Relatório do Sistema de Monitoramento Distribuído de Falhas e Recursos

1. Visão Geral

Este sistema simula um ambiente distribuído para monitoramento de falhas e métricas (CPU/memória) de múltiplos nodes, utilizando containers Docker para isolar cada serviço. O monitor central recebe métricas e detecta falhas, enquanto uma interface web exibe eventos e permite exportação de logs e configuração dinâmica.

2. Funcionalidades

- Monitoramento de nodes via heartbeat e métricas (CPU/memória)
- Detecção de falhas e recuperação automática
- Dashboard em terminal (UIMonitor) com tabela colorida, feed de eventos, ordenação e destaques
- Interface web (Flask) para visualização, exportação de logs (CSV/JSON) e alteração dinâmica de parâmetros
- Logs sincronizados entre host e containers
- Configuração dinâmica via `config.txt`
- Script para abrir monitor, nodes e web em terminais separados
- Suporte a múltiplos nodes simulados
- Fácil apresentação e depuração

3. Como Executar

Pré-requisitos

- Docker e Docker Compose instalados
- Python 3 instalado no host (para interface web)

Passos

1. Edite `config.txt` se desejar alterar timeout/intervalo padrão.
2. Execute o script `abrir_terminais.bat` e informe o número de nodes desejado.
3. O monitor, nodes e interface web abrirão em terminais separados.
4. Acesse a interface web em `http://localhost:5000`.
5. Para exportar logs ou alterar configurações, use a interface web.

4. Estrutura dos Arquivos e Explicação

`docker-compose.yml` e `Dockerfile`

- Definem os serviços (monitor, nodes) e ambiente de execução.
- Apenas o monitor expõe porta para o host.
- Volumes sincronizam logs entre containers e host.

`abrir_terminais.bat`

- Script Windows que pergunta o número de nodes e abre um terminal para cada serviço (monitor, nodes, web).
- Facilita a visualização e controle manual.

`src/detector/monitor/NodeMonitor.java`

- Classe principal do monitor.
- Gerencia recebimento de métricas, detecção de falhas e integração com UIMonitor.
- Salva logs de eventos de falha/recuperação.

`src/detector/monitor/UIMonitor.java`

- Interface de dashboard em terminal.
- Exibe tabela de nodes, eventos recentes, tempo desde último heartbeat, destaques visuais.

`src/detector/monitor/FailureDetector.java`

- Lógica de detecção de falhas baseada em timeout e heartbeats.
- Notifica UIMonitor e registra eventos.

`src/detector/monitor/MetricReceiver.java`

- Recebe métricas dos nodes via TCP.
- Atualiza status dos nodes no monitor.

`src/detector/monitor/HeartbeatListener.java` e `HeartbeatSender.java`

- (Se aplicável) Gerenciam envio e recebimento de heartbeats entre nodes e monitor.

`src/detector/simulador/NodeSimulator.java`

- Simula nodes enviando métricas e heartbeats ao monitor.
- Suporta comandos interativos, retry/backoff, status visual e ajuda.

`web_monitor.py`

- Interface web (Flask) para visualização de eventos, exportação de logs e alteração dinâmica de configuração.
- Detecta automaticamente o arquivo de log correto (host ou Docker).

`src/logs/failwatch_log.txt` **e** `src/logs/config.txt`

- Arquivo de log de eventos (falha/recuperação) e configuração dinâmica (timeout/interval).

`README.md`

- Documentação detalhada do sistema, instruções de uso e dicas de apresentação.

5. Observações Finais

- O sistema é modular e fácil de expandir (ex: notificações externas, healthchecks, etc).
- O uso de Docker garante portabilidade e reprodutibilidade.
- O script facilita a apresentação e depuração, abrindo todos os serviços em terminais separados.

Relatório gerado automaticamente em 28/06/2025.