DevOps

UNIDADE 01

Entendendo o que é DevOps

Entendendo o que é DevOps

Olá! Seja bem-vindo à disciplina **DevOps**! É muito provável que você já tenha escutado esse termo durante o andamento do nosso curso. Alguns já possuem uma experiência com isso, enquanto outros ainda não tiveram a oportunidade de trabalhar com DevOps. Não se preocupe: vamos explorar esse mundo juntos, a partir de agora. O nosso primeiro passo será, naturalmente, entender **o que é** DevOps. Vamos lá?

Como você viu no vídeo, DevOps é uma combinação das palavras *development* (desenvolvimento) e *operations* (operações). Algumas pessoas podem interpretar isso como um jeito de garantir uma confiabilidade melhor dos trabalhos, uma qualidade mais alta das entregas, e um desenvolvimento mais rápido dos projetos. Outras pessoas podem entender isso como sendo somente uma forma de jogar mais trabalho nas costas dos desenvolvedores que, além de desenvolver novas funcionalidades, precisam resolver *bugs* enquanto todas as entregas permanecem sendo "para ontem". Antes de você emitir a sua opinião, o que acha de entendermos melhor alguns dos conceitos principais de DevOps?

Por que DevOps?

Vamos pensar no mercado de trabalho atual. As empresas estão sempre buscando maneiras de entregar produtos de *software* mais rápidos, mais baratos, e com mais qualidade. Antes do DevOps existir, era comum termos **uma separação** entre as equipes que criavam *software* e as equipes que cuidavam daquele *software*.

Isto é: tínhamos uma equipe de projetos encarregada de desenvolver código do zero, e o resultado era uma nova versão de um *software*. Isso se assemelha muito a um projeto com a metodologia *waterfall* (cascata): as pessoas trabalham em um projeto do início

ao final. Quando aquele projeto é finalizado, o *software* é lançado e disponibilizado aos usuários. A responsabilidade da equipe do projeto termina naquele momento ou, no máximo, alguns dias depois.

Já a equipe de operações garantia que aquele projeto (para o qual eles não contribuíram no desenvolvimento) está funcionando sem *bugs* e sem problemas. Qualquer novo ajuste, solicitação ou problema é resolvido por essa equipe. Eles não são necessariamente uma equipe de *help desk*, mas são eles que resolvem os principais problemas da aplicação.

E, quer saber? Quase sempre isso era ruim. Esse tipo de organização gera uma separação muito grande entre quem desenvolvia o *software* e quem o colocava em produção. Isso frequentemente causava atrasos, *bugs* e muita frustração. Digo isso por experiência própria, mesmo: já trabalhei em equipes de projetos e de operações. O que percebi trabalhando com pessoas diferentes em empresas diferentes foi:

1. Geralmente é muito mais legal, e muito mais motivador, trabalhar em projetos do que em operações. Pense só: você já pensou trabalhar mais do que dois anos só mexendo no código (bugado, às vezes) desenvolvido por outras pessoas? E, ainda, sabendo que essas pessoas continuarão desenvolvendo código ruim para você resolver? Afinal, elas são promovidas por desenvolverem projetos rápidos, e não por desenvolverem um código bonito. Logo, as pessoas que trabalham em operações podem não necessariamente desenvolver as suas carreiras com a mesma velocidade das pessoas de projetos, o que não é legal.
2. A equipe de projetos precisa obedecer a prazos apertados e negociar com os trabalhos de consultoria. Em TI, consultores são pessoas terceirizadas que ajudam a desenvolver projetos. Contudo, a qualidade do trabalho das consultorias precisa ser garantida pela equipe de projetos, o que pode gerar mais dor de cabeça ao longo do prazo. Afinal, como você se sentiria sabendo que a pessoa ao seu lado está sendo contratada pelo dobro do seu valor-hora para fazer um trabalho pior do que você poderia ter feito? Ou, ainda, tendo que virar noites se aproximando da data final de um projeto?
3. A equipe de operações precisa perder muito tempo para tentar entender a documentação (quase sempre inexistente ou desatualizada) feita pela equipe de projetos. Tem um *bug* que precisa descobrir até o final do dia? Precisa de ajuda de alguém de projetos? Boa sorte.

É para resolver esse tipo de conflito que a cultura de DevOps foi criada. Algumas pessoas também dizem que DevOps é uma metodologia, ou um *framework*. Independentemente do caso, DevOps serve para quebrar este tipo de barreira. A ideia é

permitir que a equipe de desenvolvimento (projetos) e operações trabalhem em conjunto desde o início até o fim do ciclo de vida do *software*.

DICA

DevOps se traduz em práticas e ferramentas que ajudam os desenvolvedores e a equipe de operações a colaborarem melhor, automatizando processos e melhorando a entrega de software. Ou, como muitas empresas fazem, de que **a mesma equipe que cria um software também é responsável por mantê-la**

Em tese, isso não só acelera a entrega, mas também melhora a qualidade do produto. Se pesquisarmos sobre DevOps imagens como a que veremos a seguir serão bem comuns:

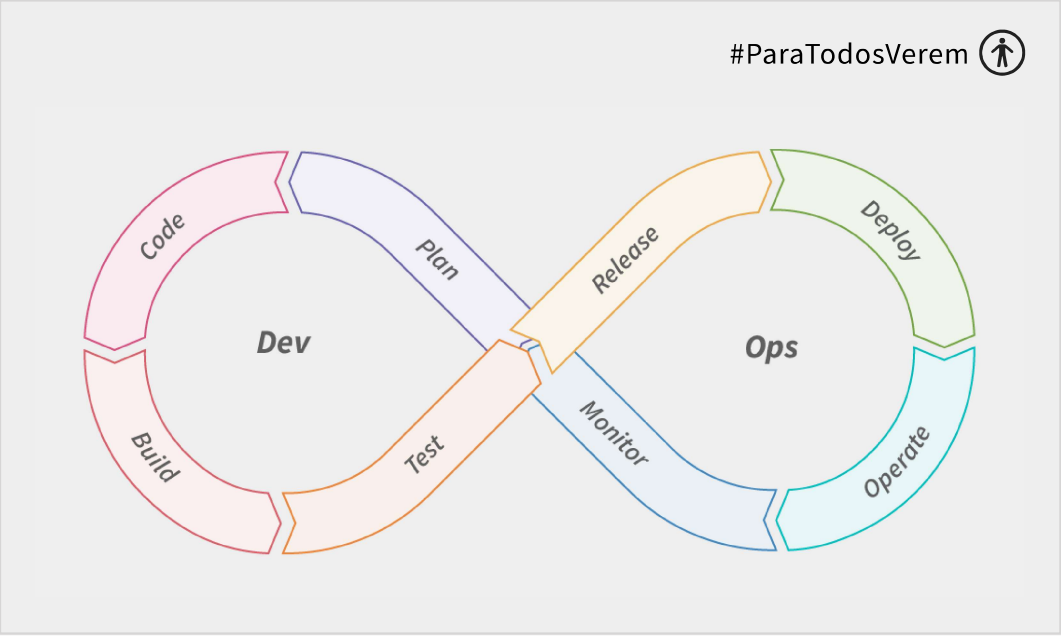

Figura 1: Ciclo DevOps

O que acha de entendermos juntos o que a imagem quer dizer?

Para explicar o ciclo de DevOps, vale a pena pensarmos em um caso real.

EXEMPLO

O que acha de usarmos o iFood como exemplo? O iFood é um

aplicativo de entrega (delivery) de comida, itens de farmácia e de mercado aos seus clientes.

As fases do DevOps

Desenvolvimento (**Dev**): aqui, o foco é criar um *software*, ou uma nova funcionalidade em um produto já existente. Para desenvolver um software, precisamos de ao menos quatro fases:

Fase 1: Planejamento (Plan)

- Imagine que a equipe do iFood está pensando em uma nova funcionalidade, como um sistema de recomendações personalizadas para os usuários. Nesta fase, os gerentes de produto, designers e desenvolvedores discutem e definem o que esta funcionalidade precisa fazer, quais são os requisitos e como ela será integrada ao aplicativo.

- Nesta fase ninguém abre uma linha de código: é uma fase destinada ao planejamento sobre como a funcionalidade deve ser.

Fase 2: Desenvolver (Code)

- Agora que sabemos o que queremos adicionar ao aplicativo do iFood, é hora de começar a construir essa funcionalidade.

- Os desenvolvedores escrevem o código para implementar o sistema de recomendações. Eles criam os algoritmos, desenvolvem a interface de usuário e integram esta nova funcionalidade ao aplicativo existente.

- Esta é a fase "divertida" para quem trabalha com código: afinal, é o momento em que você cria algo do zero.

Fase 3: Compilar (Build)

- Antes de disponibilizar a nova funcionalidade para todos os usuários, precisamos garantir que ela esteja funcionando corretamente.

- Isto significa garantir que o seu código funciona como deveria como, por exemplo, não ter erros no momento da compilação do código.

Fase 4: Testar (Test)

- Nesta fase, os testadores (profissionais de quality assurance, ou QA) colocam a funcionalidade à prova, procurando por bugs e verificando se ela realmente recomenda os pratos certos para os usuários.

- É possível termos testes automatizados para ajudar neste processo de verificação.

- Este também é o momento em que fazemos testes em um ambiente de homologação, também chamado de "aceitação" ou "pré-produção".

- No caso do iFood, seriam uma fase de testes adicional com uma base de dados interna e similar aos dados reais. Seria uma versão de testes com vários clientes e itens fictícios. Afinal, se der um problema aqui não queremos causar nenhum estresse com clientes reais, certo?

Fase 5: Lançar (Release)

- Desenvolvemos e testamos tudo? Então é o momento em que temos já uma versão do aplicativo pronta para lançar.

- Aqui, garantimos que toda a infraestrutura está pronta para a nossa funcionalidade. Afinal, de nada adianta se tudo estiver bonito no celular do usuário, mas não funcionar de verdade ou dar vários erros porque esquecemos de mudar alguma coisa no banco de dados ou no servidor, certo?

- Agora é hora de lançar a nova funcionalidade para os usuários do iFood. Publicamos a atualização do aplicativo nas lojas de aplicativos (App Store e Google Play), permitindo que os usuários baixem e instalem a nova versão.

Operações (***Ops***): a fase de lançamento da funcionalidade marca a passagem do processo de desenvolvimento para a fase de operações. Dito isso, não é porque a funcionalidade foi testada sem erros e enviada aos usuários que o nosso trabalho acabou. Afinal, *bugs* podem acontecer. Agora, precisaremos garantir que tudo está funcionando como planejado.

Fase 6: Implantar (Deploy)

- O aplicativo foi atualizado em todos os celulares. Está tudo certo agora, né? Não.

- O principal objetivo desta fase é garantir que o software liberado seja instalado, configurado e funcione corretamente no ambiente de produção. É quando a nova versão do software realmente começa a ser usada pelos usuários finais.

- Depois que os usuários atualizam o aplicativo, precisamos garantir que a nova funcionalidade funcione corretamente em diferentes dispositivos e configurações. Ajustamos quaisquer detalhes necessários para garantir uma experiência perfeita.

- Nesta fase, é comum fazermos um lançamento em fases: liberamos a funcionalidade aos poucos e para garantir que tudo esteja funcionando como esperado. Se houver algum erro, é muito mais fácil voltar atrás e corrigir.

Fase 7: Operar (Operate)

- Este é o dia-a-dia da equipe de operações. O foco é garantir que o sistema está funcionando e trabalhar para que permaneça assim.

- Aqui, é possível que precisemos alocar mais ou menos recursos como memória, espaço em disco, CPU, ou servidores e containers do Docker.

Fase 8: Monitorar (Monitor)

- Uma vez que a nova funcionalidade está em uso, monitoramos continuamente seu desempenho.

- Coletamos feedback dos usuários, analisamos dados de uso e identificamos qualquer problema que possa surgir.

- É comum medirmos quantos usuários estão usando a nossa funcionalidade, e se os valores negociados fazem sentido. Já imaginou o alerta e o prejuízo que seria se todas as pessoas passassem a comprar hambúrgueres por 50 centavos porque você se esqueceu de remover um cupom de desconto de testes?

Fase 9: Continuar o Ciclo (Plan)

- E voltamos ao início. A partir do feedback e nos dados coletados, a equipe do iFood planejaria melhorias e novas funcionalidades.

- O ciclo começa novamente com o planejamento de novas atualizações, garantindo que o aplicativo continue evoluindo e atendendo às necessidades dos usuários.

IMPORTANTE

Se você já viu alguma das minhas videoaulas em outras disciplinas no passado é muito provável que você já tenha se deparado com frases como "aprender algoritmos e linguagens de programação não é tudo: as outras disciplinas são igualmente importantes". O ciclo de DevOps ajuda a explicar isso: calouros

geralmente estão preocupados somente com aquela fase 2 (Create). Contudo, sem as outras sete fases,não teremos um bom produto. E são perfis com bom pensamento crítico, comunicação interpessoal e olhar sobre a arquitetura que conseguem se dar bem e crescer mais rápido na carreira

Tenho que usar DevOps em tudo?

Para ser sincero com você, não gosto muito daquela imagem do *loop*. Acho ela um pouco confusa, mas ainda assim necessária para entendermos os passos básicos e o conceito de repetição: começamos do planejamento e avançamos até monitorar e retornar ao planejamento de mais funcionalidades ou da correção daquilo que já construímos.

Dito isso, eu prefiro uma imagem como esta para explicar DevOps:

#ParaTodosVerem

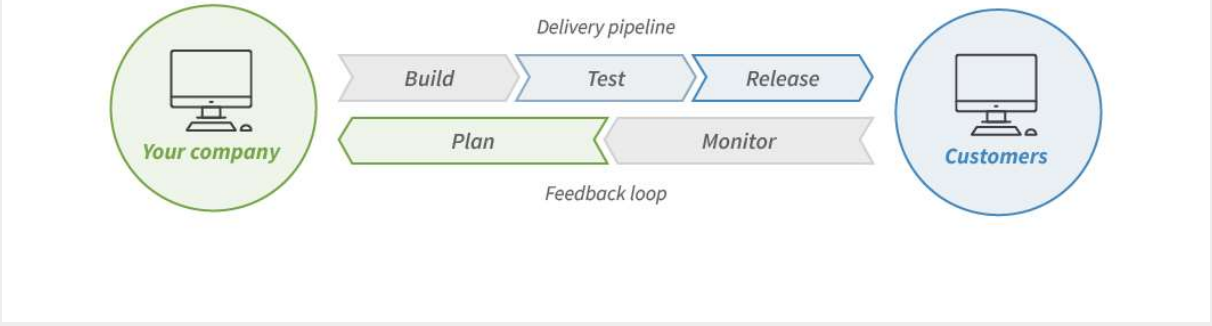

Figura 2: Modelo conceitual simplificado de um pipeline de entrega de software

Vamos entender juntos no vídeo abaixo?

IMPORTANTE

Resumidamente, DevOps = ferramentas + formas de trabalho + práticas para entregar código rápido e com boa qualidade.

Ficou um pouco mais claro? DevOps não é só um conceito, ele se materializa em várias práticas e ferramentas. Algumas das práticas mais comuns incluem:

a) Integração contínua (*continuous integration*, ou CI): isso significa garantir que as mudanças no código sejam **integradas regularmente e testadas automaticamente**.

b) Entrega contínua (*continuous delivery,* ou CD): isso significa **automatizar a entrega do código até a produção** para que possa ser liberado de forma segura a qualquer momento.

c) Infraestrutura como código (*infrastructure as code,* ou IaC): isso quer dizer **gerenciar e provisionar recursos de TI por meio de código**, facilitando a automação e a consistência.

Benefícios para quem usa

Se usado certo, o DevOps pode trazer alguns benefícios. Posso citar, por exemplo:

- **Colaboração e comunicação:** lembra daqueles problemas que comentei entre equipes de projeto e de operações? O DevOps resolve isso no desenvolvimento de *software* ao promover uma comunicação fluida entre os desenvolvedores (quem escreve o código) e os demais analistas (quem mantém o código rodando). De fato, existem equipes que a mesma pessoa que desenvolve é aquela que mantém o código rodando. Logo, ela tentaria desenvolver um bom código para não ter dor de cabeça depois, não é?
- **Automação de processos:** a ideia do DevOps é a de automatizar muitas tarefas repetitivas e demoradas, como testes e implantação de código, usando ferramentas como **GitHub Actions, Docker e Kubernetes**. Isso permite (ao menos na teoria) às pessoas se concentrarem em criar novas funcionalidades, em vez de perder tempo com tarefas manuais.
- **Entrega contínua**: achou um *bug*? Precisa corrigir alguma coisa rapidamente? No DevOps, a entrega contínua ajuda a garantir isso – esse conceito significa que podemos lançar com frequência pequenas mudanças de código. Significa que novos recursos, melhorias e correções de bugs chegam aos usuários rapidamente, aumentando a satisfação e mantendo a competitividade no mercado. Não precisamos esperar aquela "janela de transporte" que comentei no vídeo ou, ainda, um dia específico para fazermos uma mudança no sistema, sabe?
- ***Feedback* rápido:** no DevOps, os feedbacks dos usuários sobre as novas funcionalidades chegam rapidamente. Isso significa que podemos arrumar e melhorar as coisas com velocidade. Temos **ferramentas de monitoramento e histórico** (***logging***) que ajudam a detectar problemas e oportunidades de melhoria em tempo real.
- **Confiabilidade e qualidade:** quanto mais mexemos em alguma coisa, maior a chance de dar problema, não é? Se muita gente mexe todos os dias em um mesmo código, as chances de quebrarmos algo ou de introduzirmos *bugs* é maior. Para evitar isso, em DevOps temos conceitos como a **integração contínua** e os **testes automatizados** garantem que cada alteração de código seja rigorosamente testada antes de ser lançada. Isso aumenta a qualidade e a confiabilidade do *software*.
- **Escalabilidade:** e se você tiver um aplicativo que viraliza, da noite para o dia? A ideia do DevOps é facilitar a escalabilidade do *software*, permitindo a ele suportar um número crescente de usuários sem perder desempenho. Aqui, as **ferramentas de orquestração como o Kubernetes** permitem ajustes aos recursos conforme necessário.
- **Cultura de melhoria contínua:** lembra daquele desenho do *loop* representando o ciclo de DevOps? De fato, trabalhar com DevOps incentiva uma cultura de melhoria contínua, em que a equipe está sempre procurando maneiras de otimizar

processos, aprender com os erros e adotar novas tecnologias e práticas para melhorar o desenvolvimento e a operação do *software*.

DevOps não é todo mundo

É claro que não são todos os lugares que precisam usar DevOps, bem como não são todos os lugares que trabalham em *sprints* usando o *scrum* como uma metodologia ágil. Vejamos alguns casos em que isso não faria muito sentido:

- Grandes empresas e as multinacionais usam *softwares* de ERP (*enterprise resource planning,* ou planejamento de recursos empresariais). Aqui, sistemas como o da SAP e da TOTVS cuidam de toda a empresa, desde a linha de produção até a contabilidade. Sistemas como este praticamente **não podem parar**. Em casos como esses, DevOps não faz muito sentido porque o objetivo é mais a estabilidade e o controle do que a velocidade e a fluidez.
- Você ou alguma pessoa conhecida sua deve ter algum tio ou tia que trabalha com ***softwares* jurássicos**, como algo feito em COBOL, Delphi ou ASP clássico. Ou, quem sabe, com *mainframes*. Para casos como esse, não faz muito sentido trabalhar com DevOps – afinal, a dor de cabeça de fazer a migração dos seus processos para DevOps pode ser tão cara e demorada quanto refazer o projeto do zero.
- Aquelas equipes que também estão em áreas **fortemente reguladas** podem ter problemas em usar DevOps. Algumas áreas de empresas públicas, bancos, empresas na área de saúde e finanças possuem tantas leis e regras que a conversão pode ser uma tarefa impossível. Nesses casos, cada pequena mudança deve ser justificada, e eles preferem a regulação do que a fluidez.

Agora, quem usaria isso afinal? Tenho também alguns exemplos para te dar:

- **Netflix e Spotify**: são alguns dos exemplos clássicos. Eles usam DevOps para implantar milhares de alterações de código por dia – não somente nos seus aplicativos, mas também em seus serviços internos de gestão de conteúdo (como os filmes e músicas), infraestrutura, monitoramentos e afins.
- **Amazon**: não somente no *site* de *e-commerce* ou no Prime Video, mas também na infraestrutura da Amazon Web Services (AWS). Eles usam práticas de DevOps para gerenciar a sua enorme infraestrutura de TI, permitindo escalabilidade e alta disponibilidade de seus serviços.
- **Bancos**: ué, mas não acabei de falar que bancos não usariam tanto isso? Pois é, e isso é verdade. Porém, existem algumas áreas dos bancos que podem usar DevOps. Algumas equipes de inteligência artificial, análise de dados e aplicativos

podem usar estas técnicas sem problemas. É também usual termos *fintechs* usando extensivamente DevOps.
- ***Startups***: de forma geral, elas usam muito DevOps. Afinal de contas, elas precisam crescer (escalar) rápido, testar e produzir rápido, e desenvolver novas funcionalidades com equipes enxutas.

Preparando para colocar a mão na massa

Ao longo desta disciplina, vamos mergulhar mais a fundo nesses conceitos e práticas de DevOps, e você terá a chance de trabalhar com algumas dessas ferramentas. O objetivo é que, ao final da nossa disciplina, você não só entenda o que é DevOps, mas também seja capaz de aplicar essas práticas no seu estágio ou emprego. Ou, quem sabe, na sua própria *startup*.

Só que temos um **pequeno** problema: veja a imagem abaixo.

Figura 3

Uma quantidade grande de logotipos, certo? E isso é apenas uma pequena amostra das ferramentas de DevOps que temos. O objetivo não é que você decore todas elas, mas que você:

1. Saiba que cada empresa pode usar uma solução diferente de DevOps por questões contratuais e de custo.
2. Entenda que você pode se candidatar a vagas que precisem de experiência em DevOps mesmo que você não conheça todas as ferramentas que a empresa pede. Afinal de contas, você trabalhará aqui com ferramentas iguais ou similares. As ferramentas mudam: os conceitos permanecem.

3. Saiba que aprenderemos aqui o conceito geral de DevOps com ferramentas que são conhecidas pela **maioria** do mercado.

DICA

Portanto, encare esta disciplina como sendo algo mais prático do que teórico. A partir de agora nos focaremos muito mais na prática do que em teorias. DevOps é o tipo de disciplina em que 1h investida em prática vale muito mais do que 5h de conteúdos.

CONCLUSÃO

E, assim, chegamos ao final da nossa primeira semana sobre DevOps! Você viu como DevOps não é apenas uma ferramenta ou um conjunto de práticas, mas uma cultura que busca integrar e automatizar os processos entre desenvolvimento e operações. Desde a integração contínua (CI), que garante que cada pedaço do código é testado e verificado, até a entrega contínua (CD) e a implantação contínua (CD), que permitem aos desenvolvedores implantarem novas funcionalidades prontamente ou automaticamente . Vimos como esses processos podem transformar a maneira como o *software* é desenvolvido e entregue.

Mas é importante lembrar que DevOps não é a única forma de trabalhar e resolver todos os problemas. Em algumas situações, como em projetos pequenos ou equipes menos maduras, a implementação completa de DevOps pode não ser necessária. Cada equipe deve avaliar suas próprias necessidades e capacidades antes de adotar essas práticas. Então, caso você já esteja trabalhando em alguma empresa, sugiro absorva o que aprendeu nesta semana, reflita sobre como essas práticas podem se aplicar ao seu trabalho, e não tenha medo de adaptar e experimentar processos até encontrar a abordagem que melhor se ajuste às suas necessidades.

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.