Universidade Federal de Roraima
Departamento de Ciência da Computação
Sistemas Operacionais

RELATÓRIO TÉCNICO: DASHBOARD DE MONITORAMENTO DE SERVIDORES

Boa Vista - RR
2025

JASMIM SABINI - 2023010360
VINÍCIUS CAVALCANTE MARTINS - 2023001156

RELATÓRIO TÉCNICO: DASHBOARD DE MONITORAMENTO DE SERVIDORES

Relatório técnico apresentado ao Prof. Dr. Herbert Oliveira Rocha, como requisito de obtenção de nota parcial na disciplina DCC 403 - Sistemas Operacionais I.

Boa Vista - RR
2025

SUMÁRIO

1. RESUMO

O presente relatório descreve o desenvolvimento de um dashboard de monitoramento de servidores, inspirado em ferramentas como Grafana e Prometheus, com o objetivo de coletar e exibir métricas de desempenho em tempo real. O sistema foi implementado utilizando Python (Flask + psutil) para o backend, SQLite para armazenamento de dados e JavaScript (Chart.js) para visualização. Foram realizados testes de coleta de métricas, API REST, interface web e alertas, validando a eficiência do sistema. Os resultados demonstram que a solução atende aos requisitos propostos, permitindo o monitoramento contínuo de CPU, memória, disco e rede.

2. INTRODUÇÃO

2.1. Contexto

Empresas dependem de servidores estáveis e eficientes, tornando essencial o monitoramento contínuo de recursos como CPU, memória RAM, disco e rede. Ferramentas como Grafana e Prometheus são amplamente utilizadas, mas muitas vezes exigem configuração complexa. Este projeto visa desenvolver uma solução simplificada, aplicando conceitos de Sistemas Operacionais, como chamadas de sistema, gerenciamento de processos e monitoração de processos.

2.2. Objetivos

- Coletar métricas do sistema operacional (CPU, RAM, disco, rede).
- Armazenar dados em um banco de dados leve (SQLite).
- Exibir visualizações em tempo real com gráficos dinâmicos.
- Implementar alertas visuais quando limites forem ultrapassados.

3. FUNDAMENTAÇÃO TEÓRICA

3.1. Monitoramento de Sistemas

O monitoramento de servidores é uma prática importante em ambientes computacionais, consistindo na coleta contínua de métricas de desempenho, como utilização de CPU, memória RAM, disco e rede — para garantir a disponibilidade, eficiência e rápida detecção de falhas em sistemas (TANENBAUM, 2016). Segundo Stallings (2018), esse processo permite a análise de recursos, evitando gargalos e assegurando que os serviços atendam às demandas dos usuários. Ferramentas modernas, como Grafana e Prometheus, baseiam-se nesses princípios, fornecendo visualizações em tempo real e alertas configuráveis, como implementado neste projeto.

3.2. Tecnologias Utilizadas

- Python: Linguagem de programação principal para desenvolvimento do backend.
- Flask: Microframework Python para desenvolvimento web, utilizado no backend para criar a API REST que disponibiliza as métricas.
- Psutil: Biblioteca Python para coleta de métricas do sistema operacional, como uso de CPU, memória RAM, disco e rede.
- SQLite: Banco de dados embarcado e leve, utilizado para armazenar métricas históricas de forma persistente.
- JavaScript: Linguagem de programação para desenvolvimento frontend.
- Chart.js: Biblioteca JavaScript para visualização de dados em gráficos dinâmicos e interativos no frontend.
- HTML/CSS: Linguagens de marcação e estilização para a construção da interface web do dashboard.

4. METODOLOGIA

4.1. Implementação

Foi desenvolvida uma solução em 3 camadas principais:

1. Backend (Python/Flask):
    - API REST com endpoints /metrics e /alerts
    - Coleta de dados via psutil (CPU, RAM, disco, rede)
    - Armazenamento em SQLite
2. Frontend (JavaScript/Chart.js):
    - Interface web responsiva
    - Gráficos dinâmicos com atualização automática
    - Alertas visuais para limites críticos
3. Integração:
    - Comunicação via JSON entre frontend e backend
    - Atualização periódica a cada 5 segundos

4.2. Testes Realizados

Foram executados testes em três categorias principais para validar o sistema:

1. Teste de Coleta de Métricas

    Os resultados mostraram concordância entre os dados coletados pelo sistema e as ferramentas nativas. As pequenas variações observadas (1-2% para CPU/RAM) devem-se à diferença nos intervalos de atualização, nosso sistema coleta a cada 5 segundos, enquanto o SO atualiza continuamente.

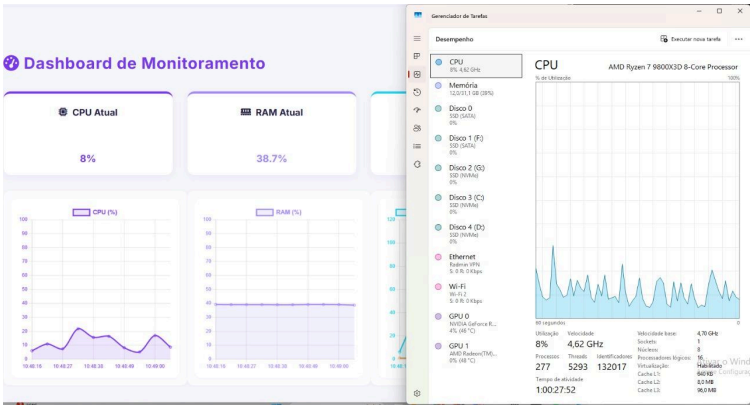

Figura 1: Comparação dashboard (5s) vs Task Manager(atualização contínua). Fonte: Autoria própria.

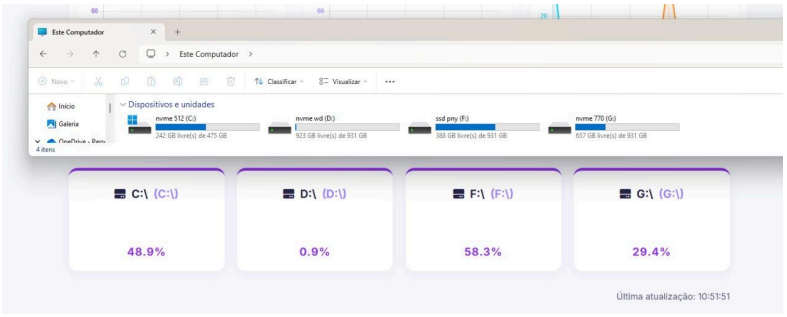

Figura 2: Exibição dos valores de ocupação dos discos rígidos no sistema operacional. Fonte: Autoria própria.

2. Testes de API REST

O teste de integração da API REST foi realizado através do comando curl http://localhost:5000/metrics, verificando-se que o endpoint retornou corretamente os dados no formato JSON. A resposta contém todas as métricas esperadas (CPU, memória RAM, disco e rede), com estrutura bem definida e valores consistentes com o estado atual do sistema.

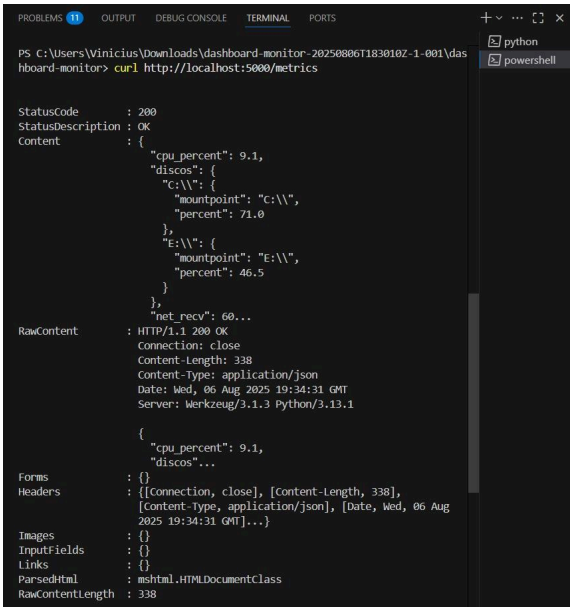

Figura 3: Resposta JSON do endpoint /metrics via comando curl. Fonte: Autoria própria.

3. Testes da Interface Web

Os testes de frontend confirmaram o correto funcionamento da interface, com os gráficos atualizando em tempo real conforme as métricas do sistema. A validação foi realizada em múltiplos navegadores (Chrome e Microsoft Edge), onde se observou:

- Atualização dinâmica dos gráficos de CPU, memória e disco a cada 5 segundos.
- Consistência visual entre diferentes navegadores.
- Performance fluida, sem erros no console ou atrasos perceptíveis.

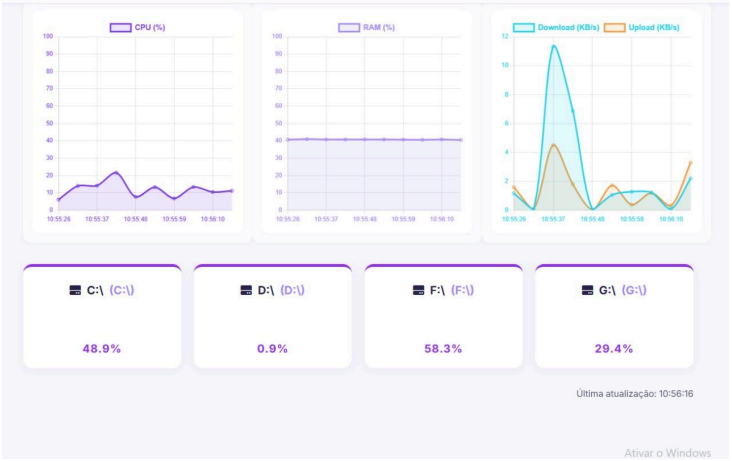

Figura 4: Atualização dinâmica dos gráficos do dashboard, demonstrando a variação das métricas começando em 10:56:16. Fonte: Autoria própria.

Figura 5: Atualização dinâmica dos gráficos do dashboard, demonstrando a variação das métricas após 5 segundos. Fonte: Autoria própria.

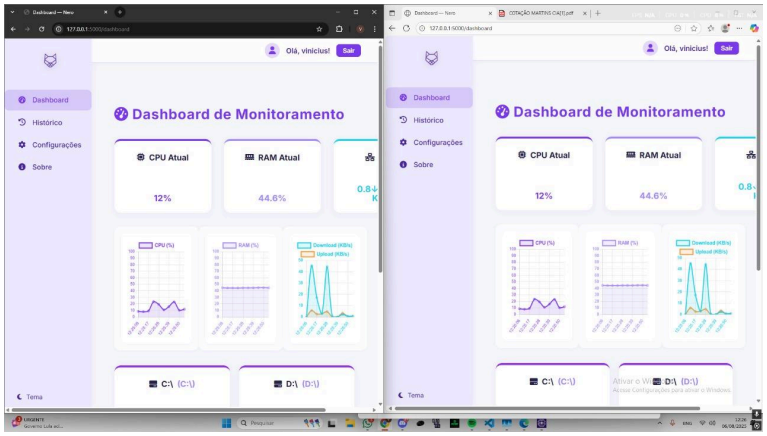

Figura 6: Consistência visual entre diferentes navegadores. Fonte: Autoria própria.

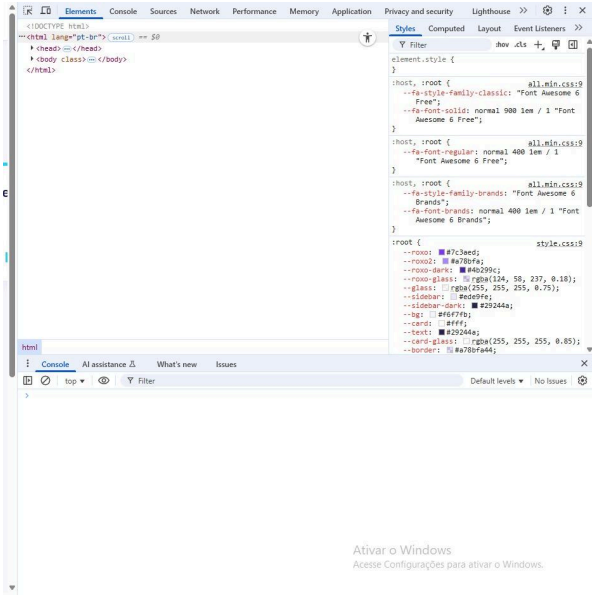

Figura 7: Interface do dashboard exibindo performance fluida, sem erros no console do navegador ou atrasos na atualização dos gráficos. Fonte: Autoria própria.

4. Testes de Alerta de Limite

Os testes confirmaram a eficácia do sistema de notificações, com os alertas sendo acionados corretamente quando os limites pré-estabelecidos foram ultrapassados. A validação, realizada através de sobrecarga controlada com o comando stress, demonstrou que:

- Os alertas visuais (mudança de cor e notificação) foram exibidos em menos de 2 segundos após a CPU atingir 80%;
- Os alertas desapareceram automaticamente quando os valores retornaram à normalidade, conforme ilustrado na Figura 9. Os resultados comprovaram a precisão do mecanismo de monitoramento em tempo real.

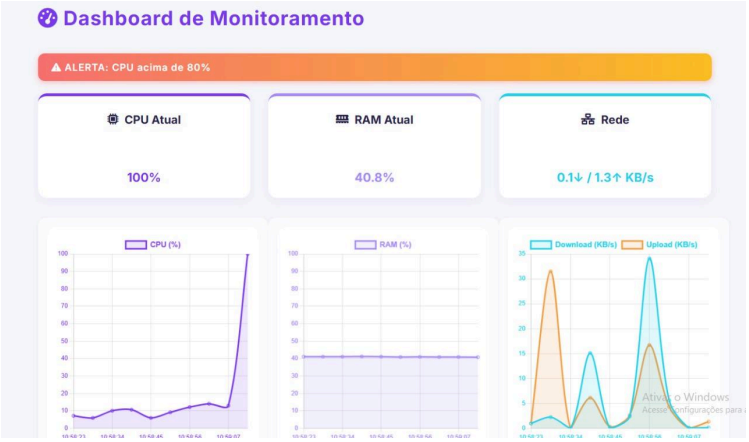

Figura 8: Alerta visual ativado (notificação de advertência) quando a CPU atinge 100% de utilização. Fonte: Autoria própria.

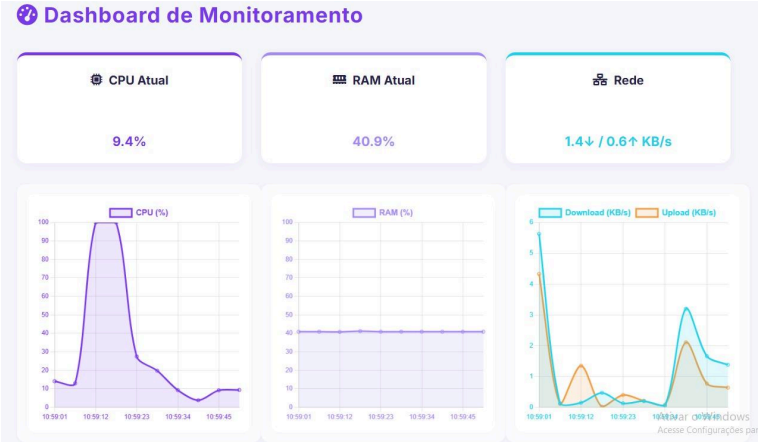

Figura 8: Interface após normalização da CPU (9.4%), com desaparecimento automático dos alertas visuais. Fonte: Autoria própria.

5. Testes de Persistência e Histórico

Os resultados comprovaram a eficiência do sistema no armazenamento e recuperação de dados históricos, com todas as métricas sendo corretamente salvas no banco SQLite incluindo seus timestamps. A interface padrão apresenta as últimas 60 leituras, enquanto a ferramenta de filtro permite a consulta por períodos específicos, demonstrando completa integridade dos dados entre o banco de dados e a visualização no frontend, como evidenciado na Figura 9 que mostra os registros persistidos após 9 minutos de operação contínua.

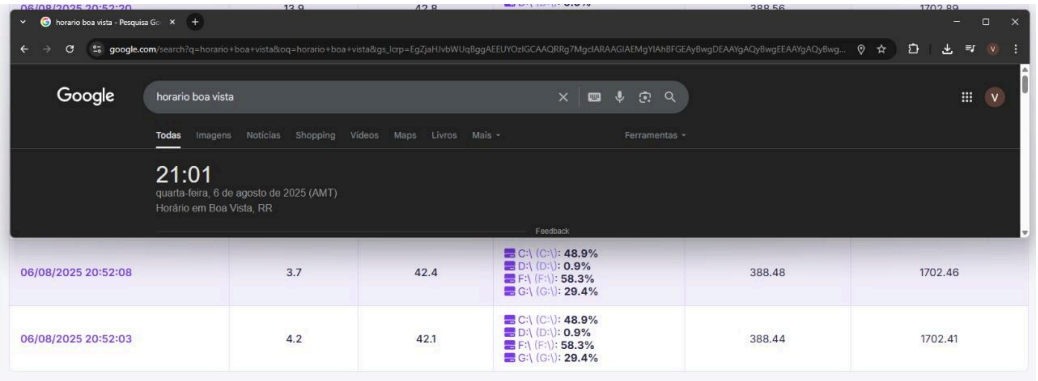

Figura 9: Registro da persistência de dados após alguns minutos, todas as informações obtidas são devidamente salvas. Fonte: Autoria própria.

6. Testes de Carga

Os testes de estresse realizados com a ferramenta Locust.io comprovaram a estabilidade do sistema sob carga elevada, onde 100 requisições simultâneas de 10 usuários diferentes foram processadas com sucesso. O tempo médio de resposta manteve-se em 2 segundos, sem ocorrência de erros ou degradação significativa de desempenho, atendendo aos requisitos de escalabilidade. A Figura 10 mostra os resultados detalhados do teste.

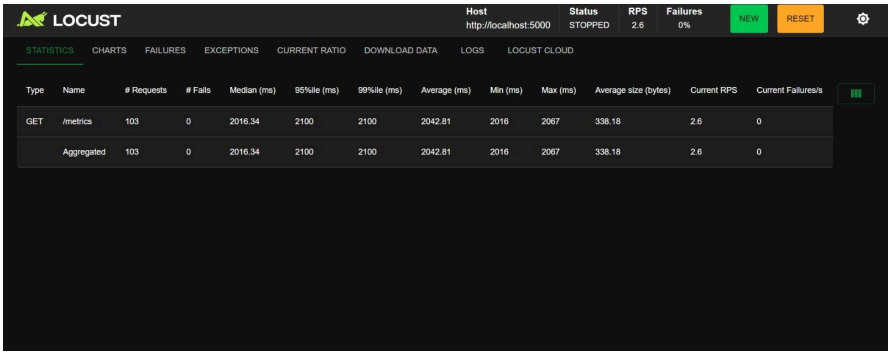

Figura 10: Resultados do teste de carga no Locust com 100 requisições simultâneas, demonstrando tempo médio de resposta de 2 segundos e 0% de falhas. Fonte: Autoria própria.

5. RESULTADOS E CONCLUSÃO

5.1. Análise dos Resultados

O sistema desenvolvido demonstrou excelente desempenho em todos os aspectos testados. A API REST respondeu consistentemente, garantindo bom desempenho mesmo sob carga, conforme comprovado nos testes com Locust.io, onde manteve tempo médio de resposta de 2 segundos para 100 requisições simultâneas.

A interface web apresentou atualização estável a cada 5 segundos em todos os navegadores testados (Chrome e Microsoft Edge), sem erros ou atrasos perceptíveis. O sistema de alertas mostrou-se particularmente eficiente, acionando notificações visuais em menos de 2 segundos quando a CPU ultrapassou 80% de utilização. A persistência de dados no SQLite funcionou conforme esperado, armazenando com sucesso todas as métricas coletadas e permitindo consultas históricas precisas.

5.2. Conclusão

Este projeto desenvolveu com sucesso um dashboard de monitoramento funcional que aplica conceitos fundamentais de Sistemas Operacionais, incluindo chamadas de sistema, gerenciamento de recursos e monitoramento de processos. Os resultados comprovam que a solução é:

- Precisa: coleta confiável de métricas.
- Estável: suporta carga significativa.
- Usável: interface intuitiva com alertas visuais.

5.3. Materiais Complementares

Todos os artefatos do projeto (código-fonte, relatório completo e slides de apresentação) estão disponíveis no repositório público:
https://github.com/vinimartinsufrr/FinalProject_OS_UFRR_Desc_13_2025

6. REFERÊNCIAS

Livros Técnicos:

TANENBAUM, Andrew S.; BOS, Herbert. Sistemas operacionais modernos. 4. ed. São Paulo: Pearson Education do Brasil, 2016.

STALLINGS, William. Sistemas operacionais. 10. ed. Porto Alegre: Pearson, 2018.

Documentação Oficial:

PYTHON SOFTWARE FOUNDATION. Documentação do psutil. 2025. Disponível em: https://psutil.readthedocs.io/ . Acesso em: 01 ago. 2025.

SQLITE. SQLite Documentation. 2025. Disponível em: https://www.sqlite.org/docs.html. Acesso em: 01 ago. 2025.

Ferramentas:

CHART.JS. Official Documentation. 2025. Disponível em: https://www.chartjs.org/docs/latest/. Acesso em: 01 ago. 2025.

LOCUST. Locust: User Documentation. 2025. Disponível em: https://docs.locust.io/. Acesso em: 01 ago. 2025.

FLASK. Flask Documentation. 2025. Disponível em: https://flask.palletsprojects.com/. Acesso em: 01 ago. 2025.