PUC Minas

Laboratório de Desenvolvimento de Software

Prof. Dr. João Paulo Aramuni

Sumário

- Monitoramento
  - Logging
  - Dashboards
  - Alertas

Monitoramento

- Um princípio importante para garantir a disponibilidade de um microsserviço é seu adequado **monitoramento**.

- Um bom monitoramento tem **três componentes**: um adequado **logging** (registro) de todas as informações importantes e relevantes; interfaces gráficas úteis (**dashboards** de controle) que sejam facilmente compreendidas por todo desenvolvedor na empresa e que reflitam com precisão a saúde dos serviços; e um sistema de **alerta** eficaz e acionável sobre as principais métricas.

Monitoramento

- O logging pertence a cada microsserviço e começa no código-base dele.

- Determinar com precisão quais informações registrar difere para cada serviço, mas o objetivo do logging é bem simples: quando ocorre um erro – mesmo um erro em uma implantação antiga –, o logging é necessário para determinar o que exatamente deu errado e onde as coisas começaram a descambar.

Monitoramento

- Em ecossistemas de microsserviços não se recomenda atribuir versões aos microsserviços, portanto não haverá uma versão precisa à qual se referir para encontrar os erros ou problemas.

- O código é revisado frequentemente; implantações ocorrem muitas vezes por semana, recursos são adicionados constantemente e dependências estão sempre mudando, mas os logs permanecem os mesmos, preservando as informações necessárias para identificar qualquer problema.

Monitoramento

- Certifique-se de que seus **logs** contenham as informações necessárias para detectar possíveis problemas. Isto é fundamental para garantir uma resposta eficaz e rápida em caso de incidentes, permitindo uma análise detalhada e precisa.

- Todas as principais métricas (como utilização de hardware, conexões ao database, respostas e tempos médios de respostas, e o status de endpoints de API) devem ser exibidas graficamente em tempo real em um dashboard de fácil acesso.

Monitoramento

- **Dashboards** são um importante componente para construir um microsserviço pronto para produção e bem monitorado:
  - Eles ajudam a avaliar a saúde de um microsserviço com uma rápida olhada e permitem que desenvolvedores detectem padrões estranho e anomalias que talvez não sejam extremas o suficiente para disparar os limites de alerta.
  - Eles oferecem uma visão consolidada e intuitiva do estado geral do microsserviço, permitindo uma supervisão eficaz de métricas-chave, como desempenho, utilização de recursos e integridade do sistema.

Monitoramento

- Quando usados com sabedoria, os dashboards permitem que desenvolvedores detectem se um microsserviço está ou não funcionando corretamente por meio de uma simples olhada, mas não é necessário que os desenvolvedores observem os dashboards para detectar incidentes e interrupções, e o retorno a versões anteriores estáveis deve ser totalmente automatizado.

- A detecção real de falhas é feita por meio de **alertas**.

Monitoramento

- Todas as principais métricas devem gerar alertas, incluindo (no mínimo) utilização de **CPU e RAM**, o número de descritores de arquivos (file descriptors), o **número de conexões ao database**, o SLA (*Service Level Agreement*) do serviço, solicitações (**requests**) e respostas (**responses**), o **status dos endpoints da API**, **erros** e exceções, a saúde das **dependências do serviço**, informações sobre qualquer database e o número de tarefas que estão sendo processadas (se aplicável).

Monitoramento

- É preciso configurar limites do tipo normal, alerta e crítico para cada uma dessas métricas, e qualquer desvio da norma (isto é, quando os limites de alerta ou crítico forem atingidos) deve disparar um alerta para os desenvolvedores que estão de prontidão para o serviço.

- Os limites devem ser indicativos: altos o suficiente para evitar ruído, mas baixos o suficiente também para detectar qualquer problema real.

Monitoramento

- Os alertas devem ser úteis e acionáveis. Um alerta não acionável não é um alerta útil e é um desperdício de horas de engenharia.

- Cada alerta acionável – isto é, todo alerta, – deve ser acompanhado de um **roteiro**. Por exemplo, se um alerta for disparado em um número alto de exceções de certo tipo, então é preciso haver um roteiro contendo as estratégias de mitigação que qualquer desenvolvedor de prontidão possa usar enquanto tenta resolver o problema.

Monitoramento

- Requisitos de monitoramento:
  - Os requisitos para construir um microsserviço adequadamente monitorado são:
    - **Logging** e rastreamento adequados em toda a pilha;
    - **Dashboards** bem projetados que sejam fáceis de entender e reflitam com precisão a saúde do serviço;
    - **Alertas** eficazes e acionáveis acompanhados de roteiros;
    - Implementar e manter uma rotação das equipes de prontidão.

Monitoramento

- Princípios do monitoramento de microsserviços:
  - A maioria das interrupções em um ecossistema de microsserviços é causada por implantações ruins. A segunda causa mais comum de interrupções é a falta de um adequado **monitoramento**.
  - Se o estado de um microsserviço for desconhecido, se as métricas principais não forem rastreadas, então quaisquer falhas resultantes permanecerão desconhecidas até ocorrer uma real interrupção.

Monitoramento

- Princípios do monitoramento de microsserviços:
  - Quando um microsserviço sofre uma interrupção em consequência da falta de monitoramento, sua **disponibilidade** já foi comprometida.
  - Durante essas interrupções, o tempo de mitigação e o tempo de reparo são prolongados, derrubando ainda mais a disponibilidade do microsserviço: sem informações facilmente acessíveis sobre as métricas principais do microsserviço, os devs ficam no escuro, despreparados para resolver rapidamente o problema.

Monitoramento

- Princípios do monitoramento de microsserviços:
  - É por isso que um monitoramento adequado é essencial: ele fornece à equipe de desenvolvimento todas as informações relevantes sobre o microsserviço.
  - Quando um microsserviço é adequadamente monitorado, seu estado nunca é desconhecido.

Monitoramento

- <u>Princípios do monitoramento de microsserviços:</u>
  - Monitorar um microsserviço pronto para produção requer quatro componentes:
  - I) O primeiro é um adequado *logging* de todas as informações relevantes e importantes, o que permite que os desenvolvedores entendam o estado do microsserviço em qualquer momento do presente ou do passado.

Monitoramento

- <u>Princípios do monitoramento de microsserviços:</u>
  - Monitorar um microsserviço pronto para produção requer quatro componentes:
  - II) O segundo é o uso de bem projetados dashboards que refletem com precisão a saúde do microsserviço e são organizados de forma que qualquer pessoa na empresa seja capaz de ver o dashboard e entender a saúde e status do microsserviço sem dificuldade.

Monitoramento

- <u>Princípios do monitoramento de microsserviços:</u>
  - Monitorar um microsserviço pronto para produção requer quatro componentes:
  - III) O terceiro componente são alertas eficazes e acionáveis sobre todas as métricas principais, uma prática que facilita para os desenvolvedores mitigarem e resolverem problemas com o microsserviço antes que eles causem interrupções.

Monitoramento

- <u>Princípios do monitoramento de microsserviços:</u>
  - Monitorar um microsserviço pronto para produção requer quatro componentes:
  - IV) O componente final é a implementação e a prática de executar um turno de plantão sustentável e responsável por monitorar o microsserviço.
  - Com logging, dashboards, alertas e turnos de plantão eficazes, a disponibilidade do microsserviço pode ser protegida: falhas e erros serão detectados, mitigados e resolvidos antes de derrubarem qualquer parte do ecossistema.

Monitoramento

- Um serviço pronto para produção é adequadamente monitorado quando:
  - Suas métricas principais são identificadas e monitoradas nos âmbitos de servidor, infraestrutura e microsserviço;
  - Ele tem um adequado logging que reflete com precisão os estados passados do microsserviço;
  - Seus dashboards são fáceis de interpretar e contêm todas as métricas principais;

Monitoramento

- Um serviço pronto para produção é adequadamente monitorado quando:
  - Seus alertas são acionáveis e definidos por limites sinalizadores;
  - Há um turno de plantão dedicado responsável por monitorar e responder a quaisquer incidentes e interrupções;
  - Há um procedimento de plantão claro, bem definido e padronizado para tratar incidentes e interrupções.

Monitoramento

- Métricas principais:
  - Antes de iniciar o monitoramento, é importante identificar precisamente **o que** queremos e precisamos monitorar: queremos monitorar um microsserviço, mas o que isso realmente quer dizer?

  - Implantado em dezenas, quando não em centenas de servidores, o comportamento de um microsserviço é a soma de seu comportamento em todas estas instâncias, o que não é algo fácil de quantificar.

Monitoramento

- Métricas principais:
  - O essencial é identificar quais propriedades de um microsserviço são necessárias e suficientes para descrever seu comportamento e então determinar o que as mudanças nessas propriedades nos dizem sobre o status geral e a saúde do microsserviço.

  - Chamaremos estas propriedades de **métricas principais**.

Monitoramento

- Métricas principais:
  - Há dois tipos de métricas principais: métricas de servidor e infraestrutura, e métricas de microsserviço.

  - Métricas de servidor e infraestrutura são as que dizem respeito ao status da infraestrutura e dos servidores nos quais o microsserviço está sendo executado, enquanto as métricas de microsserviço são métricas exclusivas do microsserviço individual.

Monitoramento

- Métricas principais:
  - Separar as métricas principais nestas duas categorias diferentes é importante tanto do ponto de vista organizacional como técnico.

  - Métricas de servidor e infraestrutura em geral afetam mais de um microsserviço: por exemplo, se houver um problema com um servidor particular e o ecossistema de microsserviços compartilhar os recursos de hardware entre vários microsserviços, as métricas de servidor serão relevantes para todas as equipes que tenham um microsserviço implantado naquele servidor.

Monitoramento

- Métricas principais:
  - Da mesma forma, métricas específicas do microsserviço raramente serão aplicáveis ou úteis para alguém que não seja a equipe de desenvolvedores trabalhando nesse microsserviço particular.

  - As equipes devem monitorar ambos os tipos de métricas principais (isto é, todas as métricas relevantes para seu microsserviço), e quaisquer métricas relevantes para os vários microsserviços devem ser monitoradas e compartilhadas entre as equipes adequadas.

Monitoramento

- Métricas principais:
  - As métricas de servidor e infraestrutura que devem ser monitoradas para cada microsserviço são a **CPU** utilizada pelo microsserviço em cada servidor, a **RAM** utilizada pelo microsserviço em cada servidor, as **threads** disponíveis, os descritores de **arquivo** (FD) abertos do microsserviço e o número de conexões com os **databases** usados pelo microsserviço.

Monitoramento

- Métricas principais:
  - Monitorar essas métricas principais deve ser feito de modo que o status de cada métrica seja acompanhado por informações sobre a infraestrutura e o microsserviço.

  - Isso significa que o monitoramento deve ser granular o suficiente para que os desenvolvedores possam ver as métricas principais de seu microsserviço em um servidor particular e em todos os servidores nos quais ele é executado.

Monitoramento

- Métricas principais:
  - Por exemplo, os desenvolvedores devem ser capazes de saber quanta CPU seu microsserviço está usando em um servidor particular e quanta CPU seu microsserviço está usando em todos os servidores nos quais ele é executado.

Monitoramento

- Métricas principais:
  - Aviso: Alguns ecossistemas de microsserviços podem usar aplicações de gerenciamento de cluster (como o Mesos), em que os recursos (CPU, RAM, etc.) são abstraídos do âmbito do servidor.

  - As métricas de servidor não estarão disponíveis da mesma forma para os desenvolvedores nessas situações, mas todas as métricas principais do microsserviço devem continuar sendo monitoradas pela equipe.
    - Apache Mesos: https://mesos.apache.org/

Monitoramento

- Métricas principais:

Monitoramento

- Métricas principais:
  - Determinar as métricas necessárias e suficientes no **âmbito do microsserviço** é um pouco mais complicado, pois elas dependem da linguagem particular na qual o microsserviço foi escrito.

  - Cada linguagem traz sua forma própria e especial de processar tarefas, por exemplo, e esses recursos específicos da linguagem precisam ser monitorados de perto na maioria dos casos.

Monitoramento

- Métricas principais:
  - Além das métricas específicas da linguagem, também precisamos monitorar a **disponibilidade** do serviço, o acordo de nível de serviço (**SLA**) desse serviço, a **latência** (do serviço como um todo e de seus endpoints de API), o **sucesso dos endpoints** de API, as **respostas** e o **tempo médio de resposta** dos endpoints de API, os serviços (**clientes**) dos quais as solicitações de API são originadas (junto aos endpoints para os quais elas enviam solicitações), os **erros** e as **exceções** (tanto tratados como não tratados) e a saúde e o **status das dependências**.

Monitoramento

- Métricas principais:
  - E, o mais importante, todas as métricas principais devem ser monitoradas em todo local em que a aplicação estiver implantada. Isso significa que todo estágio do pipeline de deployment deve ser monitorado.

  - A fase de staging deve ser monitorada de perto para detectar quaisquer problemas antes que uma nova candidata à produção (uma nova versão) seja implantada em servidores que recebam tráfego de produção.

Monitoramento

- Métricas principais:
  - É quase desnecessário dizer que todas as implantações em servidores de produção devem ser monitoradas atentamente tanto na fase de pré-release como em produção.

  - Uma vez identificadas as métricas principais para um microsserviço, o próximo passo é capturar as métricas emitidas por seu serviço. Capture-as, grave seus logs, crie gráficos com elas e gere alertas com base nelas.

Monitoramento

- Métricas principais:
  - Resumo das métricas principais:
    - **Métricas principais de servidor e infraestrutura**:
      - CPU
      - RAM
      - Threads
      - Descritores de arquivo
      - Conexoes com o database

Monitoramento

Monitoramento

- Métricas principais:
  - Resumo das métricas principais:
    - **Métricas principais de microsserviço**:
      - Métricas específicas da linguagem
      - Disponibilidade
      - SLA
      - Latência
      - Sucesso do endpoint
      - Respostas do endpoint
      - Tempos de resposta do endpoint
      - Clientes
      - Erros e exceções
      - Dependências

Monitoramento

Monitoramento

- **Observabilidade vs Monitoramento**
  - Existe uma diferença conceitual entre observabilidade (observability) e monitoramento (monitoring) no contexto de APIs.

  - Embora ambos estejam relacionados à capacidade de entender e gerenciar o desempenho e a disponibilidade de sistemas, eles abordam aspectos diferentes e têm objetivos distintos.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Monitoramento (Monitoring):
    - O monitoramento é mais voltado para a coleta de dados e métricas sobre o estado atual do sistema.
    - Geralmente, envolve o uso de ferramentas específicas para medir e registrar métricas predefinidas, como uso de recursos (CPU, memória), latência, erros e outros indicadores de saúde do sistema.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Monitoramento (Monitoring):
    - O monitoramento é mais **reativo** e é frequentemente usado para detectar problemas e alertar quando algo está fora do normal.
    - Pode ser útil para rastrear tendências ao longo do tempo e tomar decisões informadas com base em dados históricos.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Observabilidade (Observability):
    - A observabilidade está mais relacionada à capacidade de entender o comportamento interno de um sistema por meio da análise de dados gerados por ele.
    - Envolve a coleta e análise de dados diversos e muitas vezes não estruturados, permitindo a compreensão de como os diferentes componentes interagem entre si.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Observabilidade (Observability):
    - Vai além do simples monitoramento de métricas predefinidas e busca proporcionar uma visão mais profunda e contextualizada do sistema.

    - A observabilidade é mais **proativa**, ajudando a entender o "porquê" de determinados comportamentos e a identificar as causas raízes de problemas.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Para APIs, a observabilidade pode envolver a captura e análise de logs, rastreamento de solicitações (tracing), análise de eventos e outras práticas que proporcionem uma visão mais completa do que está acontecendo nos bastidores.

  - O monitoramento, por outro lado, pode se concentrar em métricas mais específicas, como o tempo de resposta das chamadas de API, taxas de erro, etc.

Monitoramento

- **Observabilidade vs Monitoramento**
  - Ambos são importantes em um ambiente de desenvolvimento e operações (**DevOps**) eficaz, pois o monitoramento fornece alertas rápidos sobre problemas imediatos, enquanto a observabilidade oferece insights mais profundos para diagnóstico e resolução de problemas a longo prazo.

  - Idealmente, uma abordagem equilibrada que inclui <u>tanto monitoramento quanto observabilidade</u> é recomendada para garantir um sistema robusto e de alto desempenho.

- Exemplo de ferramenta de monitoramento:
  - https://prometheus.io/ [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://prometheus.io/ [Open source]

Dimensional data

Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.

Powerful queries

PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.

Great visualization

Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.

Efficient storage

Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.

Simple operation

Each server is independent for reliability, relying only on local storage. Written in Go, all binaries are statically linked and easy to deploy.

Precise alerting

Alerts are defined based on Prometheus's flexible PromQL and maintain dimensional information. An alertmanager handles notifications and silencing.

Many client libraries

Client libraries allow easy instrumentation of services. Over ten languages are supported already and custom libraries are easy to implement.

Many integrations

Existing exporters allow bridging of third-party data into Prometheus. Examples: system statistics, as well as Docker, HAProxy, StatsD, and JMX metrics.

- Exemplo de ferramenta de monitoramento:
  - https://www.elastic.co/pt/elasticsearch [Open source]

- Elasticsearch

Para que exatamente eu posso usar o Elasticsearch?

Dados numéricos, textuais, geográficos, estruturados, não estruturados. Todos os tipos de dados são bem-vindos. A busca de texto integral só arranha a superfície de como as empresas no mundo todo estão dependendo do Elasticsearch para encarar inúmeros desafios. Veja uma lista completa de soluções desenvolvidas diretamente com base no Elastic Stack.

Monitoramento de log

Logging rápido, escalável e incansável.

→

Monitoramento de infraestrutura

Monitore e visualize as métricas do sistema.

→

APM

Obtenha insights sobre o desempenho da sua aplicação.

→

Monitoramento sintético

Faça o monitoramento e responda a problemas de disponibilidade.

→

Enterprise Search

Experiências de busca e descoberta para qualquer caso de uso.

→

Mapas

Explore dados de localização em tempo real.

→

SIEM (Fundamentos do Elastic Security: SIEM)

Investigação interativa e detecção automatizada de ameaças.

→

Segurança de endpoint

Bloqueie, detecte, intercepte e responda a ameaças.

→

- Exemplo de ferramenta de monitoramento:
  - https://grafana.com/ [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://grafana.com/ [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://grafana.com/ [Open source]

- Logs
  - https://grafana.com/

Logs

Grafana's log aggregation and storage system allows you to bring together logs from all your applications and infrastructure in a single place. Easily export logs using Grafana Loki, Promtail, Fluentd, Fluentbit, Logstash, and more.

Grafana Loki
Grafana Cloud Logs
Grafana Enterprise Logs

Watch the demo →

Product docs →

Get started for free →

- Dashboards
  - https://grafana.com/

Dashboards

Create dynamic and reusable dashboards with a multitude of options that allow you to visualize your data any way you want.

See all dashboards →

Demo video →

Product docs →

Get started for free →

- Alertas
  - https://grafana.com/

- Exemplo de ferramenta de monitoramento:
  - https://www.zabbix.com/br [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://www.zabbix.com/br [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://www.zabbix.com/br [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://opentelemetry.io/ [Open source]

- Exemplo de ferramenta de monitoramento:
  - https://opentelemetry.io/docs/

- Exemplo de ferramenta de monitoramento:
  - https://newrelic.com/

- Exemplo de ferramenta de monitoramento:
  - https://newrelic.com/

- Logs
  - https://newrelic.com/

Deeper visibility into all your logs—fast, in context, and affordable.

Log Management

- Dashboards
  - https://newrelic.com/

- Alertas
  - https://newrelic.com/

- Exemplo de ferramenta de monitoramento:
  - https://www.uptrends.com/

ITRS uptrends
Press '/' to focus, type to search
Monitoring
Alerting
Dashboards
Synthetics
Transactions
Browser
API
Uptime
Private locations Beta
RUM
Real users
Account setup
Tools & apps
Buzz Lightyear Admin
Account overview
All monitors
All checkpoints
Last 24 hours
Performance - All monitors
Seconds
12:00
16:00
20:00
23. Nov
04:00
08:00
Total time
Last updated 16:31:43
Load time by checkpoint
30
25
20
15
10
5
0
Total time
Last updated 16:31:43
Load time by country
Network duration
0,021 seconds
Backend duration
0,486 seconds
Frontend duration
1,869 seconds
Load time
2,376 seconds
Time to first byte
0,528 seconds
Last updated 16:31:43
Account status - All monitors
Book Resort UK
3031
Transaction
Galacticresorts.com
3031
HTTPS
Galacticresorts.fr
3031
HTTPS
Galacticresorts.co.uk
2000
FPC
Galactic shirts IE
200
FPC
Galacticshirts.com
200
HTTPS
1 2 3
Last updated 16:31:43
Order process
Seconds
12:00
16:00
20:00
23. Nov
04:00
08:00
Login
Select product
Order and pay
Last updated 16:31:43
Uptime & confirmed errors - All monitors
Uptime %
Errors
12:00
16:00
20:00
23. Nov
04:00
08:00
Uptime percentage
Confirmed errors
Last updated 16:31:43

- Exemplo de ferramenta de monitoramento:
  - https://www.uptrends.com/

Web Application Monitoring

Monitoramento de aplicativos da Web

Monitore as jornadas dos usuários em seu aplicativo web, como carrinhos de compras, checkouts e logins. Grave facilmente seus scripts ou deixe o suporte ajudá-lo a configurá-los.

Web Performance Monitoring

Monitoramento de Desempenho Web

Com o monitoramento de desempenho da web, você sabe exatamente qual elemento torna seu site mais lento. Use os navegadores Chrome e Edge mais recentes para monitorar Core Web Vitals (principais sinais vitais da Web) e métricas W3C.

API Monitoring

Monitoramento de API

Diagnostique problemas subjacentes e configure o monitoramento contínuo da API para monitorar mudanças nas circunstâncias. Monitore o tempo de atividade da API e configure uma série de chamadas de API para verificar suas respostas de API.

Uptime Monitoring

Monitoramento de tempo de atividade

Conheça o tempo de atividade e a velocidade do seu site em qualquer lugar do mundo. Com 233 pontos de verificação, temos uma das coberturas mais altas do setor e você pode ter certeza de que não perderá nenhum erro (local).

- Exemplo de ferramenta de monitoramento:
  - https://aws.amazon.com/pt/cloudwatch/

- Exemplo de ferramenta de monitoramento:
  - https://aws.amazon.com/pt/cloudwatch/

Amazon CloudWatch | Visão geral | Recursos | Preço | Conceitos básicos | Perguntas frequentes | Clientes

Por que o Amazon CloudWatch?

O Amazon CloudWatch é um serviço que monitora aplicações, responde às mudanças de desempenho, otimiza o uso de recursos e fornece insights sobre a integridade operacional. Ao coletar dados de todos os recursos da AWS, o CloudWatch fornece visibilidade sobre o desempenho de todo o sistema e permite que os usuários definam alarmes, reajam automaticamente às mudanças e obtenham uma visão unificada da integridade operacional.

Introdução ao Amazon CloudWatch

- Exemplo de ferramenta de monitoramento:
  - https://aws.amazon.com/pt/cloudwatch/

- Exemplo de ferramenta de monitoramento:
  - https://aws.amazon.com/pt/cloudwatch/

Amazon CloudWatch | Visão geral | Recursos | Preço | Conceitos básicos | Perguntas frequentes | Clientes

Casos de uso

Monitore a performance da aplicação

Visualize performance data, create alarms, and correlate data to understand and resolve the root cause of performance issues in your AWS resources

Execute a análise de causa-raiz

Analyze metrics, logs, logs analytics, and user requests to speed up debugging and reduce overall mean time to resolution

Otimize os recursos de forma proativa

Automate resource planning and lower costs by setting actions to occur when thresholds are met based on your specifications or machine learning models

Teste os impactos do site

Find out exactly when your website is impacted and for how long by viewing screenshots, logs, and web requests at any point in time

- Exemplo de ferramenta de monitoramento:
  - https://azure.microsoft.com/pt-br/products/monitor

- # Azure Monitor

- Exemplo de ferramenta de monitoramento:
  - https://azure.microsoft.com/pt-br/products/monitor

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

- Datadog

Application + infrastructure overview
Add Template Variables
The Past Hour
Metrics
Traces
Logs
Active users
ELB requests per second
us-east-1a
us-east-1d
us-east-1c
CPU %
Disk space available
System load
Memory
Lambda invocations by function
Lambda execution duration by function
Postgres committed transactions
Postgres DB connections
Latency SLO
Over 7 days
99.50%
Error Budget
50m
Month-to-date
99.51%
Error Budget
2h 23m
App performance
Total Requests
Total Errors
Latency
Total time spent by service
Name
Hits
Avg Latency
Total time
Errors
Error rate
GET /coffeehouse
GET /orders/muffin
GET /orders/milk
GET /orders/bacon
GET /orders/crepe
Service dependencies
coffee-house
dotnet-coffeehouse
2xx/3xx responses
19.3K rsps
4xx errors
325.42 rsps
5xx errors
240.05 rsps
Service logs
Error logs by endpoint
4xx/5xx errors
Error logs

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

Reduza os custos de TI

- Reduza a proliferação de ferramentas e elimine custos compostos de investimento, licenciamento e manutenção com uma solução de monitoramento unificada (em vez de ter mais de 9 ferramentas de observabilidade isoladas)
- Obtenha maior visibilidade e controle sobre a ingestão e o gerenciamento de dados, para que suas equipes possam monitorar com mais precisão o uso e os custos gerais
- Tome decisões informadas sobre como alocar gastos e garantir que as equipes permaneçam dentro do orçamento

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

Receba alertas apenas sobre os problemas importantes e resolva os problemas com mais rapidez

- Detecte automaticamente valores discrepantes, anomalias e erros inesperados com o Watchdog
- Elimine falsos positivos com alertas compostos, baseados em lógica booleana, e alertas mais inteligentes de anomalias e valores discrepantes que levam em conta flutuações diárias, semanais e sazonais
- Receba apenas alertas específicos, acionáveis e contextuais, mesmo em ambientes de grande escala e altamente efêmeros

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

Melhore a experiência do usuário e otimize facilmente o desempenho

- Colete 100% das suas sessões de usuário e entenda a experiência do usuário em todos os aplicativos com dados críticos de desempenho
- Monitore as principais jornadas dos usuários com um gravador da web fácil de usar e sem código — sem a necessidade de scripts, basta clicar em seu aplicativo como um usuário faria; economize recursos de engenharia com testes de automanutenção alimentados por IA
- Identifique a causa raiz dos tempos de carregamento lentos, seja um problema com o código, a rede ou a infraestrutura com visibilidade de ponta a ponta

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

Acelere o tempo de lançamento no mercado

- Detecte problemas automaticamente antes que eles aumentem; detecte problemas críticos em seus pipelines antes mesmo de entrarem em produção com testes de CI/CD
- Aumente a visibilidade das implantações e avalie facilmente o estado da produção após implantações individuais para identificar regressões e automatizar reversões
- Identifique rapidamente gargalos, erros, problemas de tráfego intenso, consultas lentas e muito mais
- Libere novos lançamentos no mercado com mais rapidez; monitore novo código em todas as fases do ciclo de desenvolvimento

- Exemplo de ferramenta de monitoramento:
  - https://www.datadoghq.com/pt/dg/monitor/free-trial/

Reduza os riscos de segurança

- Obtenha visibilidade sem precedentes dos fluxos de ataque a aplicativos com o Application Security Management (ASM)
- Visão unificada de suas ameaças mais críticas e configurações incorretas em sua infraestrutura de produção com Datadog Cloud Security Management
- Encontre, priorize e mitigue as ameaças mais críticas antes que elas comprometam seu ambiente com um SIEM em nuvem econômico e de baixa manutenção

- Exemplo de ferramenta de monitoramento:
  - https://graphiteapp.org/

- Exemplo de ferramenta de monitoramento:
  - https://graphiteapp.org/

- Exemplo de ferramenta de monitoramento:
  - https://graphiteapp.org/

- Exemplo de ferramenta de monitoramento:
  - https://www.splunk.com/
  - Leitura sugerida: https://www.splunk.com/en_us/blog/learn/api-monitoring.html

- Exemplo de ferramenta de monitoramento:
  - https://www.splunk.com/

- Exemplo de ferramenta de monitoramento:
  - https://www.splunk.com/

- Splunk

Keep your digital systems securely up and running

Fend off threat actors. Diminish downtime. Fix issues faster. Do it all with Splunk, the key to enterprise resilience.

Explore Why Splunk

Prevent major issues

Identify key risks and detect threats with artificial intelligence before they become major incidents.

Absorb shocks

Restore critical services faster to minimize the impact of outages and breaches.

Accelerate transformation

Adapt quickly and safely with the visibility you need to stay secure, compliant and reliable.

- Exemplo de ferramenta de monitoramento:
  - https://www.postman.com/
    - [DOC] https://learning.postman.com/docs/monitoring-your-api/intro-monitors/
    - [DOC] https://learning.postman.com/docs/monitoring-your-api/viewing-monitor-results/

- Exemplo de ferramenta de monitoramento:
  - https://www.postman.com/

- Exemplo de ferramenta de monitoramento:
  - https://www.postman.com/

- Exemplo de ferramenta de monitoramento:
  - https://www.postman.com/

- Exemplo de ferramenta de monitoramento:
  - https://smartbear.com/product/alertsite/api-free-trial/

SMARTBEAR AlertSite

Features | Integrations | Resources | Pricing | Request Demo

Don't Settle for Basic API Monitoring

Engage The Right People. Immediately.

- Advanced **customizable routed alerts** sent **directly from the monitoring nodes**
- Instant **actionable insights** from your monitoring data with our waterfalls and reports
- 360-degree insight of REST or SOAP APIs **internally and/or externally** with AlertSite's global network of over **340 monitoring nodes** spread across **80+ locations.**

Start Your Free Trial

Work Email

First Name

Last Name

Company

Phone

Your privacy matters. This information will be used to get back to you as soon as possible and provide best practices gained from our community of 16 million+ software experts, just like you.

Request Demo

By submitting this form, you agree to our Terms of Use and Privacy Policy

- Exemplo de ferramenta de monitoramento:
  - https://smartbear.com/product/alertsite/api-free-trial/

- Exemplo de ferramenta de monitoramento:
  - https://smartbear.com/product/alertsite/api-free-trial/

- Exemplo de ferramenta de monitoramento:
  - https://smartbear.com/product/alertsite/api-free-trial/

Ferramentas

- No caso desta disciplina, usaremos a seguinte combinação para **monitorização**:
  - **Spring Boot Actuator + Micrometer + Prometheus + Grafana**
    - https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html
    - https://micrometer.io/
    - https://prometheus.io/docs/introduction/overview/
    - https://grafana.com/docs/

Ferramentas

- No ecossistema de monitoramento de aplicativos Spring Boot, o fluxo envolvendo o Spring Boot Actuator, o Micrometer, Prometheus, e Grafana segue um caminho bem definido para coletar, armazenar e visualizar métricas de desempenho.

- O **Spring Boot Actuator**, uma extensão do Spring Boot, fornece endpoints operacionais, incluindo métricas, enquanto o **Micrometer** atua como a biblioteca padrão para instrumentação de métricas no código-fonte do aplicativo. O Micrometer simplifica a coleta de métricas, oferecendo uma API consistente.

Ferramentas

- Em seguida, o **Prometheus**, um sistema de monitoramento de código aberto, atua como um coletor de métricas, buscando dados diretamente dos endpoints do Micrometer.

- Por fim, o **Grafana**, uma ferramenta de visualização, é configurado para se conectar ao Prometheus, permitindo a criação de painéis interativos e personalizados para análise visual das métricas coletadas.

Ferramentas

- Esse fluxo acontece dessa forma para proporcionar uma arquitetura modular e flexível, permitindo a escolha de diferentes componentes de monitoramento de acordo com as necessidades específicas do desenvolvedor.

- Essa abordagem facilita a padronização na instrumentação de métricas e oferece uma solução eficaz para monitorar e otimizar o desempenho de aplicativos Spring Boot.

- Micrometer
  - https://micrometer.io/

MICROMETER
application observability

Documentation Support

Vendor-neutral application observability facade

Micrometer provides a facade for the most popular observability systems, allowing you to instrument your JVM-based application code without vendor lock-in. Think SLF4J, but for observability.

Support for popular observability systems

As an instrumentation facade, Micrometer lets you instrument your code with a vendor-neutral interface and decide on the observability system as a last step. Instrumenting a library with Micrometer lets it be used in applications that ship data to different backends or even multiple backends at the same time.

Micrometer supports publishing metrics to **AppOptics**, **Azure Monitor**, **Netflix Atlas**, **CloudWatch**, **Datadog**, **Dynatrace**, **Elastic**, **Ganglia**, **Graphite**, **Humio**, **Influx/Telegraf**, **JMX**, **KairosDB**, **New Relic**, **OpenTelemetry** Protocol (OTLP), **Prometheus**, **SignalFx**, **Google Stackdriver**, **StatsD**, and **Wavefront**.

Through Micrometer Observation and Micrometer Tracing you can ship your spans via OpenZipkin Brave or OpenTelemetry tracers to different backends (e.g. OpenZipkin or Wavefront).

- Micrometer
  - https://micrometer.io/

Integrated into Frameworks

Popular frameworks that integrate with Micrometer include Helidon, Micronaut, Quarkus, and Spring. You can use the idioms and configuration model native to your application framework to get started with Micrometer.

Instrumentation Provided

Out-of-the-box instrumentation is available in micrometer-core and in libraries. You do not need to write your own instrumentation for many common components.

Works in your Environment

Micrometer can directly publish to most backends for storing your observability data. You can use what you have or switch. Micrometer makes it easy. See below and the documentation for supported backends.

Dimensional Metrics

Vendor-neutral abstractions for timers, gauges, counters, distribution summaries, and long task timers are provided with a dimensional data model. You can publish to a backend that supports dimensional metrics for efficient access to named metrics where you can drill down across its dimensions.

Distributed Tracing

Micrometer Tracing is a facade over the Brave and OpenTelemetry tracers that gives insight into complex distributed systems at the level of an individual user request. Identify the root cause of issues faster with distributed tracing. Micrometer Tracing is the successor to the Spring Cloud Sleuth project.

Unified Observability

You can instrument with the Micrometer Observation API, a single abstraction that can produce metrics, tracing, logs and more. You can instrument once, get multiple benefits, and keep metadata consistent across your observability data.

Micrometer

- O Micrometer é a biblioteca padrão de métricas no ecossistema do Spring Boot.

- Ele é integrado com o Spring Boot Actuator, permitindo a exposição de métricas via endpoints, como **/actuator/metrics**.
  - https://docs.micrometer.io/micrometer/reference/

Micrometer

- O Micrometer é uma biblioteca Java projetada para coletar, armazenar e expor métricas em ambientes de aplicativos distribuídos.

- Ela fornece uma API simples e consistente para instrumentar códigos Java e registrar métricas em vários sistemas de monitoramento.

Micrometer

- O principal objetivo do Micrometer é oferecer uma camada de abstração para métricas, permitindo que os desenvolvedores instrumentem seus aplicativos uma vez e, em seguida, escolham para onde desejam enviar essas métricas.

- O Micrometer suporta uma variedade de sistemas de monitoramento populares, como Prometheus, Grafana, Datadog, New Relic, entre outros.

Spring Boot Actuator

- O Spring Boot Actuator é um módulo do Spring Boot que fornece recursos adicionais para monitoramento e gerenciamento de aplicações Spring Boot em execução.

- Ele inclui uma série de endpoints e recursos que oferecem informações detalhadas sobre o estado da aplicação, métricas de desempenho, informações sobre o ambiente de execução e muito mais.

Spring Boot Actuator

- Alguns dos principais recursos e endpoints fornecidos pelo Spring Boot Actuator:
  - **Health (Saúde)**: O endpoint /actuator/health fornece informações sobre o estado de saúde da aplicação. Isso é útil para verificar se a aplicação está em execução e se todos os componentes estão funcionando corretamente.

  - **Info (Informações)**: O endpoint /actuator/info permite a exposição de informações personalizadas sobre a aplicação. Isso pode incluir detalhes como versão, descrição e outras informações personalizadas.

Spring Boot Actuator

- Alguns dos principais recursos e endpoints fornecidos pelo Spring Boot Actuator:
  - **Metrics (Métricas)**: O endpoint /actuator/metrics fornece métricas detalhadas sobre diversos aspectos da aplicação, como uso de memória, tempos de resposta de endpoints, contagem de requisições e muito mais.

  - **Environment (Ambiente)**: O endpoint /actuator/env expõe informações sobre as propriedades e configurações do ambiente de execução da aplicação.

Spring Boot Actuator

- Alguns dos principais recursos e endpoints fornecidos pelo Spring Boot Actuator:
  - **Auditing (Auditoria)**: O Actuator oferece recursos para auditoria, incluindo endpoints relacionados à auditoria de eventos, como criação, modificação e exclusão de entidades.
  - **Trace (Rastreamento)**: O endpoint /actuator/trace fornece informações detalhadas sobre as solicitações HTTP recentes, incluindo cabeçalhos, parâmetros e muito mais.
  - **Shutdown (Desligamento)**: O endpoint /actuator/shutdown permite o desligamento controlado da aplicação.

Spring Boot Actuator

- Estes recursos são extremamente úteis durante o desenvolvimento, teste e operação de aplicações, permitindo a **monitorização em tempo real** e a obtenção de informações valiosas sobre o estado e o desempenho da aplicação.

- É importante notar que, embora o Spring Boot Actuator ofereça muitos recursos prontos para uso, ele também fornece extensibilidade, permitindo que você adicione métricas personalizadas, audite eventos específicos e **integre-se a sistemas de monitoramento externos**.

Sumário

- Monitoramento
  - **Logging**
  - Dashboards
  - Alertas

Monitoramento

- **Logging**
  - *Logging* é o primeiro componente de um monitoramento pronto para produção. Ele começa no código-base de cada microsserviço e faz parte desse código-base, está contido no código de cada serviço, capturando todas as informações necessárias para descrever o estado do microsserviço.

  - Descrever o estado do microsserviço em qualquer momento do passado recente é o objetivo básico do logging.

Monitoramento

- **Logging**
  - Um dos benefícios da arquitetura de microsserviço é a liberdade que os desenvolvedores têm de implantar novos recursos e mudanças de código frequentemente, e uma das consequências desta recém-descoberta liberdade do desenvolvedor e da maior velocidade de desenvolvimento é que o microsserviço está sempre mudando.

Monitoramento

- **Logging**
  - Na maioria dos casos o serviço não será o mesmo de doze horas atrás, muito menos de vários dias atrás, e reproduzir qualquer problema será impossível.

  - Quando nos confrontamos com um problema, muitas vezes, a única maneira de determinar a causa fundamental de um incidente ou interrupção é vasculhar os logs, descobrir o estado do microsserviço no momento da interrupção e descobrir por que o serviço falhou naquele estado.

Monitoramento

- **Logging**
  - O logging deve ser tal que os desenvolvedores possam determinar a partir dos logs exatamente o que deu errado e onde as coisas começaram a desmoronar.

  - Determinar precisamente **o que** deve ser gravado no log é específico de cada microsserviço. A melhor orientação para determinar o que precisa ser gravado no log é, infelizmente, necessariamente vaga: grave qualquer informação essencial para descrever o estado do serviço em determinado momento.

Monitoramento

- **Logging**
  - Felizmente podemos limitar quais informações são necessárias restringindo nosso logging a tudo aquilo que pode ser contido no código do serviço.

  - Informações em nível de servidor e de infraestrutura não serão (e não devem ser) gravadas no log pela própria aplicação, mas pelos serviços e ferramentas que executam a plataforma de aplicação.

Monitoramento

- **Logging**
  - Algumas métricas principais e informações de microsserviço, como IDs de usuário e detalhes da solicitação e resposta, podem e devem ser localizadas nos logs dos microsserviços.

  - Claro que existem coisas que nunca devem ser gravadas em logs. Os logs nunca devem conter informações de identificação, como nomes de clientes, números da previdência social e outros dados privados.

Monitoramento

- **Logging**
  - Os logs nunca devem conter informações que possam apresentar um risco à segurança, como senhas, chaves de acesso ou segredos.

  - Na maioria dos casos, mesmo informações aparentemente inócuas como IDs de usuário e nomes de usuário nunca devem ser gravadas em log, exceto quando criptografadas.

Monitoramento

- **Logging**
  - Às vezes, gravar logs no âmbito do microsserviço individual não é suficiente.
  - Como vimos, os microsserviços não existem sozinhos, mas dentro de complexas cadeias de clientes e dependências dentro do ecossistema de microsserviços.

Monitoramento

- **Logging**
  - Embora os desenvolvedores possam fazer seu melhor para gravar logs e monitorar tudo que for importante para seu serviço, gravar logs de solicitações e respostas por toda a cadeia de cliente e dependências fim a fim pode revelar informações importantes sobre o sistema que de outro modo permaneceriam desconhecidas (como a latência total e a disponibilidade da pilha).

  - Para tornar estas informações acessíveis e visíveis, construir um ecossistema de microsserviços requer rastrear cada solicitação por toda a pilha.

Monitoramento

- **Logging**
  - Vimos que muitas informações precisam ser gravadas em log. Logs são dados, e gravá-los é caro: armazená-los é caro, acessá-los é caro, e tanto armazenar como acessar logs têm um custo adicional associado a fazer chamadas de rede caras.

  - O custo de armazenar logs pode não parecer muito para um microsserviço individual, mas se somarmos todas as necessidades de logging de todos os microsserviços dentro de um ecossistema, veremos que seu custo é alto.

Monitoramento

- **Logging**
  - Evite adicionar logs de depuração (debug) no código que será implantado em produção – tais logs são muito caros.
  - Se qualquer log for adicionado especificamente com a finalidade de depuração, os desenvolvedores deverão prestar atenção para garantir que qualquer ramo ou versão contendo esses logs adicionais nunca cheguem ao ambiente de produção.

Monitoramento

- **Logging**
  - O logging precisa ser escalável, disponível, facilmente acessível e pesquisável.
  - Para manter o custo dos logs baixo e garantir escalabilidade e alta disponibilidade, geralmente é necessário impor cotas de logging por serviço junto a limites e padrões sobre quais informações podem ser gravadas em log, quantos logs cada microsserviço pode armazenar e por quanto tempo os logs serão armazenados antes de serem apagados.

- **Logging**
  - https://grafana.com/docs/grafana-cloud/send-data/logs/

- Graylog
  - https://graylog.org/

- Graylog
  - https://graylog.org/

  - O **Graylog** é uma plataforma de gerenciamento de logs e análise de registros, projetada para ajudar organizações a coletar, armazenar, analisar e visualizar registros de sistemas diversos.

- Graylog Audit Log
- https://go2docs.graylog.org/5-2/interacting_with_your_log_data/audit_log_use_cases.html

- Loggly
  - https://www.loggly.com/

- Loggly
  - https://www.loggly.com/

  - O Loggly é um serviço de gerenciamento de logs em nuvem, projetado para ajudar organizações a lidar com o volume crescente de registros (logs) gerados por seus aplicativos e sistemas. Ele oferece uma plataforma centralizada para coleta, armazenamento, pesquisa e análise de logs.

- Loggly
  - https://www.loggly.com/

- Grafana Loki
  - https://grafana.com/oss/loki/

- Grafana Loki
  - https://grafana.com/oss/loki/

Loki takes a unique approach by **only indexing the metadata** rather than the full text of the log lines:

```
2019-12-11T10:01:02.123456789Z  {app="nginx",cluster="us-west1"}  GET /about
```

Timestamp
with nanosecond precision

Prometheus-style Labels
key-value pairs

Content
logline

indexed

unindexed

Loki's minimal indexing approach means that storing the same set of logs in Loki requires far less storage than with other solutions

- ✓ Log any and all formats
- ✓ Fast writes
- ✓ Tiny indexes
- ✓ Cheaper to run
- ✓ Simpler to operate
- ✓ Fast queries
- ✓ Cut and slice your logs in dynamic ways (Flexible)

Sumário

- Monitoramento
  - Logging
  - **Dashboards**
  - Alertas

Monitoramento

- **Dashboards**
  - Cada microsserviço deve ter pelo menos um dashboard no qual todas as métricas principais (como utilização de hardware, conexões a databases, disponibilidade, latência, respostas e o status de endpoints de API) são coletadas e exibidas.

  - Um dashboard é uma **visualização gráfica** que é atualizada em tempo real para refletir todas as mais importantes informações sobre um microsserviço.

Monitoramento

- **Dashboards**
  - Dashboards devem ser facilmente acessíveis, centralizados e padronizados em todo o ecossistema de microsserviços.
  - Dashboards devem ser fáceis de interpretar de modo que alguém de fora possa rapidamente determinar a saúde do microsserviço: qualquer um deve ser capaz de olhar o dashboard e saber imediatamente se o microsserviço está funcionando corretamente ou não.

Monitoramento

- **Dashboards**
  - Isso requer encontrar um equilíbrio entre sobrecarregar o usuário com informações (o que tornaria o dashboard praticamente inútil) e não exibir informações suficientes (o que também tornaria o dashboard inútil): apenas o mínimo necessário de informações sobre as métricas principais deve ser exibido.

- Downdetector
  - https://downdetector.com.br/

Downdetector®

- Leitura sugerida:
  - https://relatosti.com.br/artigo/criando-dashboard-no-grafana-com-os-dados-do-downdetector-no-zabbix&id=220
  - https://relatosti.com.br/artigo/integrando-o-zabbix-com-o-downdetector&id=218

- Downdetector
  - https://downdetector.com.br/

O Downdetector é usado para rastrear interrupções de serviços online populares e fornecer informações sobre a situação de um serviço específico em tempo real.

Monitoramento

- **Dashboards**
  - Um dashboard também serve como um reflexo preciso da qualidade geral do monitoramento de todo o microsserviço.
  - Qualquer métrica principal que gere alertas deve ser incluída no dashboard: a exclusão de qualquer métrica do dashboard principal refletirá um monitoramento precário do serviço, enquanto a inclusão de métricas que não são necessárias refletirá uma não observância das melhores práticas de alerta (e, consequentemente, do monitoramento).

Monitoramento

- **Dashboards**
  - Há várias exceções à regra contra a inclusão de métricas não essenciais. Além das métricas principais, informações sobre cada fase do pipeline de deployment devem ser exibidas, embora não necessariamente dentro do mesmo dashboard.
  - Os devs trabalhando em microsserviços que requerem monitorar um grande número de métricas principais precisam optar por configurar um **dashboard separado** para cada fase de implantação (staging, pré-release e produção) para refletir com precisão a saúde do microsserviço em cada fase da implantação.

Monitoramento

- **Dashboards**
  - Aviso: Embora dashboards possam exibir anomalias e tendências negativas das métricas principais de um microsserviço, desenvolvedores nunca devem precisar observar o dashboard de um microsserviço para detectar incidentes e interrupções.
  - Fazê-lo é um antipadrão que leva a deficiências na geração de alertas e no monitoramento geral.

Monitoramento

- **Dashboards**
  - Para ajudar a determinar problemas introduzidos por novas implantações, é útil incluir informações sobre quando ocorreu uma implantação no dashboard. O modo mais eficaz e útil de fazê-lo é garantir que os tempos de implantação sejam exibidos nos gráficos de cada métrica essencial.

  - Isso permite que os desenvolvedores rapidamente verifiquem os gráficos depois de cada implantação para ver se qualquer padrão estranho surgiu em alguma das métricas principais.

Monitoramento

- **Dashboards**
  - Dashboards bem projetados também oferecem aos desenvolvedores uma maneira fácil e visual de detectar anomalias e determinar limites de alerta.
  - Mudanças ou perturbações muito pequenas ou graduais correm o risco de não serem detectadas pela geração de alertas, mas uma olhada atenta em um dashboard preciso pode revelar anomalias que, caso contrário, **não seriam detectadas**.

Monitoramento

- **Dashboards**
  - Limites de alerta são notoriamente difíceis de determinar, mas podem ser configurados adequadamente quando examinamos dados de histórico no dashboard: os desenvolvedores podem ver padrões normais nas métricas principais, ver picos nos valores das métricas que ocorreram com as interrupções (ou levaram às interrupções) no passado e então definir limites de acordo.

Monitoramento

- **Dashboards**
- https://grafana.com/docs/grafana/latest/fundamentals/dashboards-overview/

Monitoramento

- **Dashboards**
- https://grafana.com/about/events/grafanacon/2024/golden-grot-awards/

Monitoramento

- **GitLab**
  - O GitLab é uma plataforma completa de gerenciamento de ciclo de vida de desenvolvimento de software (**DevOps**) que fornece uma ampla gama de recursos para equipes de desenvolvimento colaborarem efetivamente em projetos de software.
    - https://about.gitlab.com/
    - https://docs.gitlab.com/ee/administration/monitoring/
    - https://docs.gitlab.com/ee/administration/monitoring/prometheus/gitlab_metrics.html

Monitoramento

- **GitLab**
  - O GitLab é uma empresa famosa por ser transparente. Eles até transmitiram interrupções internas ao vivo no passado.
  - Portanto, não é surpreendente que eles tornem públicos vários dashboards internos do Grafana para sua infraestrutura em nuvem. O dashboards GitLab oferece gráficos sobre tudo, desde estatísticas de disco até visões gerais da frota, relatórios de alertas e triagem.

Monitoramento

- **GitLab Dashboard**

- Graylog Dashboard
  - https://archivedocs.graylog.org/en/latest/pages/dashboards.html

Sumário

- Monitoramento
  - Logging
  - Dashboards
  - **Alertas**

Monitoramento

- **Alertas**
  - O terceiro componente do monitoramento de um microsserviço pronto para produção são os *alertas* em tempo real. A detecção de falhas, assim como a detecção de mudanças nas métricas principais que possam levar a uma falha, é efetuada por meio de alertas.

  - Pra isso, todas as métricas principais – métricas de servidor, métricas de infraestrutura e métricas específicas de microsserviço – devem gerar alertas, e os alertas devem ser configurados em vários limites.

Monitoramento

- **Alertas**
  - Alertas eficazes e acionáveis são essenciais para preservar a disponibilidade de um microsserviço e evitar downtime (período durante o qual um serviço específico não está disponível ou não está funcionando conforme o esperado).
  - É preciso configurar alertas para todas as métricas principais. Qualquer mudança em uma métrica no âmbito do servidor, de infraestrutura ou microsserviço que possa levar a uma interrupção, causar um pico de latência ou de alguma forma prejudicar a disponibilidade do microsserviço deve disparar um alerta.

Monitoramento

- **Alertas**
  - E, o mais importante, alertas também devem ser disparados sempre que uma métrica não for vista.
  - Todos os alertas devem ser úteis: eles devem ser definidos por adequados limites sinalizadores. Três tipos de limites devem ser configurados para cada métrica com limites superiores e inferiores: **normal**, **advertência** e **crítico**.

Monitoramento

- **Alertas**
  - **Limites normais** refletem os limites inferiores e superiores adequados e usuais de cada métrica e não devem disparar um alerta.
  - **Limites de advertência** irão disparar alertas quando houver um desvio de norma que possa levar a um problema com o microsserviço; limites de advertência devem ser configurados para dispararem alertas **antes** que qualquer desvio da norma cause uma interrupção ou afete negativamente o microsserviço.

Monitoramento

- **Alertas**
  - **Limites críticos** devem ser configurados com base em quais limites inferiores e superiores das métricas principais realmente causam uma interrupção, um pico de latência ou prejudicam a disponibilidade de um microsserviço.
  - O ideal é que os limites de advertência disparem alertas que levem a uma detecção, mitigação e resolução rápidas antes que limites críticos sejam atingidos.

Monitoramento

- **Alertas**
  - Em cada categoria, os limites devem ser altos o suficiente para evitar ruído, mas baixos o suficiente para detectar qualquer problema real com as métricas principais.
  - Um desafio em particular da fase inicial de um projeto de microsserviços, é determinar quais serão os limites para os alertas sem nenhum dado histórico para utilizar como base de conhecimento.

Monitoramento

- **Alertas**
  - Sem dados de histórico, pode ser muito difícil configurar limites para as métricas principais. Qualquer limite configurado no início do ciclo de vida de um microsserviço corre o risco de ser inútil ou disparar muitos alertas.

  - Para determinar os limites adequados para um novo microsserviço (ou mesmo um antigo), os desenvolvedores podem executar **testes de carga** no microsserviço para medir onde devem ficar os limites.

Monitoramento

- **Alertas**
  - Submeter o microsserviço a cargas “normais” de tráfego pode determinar os limites normais, enquanto cargas de tráfego acima do esperado podem ajudar a determinar os limites de advertência e críticos.

  - Leitura sugerida:
    - https://medium.com/fretebras-tech/criação-de-alertas-no-grafana-com-integração-ao-slack-8924af348a5b

Monitoramento

- **Alertas**

Monitoramento

- **Alertas**

- Leitura sugerida:
  - https://grafana.com/blog/2023/07/10/how-we-improved-grafanas-alert-state-history-to-provide-better-insights-into-your-alerting-data/

Monitoramento

- **Alertas**
  - Todos os alertas precisam ser **acionáveis**.
  - Alertas não acionáveis são aqueles que são disparados e então resolvidos (ou ignorados) pelo(s) desenvolvedor(es) de plantão para o microsserviço, pois não são importantes, não são relevantes, não significam que exista algo de errado com o microsserviço, ou alertam sobre um problema que não pode ser resolvido pelo(s) desenvolvedor(es).

Monitoramento

- **Alertas**
  - Qualquer alerta que não possa ser imediatamente tratado pelo(s) desenvolvedor(es) de plantão deve ser removido do pool de alertas, atribuído novamente ao turno de plantão relevante ou (se possível) alterado para se tornar um alerta acionável.
  - Algumas das métricas principais do microsserviço correm o risco de ser <u>não acionáveis</u>.

Monitoramento

- **Alertas**
  - Por exemplo, alertar sobre a disponibilidade de dependências pode facilmente levar a alertas não acionáveis se interrupções da dependência, aumentos da latência da dependência ou downtime da dependência não requerem que uma ação seja tomada por seu(s) cliente(s).

  - Se nenhuma ação precisa ser tomada, então os limites devem ser configurados adequadamente, ou, em casos mais extremos, nenhum alerta deve ser configurado para as dependências.

Monitoramento

- **Alertas**
  - Entretanto, se alguma ação precisa ser tomada, mesmo algo pequeno como contatar o turno de plantão ou a equipe de desenvolvimento da dependência para alertá-los sobre o problema e/ou coordenar a mitigação e resolução, então um alerta deve ser disparado.

- Leitura sugerida:
  - https://grafana.com/blog/2023/12/08/grafana-alerting-how-to-monitor-alerts-for-better-alert-management/

Monitoramento

- **Alertas** - Tratando alertas
  - Uma vez disparado o alerta, ele precisa ser tratado de forma rápida e eficaz. A causa fundamental do alerta disparado deve ser mitigada e resolvida.
  - Para tratar alertas de forma rápida e eficaz, há várias medidas que podem ser tomadas.
  - O primeiro passo é criar **instruções** para cada alerta conhecido, detalhando como selecionar, mitigar e resolver cada alerta.

Monitoramento

- **Alertas** - Tratando alertas
  - Estas instruções passo a passo devem fazer parte do roteiro do plantão dentro da **documentação** centralizada de cada microsserviço, tornando-as facilmente acessíveis para todos que estiverem de plantão para o microsserviço.

  - Roteiros são cruciais para o monitoramento de um microsserviço: eles fornecem a qualquer desenvolvedor de plantão instruções sobre como mitigar e resolver as causas fundamentais de cada alerta.

Monitoramento

- **Alertas** - Tratando alertas
  - Uma vez que cada alerta está vinculado a um desvio de uma métrica principal, roteiros podem ser escritos para tratar de cada métrica principal, das causas conhecidas dos desvios da norma e de como depurar o problema.

  - Devem ser criados dois tipos de roteiros de plantão. O primeiro faz parte dos roteiros para alertas de servidor e infraestrutura que devem ser compartilhados entre toda a organização de engenharia de software – eles devem ser escritos abordando todas as principais métricas de servidor e infraestrutura.

Monitoramento

- **Alertas** - Tratando alertas
  - O segundo tipo faz parte dos roteiros de plantão para microsserviços específicos que contêm instruções referentes aos alertas específicos de microsserviço e que são disparados por mudanças nas métricas principais.
  - Por exemplo, um pico de latência deve disparar um alerta, e o roteiro deve conter instruções passo a passo que documentem claramente como depurar, mitigar e resolver picos de latência do microsserviço.

Monitoramento

- **Alertas** - Tratando alertas
  - O segundo passo é identificar antipadrões de alerta.
  - Se o turno de plantão do microsserviço estiver sobrecarregado por alertas e mesmo assim o microsserviço aparentemente funcionar como esperado, então quaisquer alertas que forem vistos mais de uma vez, mas puderem ser facilmente mitigados e/ou resolvidos, **deverão ser automatizados**.

Monitoramento

- **Alertas** - Tratando alertas
  - Isto é, os passos para mitigação e/ou resolução devem ser incluídos no próprio microsserviço.
  - Isso vale para qualquer alerta, e escrever instruções passo a passo para alertas dentro dos roteiros de plantão permite uma execução eficaz dessa estratégia. Na verdade, qualquer alerta que, uma vez disparado, requer um simples conjunto de passos para ser mitigado e resolvido pode ser facilmente automatizado.

Monitoramento

- **Alertas** - Tratando alertas
  - Se for estabelecido este nível de monitoramento pronto para produção, um microsserviço nunca mais sofrerá o mesmo problema duas vezes.
  - Agora que você tem mais entendimento sobre monitoramento, use a lista de perguntas a seguir para avaliar a disponibilidade de produção de seu(s) microserviço(s) e do ecossistema de microsserviços.

Monitoramento

- **Avaliação** - Métricas principais:
  - 1) Quais são as métricas principais do microsserviço?
  - 2) Quais são as métricas de servidor e infraestrutura?
  - 3) Quais são as métricas no âmbito de microsserviço?
  - 4) Todas as métricas principais do microsserviço são monitoradas?

- **Avaliação** - Logging:
  - 1) Quais informações este microsserviço precisa gravar em log?
  - 2) Este microsserviço grava em log todas as solicitações importantes?
  - 3) O logging reflete o estado do microsserviço em qualquer momento?
  - 4) Esta solução de logging é escalável e eficaz em termos de custo?

Monitoramento

- **Avaliação** - Dashboards:
  - 1) Este microsserviço tem um dashboard?
  - 2) O dashboard é fácil de interpretar? Todas as métricas principais são exibidas no dashboard?
  - 3) Posso determinar se este microsserviço está funcionando corretamente ou não olhando o dashboard?

Monitoramento

- **Avaliação** - Alertas:
  - 1) Existe um alerta para todas as métricas principais?
  - 2) Todos os alertas são definidos por adequados limites sinalizadores?
  - 3) Os limites de alerta são configurados adequadamente para disparar um alerta antes que ocorra uma interrupção?
  - 4) Todos os alertas são acionáveis?
  - 5) Há instruções passo a passo para triagem, mitigação e resolução para cada alerta no roteiro de plantão?

Monitoramento

- **Avaliação** - Turnos de plantão:
  - 1) Há um turno de plantão dedicado e responsável por monitorar este microsserviço?
  - 2) Cada turno de plantão contém pelo menos dois desenvolvedores?
  - 3) Existem procedimentos padronizados de plantão em toda a organização de engenharia de software?

Bônus

- **Profissional DevOps**
  - O profissional DevOps é responsável por melhorar a colaboração e a comunicação entre as equipes de desenvolvimento de software e operações de TI (Tecnologia da Informação).
  - O termo "DevOps" é uma combinação de "Desenvolvimento" (Dev) e "Operações" (Ops).

Bônus

- **Profissional DevOps**
  - O principal objetivo do DevOps é aumentar a eficiência e a agilidade no ciclo de vida do desenvolvimento de software, promovendo a automação, a integração contínua e a entrega contínua.

Bônus

- **Profissional DevOps**

- As principais responsabilidades de um profissional DevOps incluem:
  - **Automação**: Desenvolver scripts e ferramentas para automatizar processos de desenvolvimento, testes e implantação.
  - **Integração Contínua (CI)**: Implementar práticas de integração contínua para garantir que as alterações de código sejam integradas e testadas automaticamente.
  - **Entrega Contínua (CD)**: Implementar práticas de entrega contínua para automatizar o processo de implantação de software em ambientes de produção.

Bônus

- **Profissional DevOps**

- As principais responsabilidades de um profissional DevOps incluem:
  - **Monitoramento e Logging**: Implementar soluções de monitoramento e logging para garantir a visibilidade e a detecção precoce de problemas.
  - **Colaboração**: Facilitar a comunicação e a colaboração entre equipes de desenvolvimento e operações.
  - **Segurança**: Integrar práticas de segurança no ciclo de vida do desenvolvimento de software.

Bônus

- **Profissional DevOps**
  - A média de salário para profissionais DevOps pode variar significativamente com base na localização, experiência, certificações e setor de trabalho.

  - Em países como os Estados Unidos, por exemplo, um profissional DevOps pode ganhar em média entre $90.000 a $140.000 por ano, mas esses números são apenas uma referência geral e podem variar. Em outras regiões do mundo, os salários podem ser diferentes.

Bônus

- **Profissional DevOps**
  - Em média, um profissional DevOps no Brasil pode ganhar entre R$ 8.000,00 a R$ 15.000,00 por mês. Esses valores são aproximados e podem variar em diferentes estados e cidades do país.

  - Além disso, a obtenção de certificações relevantes, como Certified Kubernetes Administrator (CKA), AWS Certified DevOps Engineer, ou Certified Jenkins Engineer, pode impactar positivamente o salário de um profissional DevOps.

Bônus

- **Profissional DevOps**

Vamos à prática!

- **Monitorização com Spring Boot Actuator + Prometheus + Grafana:**
  - https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html
  - https://micrometer.io/
  - https://prometheus.io/docs/introduction/overview/
  - https://grafana.com/docs/
- Leitura sugerida:
  - https://www.baeldung.com/spring-boot-actuators

Vamos à prática!

- **Monitorização com Spring Boot Actuator + Prometheus + Grafana:**
  - O projeto a seguir é uma demonstração que ilustra o processo de exposição e instrumentalização de métricas em uma aplicação Spring Boot.

  - Essa ação é fundamental para integrar eficientemente a aplicação Spring Boot com o Prometheus, possibilitando a coleta e análise de dados operacionais essenciais para o monitoramento e otimização contínua da aplicação.

Vamos à prática!

- **Monitorização com Spring Boot Actuator + Prometheus + Grafana:**
  - Essa integração não apenas viabiliza a coleta eficiente de dados operacionais, mas também possibilita a visualização e análise destas métricas através do Grafana.

  - O Grafana atua como uma interface gráfica poderosa, permitindo uma compreensão mais intuitiva e uma análise aprofundada do desempenho da aplicação, facilitando a tomada de decisões para otimização contínua.

- Primeiramente, crie a aplicação Spring Boot:
  - https://start.spring.io/

spring initializr

Project
- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language
- Java
- Kotlin
- Groovy

Spring Boot
- 3.3.0 (SNAPSHOT)
- 3.3.0 (M1)
- 3.2.3 (SNAPSHOT)
- 3.2.2
- 3.1.9 (SNAPSHOT)
- 3.1.8

Project Metadata

Group	com.example
Artifact	Monitoring
Name	Monitoring
Description	Monitoring
Package name	com.example.Monitoring
Packaging	Jar / War
Java	21 / 17

Dependencies — ADD DEPENDENCIES... ⌘ + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Prometheus OBSERVABILITY
Expose Micrometer metrics in Prometheus format, an in-memory dimensional time series database with a simple built-in UI, a custom query language, and math operations.

GENERATE ⌘ + ↵ | EXPLORE CTRL + SPACE | SHARE...

- Arquivo **pom.xml**:
  - Vejamos como ficam as nossas dependências, geradas pelo Spring Boot, no arquivo de configuração **pom.xml**.

Spring Boot Actuator

Micrometer

Prometheus

pom.xml (Monitoring)

```
19  <dependencies>
20      <!-- Spring Boot Starter -->
21      <dependency>
22          <groupId>org.springframework.boot</groupId>
23          <artifactId>spring-boot-starter-web</artifactId>
24      </dependency>
25      <dependency>
26          <groupId>org.springframework.boot</groupId>
27          <artifactId>spring-boot-starter-test</artifactId>
28          <scope>test</scope>
29      </dependency>
30      <dependency>
31          <groupId>org.springframework.boot</groupId>
32          <artifactId>spring-boot-starter-actuator</artifactId>
33      </dependency>
34      <!-- Prometheus -->
35      <dependency>
36          <groupId>io.micrometer</groupId>
37          <artifactId>micrometer-registry-prometheus</artifactId>
38          <scope>runtime</scope>
39      </dependency>
40  </dependencies>
```

- Arquivo **ApiController**:
  - Faça um controlador simples com um endpoint **/hello**.

- Arquivo **MonitoringApplication**:
  - Não se esqueça de adicionar a anotação:
    - @SpringBootApplication(scanBasePackages = {"com.example"})

- Arquivo **application.properties**:
  - Precisamos agora habilitar os endpoints do Spring Boot Actuator.

- Leitura sugerida:
  - https://www.baeldung.com/spring-boot-actuator-enable-endpoints

- Arquivo **application.properties**:
  - Vamos analisar cada uma das configurações:
    - #management.endpoints.web.exposure.include=prometheus,info,health,metrics,loggers,mappings
    - Define explicitamente quais endpoints do Spring Boot Actuator serão expostos via HTTP.
    - Neste caso, estão incluídos os seguintes endpoints: /actuator/prometheus, /actuator/info, /actuator/health, /actuator/metrics, /actuator/loggers, /actuator/mappings.
    - Esta configuração está comentada # pois utilizaremos o include=*.

    - management.endpoints.web.exposure.include=*
    - Define que todos os endpoints do Spring Boot Actuator estarão expostos via HTTP.
    - O caractere * é um curinga que significa "todos os endpoints".

- Vamos analisar cada uma das configurações:
  - management.endpoint.prometheus.enabled=true
  - Habilita o endpoint /actuator/prometheus.
  - Este endpoint é utilizado para exportar métricas no formato Prometheus, que é comumente usado em sistemas de monitoramento.

  - management.endpoint.info.enabled=true
  - Habilita o endpoint /actuator/info.
  - Este endpoint fornece informações gerais sobre a aplicação, geralmente configuradas no arquivo application.properties ou application.yml.

  - management.endpoint.health.enabled=true
  - Habilita o endpoint /actuator/health.
  - Este endpoint fornece informações sobre a saúde da aplicação, indicando se todos os componentes importantes estão funcionando corretamente.

- Vamos analisar cada uma das configurações:
  - management.endpoint.metrics.enabled=true
  - Habilita o endpoint /actuator/metrics.
  - Este endpoint expõe métricas detalhadas sobre o estado do aplicativo, incluindo métricas relacionadas à JVM, ao sistema operacional e a outros aspectos.

  - management.endpoint.loggers.enabled=true
  - Habilita o endpoint /actuator/loggers.
  - Este endpoint é usado para visualizar e configurar as configurações de log em tempo de execução.

  - management.endpoint.mappings.enabled=true
  - Habilita o endpoint /actuator/mappings.
  - Este endpoint fornece informações sobre os mapeamentos de URL no aplicativo, mostrando como as solicitações HTTP são roteadas para os controladores.

- Arquivo **MonitoringApplication**:
  - Execute então a aplicação para acessar os endpoints:

Vamos à prática!

- Acesse as URLs fornecidas pelo Spring Boot Actuator:
  - http://localhost:8080/actuator
  - http://localhost:8080/actuator/prometheus
  - http://localhost:8080/actuator/metrics
  - http://localhost:8080/actuator/metrics/http.server.requests
  - http://localhost:8080/actuator/metrics/jvm.memory.used
  - http://localhost:8080/actuator/health
  - http://localhost:8080/actuator/loggers
  - http://localhost:8080/actuator/mappings
  - http://localhost:8080/actuator/info

- Acesse as URLs fornecidas pelo Spring Boot Actuator:
  - http://localhost:8080/actuator

localhost:8080/actuator


```
{"_links":{"self":{"href":"http://localhost:8080/actuator","templated":false},"beans":
{"href":"http://localhost:8080/actuator/beans","templated":false},"caches-cache":
{"href":"http://localhost:8080/actuator/caches/{cache}","templated":true},"caches":
{"href":"http://localhost:8080/actuator/caches","templated":false},"health-path":
{"href":"http://localhost:8080/actuator/health/{*path}","templated":true},"health":
{"href":"http://localhost:8080/actuator/health","templated":false},"info":
{"href":"http://localhost:8080/actuator/info","templated":false},"conditions":
{"href":"http://localhost:8080/actuator/conditions","templated":false},"configprops":
{"href":"http://localhost:8080/actuator/configprops","templated":false},"configprops-prefix":
{"href":"http://localhost:8080/actuator/configprops/{prefix}","templated":true},"env":
{"href":"http://localhost:8080/actuator/env","templated":false},"env-toMatch":
{"href":"http://localhost:8080/actuator/env/{toMatch}","templated":true},"loggers":
{"href":"http://localhost:8080/actuator/loggers","templated":false},"loggers-name":
{"href":"http://localhost:8080/actuator/loggers/{name}","templated":true},"heapdump":
{"href":"http://localhost:8080/actuator/heapdump","templated":false},"threaddump":
{"href":"http://localhost:8080/actuator/threaddump","templated":false},"prometheus":
{"href":"http://localhost:8080/actuator/prometheus","templated":false},"metrics":
{"href":"http://localhost:8080/actuator/metrics","templated":false},"metrics-requiredMetricName":
{"href":"http://localhost:8080/actuator/metrics/{requiredMetricName}","templated":true},"scheduledtasks":
{"href":"http://localhost:8080/actuator/scheduledtasks","templated":false},"mappings":
{"href":"http://localhost:8080/actuator/mappings","templated":false}}}
```

- Acesse as URLs fornecidas pelo Spring Boot Actuator:
  - http://localhost:8080/actuator/prometheus

localhost:8080/actuator/prometheus

```
# HELP executor_completed_tasks_total The approximate total number of tasks that have completed execution
# TYPE executor_completed_tasks_total counter
executor_completed_tasks_total{name="applicationTaskExecutor",} 0.0
# HELP jvm_threads_states_threads The current number of threads
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{state="runnable",} 10.0
jvm_threads_states_threads{state="blocked",} 0.0
jvm_threads_states_threads{state="waiting",} 11.0
jvm_threads_states_threads{state="timed-waiting",} 4.0
jvm_threads_states_threads{state="new",} 0.0
jvm_threads_states_threads{state="terminated",} 0.0
# HELP jvm_gc_max_data_size_bytes Max size of long-lived heap memory pool
# TYPE jvm_gc_max_data_size_bytes gauge
jvm_gc_max_data_size_bytes 2.147483648E9
# HELP process_cpu_usage The "recent cpu usage" for the Java Virtual Machine process
# TYPE process_cpu_usage gauge
process_cpu_usage 0.0021495218374130215
# HELP system_cpu_usage The "recent cpu usage" of the system the application is running in
# TYPE system_cpu_usage gauge
system_cpu_usage 0.06504065040650407
# HELP process_files_max_files The maximum file descriptor count
# TYPE process_files_max_files gauge
process_files_max_files 10240.0
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
jvm_buffer_total_capacity_bytes{id="direct",} 90112.0
```

- Acesse as URLs fornecidas pelo Spring Boot Actuator:
  - http://localhost:8080/actuator/metrics

localhost:8080/actuator/metrics


```
{"names":
["application.ready.time","application.started.time","disk.free","disk.total","executor.active","executor.completed","executor.pool.core","executor.pool.max","executor.pool.size","executor.queue.remaining","executor.queued","http.server.requests","http.server.requests.active","jvm.buffer.count","jvm.buffer.memory.used","jvm.buffer.total.capacity","jvm.classes.loaded","jvm.classes.unloaded","jvm.compilation.time","jvm.gc.live.data.size","jvm.gc.max.data.size","jvm.gc.memory.allocated","jvm.gc.memory.promoted","jvm.gc.overhead","jvm.info","jvm.memory.committed","jvm.memory.max","jvm.memory.usage.after.gc","jvm.memory.used","jvm.threads.daemon","jvm.threads.live","jvm.threads.peak","jvm.threads.started","jvm.threads.states","logback.events","process.cpu.usage","process.files.max","process.files.open","process.start.time","process.uptime","system.cpu.count","system.cpu.usage","system.load.average.1m","tomcat.sessions.active.current","tomcat.sessions.active.max","tomcat.sessions.alive.max","tomcat.sessions.created","tomcat.sessions.expired","tomcat.sessions.rejected"]}
```


localhost:8080/actuator/metrics/http.server.requests


```
{"name":"http.server.requests","baseUnit":"seconds","measurements":[{"statistic":"COUNT","value":127.0},
{"statistic":"TOTAL_TIME","value":0.884446827},{"statistic":"MAX","value":0.029197792}],"availableTags":
[{"tag":"exception","values":["none"]},{"tag":"method","values":["GET"]},{"tag":"error","values":["none"]},
{"tag":"uri","values":["/actuator","/actuator/metrics","/actuator/prometheus"]},{"tag":"outcome","values":
["SUCCESS"]},{"tag":"status","values":["200"]}]}
```

localhost:8080/actuator/loggers


```
{"levels":["OFF","ERROR","WARN","INFO","DEBUG","TRACE"],"loggers":{"ROOT":
{"configuredLevel":"INFO","effectiveLevel":"INFO"},"_org":
{"effectiveLevel":"INFO"},"_org.springframework":
{"effectiveLevel":"INFO"},"_org.springframework.web":
{"effectiveLevel":"INFO"},"_org.springframework.web.servlet":
{"effectiveLevel":"INFO"},"_org.springframework.web.servlet.HandlerMapping":
{"effectiveLevel":"INFO"},"_org.springframework.web.servlet.HandlerMapping.Mappings":
{"effectiveLevel":"INFO"},"com":{"effectiveLevel":"INFO"},"com.example":
{"effectiveLevel":"INFO"},"com.example.Monitoring":
{"effectiveLevel":"INFO"},"com.example.Monitoring.application":
{"effectiveLevel":"INFO"},"com.example.Monitoring.application.MonitoringApplication":
{"effectiveLevel":"INFO"},"io":{"effectiveLevel":"INFO"},"io.micrometer":
{"effectiveLevel":"INFO"},"io.micrometer.common":
{"effectiveLevel":"INFO"},"io.micrometer.common.util":
{"effectiveLevel":"INFO"},"io.micrometer.common.util.internal":
{"effectiveLevel":"INFO"},"io.micrometer.common.util.internal.logging":
{"effectiveLevel":"INFO"},"io.micrometer.common.util.internal.logging.InternalLoggerFactory":
{"effectiveLevel":"INFO"},"io.micrometer.core":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.binder":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.binder.jvm":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.binder.jvm.ExecutorServiceMetrics":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.binder.jvm.JvmGcMetrics":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.internal":
{"effectiveLevel":"INFO"},"io.micrometer.core.instrument.internal.DefaultGauge":
{"effectiveLevel":"INFO"},"io.micrometer.observation":
{"effectiveLevel":"INFO"},"io.micrometer.observation.SimpleObservation":
{"effectiveLevel":"INFO"},"io.micrometer.observation.SimpleObservation$SimpleScope":
{"effectiveLevel":"INFO"},"org":{"effectiveLevel":"INFO"},"org.apache":
{"effectiveLevel":"INFO"},"org.apache.catalina":
{"effectiveLevel":"INFO"},"org.apache.catalina.core":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.ContainerBase":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.ContainerBase.[Tomcat]":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.ContainerBase.[Tomcat].[localhost]":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.ContainerBase.[Tomcat].[localhost].[/]":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.StandardEngine":
{"effectiveLevel":"INFO"},"org.apache.catalina.core.StandardService":
{"effectiveLevel":"INFO"},"org.apache.catalina.startup":
{"effectiveLevel":"INFO"},"org.apache.catalina.startup.DigesterFactory":
```


- http://localhost:8080/actuator/health
- http://localhost:8080/actuator/loggers

localhost:8080/actuator/health


```
{"status":"UP"}
```

JSON Formatter & Beautifier

- Visualizar o JSON desformatado é triste!

- É por isso que existem ferramentas online que têm como objetivo formatar, embelezar e tornar mais legível o código JSON.

- Faça o teste:
  - https://jsonformatter.org/
  - https://jsonbeautifier.org/

- JSON Beautifier
  - https://jsonbeautifier.org/

Prometheus UI

- O Prometheus possui uma interface gráfica para interagirmos e escrevermos expressões de consulta em **promql** (a linguagem de consulta do Prometheus).
- Antes de acessá-la, é necessário fazer o download e instalar o Prometheus:
  - https://prometheus.io/download/

- Links úteis:
  - https://github.com/prometheus/prometheus
  - https://prometheus.io/docs/prometheus/latest/getting_started/
  - https://prometheus.io/docs/prometheus/latest/querying/basics/

Prometheus UI

- No macOS:
  - brew install **prometheus**
  - Em seguida: **prometheus**

Terminal Local

```
joaopauloaramuni@MacBook-Pro-de-Joao Monitoring % prometheus
```

Downloading and running Prometheus

Download the latest release of Prometheus for your platform, then extract and run it:

```
tar xvfz prometheus-*.tar.gz
cd prometheus-*
```

prometheus

The Prometheus monitoring system and time series database. prometheus/prometheus

2.49.1 / 2024-01-15 Release notes

File name	OS	Arch	Size
prometheus-2.49.1.darwin-amd64.tar.gz	darwin	amd64	92.91 MiB
prometheus-2.49.1.linux-amd64.tar.gz	linux	amd64	92.68 MiB
prometheus-2.49.1.windows-amd64.zip	windows	amd64	95.78 MiB

2.45.3 / 2024-01-24 LTS Release notes

File name	OS	Arch	Size
prometheus-2.45.3.darwin-amd64.tar.gz	darwin	amd64	88.57 MiB
prometheus-2.45.3.linux-amd64.tar.gz	linux	amd64	88.32 MiB
prometheus-2.45.3.windows-amd64.zip	windows	amd64	91.26 MiB

alertmanager

Prometheus Alertmanager prometheus/alertmanager

0.26.0 / 2023-08-23 Release notes

File name	OS	Arch	Size
alertmanager-0.26.0.darwin-amd64.tar.gz	darwin	amd64	27.93 MiB
alertmanager-0.26.0.linux-amd64.tar.gz	linux	amd64	28.34 MiB
alertmanager-0.26.0.windows-amd64.zip	windows	amd64	29.00 MiB

- Arquivo **prometheus.yml**:

O arquivo **prometheus.yml** é o arquivo de configuração principal do Prometheus.

- Arquivo **prometheus.yml**:
  - Ele desempenha um papel crucial na definição de como o Prometheus coleta métricas, quais alvos (targets) deve monitorar e como as regras de gravação e alerta são configuradas.

  - Os "targets" são os endpoints específicos que o Prometheus monitora para coletar métricas. Cada target representa um serviço, aplicação ou componente que expõe métricas no formato adequado para serem coletadas pelo Prometheus.

  - No caso, temos dois targets:
    - - targets: [‘localhost:9090’]
    - - targets: [‘localhost:8080’]

- Com o Prometheus em execução, acesse:
  - http://localhost:9090/

- Com o Prometheus em execução, acesse:
  - http://localhost:9090/targets

- Insira qualquer métrica presente em **actuator/prometheus** no campo Expression do Prometheus (http_server_requests_seconds_count):
  - http://localhost:8080/actuator/prometheus

- Faça algumas solicitações para o endpoint **/hello** (atualize a página algumas vezes):
  - http://localhost:8080/hello
  - http://localhost:9090/graph

- jvm_memory_used_bytes
  - http://localhost:9090/graph

Prometheus Alerts Graph Status Help

Use local time | Enable query history | Enable autocomplete | Enable highlighting | Enable linter

jvm_memory_used_bytes Execute

Table Graph

Load time: 20ms Resolution: 14s Result series: 6

Evaluation time

jvm_memory_used_bytes{**area**="heap", **id**="G1 Eden Space", **instance**="localhost:8080", **job**="spring-actuator"}	51380224
jvm_memory_used_bytes{**area**="heap", **id**="G1 Old Gen", **instance**="localhost:8080", **job**="spring-actuator"}	16928768
jvm_memory_used_bytes{**area**="heap", **id**="G1 Survivor Space", **instance**="localhost:8080", **job**="spring-actuator"}	3145728
jvm_memory_used_bytes{**area**="nonheap", **id**="CodeCache", **instance**="localhost:8080", **job**="spring-actuator"}	11900928
jvm_memory_used_bytes{**area**="nonheap", **id**="Compressed Class Space", **instance**="localhost:8080", **job**="spring-actuator"}	5031664
jvm_memory_used_bytes{**area**="nonheap", **id**="Metaspace", **instance**="localhost:8080", **job**="spring-actuator"}	35076224

Remove Panel

Add Panel

- prometheus_http_requests_total
  - http://localhost:9090/graph

Prometheus Alerts Graph Status Help

prometheus_http_requests_total{**code**="200", **handler**="/api/v1/targets/metadata", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/api/v1/write", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/classic/static/*filepath", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/config", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/consoles/*filepath", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/debug/*subpath", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/favicon.ico", **instance**="localhost:9090", **job**="prometheus"}	15
prometheus_http_requests_total{**code**="200", **handler**="/federate", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/flags", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/graph", **instance**="localhost:9090", **job**="prometheus"}	27
prometheus_http_requests_total{**code**="200", **handler**="/manifest.json", **instance**="localhost:9090", **job**="prometheus"}	29
prometheus_http_requests_total{**code**="200", **handler**="/metrics", **instance**="localhost:9090", **job**="prometheus"}	820
prometheus_http_requests_total{**code**="200", **handler**="/rules", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/service-discovery", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/starting", **instance**="localhost:9090", **job**="prometheus"}	0
prometheus_http_requests_total{**code**="200", **handler**="/static/*filepath", **instance**="localhost:9090", **job**="prometheus"}	63

- Experimente também a aba Graph, ao lado de Table:
  - http://localhost:9090/graph

Exercício 1

- Utilize o Spring Boot Actuator + Micrometer + Prometheus para incorporar monitoramento ao projeto JWT_RestAPI.

- Da mesma forma que monitoramos o endpoint **/hello**, monitore também os endpoints do projeto JWT_RestAPI:
  - http://localhost:8080/login
  - http://localhost:8080/username/{token}
  - http://localhost:8080/user
  - http://localhost:8080/admin

Grafana

- Agora que já expomos e instrumentalizamos as métricas em uma aplicação Spring Boot com Prometheus, podemos criar os dashboards no Grafana.

- Primeiro, faça o download do Grafana:
  - https://grafana.com/get/?plcmt=top-nav&cta=downloads&tab=self-managed
  - https://grafana.com/grafana/download
  - https://grafana.com/docs/grafana/latest/

Grafana

- No macOS:
  - brew install **grafana**
  - Em seguida:
    - brew services start **grafana**

Grafana

- Agora basta acessar:
  - http://localhost:3000/login
  - Username: **admin**
  - Password: **admin**
  - Será solicitada a criação de uma nova senha.

- Grafana Home
  - http://localhost:3000/?orgId=1

Grafana

- O menu lateral esquerdo se expande ao clicar em Home.

- Veja como é fácil acessar os Dashboards e Alertas (Alerting).

- Vamos então criar nosso primeiro dashboard.

Grafana

- Primeiramente, será necessário indicar o data source (fonte de dados) que, no nosso caso, é o **Prometheus**.

Grafana

- Add data source: **Prometheus**.

Grafana

- Insira a URL **http://localhost:9090** do Prometheus:

Grafana

- Agora sim, clique em **Create your first dashboard**:
  - http://localhost:3000/dashboard/new?orgId=1

Grafana

- Clique em **+ Add visualization**:
  - http://localhost:3000/dashboard/new?orgId=1

Grafana

- Selecione o data source do Prometheus.
  - http://localhost:3000/dashboard/new?orgId=1

- Adicione a seguinte métrica: **http_server_requests_seconds_count**
  - Ou qualquer outra métrica disponível em **actuator/prometheus**

- Caso queira adicionar mais queries (ou mais métricas), basta clicar em:
  - **+ Add query**

- Rode a query para testar em **Run queries**
  - Em seguida clique em **Save**

- Salve o dashboard
  - No caso, inserimos apenas uma query/métrica.

Grafana

- Seu dashboard foi publicado em **Dashboards**.

Grafana

- Pratique inserindo mais métricas (queries) ao dashboard.

Exercício 2

- Crie um dashboard no Grafana para o projeto JWT_RestAPI.

- Faça um dashboard simples para monitorar performance/desempenho, uso de CPU, Memória RAM, etc. Utilize as métricas disponíveis em **actuator/prometheus** para configurar o seu dashboard:
  - http://localhost:8080/actuator/prometheus

Exercício 3

- Crie um alerta no Grafana para o projeto JWT_RestAPI.

- Crie um alerta no Grafana para o projeto JWT RestAPI.
  - https://grafana.com/docs/grafana/latest/alerting/

Referências básicas

Referências complementares

Outras referências

Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com

GitHub LinkedIn Lattes

...