Aula 43 – Observabilidade: Monitoramento e Alertas

Imagine que você é o capitão de um navio gigantesco, repleto de compartimentos, motores e sistemas interconectados. Em mar aberto, a última coisa que você quer é uma falha inesperada. Como você saberia se um motor está superaquecendo, se o radar está com interferência ou se há um vazamento em um dos porões, antes que se torne uma crise? No mundo do desenvolvimento de software, especialmente com a ascensão de arquiteturas complexas como microserviços e serverless, essa é a realidade diária de quem gerencia aplicações web.

A complexidade crescente dos sistemas modernos, onde uma única requisição pode passar por dezenas de serviços diferentes, torna o diagnóstico de problemas um verdadeiro desafio. Não é mais suficiente apenas saber se um serviço está "ligado" ou "desligado". Precisamos entender o comportamento interno, as interações e a saúde de cada componente em tempo real. É nesse cenário que a observabilidade se torna não apenas uma ferramenta, mas uma filosofia essencial para garantir a estabilidade, performance e resiliência de nossas aplicações.

Nesta aula, embarcaremos juntos na jornada da observabilidade. Nosso objetivo é que, ao final, você seja capaz de compreender os pilares que sustentam sistemas observáveis, aprender a coletar métricas vitais de suas aplicações e infraestrutura usando ferramentas como o Prometheus, e transformar esses dados brutos em insights acionáveis através de dashboards visuais no Grafana. Além disso, você aprenderá a configurar alertas proativos, garantindo que você seja o primeiro a saber quando algo não está conforme o esperado, muito antes que seus usuários percebam. Prepare-se para desvendar os segredos por trás de sistemas robustos e proativos.

Desvendando a Observabilidade: Além do Monitoramento Tradicional

No passado, quando as aplicações eram monolíticas e rodavam em um único servidor, o monitoramento era relativamente simples. Verificávamos o uso de CPU, memória, disco e se o serviço estava respondendo. Era como olhar para um único medidor de combustível em um carro. Mas o cenário mudou drasticamente. Com a arquitetura de microserviços, por exemplo, temos dezenas ou centenas de "carros" pequenos, cada um com sua função, rodando em diferentes "estradas" e se comunicando constantemente.

Nesse novo paradigma, o monitoramento tradicional se mostra insuficiente. Ele nos diz "o quê" está acontecendo (ex: CPU alta), mas raramente nos explica "por que" ou "como" isso afeta o sistema como um todo. É aqui que a observabilidade entra em cena. Ela não se limita a verificar a saúde superficial, mas busca entender o estado interno de um sistema a partir de seus dados externos, permitindo que façamos perguntas complexas sobre o comportamento do sistema sem precisar depurá-lo diretamente.

Pense na observabilidade como a capacidade de um médico de diagnosticar uma doença complexa. Ele não apenas mede a febre (monitoramento), mas também pede exames de sangue, radiografias, ouve os sintomas do paciente e analisa o histórico (observabilidade). Com essas informações, ele pode entender a causa raiz do problema e prescrever o tratamento correto. Da mesma forma, em sistemas distribuídos, precisamos de uma visão holística e profunda para identificar e resolver problemas rapidamente.

Os Três Pilares da Observabilidade: Métricas, Logs e Traces

Para que um sistema seja verdadeiramente observável, ele precisa expor dados em três categorias principais, que são frequentemente chamadas de "os três pilares da observabilidade". Cada pilar oferece uma perspectiva única e complementar sobre o comportamento do sistema, e a combinação deles nos dá a capacidade de entender o que está acontecendo em qualquer momento.

Métricas

Dados numéricos agregados ao longo do tempo, como a taxa de requisições por segundo, o uso de CPU, a latência média de uma API ou o número de erros. Métricas são ideais para identificar tendências, detectar anomalias e criar dashboards de alto nível que mostram a saúde geral do sistema.

Logs

Registros de eventos discretos que ocorrem em um sistema, geralmente mensagens textuais com carimbos de data e hora. Eles são cruciais para entender o que aconteceu em um ponto específico no tempo, fornecendo detalhes sobre erros, avisos, informações de depuração e eventos de negócio.

Traces

Rastreamentos distribuídos que nos permitem seguir o caminho de uma única requisição através de múltiplos serviços em uma arquitetura distribuída. Cada etapa da requisição é registrada como um "span", e a coleção desses spans forma um trace completo.

Juntos, esses pilares formam a base para uma compreensão profunda e proativa de qualquer sistema. Eles respondem a perguntas como "Quantas requisições estamos recebendo?", "Qual foi a mensagem de erro exata?" e "Onde a requisição passou a maior parte do tempo?".

Métricas: O Pulso Vital do Seu Sistema

As métricas são, sem dúvida, o ponto de partida para a maioria das estratégias de observabilidade. Elas nos oferecem uma visão quantitativa e resumida do comportamento do nosso sistema, permitindo que identifiquemos rapidamente se algo está fora do padrão. Pense nelas como os sinais vitais de um paciente: batimentos cardíacos, pressão arterial, temperatura. Cada um desses números, por si só, pode não contar a história completa, mas juntos, eles pintam um quadro claro da saúde geral.

No contexto de aplicações web, as métricas podem ser tão variadas quanto o número de usuários logados, a quantidade de requisições HTTP por segundo, a latência média de uma chamada de API, o uso de CPU e memória dos servidores, ou até mesmo métricas de negócio, como o número de vendas realizadas. A beleza das métricas reside na sua capacidade de serem agregadas, filtradas e visualizadas em gráficos, tornando padrões e anomalias facilmente detectáveis.

1	2	3
Contador	**Gauge**	**Histogramas e Sumários**
Um valor que só aumenta, como o número total de requisições recebidas.	Um valor que pode subir ou descer, como a temperatura de um servidor ou o número de itens em uma fila.	Mais complexos, usados para amostrar observações (como latências de requisições) e fornecer distribuições estatísticas, permitindo que você veja não apenas a média, mas também percentis.

Entender esses tipos é fundamental para escolher a métrica certa para cada situação.

Introdução ao Prometheus: O Coletor de Métricas Robusto

Com a proliferação de microserviços e a necessidade de coletar métricas de inúmeros pontos em um sistema distribuído, surge a questão: como fazer isso de forma eficiente e escalável? É nesse ponto que o **Prometheus** se destaca como uma das ferramentas mais populares e eficazes do mercado para a coleta e armazenamento de métricas de séries temporais. Ele se tornou um padrão de fato para monitoramento de sistemas em ambientes de nuvem e Kubernetes.

O Prometheus opera com um modelo de **"pull"**, o que significa que ele ativamente busca as métricas dos seus alvos configurados, em vez de esperar que os alvos as enviem (modelo "push"). Essa abordagem simplifica a configuração dos serviços a serem monitorados, pois eles só precisam expor um endpoint HTTP com suas métricas em um formato específico. O Prometheus então se encarrega de descobrir esses serviços, coletar os dados periodicamente e armazená-los em seu banco de dados de séries temporais.

Imagine o Prometheus como um carteiro muito organizado que, em vez de esperar que as pessoas postem suas cartas, vai de casa em casa (seus serviços e servidores) em intervalos regulares, recolhendo as correspondências (métricas) que estão prontas para serem entregues. Ele sabe exatamente onde ir, o que coletar e como armazenar tudo para que você possa consultar depois. Essa metodologia garante que o Prometheus tenha uma visão consistente e atualizada da saúde de todo o seu ecossistema.

Configurando o Prometheus: Dando Vida ao Seu Monitoramento

Para que o Prometheus comece a coletar dados, ele precisa saber o que monitorar e onde encontrar essas informações. Isso é feito através de um arquivo de configuração YAML, geralmente chamado `prometheus.yml`. Este arquivo é o coração da sua instalação do Prometheus, definindo as regras de coleta, os alvos e outras configurações importantes.

A parte mais crucial do `prometheus.yml` são as seções `scrape_configs`. Aqui, você define "jobs" (tarefas de coleta) que instruem o Prometheus sobre quais serviços ele deve "raspar" (scrape) para obter métricas. Cada job pode ter múltiplos targets (alvos), que são os endereços IP ou nomes de host dos seus serviços que expõem métricas.

O Prometheus utiliza um mecanismo de "service discovery" para encontrar esses alvos automaticamente, especialmente útil em ambientes dinâmicos como Kubernetes, onde os IPs dos serviços mudam constantemente.

Exemplo de Configuração

Por exemplo, você pode configurar um job para monitorar todas as suas instâncias de um serviço de autenticação, outro para o banco de dados e um terceiro para os servidores de aplicação. Cada um desses serviços exporia suas métricas em um endpoint como `/metrics`, e o Prometheus faria requisições HTTP para esses endpoints em intervalos regulares (por exemplo, a cada 15 segundos) para coletar os dados.

```
# Exemplo básico de prometheus.yml
global:
  scrape_interval: 15s # Coleta métricas a cada 15 segundos

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090'] # Monitora o próprio Prometheus

  - job_name: 'minha_aplicacao_web'
    static_configs:
      - targets: ['minha-api:8080', 'meu-servico-auth:8081'] # Monitora duas instâncias da sua aplicação
    metrics_path: '/metrics' # Caminho onde as métricas são expostas
```

Essa flexibilidade permite que você adapte o monitoramento às necessidades específicas de cada componente da sua arquitetura.

Métricas de Aplicação vs. Infraestrutura: Onde Focar a Atenção

Ao configurar o monitoramento, é comum surgirem dúvidas sobre quais métricas são as mais importantes. Uma distinção fundamental a ser feita é entre métricas de infraestrutura e métricas de aplicação. Ambas são cruciais, mas fornecem diferentes níveis de granularidade e insights sobre a saúde do seu sistema. Entender a diferença ajuda a construir um painel de observabilidade mais completo e eficaz.

Métricas de Infraestrutura

Focam nos recursos subjacentes que sustentam suas aplicações. Isso inclui o uso de CPU, memória RAM, espaço em disco, taxa de I/O de rede e disco, e a disponibilidade de servidores ou contêineres.

- Uso de CPU
- Memória RAM
- Espaço em disco
- Taxa de I/O de rede

Essenciais para garantir que sua base tecnológica esteja sólida e que não haja gargalos de recursos.

Métricas de Aplicação

Mergulham no comportamento da sua própria aplicação. Elas incluem a taxa de requisições HTTP (RPS), a latência das chamadas de API, o número de erros HTTP (4xx, 5xx), o tempo de processamento de transações de negócio.

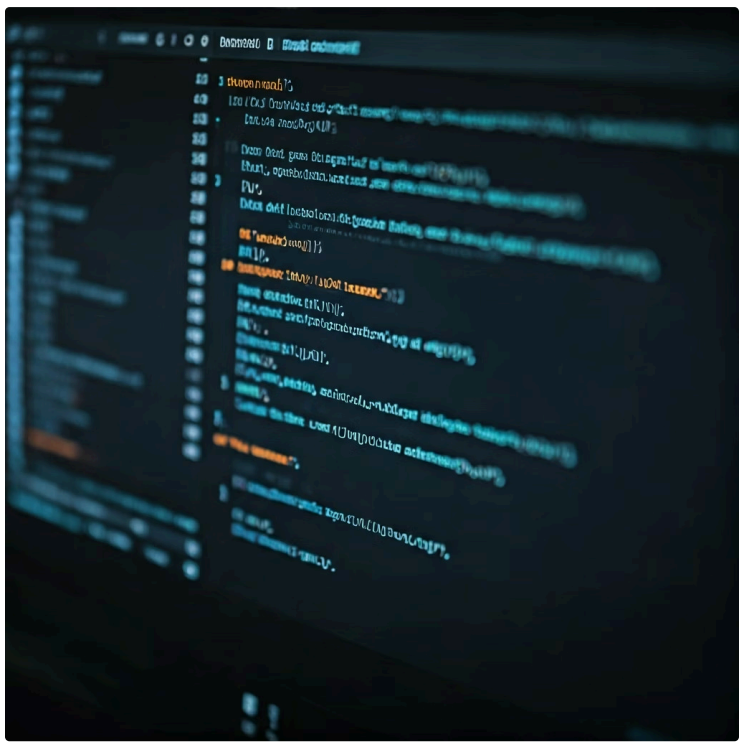
- Taxa de requisições (RPS)
- Latência de API
- Erros HTTP
- Usuários ativos

Vitais para entender a experiência do usuário e a performance real do seu software.

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
Métricas de Infra.	Saúde dos recursos físicos/virtuais	Sistema operacional, hypervisor, orquestrador	Uso de CPU, Memória, Disco, Rede
Métricas de Aplicação	Comportamento e performance do software	Código da aplicação, frameworks, bibliotecas	Latência de API, Taxa de Erros HTTP, RPS, Usuários Ativos

Ambos os tipos de métricas são complementares. As métricas de infraestrutura fornecem o contexto do ambiente, enquanto as métricas de aplicação revelam o impacto direto no negócio. Ignorar um em favor do outro é como tentar dirigir um carro olhando apenas para o velocímetro ou apenas para o nível de combustível; você precisa de ambos para uma viagem segura e eficiente.

PromQL: A Linguagem para Desvendar Suas Métricas



Coletar métricas é o primeiro passo, mas para transformá-las em insights acionáveis, precisamos de uma maneira de consultá-las, agregá-las e manipulá-las. É para isso que existe o **PromQL (Prometheus Query Language)**, a poderosa linguagem de consulta do Prometheus. Com o PromQL, você pode extrair informações específicas do seu banco de dados de séries temporais, criar gráficos complexos e definir condições para alertas.

Pense no PromQL como o SQL para dados de tempo. Assim como o SQL permite que você faça perguntas complexas a um banco de dados relacional, o PromQL permite que você faça perguntas sobre o comportamento do seu sistema ao longo do tempo. Ele é otimizado para trabalhar com séries temporais, o que significa que você pode facilmente calcular taxas de mudança, médias móveis, somas e outras operações que são cruciais para entender a dinâmica do seu sistema.

A sintaxe do PromQL é baseada em seletores de série temporal, que permitem filtrar métricas por nome e por rótulos (labels). Por exemplo, `http_requests_total{job="minha_aplicacao_web", status="200"}` selecionaria todas as métricas de requisições HTTP bem-sucedidas da sua aplicação. A partir daí, você pode aplicar funções e operadores.

Taxa de Requisições

Quer saber a taxa de requisições por segundo nos últimos 5 minutos?

`rate(http_requests_total[5m])`

Agregação por Endpoint

Quer somar as requisições por endpoint?

`sum by (endpoint) (http_requests_total)`

Uso de CPU

Quer calcular o uso médio de CPU?

`avg(node_cpu_seconds_total{mode="idle"}[1h])`

```
# Exemplos de queries PromQL

# Taxa de requisições HTTP por segundo nos últimos 5 minutos
rate(http_requests_total{job="minha_aplicacao_web"}[5m])

# Uso médio de CPU em todos os servidores nos últimos 1 hora
avg(node_cpu_seconds_total{mode="idle"}[1h])

# Número de erros 5xx por serviço
sum by (service) (http_requests_total{status=~"5.."})
```

Dominar o PromQL é a chave para extrair o máximo valor dos dados coletados pelo Prometheus e construir dashboards e alertas inteligentes.

Introdução ao Grafana: Visualizando o Universo de Dados

Ter um tesouro de métricas coletadas pelo Prometheus é fantástico, mas dados brutos, por mais valiosos que sejam, não são facilmente interpretáveis. É como ter um mapa do tesouro escrito em código. Precisamos de uma ferramenta que transforme esses números em algo visualmente compreensível, permitindo que padrões, tendências e anomalias saltem aos olhos. É exatamente isso que o **Grafana** faz.

O Grafana é uma plataforma de código aberto para visualização e análise de dados de séries temporais. Ele atua como a interface gráfica para seus dados, permitindo que você crie dashboards interativos e dinâmicos a partir de diversas fontes de dados, incluindo o Prometheus. Com o Grafana, você pode transformar suas queries PromQL em gráficos de linha, gráficos de barras, tabelas, medidores e muitos outros tipos de visualização, tudo em um único painel de controle.

Pense no Grafana como o painel de controle de um avião. O piloto não precisa entender cada sensor individualmente ou decifrar linhas de código. Ele precisa de medidores claros, gráficos de altitude e velocidade, e luzes de advertência que mostrem o estado geral da aeronave de forma intuitiva. Da mesma forma, o Grafana permite que você construa painéis personalizados que fornecem uma visão clara e concisa da saúde e performance de suas aplicações, tornando a tomada de decisão muito mais rápida e informada.

Criando Dashboards no Grafana: Sua Central de Comando

Com o Grafana como sua ferramenta de visualização, o próximo passo é construir dashboards que realmente contem a história dos seus dados. Um dashboard eficaz não é apenas uma coleção aleatória de gráficos; é uma narrativa visual que guia o observador através das informações mais importantes, permitindo que ele entenda rapidamente o estado do sistema e identifique áreas que precisam de atenção.

01

Criar Novo Dashboard

Você começa criando um novo dashboard no Grafana.

02

Adicionar Painéis

Em seguida, adiciona "painéis" (panels) a ele. Cada painel é uma visualização individual.

03

Configurar Fonte de Dados

Você seleciona sua fonte de dados (neste caso, o Prometheus).

04

Escrever Query PromQL

Escreve sua query PromQL para buscar os dados desejados.

05

Escolher Visualização

Escolhe o tipo de visualização que melhor representa a informação.

Exemplo de Organização

Por exemplo, você pode ter um painel mostrando a taxa de requisições por segundo (RPS) da sua API principal, outro exibindo a latência média das requisições, um terceiro com o número de erros HTTP 5xx, e um quarto com o uso de CPU dos servidores. Ao organizar esses painéis de forma lógica, você cria uma "central de comando" que permite monitorar a performance e a saúde da sua aplicação em tempo real.

A capacidade de customizar cada detalhe, desde as cores até os eixos, garante que seus dashboards sejam não apenas funcionais, mas também esteticamente agradáveis e fáceis de ler.

Dashboards Eficazes: Dicas para uma Visualização Clara

Criar um dashboard é fácil, mas criar um *dashboard eficaz* é uma arte. Um bom dashboard deve ser como um mapa de trânsito: ele mostra rapidamente onde estão os problemas (congestionamentos) e permite que você tome decisões rápidas. Um dashboard ruim, por outro lado, é como um mapa turístico cheio de detalhes irrelevantes quando você está com pressa. Para evitar a "fadiga de dashboard" e garantir que suas visualizações sejam realmente úteis, algumas práticas são essenciais.

Princípio KISS

Keep It Simple, Stupid. Evite poluir seu dashboard com informações desnecessárias. Cada painel deve ter um propósito claro e responder a uma pergunta específica. Priorize as métricas mais importantes para a saúde e performance da sua aplicação, como as "Golden Signals" (Latência, Tráfego, Erros, Saturação).

Variáveis e Templates

O Grafana permite criar variáveis que podem ser usadas nas suas queries PromQL. Isso significa que você pode ter um único dashboard que serve para monitorar diferentes serviços, ambientes (produção, staging) ou instâncias, apenas selecionando a opção desejada em um menu suspenso. Isso economiza tempo e garante consistência.

Pense na Audiência

Um dashboard para desenvolvedores pode incluir métricas de baixo nível, como contadores de exceções. Um dashboard para gerentes de produto pode focar em métricas de negócio, como usuários ativos ou vendas. Adapte a complexidade e o tipo de informação ao público-alvo.

Um dashboard bem projetado é uma ferramenta poderosa para comunicação e colaboração dentro da equipe.

Alertas: Ação Proativa para Manter a Estabilidade

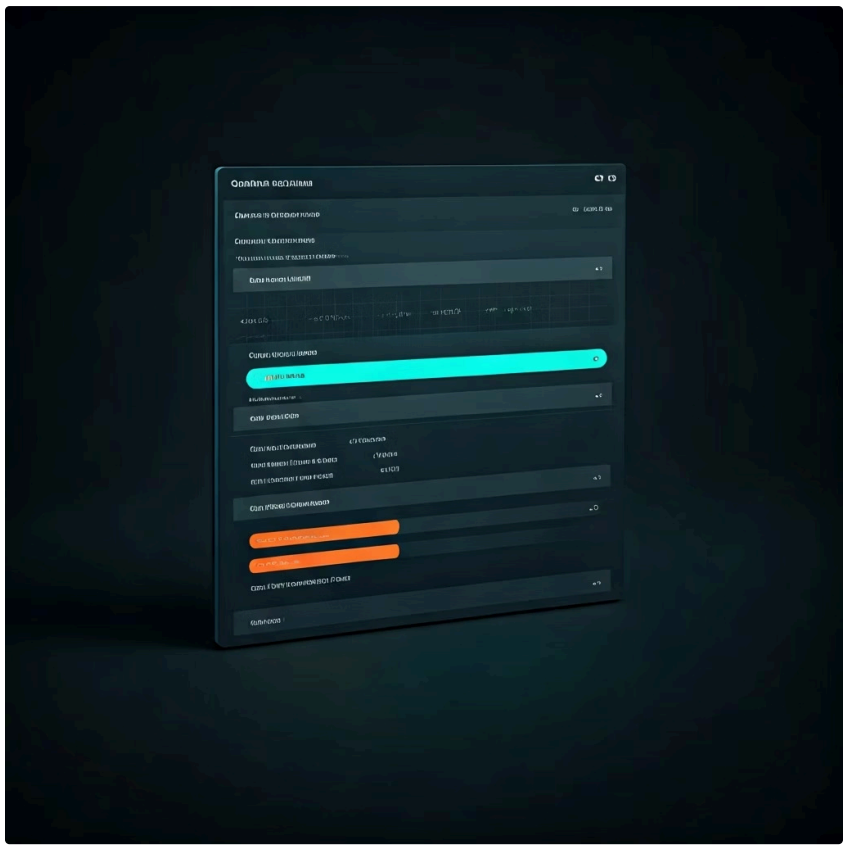
Monitorar seus sistemas e visualizar seus dados em dashboards é um passo gigantesco, mas a observabilidade não termina aí. O que acontece se você não estiver olhando para o dashboard no momento exato em que um problema surge? É aqui que os **alertas** se tornam indispensáveis. Alertas são notificações automáticas que informam você e sua equipe quando certas condições predefinidas são atendidas, indicando um possível problema ou uma anomalia que requer atenção.

A importância dos alertas reside na sua capacidade de transformar uma postura reativa em uma postura proativa. Em vez de esperar que os usuários relatem um problema (o que já é tarde demais), os alertas permitem que sua equipe seja a primeira a saber que algo está errado. Isso reduz drasticamente o MTTR (Mean Time To Recovery - Tempo Médio para Recuperação), minimizando o impacto de incidentes e garantindo a continuidade do serviço.

Pense nos alertas como o alarme de incêndio da sua casa. Você não fica olhando para o detector de fumaça o tempo todo. Ele está lá, silenciosamente monitorando, e só dispara quando detecta uma condição crítica. Da mesma forma, um alerta pode ser configurado para disparar quando a latência da sua API excede um limite por um determinado período, quando a taxa de erros 5xx atinge um pico, ou quando o uso de CPU de um servidor permanece alto por muito tempo. Configurar alertas inteligentes é crucial para a resiliência de qualquer aplicação moderna.

Configurando Alertas no Grafana: Transformando Dados em Notificações

Com o Grafana, não apenas visualizamos dados, mas também podemos definir regras de alerta diretamente a partir dos painéis dos nossos dashboards. Isso significa que a mesma query PromQL que você usa para exibir um gráfico pode ser usada para definir uma condição de alerta, garantindo consistência e simplificando o gerenciamento.



Para configurar um alerta no Grafana, você geralmente seleciona um painel existente e adiciona uma regra de alerta a ele. Essa regra consiste em uma ou mais condições baseadas em queries PromQL. Por exemplo, você pode definir uma condição que dispara um alerta se a `rate(http_requests_total{status=~"5.."}[5m])` (taxa de erros 5xx nos últimos 5 minutos) for maior que 0.1 (ou seja, 10% de erros) por mais de 1 minuto. Você também pode definir a severidade do alerta (info, warning, critical) e mensagens personalizadas.

Definir Condição

Configure a query PromQL e os limites que disparam o alerta

Escolher Severidade

Defina se é info, warning ou critical

Configurar Canal

Selecione onde enviar: email, Slack, PagerDuty, etc.

Canais de Notificação

Após definir a condição, você precisa configurar os **canais de notificação**. O Grafana suporta uma vasta gama de canais, como e-mail, Slack, PagerDuty, Opsgenie, Webhooks genéricos e muitos outros. Isso permite que você direcione os alertas para as ferramentas de comunicação que sua equipe já utiliza, garantindo que a pessoa certa seja notificada no momento certo.

A chave é criar alertas que sejam acionáveis e que evitem a "fadiga de alertas", onde a equipe é sobrecarregada com notificações irrelevantes, levando à ignorância de alertas importantes.

Gerenciamento de Incidentes e a Cultura de Alertas

Configurar alertas é apenas o primeiro passo para uma estratégia robusta de observabilidade. O verdadeiro valor surge quando esses alertas são integrados a um processo eficaz de gerenciamento de incidentes. Um alerta disparado é o início de um ciclo que envolve detecção, resposta, resolução e aprendizado. Sem um plano claro, mesmo os melhores alertas podem ser ineficazes.

Detecção

Alerta é disparado quando condição crítica é identificada

Resposta

Equipe on-call é notificada e inicia investigação

Aprendizado

Post-mortem analisa causa raiz e previne recorrência

Resolução

Problema é identificado e corrigido seguindo runbooks

Durante o Incidente

- Identificar quem é o responsável (on-call)
- Seguir runbooks com passos de investigação
- Escalar se necessário
- Comunicar status aos stakeholders

Após o Incidente

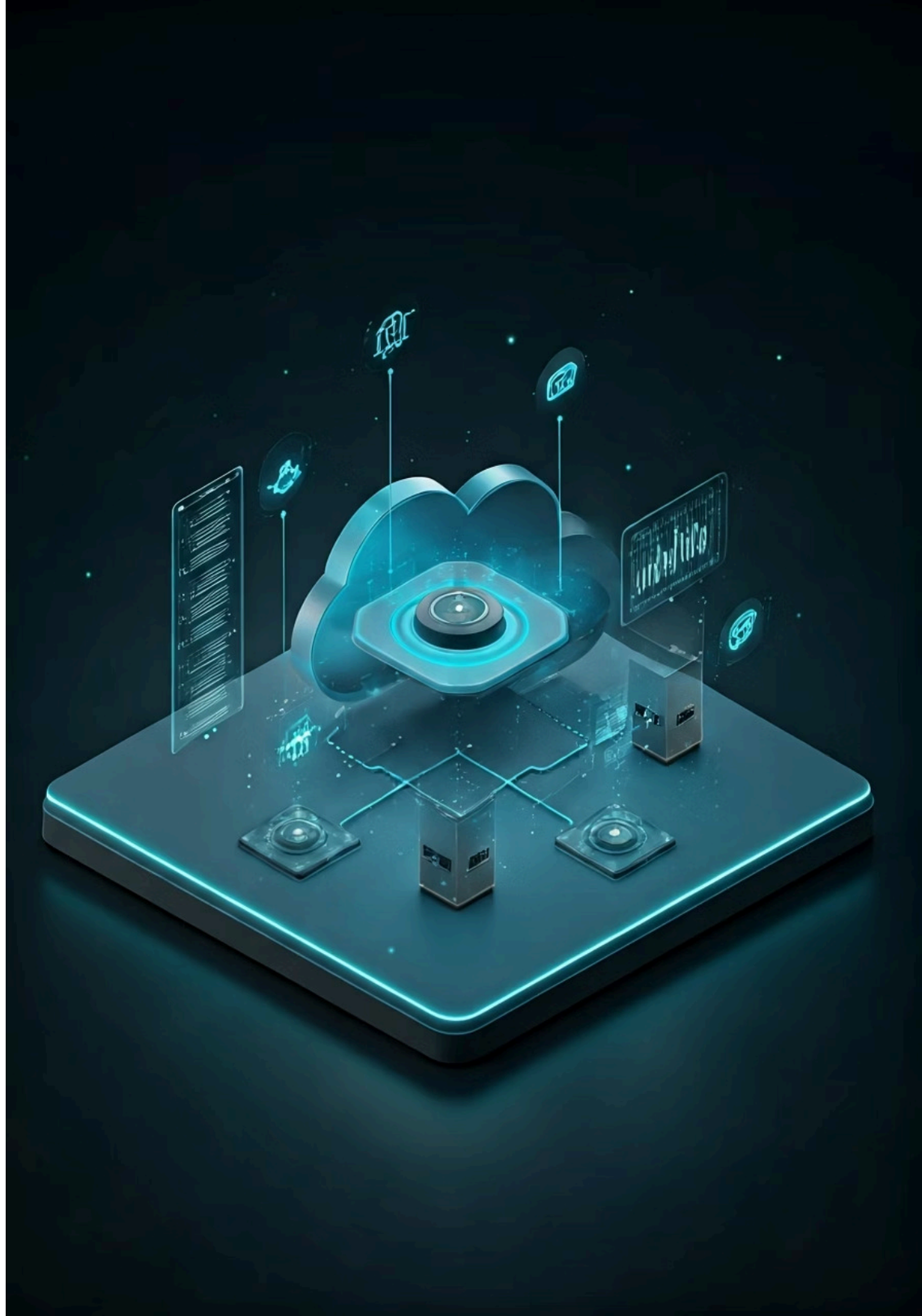
- Realizar post-mortem sem culpa
- Documentar causa raiz
- Aprimorar alertas e instrumentação
- Atualizar runbooks

Essa mentalidade de aprendizado contínuo é um pilar da cultura DevOps e SRE (Site Reliability Engineering), transformando incidentes em oportunidades de melhoria.

Observabilidade em Arquiteturas Modernas: Tendências e Futuro

A observabilidade não é um conceito estático; ela evolui constantemente para acompanhar as complexidades das arquiteturas de software modernas. Com a adoção massiva de microserviços, arquiteturas serverless e orquestração de contêineres como Kubernetes, a necessidade de ferramentas e abordagens de observabilidade mais sofisticadas se tornou ainda mais premente.

Em ambientes de microserviços, por exemplo, a capacidade de rastrear uma requisição através de dezenas de serviços diferentes (com traces distribuídos) é fundamental. O Prometheus e o Grafana se integram perfeitamente com Kubernetes, permitindo o monitoramento automático de pods, nós e serviços, utilizando o service discovery para encontrar e coletar métricas de forma dinâmica. Isso é crucial para gerenciar a escala e a efemeridade dos contêineres.



OpenTelemetry

Um conjunto de ferramentas, APIs e SDKs que padronizam a forma como os dados de telemetria (métricas, logs e traces) são gerados e coletados, independentemente do fornecedor. Isso promete simplificar a instrumentação de aplicações e evitar o "vendor lock-in".

AIOps

Inteligência Artificial para Operações busca usar machine learning para analisar grandes volumes de dados de observabilidade, identificar padrões, prever problemas e até mesmo automatizar respostas, levando a uma observabilidade ainda mais inteligente e proativa.

Olhando para o futuro, essas tendências prometem tornar a observabilidade ainda mais poderosa e acessível, permitindo que equipes gerenciem sistemas cada vez mais complexos com maior confiança e eficiência.

Consolidação e Próximos Passos

Chegamos ao fim da nossa jornada pela observabilidade, um campo essencial para qualquer arquiteto ou desenvolvedor que trabalha com sistemas modernos e distribuídos. Vimos que a observabilidade vai além do monitoramento básico, buscando uma compreensão profunda do estado interno de um sistema através de seus dados externos. Exploramos os três pilares – métricas, logs e traces – e mergulhamos no uso prático de ferramentas líderes de mercado como Prometheus para coleta de métricas e Grafana para visualização e alertas.

Em prática:

1. Sempre comece instrumentando suas aplicações com métricas claras e significativas.
2. Utilize o Prometheus para coletar essas métricas de forma eficiente em ambientes distribuídos.
3. Crie dashboards no Grafana que contem uma história clara sobre a saúde do seu sistema.
4. Configure alertas inteligentes para ser proativo na detecção e resolução de problemas.
5. Adote uma cultura de gerenciamento de incidentes e aprendizado contínuo.

Autoavaliação

1

Questão 1

Qual dos seguintes não é considerado um dos três pilares da observabilidade?

a) Métricas
b) Logs
c) Backups
d) Traces

2

Questão 2

Qual ferramenta é amplamente utilizada para a coleta e armazenamento de métricas de séries temporais, operando com um modelo de "pull"?

a) Grafana
b) Kibana
c) Prometheus
d) Jaeger

3

Questão 3

No contexto de métricas, qual tipo de valor pode subir ou descer, sendo ideal para representar a temperatura de um servidor ou o número de itens em uma fila?

a) Contador
b) Gauge
c) Histograma
d) Sumário

4

Questão 4

Qual das seguintes afirmações melhor descreve a função do Grafana?

a) É um banco de dados para armazenar logs de aplicações.
b) É uma ferramenta para orquestração de contêineres.
c) É uma plataforma para visualização e análise de dados de séries temporais.
d) É uma linguagem de programação para desenvolvimento web.

5

Questão 5

Explique a importância dos alertas em uma estratégia de observabilidade e como eles contribuem para a resiliência de uma aplicação.

Gabarito: 1. c) | 2. c) | 3. b) | 4. c)