Aula 26 – Observabilidade: Métricas e Monitoramento

Imagine que você está dirigindo um carro de alta performance, mas com o painel de instrumentos completamente apagado. Você não sabe a velocidade, o nível de combustível, a temperatura do motor ou se há alguma luz de advertência acesa. Seria uma experiência de direção arriscada e, provavelmente, de curta duração, não é mesmo? No mundo do desenvolvimento de software, especialmente com a complexidade crescente das arquiteturas de microserviços e a adoção massiva de contêineres e orquestradores como Kubernetes, operar "às cegas" é igualmente perigoso.

A observabilidade surge como a capacidade de entender o estado interno de um sistema complexo a partir de seus dados externos. Ela nos permite "ligar o painel" e ver o que está acontecendo dentro de nossas aplicações e infraestrutura, mesmo quando não sabemos exatamente o que procurar. É a diferença entre reagir a um problema quando ele já causou um impacto significativo e ser proativo, detectando anomalias antes que se tornem falhas catastróficas.

Nesta aula, vamos desvendar os pilares da observabilidade focando especificamente em **métricas** e **monitoramento**. Você aprenderá a coletar informações cruciais sobre a saúde e a performance dos seus serviços, e como utilizar ferramentas poderosas como Prometheus e Grafana para visualizar esses dados e configurar alertas inteligentes. Ao final, você será capaz de aplicar esses conceitos para construir sistemas mais robustos, confiáveis e fáceis de gerenciar, uma habilidade indispensável no cenário tecnológico de 2025.

O Coração da Observabilidade: Entendendo as Métricas

Em um sistema distribuído moderno, como uma arquitetura de microserviços rodando em contêineres orquestrados por Kubernetes, a quantidade de eventos e interações é gigantesca. Tentar entender o que está acontecendo apenas com logs pode ser como procurar uma agulha em um palheiro. É aqui que as métricas entram em cena, oferecendo uma visão quantitativa e agregada do comportamento do sistema, permitindo-nos identificar tendências e anomalias rapidamente.

> Pense nas métricas como os sinais vitais de um paciente em uma UTI. O batimento cardíaco, a pressão arterial, a temperatura corporal – cada um desses dados, coletado e monitorado continuamente, oferece uma indicação clara da saúde geral do indivíduo.

As métricas são pontos de dados numéricos coletados em intervalos regulares ao longo do tempo. Elas são otimizadas para armazenamento e consulta eficientes, permitindo a criação de gráficos e dashboards que revelam padrões e comportamentos. Diferente de logs, que registram eventos específicos, as métricas fornecem uma visão estatística e agregada, ideal para monitoramento contínuo e detecção de tendências.

Tipos Comuns de Métricas

Para coletar informações relevantes, precisamos entender os diferentes tipos de métricas que podemos instrumentar em nossas aplicações e infraestrutura. Cada tipo serve a um propósito específico e nos ajuda a contar uma parte diferente da história do nosso sistema.

Contadores (Counters)

São métricas que só podem aumentar ou ser resetadas para zero. Ideais para contar eventos, como o número total de requisições HTTP recebidas, erros ocorridos ou tarefas concluídas.

Medidores (Gauges)

Representam um valor numérico que pode subir e descer arbitrariamente. Perfeitos para medir valores atuais, como o uso de CPU, a quantidade de memória livre, o número de usuários ativos ou o tamanho de uma fila de mensagens.

Histogramas (Histograms)

Amostram observações (como latências de requisição ou tamanhos de resposta) e as agrupam em buckets configuráveis. Além disso, fornecem a soma de todas as observações e a contagem de observações, permitindo calcular médias e percentis.

Resumos (Summaries)

Semelhantes aos histogramas, mas calculam quantis (percentis) no lado do cliente. Úteis para obter informações sobre a distribuição de valores, como o 99º percentil da latência de uma API, que indica que 99% das requisições foram processadas em um tempo menor ou igual a esse valor.

Coletando Métricas: Onde e Como Extrair os Sinais Vitais

Ter um painel de instrumentos é inútil se os sensores não estiverem conectados. No contexto da observabilidade, a coleta de métricas é o processo de instrumentar nossos sistemas para extrair esses "sinais vitais" de forma contínua e confiável. Isso envolve decidir o que medir, onde medir e como transportar esses dados para um local onde possam ser armazenados e analisados.

> **Visão Holística:** A coleta de métricas não se limita apenas ao código da sua aplicação. Ela abrange a infraestrutura subjacente, o sistema operacional, a rede e até mesmo os serviços de terceiros que você utiliza. Em um ambiente de contêineres, por exemplo, é crucial monitorar não apenas a aplicação dentro do Docker, mas também o próprio contêiner (uso de CPU/memória) e o nó do Kubernetes onde ele está rodando.

Existem duas abordagens principais para a coleta de métricas: o modelo **push** e o modelo **pull**. No modelo push, a aplicação ou o agente envia as métricas para um servidor central. Já no modelo pull, o servidor de monitoramento "puxa" as métricas de endpoints expostos pelas aplicações ou agentes. A escolha entre eles depende do cenário, mas o modelo pull, popularizado por ferramentas como o Prometheus, oferece vantagens em ambientes dinâmicos e efêmeros, como os baseados em Kubernetes, onde os serviços podem surgir e desaparecer rapidamente.

Instrumentação e Exposição de Métricas

Instrumentação de Código

Para que as métricas possam ser coletadas, elas precisam ser expostas de alguma forma. A **instrumentação** é o processo de adicionar código à sua aplicação para gerar essas métricas. Muitas linguagens e frameworks oferecem bibliotecas que simplificam esse processo.

Por exemplo, em Java com Spring Boot, o Spring Boot Actuator pode expor métricas automaticamente em um endpoint /actuator/prometheus, seguindo o formato esperado pelo Prometheus.

Exporters

Além da instrumentação de código, utilizamos **exporters** para coletar métricas de sistemas que não podem ser instrumentados diretamente. Um exporter é um pequeno serviço que traduz métricas de um sistema específico (como um banco de dados, um sistema operacional ou um servidor web) para um formato que o sistema de monitoramento possa entender.

Exemplo Prático

Imagine que você tem um microserviço em Python que processa pedidos. Você pode instrumentá-lo para expor métricas como:

pedidos_total_processados_counter

Um contador para o número total de pedidos processados.

pedidos_erros_total_counter

Um contador para o número de erros ao processar pedidos.

pedidos_latencia_ms_histogram

Um histograma para a latência de processamento de cada pedido.

Essas métricas seriam expostas em um endpoint HTTP (e.g., /metrics) que um coletor de métricas poderia acessar.

Prometheus: O Guardião das Métricas do Futuro

Com as métricas devidamente instrumentadas e expostas, precisamos de um sistema robusto para armazená-las, consultá-las e, o mais importante, dar sentido a elas. É aqui que o **Prometheus** entra em cena. Nascido no SoundCloud e agora um projeto de código aberto da Cloud Native Computing Foundation (CNCF), o Prometheus se tornou o padrão de fato para monitoramento de sistemas em ambientes de contêineres e microserviços.

O Prometheus é um sistema de monitoramento e alerta de código aberto que coleta métricas de alvos configurados em intervalos definidos, avalia regras de expressão, exibe os resultados e pode acionar alertas se certas condições forem atendidas. Sua arquitetura é baseada no modelo "pull", onde o servidor Prometheus periodicamente "raspa" (scrape) os endpoints /metrics de seus alvos, como seus microserviços, exporters ou até mesmo o próprio Kubernetes.

> A grande força do Prometheus reside em sua base de dados de séries temporais (Time-Series Database - TSDB) otimizada para métricas, e sua poderosa linguagem de consulta, o PromQL.

Componentes Chave do Ecossistema Prometheus

O Prometheus não é apenas um único software, mas um ecossistema de ferramentas que trabalham em conjunto para fornecer uma solução completa de monitoramento:

Servidor Prometheus

O componente principal que raspa e armazena as métricas.

Exporters

Pequenos serviços que expõem métricas de sistemas de terceiros (como bancos de dados, sistemas operacionais) em um formato compatível com Prometheus.

Pushgateway

Um intermediário para serviços de curta duração que não podem ser raspados diretamente pelo Prometheus (pois não ficam online tempo suficiente). Eles "empurram" suas métricas para o Pushgateway, que então é raspado pelo servidor Prometheus.

Alertmanager

Lida com os alertas enviados pelo servidor Prometheus, agrupando-os, deduplicando-os e roteando-os para os canais de notificação corretos (e-mail, Slack, PagerDuty, etc.).

Exemplo de PromQL

Para saber o número médio de requisições HTTP por segundo nos últimos 5 minutos para um serviço específico, você poderia usar uma query como:

```
rate(http_requests_total{job="meu-servico"}[5m])
```

Esta query calcula a taxa de aumento do contador http_requests_total para o serviço meu-servico ao longo dos últimos 5 minutos, fornecendo uma métrica de "requisições por segundo".

Prometheus em Ação: Configuração e Exporters Essenciais

Configurar o Prometheus para começar a coletar métricas é um passo fundamental para qualquer estratégia de observabilidade. A beleza do Prometheus está na sua simplicidade de configuração, baseada em um arquivo YAML, e na vasta gama de exporters disponíveis que permitem monitorar praticamente qualquer tipo de sistema, desde um servidor Linux até um banco de dados PostgreSQL ou um cluster Kafka.

A configuração principal do Prometheus é feita através do arquivo `prometheus.yml`. Nele, você define os trabalhos de raspagem (scrape_configs), que informam ao Prometheus quais alvos ele deve monitorar e em quais intervalos. Cada trabalho de raspagem especifica um conjunto de alvos e um nome para o trabalho, o que ajuda a organizar e identificar as métricas coletadas.

> **Exporters Cruciais:** Em um ambiente de microserviços e contêineres, os exporters se tornam ainda mais cruciais. Eles são a ponte entre o Prometheus e os componentes da sua infraestrutura que não expõem métricas nativamente no formato Prometheus.

Configurando o Prometheus para Raspar Alvos

Vamos ver um exemplo simplificado de como configurar o prometheus.yml para monitorar um microserviço e um Node Exporter:

```
global:
  scrape_interval: 15s  # Raspar métricas a cada 15 segundos

scrape_configs:
  - job_name: 'prometheus'  # O próprio Prometheus se monitora
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'meu-microservico'  # Monitora um microserviço
    static_configs:
      - targets: ['meu-servico:8080']  # Endereço do microserviço, expondo /metrics

  - job_name: 'node-exporter'  # Monitora um servidor com Node Exporter
    static_configs:
      - targets: ['servidor-app:9100']  # Endereço do Node Exporter
```

Node Exporter

Para monitorar o uso de recursos de um servidor (CPU, memória, disco).

cAdvisor

Integrado ao Kubelet, fornece métricas sobre os contêineres e pods no Kubernetes.

Kube-state-metrics

Fornece métricas sobre o estado dos objetos do Kubernetes.

Neste exemplo, o Prometheus está configurado para raspar três alvos: ele mesmo, um microserviço na porta 8080 e um Node Exporter na porta 9100. Em um ambiente Kubernetes, a descoberta de serviços (service discovery) é ainda mais dinâmica, permitindo que o Prometheus encontre automaticamente novos pods e serviços sem a necessidade de configurar cada um manualmente.

A adoção de contêineres com Docker e a orquestração com Kubernetes tornaram a configuração de monitoramento mais padronizada. Muitos frameworks, como Spring Boot, já vêm com módulos de Actuator que expõem métricas no formato Prometheus, facilitando a integração. Essa padronização é um dos pilares da observabilidade moderna, permitindo que as equipes se concentrem em construir e operar, em vez de reinventar a roda do monitoramento.

Visualizando o Invisível: A Magia do Grafana

Coletar e armazenar métricas é apenas metade da batalha. Para que esses dados se transformem em informações acionáveis, precisamos de uma forma eficaz de visualizá-los. É aqui que o **Grafana** brilha. O Grafana é uma plataforma de código aberto para análise e visualização de dados, que permite criar dashboards interativos e altamente personalizáveis a partir de diversas fontes de dados, incluindo o Prometheus.

> Pense no Grafana como o painel de controle definitivo para o seu carro de alta performance. Enquanto o Prometheus é o motor e os sensores que coletam os dados, o Grafana é a interface elegante e intuitiva que transforma esses dados brutos em gráficos, medidores e tabelas compreensíveis.

A grande vantagem do Grafana é sua flexibilidade. Ele suporta uma vasta gama de fontes de dados (Prometheus, Loki, Elasticsearch, InfluxDB, bancos de dados SQL, etc.), o que o torna uma ferramenta central para a observabilidade, consolidando diferentes tipos de dados em dashboards unificados. Além disso, a comunidade Grafana é enorme, oferecendo uma infinidade de dashboards pré-construídos que você pode importar e adaptar para suas necessidades.

Conectando Grafana ao Prometheus e Criando Dashboards

O primeiro passo para usar o Grafana com o Prometheus é configurar o Prometheus como uma fonte de dados. Isso é um processo simples na interface do Grafana, onde você informa o endereço do seu servidor Prometheus. Uma vez conectado, você pode começar a construir seus dashboards.

Um dashboard no Grafana é composto por **painéis** (panels), que são os elementos visuais individuais (gráficos de linha, barras, medidores, tabelas, etc.). Cada painel é configurado com uma query específica para a fonte de dados (por exemplo, uma query PromQL para o Prometheus) e opções de visualização.

Exemplo de um Dashboard Essencial

Um dashboard básico para um microserviço poderia incluir painéis para:

Requisições por Segundo (RPS)

Um gráfico de linha mostrando a taxa de requisições HTTP ao longo do tempo.

Latência Média

Um gráfico de linha ou medidor mostrando a latência média das requisições.

Taxa de Erros

Um gráfico de linha ou medidor mostrando a porcentagem de requisições que resultaram em erro.

Uso de CPU e Memória

Gráficos de linha mostrando o consumo de recursos do contêiner ou pod.

Esses painéis, combinados em um dashboard, fornecem uma visão rápida e abrangente da saúde e desempenho do seu serviço, permitindo que você reaja rapidamente a qualquer problema.

Construindo Dashboards Eficazes no Grafana: Transformando Dados em Insights

Ter uma ferramenta poderosa como o Grafana é um excelente começo, mas a verdadeira arte está em construir dashboards que não apenas exibam dados, mas que contem uma história clara e forneçam insights acionáveis. Um dashboard eficaz é como um mapa bem desenhado: ele guia você através da complexidade, destacando os pontos mais importantes e permitindo que você navegue para detalhes quando necessário.

Muitas vezes, a tentação é colocar o máximo de gráficos possível em um único dashboard, resultando em uma "parede de dados" que é difícil de interpretar. No entanto, a chave para um bom dashboard é a **clareza** e a **relevância**. Cada painel deve ter um propósito claro e contribuir para a compreensão da saúde ou desempenho do sistema. Pergunte-se: "Que pergunta este gráfico responde?" Se não houver uma resposta clara, talvez ele não precise estar ali.

Uma boa prática é organizar os dashboards por contexto: um dashboard para a saúde geral da infraestrutura, outro para o desempenho de um serviço específico, e assim por diante. Utilize cores de forma consistente para indicar status (verde para bom, amarelo para atenção, vermelho para crítico) e agrupe métricas relacionadas para facilitar a comparação.

Princípios de Visualização e Painéis Comuns

Ao construir seus painéis no Grafana, considere os seguintes princípios:

1

Contexto é Rei

Sempre forneça contexto para os dados. Um pico de CPU pode ser normal durante um pico de tráfego, mas alarmante em horários de baixo uso.

2

Hierarquia Visual

Use o tamanho e a posição dos painéis para guiar o olhar do usuário para as informações mais críticas primeiro.

3

Simplicidade

Evite gráficos excessivamente complexos. Um gráfico de linha simples pode ser mais eficaz do que um gráfico 3D complicado.

4

Ação

O dashboard deve levar a uma ação ou a uma investigação mais aprofundada. Se você vê um problema, o que você fará a seguir?

Exemplo Prático: Dashboard de Performance de API

Vamos criar um cenário para um painel de monitoramento de uma API de e-commerce:

- **Painel 1 (Gráfico de Linha):** "Taxa de Requisições (RPS)" – Mostra rate(http_requests_total{job="api-ecommerce"}[1m]). Isso indica o volume de tráfego.
- **Painel 2 (Gráfico de Linha):** "Latência P99 da API" – Mostra histogram_quantile(0.99, sum by (le) (rate(http_request_duration_seconds_bucket{job="api-ecommerce"}[5m]))). Isso revela o tempo que 99% das requisições levam, um indicador crucial de experiência do usuário.
- **Painel 3 (Gráfico de Linha):** "Taxa de Erros (5xx)" – Mostra sum(rate(http_requests_total{job="api-ecommerce", code=~"5.."}[1m])) / sum(rate(http_requests_total{job="api-ecommerce"}[1m])) * 100. Isso indica a porcentagem de requisições que falharam.
- **Painel 4 (Medidor):** "Uso de CPU do Pod" – Mostra a porcentagem de CPU utilizada pelo pod da API.

Com esses painéis, você pode rapidamente identificar se o tráfego está alto, se a API está lenta ou com muitos erros, e se o problema está relacionado ao consumo de recursos.

Monitoramento e Alertas: Agindo Antes do Desastre

Visualizar o estado do seu sistema em um dashboard é fundamental, mas o monitoramento reativo – ou seja, só perceber um problema quando você está olhando para o dashboard – não é suficiente em sistemas de produção. Precisamos de um mecanismo que nos avise proativamente quando algo está errado ou prestes a dar errado. É aí que entram o **monitoramento ativo** e os **alertas**.

> Pense em um detector de fumaça em sua casa. Ele não espera que você veja a fumaça ou sinta o cheiro de queimado para avisá-lo. Ele detecta a anomalia (fumaça) e dispara um alarme imediatamente, permitindo que você tome uma ação rápida para evitar um desastre maior.

Os alertas são definidos com base em regras que avaliam as métricas coletadas. Quando uma métrica cruza um determinado limite ou exibe um comportamento inesperado por um período específico, um alerta é disparado. A eficácia de um sistema de alerta reside não apenas em sua capacidade de detectar problemas, mas também em sua inteligência para evitar "ruído" (alertas falsos positivos) e em sua capacidade de rotear as notificações para as pessoas certas, no momento certo.

Prometheus Alertmanager e Alertas no Grafana

Prometheus Alertmanager

No ecossistema Prometheus, o **Alertmanager** é o componente responsável por gerenciar os alertas. Ele recebe os alertas do servidor Prometheus, agrupa alertas semelhantes para evitar sobrecarga de notificações (o que é crucial em incidentes de grande escala), silencia alertas temporariamente e os roteia para os canais de notificação configurados (e-mail, Slack, PagerDuty, etc.).

Alertas no Grafana

O Grafana também possui sua própria funcionalidade de alerta, que permite configurar alertas diretamente nos painéis dos dashboards. Isso é particularmente útil para alertas baseados em limites visuais ou para equipes que já estão acostumadas a trabalhar com o Grafana.

As regras de alerta são definidas no Prometheus usando PromQL. Por exemplo, você pode definir uma regra que dispara um alerta se a taxa de erros HTTP 5xx exceder 5% por mais de 5 minutos.

Exemplo de Regra de Alerta

```
# Exemplo de regra de alerta no Prometheus
groups:
  - name: api-alerts
    rules:
      - alert: HighErrorRate
        expr: sum(rate(http_requests_total{job="api-ecommerce", code=~"5.."}[5m])) /
sum(rate(http_requests_total{job="api-ecommerce"}[5m])) * 100 > 5
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "API de e-commerce com alta taxa de erros 5xx"
          description: "Mais de 5% das requisições para a API de e-commerce estão retornando erros 5xx nos últimos
5 minutos."
```

A combinação de Prometheus e Grafana para monitoramento e alertas oferece uma solução poderosa para manter seus sistemas saudáveis e garantir que você seja o primeiro a saber quando algo precisa de atenção.

Métricas e Monitoramento no Mundo Real: Kubernetes e Microserviços

A verdadeira complexidade e o valor da observabilidade se revelam quando aplicamos esses conceitos a ambientes de produção modernos, caracterizados por microserviços, contêineres e orquestração. A adoção de Docker para empacotar aplicações e Kubernetes (K8s) para gerenciar, escalar e automatizar a implantação de contêineres tornou-se um padrão da indústria. No entanto, essa flexibilidade e escalabilidade vêm com o desafio de monitorar um sistema altamente distribuído e dinâmico.

Em um cluster Kubernetes, os pods e serviços são efêmeros, nascendo e morrendo constantemente. Monitorar cada instância individualmente seria inviável. É aqui que as ferramentas de observabilidade como Prometheus e Grafana se integram perfeitamente, fornecendo uma visão agregada e dinâmica do estado do cluster e das aplicações nele contidas. A "Trindade da Observabilidade" – Logs, Métricas e Tracing – torna-se essencial para entender o comportamento desses sistemas complexos.

> **Monitoramento Além da Aplicação:** O monitoramento de métricas em Kubernetes vai além das métricas da aplicação. Precisamos também monitorar o próprio cluster: a saúde dos nós, o uso de recursos pelos pods, o estado dos deployments, serviços e outros objetos do Kubernetes.

Desafios e Melhores Práticas em Sistemas Distribuídos

Monitorar microserviços em Kubernetes apresenta desafios únicos:

Cardinalidade

O grande número de instâncias de serviços e contêineres pode gerar uma quantidade massiva de métricas, exigindo um TSDB eficiente.

Contexto

Entender a relação entre métricas de diferentes serviços para diagnosticar a causa raiz de um problema.

Dinamicidade

Alvos de monitoramento aparecem e desaparecem, exigindo descoberta de serviços automática.

Golden Signals

Para enfrentar esses desafios, algumas melhores práticas são cruciais. Concentre-se nas quatro métricas essenciais para qualquer serviço voltado para o usuário:

Latência

Tempo que leva para uma requisição ser atendida.

Tráfego

Volume de requisições que o serviço está recebendo.

Erros

Taxa de requisições que falham.

Saturação

Quão "cheio" o serviço está (uso de CPU, memória, I/O).

- **Padronização:** Use bibliotecas de instrumentação que sigam padrões (como o formato Prometheus) para facilitar a coleta.
- **Alertas Inteligentes:** Configure alertas que sejam acionáveis e evitem "ruído", focando nos impactos reais ao usuário.

A observabilidade é um pilar da segurança "API-First", garantindo que as APIs, que são a espinha dorsal da comunicação em microserviços, estejam sempre saudáveis e performáticas.

Consolidação e Próximos Passos

Chegamos ao fim da nossa jornada sobre métricas e monitoramento, dois pilares fundamentais da observabilidade em sistemas modernos. Vimos que, em um mundo de microserviços, contêineres e orquestração com Kubernetes, operar "às cegas" é um risco inaceitável. A capacidade de coletar, visualizar e alertar sobre o estado interno de nossos sistemas é o que nos permite construir e manter aplicações robustas, performáticas e confiáveis.

Exploramos o que são métricas, seus diferentes tipos e como instrumentar nossas aplicações e infraestrutura para coletá-las. Mergulhamos no Prometheus, o guardião das métricas, entendendo sua arquitetura pull-based, o PromQL e a importância dos exporters. Em seguida, desvendamos o Grafana, a ferramenta que transforma dados brutos em dashboards visuais e acionáveis, permitindo-nos identificar tendências e anomalias. Por fim, compreendemos a importância dos alertas para uma postura proativa, agindo antes que pequenos problemas se tornem grandes desastres, especialmente no contexto dinâmico do Kubernetes.

Em Prática

Instrumente suas aplicações

Adicione bibliotecas de métricas ao seu código para expor dados de performance e saúde.

Utilize exporters

Monitore sua infraestrutura (servidores, bancos de dados) com exporters Prometheus.

Configure o Prometheus

Defina seus alvos de raspagem e regras de alerta de forma estratégica.

Crie dashboards no Grafana

Desenvolva painéis claros e relevantes que contem a história do seu sistema.

Refine seus alertas

Evite o "ruído" e garanta que os alertas sejam acionáveis e cheguem às pessoas certas.

Autoavaliação

1. Qual das seguintes opções descreve melhor a função de um "Gauge" no contexto de métricas?
   - a) Uma métrica que só pode aumentar ou ser resetada para zero.
   - b) Uma métrica que representa um valor numérico que pode subir e descer arbitrariamente.
   - c) Uma métrica que amostra observações e as agrupa em buckets.
   - d) Uma métrica que calcula quantis no lado do cliente.
2. Em um ambiente de microserviços e Kubernetes, qual ferramenta é primariamente responsável por "raspar" (scrape) métricas de alvos configurados e armazená-las em uma base de dados de séries temporais?
   - a) Grafana
   - b) Alertmanager
   - c) Prometheus
   - d) Docker
3. Qual dos seguintes componentes do ecossistema Prometheus é responsável por agrupar, deduplicar e rotear alertas para os canais de notificação corretos?
   - a) Prometheus Server
   - b) Exporter
   - c) Pushgateway
   - d) Alertmanager
4. Ao construir um dashboard no Grafana, qual das "Golden Signals" de monitoramento se refere ao volume de requisições que um serviço está recebendo?
   - a) Latência
   - b) Tráfego
   - c) Erros
   - d) Saturação
5. Explique a diferença fundamental entre métricas e logs no contexto da observabilidade de sistemas distribuídos, e como cada um contribui para uma visão completa do estado do sistema.

Gabarito e Recursos

Gabarito

Questão 1 b) Uma métrica que representa um valor numérico que pode subir e descer arbitrariamente.	**Questão 2** c) Prometheus
Questão 3 d) Alertmanager	**Questão 4** b) Tráfego

Próxima Aula

Na **Aula 27 – Observabilidade: Tracing Distribuído**, aprofundaremos na terceira perna da Trindade da Observabilidade, explorando como rastrear requisições através de múltiplos serviços em uma arquitetura distribuída, identificando gargalos e falhas com precisão.

Recursos Adicionais

Documentação Oficial do Prometheus

Para aprofundar na configuração e PromQL.

Documentação Oficial do Grafana

Para explorar a criação de dashboards e alertas.

Livro "Site Reliability Engineering" (Google)

Para entender os princípios de monitoramento e observabilidade em larga escala.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.