Capítulo

1

Estudo de caso comparativo entre as configurações Sendfile e Gzip no servidor *web* Nginx 1.24 em um ambiente Debian 12 sob a perspectiva de requisições por segundo, consumo de CPU e consumo de memória

Gabriel Godinho dos Santos, Mateus Mannes de Medeiros, Patrick Placido Bez Fontana, Yasmin de Oliveira Theobaldo Fadel

Resumo

Este tutorial compara duas configurações do servidor web Nginx na versão 1.24 no sistema operacional Debian 12. A primeira configura o Sendfile, permitindo a transferência direta de dados entre descritores de arquivo. A segunda configura o Gzip para comprimir os arquivos antes de servi-los. Para comparação são utilizadas duas instâncias do serviço de nuvem Amazon EC2 com Debian 12 instalado. Uma instância serve uma página web estática simulando um blog de tecnologia, enquanto a outra realiza requisições através do ApacheBench, medindo requisições por segundo. Além disso, o consumo de CPU e memória são monitorados com o Pidstat em quatro cenários: (i) sem Gzip e Sendfile ativados; (ii) com Gzip ativado e Sendfile desativado; (iii) com Gzip desativado e Sendfile ativado; e (iv) com Gzip para arquivos de texto e Sendfile para arquivos de imagem.

1.1 Introdução aos Servidores *Web*

O servidor *web* trata-se de um programa operando em um servidor que pode processar pedidos provenientes de computadores cliente [Moser 2014]. A partir disso, este é um conjunto de três fatores essenciais: plataforma, software e informação; ou seja, necessita de uma base, uma forma de desenvolver-se e, principalmente, um objetivo para servir. Dessa forma, seu objetivo é funcionar como uma ponte que provê a sustentação entre o conjunto de documentos e a plataforma computacional.

Conceitualmente, os servidores *web* podem ser acessados por meio do *Hypertext Transfer Protocol* (HTTP), este que é o fundamento de toda comunicação de dados na

internet e um protocolo de cliente-servidor, no qual as requisições são instigadas pelo receptor, frequentemente um navegador *web*. [MDN - *Overview of HTTP* 2023].

A partir de um panorama abstrato, o funcionamento do servidor *web* é apresentado na Figura 1: Inicialmente, o cliente envia uma requisição ao servidor usando o protocolo HTTP. Esta requisição é processada pelo servidor *web*. Depois que o servidor executa o serviço solicitado, ele envia uma resposta de volta ao cliente, permitindo assim que o cliente realize operações conforme as informações são trocadas entre os dois.

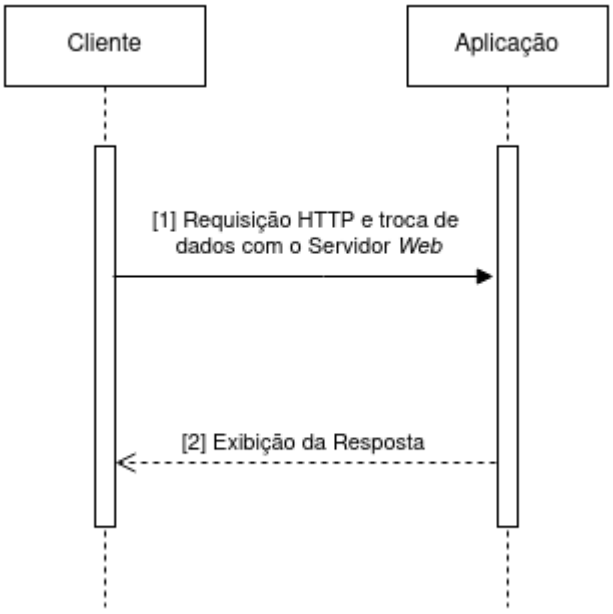

Figura 1. Funcionamento dos Servidores *Web*. *Criado pelos autores.*

Na Figura 1 a interação entre o cliente e uma aplicação é delineada por dois retângulos representando as entidades envolvidas. A seta que vai do cliente para a aplicação simboliza a requisição HTTP e a subsequente troca de dados com o servidor *web*. A seta pontilhada retornando à aplicação indica a resposta do servidor, demonstrando que as operações disponibilizadas pelo servidor foram acessadas e que os dados correspondentes são transmitidos de volta ao cliente.

Logo, percebe-se que os servidores *web* revelam-se como alicerces da comunicação na internet, facilitando a interação entre plataformas e clientes. Esses servidores atuam como intermediários vitais, operando por meio do HTTP, que serve como base para a troca de dados na *web*. A representação visual do processo destaca a dinâmica intrínseca entre as operações na rede e a interação do cliente com o serviço, enfatizando a essência do papel central desempenhado pelos servidores *web* na era digital.

1.2 Histórico dos Servidores *Web*

A história dos servidores *web* remonta ao final da década de 1980, quando Tim Berners-Lee propôs o conceito inicial da *World Wide Web* em 1989, seguido por uma proposta mais formal em 1990, em parceria com o engenheiro de sistemas belga Robert Cailliau.

O documento delineou os conceitos essenciais por trás da *Web* e introduziu termos fundamentais, incluindo o projeto de hipertexto denominado "*WorldWideWeb*", que permitiria a visualização de documentos de hipertexto por meio de "navegadores". Em 1990, Berners-Lee implementou o primeiro servidor e navegador da *Web* no CERN, desenvolvendo o código para o servidor em um computador NeXT [CERN 2023].

Após isso, conforme Apache (2014), em fevereiro de 1995, surgiu o *software* de servidor da *web* de Rob McCool, no Centro Nacional de Aplicações de Supercomputação da Universidade de Illinois, em Urbana-Champaign. No entanto, o progresso desse servidor foi interrompido quando Rob deixou a equipe NCSA em meados de 1994. Dessa forma, *webmasters* começaram a elaborar suas próprias extensões e a corrigir os *bugs* existentes, o que criou a necessidade de uma distribuição comum.

Dessa forma, um pequeno grupo desses *webmasters*, contactados através de e-mails privados, uniu esforços para coordenar suas modificações. Então, Brian Behlendorf e Cliff Skolnick estabeleceram uma lista de discussão, ou seja, um espaço compartilhado de informações e acesso para os principais desenvolvedores, em uma máquina na região da baía da Califórnia com largura de banda fornecida pela HotWired. Partindo da base do servidor NCSA 1.3, todas as correções e melhorias foram implementadas, testadas em seus próprios servidores e, em abril de 1995, foi feito o primeiro lançamento público oficial do servidor Apache, versão 0.6.2. Após testes beta, adaptações para diversas plataformas, uma nova documentação e a adição de recursos na forma de módulos padrão, o Apache 1.0 foi oficialmente lançado em 1º de dezembro de 1995. Menos de um ano após a formação do grupo, o servidor Apache ultrapassou o NCSA como o servidor número um da internet — posição ocupada por Cloudflare em 2023 [Netcraft – July 2023].

Em seguida, em 1999, os membros do Grupo Apache fundaram a Apache Software Foundation, com o objetivo de oferecer suporte organizacional, jurídico e financeiro para o Apache HTTP *Server*. Essa fundação solidificou as bases do *software* para o seu desenvolvimento futuro e ampliou consideravelmente o número de projetos de *software* de código aberto que estão sob a égide da Fundação.

Todavia, o Apache ganhou um concorrente como o servidor *web* mais utilizado na Internet, a partir do surgimento do Nginx. Segundo Nginx (2023), o servidor foi desenvolvido por Igor Sysoev, para lidar com o desafio das conexões simultâneas em massa (problema C10K — incapacidade de um servidor *web* escalar além de 10.000 conexões simultâneas), trazendo uma abordagem assíncrona e orientada a eventos. Sua arquitetura eficiente e capacidade de lidar com um alto número de requisições o tornaram uma escolha popular entre os administradores de sistemas em busca de desempenho superior.

Posteriormente à abertura do código do projeto em 2004 e o crescimento exponencial de sua popularidade, Sysoev fundou a Nginx, Inc. — para impulsionar o contínuo desenvolvimento do servidor — e o Nginx Plus — para oferecer uma versão comercial —, com recursos adicionais projetados para atender às demandas de clientes corporativos. Depois disso, houve a incorporação da Nginx, Inc. pela F5, Inc. em 2019, o que solidificou sua posição no mercado [Nginx 2023].

Portanto, tanto o pioneirismo do Apache quanto a ascensão do Nginx destacam a importância de servidores *web* robustos e adaptáveis para o cenário em constante evolução da Internet. Enquanto o Apache solidificou sua posição como líder de mercado,

o Nginx trouxe inovações arrojadas, estabelecendo um novo padrão de desempenho e eficiência. A competitividade entre essas duas plataformas, impulsionada por melhorias contínuas e adaptações às demandas dos usuários, exemplifica a natureza dinâmica e competitiva do ambiente de tecnologia da informação. A interação entre essas duas soluções, juntamente com outras ferramentas de *software* de código aberto, ilustra a relevância da busca por inovação para impulsionar avanços significativos de desempenho no mundo computacional.

1.3 Requisições e respostas em Servidores *Web*

A infraestrutura da *web* depende crucialmente dos servidores *web*, que desempenham um papel central na entrega de conteúdo e serviços *on-line* [MDN 2023]. Por via da comunicação servidor-cliente através de requisições, estes atuam como a base fundamental para a disponibilidade e entrega de conteúdo na Internet.

A troca de mensagens HTTP entre o servidor e o cliente ocorre a partir de dois fatores: requisições (*requests*) e respostas (*responses*). Esta interação é ilustrada na Figura 2, que mostra o usuário executando uma ação no navegador, como clicar em um link ou digitar um endereço *web*. Em resposta, o navegador envia uma solicitação HTTP ao servidor *web*. O servidor, então, processa essa solicitação e retorna os dados requeridos na forma de uma página *web*. Finalmente, o navegador interpreta esses dados e os exibe ao usuário.

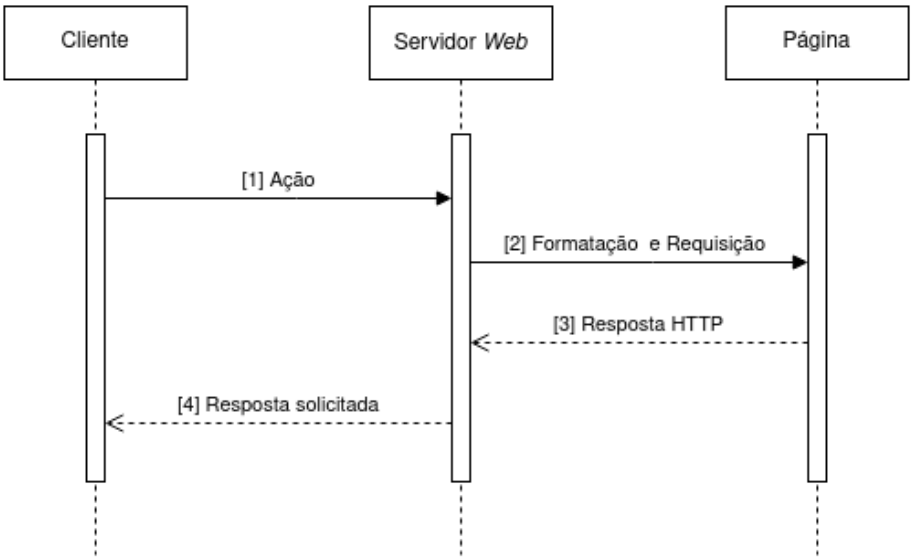

Figura 2. Requisições e respostas em uma página *web*. *Criado pelos autores.*

A Figura 2, é dividida em três partes, respectivamente, o Cliente, o Servidor *Web* e a Página, representados por três retângulos lado a lado. A primeira flecha completa parte do Cliente e vai ao Servidor *Web*, indicando a ação realizada pelo usuário. Posteriormente, a segunda flecha completa segue do Servidor *Web* a Página, representando a formatação e a requisição do servidor *web*. Após encontrar a página, volta

uma flecha pontilhada da Página ao Servidor *Web* e em seguida outra do Servidor para o Cliente, indicando a resposta HTTP.

Assim, os servidores *web* agem na disponibilização desses arquivos essenciais para o funcionamento dos sites. Estes armazenam e entregam documentos HTML (para estruturação do conteúdo), CSS (para estilização), arquivos JavaScript e outros ativos associados, garantindo a disponibilidade contínua desses elementos para os usuários finais [MDN - *Overview of* HTTP 2023].

Em relação ao Nginx, o servidor é reconhecido por sua eficiência e desempenho na entrega de conteúdo *web*. Dentre os componentes essenciais do sistema operacional usados pelo servidor Nginx, destacam-se os sockets, que são cruciais para a comunicação e a transferência de dados entre o servidor e os clientes. De acordo com IBM (2021), estes são gerenciados pelo sistema operacional via descritores de arquivos e podem ser vistos como canais de comunicação que permitem a conexão e a troca de informações entre diferentes computadores.

Além disso, o Nginx depende do sistema de arquivos do sistema operacional para acessar e entregar os arquivos estáticos e dinâmicos solicitados pelos clientes. O gerenciador de processos e *threads* também desempenha um papel fundamental, garantindo o tratamento eficiente e oportuno das solicitações. A memória do sistema é utilizada para armazenar temporariamente os dados durante o processamento das requisições, contribuindo para uma entrega ágil e eficaz do conteúdo [Nginx 2023]. Ademais, segundo IBM (2023), outras ferramentas do sistema operacional podem ser configuradas para serem utilizadas no servidor, como as chamadas de sistemas SendFile — que permitem a transferência de dados de um arquivo para outro, sem que os dados passem pela memória do espaço do usuário — e a ferramenta de compressão Gzip, esta que compacta e descompacta arquivos, a fim de reduzir o tamanho de arquivos e economizar espaço em disco, bem como para acelerar o tempo de transferência de arquivos pela rede.

Em resumo, o servidor Nginx é dependente de uma série de componentes do sistema operacional para executar de forma eficaz as requisições HTTP entre o servidor e o cliente. A interdependência entre esses elementos destaca a importância de uma infraestrutura robusta e eficiente para a entrega de conteúdo *web* de maneira ágil e confiável. Dessa forma, contribui-se para a otimização dos recursos do sistema operacional, essencial para a performance e confiabilidade do servidor Nginx no contexto da crescente demanda por serviços *web* de alta qualidade e velocidade.

1.4 Métricas de análise

Pode-se entender como métrica de análise um critério o qual será utilizado para avaliar e medir o desempenho de um sistema computacional. No caso de servidores *web*, é necessário definir e especificar um conjunto de métricas de análise, visto que apenas uma não é suficiente para a realização de uma avaliação eficiente e adequada [AMARAL 2002].

Diante disso, a seleção de métricas eficientes depende sobretudo de dois fatores: o tipo de sistema que será avaliado e as ferramentas disponíveis, as quais possibilitam a coleta de dados e a execução dos critérios selecionados [AMARAL 2002]. No contexto

de sistemas computacionais compartilhados, como é o caso dos servidores *web*¸ várias métricas podem ser utilizadas para avaliar o seu desempenho. Dentre estas estão o tempo de resposta, consumo de CPU e consumo de memória, disponibilidade, nível de multiprogramação e taxa de requisições. Em meio aos exemplos citados, foram escolhidos três, os quais serão detalhados nas Subseções 1.4.1 e 1.4.2 para fins de avaliação e medição de desempenho de um servidor *web* Nginx, dentro de quatro cenários diferentes.

Por fim, segundo Rodrigues (2009), diante de uma expressiva variedade de ferramentas e *softwares* disponíveis para uso e tendo conhecimento acerca das principais métricas a serem utilizadas, escolher a ferramenta que melhor atenda às necessidades do cenário de acordo com a quantidade e o nível de detalhes de informações que fornecem é essencial. Portanto, conforme as necessidades deste estudo de caso, as ferramentas escolhidas foram o ApacheBench, responsável pela realização das requisições ao servidor Nginx, e o Pidstat, que é responsável por monitorar o consumo de CPU e de memória do servidor.

1.4.1 Taxa de requisições

No âmbito de servidores *web,* a taxa de requisições pode ser entendida como o total de solicitações feitas ao servidor atendidas com sucesso dentro de um intervalo de tempo e, normalmente, as unidades de tempo utilizadas são "segundos" ou "minutos".

O cálculo dessa métrica é dado pelo número de requisições feitas ao servidor dividido pelo intervalo de tempo total, que é calculado a partir do início da primeira requisição feita até o término da última requisição. Isso inclui qualquer intervalo entre as requisições feitas. Para exemplificar, dada uma amostra de 20 requisições, as quais serão feitas para um determinado servidor, o tempo total é obtido observando-se o intervalo de tempo entre a 1ª requisição feita e a 20ª requisição feita e, caso haja um atraso de 2 segundos entre a execução de cada requisição, também deve ser considerado. A fórmula visualizada do cálculo correto da taxa é:

$$Taxa\ de\ requisições = \frac{(quantidade\ de\ requisições)}{(tempo\ total)}$$

Fórmula de cálculo da taxa*. Adaptado de Silva (2015).*

1.4.2 Consumo de CPU e consumo de memória

O desempenho de um servidor web também pode ser influenciado pelo consumo de memória e de CPU. Esses recursos são essenciais para o funcionamento adequado do servidor e seu gerenciamento eficiente é relevante para a capacidade do servidor de atender às várias demandas.

O consumo de CPU refere-se à porcentagem do processador que é utilizada pelo servidor. Uma taxa alta de utilização do processador pode impactar no tempo de resposta e no consumo de energia, resultando na demora em atender as solicitações de usuários e aumento nos custos operacionais. Segundo Castro, Corrêa e Cardoso (2013 apud FAN; WEBER; BARROSO, 2007), mesmo ociosos, os servidores atuais consomem cerca de

70% da energia que é necessária durante um período de pico de trabalho. Diante disso, é necessário que haja um consumo eficiente de energia dos servidores, não apenas durante os picos de atividade, mas também nos períodos de ociosidade. Sendo assim, evitar um alta taxa de utilização da CPU configura-se como uma solução para as questões energéticas.

Agora, para a execução dessa métrica serão observados os picos de consumo do processador feito pelo servidor durante a realização dos testes. Em servidores de banco de dados é comum haver um consumo mais elevado do processador — podendo atingir 100% de utilização da CPU em determinados momentos — por conta dos processos referentes ao Sistema de Gerenciamento de Banco de Dados (SGBD), os quais estão sendo executados. Entretanto, em ambientes com índices baixos de E/S em disco, como é o caso dos servidores *web* e a maior parte dos servidores de aplicação, é esperado que o consumo do processador esteja próximo de zero [RODRIGUES, 2009]. Sendo assim, é esperado que o servidor *web* Nginx desse estudo de caso não apresente picos os quais possam se aproximar de 100% de utilização do processador.

Já o consumo de memória refere-se à quantidade da memória principal utilizada pelo servidor para a execução das tarefas e demandas. Um alto consumo de memória pode impactar a escalabilidade impedindo-o de atender de forma eficiente um alto número de solicitações simultaneamente. Além disso, pode levar à fragmentação da memória. Para a execução dessa métrica são observados, semelhantemente ao consumo de CPU, os picos de utilização da memória principal em *kilobytes*, feita pelo servidor durante a realização dos testes, juntamente com o espaço livre disponível durante os picos apresentados.

1.5 O Debian

Primeiramente, é necessário entender o conceito de *Software* Livre e o que é a *Free Software Foundation* (FSF). Para começar, *Software* Livre refere-se a todo programa de computador que pode ser executado, copiado, modificado e redistribuído sem nenhuma necessidade de autorização [GNU, 2023]. Esse tipo de programa torna acessível a todos, sejam usuários ou desenvolvedores, o seu código-fonte por inteiro. Mais precisamente, um programa pode ser considerado *software* livre quando se enquadra nas quatro liberdades essenciais:

- **Liberdade 0:** liberdade de execução do programa independente da finalidade [GNU, 2023].
- **Liberdade 1:** liberdade de estudar como o programa funciona e adaptá-lo. Para isso, ter acesso ao código fonte é essencial [GNU, 2023].
- **Liberdade 2:** liberdade de distribuir cópias para fins legais [GNU, 2023].
- **Liberdade 3:** liberdade de distribuir cópias modificadas do programa para fins legais. Desta forma, dando uma chance para que a comunidade se beneficie das modificações feitas. Diante disso, o acesso ao código fonte fundamental [GNU, 2023].

Ademais, a criadora e responsável pela difusão desse conceito é a FSF, uma organização sem fins lucrativos fundada por Richard Stallman em 1985, dois anos após a criação do projeto GNU, também de autoria de Stallman [GNU, 2023]. Diante disso, diversos *softwares* surgiram seguindo os princípios do *Software* Livre como o WordPress,

7-ZIP, Audacity, Blender, PDFCreator e uma das primeiras distribuições GNU/Linux criadas, o Debian.

O Debian é um sistema operacional de código aberto mantido e distribuído pelo projeto Debian [Debian, 2023]. Atualmente, o sistema operacional utiliza o núcleo do Linux, mas também possui o Debian GNU/kFreeBSD, uma distribuição que conta com as ferramentas GNU em espaço de usuário e usa a biblioteca GNU C em cima do núcleo do FreeBSD, mas teve seu desenvolvimento oficialmente encerrado em julho de 2023, devido à falta de interesse e de voluntários para mantê-la [Debian GNU/kFreeBSD, 2023]. Ademais, há o Debian GNU/Hurd, que conta com o núcleo GNU Hurd — um conjunto de servidores que rodam sobre o *microkernel* GNU Mach — e está sob desenvolvimento ativo, mas não oferece desempenho e estabilidade [Debian GNU/Hurd, 2023].

O sistema operacional foi idealizado em agosto de 1993 por Ian Murdock, um estudante universitário que liderou o projeto entre 1993 e 1996, antes de transformá-lo em um projeto comunitário, com o objetivo de ser uma nova distribuição GNU/Linux estável baseada totalmente nos princípios do *Software* Livre e desenvolvimento colaborativo [Debian, 2023]. Seguindo esses princípios, o Debian tornou-se conhecido por sua aderência estrita às diretrizes do projeto de *Software* Livre e por sua política de não incluir *software* proprietário em suas distribuições principais. Além disso, o desenvolvimento do Debian é conduzido de forma aberta e transparente, possibilitando que qualquer pessoa possa envolver-se no processo de desenvolvimento e tomar decisões através de votações [Debian, 2023]. Essa abordagem democrática tem sido fundamental para a comunidade Debian, que conta com desenvolvedores em mais de 60 países [Debian, 2023].

Também, o Debian é a base para outras distribuições Linux populares como o Ubuntu, Knoppix, PureOS e Tails e oferece suporte para uma lista longa de arquiteturas de CPU, incluindo AMD64 (ou x64, ou ainda, x86-64), i386, várias versões de AMR e MIPS, POWER7, POWER8, IBM *System* e RISC-V [Debian, 2023]. Além do mais, o Debian pode ser rodado em dispositivos embarcados como o Raspberry PI, variantes do QNAP, dispositivos móveis, roteadores domésticos e computadores de placa única (*Single Board Computers*) tendo potencial para uso na Internet das Coisas (*Internet of Things - IoT*) [Debian, 2023].

Ainda mais, O Debian trouxe consigo o sistema APT (*Advanced Package Tool*), principal gerenciador de pacotes de linha de comando do Debian e seus derivados. Esse gerenciador não só fornece comandos para pesquisa, administração e consulta de informações sobre pacotes, como também acesso de baixo nível a todos os recursos fornecidos pelas bibliotecas *libapt-pkg* e *libapt-inst,* das quais podem depender os gerenciadores de pacotes de alto nível [Debian APT, 2023]. Também, há um sistema de rastreamento de *bugs* (*Bug Tracking System* - BTS) do Debian disponível publicamente a todas as pessoas na Internet, que permite o envio de novos relatórios de *bugs* encontrados, assim atribuídos a um número e salvos em um arquivo até que sejam tratados [Debian, 2023].

Outro ponto de destaque do Debian é a sua utilização em servidores. Porém, antes é necessário entender brevemente as três versões que o Debian possui em manutenção ativa: "*Stable*", "*Testing*" e "*Unstable*". A versão "*Stable*" (estável, em português) contém a última versão estável oficialmente lançada do Debian sendo a atual a versão 12,

ou *bookworm,* lançada como 12.0 em 10 de junho de 2023 [Debian, 2023]. A versão "*Testing*" (teste, em português) contém pacotes que ainda não foram incorporados à versão "*Stable*", mas que estão na fila para serem aceitos, e a atual é a versão *trixie* [Debian 2023]. E, por fim, a versão "*Unstable*" (instável, em português) é a qual o desenvolvimento ativo do Debian ocorre e, como o próprio nome sugere, é instável. A versão "*Unstable*" é sempre chamada *sid*, um acrônimo para "*Still in development*" [Debian 2023].

Diante disso, tem-se o primeiro ponto da eficiência do Debian em servidores, a estabilidade, caso a versão utilizada seja a "*Stable*". Novos pacotes só são integrados a versão estável do Debian após uma bateria exaustiva de testes e a certeza de que não existam *bugs* ou falhas que atrapalhem o funcionamento do sistema, o desenvolvimento de uma aplicação ou serviços. Essa estabilidade fez com que diversas organizações e empresas passassem a usar o Debian internamente, incluindo a Agência Nacional de Vigilância Sanitária – ANVISA, que o utiliza em serviços como *e-mails*, monitoramento, varreduras de segurança, servidores de compartilhamento de arquivos e servidores de aplicação [Debian Users, 2023]; e os departamentos de informática e matemática da Universidade Federal do Paraná, que o utilizam em 3 servidores, em mais de 30 estações de trabalho do departamento de matemática e em 90% dos computadores do departamento de informática [Debian Users, 2023]. Outro ponto que faz o Debian ser o preferido para servidores é a sua segurança; o sistema operacional conta com suporte à segurança para as versões "*Stable*" e, também, fornece atualizações de segurança regulares [Debian, 2023]. Além disso, o sistema fornece ferramentas de segurança com propósitos de proteção através de *firewalls*, detecção de intrusão, verificação de vulnerabilidades, antivírus e redes privadas virtuais (*Virtual Private Network* - VPN) [Debian, 2023].

Por fim, outro ponto que garante a eficiência do Debian em servidores é a sua leveza. Os requisitos mínimos para a instalação do Debian incluem 512 MB de memória RAM (*Random Access Memory*) e 4 GB em disco, mas, os requisitos reais de memória são inferiores aos mencionados pois é possível instalar o sistema com apenas 32 MB de disco [Debian, 2023]. Entretanto, é importante salientar que o espaço ocupado pelo Debian depende do que será instalado e utilizado, principalmente nos servidores.

1.6 Diretiva *SendFile*

Para começar, deve-se abordar o conceito de descritores de arquivo. Quando um arquivo é aberto para leitura ou escrita, ou até mesmo quando um *socket* é aberto para comunicação entre processos, o sistema operacional cria um descritor de arquivo (em inglês, *file descriptor*) para referenciar-se às operações. Conforme apresentado por Kerrisk (2010), um descritor de arquivo é um inteiro pequeno sem sinal criado para se referir as operações de E/S em diferentes tipos de arquivos abertos, incluindo *pipes*, FIFOs, *sockets*, terminais, dispositivos e arquivos comuns. Cada processo tem seu próprio conjunto de descritores de arquivo e sempre existirá ao menos um descritor para cada arquivo aberto no sistema operacional.

Deste modo, por convenção, cada processo sempre possui três descritores de arquivo padrões criados já no início do seu ciclo de vida. Estes, listados na Tabela 1, representam três operações básicas que podem ser executadas sobre um arquivo aberto. São estes: o descritor 0, que representa a entrada de dados padrão, pré-definida como

sendo o teclado. O descritor 1, o qual representa a saída de dados padrão, tendo o terminal como saída pré-definida. Por fim, o descritor 2, o qual representa a saída de erros que aconteceram durante as operações de E/S realizadas sobre o arquivo aberto. Tem, também, o terminal do sistema como saída pré-definida [IF UFRGS, 2002].

Tabela 1. Descritores de arquivo padrões. ***Adaptado de Kerrisk (2010).***

Descritor de arquivo	Operação padrão
0	Entrada de dados (*standard input*)
1	Saída de dados (*standard output*)
2	Erro (*standard error*)

Diante do exposto, pode-se entender a diretiva *sendfile* do servidor *web* Nginx. Por padrão, o próprio Nginx lida com a transmissão de dados de um arquivo para outro copiando os dados em *buffer* na memória principal antes de enviá-los ao arquivo de destino [Nginx, 2023]. Porém, habilitar a diretiva *sendfile* elimina a tarefa de armazenar os dados em *buffer* informando as chamadas de sistemas "*sendfile()*" a realizar a cópia direta dos dados de um descritor de arquivo para outro a nível de núcleo [Nginx, 2023]. Além disso, é possível limitar a quantidade de dados transferidos em uma chamada *sendfile* única utilizando a diretiva *sendfile_max_chunk* [Nginx, 2023]. Por exemplo, ao atribuir o parâmetro "1m" à diretiva *sendfile_max_chunk* o servidor limita a transferência de dados por chamada em 1MB.

Para finalizar, usar a diretiva *tcp_nopush* juntamente com a diretiva *sendfile* faz com que o servidor Nginx instrua o sistema operacional a realizar uma chamada de sistema "*sendfile()*" somente quando o tamanho máximo de dados transmitidos for atingido ou quando algum sinal para transmissão de dados pendentes for mandado [Nginx 2023]. É importante ressaltar que as diretivas *sendfile_max_chunk* e *tcp_nopush* só podem ser usadas caso a diretiva *sendfile* esteja habilitada [Nginx, 2023]. Portanto, ativar a diretiva *sendfile* do servidor Nginx tem como objetivo principal a redução da utilização de memória, visto que não são mais necessários *buffers* para alocar os dados que serão transmitidos entre arquivos. Além disso, evitar a etapa de *buffering* pode trazer como consequência positiva uma redução no tempo de processamento e no consumo de CPU realizado pelo servidor, uma vez que não é mais necessário alocar espaços na memória física e pré-carregar os dados.

1.7 Compressão de dados e a diretiva *gzip*

A compressão de dados pode ser definida como a redução do tamanho das informações a serem transmitidas pela rede. O objetivo da compressão tornou-se codificar dados usando a menor quantidade de símbolos possíveis, no menor número de *bits* possíveis [DE OLIVEIRA JÚNIOR; OYAMADA, 2020] apud [MCANLIS; HAEKY, 2016]. Os algoritmos de compressão são divididos em duas categorias: com perda de dados e sem perda de dados. Os algoritmos com perda de dados admitem uma perda controlada de dados em troca de uma compressão maior e são utilizados em imagens gráficas e voz digitalizada [DE OLIVEIRA JÚNIOR; OYAMADA, 2020]. Diante disso, um arquivo

comprimido utilizando algum algoritmo com perda de dados não será igual ao original após a descompressão. Já os algoritmos sem perda de dados geram um fluxo de dados idênticos ao original após a descompressão; são geralmente utilizados em bancos de dados, arquivos de texto e planilhas [DE OLIVEIRA JÚNIOR; OYAMADA, 2020]. Dentre as duas categorias de algoritmos de compressão, a que mais se adequa aos servidores *web* é a segunda abordagem, sem perda de dados, visto que qualquer dado perdido pode ocasionar erros e perda de informações relevantes. À vista disso, segundo Deutsch (1996), um conjunto de dados comprimidos consiste em uma série de blocos, e cada bloco é comprimido utilizando uma combinação do algoritmo de compressão de dados sem perdas LZ77 e da codificação de Huffman — um método de codificação e compactação que procura utilizar o menor número possível de *bits* para representar caracteres que aparecem múltiplas vezes.

Diante do exposto, em servidores *web* e de aplicação, a compressão diminui o tamanho do conjunto de dados transmitidos pela rede exigindo menos esforço do servidor e aumentando a velocidade de entrega do conteúdo. No caso do Nginx, a compressão é feita antes da resposta ser enviada ao cliente e dados já comprimidos não passam pela compressão novamente [Nginx, 2023]. Para ativar a compressão de dados em um servidor Nginx usa-se a diretiva *gzip* com o parâmetro "*on*" e, por padrão, o Nginx comprime as respostas apenas com o MIME *type* (*Multipurpose Internet Mail Extensions*) "*text/html*" [Nginx, 2023]. Entretanto é possível comprimir respostas com outros tipos utilizando a diretiva "*gzip-types*" e passando como parâmetro a lista de MIME *types* desejados [Nginx, 2023].

Porém, é possível que o cliente ao qual o servidor está servindo não suporte respostas com a compressão da diretiva *gzip* ou que não seja apto a armazenar dados compactados. Para resolver essa questão e suprir tanto os clientes que aceitam tanto os que não aceitam dados comprimidos, o Nginx dispõe da diretiva *gunzip* que descompacta os dados antes de, definitivamente, chegarem ao cliente [Nginx, 2023]. Além do mais, é possível especificar o tamanho mínimo para que a compressão seja realizada utilizando a diretiva *gzip_min_length* e passando como parâmetro o tamanho em bytes [Nginx, 2023]. Para exemplificar, caso a diretiva *gzip_min_length* esteja definida com o parâmetro 1024 (1 *kilobyte*), a compressão só será feita caso o tamanho total dos dados ultrapasse 1024 *bytes*. Caso contrário, nada será feito. Também, o Nginx dispõe de uma diretiva para o envio de arquivos compactados, a *gzip_static* [Nginx, 2023]. Com essa diretiva habilitada, o servidor envia ao cliente uma versão compactada de um arquivo ao invés da versão original [Nginx, 2023]. Entretanto, caso o cliente não tenha suporte ao recebimento de arquivos compactados, o Nginx enviará a versão original sem fazer uso do processo de compactação [Nginx, 2023].

Assim, a diretiva *gzip* pode ocasionar um consumo mais baixo de memória por parte do servidor, visto que dados comprimidos são representados por uma sequência de bits menor que a original e, portanto, ocupam menos espaço. Entretanto, como o processo de compactação dos dados ocorre durante a execução da resposta, podem acontecer sobrecargas consideráveis de processamento, podendo aumentar o consumo de CPU e afetar o desempenho do servidor.

1.8 Nginx

Antes do Nginx surgir como uma solução revolucionária para servidores na Internet, o cenário enfrentava desafios significativos. Nos primórdios da rede, servidores convencionais estavam em uso, e estes baseavam-se em arquiteturas que, rapidamente, mostraram-se inadequadas para atender às demandas crescentes. Conforme observado por Nguyen (2017), à medida que o uso da Internet continuou a crescer, surgiu a necessidade de aprimorar o desempenho dos servidores. No entanto, a atualização do *hardware* dos servidores pode ser tanto custosa em termos financeiros, quanto limitada em termos de recursos disponíveis. Diante desse cenário, foi necessário buscar soluções na otimização da arquitetura de *software* para enfrentar desafios como os gargalos de desempenho e a gestão eficiente dos recursos. O "problema C10K" era a questão predominante na época. Esse problema referia-se à capacidade de um servidor *web* em lidar com mais de 10.000 conexões simultâneas sem sobrecarregar o sistema.

Servidores *web* tradicionais, que alocavam um processo ou *thread* para cada conexão, estavam tornando-se ineficientes para o uso adotado, resultando em um uso excessivo de memória e processamento. Antes do surgimento dessa ferramenta, as arquiteturas de servidores *web* eram construídas em torno de dois modelos principais: *threads* e processos. Esses modelos dependiam de *threads* simultâneas e *call-backs* de processamento, respectivamente. Em ambas as abordagens, as operações de entrada/saída (E/S), como o uso de *sockets* e memória, frequentemente levavam ao bloqueio caracterizado pelo impedimento temporário do avanço da execução do programa. Esse bloqueio ocorria devido ao comportamento dos processos ou *threads*, que demandava a espera por determinados eventos, como a conclusão de operações de leitura ou gravação em disco. Além disso, a mudança de contexto necessária para gerenciar os processos contribuía para mais desperdício de tempo de CPU, agravando a eficiência do sistema [Nguyen 2017]. Algo precisava ser feito para enfrentar esse desafio. Nesse contexto, o Nginx foi concebido. Sua motivação era clara: ser eficaz, otimizado, escalável e consumir menos recursos do sistema. A necessidade de servir conteúdo de forma rápida e corretamente, mesmo sob cargas de tráfego intensas, como visto no problema C10K, impulsionou a motivação para criar uma solução que atendesse a essa demanda crescente.

No primeiro lançamento, a ferramenta tinha como objetivo trabalhar juntamente ao Apache, tratando os conteúdos estáticos a fim de que o conteúdo servido não onerasse com processamento de concorrência e latência dos servidores baseados no Apache. Conforme foi sendo desenvolvido, algumas tecnologias foram sendo implementadas, tais como integração com aplicativos por meio de protocolos FastCGI, uWSGI, SCGI, além de sistemas distribuídos de armazenamento em memória [Alexeev 2012].

Para aprofundar o estudo em como esta ferramenta foi arquitetada de maneira a se comunicar de forma harmoniosa com o sistema operacional (SO), é fundamental entender como o Nginx faz uso das características do SO para otimizar seu desempenho. Este servidor é uma ferramenta multifuncional amplamente reconhecida por seu desempenho e eficiência, sendo um dos servidores *web* mais populares do mundo. Desta forma, desempenha diversas funções em ambientes de hospedagem de *websites* e aplicativos *web* que o tornam uma escolha atrativa em relação aos concorrentes.

Além disso, contém a funcionalidade de *proxy* reverso, encaminhando solicitações dos clientes para servidores de aplicativos de forma rápida e segura, aprimorando a escalabilidade e a segurança de aplicativos *web*. Atua também como um

balanceador de carga, distribuindo solicitações uniformemente entre servidores de aplicativos, otimizando recursos e garantindo alta disponibilidade, o que é valioso para aplicativos críticos e de tráfego com várias conexões simultâneas. Este também oferece armazenamento em cache de conteúdo *web*, aliviando a carga nos servidores de origem e acelerando a entrega de conteúdo aos usuários, reduzindo a necessidade de recarregar conteúdo estático repetidamente. Além do mais, o software atua como um *Firewall* de Aplicativos *Web* (WAF) eficaz, protegendo aplicativos e sites contra ameaças cibernéticas, incluindo ataques DDoS e injeções de SQL, aprimorando a segurança dos sistemas.

Diante das opções acima citadas e por oferecer suporte para a entrega de conteúdo multimídia de maneira eficaz, é uma escolha popular para serviços de streaming de áudio e vídeo, se tornando uma escolha versátil que evoluiu continuamente para atender às crescentes demandas da internet moderna. Conforme encontrado no glossário do Nginx (2023), esta evolução pode ser notada desde seu lançamento original, onde a ferramenta foi desenvolvida e ganhou tecnologias como WebSocket, HTTP/2, gRPC além do suporte a *streaming* de vídeo. Sua escalabilidade e arquitetura otimizada o tornam uma escolha pioneira para empresas de alto tráfego, como Dropbox, Netflix e Zynga.

Para compreender plenamente por que o Nginx se destaca em termos de eficiência e baixo uso de recursos, além das funcionalidades que possui, é fundamental analisar a sua arquitetura. Segundo Wang (2021), a arquitetura do Nginx é assíncrona e orientada a eventos, o que o torna produtivo em situações de alto tráfego. A arquitetura assíncrona e orientada a eventos é uma das principais características que diferenciam o Nginx de outros servidores *web* tradicionais. Este foi projetado para atender a um número de conexões simultâneas maior do que as ferramentas antecessoras de maneira eficiente, tornando-o uma escolha popular para servidores de alto desempenho.

Sendo assim, diferentemente dos servidores tradicionais, que alocam um *thread* ou processo dedicado para cada conexão, lidando com as conexões de forma assíncrona. Isso significa que, mesmo em situações de alto tráfego com milhares de conexões simultâneas, o Nginx não precisa alocar um *thread* ou processo separado para cada uma destas. Isso reduz a sobrecarga e o consumo de memória e consequentemente gerindo melhor os recursos de CPU e memória. Essa eficiência é particularmente valiosa quando o servidor opera com recursos limitados. Em ambientes de servidor com milhares de conexões simultâneas, a alocação excessiva de recursos do sistema para cada conexão pode sobrecarregar o servidor e causar um uso ineficiente de CPU e memória. Porém, o Nginx *Web Server*, com sua arquitetura assíncrona, aborda esse problema de maneira eficaz, garantindo que o servidor possa processar milhares de conexões concorrentes sem degradar o desempenho ou esgotar os recursos do sistema.

Já a orientação a eventos, segundo Nguyen (2017), envolve a capacidade de responder a eventos de maneira eficaz e econômica. Ao invés de alocar um *thread* ou processo dedicado para cada conexão, utiliza-se um *loop* de eventos para monitorar e processar eventos de E/S Isso significa que o servidor responde a eventos, como uma requisição HTTP recebida ou uma resposta pronta para ser enviada, sem a necessidade de criar processos ou *threads* para cada evento. Essa abordagem é eficiente pois reduz a sobrecarga de criação e gerenciamento de *threads* ou processos. Como resultado da combinação de assincronismo e orientação a eventos, o Nginx oferece escalabilidade dinâmica. Conforme mais conexões são estabelecidas, o servidor ajusta o uso de recursos

automaticamente. Isso é crucial para servidores que precisam atender a grandes volumes de tráfego e exige uma gestão inteligente de recursos.

Uma outra vantagem é a maneira em que o Nginx serve arquivos estáticos. O servidor utiliza um modelo de processos leves para lidar com solicitações de arquivos estáticos. A arquitetura de processos leves no Nginx é projetada para minimizar o consumo de recursos, especialmente em termos de memória e CPU. Os processos leves são estruturas mais eficientes em comparação com os processos tradicionais, pois requerem menos recursos do sistema. Isso é especialmente valioso ao lidar com a concorrência de conexões.

Para explicar a arquitetura do Nginx, é necessário começar com uma visão geral dos seus componentes. O Nginx opera com um processo mestre (*master process)* e vários processos trabalhadores (*worker processes*). O processo mestre é responsável pela gestão e coordenação dos processos trabalhadores e é iniciado durante o *boot* do servidor, sendo executado com privilégios elevados para gerenciar a configuração e as tarefas de controle. Os processos trabalhadores são responsáveis por lidar com as conexões e as solicitações dos clientes. O número de processos trabalhadores é ajustável e pode ser configurado de acordo com as necessidades. A capacidade de adicionar ou remover processos trabalhadores dinamicamente permite que o Nginx se ajuste automaticamente à carga do servidor.

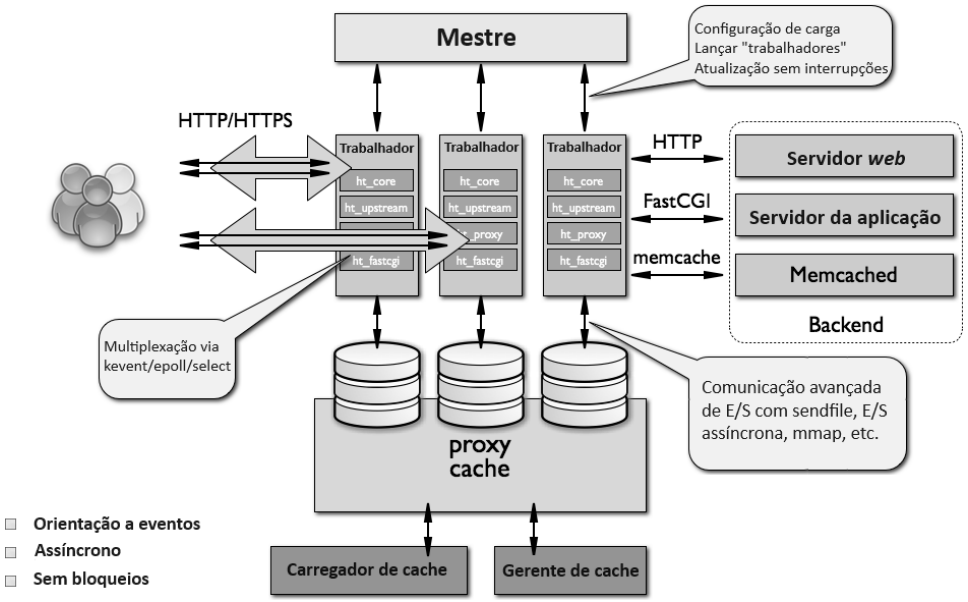

Figura 3. Visão geral da arquitetura de um servidor Nginx. ***Tradução livre de "The Architecture of Open-Source Applications, Volume 2", Andrew Alexeev, 2012, p. 215.***

A Figura 3 apresenta uma visão geral da arquitetura do Nginx, destacando os principais componentes e sua interação. Estes operam em um modelo de processamento de dois níveis, com um "*master*" supervisionando o servidor e "*workers*" realizando o processamento real das solicitações. Os "usuários" representam as origens das requisições HTTP, enquanto as "setas" indicam essas solicitações. Os "*workers*" encaminham as solicitações para o "*backend*," que consiste em servidores HTTP, servidores de aplicativos e o sistema de cache em memória Memcached. O Nginx utiliza técnicas

avançadas de E/S para otimizar o desempenho, como "*sendfile*" e "*mmap*." Além disso, a ferramenta implementa um sistema de cache que armazena respostas para economizar recursos e acelerar o tempo de resposta, com um "*Cache Loader*" para preencher o cache e um "*Cache Manager*" para gerenciar sua validade.

Ao receber uma solicitação de arquivo estático, o processo principal encaminha a solicitação para um dos processos de trabalho disponíveis. Cada processo de trabalho é capaz de manipular várias solicitações simultaneamente, graças ao modelo assíncrono e orientado a eventos do Nginx. A instalação padrão da ferramenta pode ser encontrada em https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/.

1.8.1 Configuração e arquivos-chave

Para compreender o funcionamento da configuração do Nginx, faz-se necessário estar familiarizado com a estrutura de seus arquivos de configuração. O caminho absoluto necessário para acessar esses arquivos pode variar de acordo com a distribuição e a instalação do Nginx, no entanto, algumas das principais localizações comumente utilizadas são:

- Arquivo de Configuração Principal: **/etc/nginx/nginx.conf**;
- Diretório de Configuração de Sites Disponíveis: **/etc/nginx/sites-available/**;
- Diretório de Configuração de Sites Ativos (Link Simbólico): **/etc/nginx/sites-enabled/;**
- Diretório de Configurações mais Específico (pode variar): **/etc/nginx/conf.d/.**

Sendo que as configurações essenciais, como as relacionadas às diretivas "gzip" e "sendfile", geralmente são encontradas no arquivo principal nginx.conf.

O caminho absoluto para acessar o diretório "www" em um sistema GNU/Linux, no qual são hospedados sites da *web* no servidor, pode variar dependendo da configuração específica do mesmo. No entanto, no ambiente Debian 12, que é o foco do presente estudo, o diretório "www" pode ser encontrado em ***/var/www***. Dentro do diretório "www", a página inicial, comumente nomeada "index.html" deve estar localizada. Caso a página inicial tenha outro nome, é necessário ajustar o arquivo de configuração do servidor Nginx, que geralmente é encontrado em ***/etc/nginx/sites-available/***, adicionando o nome da página no mesmo para que esta seja servida corretamente. O diretório ***conf.d*** pode ter a mesma função dos diretórios de sites mencionados anteriormente, apenas variando na estrutura de arquivos de acordo com o ambiente utilizado. Portanto, certifique-se de verificar as configurações específicas do servidor para confirmar o caminho absoluto correto e o nome do arquivo da página inicial, conforme necessário.

1.9 Ambiente de Testes

Antes de explorar as transformações que a computação em nuvem trouxe para o mundo da TI e como esta impactou o papel dos sistemas operacionais e servidores, é fundamental entender o que é computação em nuvem.

Segundo a Amazon *Web Services* (AWS), a computação em nuvem, refere-se a uma abordagem de fornecimento de recursos de TI, como servidores, armazenamento, bancos de dados e serviços, pela Internet. Esses recursos são hospedados e gerenciados em *data centers* escaláveis e virtualizados, que estão distribuídos geograficamente em todo o mundo.

Uma das principais características da computação em nuvem é a capacidade de provisionar e dimensionar recursos sob demanda. Isso significa que as empresas podem acessar rapidamente recursos de *hardware* e *software*, como servidores, sem a necessidade de investimentos em infraestrutura física. Este método de computação também oferece a flexibilidade de pagar apenas pelos recursos que são realmente utilizados, em vez de comprar antecipadamente capacidade que pode ficar subutilizada.

Os serviços em nuvem são categorizados em três principais modelos de implantação: nuvem pública, nuvem privada e nuvem híbrida. A nuvem pública oferece serviços a partir de provedores em nuvem acessíveis ao público em geral. A nuvem privada é operada por uma única organização, geralmente para atender a requisitos de segurança e conformidade. A nuvem híbrida combina ambas as abordagens, permitindo que as empresas mantenham o controle sobre algumas partes de sua infraestrutura enquanto aproveitam serviços em nuvem públicos.

Agora, com um entendimento básico da computação em nuvem, é possível explorar como essa tecnologia transformou a forma como os sistemas operacionais e servidores *web* são utilizados no cenário de TI.

No tempo em que hospedagem em nuvem não era uma realidade, os sistemas operacionais eram amplamente usados para gerenciar servidores físicos em *data centers* locais. Sistemas operacionais tradicionais, como o GNU/Linux e o MS-Windows, desempenhavam um papel fundamental no gerenciamento de recursos de *hardware*, execução de aplicativos e fornecimento de serviços.

No entanto, esses sistemas operacionais estavam sujeitos às limitações dos servidores físicos. A escalabilidade era um desafio, pois a adição de recursos de *hardware* frequentemente implicava custos significativos e tempos de implementação demorados. Além disso, o desempenho de servidores *web*, como o Nginx, estava intrinsecamente ligado à eficiência do sistema operacional e à alocação de recursos.

O surgimento do AWS e de outras plataformas de computação em nuvem revolucionou a forma como os sistemas operacionais e os servidores *web* são usados. As motivações por trás da criação destes serviços - escalabilidade, flexibilidade e eficiência - foram uma resposta direta aos desafios enfrentados pelo mercado.

Com essa tecnologia disruptiva, a infraestrutura de TI tornou-se mais escalável e flexível. O que antes estava sob responsabilidade das empresas não mais requer sua intervenção. Sistemas operacionais e servidores *web*, agora podem ser implantados em instâncias virtuais em questão de minutos. A alocação de recursos tornou-se dinâmica, permitindo que as empresas atendam às demandas flutuantes sem investimentos em *hardware* caro.

A economia de recursos financeiros e de tempo, aliada à segurança proporcionada por ambientes em nuvem, juntamente com a gama de serviços oferecidos pela Amazon *Web Services* (AWS) a custos acessíveis e personalizados, tem se revelado uma receita de sucesso, conforme observado nos cases publicados no *website* da Amazon *Web*

Services. Isso possibilitou que companhias que anteriormente não tinham a capacidade de manter um *data center* local para hospedar seus projetos, agora pudessem implantar seus sistemas de forma econômica e eficiente. Esse cenário resulta em uma vantagem competitiva significativa, com redução de custos, rápida implementação, alta disponibilidade do serviço e todos os benefícios inerentes à hospedagem em ambientes de nuvem.

Essa evolução na hospedagem tem implicações diretas para o estudo de caso em questão. Ao testar as configurações *Sendfile* e Gzip no Nginx 1.24 em um ambiente Debian 12 hospedado na AWS, se explora como as configurações do servidor *web* interagem com a infraestrutura moderna. A escalabilidade dinâmica e a eficiência dos sistemas operacionais em ambientes de nuvem podem influenciar o desempenho do Nginx e a otimização de recursos.

1.9.1 Amazon EC2

O Amazon *Elastic Compute Cloud*, conhecido como Amazon EC2, é um serviço de computação em nuvem oferecido pela AWS. O EC2 desempenha um papel central na infraestrutura de nuvem, fornecendo instâncias virtuais escaláveis nas quais as organizações podem executar aplicativos, hospedar *sites* e realizar uma variedade de tarefas computacionais conforme descrito pela própria fornecedora deste serviço.

Uma das características mais notáveis do Amazon EC2 é a escalabilidade. Segundo a AWS (2023), estas instâncias permitem que os usuários dimensionem recursos de computação para cima ou para baixo conforme necessário, respondendo a picos de tráfego ou alterações nas demandas de recursos. Essa flexibilidade é indispensável para garantir que as aplicações e servidores possam lidar com cargas variáveis de trabalho. O EC2 oferece uma variedade de tipos de instância, cada um otimizado para diferentes casos de uso, desde instâncias de uso geral até instâncias otimizadas para computação, armazenamento, GPU etc. Isso permite a escolha das instâncias mais adequadas para as necessidades específicas do aplicativo ou servidor.

Os usuários pagam apenas pelos recursos que utilizam. Isso significa que não há necessidade de investimentos significativos em uma estrutura física, tornando-o uma opção econômica para empresas de todos os portes. A implantação de instâncias do Amazon EC2 é rápida e fácil, permitindo que as configurações do servidor sejam provisionadas em questão de minutos, especialmente relevante para o estudo de caso, no qual a configuração rápida e precisa é fundamental.

1.9.2 Máquina t2.micro

A instância t2.micro é uma das instâncias de menor custo e menor capacidade oferecidas pelo Amazon EC2 conforme observado na Figura 4. Sendo assim, é amplamente utilizada para fins de desenvolvimento, testes e para cenários de baixo tráfego. A instância t2.micro possui:

- 1 núcleo de CPU virtual;
- 1 GB de memória RAM;
- O armazenamento é baseado em instâncias do tipo EBS (Elastic Block Store);

- A instância t2.micro é conectada à rede Amazon *Virtual Private Cloud* (Amazon VPC) e tem largura de banda de rede moderada.

Nome	vCPUs	RAM (GiB)	Créditos de CPU/h	Preço sob demanda/hora*	Instância reservada por 1 ano – por hora*	Instância reservada por 3 anos – por hora*
t2.nano	1	0,5	3	0,0058 USD	0,003 USD	0,002 USD
t2.micro	1	1,0	6	0,0116 USD	0,007 USD	0,005 USD
t2.small	1	2.0	12	0,023 USD	0,014 USD	0,009 USD
t2.medium	2	4,0	24	0,0464 USD	0,031 USD	0,021 USD
t2.large	2	8,0	36	0,0928 USD	0,055 USD	0,037 USD
t2.xlarge	4	16,0	54	0,1856 USD	0,110 USD	0,074 USD
t2.2xlarge	8	32,0	81	0,3712 USD	0,219 USD	0,148 USD

Figura 4. Valores das instâncias ofertadas do Amazon EC2. Retirado de: https://aws.amazon.com/pt/ec2/instance-types/t2/. Acessado em: 15 nov. 2023.

Um dos recursos distintivos da instância t2.micro é que esta faz parte da família de instâncias "t2", que oferece a capacidade de escalabilidade baseada em créditos. Isso significa que, sob condições normais, a instância t2.micro acumula créditos de CPU. Quando a carga de trabalho excede a capacidade da CPU, usa os créditos para fornecer mais capacidade de processamento temporariamente. Este fator é útil para lidar com picos de carga, mas a instância pode ser limitada se não houver créditos disponíveis. Ademais, a instância t2.micro é compatível com uma variedade de sistemas operacionais, incluindo Linux, Windows, FreeBSD, Debian e outros, o que amplia ainda mais as possibilidades de utilização da mesma.

1.9.3 Conexão SSH: Uma Forma Segura de Acesso a Servidores

Considerando a importância da segurança na administração de servidores, a utilização do *Secure Shell* (SSH) destaca-se como uma abordagem fundamental para garantir acesso remoto de forma segura, especialmente em ambientes baseados em Linux e Unix. De acordo com Seggelmann, Tüxen e Rathgeb (2012), o *Secure Shell* é uma tecnologia empregada como uma abordagem segura e cifrada para conexão de servidores e máquinas remotas. Esta permite que profissionais de TI acessem, gerenciem e administrem servidores e máquinas que executem diversos sistemas operacionais. Em ambientes de servidores baseados em Linux e Unix, a conexão SSH é utilizada para acesso seguro a terminais remotos, possibilitando as operações em servidores sem a necessidade de acesso físico ao local destes. É importante ressaltar que a escolha do sistema operacional influencia o tipo de autenticação e configuração necessária neste tipo de conexão.

O cliente usa este protocolo de autenticação para solicitar permissão para utilização de um serviço, fornecendo um usuário e o servidor, em contrapartida oferece métodos aceitáveis de autenticação, tais como senha ou chave pública. Uma vez que a autenticação é bem-sucedida, o cliente tem permissão para solicitar o serviço necessário [SEGGELMANN; TÜXEN; RATHGEB 2012]. Utilizando métodos robustos, como senhas ou chaves públicas, o SSH fortalece a autenticação do cliente para acessar os serviços necessários com sucesso. Portanto, a integração do SSH simplifica operações em

ambientes distribuídos e reforça a segurança dos sistemas, contribuindo para um gerenciamento eficaz e remoto dos recursos de TI.

1.9.4 Arquivos hospedados no servidor do projeto *Tech Blog*

Neste tutorial, a página *Tech Blog* foi hospedada no servidor criado, sendo esta uma página estática, visto o desempenho do Nginx ao servir este tipo de arquivo. A estrutura de arquivos dá-se conforme a Tabela 2, partindo do diretório *src*:

Tabela 2. Casos de simulação de requisições.

Arquivo/Diretório	Quantidade	Espaço Utilizado
index.html	1	9,94 KB
/assets/css	2	118,3 KB
/assets/js	5	108 KB
/assets/sass/base	3	5,28 KB
/assets/sass/componentes	16	20 KB
/assets/sass/layout	7	9,61 KB
/assets/sass/libs	6	18,9 KB
/assets/sass/main.scss	1	1,51 KB
/assets/webfonts	15	2,79 MB
/images	10	614 KB

A Tabela 2 apresenta a estrutura de diretórios de um projeto em uma página estática da *web*, oferecendo uma visão detalhada da organização hierárquica. Cada entrada na tabela representa uma pasta no projeto, acompanhada pelo número correspondente de arquivos contidos e o tamanho total desses arquivos. A Tabela 2 oferece uma representação visual clara, contribuindo para uma análise da composição do projeto e sua estrutura de arquivos.

Ao explorar a estrutura de arquivos do projeto hospedado no servidor dedicado, ganha-se uma perspectiva da organização por trás dessa página estática. A tabela fornecida revela não apenas a distribuição de arquivos em diferentes diretórios, mas também oferece uma compreensão clara da quantidade e do tamanho desses arquivos. A estrutura organizada não só otimiza o desempenho, mas também facilita a manutenção e expansão futura do projeto. Em suma, ao desvendar os detalhes da arquitetura de arquivos, solidificamos a base para um futuro desenvolvimento do projeto e da página hospedada.

1.9.5 Métricas de monitoramento

Para conduzir o teste de desempenho no servidor *web* Nginx 1.24 com as configurações *Sendfile* e *Gzip*, é essencial preparar um ambiente adequado no lado do cliente. O ambiente do cliente refere-se ao sistema a partir do qual os testes de desempenho serão iniciados. Nesta configuração, um ambiente com base no sistema operacional Debian 12 foi escolhido, visto que é conhecido por sua estabilidade e ampla utilização em ambientes de servidor.

O ApacheBench, ou simplesmente "ab," é uma ferramenta de análise de desempenho de servidores HTTP que gera requisições HTTP para um endereço URL específico, proporcionando insights sobre o comportamento do servidor por meio de um processo de teste [TOMIŠA; MILKOVIĆ; ČAČIĆ 2019]. É uma excelente escolha para realizar testes de carga em servidores *web*, incluindo o Nginx. O ab é uma ferramenta amplamente utilizada para medir a capacidade de resposta e desempenho de um servidor *web* em termos de requisições por segundo, tempo de resposta e entre outros. A instalação padrão da ferramenta em sistemas operacionais Linux pode ser realizada utilizando o comando ***sudo apt install apache2-utils.***

Após definida a ferramenta que irá gerar as requisições, para a realização deste estudo comparativo é necessário medir o quanto de recurso do servidor foi consumido. Neste sentido, o comando Pidstat é uma ferramenta do ambiente GNU/Linux, usada para monitorar atividades de tarefas individuais gerenciadas pelo núcleo. Esta fornece informações sobre o desempenho dessas tarefas, incluindo detalhes como a utilização da CPU. Ao selecionar tarefas específicas ou todas as tarefas, o Pidstat oferece insights sobre o comportamento do sistema em relação a essas atividades. Sua funcionalidade básica envolve a geração de relatórios que revelam dados essenciais, tornando-o uma peça valiosa para análise e compreensão do desempenho do sistema [Pidstat. Linux Manual Page, 2023]. Dessa forma, o comando Pidstat será utilizado para coletar as métricas de desempenho do servidor durante a realização das requisições. Essas métricas serão analisadas para comparar o consumo de recursos do servidor conforme alteração nos cenários descritos na Subseção 1.10.

1.10 Cenário de teste

Os cenários de teste definidos buscam simular dois casos de múltiplas requisições simultâneas e não simultâneas no site Tech Blog, de forma que seja possível avaliar a quantidade de requisições por segundo, a porcentagem de CPU consumida e memória em *kilobytes* para as quatro configurações do servidor alvos de comparação. Na Tabela 3, estão descritos esses dois casos: o primeiro, totalizando 10.000 requisições sem concorrência entre estas; e o segundo totalizando 40.000 requisições com uma concorrência de 10 conexões.

Tabela 3. Casos de simulação de requisições.

Caso	Quantidade de requisições	Conexões concorrentes	Quantidade de amostras
1	10000	1	15
2	40000	10	15

Os casos apresentados na Tabela 3 representam, respectivamente, uma simulação com um número menor de requisições e outra com um número maior de requisições e concorrência. Cada caso será testado 15 vezes (representando 15 amostras) para assegurar consistência nos resultados. Uma 'amostra' refere-se a uma execução completa e ininterrupta das requisições via ApacheBench. Assim, haverá um total de 30 amostras para cada uma das quatro configurações de servidor discutidas nas próximas subseções.

1.11 Cenários de configuração

O propósito dos cenários de configuração é avaliar a combinação das diretivas *gzip* e *sendfile* no desempenho do servidor, focando em requisições por segundo, consumo de CPU e memória durante simulações de requisições na página TechBlog. Segundo Nelson (2014), ambas são recomendadas para potencializar o desempenho do Nginx. Contudo, é crucial entender que estas não podem ser usadas simultaneamente para servir o mesmo arquivo. Esta restrição deve-se ao fato de que a diretiva *sendfile* opera a nível de núcleo, transferindo dados diretamente entre descritores de arquivos, enquanto a *gzip* atua a nível de usuário dentro do Nginx, tornando a combinação das duas ineficaz. Dessa forma, foram definidos quatro cenários de configuração do servidor Nginx a partir das duas diretivas, descritos nas Subseções 1.11.1 até 1.11.4.

1.11.1 Cenário 1

O primeiro cenário mantém ambas as diretivas desligadas para todos os arquivos, possibilitando que, ao comparar com outros cenários que apresentam as diretivas ativadas, seja possível verificar qual foi o impacto em questão de requisições por segundo, consumo de CPU e memória que tais ativações causaram. Portanto, mesmo esta configuração podendo não trazer nenhuma vantagem para o desempenho do servidor, esse cenário inicial é importante e serve como base na comparação e análise de todos os outros cenários.

1.11.2 Cenário 2

Neste segundo cenário, a diretiva *sendfile* é mantida desligada, enquanto a diretiva *gzip* é mantida ativada. O objetivo é observar o impacto da compressão *gzip* nas métricas escolhidas. Vale ressaltar que, neste cenário, nem todas as operações de transferência de dados se beneficiarão da diretiva *gzip*: arquivos JPG, por exemplo, já possuem uma forma de compressão nativa e, por isso, não receberão otimizações adicionais por meio desta diretiva. Ao analisar este cenário, é buscado entender não apenas os benefícios diretos da compressão, mas também como esta se compara com os demais cenários.

1.11.3 Cenário 3

No terceiro cenário, são invertidas as aplicações das diretivas: a diretiva *sendfile* será ativada e a *gzip* desativada. Ao contrário do cenário anterior, onde a compressão *gzip* se mostrava seletiva em relação aos tipos de arquivos otimizados, o *sendfile* tem a vantagem de aplicar sua otimização de transmissão de dados a todos os arquivos servidos, incluindo

imagens. A partir desse cenário, é possível não só observar o impacto direto da diretiva *sendfile*, mas também comparar seu efeito de desempenho em relação ao cenário que utilizava a compressão *gzip*.

1.11.4 Cenário 4

O quarto e último cenário oferece uma abordagem híbrida, ativando as diretivas *sendfile* e *gzip* para diferentes tipos de arquivo. Neste arranjo, a diretiva *sendfile* é aplicada para arquivos de imagem (JPG), os quais já possuem compressão e podem se beneficiar do transporte eficiente de dados que o *sendfile* oferece. Por outro lado, os arquivos de texto como HTML, CSS e JS serão comprimidos usando a diretiva *gzip*. Esta configuração se propõe analisar uma solução personalizada que visa a utilização de cada diretiva segundo as características específicas dos arquivos. Dessa forma, os quatro cenários podem ser representados na Tabela 4, com as suas devidas características.

Tabela 4. Cenários de teste e suas características.

Cenário	Utiliza *gzip*	Utiliza *sendfile*
1	Não.	Não.
2	Sim, para todos os arquivos com exceção de imagens.	Não.
3	Não.	Sim, para todos os arquivos.
4	Sim, para arquivos de texto.	Sim, para arquivos de imagem.

Na Tabela 4 é representado cada cenário de teste em uma linha, e as duas colunas finais informam se o cenário de configuração utiliza ou não cada diretiva, informando também para quais tipos de arquivos, caso aplicada, a diretiva atuará.

1.12 Resultados dos experimentos

Os experimentos foram executados a partir das configurações de cada cenário que podem ser verificadas no arquivo *README.md* do repositório do site TechBlog disponível em https://github.com/Mateus-Mannes/tech-blog-web-page-example, na Seção 2 (“Configurações aplicadas”). Além disso, os comandos utilizados da ferramenta ApacheBench para simulação das requisições e da ferramenta PidStat para monitoramento de CPU e memória também estão presentes no mesmo arquivo na Seção 3 (“Scripts utilizados”). Com base nisso, os resultados dos testes de caso foram representados na Tabela 5, no qual cada linha representa uma amostra.

Tabela 5. Resultados dos experimentos.

Cenário	Caso	Amostra	Requisições por segundo	Consumo de CPU	Consumo de memória
1	1	1	934,34	10,00%	4104 KB
1	1	2	932,73	11,00%	4104 KB
1	1	3	932,18	10,00%	4104 KB

1	1	4	929,83	11,00%	4104 KB
1	1	5	923,49	10,00%	4104 KB
1	1	6	922,32	11,00%	4104 KB
1	1	7	920,58	10,00%	4104 KB
1	1	8	916,32	10,00%	4104 KB
1	1	9	915	11,00%	4104 KB
1	1	10	913,32	10,00%	4104 KB
1	1	11	906,63	11,00%	4104 KB
1	1	12	902,09	11,00%	4104 KB
1	1	13	899,44	10,00%	4104 KB
1	1	14	891,31	10,00%	4104 KB
1	1	15	852,73	11,00%	4104 KB
1	2	1	2459,15	32,00%	2556 KB
1	2	2	2676,16	33,00%	2556 KB
1	2	3	2622,69	34,00%	2556 KB
1	2	4	2553,12	34,00%	2556 KB
1	2	5	2491,72	34,00%	2556 KB
1	2	6	2692,5	34,00%	2556 KB
1	2	7	2767,45	34,00%	2556 KB
1	2	8	2744,25	34,00%	2556 KB
1	2	9	2524,79	35,00%	2556 KB
1	2	10	2810,83	35,00%	2556 KB
1	2	11	2621,78	36,00%	2556 KB
1	2	12	2657,14	36,00%	2556 KB
1	2	13	2841,12	36,00%	2556 KB
1	2	14	2740,14	36,00%	2556 KB
1	2	15	2532,01	36,00%	2556 KB
2	1	1	859,51	11,00%	4104 KB
2	1	2	871,64	10,00%	4104 KB
2	1	3	891,67	11,00%	4104 KB
2	1	4	897,22	10,00%	4104 KB
2	1	5	897,88	10,00%	4104 KB
2	1	6	898,96	10,00%	4104 KB
2	1	7	907,45	10,00%	4104 KB
2	1	8	908,27	10,00%	4104 KB
2	1	9	908,47	10,00%	4104 KB
2	1	10	912,99	11,00%	4104 KB
2	1	11	914,51	11,00%	4104 KB
2	1	12	920,9	11,00%	4104 KB
2	1	13	927,29	11,00%	4104 KB
2	1	14	927,65	11,00%	4104 KB

2	1	15	928,04	11,00%	4104 KB
2	2	1	2580,01	32,00%	2540 KB
2	2	2	2526,52	32,00%	3792 KB
2	2	3	2523,65	32,00%	3792 KB
2	2	4	2521,02	32,00%	2540 KB
2	2	5	2504,89	32,00%	3728 KB
2	2	6	2491,05	32,00%	3792 KB
2	2	7	2569,22	33,00%	2540 KB
2	2	8	2564,63	33,00%	2540 KB
2	2	9	2536,75	33,00%	2540 KB
2	2	10	2575,16	34,00%	3792 KB
2	2	11	2540,32	34,00%	2540 KB
2	2	12	2535,05	34,00%	3728 KB
2	2	13	2533,85	34,00%	2540 KB
2	2	14	2515,59	34,00%	2540 KB
2	2	15	2620,08	35,00%	3728 KB
3	1	1	900,01	11,00%	2540 KB
3	1	2	896,24	11,00%	2540 KB
3	1	3	894,96	11,00%	2540 KB
3	1	4	887,61	10,00%	2540 KB
3	1	5	882,66	11,00%	2540 KB
3	1	6	876,74	11,00%	2540 KB
3	1	7	874,44	10,00%	2540 KB
3	1	8	873,46	10,00%	2540 KB
3	1	9	869,7	11,00%	2540 KB
3	1	10	869,34	10,00%	2540 KB
3	1	11	869,23	10,00%	2540 KB
3	1	12	859,44	11,00%	2540 KB
3	1	13	858,16	10,00%	2540 KB
3	1	14	841,7	11,00%	2540 KB
3	1	15	840,61	10,00%	2540 KB
3	2	1	2643,8	31,00%	2600 KB
3	2	2	2612,94	33,00%	2600 KB
3	2	3	2574,11	30,00%	2600 KB
3	2	4	2568,52	29,00%	2600 KB
3	2	5	2556,21	33,00%	2600 KB
3	2	6	2522,36	30,00%	2600 KB
3	2	7	2518,71	30,00%	2600 KB
3	2	8	2478,43	30,00%	2600 KB
3	2	9	2471,39	28,00%	2600 KB
3	2	10	2453,48	30,00%	2600 KB

3	2	11	2448,54	32,00%	2600 KB
3	2	12	2446,58	28,00%	2600 KB
3	2	13	2444,7	28,00%	2600 KB
3	2	14	2426,12	29,00%	2600 KB
3	2	15	2412,76	28,00%	2600 KB
4	1	1	834,06	10,00%	4148 KB
4	1	2	830,76	10,00%	4148 KB
4	1	3	829,91	10,00%	4148 KB
4	1	4	828,35	10,00%	4148 KB
4	1	5	826,09	10,00%	4148 KB
4	1	6	821,58	10,00%	4148 KB
4	1	7	819,25	10,00%	4148 KB
4	1	8	818,34	10,00%	4148 KB
4	1	9	817,44	10,00%	4148 KB
4	1	10	807,44	10,00%	4148 KB
4	1	11	806,8	9,00%	4148 KB
4	1	12	798,03	9,00%	4148 KB
4	1	13	792,99	10,00%	4148 KB
4	1	14	788,45	9,00%	4148 KB
4	1	15	775,5	10,00%	4148 KB
4	2	1	2356,84	28,00%	4104 KB
4	2	2	2361,69	29,00%	4104 KB
4	2	3	2375,53	29,00%	4104 KB
4	2	4	2383,36	28,00%	4104 KB
4	2	5	2411,47	27,00%	4104 KB
4	2	6	2413,23	29,00%	4104 KB
4	2	7	2420,45	29,00%	4104 KB
4	2	8	2430,6	28,00%	4104 KB
4	2	9	2434,17	28,00%	4104 KB
4	2	10	2448,19	29,00%	4104 KB
4	2	11	2449,48	29,00%	4104 KB
4	2	12	2449,7	29,00%	4104 KB
4	2	13	2457,51	29,00%	4104 KB
4	2	14	2464	28,00%	4104 KB
4	2	15	2467,1	28,00%	4104 KB

A Tabela 5 consolida os resultados obtidos nos testes, distribuídos em colunas conforme a seguinte ordem: cenário de configuração, que varia de 1 a 4 conforme detalhado nas Subseções 1.11.1 a 1.11.4; casos de teste, enumerados de 1 a 2 e descritos na Subseção 1.10; o número da amostra, que progride de 1 a 15 para cada caso; o número

de requisições por segundo, que foi explicado na Subseção 1.4.1; consumo percentual de CPU e consumo de memória principal em *kilobytes*, ambos detalhados na Subseção 1.4.2.

Para simplificar a análise comparativa dos resultados, os dados de cada cenário e caso foram agrupados, calculando-se a média aritmética para os valores de requisições por segundo, consumo de CPU e uso de memória. Esta abordagem permite uma visão consolidada dos resultados para o Caso 1 na Tabela 6 e para o Caso 2 na Tabela 7. Adicionalmente, as Figuras 5 e 6 complementam essas informações com gráficos de bolhas que ilustram visualmente os dados tabulados, facilitando o entendimento das variações observadas entre os diferentes cenários e casos testados.

Tabela 6. Resultados agrupados do Caso 1.

Cenário	Média requisições por segundo	Desvio padrão requisições por segundo	Média consumo CPU	Desvio padrão consumo CPU	Média consumo memória	Desvio padrão consumo memória
1	912,82	20,35	10,47%	0,50%	4104 KB	0 KB
2	904,83	19,12	10,53%	0,50%	4104 KB	0 KB
3	872,95	17,41	10,53%	0,50%	2540 KB	0 KB
4	813,00	16,96	9,80%	0,40%	4148 KB	0 KB

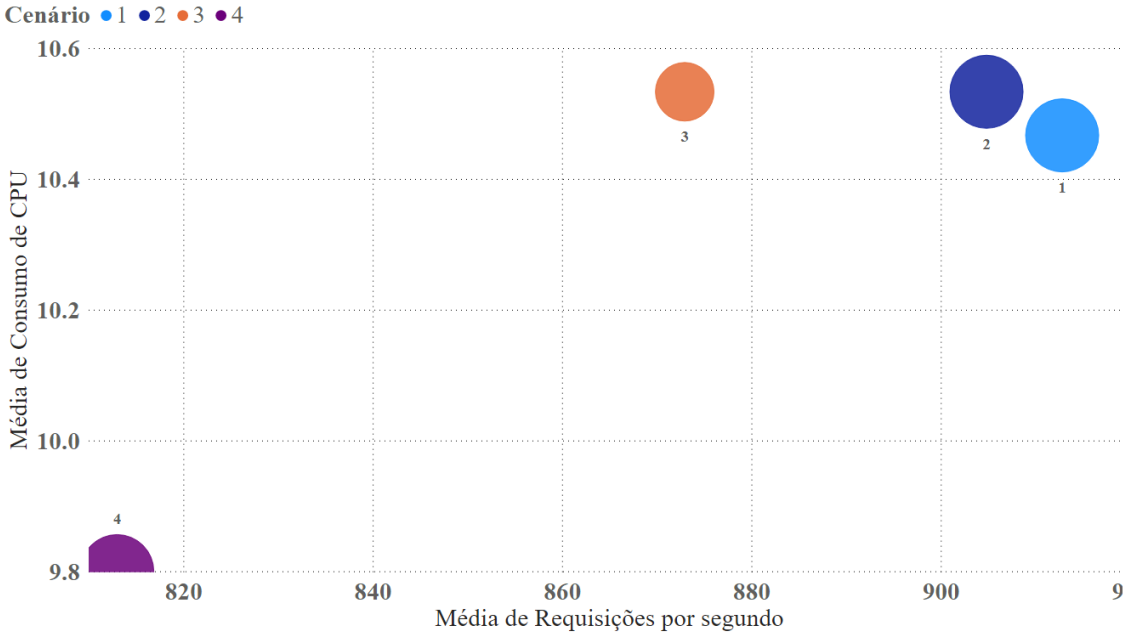

Figura 5. Gráfico de bolhas do Caso 1. *Criado pelos autores.*

Tabela 7. Resultados agrupados do Caso 2.

Cenário	Média requisições por segundo	Desvio padrão requisições por segundo	Média consumo CPU	Desvio padrão consumo CPU	Média consumo memória	Desvio padrão consumo memória
1	2648,99	114,81	34,6%	1,20%	2556 KB	0 KB

2	2542,56	32,37	33,06%	1,00%	3111 KB	611,31 KB
3	2505,24	69,42	29,93%	1,65%	2600 KB	0 KB
4	2421,55	35,79	28,46%	0,62%	4104 KB	0 KB

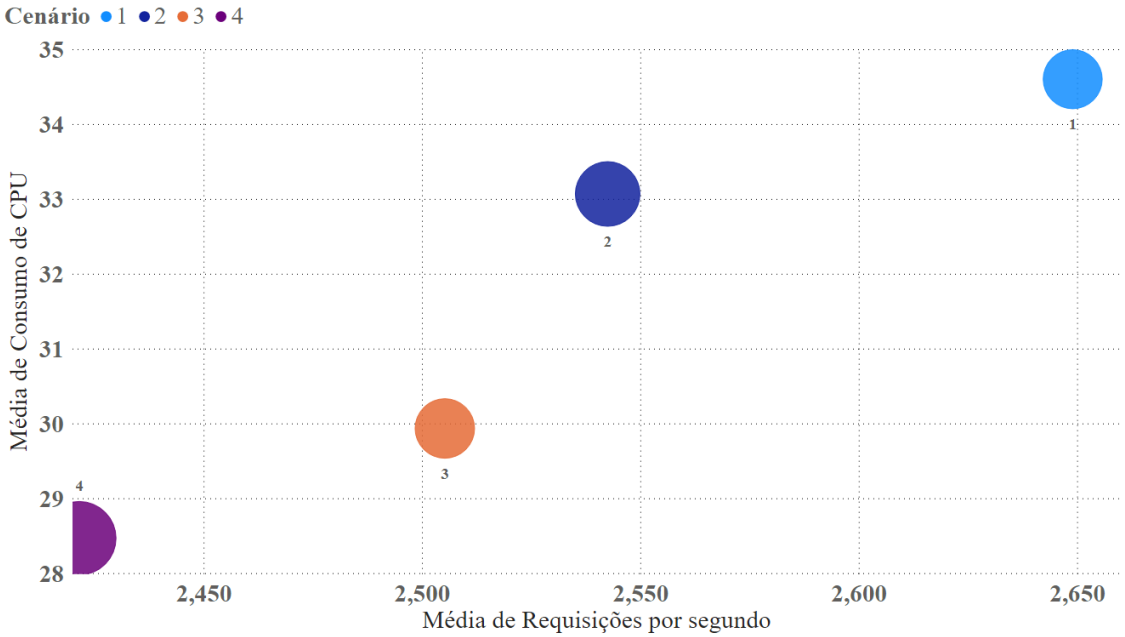

Figura 6. Gráfico de bolhas do Caso 2. *Criado pelos autores.*

As Tabelas 6 e 7 seguem uma estrutura padronizada para ilustrar os resultados dos Casos 1 e 2, respectivamente. Cada linha corresponde a um cenário testado, identificado na primeira coluna. As colunas subsequentes detalham os valores médios obtidos a partir das 15 amostras coletadas para cada caso. Em paralelo, as Figuras 5 e 6 adotam uma representação gráfica compatível, utilizando gráficos de bolha para uma interpretação visual dos dados. Nestes gráficos, o eixo vertical mapeia o consumo percentual médio de CPU, o eixo horizontal indica o número médio de requisições por segundo, e o tamanho das bolhas reflete o volume médio de memória consumido, medido em *kilobytes*. Essa configuração gráfica permite uma comparação direta entre os cenários, ressaltando as relações entre o desempenho em termos de requisições por segundo, o uso da CPU e o consumo de memória.

1.13 Considerações finais

Uma abordagem para realizar ajustes eficazes nas configurações do Nginx é alterar uma configuração de cada vez e analisar o impacto no desempenho do servidor [NELSON 2014]. Nesse contexto, este estudo testou diferentes combinações das diretivas *sendfile* e *gzip* em requisições a um *website* de teste. As medições de requisições por segundo, uso de CPU e memória forneceram dados para comparar o efeito de cada configuração.

Dentro dos 120 testes executados, o consumo de memória do servidor atingiu um pico de 4148 *kilobytes*, o que representa apenas 0,39% do 1 gigabyte de capacidade total da máquina t2.micro utilizada nos testes. Das 8 combinações de cenários e casos executados, apenas em uma destas as amostras apresentaram variação do consumo de

memória (Cenário 2, Caso 2). Curiosamente, apesar do baixo e consistente uso de memória através dos cenários, o cenário 4 destacou-se com uma média de consumo de memória maior em comparação à média dos outros cenários testados nos Casos 1 e 2.

Quanto ao consumo de CPU, é perceptível que, em todos os cenários, houve um aumento no caso 2 — caracterizado pelo maior número de conexões concorrentes em comparação com o caso 1. Notavelmente, o primeiro caso apresentou uma variação de CPU baixa entre os cenários, indo de 9,80% a 10,53%. Já no Caso 2, os Cenários 1 e 2 apresentaram um maior consumo, enquanto os Cenário 3 e 4, que utilizaram o sendfile, apresentaram um menor consumo.

Analisando o número médio de requisições por segundo, observa-se um padrão similar ao consumo de CPU, com um acréscimo no desempenho ao se transitar do caso 1 para o caso 2. Surpreendentemente, o Cenário 1 superou os outros nessa medida, seguido pelos Cenários 2, 3 e 4, nessa ordem, para ambos os casos.

Em conclusão, os dados revelam que a configuração do Cenário 1, com as diretivas desativadas, destacou-se pelo desempenho superior em requisições por segundo. Em contraste, as configurações que empregaram a diretiva *sendfile* apresentaram os menores números de requisições por segundo. Entretanto, é notável que os cenários com *sendfile* tiveram os menores consumos de CPU, indicando um impacto menor sobre os recursos de processamento do servidor. Já o Cenário 2 que utilizava apenas o *gzip* ficou mais próximo do Cenário 1 em requisições por segundo, porém teve um consumo maior ou igual de CPU se comparado com os cenários que utilizaram o *sendfile.*

Referências

AMARAL, Francisco Eudes do. Análise de Desempenho do Servidor Web Apache no provedor FlorianoNet. 2002. 94 f. Dissertação (Mestrado) - Curso de Pós-Graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2002. Cap. 3. Disponível em: https://repositorio.ufpe.br/handle/123456789/2571. Acesso em: 26 out. 2023.

Amazon EC2. Amazon EC2 features, disponível em: https://aws.amazon.com/ec2/features. Acesso em: 06 out. 2023.

ApacheBench. ApacheBench docs, disponível em: https://httpd.apache.org/docs/2.4/programs/ab.html. Acesso em: 06 out. 2023.

ALEXEEV, Andrew. Em: BROWN, Amy; WILSON, Greg (Eds.). The Architecture of Open-Source Applications, Volume 2. Creative Commons Attribution 3.0 Unported license (CC BY 3.0). Disponível em: http://www.aosabook.org/. Acesso em: 23 out. 2023.

CASTRO, Pedro H. P.; CORRÊA, Sand; CARDOSO, Kleber V.. Uma Abordagem Baseada no Consumo de CPU e RAM para a Eficiência Energética em Centros de Dados para Computação em Nuvem. In: SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 14., 2013, Porto de Galinhas. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2013. p. 118-125. DOI: https://doi.org/10.5753/wscad.2013.16781.

CERN. A short history of the Web. [*S. l.*], 2023. Disponível em: https://home.cern/science/computing/birth-web/short-history-web#:~:text=By%20the%20end%20of%201990,This%20machine%20is%20a%20server. Acesso em: 16 out. 2023.

Debian. Debian docs, disponível em: https://www.debian.org/doc. Acesso em: 06 out. 2023.

Debian APT. Apt, disponível em: https://wiki.debian.org/pt_BR/Apt. Acesso em: 28 out. 2023.

Debian Users. Quem está usando o Debian. Disponível em: https://www.debian.org/users/index.pt.html. Acesso em: 28 out. 2023.

Debian GNU/Hurd. Debian GNU/Hurd, disponível em: https://www.debian.org/ports/hurd/. Acesso em: 28 out. 2023.

Debian GNU/kFreeBSD. Debian GNU/kFreeBSD, disponível em: https://www.debian.org/ports/kfreebsd-gnu/. Acesso em: 28 out. 2023.

DE OLIVEIRA JÚNIOR, Javan; OYAMADA, Marcio. Avaliando o impacto da compressão de dados no desempenho e energia em redes LoRa. In: ARTIGOS COMPLETOS - SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SISTEMAS COMPUTACIONAIS (SBESC), 10., 2020, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020. p. 81-88. ISSN 2763-9002. DOI: https://doi.org/10.5753/sbesc_estendido.2020.13094.

Deutsch, P. GZIP File Format Specification Version 4.3, RFC 1952, May 1996. disponível em: https://www.ietf.org/rfc/rfc1951.txt. Acesso em: 06 out. 2023.

FAN, X.; WEBER, W.-D.; BARROSO, L. A. Power provisioning for a warehouse-sized computer. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, 2007. p. 13–23.

GNU Hurd. What is the GNU Hurd, disponível em https://www.gnu.org/software/hurd/. Acesso em: 28 out. 2023.

GNU. O que é software livre, disponível em: https://www.gnu.org/philosophy/free-sw.pt-br.html. Acesso em 28 out. 2023.

FRANCO, Márcia H I.; MACHADO, Rodrigo P.; BERTAGNOLLI, Silvia C. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java. (Tekne). Grupo A, 2016. E-book. ISBN 9788582603710. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788582603710/. Acesso em: 21 out. 2023.

Gzip. Gzip docs, disponível em: https://www.gnu.org/software/gzip/manual/gzip.html. Acesso em: 06 out. 2023.

IF UFRGS. Descritores padrão de arquivos, disponível em: https://www.if.ufrgs.br/~leon/Livro_3_ed/node35.html. Acesso em: 18 nov. 2023.

IBM, 2021. How sockets works, disponível em: https://www.ibm.com/docs/en/i/7.1?topic=programming-how-sockets-work. Acesso em: 24 our. 2023.

IBM, 2023. Transferindo dados de arquivo usando APIs send_file() e accept_and_recv(), disponível em: https://www.ibm.com/docs/pt-br/i/7.5?topic=esad-examples-transferring-file-data-using-send-file-accept-recv-apis. Acesso em: 24 out. 2023.

KERRISK, Michael. Chapter 4: File I/O: The Universal I/O Model. In: The Linux Programming Interface. 2010. Disponível em: https://man7.org/tlpi/. Acesso em: 29 out. 2023.

MDN. Qual a diferença entre página web, site, servidor web e mecanismo de busca?. [S. l.], Out. 2023. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/Web_mechanics/Pages_sites_servers_and_search_engines. Acesso em: 16 out. 2023.

MDN. Overview of HTTP: A typical HTTP session. [S. l.], Ago. 2023. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acesso em: 14 out. 2023.

Moser Badalotti, G. 2014 Introdução ao desenvolvimento de Sistemas Web. [s.l.] Uniasselvi, 2015. p. 58

NELSON, Rick. Tuning NGINX for Performance. 10 out. 2014. Disponível em: https://www.nginx.com/blog/tuning-nginx. Acesso em: 01 out. 2023.

Netcraft. Web Server Survey – July 2017, disponível em: https://www.netcraft.com/blog/july-2017-web-server-survey/. Acesso em: 22 out. 2023.

Netcraft. Web Server Survey – January 2023, disponível em: https://www.netcraft.com/blog/january-2023-web-server-survey/. Acesso em: 22 out. 2023.

Nginx. Nginx docs, disponível em: https://nginx.org/en/docs. Acesso em: 06 out. 2023.

Nginx. Nginx glossary, disponível em: https://www.nginx.com/resources/glossary/nginx/. Acesso em: 26 out. 2023.

NGUYEN, Van Nam. Comparative Performance Evaluation of Web Servers. VNU Journal of Science: Comp. Science & Com. Eng., vol. 31, no. 3, p. 28–34, 2017. DOI: 10.25073/2588-1086/vnucsce.2932. Disponível em: http://eprints.uet.vnu.edu.vn/eprints/id/eprint/2781/1/TR2017-FIT-004.pdf. Acesso em: 20 out. 2023.

Pidstat. Linux manual page, disponível em: https://man7.org/linux/man-pages/man1/pidstat.1.html. Acesso em: 06 out. 2023.

RODRIGUES, R. A. B. Métricas e ferramentas livres para análise de capacidade em servidores linux. 2009. 67 p. Monografia (Especialização em Administração de Redes Linux) - Universidade Federal de Lavras, Lavras, 2009. Disponível em: http://repositorio.ufla.br/jspui/handle/1/5596. Acesso em: 26 out. 2023.

Sendfile. Linux manual page, disponível em: https://man7.org/linux/man-pages/man2/sendfile.2.html. Acesso em: 06 out. 2023.

SILVA, Marcelo José Santos da. Uma abordagem para avaliação de desempenho de serviços web. 2015. 85 f. Dissertação (Mestrado) - Curso de Pós-Graduação em

Informática Aplicada, Departamento de Informática, Universidade Federal Rural de Pernambuco, Recife, 2015.

SEGELLMANN, Robin; TUXEN, Michael; RATHGEB, Erwin P. SSH Over SCTP — Optimizing a Multi-Channel Protocol by Adapting It to SCTP. Munster University of Applied Sciences, Stegerwaldstrasse 39, 48565 Steinfurt, Germany. Disponível em: https://ieeexplore.ieee.org/document/6292659. Acesso em: 03 nov. 2023.

TOMIŠA, Mario; MILKOVIĆ, Marin; ČAČIĆ, Marko. Performance Evaluation of Dynamic and Static WordPress-based Websites. University North, Koprivnica, Croatia, Europe, 2023. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8974709. Acesso em: 03 nov. 2023.

WANG, Jundi; KAI, Zhao. Performance Analysis and Optimization of Nginx-based Web Server. Journal Of Physics: Conference Series, [S.L.], v. 1955, n. 1, p. 012033, 1 jun. 2021. IOP Publishing. http://dx.doi.org/10.1088/1742-6596/1955/1/012033. Disponível em: https://iopscience.iop.org/article/10.1088/1742-6596/1955/1/012033. Acesso em: 26 out. 2023.