DevOps

UNIDADE 08

Tópicos Especiais de DevOps

MLOps, DataOps, AIOps, ITOps, CloudOps, NoOps, GitOps

Você já viu em algum lugar algo sobre as variantes do DevOps? Além do DevOps tradicional e do DevSecOps que vimos na semana passada, surgiram nos últimos anos várias especializações como MLOps, DataOps, AIOps, ITOps, CloudOps, NoOps e GitOps. Cada uma dessas variantes tem um foco específico. Por exemplo, o MLOps é útil para cientistas de dados e demais profissionais que trabalham com modelos de *machine learning*. Já o DataOps otimiza a qualidade e a entrega de dados, algo bem importante para empresas que dependem de grandes volumes de informações. E o AIOps usa inteligência artificial para detectar e resolver problemas de TI de forma proativa.

Saber dessas variantes pode ser importante para você, pois cada uma delas aborda desafios específicos e oferece soluções adaptadas para diferentes profissionais. Pensemos: se você está lidando com a complexidade da infraestrutura de TI, o ITOps pode fornecer as ferramentas necessárias para monitoramento e gerenciamento eficazes. Se você trabalha em uma empresa que está migrando para a nuvem, o CloudOps pode ajudar a automatizar e otimizar a gestão de recursos. E para aqueles que querem eliminar a necessidade de operações manuais, o NoOps e o GitOps são caminhos promissores, com o NoOps focando na automação completa e o GitOps utilizando repositórios Git para gerenciar e automatizar mudanças na infraestrutura. Acredito que a tabela abaixo pode te ajudar a entender melhor as diferenças entre estes tipos:

Aspecto	DevOps	MLOps	DataOps	AIOps	ITOps	CloudOps	NoOps	GitOps
Diferença princip	-	Envolve prática	Foco na qualidade e	Usa IA para melhor	Foco no monito	Foco na automação e	Busca eliminar a	Usa repositórios

al para o DevOps clássico		s específicas para o ciclo de vida de modelos de machine learning (ML).	automação de pipelines de dados.	ar e automatizar operações de TI.	ramento e gerenciamento de infraestruturas de TI.	gestão de recursos em nuvem.	necessidade de operações manuais.	Git para gerir e automatizar mudanças na infraestrutura.
Qual é o objetivo?	Melhorar a colaboração entre desenvolvimento e operações.	Acelerar o ciclo de vida de desenvolvimento de modelos de ML.	Otimizar a qualidade e a entrega de dados.	Detectar e resolver problemas de TI proativamente.	Gerenciar e monitorar infraestruturas de TI.	Automatizar e otimizar a gestão de recursos em nuvem.	Minimizar a intervenção humana em operações.	Usar repositórios Git para automatizar mudanças de infraestrutura.
Qual é o foco?	Integração e entrega contínua de software.	Integração e entrega contínua de modelos de machine learning.	Integração e entrega contínua de pipelines de dados.	Automação de operações de TI usando inteligência artificial.	Gerenciamento de operações de TI.	Gerenciamento de operações na nuvem.	Automação máxima para eliminar a necessidade de operações.	Gestão de infraestrutura como código usando Git.

Práticas	CI/CD, automação de deploy, monitoramento contínuo.	CI/CD para modelos de ML, versionamento de modelos, versionamento de bases de dados.	Integração de dados, testes automatizados de pipelines.	Monitoramento baseado em IA, análise de logs.	Monitoramento, gerenciamento de incidentes.	Gestão de recursos em nuvem, automação de provisionamento.	Infraestrutura como código, automação completa de pipelines	Pull requests, monitoramento baseado em repositórios Git
Ferramentas	Jenkins, GitLab CI, Docker, Kubernetes.	MLflow, TensorFlow Extended (TFX), Kubeflow.	Apache Nifi, Airflow, dbt.	Splunk, New Relic, Datadog.	Nagios, Zabbix, SolarWinds.	AWS CloudFormation, Terraform, Google Cloud Deployment Manager.	Serverless frameworks, AWS Lambda, Google Cloud Functions.	Flux, Argo CD, Jenkins X, GitHub Actions.
Benefícios	Redução do ciclo de deploy, aumento da colaboração.	Redução do tempo de desenvolvimento e implantação de modelos de ML.	Melhor qualidade e entrega de dados	Resolução proativa de problemas, aumento da eficiência operacional.	Maior visibilidade e controle sobre a infraestrutura	Redução de custos, maior agilidade na gestão de recursos.	Redução de custos operacionais, maior eficiência.	Controle de versões, maior consistência nas mudanças de infraestrutura.
Desafios	Cultura organizacional,	Integração de fluxos de	Governança de dados, integraç	Complexidade na imple	Escalabilidade, integra	Gestão de custos,	Limitação em cenários	Sincronização de estado

	integração de ferramentas.	trabalho de ML, segurança de modelos.	ão de diferentes fontes de dados.	mentação de IA, gerenciamento de dados.	ção com novas tecnologias.	segurança na nuvem.	complexos, dependência de automação.	entre repositórios e infraestrutura.
Exemplos	CI/CD, testes automatizados.	Treino e deploy automático de modelos de ML, monitoramento de modelos de ML.	Automação da pipeline de dados, monitoramento da qualidade de dados.	Detecção de anomalias, resposta automática a incidentes de sistema.	Gestão de incidentes e monitoramento de infraestrutura de TI.	Provisionamento automático de recursos da nuvem, gestão de custos da nuvem.	Deploy completamente automatizado, arquiteturas serverless.	Atualizações de infraestrutura a partir de commits do Git, deploy automático de recursos baseados em Git.

Chaos Engineering

Você já se perguntou como grandes empresas como a Amazon e o TikTok garantem que os seus sistemas estejam sempre funcionando, mesmo diante de falhas inesperadas? É aí que entra o ***chaos engineering***. *Chaos engineering* é uma disciplina focada em identificar e mitigar falhas nos sistemas distribuídos complexos, mas de forma proativa. **A ideia é introduzir falhas controladas no ambiente de produção, para observar como o sistema reage, e identificar pontos fracos antes que eles causem problemas reais**. Parece um pouco assustador, não é? Mas essa prática ajuda a fortalecer a resiliência do sistema e a preparar a equipe para responder eficientemente a falhas imprevistas.

REFLEXÃO

Agora, você deve estar se perguntando: "Tá, mas o que isso tem a ver com DevOps?"

O *chaos engineering* complementa o DevOps ao fornecer uma maneira prática de testar e validar a resiliência das aplicações que estão sendo continuamente integradas e entregues. Quando a equipe que trabalha com DevOps implementa *chaos engineering*, ela está essencialmente fortalecendo o sistema contra falhas potenciais.

O que acha de pensarmos em um exemplo prático? Imagine que a sua equipe acabou de fazer o *deploy* de uma nova funcionalidade em um aplicativo crítico. Se usarem *chaos engineering*, vocês poderiam simular falhas como a queda de um serviço ou a latência aumentada em um banco de dados para ver como o sistema se comporta. Ao fazer isso regularmente, a equipe se torna mais preparada e confiante para lidar com situações de crise reais. Cada experimento de falha controlada é uma oportunidade para aprender mais sobre o comportamento do sistema e para desenvolver melhores práticas de mitigação de riscos.

Platform Engineering

Platform engineering se concentra na construção e manutenção de plataformas de tecnologia que facilitam o trabalho de desenvolvedores e engenheiros de *software*. **A ideia é criar e gerenciar uma camada de infraestrutura compartilhada e ferramentas que permitem às equipes de desenvolvimento se concentrarem na criação de *software* sem se preocuparem com os detalhes da infraestrutura que existem por trás.** Isso inclui sistemas de integração contínua (CI), entrega contínua (CD), ferramentas de automação e até ambientes de desenvolvimento padronizados.

Platform engineering pega os princípios do DevOps e os aplica à criação de uma plataforma robusta e reutilizável. Enquanto DevOps foca em práticas e cultura para facilitar a colaboração e CI/CD, *platform engineering* fornece as ferramentas e

infraestrutura que tornam essas práticas possíveis. A ideia é que, ao padronizar e automatizar a infraestrutura e os fluxos de trabalho, as equipes de desenvolvimento podem se mover mais rápido e com menos estresse.

Pensemos em um caso: vamos supor que você trabalha em uma *startup* que esteja crescendo muito rápido. Várias pessoas estão sendo contratadas e tem várias equipes de desenvolvimento trabalhando em diferentes partes do produto. Sem uma abordagem padronizada, cada equipe poderia estar usando diferentes ferramentas e configurações, levando a inconsistências e problemas de integração. *Platform engineering* entra em cena para criar uma plataforma centralizada que todas as equipes podem usar. Isso pode incluir um *pipeline* de CI/CD padronizado, ambientes de desenvolvimento consistentes e ferramentas de monitoramento compartilhadas. Com essa plataforma, as equipes de desenvolvimento podem se concentrar na escrita de código e na inovação, sabendo que a infraestrutura que existe por trás é confiável.

eXtreme Go-Horse

#ParaTodosVerem

Fonte: Wikipedia Commons

O logotipo do eXtreme Go-Horse. Sim, é sério.

Acho importante falarmos sobre o *eXtreme Go-Horse*. É uma "metodologia" de desenvolvimento de *software* que surgiu como uma paródia das práticas ágeis, com um foco especial nos comportamentos caóticos que acontecem na indústria de *software*. Em vez de seguir processos bem definidos e melhores práticas, esta "metodologia" adota a filosofia de "faça funcionar de qualquer jeito, não importa como". Ela caminha contra os padrões que discutimos nesta e nas outras disciplinas.

Embora seja tratado com humor, esse conceito traz à tona problemas reais que podem surgir em projetos de *software*, como a falta de documentação, testes inadequados, código desorganizado e decisões tomadas sem planejamento adequado. Essas práticas podem levar a um *software* difícil de manter, *bugs* constantes e, eventualmente, falhas catastróficas. No ambiente de trabalho, seguir essa abordagem pode criar uma cultura de curto prazo, na qual as gambiarras são priorizadas sobre a qualidade e sustentabilidade em longo prazo. Em última análise, isso pode resultar em mais tempo gasto corrigindo erros do que construindo novas funcionalidades.

DICA

O eXtreme Go-Horse não é uma metodologia de verdade, mas, sim, uma paródia. Se você está trabalhando em uma empresa que segue isso, talvez seja o momento de repensar. Veja mais sobre a metodologia em seu site.

Infraestrutura como Código (IaC)

Infraestrutura como código (*infrastructure-as-code*, ou IaC) é uma prática bem interessante para gerenciarmos e provisionarmos infraestrutura em TI. Não sei se você já trabalhou em alguma empresa que fornece em nuvem, como Amazon (AWS), Google (GCP) e Microsoft (Azure). Nesses lugares, é comum usarmos comandos pelo terminal ou pelo *site* deles para provisionarmos recursos, por exemplo, criarmos um ambiente com o Docker, ou um novo banco de dados.

O IaC permite que a infraestrutura seja descrita e gerida usando código, da mesma forma que desenvolvemos *software*. Em vez de configurar manualmente servidores, redes e outros recursos de TI, escrevemos *scripts* e arquivos de configuração que automatizam esses processos. Como você deve imaginar, isso não apenas acelera o

provisionamento de recursos, mas também garante que a infraestrutura seja replicável em diferentes ambientes, como desenvolvimento, teste (ou homologação/aceitação) e produção.

IaC é uma adição interessante a um ciclo usando DevOps. Explico: se DevOps procura integrar desenvolvimento e operações para entregar *software* de alta qualidade com eficiência e rápido, o IaC é uma peça fundamental nesse quebra-cabeça ao aplicar práticas de desenvolvimento de *software*, como controle de versão e testes automatizados, à infraestrutura. Isso significa que qualquer mudança na infraestrutura pode ser revisada, testada e implantada da mesma forma que o código de uma aplicação qualquer.

Pensemos em um outro exemplo: vamos supor que você precise configurar um ambiente de produção com várias instâncias de servidores, load balancers (que distribuem os usuários entre vários dos seus servidores) e bancos de dados. Se você estiver usando IaC, você usaria uma solução como o Terraform ou o Ansible, e escreveria um *script* que define todos esses recursos e suas configurações. Esse *script* pode ser armazenado em um repositório Git, no qual qualquer membro da equipe pode revisá-lo e sugerir melhorias. Quando uma mudança é necessária, como adicionar uma nova instância de servidor, você simplesmente atualiza o *script* e reaplica as mudanças.

Conclusão

E chegamos ao final das unidades de estudo desta disciplina. Ainda que nesta última semana tenhamos tratado de alguns tópicos especiais de DevOps, nas semanas anteriores trabalhamos com todo o ciclo tradicional de um processo de DevOps. Na prática, trabalhamos com Git, CI/CD com GitHub Actions, e testes. Você pode ter percebido que tentamos trazer exemplos práticos e demonstrações em conjunto. Assim, esperamos que você esteja no caminho para ser uma pessoa que desenvolve algoritmos de alta qualidade e com preocupação na performance e na eficiência. Até breve!

Referências

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

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

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