gRPC - PROMETHEUS - GRAFANA

1️⃣ Observabilidade ≠ Log ≠ Métrica isolada

“Aprendi que observabilidade de verdade não é só log ou gráfico bonito.
Ela conecta **métricas, alertas e contexto**, permitindo entender *o que está acontecendo* e *agir rápido*.”

2️⃣ Golden Signals aplicados na prática (gRPC)

“Implementei observabilidade seguindo os **Golden Signals**:

- **Traffic** → Requests por segundo
- **Errors** → Taxa de erro e volume nos últimos 5 minutos
- **Availability** → Serviço UP/DOWN
- **Latency** → (preparado para evoluir)”

3️⃣ Alertas que não fazem barulho à toa

“Criei alertas baseados em comportamento real, não em pânico:

- Só dispara se erro persistir por 1 minuto
- Usa janelas de tempo (5 min)
- Tem severidade, labels e mensagem clara”

4️⃣ Dashboards com propósito (não só gráfico)

“Separei dashboards por objetivo:

- **Observability** → entender comportamento
- **Errors (Detalhado)** → investigar incidentes
- **Health / Golden Signals** → decisão rápida”

5️⃣ Reprodutibilidade (nível time / empresa)

“Tudo está dockerizado:

- Prometheus
- Grafana
- gRPC service
- Provisionamento automático de dashboards e datasources”

“Tenho estudado observabilidade aplicada.
Montei um serviço gRPC instrumentado com Prometheus, criei dashboards baseados nos Golden Signals e alertas que só disparam quando o erro é real e persistente.
Tudo dockerizado e reprodutível. O foco foi aprender a **operar** o sistema, não só rodar.”

gRPC - PROMETHEUS - GRAFANA

- “Nos últimos dias eu venho estudando **observabilidade aplicada**, não só teoria.
  Montei um serviço **gRPC em .NET**, instrumentei com **Prometheus** e criei **dashboards no Grafana** baseados nos **Golden Signals**: tráfego, erros, disponibilidade e taxa de erro.
- Separei dashboards por propósito: um para visão executiva de saúde do serviço e outro focado em **investigação de erros**.
- Além disso, configurei **alertas reais**, com janela de tempo e período de persistência para evitar falso positivo. Validei o disparo do alerta via métricas e logs do Grafana.
- O alerta está funcionando corretamente; o ponto que estou ajustando agora é apenas o **canal de notificação (SMTP)**, que é infraestrutura, não regra.
- Todo o ambiente está **dockerizado**, com persistência de dados, então qualquer pessoa do time consegue subir e reproduzir. O foco foi aprender a **operar o sistema**, não só rodar código.”

Se ele perguntar sobre o email não ter chegado
O alerta está disparando corretamente. Eu confirmei isso pelos logs. O que falta é apenas a configuração do SMTP do Grafana, que é um detalhe de infraestrutura.

🧳 PARTE 1 — O que é o “docker-compose show case”

É um **pacote completo de observabilidade**, contendo:

✅ gRPC Server (.NET)
✅ Prometheus (scraping /metrics)
✅ Grafana
✅ Dashboards **pré-carregados**
✅ Alert rules **pré-carregadas**
✅ Webhook para provar que alertas disparam
✅ Persistência (nada some)

“Eu montei um ambiente completo de observabilidade para um serviço gRPC usando Prometheus e Grafana. Separei o tráfego gRPC em HTTP/2 com TLS e expus métricas em HTTP/1.1, o que evita problemas de scraping.
Criei dashboards baseados nos Golden Signals — tráfego, latência e erros — além de um dashboard específico para análise de erros por status.
Também configurei alertas com janela de persistência para evitar falsos positivos e validei tudo simulando falhas reais no serviço.
Todo o stack é reproduzível via Docker Compose, então qualquer pessoa consegue subir o ambiente e testar em minutos.”

gRPC - PROMETHEUS - GRAFANA

“Os alertas estão todos firing porque eu mantive erro contínuo no serviço. Em produção, o warning dispara primeiro e o critical só entra se o erro se sustentar por um período configurado.”

🧩 PARTE 2 — Estrutura de pastas (recomendada)

```
observability-grpc-demo/
│
├── docker-compose.yml
├── prometheus.yml
│
├── grafana/
│   ├── provisioning/
│   │   ├── datasources/
│   │   │   └── datasource.yml
│   │   ├── dashboards/
│   │   │   └── dashboards.yml
│   │   └── alerting/
│   │       └── alerts.yml
│   │
│   └── dashboards/
│       ├── grpc-observability.json
│       ├── grpc-errors.json
│       └── grpc-health.json
│
└── README.md

```

Server: GrpcServer

Client: GrpcClient

gRPC - PROMETHEUS - GRAFANA

DASHBOARDS

🟢 1) gRPC Service Observability

👉 Visão rápida / executiva / on-call

- Requests per Second
- Error Rate %

🟡 2) gRPC Service Health — Golden Signals

👉 Saúde geral do serviço

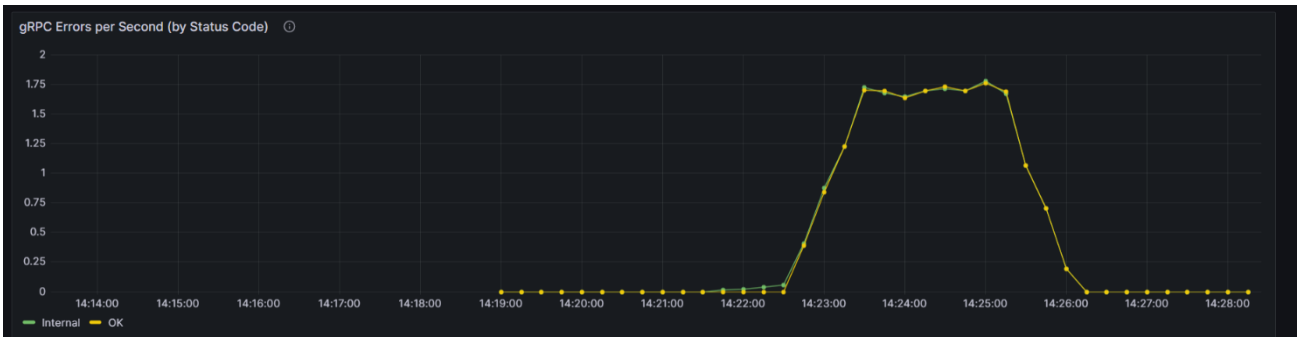
- Traffic
- Errors
- Availability
- (Latency quando instrumentar)

🔴 3) gRPC Errors — Detailed Analysis

👉 Investigação de incidentes

- Errors últimos 5 min
- Errors por status
- Error percentage
- (Latency por método no futuro)

gRPC - PROMETHEUS - GRAFANA



🔍 O que significam **Internal** e **OK** nesse painel?

gRPC Errors per Second (by Status Code)
sum by (status) (
 rate(grpc_requests_total[1m])
)

✅ OK
 Significa **chamadas gRPC que terminaram com sucesso**
 No GrpcServer:
Classe: GreeterService
GrpcRequestsTotal
  .WithLabels("SayHello", "OK")
  .Inc();

A linha **OK** mostra **quantas requisições por segundo estão sendo bem-sucedidas**
OK 1.71
→ ~1.71 req/s com sucesso naquele instante

❌ **Internal**

- Internal é um **StatusCode do gRPC**
- Equivale a um **erro 500 interno** no mundo HTTP

 No GrpcServer:
Classe: GreeterService

throw new RpcException(
  new Status(StatusCode.Internal, "Erro simulado"));
E no catch:
GrpcRequestsTotal
  .WithLabels("SayHello", ex.StatusCode.ToString())
  .Inc();

gRPC - PROMETHEUS - GRAFANA

No gráfico:

- A linha **Internal** mostra **quantos erros internos por segundo estão acontecendo**
- Exatamente o que você simulou quando o client manda Name = "error"

Este repositório é um showcase completo de observabilidade gRPC.
Para testar:

1) docker compose up -d
2) dotnet run --project GrpcServer
3) dotnet run --project GrpcLoadClient
4) Acesse http://localhost:3000 (admin/admin)

Os dashboards começam a mostrar dados em poucos segundos.