DevOps

UNIDADE 06

Dando uma olhada: logs e monitoramento

Pra que isso?

Antes de conversarmos sobre monitoramento, vamos trazer para você uma analogia. Imagine que você comprou um carro novo. Aprender a dirigir é essencial, mas saber dirigir não é suficiente para garantir que o carro sempre funcione bem. Afinal de contas, você também precisa monitorar o carro. Isso significa fazer coisas como verificar o nível do óleo, a pressão dos pneus, a temperatura do motor, entre outros. Se você ignorar esses cuidados, seu carro pode quebrar e te deixar na mão quando você menos espera.

Em *software* é a mesma coisa. Um bom desenvolvedor não é aquele que conhece profundamente uma linguagem de programação. Isso é fundamental, mas não é suficiente. Monitorar a aplicação é como cuidar do carro daquela analogia. Em DevOps, não basta criar e lançar uma aplicação: você precisa garantir que ela esteja funcionando bem o tempo todo. O **monitoramento** te ajuda a identificar problemas antes que eles afetem os usuários, a entender o desempenho da aplicação e a garantir que os serviços estejam funcionando como deveria.

Métricas

Eu suponho que você esteja estudando aqui para poder evoluir na sua carreira, certo? Isso, muitas vezes, significa trabalhar em uma boa empresa – seja como um funcionário, seja como um dono. Independentemente da sua posição, você em algum momento precisará saber responder à pergunta: *"como estão os nossos sistemas?"*. Essa é uma pergunta muito comum e importante no ambiente de trabalho. Ninguém espera respostas vagas ou baseadas no chute – uma pessoa gestora não reagiria bem a uma resposta como "Eu não sei", ou "Eu **acho** que está tudo bem". É importante termos respostas precisas e confiáveis. Para isso, precisamos de métricas.

As métricas nos dão dados concretos sobre o desempenho e a saúde dos sistemas, permitindo avaliar o que está funcionando bem e o que precisa ser melhorado. Monitorar os sistemas com métricas significa coletar continuamente dados sobre a operação, medindo, por exemplo, o tempo de resposta, a frequência de *deploys*, a taxa

de falhas, entre outras. Elas são a base para entender o estado atual dos sistemas e tomar decisões informadas. Claro: para isso, também precisaremos de métricas robustas. De nada adianta termos uma boa métrica se não medimos ela direito, não é?

Em DevOps, temos algumas métricas que usamos nas empresas. Existem quatro métricas que definem a velocidade e a estabilidade da entrega de *software*. Essas métricas foram preparadas pela equipe Dora (DevOps Research and Assessment), uma equipe originada no Google.

A tabela abaixo mostra a *performance* de várias empresas que responderam à pesquisa da Google em 2023. Veja que existem quatro níveis de desempenho: **baixo**, **médio**, **alto** e **elite**. Veja como essas métricas são distribuídas para cada nível. Como um exemplo, uma empresa com baixo desempenho em DevOps demora de um mês a seis meses para arrumar as coisas quando se tem um erro no *deploy*. Já uma empresa com alto desempenho demora menos de um dia.

Tabela 1 - Panorama das empresas que responderam à pesquisa

Nível de desempenho	**Frequência de implantação**	**Tempo de lead das mudanças**	**Taxa de falha nas mudanças**	**Tempo de recuperação após falha ao implantar**	**% de participantes**
Elite	Sob demanda.	Menos de um dia.	5%.	Menos de uma hora.	18%.
Alto	Entre uma vez por dia e uma vez por semana.	Entre um dia e uma semana.	10%.	Menos de um dia.	31%.
Médio	Entre uma vez por semana e uma vez por mês.	Entre uma semana e um mês.	15%.	Entre um dia e uma semana.	33%.
Baixo	Entre uma vez por semana e uma vez por mês.	Entre uma semana e um mês.	64%.	Entre um mês e seis meses.	17%.

Fonte: Services Google (2023)

Agora, vamos entender melhor o que são essas quatro métricas?

IMPORTANTE

Empresas menos estruturadas raramente conhecem essas métricas. De fato, algumas culturas organizacionais estimulam a falta de priorização, o estresse dos funcionários, e a desorganização da gestão por falta de conhecimento técnico (ex.: TI-como-pastelaria, ou o "método de projetos" go-horse). Às vezes um gestor seu pode querer adotar uma postura de que sabe de tudo, mas nunca ter ouvido falar nestas métricas. Se você se encontra em um cenário assim e gostaria de melhorar estas métricas, sugiro reduzir o tamanho dos deployments (ex.: o tamanho dos pull requests). Em vez de implementar grandes alterações de uma vez, dividir um recurso em várias pequenas alterações independentes pode ser bom para você. Isso torna as mudanças mais fáceis e ajuda na recuperação em caso de falha. Você pode não necessariamente conseguir mudar a cultura da empresa em que está, mas pode tentar melhorar as coisas para você mesmo.

Capacidade de Processamento (entregando as coisas rápido)

As métricas de capacidade de processamento tratam da rapidez e a eficácia com que as mudanças são implementadas e qualquer problema é detectado e resolvido. Temos duas métricas principais aqui:

1. **Tempo de *lead* das mudanças**

Esse é o tempo que leva desde o início de uma mudança até a sua implantação (ou *deploy*) em produção. Para entender isso melhor, vamos supor que você fez uma alteração no código. O tempo de *lead* inclui todo o processo desde o momento em que essa alteração é feita até que ela esteja rodando no ambiente de produção.

Pontos importantes a considerar nesta métrica:

- **Revisões de código por pares:** a sua equipe integra revisões de código por pares (também chamado de *peer review*) ao processo? Ou, em outras palavras: algum desenvolvedor da sua equipe dá uma olhada no trabalho de outro desenvolvedor antes de aprovarem qualquer coisa?
- **Tempo de revisão:** quanto tempo leva desde que o código é escrito até ser revisado?
- **Tamanho dos lotes de revisão:** qual o tamanho médio dos lotes de revisão de código? Lotes menores tendem a ser revisados mais rapidamente. Aqui, um lote pode ser um *pull request*, por exemplo.
- **Equipes envolvidas:** quantas equipes estão envolvidas nas revisões? E quantas localizações geográficas diferentes essas equipes cobrem?
- **Automação de qualidade:** a sua equipe aprimora a automação de qualidade do código com base nas sugestões das revisões de código?

Aqui, a ideia é a seguinte: **quanto mais tempo entre a escrita e a revisão do código, pior será a eficiência do desenvolvedor e na qualidade do *software* entregue**. Múltiplas equipes em diferentes locais geográficos podem prolongar o processo, diminuir o engajamento e aumentar os custos.

2. **Frequência de Implantação (deploy)**

Essa métrica é mais simples e mede a frequência na qual as mudanças são enviadas para produção. Alta frequência de *deploy* significa, geralmente, uma capacidade de entrega rápida. Ou seja: **quanto mais *deploys* em produção, melhor**.

3. **Estabilidade (entregando as coisas com qualidade)**

Já as métricas de estabilidade estão relacionadas à qualidade das mudanças implantadas. Em um ambiente de entrega contínua (CD), a estabilidade é um indicativo-chave de sucesso. Também temos duas métricas principais aqui:

- Taxa de falha em mudanças:

Essa taxa mede a frequência pela qual uma mudança implantada causa uma falha que exige intervenção imediata. Menos falhas significam maior qualidade nas mudanças implementadas. Ela é simples: se a cada 100 *deploys* existiram 5 *deploys* que exigiram

alguma remediação (ex.: *hotfixes*, reverter as alterações, *patches*, *bugfixes*), então a sua taxa de falha seria de 5%. Quanto menor, melhor.

- Tempo de recuperação após uma falha:

Essa métrica indica quanto tempo é necessário para se recuperar de uma falha de implantação (*deploy*). Isto é: alguém fez um *deploy* que ocasionou um defeito, ou um incidente. Nesse caso, quanto tempo demoraria para o *software* voltar a funcionar como deveria? Aqui, quanto menor esse tempo, melhor. O ideal seria em menos de uma hora, mas existem empresas em que esse processo pode demorar alguns meses.

DICA

Até meados de 2023, essa métrica também era conhecida como **MTTR** (mean time to recovery, ou tempo médio de restauração). Contudo, há uma certa confusão na comunidade sobre se o M significa **média** ou **mediana**. Por isso, a equipe Dora renomeou essa métrica para **tempo de recuperação após uma falha**.

Soluções de Mercado

Cada empresa terá uma solução específica para tratar de monitoramento. De fato, é comum cada empresa também fornecer um guia rápido de como utilizar cada uma dessas soluções, e várias delas geralmente são pagas. Vamos apresentar algumas dessas soluções a você antes de criarmos algo nosso, beleza?

Prometheus e Grafana

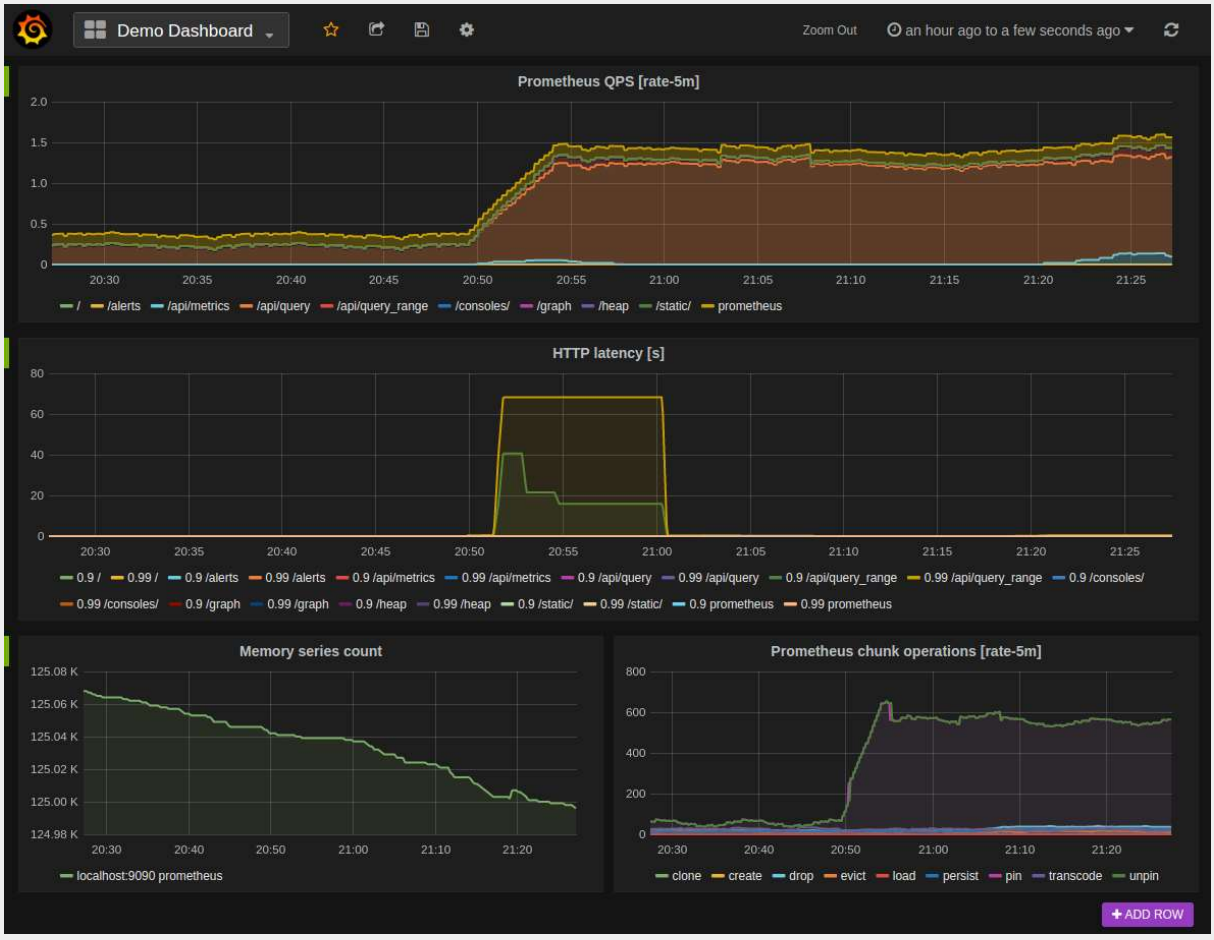
Primeiro, o Prometheus e o Grafana. O **Prometheus** é uma ferramenta *opensource* de monitoramento e alertas, desenhada especialmente para monitorar a *performance* de aplicações e infraestrutura. A principal ideia do Prometheus é coletar métricas de diferentes serviços e armazená-las em um banco de dados de séries temporais. Ou seja: ele coleta dados em intervalos regulares, como o uso de CPU, memória, número de requisições, entre outros. Aqui, as palavras-chave são **coletar e armazenar**.

Depois de coletar essas métricas, é aí que entra o **Grafana**. O Grafana é uma ferramenta de visualização de dados que se integra com Prometheus (ou outros softwares) para criar dashboards interativos. Com o Grafana você pode criar gráficos e painéis que mostram, de forma visual, o desempenho dos seus serviços, ajudando a identificar problemas rapidamente.

Veja um exemplo do Prometheus e Grafana abaixo. Neste exemplo, o Prometheus coletou para uma aplicação de exemplo (como um site, ou um serviço de backend) informações como: quantidade de consultas por segundo (*queries per second,* ou QPS); a latência para requests HTTP, o uso de memória e as operações de chunk por segundo. Estas informações ficam salvas no banco de dados do Prometheus, e o Grafana é a solução visual para vermos isto de um jeito bonito na tela.

Figura 1 – Exemplo do Prometheus e Grafana

#ParaTodosVerem

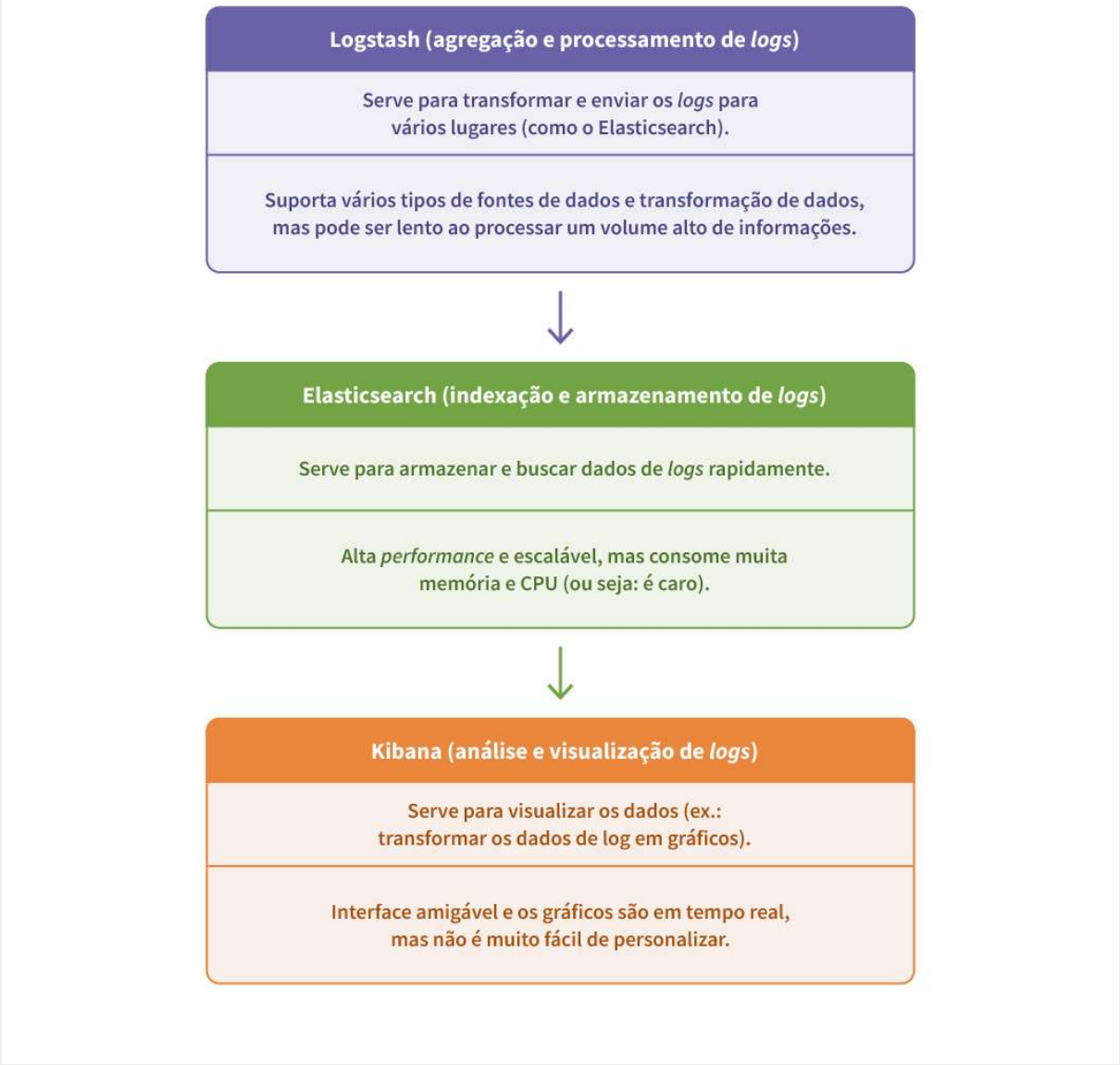


Fonte: Prometheus, 2015.

Gerenciamento de logs

Contudo, o Prometheus e o Grafana não são as únicas soluções. Consideremos os ***logs***, por exemplo. Eles registros detalhados do que está acontecendo dentro do seu sistema. Eles operam de forma parecida como um diário de bordo em que tudo é registrado: desde erros, até as operações bem-sucedidas. Uma das soluções mais populares para implementarmos isso é a "stack ELK": nome usado no mercado para designar três *softwares* que trabalham juntos: **Elasticsearch, Logstash e Kibana**. Os três são da mesma empresa, chamada de **Elastic**.

#ParaTodosVerem



Fonte: O autor (2024)

Agora, nem todas as empresas podem ou querem pagar pela implementação da stack ELK em seus ambientes. É comum encontrarmos desenvolvedores que chamam a stack ELK de "monstro", dada a sua complexidade. De fato, existem várias outras ferramentas que as empresas usam para gerenciamento de *logs*. Alguns exemplos incluem:

a. **Splunk**: ele é uma solução robusta e bastante popular, conhecida por sua capacidade de análise e busca em grandes volumes de dados. Ele costuma ser uma plataforma mais amigável, enquanto a stack ELK possui mais possibilidades de customização para as finalidades da empresa.
b. **Loki**: ele foi criado pela mesma equipe do Grafana, e se integra bem a ele. A ideia é usá-lo como substituição ao Elasticsearch e o Grafana no lugar do Kibana. Para empresas menores, ele pode ser uma boa alternativa.
c. **OpenSearch Elastic Stack**: é um *fork opensource* do Elasticsearch e Kibana, oferecendo funcionalidades similares com uma comunidade ativa de desenvolvimento.

Alertas e Notificações

Uma plataforma de visualização de dados, por mais bonita que seja, pode ser entendida como sendo um sistema **reativo**. Isto é: se você não abrir os *dashboards* diariamente para monitorá-la, não poderá descobrir se as coisas estão funcionando como deveriam. Para complementar isso, poderíamos usar sistemas **proativos**: ou seja, que avisem a você se tudo deu certo ou não. Isso é bem interessante quando estamos trabalhando em mais de uma coisa ao mesmo tempo ou, ainda, quando alguns processos de DevOps demoram alguns minutos ou horas para serem concluídos.

Logo, o que acha de implementarmos um sistema de alertas usando o GitHub Actions e o Discord? A ideia é recebermos uma mensagem nos informando se a nossa PR passou por todos os testes (ou não). Vamos lá?

Criando os nossos alertas com o Discord e GitHub Actions

Conclusão

E assim chegamos ao final da nossa unidade de estudo sobre monitoramento em DevOps. Vale reconhecer a importância de monitorar continuamente os processos de desenvolvimento e operações para garantir a eficiência e a estabilidade dos sistemas. Discutimos a importância das métricas da Dora, como tempo de implantação, frequência de implantação, tempo de restauração de serviço e taxa de falhas, que são importantes para avaliar e melhorar a performance das equipes de DevOps.

Também conversamos sobre as mais comuns plataformas de monitoramento, como Prometheus, Grafana, a stack ELK, e alternativas como o Splunk e Loki, cada uma com suas vantagens específicas para diferentes casos de uso. Além disso, avançamos para a criação de alertas personalizados utilizando GitHub Actions e o Discord, demonstrando como integrar monitoramento e comunicação para uma resposta rápida a incidentes. Com isso, você já sabe como usamos soluções de monitoramento em ambientes de DevOps, garantindo que você entenda como é importante mantermos sistemas de alta disponibilidade e desempenho nos nossos empregos.

Referências

FREEMAN, E. **DevOps para leigos**. Rio de Janeiro: Editora Alta Books, 2021.

KIM, G.; BEHR, K.; SPAFFORD, G. O projeto Fênix. Rio de Janeiro: Editora Alta Books, 2020.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **Manual de DevOps**. Rio de Janeiro: Editora Alta Books, 2018.

PROMETHEUS. **Grafana support for Prometheus.** Disponível em: https://prometheus.io/docs/visualization/grafana/. Acesso em: 05 set. 2024.