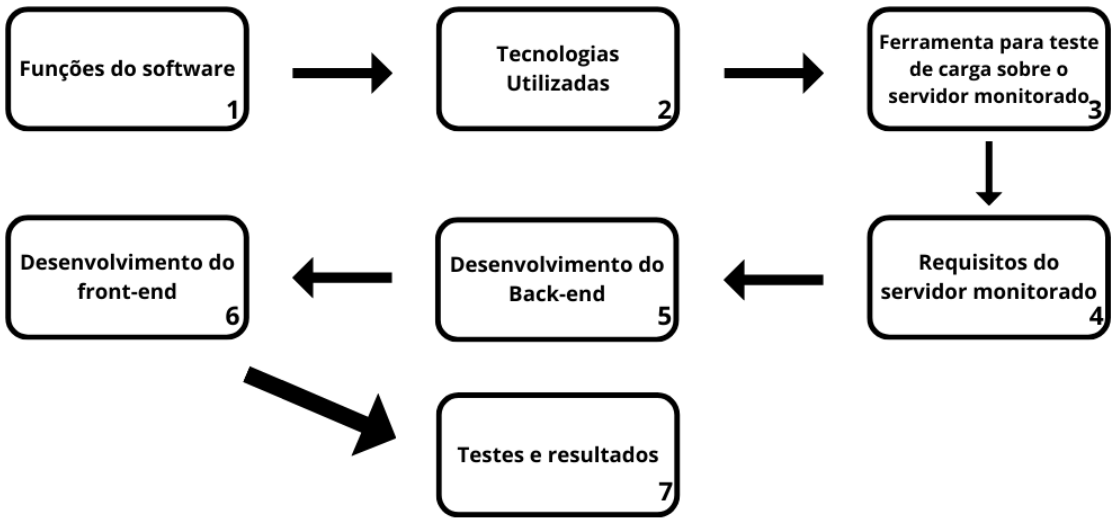
## 3.1 METODOLOGIA

Para garantir uma execução eficiente e estruturada do projeto, foi desenvolvido um cronograma de atividades que orientou o progresso das etapas. O primeiro passo foi a definição das funcionalidades do *software* web, identificando os problemas a serem resolvidos. Em seguida, foram selecionadas as tecnologias essenciais para a implementação do projeto, como as linguagens de programação, *frameworks* e banco de dados.

Simultaneamente, foi realizada a análise e escolha da ferramenta para realizar os testes no servidor web monitorado. Com as tecnologias e ferramentas definidas, iniciou-se o desenvolvimento do *back-end*, seguido pela construção da interface de *front-end*, visando proporcionar uma experiência de usuário fluida e intuitiva.

Por fim, foram conduzidos testes com a ferramenta apresentada na seção 3.4, a fim de avaliar a eficiência do *software* Lynx e identificar oportunidades de aprimoramento. A Figura 5 ilustra as etapas do projeto, detalhando o fluxo de desenvolvimento e as ações realizadas.

Figura 5 – Diagrama da metodologia




Fonte: Autor 2024.

O código-fonte completo deste projeto está disponível no repositório do GitHub: https://github.com/jbgoncalvess/Lynx

## 3.2 FUNÇÕES DO *SOFTWARE*

Entre as funcionalidades do *software* definidas no TCC A, apenas a implementação do sistema de notificação e a seleção do tipo de balanceamento de carga não foram realizadas como inicialmente planejadas, sendo esta última simplificada para uma variável configurável no *back-end*. Todas as demais funcionalidades propostas foram concluídas com êxito. Além disso, durante o desenvolvimento, foram identificadas e implementadas novas funcionalidades adicionais. Todas as funcionalidades do *software* estão descritas a seguir, seguidas de suas vantagens:

- Monitorar o balanceador de carga (servidor host), registrando diariamente os valores máximos e mínimos de contêineres em execução nos últimos 7 dias. Além disso, acompanhar o número de conexões ativas e as Requisições por Segundo (RPS) ao longo do tempo. Isso permite uma visão histórica do desempenho do servidor, facilitando a identificação de variações e ajudando na otimização dos recursos para lidar com picos de tráfego;

- Monitorar o status dos contêineres, exibindo informações como estado de execução, nome e endereços IP. Além disso, permitir a realização de ações manuais, como iniciar, pausar e reiniciar os contêineres. Também inclui a possibilidade de configurar novos endereços IPv4 e habilitar ou desabilitar endereços IPv6, que, por padrão, utilizam a *Stateless Address Autoconfiguration* (SLAAC). Esta funcionalidade facilita o gerenciamento da infraestrutura, permitindo ações rápidas e configurações de rede flexíveis, adequadas às diferentes necessidades do ambiente;

- Realizar o monitoramento de métricas dos contêineres, incluindo uso de CPU, memória RAM, disco, tempo de *uptime*, processos ativos e RPS. Esses dados são apresentados em gráficos interativos, atualizados automaticamente a cada 10 segundos. Com esses dados, o Lynx facilita a detecção imediata de problemas de desempenho e permitindo ajustes rápidos que aprimoram a eficiência do sistema;

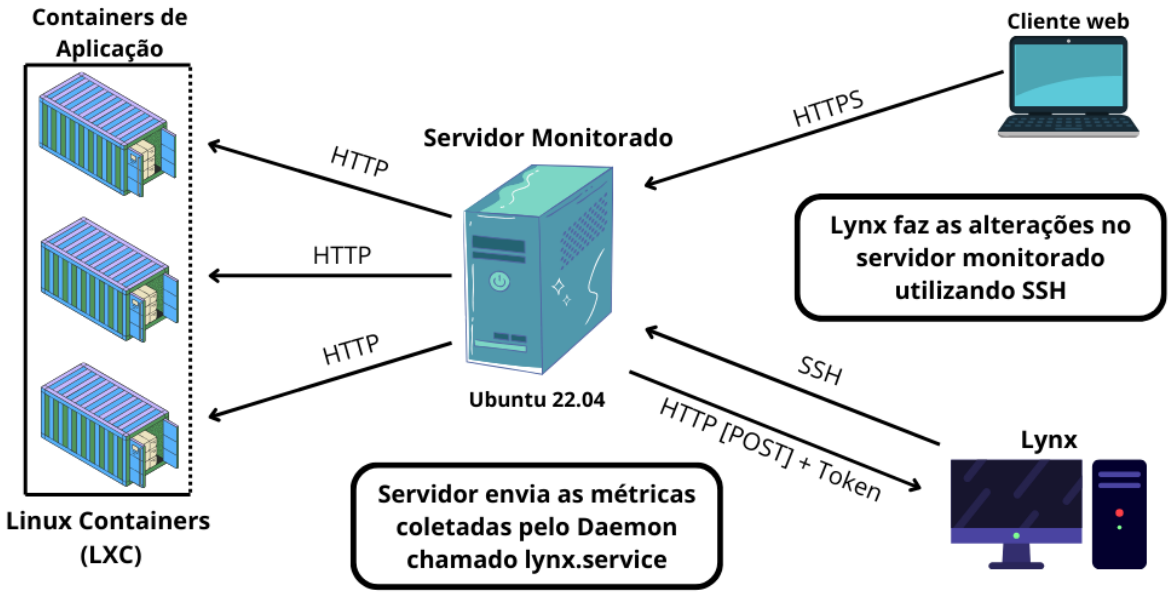
- Listar todas as imagens locais de contêineres disponíveis, exibindo informações detalhadas como nome, tipo de arquitetura, descrição, tamanho e data de *upload*, além de permitir a exclusão. Essa funcionalidade facilita a organização e a gestão das imagens, garantindo que o sistema mantenha apenas as imagens necessárias e otimize o uso do espaço de armazenamento;

- Atualizar automaticamente o arquivo de configuração do *Nginx* com base no status dos contêineres de aplicação. Contêineres em estado *RUNNING* são adicionados à lista de *upstream*, enquanto aqueles em estado *STOPPED* são removidos. Com isso, é garantido que o balanceamento de carga seja feito de maneira dinâmica, adaptando-se em tempo real às mudanças no status dos contêineres, sem necessidade de intervenção manual;

- Gerenciar automaticamente a inicialização e a interrupção dos contêineres de aplicação com base na média aritmética do uso da CPU de cada contêiner, otimizando o uso dos recursos de CPU, iniciando e interrompendo contêineres conforme necessário, garantindo eficiência e economia de recursos.

Durante o desenvolvimento da aplicação, foi identificado que seria mais vantajoso que o *software* fosse capaz de operar de forma independente, sem a necessidade de estar na mesma máquina que o servidor monitorado. Assim, o *software* foi projetado para se conectar via SSH a qualquer servidor monitorado, permitindo a coleta das métricas e a execução dos comandos necessários, seja a partir da interface web ou das funções automatizadas do *back-end*. Para que essa comunicação seja possível, o servidor monitorado deve ter o usuário lynx configurado, além de contar com um serviço desenvolvido em Python 3, responsável pelo envio das métricas para o *software*. A Figura 6 ilustra a infraestrutura utilizada pelo *software*.

Figura 6 – Infraestrutura utilizada no desenvolvimento da aplicação web

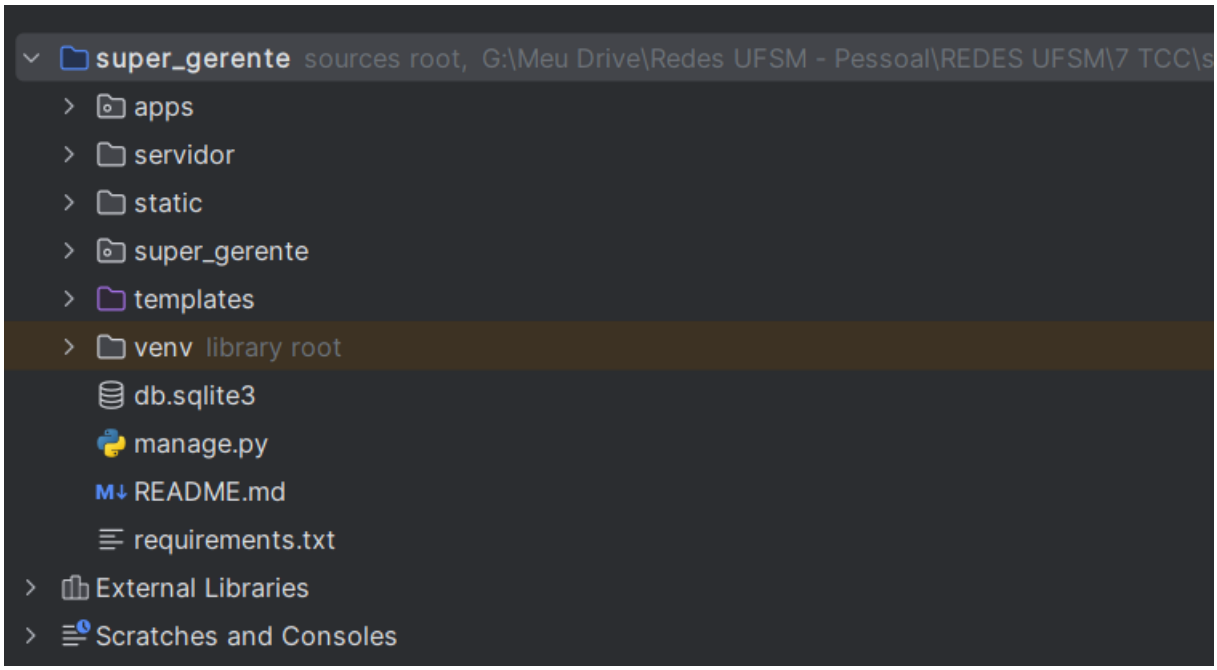



Fonte: Autor 2024.

## 3.3 TECNOLOGIAS UTILIZADAS

A Seção 2.4 detalhou as tecnologias escolhidas para o desenvolvimento do *front-end* e *back-end* da aplicação. Dentre elas, destaca-se a estrutura de diretórios do Django, que desempenhou um papel central no desenvolvimento da aplicação. A Figura 7 ilustra a organização completa dos diretórios do Django, evidenciando sua funcionalidade e contribuição para o projeto.

Figura 7 – Diretórios Django do projeto




Fonte: Autor 2024.

Como apresentado na Figura 7, a estrutura do projeto é composta por diversos diretórios e arquivos, que são explicados detalhadamente no quadro 2.

Quadro 2 – Estrutura de diretórios Django explicada

| Dados | Características |
|---|---|
| **super_gerente** | Diretório raiz do projeto, contendo a estrutura principal e todos os arquivos necessários para sua execução. |
| **apps** | Diretório que armazena os aplicativos criados, organizando suas *views*, *models* e demais arquivos específicos. |
| **servidor** | Contém os arquivos relacionados ao monitoramento do servidor, incluindo o serviço que coleta e envia informações da infraestrutura. |
| **static** | Reúne recursos estáticos, como imagens, arquivos CSS, JavaScript, YAML e chaves RSA. |
| **super_gerente (app)** | Aplicativo principal do projeto, responsável por configurações globais como fuso horário, idioma (ex.: pt-BR) e banco de dados. |
| **templates** | Diretório para os templates HTML, organizados por aplicativo. |
| **venv** | Ambiente virtual Python utilizado para gerenciar as dependências do projeto de forma isolada. |
| **db.sqlite3** | Arquivo de banco de dados utilizado pelo Django para armazenar informações, como as dos contêineres. |
| **manage.py** | Arquivo principal do projeto, usado para executar comandos administrativos e iniciar o servidor do Django. |
| **README.md** | Documento de descrição do projeto, frequentemente usado em repositórios como o GitHub. |
| **requirements.txt** | Lista de dependências do projeto, permitindo a instalação em novos ambientes sem a necessidade de copiar a *venv*. |
| **External Libraries** | Bibliotecas e dependências externas instaladas pelo gerenciador de pacotes *pip*, essenciais para o funcionamento do projeto. |
| **Scratches and Consoles** | Áreas de trabalho temporárias no ambiente de desenvolvimento, úteis para testes rápidos e uso do Python *shell*. |

Fonte: Autor 2024.

## 3.4 FERRAMENTA PARA TESTE DE CARGA SOBRE O SERVIDOR MONITORADO

A ferramenta escolhida para os testes de carga foi o Siege, conforme detalhado na seção 2.5. Executada via linha de comando, o Siege oferece grande flexibilidade, permitindo configurar diversos parâmetros, como o número de conexões simultâneas, a duração do teste e a quantidade de requisições por usuário. Esses parâmetros tornam a ferramenta altamente eficaz para simular diferentes cenários de carga e avaliar o desempenho de sistemas sob estresse.

O comando `siege -c 20 -t 5M http://192.168.2.2/` realiza um teste com 20 conexões simultâneas durante 5 minutos. É importante destacar que o número de conexões simultâneas não corresponde ao número total de requisições, uma vez que cada conexão pode realizar múltiplas requisições ao longo do tempo, tornando o teste mais robusto. Se o parâmetro `-t` for omitido, o teste continuará em execução até ser cancelado manualmente. Alternativamente, é possível substituir o parâmetro `-t` por `-r`, o que permite definir o número exato de requisições que cada usuário fará.

A utilização do Siege possibilitou a realização de testes e a análise das métricas coletadas, que foram exibidas nos gráficos desenvolvidos na aplicação Lynx. Os resultados dos testes, bem como a análise detalhada, estão apresentados na seção 4

## 3.5 REQUISITOS DO SERVIDOR MONITORADO

O servidor monitorado pelo Lynx deve atender a uma série de requisitos específicos para garantir o funcionamento adequado do sistema. Primeiramente, é necessário que o servidor utilize um sistema operacional Linux, possua um usuário chamado lynx com permissões de superusuário, e tenha o Python 3 instalado, essencial para a execução dos scripts responsáveis pelo envio de dados ao software Lynx. Além disso, o servidor precisa contar com o serviço lynx.service, configurado para ser iniciado automaticamente junto com o sistema, garantindo a execução contínua dos scripts de monitoramento.

Para simular um ambiente real, foi configurado um servidor utilizando a versão 22.04 (Jammy) do Ubuntu. Após a criação da máquina virtual que hospeda o servidor, foram realizadas diversas configurações essenciais, incluindo a instalação do Python 3 e do Nginx.

### 3.5.1 Configuração de Usuário e Grupos

O usuário lynx foi criado no servidor e configurado com permissões específicas para atender às necessidades do software Lynx. Para isso, ele foi adicionado aos grupos root, adm, www-data e lxd, garantindo acesso administrativo e capacidade de interação com os

serviços necessários ao monitoramento. Essa configuração assegura que o usuário tenha os privilégios adequados para executar tarefas críticas com segurança.

Ao integrar o usuário lynx a esses grupos, foi possível habilitar funcionalidades essenciais, como a conexão ao servidor via SSH e a execução de comandos relacionados ao gerenciamento de containers (LXC). Além disso, a inclusão no grupo www-data permite a edição de arquivos do Nginx, que, por padrão, pertencem a esse grupo, assegurando flexibilidade e controle na configuração do servidor web.

### 3.5.2 Permissões com *Visudo*

Utilizando a ferramenta visudo, foi configurado que determinados comandos, que normalmente requerem a autenticação do usuário root, possam ser executados sem a necessidade de inserir a senha. Os comandos configurados foram:

```
lynx ALL=(ALL) NOPASSWD: /bin/systemctl restart nginx
lynx ALL=(ALL) NOPASSWD: /bin/systemctl reload nginx
```

Essa configuração simplifica as operações de reinicialização ou recarregamento do Nginx durante o monitoramento, eliminando a necessidade de autenticação repetitiva. No entanto, essa alteração foi restrita a esses comandos específicos. Para qualquer outra operação que exija privilégios administrativos, o fornecimento da senha do usuário root permanece obrigatório, garantindo maior segurança no sistema.

### 3.5.3 Autenticação SSH

Para permitir a autenticação SSH sem o uso de senhas em texto plano no código, foi gerado um par de chaves RSA na máquina onde o *software* Lynx está instalado. A chave pública foi adicionada ao arquivo *authorized_keys* do usuário lynx, localizado em `/home/lynx/.ssh/authorized_keys`

### 3.5.4 Configuração dos Contêineres de Aplicação

Para fins de teste, cada contêiner de aplicação é configurado com limites específicos de recursos, incluindo o número de núcleos de CPU que podem ser utilizados, o tempo máximo de processamento permitido, além de restrições de memória RAM e espaço em disco. Essas configurações garantem o uso eficiente dos recursos disponíveis e simulam um ambiente realista para o funcionamento das aplicações.

A Figura 8 apresenta o arquivo de configuração containers.yaml, utilizado para criar e configurar os contêineres de aplicação. Nesse arquivo, são definidos os parâmetros necessários, como as especificações de hardware e as restrições que serão aplicadas a cada contêiner

Figura 8 – Arquivo containers.yaml

```yaml
config:
  limits.cpu: "1"
  limits.cpu.allowance: 50%
  limits.memory: 512MB
  user.network_mode: "bridged"
devices:
  eth0:
    nictype: bridged
    parent: lxdbr0
    type: nic
  root:
    path: /
    pool: default
    type: disk
    size: 2GB
```

Fonte: Autor 2024.

### 3.5.5 Desenvolvimento dos Scripts de Coleta de Dados

O servidor monitorado necessita de um daemon dedicado para enviar os dados ao Lynx. Esse daemon executa uma série de scripts desenvolvidos em Python 3, cuja função principal é coletar diversas métricas e informações do ambiente e transmiti-las ao Lynx.

Os *scripts* estão localizados no diretório `/home/lynx/scripts/`, sendo eles: **daemon_lynx.py**, **lxc_list_image.py** e **metrics.py**. Cada um desses *scripts* desempenha uma função específica no processo de coleta e envio de dados, conforme detalhado a seguir:

- **Daemon_lynx.py:** Opera em um *loop* contínuo, utilizando um bloco `while True` para executar periodicamente as funções dos *scripts* **lxc_list_image.py** e **metrics.py**. Após cada execução e envio dos dados coletados, o script aguarda 10 segundos utilizando a função `time.sleep(10)`, garantindo intervalos regulares entre as iterações.
- **Lxc_list_image.py:** É responsável por coletar e enviar informações detalhadas sobre os contêineres e as imagens disponíveis no LXC. Ele utiliza o comando Linux

`lxc list -format csv -columns n,s,4,6` para obter dados como nome, status, endereços IPv4 e IPv6 de cada contêiner, estruturando essas informações em dicionários armazenados em uma lista. De forma semelhante, também coleta dados das imagens do LXC, incluindo nome, descrição, arquitetura, tamanho e data de upload, organizando-os em formato CSV. Após reunir essas informações, o *script* realiza uma requisição HTTP POST para enviá-las ao software Lynx, utilizando um *token* de autenticação gerado pela biblioteca *secrets* para garantir a segurança na transmissão.

- **Metrics.py:** É projetado para coletar e enviar métricas detalhadas sobre os contêineres e o servidor *host*, que atua como balanceador de carga. Entre os dados coletados estão: número de contêineres ativos, uso de CPU, memória RAM e disco por contêiner, tempo de *uptime*, quantidade de processos e conexões ativas por contêiner, além do total de requisições, tanto de cada contêiner quanto do servidor *host*. O envio dessas informações para o Lynx segue um processo similar ao utilizado no *script* **lxc_list_image.py**, diferenciando-se apenas pela URL específica da API para onde as métricas são direcionadas.

Todos os *scripts* foram desenvolvidos no ambiente do Lynx, utilizando a IDE PyCharm para facilitar o processo de codificação e teste. Após sua conclusão, os *scripts* foram transferidos para o servidor monitorado por meio do *Secure Copy Protocol* (SCP).

### 3.5.6 Configuração do Daemon lynx.service

Após o desenvolvimento dos *scripts* de coleta de dados, foi criado o serviço lynx.service, localizado em `/etc/systemd/system`. Esse serviço tem como objetivo executar automaticamente os *scripts* responsáveis pela coleta e envio de métricas.

Além disso, o serviço foi configurado para redirecionar os logs de saída padrão para `/var/log/lynx.log` e os *logs* de erro para `/var/log/lynx_error.log`. Essa configuração centraliza o gerenciamento de *logs*, facilitando a análise de erros e o monitoramento do funcionamento do serviço. Por fim, ao ativar o lynx.service, os *scripts* são executados automaticamente toda vez que o servidor é iniciado, garantindo operação contínua e automatizada.

## 3.6 DESENVOLVIMENTO DO *BACK-END*

Como mencionado anteriormente, o Lynx foi desenvolvido utilizando o *framework* Django, com sua estrutura de *back-end* organizada principalmente nas *views* de cada

aplicativo. Entre os componentes do sistema, destaca-se o aplicativo **api**, que desempenha um papel essencial na operação do *software*. Esse aplicativo é responsável pela lógica de recebimento e processamento dos dados enviados pelo servidor monitorado, além de garantir seu armazenamento eficiente no banco de dados. O *app* **api** também centraliza a definição dos *models*, que estruturam e organizam todas as informações manipuladas pelo projeto.

Além do *app* **api**, o Lynx é composto por outros cinco aplicativos: **containers**, **contas**, **dashboard**, **images** e **nginx**. A seguir, são detalhadas as funções de cada um:

- **Api:** Responsável por receber, processar e armazenar os dados coletados do servidor monitorado no banco de dados. Esse aplicativo também contém todas as definições de *models* que estruturam o sistema.
- **Contas:** Gerencia as funcionalidades de autenticação, incluindo *login*, *logout* e criação de contas de usuários.
- **Dashboard:** Realiza a consulta de métricas relacionadas ao uso de recursos de cada contêiner, como CPU, RAM e disco, utilizando o ORM para acessar o banco de dados. Os dados são enviados ao *front-end*, onde são apresentados ao usuário de forma visual.
- **Containers:** Coleta informações sobre os contêineres, como status (ativos ou inativos) e endereços IP, do banco de dados via ORM e envia ao *front-end*. Esse aplicativo também gerencia ações do usuário, como iniciar, parar ou reiniciar contêineres, alterar o endereço IPv4 ou ativar/desativar o IPv6.
- **Images:** Trata informações sobre imagens disponíveis, como nome, descrição e tamanho, acessando o banco de dados via ORM e enviando os dados ao *front-end*. Também gerencia solicitações do usuário para excluir imagens.
- **Nginx:** Centraliza a automação relacionada ao servidor monitorado. Este aplicativo gerencia a atualização do arquivo de configuração de contêineres *upstream*, decide o momento de iniciar ou desligar contêineres de aplicação e automatiza outras tarefas essenciais para a operação do balanceador de carga.

A Figura 9 ilustra a função **check_cpu_usage**, localizada no arquivo *views.py* do aplicativo nginx. Essa função recebe como parâmetro uma lista contendo os valores de porcentagem de uso de CPU de todos os contêineres ativos. Em seguida, ela calcula a média aritmética desses valores e, com base no resultado, decide se é necessário ativar um novo contêiner para garantir a continuidade e a eficiência dos serviços.

Figura 9 – Função check_cpu_usage

```
2 usages  jbgonca
def check_cpu_usage(cpu_usages):
    # Obter todos os valores de CPU usage que chegaram a view do apps.api, passando por parâmetro
    if cpu_usages:
        # Calcular a média aritmética. Cada container só tem um cpu usage, portanto o número de items cpu_usage
        # é o número dos containers ativos
        total_cpu_usage = sum(cpu_usages)
        num_containers = len(cpu_usages)
        avg_cpu_usage = total_cpu_usage / num_containers

        # Verifica se a média de uso de CPU é igual ou maior que 70%
        if avg_cpu_usage >= 70:
            active_containers()
            containers_running = ContainerLxcList.objects.filter(status='RUNNING')
            update_upstream(containers_running)

        # Se a média for menor, para os containers de aplicação, limitando-se no mínimo 1
        elif avg_cpu_usage <= 20 and num_containers > 1:
            stop_containers()
            containers_running = ContainerLxcList.objects.filter(status='RUNNING')
            update_upstream(containers_running)
```

Fonte: Autor 2024.

## 3.7 DESENVOLVIMENTO DO *FRONT-END*

O desenvolvimento do *front-end* concentrou-se nos arquivos localizados nos diretórios `templates` e `static`. Entre os seis aplicativos implementados no *back-end*, quatro requereram a criação de uma interface dedicada no *front-end*: **contas**, **dashboard**, **containers** e **images**. A navegação entre essas interfaces é facilitada por uma barra lateral (*sidebar*).

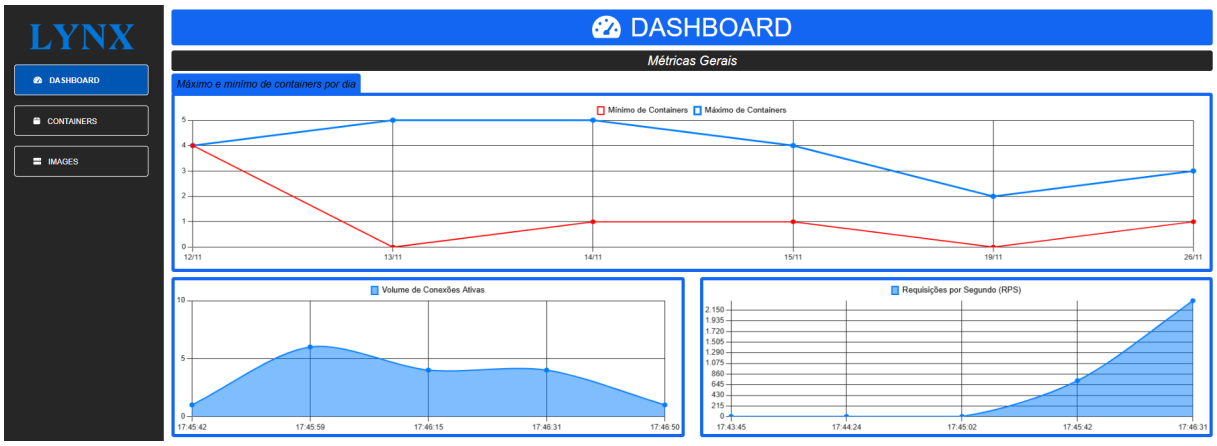
No aplicativo contas, foram desenvolvidos dois arquivos HTML: um destinado à tela de login, permitindo que o usuário acesse o sistema, e outro para a criação de novas contas, caso o usuário ainda não possua uma. O formulário de login foi baseado no padrão fornecido pelo Django, o que simplificou sua implementação. A Figura 10 ilustra a tela de login do Lynx, destacando sua interface limpa.

Figura 10 – Tela de Login


Fonte: Autor 2024.

No aplicativo dashboard, foi criado um arquivo HTML utilizando os componentes do MDBootstrap, juntamente com a biblioteca Chart.js, amplamente empregada para a criação de gráficos interativos. Também foi utilizada a biblioteca Font Awesome para a inclusão de ícones versáteis e de alta qualidade. A Figura 11 apresenta o *front-end* de dashboard, onde são exibidas as métricas de desempenho dos containers por meio de gráficos interativos criados com a biblioteca Chart.js.

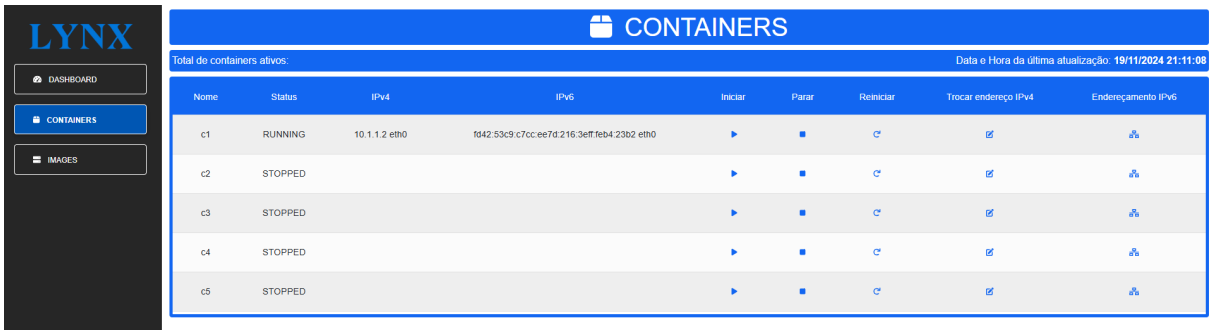
Figura 11 – Interface do aplicativo dashboard




Fonte: Autor 2024.

No aplicativo containers, foram utilizadas as mesmas bibliotecas aplicadas no dashboard, com exceção da Chart.js, já que não há necessidade de gráficos nesta seção. A interface permite que todas as interações do usuário sejam realizadas de forma dinâmica, graças à utilização do *JavaScript*, que manipula os elementos HTML e realiza requisições HTTP utilizando a API fetch. A Figura 12 apresenta a interface do aplicativo containers, exibindo a listagem de todos os contêineres, juntamente com informações detalhadas e botões para interagir com cada um deles.

Figura 12 – Interface do aplicativo containers




Fonte: Autor 2024.

Por sua vez, o aplicativo images apresenta uma interface e funcionalidades bastante semelhantes às do aplicativo contêineres. Ambas compartilham o mesmo conjunto

de bibliotecas e design, diferindo apenas em alguns detalhes, como a quantidade de interações disponíveis para o usuário. A Figura 13 ilustra a interface do aplicativo images, que permite a exibição e a gestão das imagens armazenadas no sistema.

Figura 13 – Interface do aplicativo images


Fonte: Autor 2024.

Vale destacar que, em todos os arquivos HTML, foram incorporados os elementos do MDBootstrap, incluindo sua estilização e componentes baseados em *JavaScript*. Essa integração simplificou significativamente o processo de estilização dos *templates*.

# 4 RESULTADOS

Para avaliar as principais funcionalidades do Lynx, como a automação da instanciação de contêineres de aplicação e a atualização dinâmica do arquivo responsável por gerenciar os contêineres *upstream*, foi utilizado o *software Siege*, na própria máquina que é o servidor monitorado, conforme descrito na Seção 3.4. O cenário do primeiro teste consistiu em simular 350 clientes simultâneos enviando requisições ao servidor monitorado durante um período de 5 minutos. O comando utilizado para realizar o teste foi:

```
siege -c350 -t5M http://192.168.77.2
```

Os resultados obtidos estão detalhados na Figura 14. Durante a simulação, o sistema processou 442.794 transações, com uma taxa de disponibilidade de 99,99% e apenas 46 falhas registradas. O tempo médio de resposta foi de 0,23 segundos, e a taxa de transações alcançou 1.475,68 transações por segundo, com um *throughput* de 0,90 MB/s. A concorrência média registrada foi de 346,63 conexões simultâneas.

Os dados indicam que, mesmo com a necessidade de iniciar e parar vários contêineres durante o teste, o tempo de resposta permaneceu baixo e o impacto negativo no desempenho geral foi mínimo. Isso demonstra a eficácia do gerenciamento de instância dinâmico, garantindo estabilidade mesmo sob alta demanda.

Figura 14 – Desempenho com 350 clientes simultâneos


```
"transactions":                 442794,
"availability":                  99.99,
"elapsed_time":                 300.06,
"data_transferred":             271.11,
"response_time":                  0.23,
"transaction_rate":            1475.68,
"throughput":                     0.90,
"concurrency":                  346.63,
"successful_transactions":      442794,
"failed_transactions":              46,
"longest_transaction":            4.73,
"shortest_transaction":           0.00
```


Fonte: Autor 2024.

O segundo teste teve como objetivo avaliar o desempenho do servidor monitorado em um cenário de carga moderada, simulando 200 clientes simultâneos enviando requisições durante um período de 3 minutos. O teste foi realizado em duas etapas: a primeira

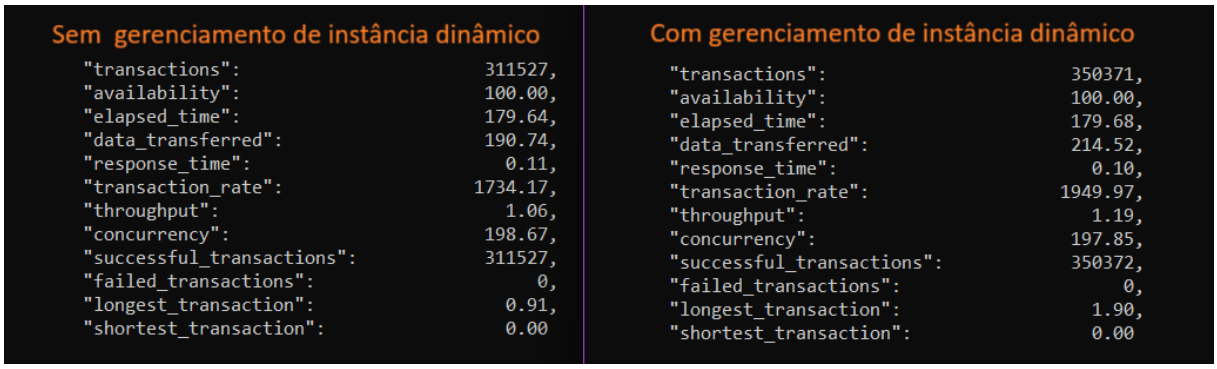
com o gerenciamento dinâmico de instâncias desativado e, em seguida, com ele ativado. O comando utilizado para executar a simulação foi:

```
siege -c200 -t3M http://192.168.77.2
```

Apesar disso, a latência da transação mais longa foi ligeiramente superior no ambiente com o gerenciamento ativo. Isso pode ser explicado pelo fato de que, durante algumas requisições, contêineres adicionais podem ter sido iniciados ou o arquivo de contêineres *upstream* pode ter sido atualizado, o que impacta pontualmente o tempo de resposta.

A Figura 15 apresenta os resultados detalhados do teste de carga para ambos os cenários, evidenciando o desempenho aprimorado proporcionado pelo gerenciamento dinâmico de instâncias.

Figura 15 – Servidor com e sem gerenciamento dinâmico




Fonte: Autor 2024.

Durante a execução dos testes, foi possível observar na interface web desenvolvida o aumento no número de contêineres em tempo real, conforme a média aritmética de uso da CPU dos contêineres alcançava ou ultrapassava o limite de 70%, conforme definido no *back-end*. Esse comportamento demonstra a capacidade do sistema de escalar, ajustando os recursos disponíveis de acordo com a demanda.

A Figura 16 ilustra três contêineres adicionais que foram automaticamente iniciados para atender às requisições do primeiro teste, evidenciando o funcionamento eficaz da lógica de escalabilidade implementada.

Figura 16 – Contêineres iniciados por escalabilidade dinâmica


Fonte: Autor 2024.

# 5 CONCLUSÃO

O desenvolvimento do *software* Lynx permitiu demonstrar a eficácia e os benefícios de um sistema de gerenciamento dinâmico de instâncias para servidores web. Além disso, evidenciou a relevância de monitorar métricas diversas do ambiente monitorado e gerenciar fatores críticos por meio de uma interface web prática e funcional.

Os testes realizados com o *software* Siege confirmaram a capacidade do sistema de aprimorar significativamente o desempenho do servidor, reduzindo tempos de resposta e aumentando a taxa de transações processadas, mesmo sob condições de alta carga. A escalabilidade dinâmica, por sua vez, destacou-se como um diferencial, permitindo que novos contêineres fossem iniciados automaticamente sempre que necessário, com impacto mínimo no desempenho do ambiente.

Este projeto destacou a importância de integrar tecnologias avançadas para desenvolver soluções robustas e flexíveis, capazes de atender às demandas modernas de aplicações web. A interface web criada demonstrou ser uma ferramenta essencial para o monitoramento e gestão eficiente do sistema, tornando o processo mais acessível e prático para os administradores.

De maneira geral, o trabalho realizado evidencia o potencial de ferramentas de monitoramento, gerenciamento e automação na administração de servidores baseados em contêineres LXC, oferecendo soluções escaláveis e eficazes para ambientes tecnológicos dinâmicos.

## 5.1 TRABALHOS FUTUROS

Existem diversas possibilidades para trabalhos futuros que podem ampliar e aprimorar o projeto desenvolvido, oferecendo novas funcionalidades e tornando o sistema ainda mais robusto e eficiente.

Uma das principais propostas seria o aprimoramento da interface web, permitindo configurações avançadas, como a seleção do tipo de balanceamento de carga utilizado pelo Nginx diretamente pela interface. Além disso, seria possível incluir recursos para gerenciar o LXD, como configurar contêineres, realizar a exclusão de contêineres obsoletos e baixar imagens diretamente pela interface, tornando o sistema mais intuitivo e funcional.

Outra proposta significativa seria a otimização do processo de coleta de métricas por meio do Simple Network Management Protocol (SNMP), criando MIBs personalizadas específicas para as métricas dos contêineres. Essa abordagem aumentaria a eficiência do Lynx na coleta e análise de dados, simplificando sua integração com diferentes ambientes de rede e reduzindo a sobrecarga de processamento.

Além disso, um avanço estratégico seria automatizar o gerenciamento de máquinas físicas, permitindo o controle remoto do acionamento e desligamento dos servidores. Essa funcionalidade poderia ser implementada utilizando o recurso Wake-on-LAN (WoL), possibilitando que servidores físicos *upstream* sejam desligados quando não estão em uso, economizando energia e recursos computacionais, especialmente em ambientes com grande escala de operação.

Por fim, um passo fundamental seria a validação do Lynx em um ambiente de produção real. A realização de estudos de caso em condições práticas de uso permitiria identificar ajustes necessários, avaliar o desempenho do sistema e confirmar sua viabilidade operacional. Contudo, essa implementação exige um planejamento rigoroso para mitigar possíveis impactos negativos e garantir o funcionamento estável do ambiente produtivo.

## REFERÊNCIAS


ABBAS, R.; SULTAN, Z.; BHATTI, S. N. Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege. In: IEEE. **2017 international conference on communication technologies (comtech)**. [S.l.]: IEEE explore, 2017. p. 122–131.

ALEKSIC, M. **What's LXC?** What are Linux containers?, 2022. Acesso em 23 jan. 2024. Disponível em: <https://ubuntu.com/blog/what-are-linux-containers>.

BALBINO, M. **Construção e gerenciamento de sistemas de monitoramento focados ao ambiente cloud**. 2021. Monografia (Trabalho de Conclusão de Curso) — Curso Superior de Tecnologia em Sistemas de Computação, Universidade Federal Fluminense, 2021.

BARTENEV, V. **Thread Pools in NGINX Boost Performance 9x!** Nginx part F5, 2015. Acesso em 08 abr. 2024. Disponível em: <https://www.nginx.com/blog/thread-pools-boost-performance-9x/>.

BATISTA, J. **Gerenciamento eficiente de infraestrutura Docker com Portainer.io**. blog 4Linux, 2022. Acesso em 10 jan. 2024. Disponível em: <https://blog.4linux.com.br-/portainer-interface-grafica-para-seu-docker/>.

CARISSIMI, A. Virtualização: da teoria a soluções. In: **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuidos**. Rio de Janeiro, RJ, Brasil: SBRC, 2008. p. 173–207.

CASALICCHIO, E.; IANNUCCI, S. The state-of-the-art in container technologies: Application, orchestration and security. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 32, n. 17, p. e5668, 2020.

COSTA, G. M. F.; SANTOS, C. E. D. Utilização de containers: Um contexto histórico. **Revista Científica e-Locução**, v. 1, n. 20, p. 23–23, 2021.

DJANGO. **Why Django?** Django Software Foundation, 2024. Acesso em 12 nov. 2024. Disponível em: <https://www.djangoproject.com/start/overview/>.

DOCKER. **Use containers to Build, Share and Run your applications**. Docker, 2023. Acesso em 20 jan. 2024. Disponível em: <https://www.docker.com/resources/what-container/>.

FLANAGAN, D. **JavaScript: o guia definitivo**. [S.l.]: Bookman Editora, 2012. 1076 p.

FULMER, J. **siege - An HTTP/HTTPS stress tester**. die.net, 2004. Acesso em 02 fev. 2024. Disponível em: <https://github.com/lxc/python3-lxc>.

GOMES, R. **Docker para desenvolvedores**. Salvador: Leanpub, 2019. 180 p.

GRABER, S. **Python 3.x binding for liblxc**. Github, 2023. Acesso em 30 jan. 2024. Disponível em: <https://github.com/lxc/python3-lxc>.

LINUXCONTAINERS.ORG. **What's LXC?** Linuxcontainers.org, 2023. Acesso em 21 jan. 2024. Disponível em: <https://linuxcontainers.org/lxc/introduction/>.

LONGEN, A. **Nginx vs Apache: Qual Servidor Web Usar na Minha VPS?** Hostinger Tutoriais, 2023. Acesso em 18 jan. 2024. Disponível em: <https://www.hostinger.com.br-/tutoriais/nginx-vs-apache#Apache_vs_Nginx_%E2%80%93_Qual_Devo_Usar.>

MA, C.; CHI, Y. Evaluation test and improvement of load balancing algorithms of nginx. **Ieee Access**, IEEE, v. 10, p. 14311–14324, 2022.

MDBOOTSTRAP. **Top quality open-source UI Kits**. MDBootstrap, 2024. Acesso em 12 nov. 2024. Disponível em: <https://mdbootstrap.com/>.

MDN. **CSS**. mdn web docs, 2022. Acesso em 18 nov. 2024. Disponível em: <https:-/-/developer.mozilla.org/pt-BR/docs/Web/CSS>.

____. **Elementos HTML**. mdn web docs, 2023. Acesso em 15 nov. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element>.

____. **O que é um servidor web (web server)?** mdn web docs, 2023. Acesso em 15 jan. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn-/Common_questions/Web_mechanics/What_is_a_web_server>.

MENDES, A.; DUARTE, A. Comparativo entre docker e lxc/lxd para a virtualização. In: **Anais da XIX Escola Regional de Computação Bahia, Alagoas e Sergipe**. Porto Alegre, RS, Brasil: SBC, 2019. p. 295–303.

MILETTO, E. M.; BERTAGNOLLI, S. de C. **Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP-Eixo: Informação e Comunicação-Série Tekne**. Rio Grande do Sul: Bookman Editora, 2014. 276 p.

MONTEIRO, L. **Python: características, noções e guia de estudo**. Universidade da Tecnologia, 2023. Acesso em 30 jan. 2024. Disponível em: <https://universidadedatecnologia-.com.br/estudo-linguagem-python-2018/>.

NETO, J. X. da S. **Gerenciamento e monitoração de redes de computadores com ênfase em disponibilidade de servidor web com ferramenta zabbix**. 2021. Monografia (Trabalho de Conclusão de Curso) — Curso de Graduação em Engenharia da Computação, Pontifícia Universidade Católica de Goiás, 2021.

OSMAN, M. **NGINX vs. Apache  Choosing the best web server in 2024**. Nexcess, 2023. Acesso em 18 jan. 2024. Disponível em: <https://www.nexcess.net/blog/nginx-vs-apache-/#:~:text=NGINX%20may%20be%20your%20best,for%20high%20traffic%20and%20scalability-.>

REESE, W. Nginx: the high-performance web server and reverse proxy. **Linux Journal**, Belltown Media Houston, TX, v. 2008, n. 173, p. 2, 2008.

SILVA, L. P. da; SANTOS, L. C. dos. Uso da tecnologia de virtualização para melhor aproveitamento de recursos de hardware. **FaSCi-Tech**, v. 1, n. 2, 2016.

SOUZA, J. et al. Rufus: Ferramenta para o gerenciamento de infraestrutura para a execuç ao de aplicaç oes em containers. In: SBRC. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuidos**. [S.l.]: SBRC, 2016. p. 39–44.

SQLITE. **What Is SQLite?** SQlite org, 2024. Acesso em 10 nov. 2024. Disponível em: <https://www.sqlite.org/>.

WEBER, E. **Web Server: O que é e Como Funciona?** Hostinger Tutoriais, 2023. Acesso em 15 jan. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/web-server>.