Trabalho Referente a Segunda Nota da Disciplina de Redes de Computadores 2

Rayssa dos Santos Alves[1]

[1 1] Sistemas de Informação – Universidade Federal do Piauí (UFPI),
Campus Senador Helvídio Nunes de Barros (CSHNB),

rayssa.alves@ufpi.edu.br

Resumo. *Este trabalho tem como objetivo configurar, testar e comparar o desempenho de dois servidores web muito utilizados: Nginx e Apache HTTP Server, em ambiente de testes totalmente containerizado com Docker. O cenário de testes foi estruturado com multiplos serviços que simulam os servidores em rede e cliente, os quais são organizados por meio de Docker Compose, possibilitando a definição de IPs e portas personalizados. O objetivo principal é avaliar o desempenho, a eficiência e o comportamento dos servidores web em diferentes cenários de carga, utilizando um stack de observalidade composta por Prometheus e Grafana. Todos os testes foram conduzidos por meio de scripts em python, que são responsáveis por gerar os testes de carga, coletar as métricas e a geração de gráficos comparativos. A partir dos resultados obtidos, serão discutidas as vantagens e limitações de cada servidor.*

1. Introdução

Este trabalho apresenta uma análise comparativa de desempenho entre dois servidores web: Nginx e Apache. O projeto foi implementado utilizando o Docker para conteinerizar, com Docker Compose para coordenar todos os serviços, garantindo que os testes possam ser replicados. A personalização da rede foi baseada nos números da matrícula *20239019558*.

O objetivo principal deste trabalho é realizar uma avaliação de desempenho e comportamental acerca de diferentes servidores web, sob diferentes cenários de carga, que variam o número de usuários (*Threads*) e o tipo de conteúdo (arquivos estáticos de tamanho pequeno, médio e grande). Foi utilizada a stack de observabilidade composta pelo Prometheus para coletar as métricas e pelo Grafana para visualizar as estatísticas dos resultados.

As próximas seções apresentam a metodologia adotada no projeto, os resultados obtidos com os testes e a conclusão sobre o desempenho dos servidores.

2. Metodologia

Esta seção descreve o processo de implementação dos servidores web, a arquitetura do ambiente de teste e a stack de observabilidade adotada para a avaliação comparativa de desempenho. O ambiente foi construido seguindo uma estrutura modular apresentada na Figura 1 abaixo:

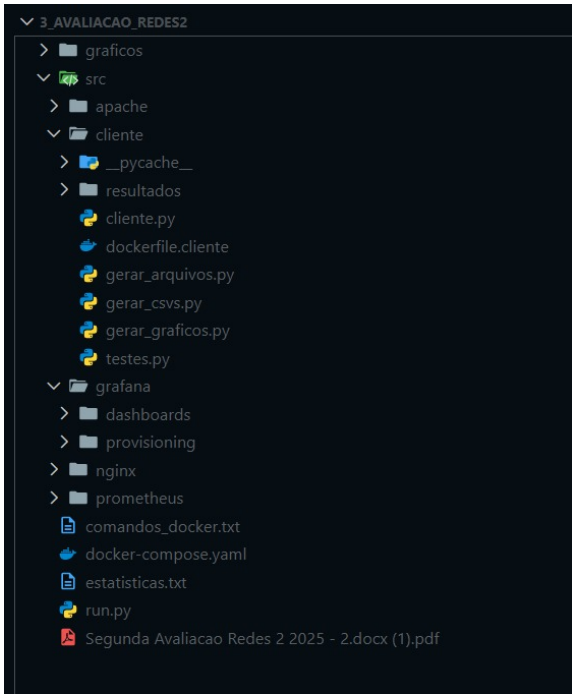

Figure 1. Estrutura do projeto

2.1. Arquitetura do ambiente e Containerização

Para garantir que o ambiente fosse padronizado, isolado e replicável, todo o sistema foi implementado em contêiners, utilizando o Docker e coordenando-se através do Docker Compose. Para a geração da subrede, foi utilizado os quatro últimos dígitos da matrícula do aluno. A tabela 2.1 abaixo mostra os serviços e os seus respctivos endereços IPs:

Table 1. Endereços IP dos containers configurados no ambiente de teste

Serviço	Container	Endereço IP
Nginx	nginx	95.58.0.2
Nginx Exporter	nginx_exporter	95.58.0.3
Apache	apache_server	95.58.0.4
Apache Exporter	apache_exporter	95.58.0.5
Prometheus	prometheus	95.58.0.6
Grafana	grafana	95.58.0.7
Cliente	cliente	95.58.0.8

2.2. Configuração dos Servidores Web

Os dois servidores web, Nginx e Apache HTTP Server, foram configurados para atender ao mesmo conjunto de recursos, garantindo a validade e a justiça da comparação de desempenho. Ambos foram configurados para servir a coleção idêntica de arquivos estáticos (pequeno, médio e grande) e possuíam um endpoint de status de API.

2.3. Servidor Nginx (Servidor Web 1)

O Nginx foi escolhido por ser um dos servidores mais utilizados globalmente e segue um modelo orientado a eventos (ou assíncrono). Essa arquitetura é caracterizada por

utilizar poucos processos de trabalho que tratam múltiplas conexões simultâneas de forma não bloqueante, o que o torna altamente eficiente e com baixo consumo de memória em cenários de alta concorrência.

Sua configuração no ambiente containerizado inclui:

- **Escuta de Serviço**: O servidor escuta na porta 80 interna do contêiner, mapeada externamente para a porta 8080 no host de testes.
- **Observabilidade**
  - **node-exporter**: Integrado ao container do nginx para coletar as métricas de CPU e Memória do container, expondo-as na porta 9100.
  - **nginx-exporter**: Responsável por coletar métricas detalhadas (como conexões ativas e requisições processadas) e expô-las ao Prometheus na porta 9113.

2.3.1. Servidor Apache HTTP Server (Servidor Web 2)

O Apache HTTP Server foi o segundo servidor escolhido para contrastar com o Nginx. Ele utiliza um modelo baseado em processos ou threads. A configuração do Apache foi ajustada para espelhar as mesmas funcionalidades e endpoints do Nginx:

- **Escuta de Serviço**: O servidor escuta na porta 80 interna do contêiner (assumindo um mapeamento externo para a porta 8081 no host).
- **Observabilidade**
  - **node-exporter**: Integrado ao contêiner do apache para coletar as métricas de CPU e Memória do contêiner, expondo-as na porta 9100.
  - **apache-exporter**: Integrado para coletar métricas detalhadas (como conexões ativas e requisições processadas) e expô-las ao Prometheus na porta 9117

2.4. Stack de Observalidade

A avaliação do desempenho foi realizada por meio da integração dos servidores com uma stack de observabilidade, um requisito fundamental do projeto que garante a coleta de dados de forma contínua e replicável. A stack é composta pelo Prometheus para a coleta e armazenamento de métricas e pelo Grafana para a visualização e análise dos dados.

2.4.1. Prometheus (Coleta de Métricas)

O Prometheus realiza a coleta de dados do ambiente. Seu arquivo de configuração (prometheus.yml) definiu os seguintes jobs de scrape:

- **prometheus**: Coleta as métricas da própria instancia do Prometheus.
- **nginx e apache**: Coleta as métricas específicas dos servidores web por meio dos seus exporters, os quais são expostos nas portas 9113 e 9117 respectivamente.
- **nginx_node e apache_node**: Coleta as métricas dos contêiners (CPU, Memória) por meio do node-exporter.

Essa configuração garantiu que todas as métricas pudessem ser rastreadas ao longo do tempo durante os testes de carga, permitindo uma análise precisa do comportamento dinâmico de cada servidor.

2.4.2. Grafana (Visualização e Análise)

O Grafana foi configurado para se conectar à fonte de dados do Prometheus e fornecer a interface visual necessária para a comparação estatística e gráfica entre os servidores Nginx e Apache. Foram desenvolvidos dashboards personalizados que apresentam os resultados lado a lado. A utilização do Grafana facilitou a análise visual dos teste, complementando as estatísticas calculadas pelos scripts em Python com evidências gráficas.

2.5. Teste de Carga

Os testes de carga foram desenvolvidos em Python, utilizando bibliotecas como concurrent.futures.ThreadPoolExecutor para gerar a concorrência (múltiplas threads) e requests para realizar as requisições HTTP.

2.5.1. Cliente

O cliente foi implementado em Python para simular usuários reais acessando os servidores Nginx e Apache. Ele é responsável por montar e enviar as requisições HTTP via sockets, incluindo o cabeçalho personalizado X-Custom-ID, além de medir o tempo de resposta.

As funções do cliente são:

- **gerar_hash**: Gera um hash SHA1 fixo da matricula + nome do aluno, usado como identificador personalizado nas requisições.
- **dividir_requisicao**: Analisa a requisição recebida e faz a separação do método, caminho e os cabeçalhos.
- **enviar_requisicao**: Recebe metodo, caminho e corpo como paramêtros, monta e envia uma requisição HTTP via socket, retornando o status, tempo e cabeçalhos.

2.5.2. Testes

Os testes de carga foram organizados em diferentes cenários, apresentados na Tabela 2, variando tanto o tamanho dos arquivos quanto o número de threads concorrentes. Cada cenário foi executado 20 vezes, além disso, 200 requisições foram enviadas em cada execução.

As funções criadas para o script de testes foram:

- **consultar_prometheus**: Envia uma requisição HTTP ao Prometheus e retorna os resultados da consulta em JSON.
- **coletar_cpu**: Monta a query PromQL para o Prometheus a fim de calcular o uso de CPU do contêiner (percentual ocupado).
- **coletar_memoria**: Monta a query PromQL para o Prometheus a fim de calcular o uso de Memória do contêiner (percentual utilizado).
- **calcular_metricas_prometheus**: Converte os valores de CPU e memória do servidor para float e retorna.
- **executar_requisicao**: Cria um cliente socket e chama a função de enviar uma requisição HTTP GET ao servidor, retornando um dicionário com status, tempo e sucesso da operação.
- **stress**: Executa requisições simultâneas para os testes, mede latência, RPS, sucessos/falhas e coleta métricas de CPU/memória.

Table 2. Cenários de Teste de Carga

Tamanho do arquivo	Número de Threads
/arquivo_10kb.txt	5
/arquivo_10kb.txt	10
/arquivo_10kb.txt	15
/arquivo_1mb.txt	5
/arquivo_1mb.txt	10
/arquivo_1mb.txt	15
/arquivo_10mb.txt	5
/arquivo_10mb.txt	10
/arquivo_10mb.txt	15

2.6. Metricas escolhidas

Esta seção apresenta as principais Métricas de Desempenho Escolhidas para realizar as análises de desempenho:

- **Requisições por Segundo**: (Variável rps) Métrica principal que mede as Requisições por Segundo. É o indicador da capacidade máxima de processamento de cada servidor sob concorrência.
- **Latência de Resposta**: (Variáveis latencia_media, latencia_min, latencia_max) Mede a velocidade de resposta. A coleta da Latência Média, Mínima e Máxima é crucial para o rigor estatístico, mostrando a variação e estabilidade da resposta.
- **Uso de CPU**: (Variável cpu) Coletado via integração com o Prometheus. É uma métrica de eficiência que compara qual servidor (Nginx ou Apache) utiliza melhor os recursos de processamento para a carga aplicada.
- **Uso de Memória**: (Variável memoria) Coletado via integração com o Prometheus. Também é uma métrica de eficiência que avalia a sobrecarga de memória imposta por cada arquitetura de servidor durante a concorrência.
- **Confiabilidade**: (Variáveis de sucesso e erros) Conta o número de requisições concluídas com sucesso e as que falharam. Confirma a estabilidade dos servidores sob estresse, garantindo que os resultados de Vazão sejam válidos.

2.7. Arquivos auxiliares

Para auxiliar a geração dos arquivos estáticos, os gráficos e os csvs, foram criado arquivos auxiliares com funções relacionadas a cada gráfico, csv e arquivo estático.

- **gerar_arquivos**: Este arquivo é responsável pela geração dos arquivos estáticos em TXT para os testes de carga.
- **gerar_csvs**: Este arquivo é responsável por gerar os csvs de todas as execuções e o csv das médias de cada cenário. Além disso, há uma função para o calculo das médias de cada cenário.
- **gerar_graficos**: Este arquivo realiza a geração de todos os gráficos utilizados para a análise dos resultados para esse relatório.

3. Resultados

Esta seção apresenta os resultados obtidos para a análise comparativa entre os dois servidores web: Nginx e Apache.

3.1. Uso de memória

A Figura 2 apresenta o consumo de memória dos servidores web Nginx e Apache. O gráfico revelou que ambos os servidores possuem um padrão de utilização muito parecidos e estáveis, variando entre 14% e 18%, mesmo que os cenários possuam arquivos de tamanhos ou número de threads diferentes. Em aproximadamente 50% dos cenários, o servidor Nginx possui um leve maior consumo de memória, mas a diferença é muito pequena e não apresenta grande impacto. A estabilidade mostra que o consumo de memória não muda de forma significativa com o aumento da carga, indicando que os dois servidores possuem um bom gerenciamento de memória e são pouco sensíveis ao tamanho do arquivo ou o aumento da concorrência.

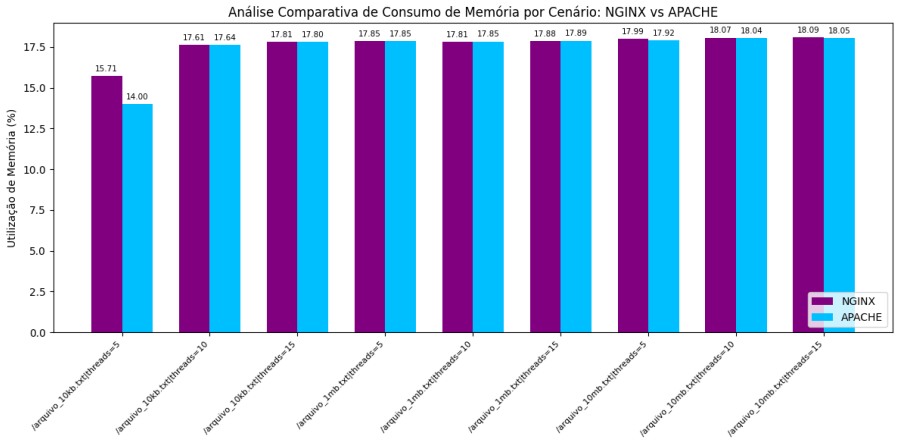

Figure 2. Gráfico do uso de memória

3.2. Uso da CPU

A Figura 3 apresenta o consumo da CPU dos servidores web Nginx e Apache. O gráfico mostra que, para arquivos pequenos, como os de 10 kb, o consumo de CPU aumenta à medida que o número de threads aumentam, atingindo picos acima de 80% para ambos os servidores. À medida que vão aumentando os tamanhos dos arquivos e o número de threads aumentam, o gasto da CPU vai ficando mais equilibrado e há uma redução no seu consumo. De modo geral, o Apache tende a consumir um pouco mais de CPU em praticamente todos os cenários.

3.3. Latência média

A Figura 4 apresenta a Latência dos servidores web Nginx e Apache. O gráfico mostra que o Apache apresenta valores de latência menores que o Nginx na maioria dos casos, com diferênças maiores em cenários com arquivos maiores. O Nginx apresenta as latências médias mais altas para cenários com maior números de threads. Em arquivos maiores, a diferença também se mantém, mostrando que o impacto do tamanho do arquivo afeta ambos, mas com uma vantagem contínua para o NGINX.

3.4. Taxa de Requisições por Segundo

A Figura 5 apresenta a Taxa de Requisições por segundo dos servidores web Nginx e Apache. O gráfico mostra como o apache é superior na maioria dos cenários, principalmente nos testes com arquivos maiores, ultrapassando os 1500 RPS. O Ngix se mantêm

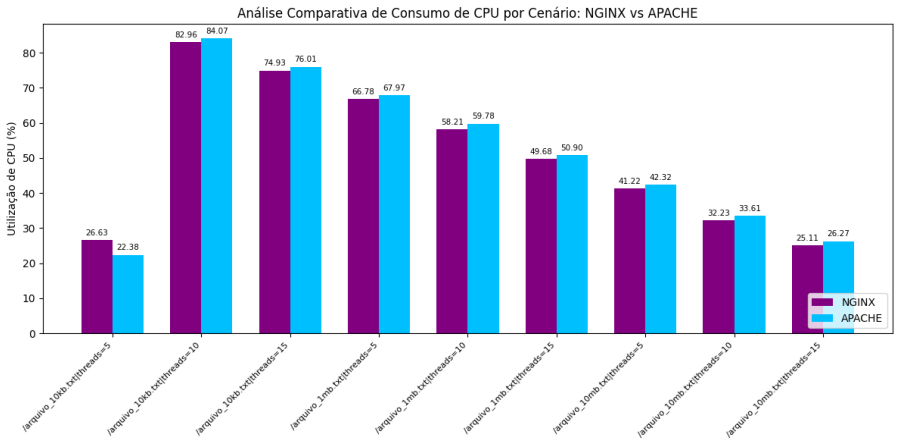

Figure 3. Gráfico do Uso da CPU

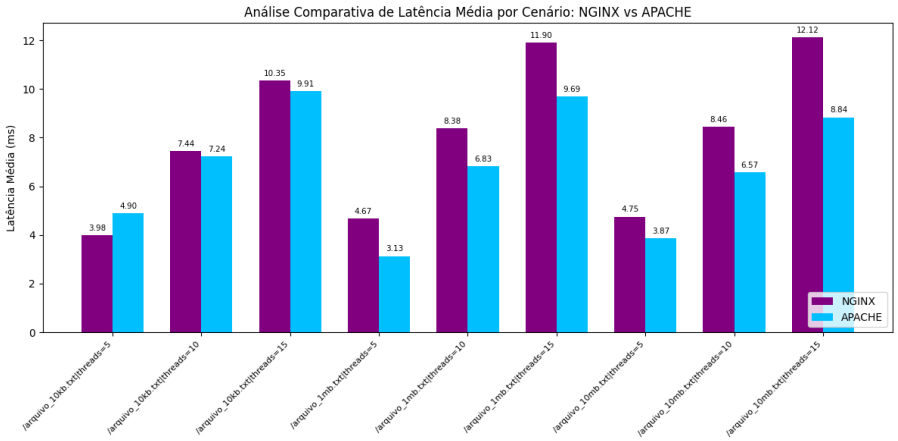

Figure 4. Gráfico da latência média

constante, mas com menor desempenho para arquivos maiores. Porém, para o arquivo menor, o Nginx possui um RPS próximo ao do Apache nas execuções com menos threads, sugerindo que ele é mais eficiente para cargas leves.

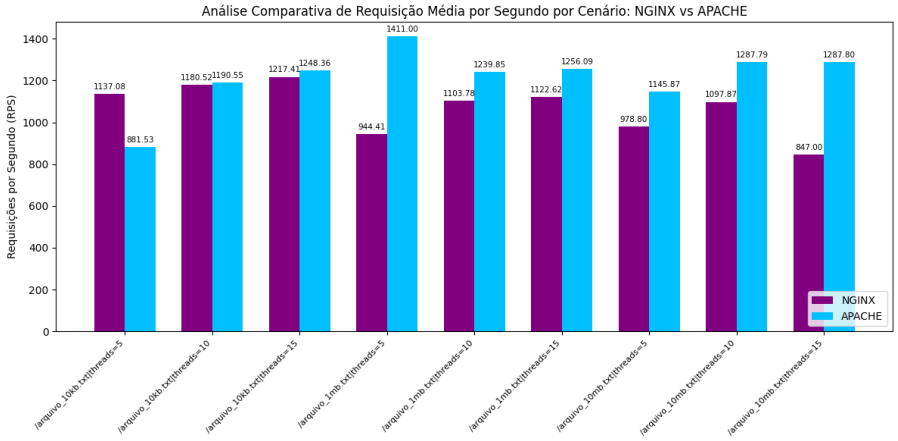

Figure 5. Gráfico da taxa de Requisições por Segundo

3.5. Tempo Total de Resposta

A Figura 6 apresenta o tempo total de resposta dos servidores Nginx e Apache. O gráfico demonstra como o Nginx apresenta tempos mais altosos, em comparação com o Apache, com exceção dos cenários do arquivo de 10 kb com 5 threads e arquivo de 1 Mb com 5 threads. Em geral, o Nginx possui um tempo de resposta maior na maioria dos cenários, sendo que no último, o Nginx teve um tempo maior que o dobro do Apache. Isso confirma que, embora o Apache entregue maior RPS em alguns casos, o custo é um aumento perceptível no tempo de resposta.

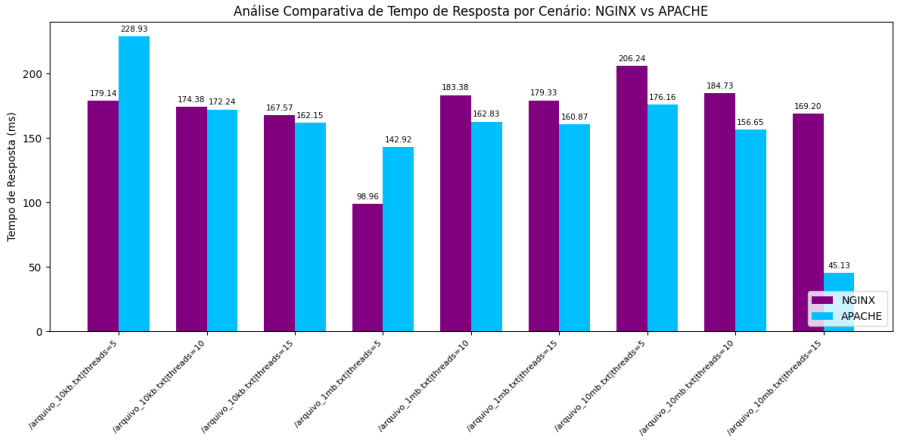

Figure 6. Gráfico do tempo total

3.6. Comparação das médias

A Tabela 3 apresenta uma comparação geral do desempenho entre os servidores web Nginx e Apache.

Cenário (Servidor)	CPU (%)	Memória (MB)	Lat. Média (ms)	RPS
Apache — /arquivo_10kb.txt — threads=5	22.38	13.99	4.90	881.53
Apache — /arquivo_10kb.txt — threads=10	84.07	17.64	7.24	1190.55
Apache — /arquivo_10kb.txt — threads=15	76.01	17.80	9.91	1248.36
Apache — /arquivo_1mb.txt — threads=5	67.96	17.85	3.13	1410.99
Apache — /arquivo_1mb.txt — threads=10	59.78	17.84	6.83	1239.85
Apache — /arquivo_1mb.txt — threads=15	50.89	17.89	9.69	1256.09
Apache — /arquivo_10mb.txt — threads=5	42.32	17.92	3.87	1145.87
Apache — /arquivo_10mb.txt — threads=10	33.60	18.03	6.57	1287.79
Apache — /arquivo_10mb.txt — threads=15	26.26	18.04	8.84	1287.79
NGINX — /arquivo_10kb.txt — threads=5	26.63	15.71	3.98	1137.07
NGINX — /arquivo_10kb.txt — threads=10	82.95	17.61	7.44	1180.52
NGINX — /arquivo_10kb.txt — threads=15	74.93	17.80	10.35	1217.40
NGINX — /arquivo_1mb.txt — threads=5	66.78	17.84	4.67	944.41
NGINX — /arquivo_1mb.txt — threads=10	58.20	17.80	8.38	1103.77
NGINX — /arquivo_1mb.txt — threads=15	49.68	17.88	11.90	1122.62
NGINX — /arquivo_10mb.txt — threads=5	41.22	17.98	4.75	978.79
NGINX — /arquivo_10mb.txt — threads=10	32.22	18.07	8.46	1097.87
NGINX — /arquivo_10mb.txt — threads=15	25.10	18.08	12.12	847.00

Table 3. Resultados simplificados dos testes de desempenho: CPU, memória, latência média e RPS, separados por servidor.

Os resultados mostram que tanto o Apache quanto o Nginx possuem uma taxa de sucesso de 100%. O Apache se destaca por oferecer latências menores na maioria

dos cenários, indicando respostas mais rápidas, embora utilize mais CPU em alta concorrência. O Nginx registra a latência mais altas, mas apresenta uma evolução mais estável com o aumento do número de threads. Nas Requisições por segundo, o Nginx ganha em cargas menores e o Apache é melhor em cenários com arquivos maiores. O uso de memória é estável para os dois servidores.

4. Conclusão

A análise comparativa entre os servidores web Nginx e Apache demonstrou desempenho estável em relação ao uso de recursos, especialmente no consumo de memória, que se manteve estável em quase todos os cenários. Em termos de CPU, a variação foi mais perceptível conforme o aumento da carga, sobretudo com arquivos pequenos e alta concorrência. Embora ambos os servidores tenham apresentado picos elevados, o Apache demonstrou um consumo levemente maior de CPU na maioria dos cenários. Esse comportamento sugere que, apesar de entregar bom desempenho, o Apache tende a exigir mais processamento, especialmente em cargas intensas.

Já nos indicadores de desempenho direto — latência, requisições por segundo e tempo total de resposta — observa-se comportamentos distintos entre os servidores. O Apache se destacou por apresentar menor latência e maior taxa de requisições por segundo, principalmente em arquivos maiores, evidenciando melhor desempenho nesses cenários. Por outro lado, o Nginx apresentou tempos totais de resposta maiores na maioria dos cenários, mesmo sendo mais eficiente em cargas leves. Assim, conclui-se que o Apache tende a ser mais vantajoso em cenários de alta demanda, enquanto o Nginx pode oferecer benefícios em ambientes enxutos ou com cargas mais leves.

Além disso, a utilização da stack de observalidade comporsta por Prometheus e Grafana mostrou-se adequada para este tipo de análise, oferecendo coleta contínua e visualização clara das métricas. Entre as vantagens, destacam-se a facilidade de integração com os exporters, para a coleta das métricas e a flexibilidade na criação dos dashboards. Entretanto, essa abordagem apresenta limitações, como a necessidade de configuração manual para a persistencia dos dashboards. Ainda assim, a combinação das duas ferramentas ofereceu visibilidade completa do ambiente e permitiu análises confiáveis para a comparação entre Nginx e Apache.

5. Anexos

Abaixo estão os links relacionados ao trabalho, incluindo o vídeo no YouTube e o repositório no GitHub.

- Link do YouTube: https://youtu.be/
- Link do GitHub: https://github.com/rayss4lves/3_avaliacao_redes2.git