

## Web Monitoring Agent

### Documentação Completa

Este documento detalha aspectos fundamentais do projeto **web-monitoring-agent**, incluindo sua arquitetura, processo de deploy contínuo (CI/CD) e orientações para utilização e personalização dos dashboards Grafana.

---

### Arquitetura (High-Level Design - HLD)

#### Arquitetura Hexagonal

A arquitetura Hexagonal (Ports and Adapters) promove uma separação clara das responsabilidades, facilitando a manutenção, testes e evolução do projeto.

#### Componentes Principais:

- **Domain:**
  - Contém entidades e regras de negócio
  - Interfaces abstratas para repositórios
- **Application:**
  - Responsável por coordenar os casos de uso e lógica de negócio
  - Interage diretamente com o domínio e infraestrutura
- **Infrastructure:**
  - Persistência de dados utilizando PostgreSQL (via SQLAlchemy)
  - Configuração de ambiente e provisionamento do Grafana
- **Presentation:**
  - Expõe endpoints REST com FastAPI

#### Estrutura Detalhada

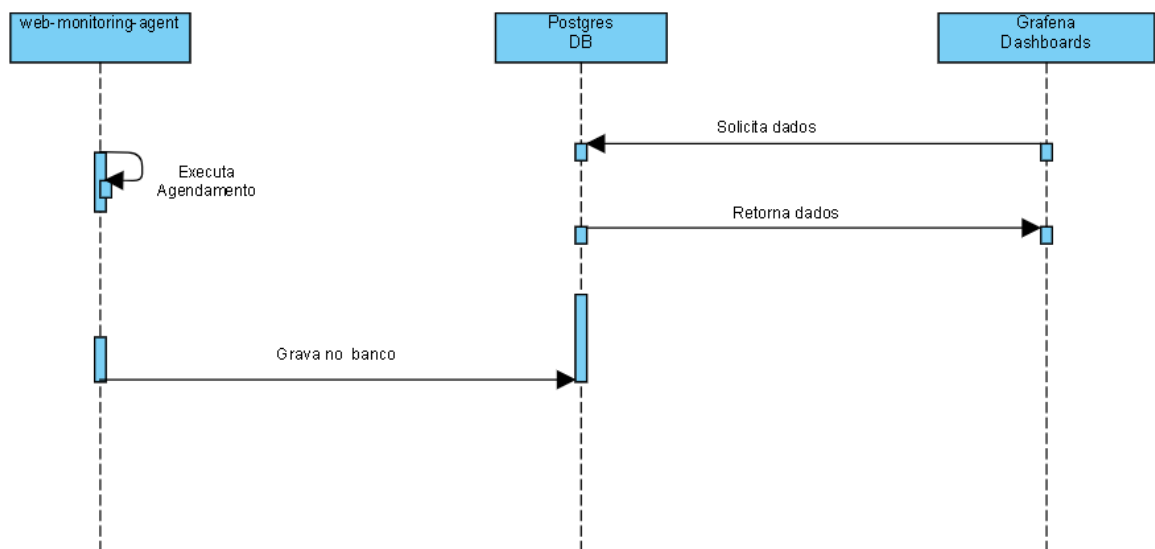
web-monitoring-agent/

```
├── .github
|   └── workflows
|       ├── ci-cd.yml
|       └── ci.yml
└── assets
```

```
└─ docker-compose.yml
└─ Dockerfile
└─ requirements.txt
└─ src
  │ └─ application
  │ │ └─ services
  │ │ └─ dtos.py
  │ └─ domain
  │ │ └─ entities.py
  │ │ └─ repositories.py
  │ └─ infrastructure
  │ │ └─ config
  │ │ │ └─ config_loader.py
  │ │ │ └─ config.yaml
  │ │ └─ database
  │ │ └─ grafana
  │ │ │ └─ provisioning
  │ │ │ └─ dashboards
  │ │ │ │ └─ network_monitoring.json
  │ │ │ └─ dashboards.yaml
  │ │ │ └─ datasources.yaml
  │ │ └─ persistence
  │ │ │ └─ connection_factory.py
  │ │ │ └─ initial.sql
  │ │ └─ scheduler
  │ │ └─ scheduler.py
  └─ presentation
```

- | | | controllers
- | | | routes.py
- | | main.py
- | tests
- | test\_api.py
- README.md

## Lógica da aplicação



### Web-monitoring-agent

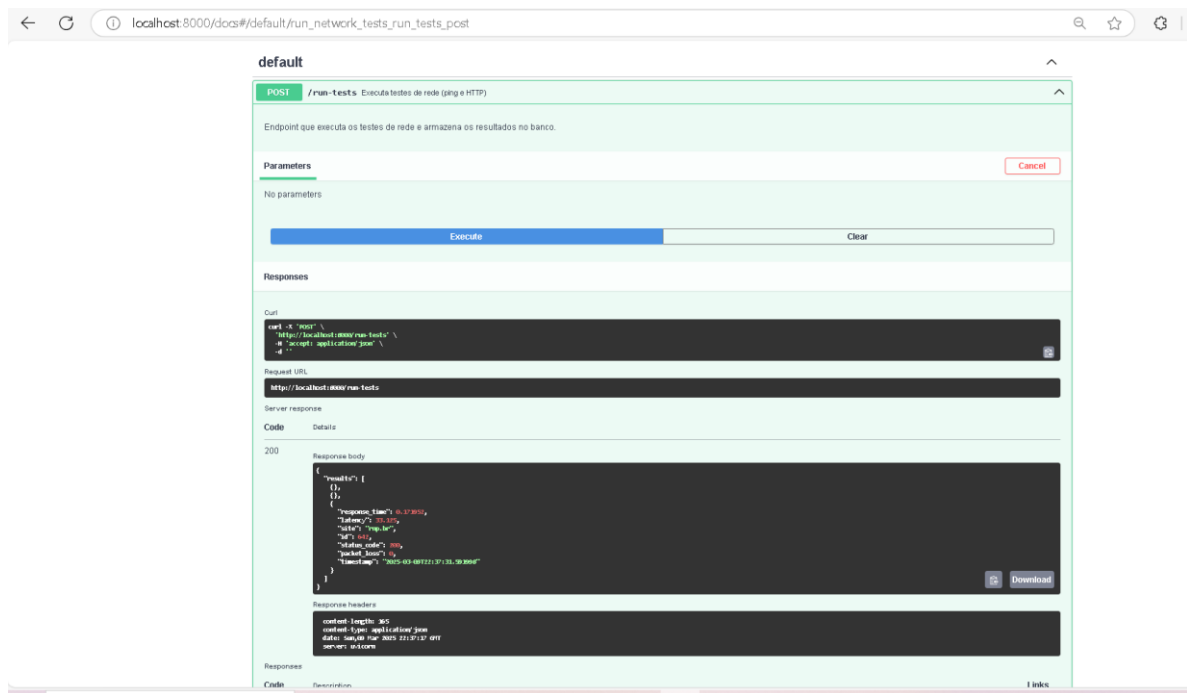
Foi criado um container Docker isolado, contendo toda a configuração, para esse microserviço funcione corretamente.

**Executar Agendamento:** Foi criado um agendamento em Python que verifica as operações a cada 30 segundos. Esse intervalo, bem como as URLs monitoradas, podem ser alterados no arquivo **config.yaml**.

```
src > infrastructure > config > config.yaml
1 monitoring:
2   interval_seconds: 30 # Tempo de agendamento (em segundos)
3   sites:
4     - google.com
5     - youtube.com
6     - rnp.br
```

**Gravar no banco:** Utilizou-se o SQLAlchemy como ORM para abstrair a interação com o banco de dados. Além disso, este microserviço é RESTful, embora nem todos os verbos RESTful tenham sido implementados.

Além disso, foi configurado o Swagger para testar a gravação do método POST, **run-tests**, que grava no banco de dados, que pode ser acessado em <http://localhost:8000/docs#>.



## Postgres

Foi criado um container Docker para isolar a aplicação, decidi criar uma única tabela para armazenar essas informações.

Query: Query History

Scratch Pad X

SQL

```
SELECT * FROM network_test_results
```

Data Output Messages Notifications

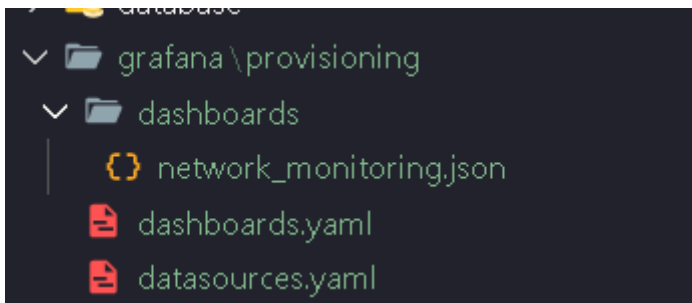
Showing rows: 1 to 680 Page No: 1 of 1

id	site	latency	packet_loss	response_time	timestamp	status_code
int	varchar	double precision	double precision	double precision	timestamp without time zone	integer
672	mpar	24.03333333333333	25	0.241997	2025-09-09 22:43:13.04889	200
673	google.com	20.6	0	0.172599	2025-09-09 22:43:34.039617	200
674	youtube.com	23.700000000000004	0	0.324195	2025-09-09 22:43:38.134267	200
675	mpar	15.720000000000001	0	0.166323	2025-09-09 22:43:43.210422	200
676	google.com	17.85	0	0.172455	2025-09-09 22:43:04.097259	200
677	youtube.com	29.270000000000002	0	1.064002	2025-09-09 22:43:10.522554	200
678	mpar	16.533333333333335	25	0.190211	2025-09-09 22:43:15.782895	200
679	google.com	16.75	50	0.177067	2025-09-09 22:43:35.108932	200
680	youtube.com	26.325	0	0.28613	2025-09-09 22:43:39.711343	200

## Dashboards Grafana

O Grafana está separado em um container e tem como objetivo escutar o banco de dados e exibir as informações na tela.

Na aplicação, temos uma camada de configuração infraestrutura, dentro dessa camada temos a pasta "grafana", onde se encontra o arquivo **datasources.yaml** e o **dashboards.yaml**, que fazem a configuração de acesso ao dashboard pelo container. Já dentro da pasta "dashboards", temos o arquivo **network\_monitoring.json**, que define a estilização do dashboard, query, essa solução serve para que ao subir o container do grafana ele já vá com o dashboard configurado.



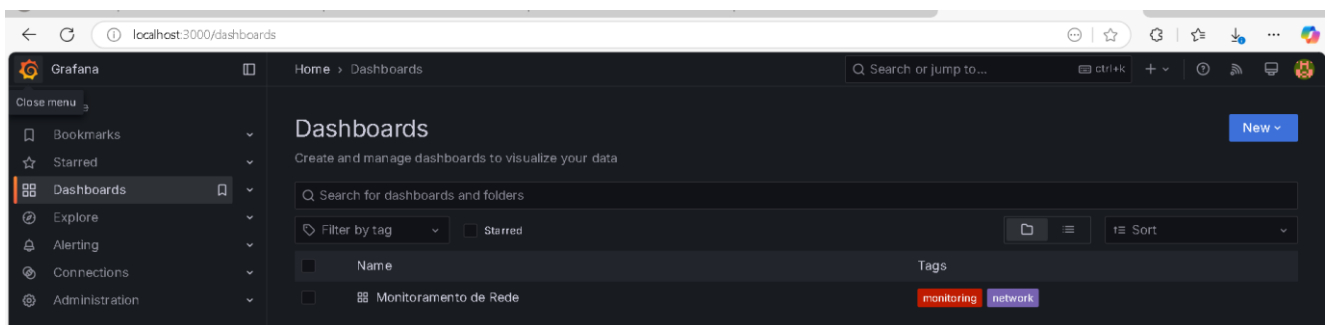
### Acesso Inicial

- URL: <http://localhost:3000>
- Credenciais: admin/admin (altere após login inicial)

### Dashboard

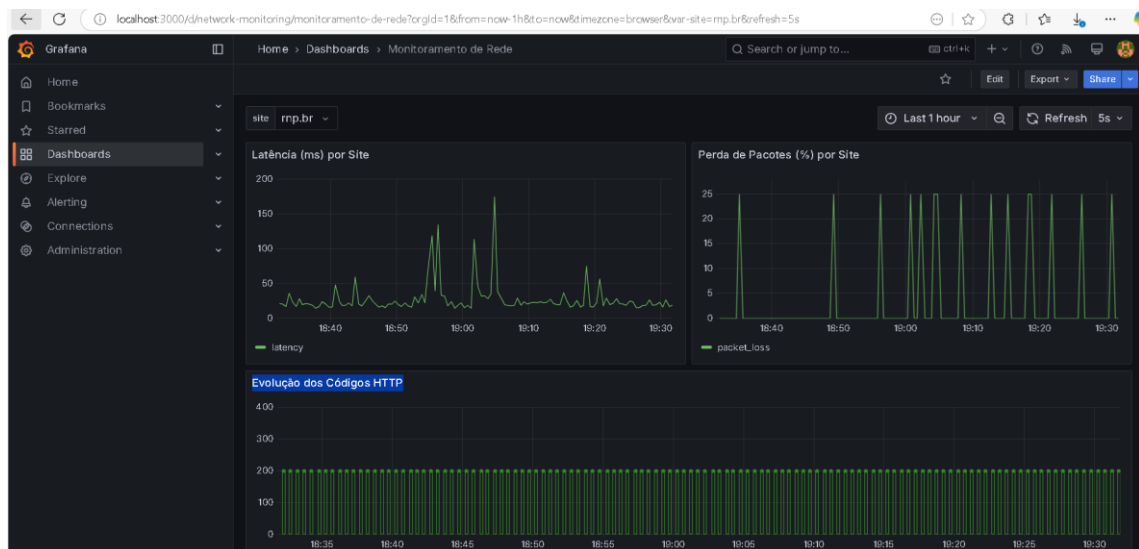
Navegue em: menu lateral > Dashboards

Clique em: Monitoramento de Rede



Exibe:

- Latência (ms) por Site
- Perda de Pacotes (%) por Site
- Evolução dos Códigos HTTP



## Guia para Deploy e CI/CD

### Pré-requisitos

- Docker Hub
- Servidor com Docker configurado
- Repositório GitHub com secrets configurados

### Configuração GitHub Secrets

Adicione ao repositório:

- DOCKER\_USERNAME
- DOCKER\_PASSWORD
- SSH\_HOST
- SSH\_USER
- SSH\_KEY

- DATABASE\_URL

## Pipeline CI/CD

- Checkout do código
- Execução de testes automatizados (pytest)
- Build e push da imagem Docker(Poderia facilmente ser alterada para AWS)
- Deploy automático via SSH

Foi realizada a configuração do GitFlow

← CI/CD Monitoramento de Rede

🟡 Update testes unitários #12

🏠 Summary

---

Jobs


🟡 Rodar Testes Unitários

---

Run details

🕒 Usage

📄 Workflow file

Triggered via push now	Status	Total duration	Artifacts
 pushed -> 7a864e7 <span>main</span>	In progress	—	—

d.yml  
on: push

🟡 Rodar Testes Unitários 22s

○ Deploy da Aplicação

Para testes locais, sugiro realizar o **Deploy Manual**

Execute localmente:

**docker-compose up -d**

---