Andrew Carvalho de Sá| 13681252
Tiago Girio Felice | 13682750
Caio Vinicius Meira Penayo | 11857406

Documentação Técnica – Sistema de Observabilidade de CPU e Memória para Dispositivos IoT Baseado em Raspberry Pi

Este documento descreve a arquitetura, implementação e funcionamento de um sistema de observabilidade de recursos computacionais desenvolvido para dispositivos de borda (*edge devices*), com foco em Raspberry Pi rodando Linux.
A solução integra:

- Coleta de métricas via parsing do pseudo-sistema de arquivos `/proc`.
- Processamento local com normalização de dados e cálculo de indicadores-chave de desempenho (KPIs).
- Exposição via API RESTful em formato JSON para consumo por clientes externos.
- Frontend interativo baseado em React, com atualização periódica e renderização dinâmica de gráficos e tabelas.

A arquitetura segue princípios de design desacoplado e responsabilidades bem definidas, permitindo escalabilidade horizontal e adaptação para múltiplos dispositivos em rede.

Visão Geral da Arquitetura

A solução está organizada em três camadas:

1. Camada de Coleta e Processamento (Backend Go)
    - Extração de métricas brutas diretamente de interfaces do kernel Linux.
    - Normalização e agregação de dados.
    - Implementação de algoritmos de cálculo para uso percentual de CPU, uso de memória, taxas de interrupção e trocas de contexto.
    - Exposição dos dados via API HTTP REST.
2. Camada de Transporte de Dados (HTTP REST)
    - Comunicação entre backend e frontend via protocolo HTTP.

- Formato de dados padronizado em JSON.

3. Camada de Visualização (Frontend React)
    - Consumo assíncrono de dados usando `fetch` API.
    - Atualização automática com *polling* periódico.
    - Interface responsiva com gráficos e tabelas, permitindo análise visual das métricas em tempo real.

Backend – Coleta e Processamento

O backend foi desenvolvido em Go devido à sua alta performance, baixo consumo de memória e suporte nativo a concorrência, o que o torna ideal para aplicações em computação de borda.

Leitura de Métricas

As métricas são coletadas através de leitura direta do pseudo-sistema de arquivos Linux:

Métrica	Fonte	Descrição
Uso de CPU	`/proc/stat`	Calcula a porcentagem de tempo gasto em estados ativos versus ociosos.
Uso de Memória	`/proc/meminfo`	Determina a fração de RAM utilizada.
Interrupções	`/proc/interrupts`	Contabiliza eventos de hardware por CPU.
Trocas de Contexto	`/proc/stat` (`ctxt`)	Mede mudanças de execução entre processos.

Algoritmos de Cálculo

Uso de CPU:

1. Leitura dos contadores acumulados de tempo da CPU.
2. Cálculo da diferença (*delta*) entre duas leituras consecutivas.

3. Determinação da fração de tempo ativo:

```
total_time = sum(valores)
idle_time = idle + iowait
uso_cpu = (total_time - idle_time) / total_time
```

Uso de memória:

```
mem_usada = MemTotal - (MemFree + Buffers + Cached)
uso_memoria = mem_usada / MemTotal
```

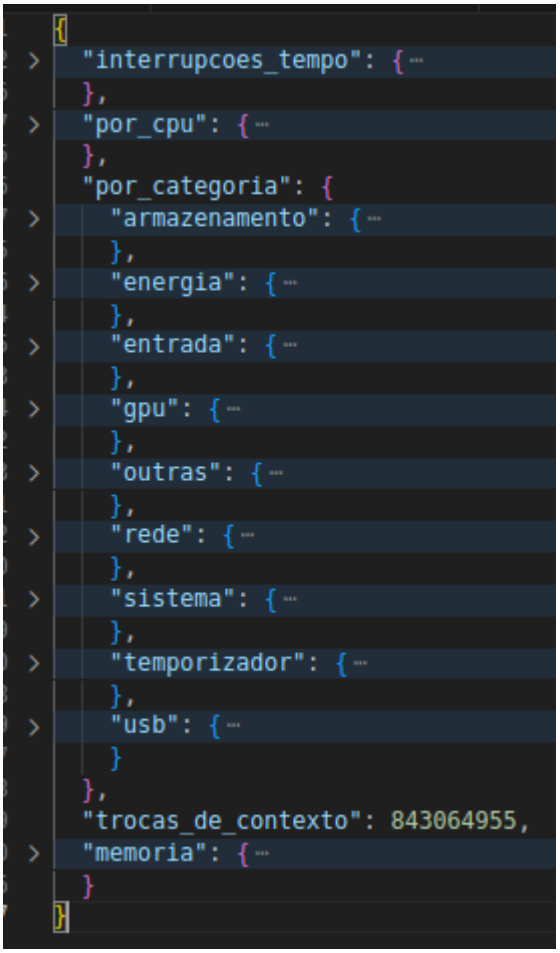
Taxa de interrupções:

```
taxa = (interrupcoes_finais - interrupcoes_iniciais) / delta_tempo
```

API REST

A camada de transporte expõe um endpoint principal para consulta das métricas:

- Endpoint: `/metrics`
- Método: GET
- Formato de resposta: JSON
- Exemplo:



A API é *stateless* e pode ser integrada a ferramentas externas como Postman, cURL ou Grafana.

Frontend – Visualização das Métricas

O frontend foi desenvolvido em React, adotando a estratégia de *client-side rendering* e atualização periódica para refletir métricas em tempo real.

Consumo da API

```
useEffect(() => {
    const interval = setInterval(() => {
        fetch("http://192.168.0.98:8080/metrics")
            .then(res => res.json())
            .then(data => setMetrics(data));
    }, 2000);
    return () => clearInterval(interval);
}, []);
```

Exibição e Visualização dos Dados

A interface de visualização, implementada em React, fornece uma experiência interativa e intuitiva para a análise das métricas monitoradas.

O frontend consome as informações disponibilizadas pela API REST desenvolvida em Go e apresenta os dados por meio de gráficos dinâmicos, que incluem:

- Gráficos de linha para representar a evolução temporal de métricas como uso de CPU, uso de memória e taxa de interrupções.

- Gráficos de barras para facilitar a comparação entre diferentes núcleos do processador ou categorias de interrupções.

- Atualização em tempo real dos valores exibidos, proporcionando uma visão instantânea do estado atual do sistema.

Essa abordagem visual possibilita a identificação rápida de tendências, gargalos e comportamentos anômalos, tornando a solução não apenas funcional, mas também amigável para usuários técnicos e não técnicos.

Segurança e Acesso Remoto

O acesso remoto à Raspberry Pi é realizado por meio do protocolo SSH (Secure Shell), garantindo a segurança na comunicação entre o computador de desenvolvimento e o dispositivo embarcado. A conexão SSH utiliza criptografia ponta a ponta para proteger a transmissão de credenciais, comandos e dados. Isso permite que o time de desenvolvimento acesse e gerencie o sistema, execute scripts, depure a aplicação e monitore logs em tempo real sem a necessidade de acesso físico ao dispositivo.

Conclusão

A solução desenvolvida representa uma arquitetura enxuta e eficiente para monitoramento de sistemas embarcados, aplicável tanto para IoT industrial quanto para home labs.
Apesar de ser executada em hardware modesto, a estrutura é suficientemente robusta para escalar em ambientes distribuídos e integrar-se a ecossistemas complexos de observabilidade.