Trabalho 3 da matéria de Redes de Computadores II

Hermeson A. De Oliveira[1]

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

2

hermeson.de@ufpi.edu.br

Resumo. *Este relatório apresenta o processo de configuração, execução e análise de dois servidores web de mercado, Nginx e Apache, em um ambiente de testes baseado em contêineres Docker. O objetivo principal é comparar o desempenho, a eficiência e o comportamento de ambas as abordagens em diferentes cenários de carga, com apoio de uma stack de observabilidade composta por Prometheus e Grafana. Todos os testes foram coordenados por scripts em Python, que automatizam a geração de carga, a coleta de métricas e a produção de gráficos comparativos. Com base nos resultados obtidos, são discutidas as vantagens e limitações de cada servidor, bem como os cenários em que Nginx ou Apache apresentam melhor desempenho.*

1. Introdução

Este relatório descreve o desenvolvimento de um ambiente de testes para comparação de desempenho entre os servidores web Nginx e Apache, utilizando contêineres Docker, Prometheus, Grafana e um cliente de carga implementado em Python. O foco está na análise de métricas de latência, vazão, taxa de sucesso e comportamento em diferentes tamanhos de resposta e níveis de concorrência.

As seções seguintes apresentam a arquitetura do ambiente, a metodologia de testes, as métricas utilizadas, os resultados obtidos e as conclusões sobre as diferenças entre Nginx e Apache.

2. Metodologia

Esta seção descreve a arquitetura em contêineres, a configuração dos servidores Nginx e Apache, a integração com Prometheus e Grafana e o cliente de carga responsável por executar os cenários de teste.

2.1. Arquitetura em Docker

O ambiente de testes foi implementado em uma subrede Docker baseada nos quatro últimos dígitos da matrícula do aluno, resultando na rede `53.82.0.0/24`. Cada serviço foi executado em um contêiner dedicado, com endereços IP e portas definidos conforme a Tabela 1.

O arquivo `docker-compose.yml` define os serviços, a rede `53.82.0.0/24` e os volumes necessários. Nginx e Apache expõem serviços HTTP que atendem arquivos

Serviço	IP	Porta de acesso
Nginx	53.82.0.10	8080
Apache	53.82.0.20	8081
Prometheus	53.82.0.30	9090
Grafana	53.82.0.40	3000

Table 1. Arquitetura de serviços na subrede Docker.

estáticos de diferentes tamanhos e um endpoint de API. Prometheus monitora o ambiente por meio de *exporters* configurados, enquanto Grafana apresenta *dashboards* com gráficos de desempenho.

2.2. Servidores Nginx e Apache

Os dois servidores web foram configurados para atender o mesmo conjunto de recursos, garantindo uma comparação justa. Ambos servem arquivos estáticos e um endpoint de API de status, além de validar um cabeçalho personalizado de identificação.

No Nginx, a configuração principal inclui:

- Escuta na porta 80 dentro do contêiner, mapeada externamente para a porta 8080.
- Servidor de arquivos estáticos (`small.txt`, `medium.txt`, `large.txt`, `xlarge.txt` e `xxlarge.txt`).
- Endpoint `/api/status` com resposta simples em formato JSON.
- Registro de logs de acesso em formato padronizado.

O Nginx segue um modelo orientado a eventos, com poucos processos de trabalho que tratam múltiplas conexões simultâneas, favorecendo cenários com alta concorrência.

No Apache, a configuração foi ajustada para espelhar as mesmas funcionalidades:

- Escuta na porta 80 dentro do contêiner, mapeada para a porta 8081.
- Servidor dos mesmos arquivos estáticos usados no Nginx.
- Endpoint `/api/status` com conteúdo equivalente ao do Nginx.
- Módulos necessários para atender requisições HTTP e produzir logs.

O Apache utiliza um modelo baseado em processos ou *threads*, dependendo do *Multi-Processing Module* configurado, o que impacta o consumo de recursos em cenários com muitas conexões simultâneas.

2.3. Cliente de Carga em Python

O cliente responsável pelos testes de carga foi desenvolvido em Python e executa todas as requisições de forma automatizada. Não foram usados scripts *shell*, conforme restrição da avaliação.

O script `main.py` coordena todo o fluxo:

- *Build* e inicialização dos contêineres via Docker e `docker-compose`.
- Execução dos cenários de teste definidos.
- Coleta e armazenamento das métricas em arquivos estruturados.
- Geração de gráficos comparativos.
- Encerramento e limpeza dos contêineres ao final da execução.

O cliente executa requisições HTTP para Nginx e Apache.

2.4. Stack de Observabilidade com Prometheus e Grafana

A stack de observabilidade foi montada com Prometheus para coleta de métricas e Grafana para visualização. O Prometheus foi configurado para:

- Coletar métricas de Nginx e Apache por meio de *exporters* ou endpoints de métricas.
- Registrar dados de séries temporais de latência, vazão e códigos de resposta HTTP.

O Grafana foi configurado com *dashboards* que apresentam:

- Gráficos de linha para latência ao longo do tempo.
- Gráficos de barras para comparação de vazão entre Nginx e Apache.
- Painéis de taxa de sucesso e distribuição de respostas.

Esses painéis permitem uma análise visual dos cenários de teste, complementando as estatísticas calculadas pelos scripts em Python.

2.5. Execução dos Testes

Os testes podem ser executados de duas formas: automática ou manual. No modo automático, o usuário executa:

```
python3 main.py
```

Esse comando realiza toda a sequência:

- *Build* e *start* dos contêineres.
- Execução dos 10 cenários de teste, totalizando 740 requisições.
- Análise comparativa de desempenho.
- Geração de 8 gráficos (5 de barras e 3 de linhas).
- *Cleanup* dos contêineres ao final.

No modo manual, as etapas são divididas:

```
docker-compose up -d
docker exec load_client python3 /app/load_test.py
docker exec load_client python3 /app/analise_resultados.py
docker exec load_client python3 /app/gerar_graficos.py
docker-compose down -v
```

Esse fluxo permite repetir testes específicos ou inspecionar os serviços durante a execução.

3. Métricas

Esta seção descreve as métricas utilizadas para avaliar o desempenho dos servidores Nginx e Apache, bem como o processo de coleta e análise dos dados.

3.1. Cenários de Teste

Os testes foram organizados em 10 cenários, variando o tamanho dos arquivos, o número de requisições e o nível de concorrência. A Tabela 2 resume as principais características.

O total de requisições nos 10 cenários é de 740, distribuídas igualmente entre os dois servidores, garantindo condições comparáveis.

#	Descrição	Requisições	Threads	Recurso
1	Pequeno - Sequencial	50	1	`small.txt` (60B)
2	Médio - Sequencial	50	1	`medium.txt` (10KB)
3	Grande - Sequencial	30	1	`large.txt` (1MB)
4	Pequeno - Concorrente	100	10	`small.txt`
5	Médio - Concorrente	100	10	`medium.txt`
6	Grande - Concorrente	50	10	`large.txt`
7	Extra Grande - Sequencial	20	1	`xlarge.txt` (10MB)
8	Extra Grande - Concorrente	30	5	`xlarge.txt`
9	XXL - Sequencial	10	1	`xxlarge.txt` (50MB)
10	API Status - Alta Concorrência	200	20	`/api/status`

Table 2. Cenários de teste para Nginx e Apache.

3.2. Métricas Utilizadas

Para a comparação entre Nginx e Apache, foram aplicadas as seguintes métricas de desempenho:

- **Latência (ms):** tempo entre o envio da requisição e o recebimento da resposta completa do servidor, incluindo média, mediana, valores mínimo e máximo.
- **Desvio padrão da latência:** variação observada nos tempos de resposta, indicando estabilidade ou oscilação do servidor.
- **Vazão (req/s):** quantidade de requisições processadas por segundo, indicando a capacidade de atendimento sob carga.
- **Taxa de sucesso (%):** proporção de requisições com resposta `2xx` em relação ao total enviado.
- **Tamanho da resposta (bytes):** tamanho do conteúdo retornado, importante para validar a consistência entre os servidores.

Essas métricas refletem de forma direta o comportamento dos servidores web em cenários com diferentes níveis de concorrência e tamanhos de conteúdo.

3.3. Coleta de Dados

A coleta de dados foi realizada por meio do cliente em Python e da integração com Prometheus. O script de carga registra:

- Tempo de início e fim de cada cenário.
- Tempo individual de cada requisição.
- Códigos de status HTTP e tamanho das respostas.

Os dados são armazenados em arquivos estruturados, como CSV, separados para Nginx e Apache. Em paralelo, o Prometheus coleta métricas de infraestrutura e de aplicação que podem ser exibidas em tempo real nos *dashboards* do Grafana.

3.4. Geração de Gráficos e Estatísticas

O script `analise_resultados.py` consolida os dados e calcula estatísticas como média, mediana, desvio padrão, mínimos e máximos de latência e vazão. Em seguida, o script `gerar_graficos.py` produz gráficos comparativos, entre eles:

- Gráficos de barras comparando a vazão de Nginx e Apache por cenário.
- Gráficos de barras comparando a latência média por cenário.
- Gráficos de linha mostrando a evolução da latência ao longo das requisições.

Um relatório estatístico é gerado com as principais métricas e percentuais de diferença entre os dois servidores em cada cenário.

4. Resultados

Nesta seção são apresentados os resultados experimentais obtidos, ilustrados por meio de gráficos gerados a partir das métricas coletadas dos servidores Nginx e Apache.

4.1. Vazão ao longo dos cenários

Os gráficos de vazão mostram que, em cenários com alta concorrência e arquivos pequenos, como o cenário 4 e principalmente o cenário 10 (`/api/status`), mostrados na Figura 1, o Nginx tende a apresentar maior quantidade de requisições processadas por segundo. Esse comportamento está alinhado com o modelo orientado a eventos, que reduz a sobrecarga na criação de processos ou *threads*.

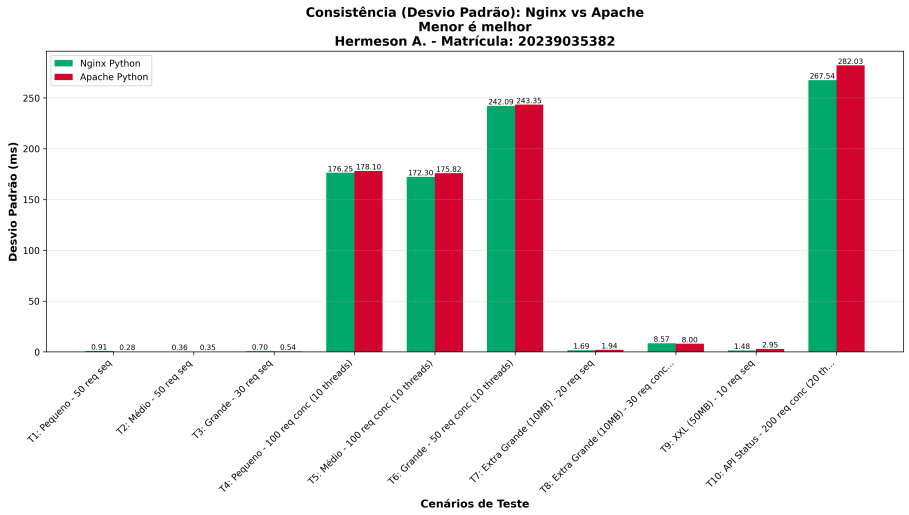

Figure 1. Comparação de vazão e desvio padrão ao longo dos cenários. Fonte: autor.

Em cenários sequenciais ou com baixa concorrência, como os cenários 1, 2 e 3, a diferença de vazão entre Nginx e Apache se torna menor. Em alguns casos, os valores são próximos, mostrando que, com poucas conexões ativas, ambos os servidores lidam bem com a carga.

4.2. Comparação de tempo de resposta

Os gráficos de tempo médio de resposta indicam que o Nginx apresenta latências menores em cenários com muitas requisições concorrentes, em especial para arquivos pequenos e para o endpoint de status, como mostrado nas Figuras 2 e 3. A redução da latência é mais evidente quando o número de *threads* no cliente aumenta.

O Apache mantém tempos de resposta competitivos em cenários sequenciais e em testes com arquivos maiores, nos quais o tempo de transferência de dados domina o tempo total da requisição. Nesses casos, o custo de gestão de processos ou *threads* tem impacto menor.

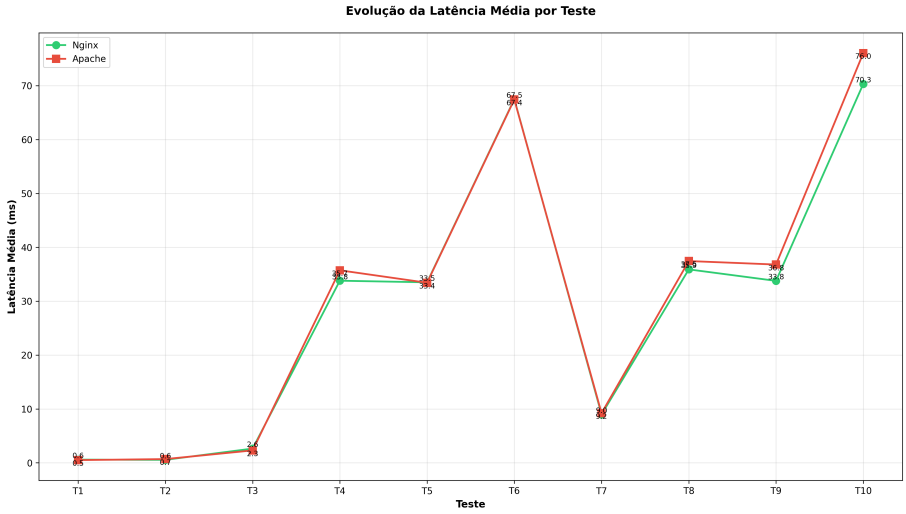

Figure 2. Evolução da latência ao longo das requisições. Fonte: autor.

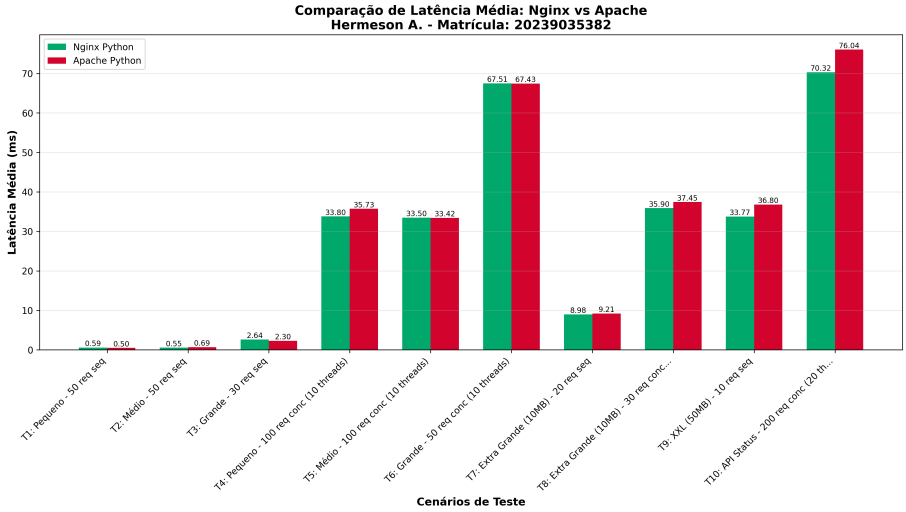

Figure 3. Comparação da latência média entre Nginx e Apache. Fonte: autor.

4.3. Comparação geral de métricas

Uma visão consolidada das métricas por cenário é apresentada na Figura 4 e mostra, de forma geral:

- Nginx com melhor vazão e menor latência média em cenários concorrentes, especialmente com payload pequeno.
- Apache com desempenho estável e próximo ao do Nginx em cenários sequenciais e em testes com arquivos grandes.
- Taxas de sucesso altas para ambos os servidores, com eventuais falhas apenas em situações de maior estresse.

4.4. Síntese dos vencedores por cenário

Por fim, a Figura 5 apresenta um "placar" de vencedores por cenário, resumindo qual servidor teve melhor desempenho considerando principalmente latência e vazão em cada

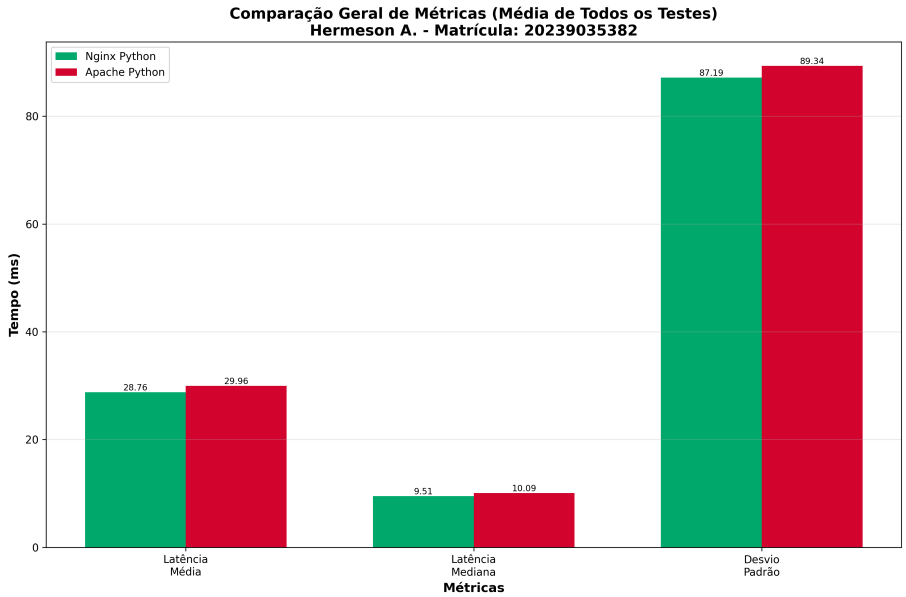

Figure 4. Comparativo geral de métricas por cenário. Fonte: autor.

tipo de teste. Esse gráfico facilita a interpretação rápida dos resultados e mostra em quais situações Nginx ou Apache são mais indicados.

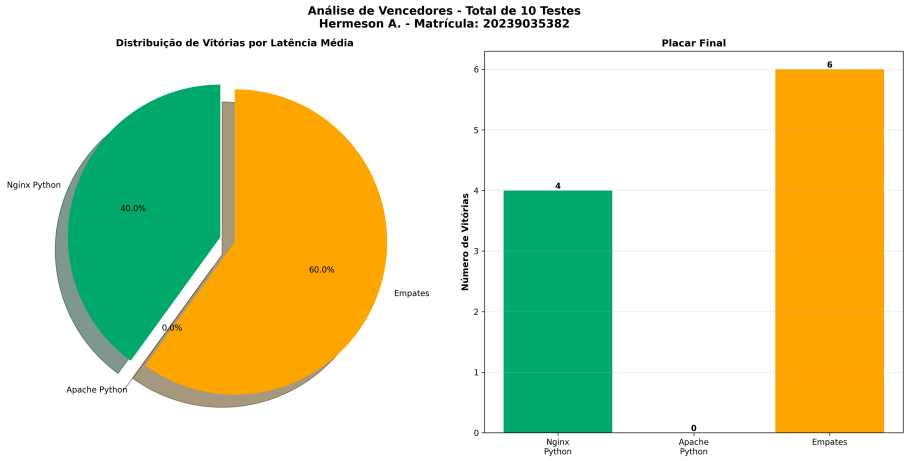

Figure 5. Resumo dos vencedores por cenário. Fonte: autor.

5. Conclusão

Os experimentos realizados demonstraram que Nginx e Apache apresentam comportamentos distintos dependendo do tipo de carga aplicada. Em cenários de alta concorrência e respostas pequenas, o Nginx tende a alcançar maior vazão e menor latência, favorecido por sua arquitetura orientada a eventos. Em cenários sequenciais ou com carga moderada, o Apache oferece desempenho estável e resultados próximos aos do Nginx.

O ambiente em Docker, a stack de observabilidade com Prometheus e Grafana e o cliente em Python atenderam aos objetivos propostos, fornecendo uma base reprodutível para avaliação de desempenho de servidores web de mercado. A partir desses resultados, a escolha entre Nginx e Apache deve considerar o perfil da aplicação: para APIs e

aplicações com grande número de conexões simultâneas, o Nginx tende a ser mais eficiente, enquanto o Apache continua adequado para cenários com menor concorrência e requisitos de compatibilidade e flexibilidade de configuração.