Definindo o DevOps

Crie sua prática de DevOps no GitHub

ESCRITO PELO GITHUB COM ♥

O que vem incluído

ESCRITO PELO GITHUB COM ♥

Introdução ao DevOps

Desde manchetes até descrições de cargos, o DevOps surgiu como uma palavra da moda na última década, e por um bom motivo.

As organizações que adotam o DevOps com sucesso costumam obter grandes ganhos em termos de velocidade de desenvolvimento de software, maior confiabilidade e iterações mais rápidas de produtos, além de terem mais facilidade em escalar seus serviços.

Porém, apesar de suas raízes no desenvolvimento de software, o DevOps é uma prática comercial geral que combina pessoas, processos, práticas culturais e tecnologias para reunir equipes anteriormente em pontos de isolamento com o objetivo de oferecer velocidade, valor e qualidade em todo o **ciclo de vida de desenvolvimento de software** (**SDLC**).

DevOps é uma prática de negócios geral que combina pessoas, processos, práticas culturais e tecnologias.

Isso significa que muitas vezes não existe uma abordagem única para todos. Mas há um conjunto comum de práticas e princípios em qualquer implementação bem-sucedida de DevOps. Este e-book descreve os principais fundamentos e princípios do DevOps, bem como como implementar um pipeline do DevOps em uma organização para agregar valor para os clientes. Em seguida, é fornecida orientação para as ferramentas que melhor complementam a prática de DevOps de uma organização.

O GitHub aborda o DevOps como uma filosofia e um conjunto de práticas que reúnem equipes de desenvolvimento, operações de TI e segurança para criar, testar, iterar e fornecer feedback regular durante todo o SDLC.

Fundamentos de DevOps

O DevOps ajuda as equipes a entregar produtos de alta qualidade com mais rapidez, reduzindo o atrito entre as etapas de escrever, testar e implantar código.

O GitHub oferece uma plataforma geral projetada para ajudar as organizações a adotar o DevOps com sucesso, facilitando a entrega contínua e a melhoria de softwares.

As pessoas frequentemente perguntam o que é um modelo de DevOps, mas isso significa não entender o ponto do DevOps. O DevOps é uma abordagem usada para criar softwares que abrange todo o ciclo de vida de desenvolvimento. É uma mistura de práticas, culturas e tecnologias que visam a oferecer valor aos clientes finais de forma contínua.

Ou seja, não há uma abordagem de tamanho único para DevOps. Sua implementação varia de organização para organização. Apesar disso, o DevOps tem uma estrutura de práticas que todas as organizações aproveitam de várias formas.

Na essência do DevOps está a ideia de que todos os responsáveis por um produto devem colaborar como uma equipe unificada. Em vez de trabalhar de forma separada com desenvolvimento, controle de qualidade e pontos de isolamento de operações, o DevOps reúne pessoas para assumir a responsabilidade de ponta a ponta do planejamento, compilação, entrega e melhoria de softwares.

Em comparação com métodos de desenvolvimento tradicionais em que as equipes de programação escrevem o código, as equipes de testes encontram bugs e as equipes de operações cuidam da infraestrutura, o DevOps pode parecer uma mudança radical. Como prática, o DevOps visa essencialmente a transformar as organizações, reunindo equipes tradicionalmente isoladas em todas as partes do SDLC (ciclo de vida de desenvolvimento de software).

Embora cada jornada de adoção de DevOps pelas organizações seja única, há princípios fundamentais que indicam sucesso. Se você fizer essas coisas, está se saindo bem com o DevOps, mas, dependendo de seu setor, haverá coisas que são específicas e necessárias para a sua prática de DevOps.

Definição dos princípios do DevOps

A implementação do DevOps será diferente em cada organização. O GitHub acredita que é melhor entender o DevOps como uma estrutura para pensar em como agregar valor por meio de software. Ele é mais do que uma única metodologia ou coleção de processos. É fundamentalmente um conjunto de princípios, tanto culturais quanto tecnológicos. Vamos explicar tudo em detalhes:

- **Colaboração:** para ter sucesso, o DevOps exige uma relação de trabalho estreita entre operações e desenvolvimento, duas equipes que tradicionalmente eram isoladas uma da outra. Ao fazer com que essas equipes colaborem estreitamente sob um modelo de DevOps, você procura incentivar a comunicação e a parceria entre essas equipes para melhorar sua capacidade de desenvolver, testar, operar, implantar, monitorar e iterar sua pilha de aplicativos e softwares.

- **Controle de versão:** o controle de versão é parte integrante do DevOps e também da maior parte do desenvolvimento de software atualmente. O objetivo de um sistema de controle de versão é registrar alterações em arquivos automaticamente e preservar registros de versões anteriores desses arquivos.

- **Automação:** a automação no DevOps geralmente significa aproveitar a tecnologia e os scripts para criar ciclos de feedback entre os responsáveis pela manutenção e escalabilidade da infraestrutura subjacente e os responsáveis pela criação do software principal. Desde ajudar a escalar ambientes até criar compilações de software e orquestrar testes, a automação no DevOps pode assumir várias formas diferentes.

- **Versões incrementais:** as versões incrementais são a base das práticas bem-sucedidas do DevOps e são definidos pelo envio rápido de pequenas alterações e atualizações com base na funcionalidade anterior. Em vez de atualizar um aplicativo inteiro, com as versões incrementais, as equipes de desenvolvimento podem integrar rapidamente pequenas alterações na branch principal, testá-las quanto à qualidade e à segurança e, em seguida, enviá-las aos usuários finais.

- **Orquestração:** orquestração refere-se a um conjunto de tarefas automatizadas que são incorporadas em um único fluxo de trabalho para resolver um grupo de funções, como gerenciar contêineres, lançar um novo servidor Web, alterar uma entrada de banco de dados e integrar um aplicativo Web. De forma mais simples, a orquestração ajuda a configurar, gerenciar e coordenar os requisitos de infraestrutura necessários para um aplicativo ser executado com eficiência.

- **Pipeline:** em qualquer conversa sobre DevOps, é provável que você ouça o termo pipeline usado com bastante frequência. Em um sentido mais simples, um pipeline do DevOps é um processo que aproveita a automação e várias ferramentas para permitir que os desenvolvedores enviem seu código rapidamente para um ambiente de teste. Em seguida, as equipes de operações e desenvolvimento testarão esse código para detectar problemas de segurança ou bugs antes de implantá-lo no ambiente de produção.

- **Compartilhamento de feedback (ou ciclos de feedback):** é um termo comum do DevOps definido pela primeira vez no livro O Projeto Fênix, de Gene Kim. Kim explica assim: "O objetivo de quase qualquer iniciativa de melhoria de processo é encurtar e ampliar os ciclos de feedback para que as correções necessárias possam ser feitas continuamente." Em termos simples, um ciclo de feedback é um processo para monitorar o desempenho de aplicativos e infraestrutura para possíveis problemas ou bugs e rastrear a atividade do usuário final dentro do próprio aplicativo.

Esses princípios do DevOps são fundamentais para a criação de uma prática de DevOps bem-sucedida. As práticas de DevOps bem-sucedidas e de alto funcionamento apresentam as seguintes características e benefícios:

- Ciclos de entrega e lançamento mais rápidos
- Mais automação e maior produtividade
- Maior qualidade por meio da colaboração
- Produtos mais escaláveis
- Escalabilidade de processo aprimorada, em que a medição contínua impulsiona a melhoria contínua
- Maior resiliência em aplicativos, infraestrutura e equipes

Esses benefícios se traduzem no objetivo final do DevOps, que é agregar valor para os clientes e mais produtividade, bem como colaboração entre equipes.

Fases importantes da jornada de adoção do DevOps

Ao criar a prática de DevOps da sua organização, entenda que é uma jornada, não um destino. Cada jornada varia dependendo do que a organização precisa. As organizações geralmente começam implementando práticas do DevOps em pequena escala enquanto evoluem e se tornam mais proficientes ao longo do tempo. As fases na tabela abaixo mostram uma evolução típica de uma organização, desde DevOps experimental (em que apenas algumas equipes praticam DevOps) até DevOps nativos (em que o DevOps é adotado em toda a organização, os silos são divididos e há um fluxo fácil de informações entre as equipes).

> Nossa filosofia é criar automação e um excelente DevOps para a empresa que você será amanhã."
>
> **Todd O'Connor**
> Engenheiro sênior de SCM da Adobe

DevOps experimental	DevOps aprendido	DevOps proativo	DevOps nativo
Uma ou duas equipes estão explorando o DevOps. Os silos baseados em função ainda estão em grande parte implantados. Alguma experimentação com automação, mas intervenção manual necessária para cada etapa. Nenhum processo formal.	Algumas partes da organização adotaram equipes de produtos colaborativos. Essas equipes estão usando ferramentas do DevOps para ter um bom efeito, mas cada equipe tem sua própria abordagem. O processo está se formando e aprendeu muito com o que outras organizações estão fazendo.	Todos os novos produtos começam no modelo do DevOps. As medidas estão em vigor para monitorar a eficácia do processo e impulsionar melhorias. O processo está começando a se adaptar às necessidades da organização. A maioria da organização está usando ferramentas do DevOps.	O DevOps é adotado em toda a organização. O processo de DevOps é ajustado com precisão para as necessidades da organização, com atualizações regulares conforme as circunstâncias mudam. Testes, compilações e implantações são automatizados usando ferramentas do DevOps. Todas as equipes estão focadas no produto, com um fluxo fácil de comunicação e colaboração em toda a organização.

Para que uma prática do DevOps atinja todo o seu potencial, isso requer a adesão de todos na organização. No entanto, as alterações ainda podem ser afetadas em menor escala dentro das equipes de produtos individuais. Isso se reflete na etapa DevOps experimental. Muitas vezes, outras áreas da organização percebem os sucessos das equipes que praticam DevOps e querem replicar o que estão fazendo. Isso se reflete na etapa DevOps aprendido. As organizações podem chegar a um ponto em que todo o desenvolvimento de novos produtos esteja seguindo o modelo do DevOps. Nessa fase, as equipes de desenvolvimento e operações estão em sincronia para formar equipes focadas no produto. Isso se reflete na etapa DevOps proativo. A fase DevOps nativo é o objetivo. Aqui, o processo é bem definido, e a comunicação e a colaboração fluem facilmente em toda a organização. Na fase DevOps nativo, todos na organização estão trabalhando juntos para agregar valor para os clientes.

Práticas fundamentais

O funcionamento do DevOps varia de empresa para empresa. Mas há três tópicos centrais que você encontrará em todas as organizações que adotam o DevOps com êxito.

Todos são responsáveis pela qualidade

O DevOps reduz as barreiras entre as disciplinas encontradas em equipes de desenvolvimento de software. Os profissionais costumam focar na criação de produtos de ponta a ponta em vez de produtos isolados e incrementais. Isso significa que o mesmo indivíduo vai colaborar em todo o SDLC, do planejamento à compilação, teste e implantação do produto.

Entrega de código quando estiver pronto

As práticas de desenvolvimento de software tradicionais frequentemente agrupam várias mudanças em lançamentos maiores. Isso significa que os clientes geralmente esperam mais tempo por atualizações de softwares. Isso também dificulta a previsão do impacto indireto causado pelas grandes alterações de código, exercendo mais pressão sobre as equipes de operações e desenvolvimento. Por outro lado, o DevOps favorece as alterações de código incrementais que são mais fáceis de desenvolver, testar e entregar quando estiverem prontas. Assim que um desenvolvedor faz mudanças de código no commit em um projeto, ferramentas de **CI/CD** (**integração e implantação contínuas**) facilitam os testes automatizados, criação de aplicações e integração de código ou relatório de problemas. Muitos profissionais do DevOps estendem o conceito da melhoria contínua em seus próprios trabalhos, avaliando e ajustando os processos ao longo do tempo.

A automação melhora a qualidade e a previsibilidade

Em uma prática de DevOps bem-sucedida, qualquer coisa que pode ser automatizada deve ser automatizada. Isso reduz o risco de erro humano e facilita a escalabilidade dos produtos. As ferramentas são usadas para automatizar a configuração e implantação de infraestruturas, enquanto as ferramentas de análise estática encontram e destacam as vulnerabilidades de segurança. Os profissionais do DevOps se esforçam para automatizar tarefas repetitivas em todas as fases.

Maturidade do DevOps

Inserir o DevOps em sua organização é uma jornada contínua com diferentes níveis de adoção nas diversas fases da entrega de produto. O DevOps é dinâmico, assim como uma empresa precisa, e sua implementação varia de organização para organização.

Isso significa que não existe um modelo de maturidade do DevOps definido. No GitHub, evitamos falar sobre modelos de maturidade do DevOps, pois implica que existe uma lista de verificação que qualquer organização pode usar para realizar o "DevOps". Isso não é verdade. Em sua essência, o DevOps é uma prática contínua. No entanto, há etapas comuns e marcadores de sucesso em que as empresas podem trabalhar, conforme mostrado na figura a seguir.

Figura 1: O modelo de maturidade de DevOps

Esse diagrama de modelo de maturidade do DevOps deve ser usado para traçar aproximadamente onde uma organização está atualmente e o que é preciso para alcançar o próximo nível. Seja qual for a fase do modelo de maturidade em que uma organização está atualmente, o conceito-chave é promover um ambiente que estimule a colaboração, a aprendizagem contínua e a melhoria iterativa.

A próxima seção apresenta a ideia do pipeline do DevOps e as ferramentas e práticas que compõem cada fase. Tenha em mente a definição do DevOps: reunir pessoas, processos, práticas culturais e tecnologias no desenvolvimento de software. É importante observar que a tecnologia é mencionada por último e é apenas uma parte do cenário geral. A tecnologia certamente pode ajudar a influenciar e otimizar suas práticas do DevOps, mas as pessoas e a cultura estão no centro. Uma organização pode ter ferramentas do DevOps de última geração, mas uma cultura colaborativa é necessária para que os benefícios do DevOps sejam totalmente realizados.

"A mentalidade que temos em nossa equipe é que sempre queremos nos automatizar para fazer um trabalho melhor."

Andrew Mulholland, // Diretor de Engenharia da Buzzfeed

O pipeline do DevOps

Um pipeline do DevOps é uma combinação da automação com ferramentas e práticas em todo o SDLC para facilitar o desenvolvimento e a implantação de software nas mãos dos usuários finais.

É importante saber que não há uma abordagem de tamanho único para criar um pipeline do DevOps e que seu planejamento e implementação costumam variar de organização para organização. No entanto, a maioria dos pipelines do DevOps envolve automação, CI/CD, testes automatizados, relatórios e monitoramento.

Alguns conceitos importantes para qualquer pipeline do DevOps são que ele é consistente, contínuo e está sempre em funcionamento. Nada em um pipeline do DevOps deve ser um evento isolado, mas deve abranger um sistema mais amplo em que cada fase é definida por sua repetibilidade.

Vale ressaltar que a compilação de um pipeline de DevOps geralmente é um dos elementos mais palpáveis para as organizações que estão buscando adotar o DevOps, que é definido tanto pela sua dimensão cultural que favorece a estreita colaboração quanto pela automação e ferramentas específicas.

Com a tecnologia certa e o investimento em pessoas e processos, qualquer organização pode compilar um pipeline de DevOps em constante operação, mesmo que seja simples, apenas para começar. Mas sem a adoção completa de uma cultura de DevOps que prioriza o trabalho de desenvolvimento incremental e estreita colaboração multifuncional por todo o SDLC (ciclo de vida de desenvolvimento de software), é pouco provável que as organizações aproveitem o valor total de um pipeline de DevOps.

> Temos um slogan em nossa equipe: não deixe um humano fazer o trabalho de uma máquina. O GitHub nos ajuda a conseguir isso."
>
> **Gabriel Kohen**
> Engenheiro-chefe de software na Blue Yonder

Fases do pipeline de DevOps

Um jeito comum que as pessoas usam para explicar um pipeline de DevOps é fazendo uma comparação com uma linha de montagem. Cada parte do SDLC é analisada para estabelecer um conjunto consistente de processos automatizados e manuais. O resultado é a eficiência e a consistência aprimoradas em termos do resultado geral.

Mas, diferente de uma linha de montagem, o DevOps não é um processo de ponta a ponta com um início e um fim definidos. Em vez disso, o DevOps é um ciclo de melhorias contínuas em que, mesmo após a entrega do software, as melhorias continuam.

Na prática, isso significa que, mesmo que um novo recurso percorra todas as fases do desenvolvimento, o sistema geral (e até mesmo esse recurso) passa por um ciclo contínuo de iteração.

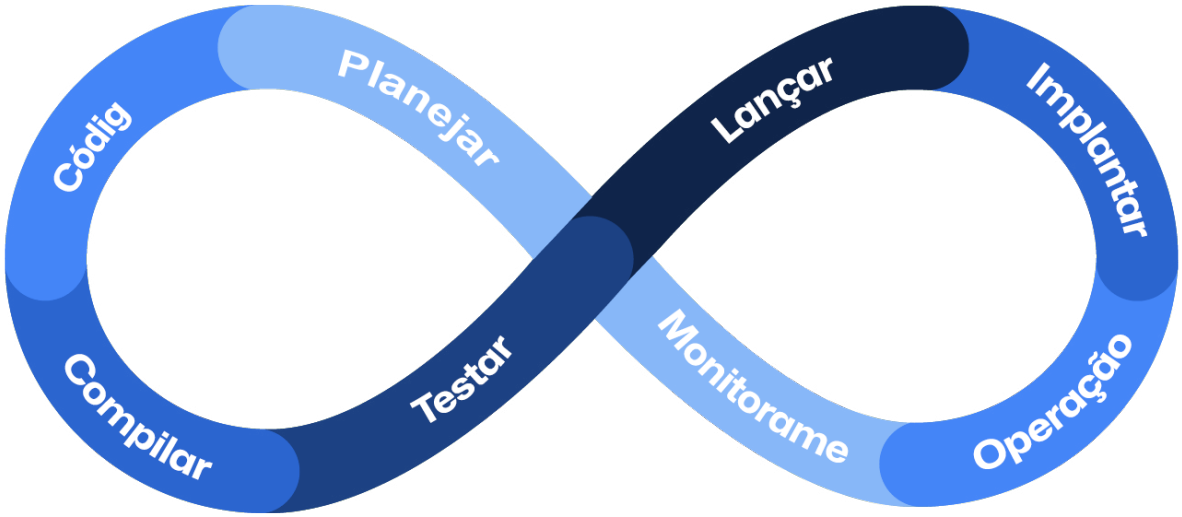

Figura 2: Um pipeline de DevOps

Para entender isso, é mais fácil dividir as fases de um pipeline de DevOps e a forma como elas trocam feedback entre si:

1. **Plano:** cada pipeline de DevOps começa na fase de planejamento, em que os novos recursos ou correções são introduzidos e programados. Nessa fase, o principal objetivo é garantir que pessoas que desempenham funções diferentes dentro da prática de DevOps mais ampla colaborem desde o início, e isso significa trabalhar juntas para entender as necessidades dos usuários, planejar uma solução, entender as implicações da mudança e garantir que ela se encaixe perfeitamente no sistema existente. Aprenda a usar o GitHub para planejar projetos com o GitHub Issues.

2. **Código:** na fase do código, as organizações começam a escrevê-lo de acordo com o planejado e acompanham o trabalho por meio de um sistema de controle de versão como o Git. Nesse ponto, em um pipeline de DevOps, os desenvolvedores podem usar várias ferramentas em seus ambientes de desenvolvimento para gerar consistência no estilo de código e identificar potenciais falhas de segurança. Isso pode incluir o uso de ferramentas como o **IDE** (**ambiente de desenvolvimento integrado**) hospedado em nuvem, geralmente utilizadas para promover consistência em todos os fluxos de trabalho dos desenvolvedores e aumentar a velocidade para a ativação de ambientes de codificação. Descubra como os desenvolvedores codificam no GitHub.

3. **Compilação:** a fase de compilação é quando um pipeline de DevOps passa a operar a todo vapor e começa assim que um desenvolvedor faz commits de mudanças de código em um repositório compartilhado. Nesse ponto, um desenvolvedor pode enviar um pull request para fazer o merge das mudanças de código na base de código. Isso alertará outro membro da equipe para que revise o código antes de aprovar o merge. Ao mesmo tempo, um pipeline de DevOps típico iniciará um processo automatizado de compilação que faz o merge na base de código e começa uma série de integrações e testes unitários. Se algum desses testes ou a compilação em si falhar, o pull request também falhará e o desenvolvedor receberá uma notificação a respeito do problema. Esse nível de automação de fluxos de trabalho em um pipeline de DevOps ajuda as organizações a mitigarem quaisquer problemas de integração de compilação e identificarem bugs ou problemas de segurança com mais antecedência no SDLC. Veja como o GitHub habilita a CI/CD para integração de código e compilação de aplicativos.

4. **Teste:** depois que uma compilação é aprovada, dá-se início à fase de testes em um pipeline de DevOps e a compilação será implementada em um ambiente de teste que se assemelha bastante ao ambiente de produção. Algumas organizações podem escolher adotar **IaC** (**infraestrutura como código**) em seus pipelines de DevOps para automatizar o provisionamento de um ambiente de teste para o preparo. Outras podem ter ambientes de teste

dedicados e predefinidos prontos para novas compilações: a escolha depende muito das necessidades e recursos de computação de uma organização. Uma vez que a compilação é implantada no ambiente de teste, ela será submetida a vários testes automatizados e manuais. Isso pode incluir testes automatizados de segurança, como **DAST** (**teste dinâmico de segurança de aplicação**) e **IAST** (**teste interativo de segurança de aplicação**) para identificar vulnerabilidades e áreas de risco. Também pode incluir um **UAT** (**teste de aceitação do usuário**), em que os membros da equipe usarão a aplicação e observarão potenciais problemas ou bugs que um cliente pode encontrar. Cada organização terá seu próprio pacote de teste automático e manual e estratégias únicas durante a fase de testes no pipeline de DevOps. Essa fase, especialmente, abre espaço para as organizações aplicarem seus testes sem interromper o fluxo de trabalho do desenvolvedor. Aprenda a usar o GitHub para criar um pipeline de testes contínuos.

5. **Versão:** a fase de lançamento marca o momento em que a compilação de um pipeline de DevOps foi totalmente testada e está pronta para a implantação. Além de testes realizados no código em si, o desempenho operacional também é apurado, fazendo com que as organizações se sintam confiantes por sua execução bem-sucedida na produção, sem ser afetada por bugs ou problemas desconhecidos. Nessa fase, algumas organizações decidirão implantar o código automaticamente quando chega nesse ponto em uma prática comumente chamada de implantação contínua. É assim que algumas equipes de software implantam várias mudanças de código por dia. Outros podem, em vez disso, decidir lançar manualmente uma nova compilação em produção e incluir uma fase de aprovação final, e outros ainda assim agendarão lançamentos automatizados para acontecer em determinados dias ou em determinados momentos. Plataformas de CI/CD e outras ferramentas de DevOps permitem que as organizações criem uma cadência de lançamento que funcione melhor para eles e apliquem a automação na fase de lançamento em seu pipeline de DevOps. Veja como as empresas lançam softwares no GitHub.

6. **Implantar:** uma vez lançada, a compilação deve estar pronta para ser implantada na produção. Nessa fase de um pipeline de DevOps, as organizações aproveitam diversas ferramentas para automatizar o processo de desenvolvimento ao provisionar novos ambientes de produção por IaC ou orquestrar uma implantação azul-verde (é aqui que as novas mudanças de código são distribuídas aos poucos para uma porcentagem de usuários em um novo ambiente, enquanto a antiga base de código permanece operacional para outros usuários em um ambiente separado). Uma estratégia de implantação azul-verde também permite que as organizações migrem rapidamente usuários de volta para uma antiga compilação caso algo dê errado. Saiba como implantar seu código com o GitHub.

7. **Operar:** um pipeline de DevOps não termina assim que uma aplicação é implementada: é aí que se inicia a fase operacional e as organizações precisam garantir que tudo esteja funcionando perfeitamente. Essa fase inclui a orquestração de infraestrutura e definições de configuração que vão impor regras para dimensionar recursos automaticamente para atender à demanda em tempo real. Também pode incluir mecanismos que registram as atividades de usuários dentro da aplicação, como registros comportamentais em log e formulários de feedback de clientes. O objetivo da fase de operações está no próprio nome da fase: operar a aplicação e a infraestrutura subjacente com êxito e buscar maneiras de melhorar o perfil operacional do software. Saiba como o GitHub possibilita operações de software.

8. **Monitorar:** com base na fase operacional de um pipeline de DevOps, as organizações definem ferramentas de monitoramento automatizadas para identificar potenciais gargalos de desempenho, problemas em aplicações e comportamento do usuário. Essa fase exige ferramentas de implementação para coletar dados de desempenho de aplicação e infraestrutura e depois retornar os itens práticos para as equipes de produtos, para que resolvam os problemas pendentes ou desenvolvam novos recursos para dar suporte a comportamentos do usuário existentes na aplicação. Embora essa seja a última fase de um pipeline de DevOps, é importante entender que o processo em si é contínuo, isto é, as ferramentas de monitoramento ajudam as organizações a identificar áreas de planejamento e interações adicionais para fornecer feedback pelo pipeline de DevOps. Cada fase se alimenta do próximo em um loop infinito. Descubra como o GitHub oferece recursos avançados de monitoramento para as organizações.

Cada uma dessas fases faz parte da imagem geral de DevOps. Assim como descrito pelo diagrama cada fase flui para a próxima em um ciclo infinito. Isso requer um esforço diligente em cada fase para que uma organização se transforme em uma máquina de DevOps bem lubrificada. Conforme as fases do pipeline de DevOps se tornarem mais naturais para uma organização, elas agregarão valor para os clientes mais rapidamente com melhor qualidade.

Automatizar o máximo possível do processo, do código à produção, é fundamental para o DevOps. A próxima seção aborda os conceitos de CI e CD e como se encaixam no pipeline de DevOps.

Conceitos de integração contínua e implantação contínua

A integração contínua (CI) é uma prática de fundação de DevOps em que as equipes de desenvolvimento integram códigos que mudam de vários contribuidores para um repositório compartilhado. **A implantação contínua (CD)** é uma prática automatizada de lançamento de software em que as alterações de código são implantadas em diferentes fases à medida que são aprovadas em testes predefinidos.

A CI permite que as organizações identifiquem rapidamente defeitos e enviem software de alta qualidade mais rapidamente, e o objetivo da CD é facilitar lançamentos mais rápidos usando a automação para ajudar a remover a necessidade de intervenção humana o máximo possível durante o processo de implantação. Juntos, elas costumam ser chamadas de CI/CD.

As próximas seções abordam os conceitos de CI/CD, bem como as diretrizes que as organizações devem seguir para produzir os melhores resultados. CI e CD compõem uma grande parte do pipeline geral de DevOps, abrangendo o código de quando é gravado, criado e testado pela primeira vez, até a versão. Além disso, os tópicos de conteinerização e segurança no DevOps (DevSecOps) são introduzidos, e veremos como eles podem desempenhar um papel crucial em uma organização de DevOps.

> CI/CD com o GitHub Actions permite a compilação, teste e implantação diretamente do GitHub. Reduzimos o tempo de compilação de 80 para 10 minutos."
>
> **Arquiteto de engenharia no Pinterest**

Integração contínua

A CI busca encorajar ciclos de desenvolvimento mais rápidos e mais eficientes ao resolver um problema importante no desenvolvimento de software: o gerenciamento dos desafios de integração de código em um repositório compartilhado com vários contribuidores.

Quando um desenvolvedor começa a trabalhar em uma atualização de software ou na solução de um bug, ele faz uma cópia da base de código com a qual trabalhar. Isso é feito usando um sistema de controle de versão como o Git, que permite aos desenvolvedores criarem uma cópia, ou "fork", da base de código.

Conforme mais desenvolvedores criam cópias da base de código, a integração das alterações feitas por vários contribuidores torna-se um desafio, especialmente quando a base de código na qual um desenvolvedor começou a trabalhar fica desatualizada e não é mais correspondente ao repositório principal.

Na pior das hipóteses, pode levar mais tempo para integrar com sucesso as mudanças de código do que para fazer as alterações em si, conforme cada desenvolvedor tenta corrigir a falta de correspondência do seu código. Os desenvolvedores geralmente chamam isso de "inferno de integração".

A CI busca impedir que isso aconteça, encorajando os desenvolvedores a integrar as mudanças conforme elas são feitas. A CI também utiliza a automação para aumentar a velocidade na qual os códigos são integrados e testados, para garantir que nenhuma mudança adicional seja necessária e reduzindo o ônus para o desenvolvedor. Essa combinação de integrações de código mais frequentes com a compilação e testagem automatizada ajuda a acelerar o processo de desenvolvimento de software.

Cada empresa vai definir sua prática de CI conforme suas necessidades únicas. Algumas empresas podem introduzir testes automatizados de segurança mais rigorosos, outras podem priorizar o code merge e reservar os testes automatizados mais demorados para uma etapa posterior no SDLC.

Apesar disso, os pipelines de CI eficientes compartilham um conjunto de ferramentas e melhores práticas em comum. Elas incluem:

- **Um repositório de código compartilhado:** um repositório de código compartilhado em um sistema de controle de versão é fundamental para criar uma prática de CI eficiente. Além de servir como um lugar para armazenar códigos, scripts, testes automatizados e tudo mais, os sistemas de controle de versão também permitem aos desenvolvedores criarem vários branches com os quais trabalhar.

- **Code commits regulares:** a automação, testes e ferramentas são importantes para criar um pipeline eficiente, mas sem uma mudança cultural da equipe para priorizar as mudanças de código no commit frequentes, você provavelmente não chegará muito longe. Não existem regras absolutas para a frequência com a qual os desenvolvedores devem fazer commit do código. Porém, uma boa regra geral é que quanto maior for a frequência dos commits individuais, mais produtivo será o ambiente de desenvolvimento.

- **Automação de compilação:** a automação de compilação é um componente crucial do pipeline de CI e permite que as equipes padronizem suas compilações de software. Um processo típico de compilação inclui compilar o código-fonte, gerar instaladores de software e garantir que todos os itens necessários estejam no lugar para apoiar uma implantação de sucesso. Em uma prática de CI, esse processo é automatizado para ajudar a integrar os commits de código na base de código.

- **Teste automatizado:** você pode fazer muitos commits e ter um processo automatizado de compilação. Mas o fato de ser possível executar um programa não significa que ele está sendo executado corretamente. É aí que entra o teste. O teste automatizado é uma parte fundamental dos pipelines de CI. Cada commit aciona um conjunto de testes para identificar bugs, falhas de segurança e problemas de commit. Esses testes são feitos para manter o branch de código principal operacional, ou "verde", e para dar feedback rápido aos desenvolvedores sobre a eficiência de suas mudanças de código. Não há dois modelos de CI iguais. Toda organização vai implementar uma CI de acordo com suas necessidades únicas e requisitos da sua equipe.

Porém, existem alguns passos comuns que todas as organizações precisam dar para implementar a CI com sucesso. Eles se resumem em cinco práticas:

1. **Crie uma estratégia de teste:** cada prática de CI começa com uma estratégia de teste racionalmente necessária e clara. Você precisará considerar quais tipos de teste está executando, o que aciona seu uso para compilar suas sequências de teste automatizado e quais testes são aplicáveis para cada branch de código no qual suas equipes de desenvolvimento estão trabalhando.

2. **Escolha uma ferramenta de CI:** escolher uma plataforma de CI é parte essencial da criação de um modelo de CI em qualquer organização. Essa é a ferramenta que vai acionar suas compilações automatizadas, testes, pacotes e lançamentos.

Será preciso fazer várias perguntas ao selecionar uma plataforma de CI. Elas incluem:

Ela se integra bem com o seu stack de tecnologia atual?

Uma plataforma de CI deve se integrar facilmente com todas as partes do seu stack, desde suas linguagens de programação até seu sistema de controle de versão ou suas ferramentas de terceiros. Também vale a pena considerar quaisquer tecnologias futuras que você possa adotar e procurar uma plataforma que possa crescer com você.

Ela oferece suporte nativo a contêineres?

Os contêineres são parte essencial de uma prática de DevOps e de CI, e é essencial garantir que sua plataforma de CI tenha suporte nativo para aplicações de contêiner como o Docker. Você pode não utilizar contêineres hoje, mas conforme sua prática de DevOps cresce, há uma boa chance de você acabar usando contêineres de alguma forma.

Ela habilita capacidades de teste de compilação de matriz?

A compilação de matriz permite que você teste compilações simultaneamente entre vários sistemas operacionais e versões de tempo de execução. Busque uma ferramenta de CI que ofereça suporte nativo para compilações de matriz, o que ajuda a otimizar seus testes e garante que sua aplicação vai funcionar para todos os seus usuários finais.

Ela oferece cobertura de código integrada e visualização de testes?

A cobertura de código e visualização de testes dão a você uma maneira simples de ver quanto da sua base de código está sendo testada no momento e como os testes existentes estão sendo executados em tempo real e foram executados no passado.

Como ela mapeia seus requisitos de segurança?

A segurança é uma consideração muito importante em qualquer investimento de tecnologia, especialmente se esse investimento vai acabar sendo integrado na sua base de código e nos seus serviços essenciais.

3. **Integre o código assim que possível:** a adoção bem-sucedida da CI começa garantindo que seus desenvolvedores estão integrando seus códigos assim que possível em um repositório compartilhado.

Isso traz dois benefícios:

- Você evita grandes conflitos de integração que podem surgir ao fazer merge de branches mais antigos de volta ao repositório principal.

- Você acaba integrando regularmente mudanças de código menores, o que ajuda na transferência de conhecimento entre suas equipes e simplifica o regime de teste.

Também é importante considerar como é o seu SDLC existente e quais mudanças processuais você pode querer fazer no futuro conforme implementa um pipeline de CI. Essa não é uma conversa sobre se é melhor usar desenvolvimento com branches ou com troncos. Na verdade, é uma conversa sobre garantir que você tem um fluxo de trabalho do desenvolvedor organizado que facilita uma corrente constante de codificação, teste, merge e revisão.

Figura 3: As fases do SDLC

4. **Corrija sua ramificação principal assim que ela quebrar:** como regra geral, você deve consertar sua ramificação principal assim que ela quebrar. Em um modelo de CI, seus desenvolvedores devem estar integrando mudanças de código assim que possível. Isso é bom. Mas se uma mudança de código quebrar seu branch principal e seus desenvolvedores continuarem adicionando mais mudanças, fica difícil identificar o que causou a falha inicial. Para fazer isso, escreva testes que notificam imediatamente os desenvolvedores se uma das suas alterações de código quebrarem o branch principal. Isso ajuda a criar um ciclo de feedback, que é uma prática importante de DevOps. Certifique-se de equilibrar a velocidade de teste com a cobertura do teste no que diz respeito a manter suas compilações verdes, ou operacionais. Se os seus testes demorarem muito, ficará mais difícil encontrar exatamente qual mudança de código causou uma falha. Os melhores pacotes de teste começam com testes simples, como testes de compilação e integração, antes de avançar para testes mais demorados.
5. **Crie novos testes para cada novo recurso que você introduz:** em um modelo de CI, seu pacote de teste deve crescer com seu software ou aplicativo. Isso significa que, conforme você cria novos recursos e prepara atualizações maiores, também deve criar testes para validar esses recursos. Considere escrever os testes assim que novos recursos e correções de bugs forem criados. Isso pode parecer demorado, mas voltar depois de tudo pronto quase certamente levará mais tempo do que escrever os testes conforme você compila o código.

A ideia por trás da CI é integrar o código com a maior frequência possível, em pequenos pedaços. É mais fácil para os desenvolvedores fazer o merge de uma ramificação de código que contém um único novo recurso ou hotfix do que fazer o merge devárias ramificações que têm semanas de trabalho. Quando os conceitos de CI estão sendo seguidos, é mais fácil para as equipes identificar qual bit de código corrompeu a compilação ou fez com que os testes falhassem. Isso permite que os desenvolvedores gastem mais tempo criando valor em vez de solucionar problemas de compilação.

Com a CI, o código é integrado no início e, muitas vezes, para minimizar conflitos de merge que ocorrem quando vários desenvolvedores estão trabalhando na mesma base de código. A próxima seção apresenta a próxima etapa do processo: CD. Com CI e CD juntas, as organizações podem criar, testar e implantar código com frequência e com confiança.

Implantação contínua

A implantação contínua, ou CD, é um dos exemplos mais avançados de automação em uma prática de DevOps. Ela requer uma combinação de testes rigorosos, colaboração sólida entre equipes, ferramentas avançadas e processos de fluxo de trabalho em todo o processo de design e desenvolvimento de aplicações.

Quando implementada com êxito, a CD funciona. Descobriu-se que as organizações de DevOps que adotam a CD enviam código com mais rapidez e superam outras empresas em 4 a 5 vezes.

O DevOps busca aumentar a velocidade da inovação e da entrega de valor aplicando automação a todas as fases do SDLC. Com essa visão, a CD é o objetivo primordial do DevOps: um SDLC totalmente automatizado que efetua push de todas as alterações de código para a produção se forem aprovadas em um conjunto predefinido de testes.

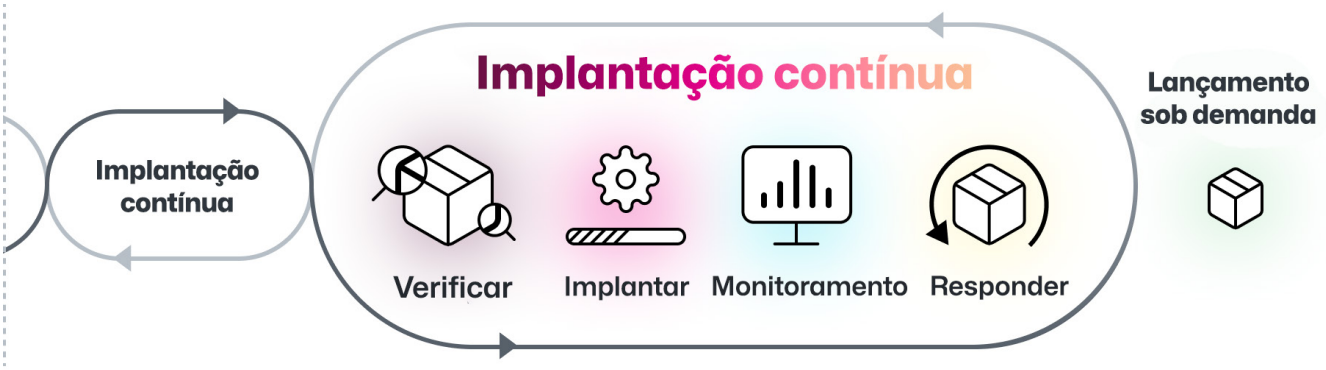

Figura 4: Implantação contínua na prática

De certa forma, criar um pipeline automatizado é uma das partes mais fáceis da adoção de um modelo de CD. Contudo, devido à mudança cultural que isso significa e à maturidade do pacote de teste exigida, pouquíssimas organizações iniciam a jornada de DevOps criando uma prática de CD.

Pensando nisso, é melhor entender o processo e a jornada de alcançar uma prática de CD totalmente funcional.

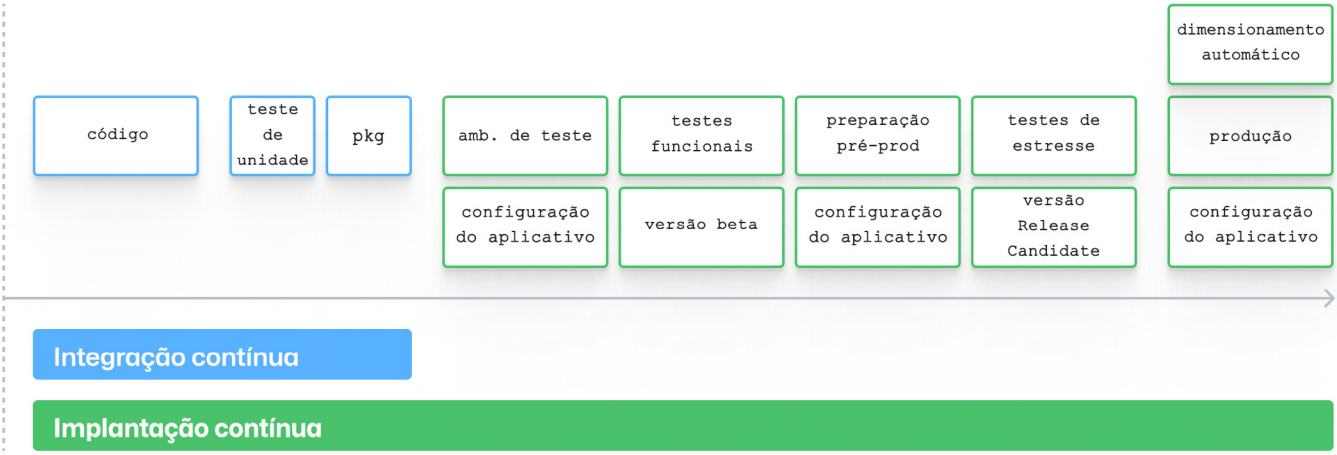

Figura 5: Etapas para a automação em CI/CD

A Figura 5 mostra um mapa de jornada de alto nível sobre como as organizações normalmente começam a pensar em automatizar o SDLC.

Para começar, as organizações precisam criar uma prática de CI. Os elementos fundamentais de uma prática sólida de CI (commits regulares de código, uma estratégia de teste, ferramentas de controle de versão e uma plataforma de CI) preparam o terreno para que as organizações comecem a desenvolver uma prática de CD.

Na sua forma mais básica, a CD reúne compilações, testes e implantações automatizados em um único fluxo de trabalho da versão. O objetivo é automatizar a implantação de compilações de software na produção. Cada empresa precisa identificar a combinação certa de testes unitários, funcionais e de estresse que compõem o pacote de teste. Também é fundamental espelhar as pressões do ambiente de produção em um ambiente pré-produção para preparar e testar com eficácia compilações e a versão Release Candidate.

Acertar em tudo isso leva a uma recompensa significativa: versões mais rápidas e estáveis. O processo também posiciona as organizações para obter CD com um pipeline de CI/CD totalmente automatizado.

Idealmente, uma prática de DevOps torna-se tão aperfeiçoada no regime de teste, nos acionadores de automação, na composição de fluxo de trabalho e nas plataformas de CI/CD que leva à CD de maneira espontânea. Na verdade, a necessidade de intervenção humana para orquestrar um lançamento de software se dissipa com o tempo.

Na prática, porém, obter um modelo de CD durável e escalável exige investimentos significativos em recursos e ferramentas de engenharia. Além disso, embora as plataformas de CI/CD e as ferramentas associadas contribuam muito para estabelecer uma prática de CD, é fundamental uma mudança cultural que enfatize a colaboração entre as equipes e os commits regulares de código.

Fases no pipeline de CD

Não há um "modelo" único para a CD. Cada organização cria um pipeline de CD exclusivo para as próprias necessidades, práticas de desenvolvimento de software e demandas dos clientes.

Apesar disso, existem quatro fases comumente aceitas em qualquer pipeline de CD que toda organização deve incluir nos planos de engenharia. Elas incluem:

1. **Verificação:** a CD baseia-se na CI, e é nesta fase que a CI é interrompida e a CD começa. Depois que são feitos o commit e a integração de um novo trecho de código na base de código, isso aciona o processo de verificação automatizado que executa uma série de testes na compilação de uma versão Release Candidate. Isso pode incluir testes funcionais, de integração, de segurança e do nível de produção para garantir que uma versão Release Candidate funcione após a implantação.

2. **Implantação:** depois que o código é verificado por meio de testes, o processo de implantação automatizado se inicia. Implementações mais avançadas normalmente criam fluxos de trabalho de automação que movem o código para a implantação logo após fazerem commit dele (claro, pressupondo que o código seja aprovado em todos os testes predefinidos na fase de CI).

3. **Monitoramento:** o monitoramento contínuo é um elemento crítico no qual as organizações precisam investir para apoiar a CD. O monitoramento deve ocorrer em todo o SDLC. Contudo, é fundamental ter a capacidade de ver o que está ou não funcionando e receber alertas em tempo real antes, durante e depois das implantações. Ferramentas que ajudam as equipes a visualizar métricas de desempenho e mostrar as sobrecargas do sistema são um investimento útil.

4. **Resposta:** seja para resolver um erro de sistema no nível de produção, seja para identificar um incidente de segurança ou um possível novo recurso para desenvolvimento, ser capaz de responder a eventos é um elemento crítico de um pipeline de CD. Um benefício da CD é o código ser imediatamente lançado para a produção. Isso também significa que as organizações precisam estar preparadas para responder e resolver quaisquer problemas que surjam após a implantação. As métricas comuns usadas para avaliar os tempos de resposta incluem o **MTTR** (**tempo médio de resolução**), que as organizações acompanham para avaliar a melhoria ao longo do tempo.

Já se foram os dias em que o software era distribuído principalmente por disquetes ou CDs. Com o advento de soluções e tecnologias de hospedagem baseadas em nuvem, praticar conceitos de CD permite que as organizações forneçam ativamente valor aos clientes por meio de atualizações de software. Isso não significa necessariamente que cada organização de DevOps precisa implantar código na produção dezenas de vezes por dia. Em vez disso, a ideia é que as organizações possam implantar assim que o código estiver pronto. A implantação na produção não é um evento em organizações com uma prática de DevOps bem-sucedida. O código que está sendo implantado é relativamente pequeno na função, já foi rigorosamente testado, e a automação de implantação não deixa espaço para erros humanos no processo de implantação.

Conteinerização

Não é apenas no código que os problemas podem surgir. O que funciona no computador de uma pessoa pode se comportar de forma diferente no laptop de um colega, ou pior ainda, em um servidor de produção. A conteinerização é um tipo de tecnologia que pode ser usada em práticas de DevOps para garantir que o ambiente de software seja consistente de um computador para outro durante as fases de desenvolvimento, teste e produção.

A conteinerização empacota o código do software com dependências e um sistema operacional na forma de um aplicativo independente que pode ser executado em outro computador. Esses ambientes virtualizados são estruturalmente leves e, de maneira comparativa, exigem pouca potência de computação. Eles também podem ser executados em qualquer infraestrutura subjacente e são portáteis ou podem ser executados consistentemente em qualquer plataforma.

Ao agrupar o código do aplicativo, os arquivos de configuração, bibliotecas do sistema operacional e todas as dependências, os contêineres ajudam a resolver um problema comum no desenvolvimento de software: o código desenvolvido em um ambiente geralmente manifesta mensagens de bugs e erros quando transferido para outro ambiente. Por exemplo, um desenvolvedor pode criar um código em um ambiente Linux e depois transferi-lo para uma máquina virtual ou computador Windows e descobrir que esse código não funciona mais como se esperava. Por outro lado, os contêineres são independentes da infraestrutura host e fornecem ambientes de desenvolvimento consistentes.

Mas o que torna os contêineres particularmente úteis é que eles são fáceis de compartilhar. Usando imagens de contêiner (arquivos que atuam como um instantâneo do código, da configuração e de outros dados do contêiner), você pode criar rapidamente ambientes consistentes em cada fase do SDLC. Isso ajuda as organizações a criar ambientes reproduzíveis que são rápidos e fáceis de trabalhar, desde o desenvolvimento e os testes até a produção.

Agora que temos uma compreensão básica do que são contêineres, a próxima seção destaca os benefícios que a conteinerização traz para o DevOps. Depois, abordaremos os aspectos específicos de CI/CD em que os contêineres podem ter mais efeito.

Os benefícios da conteinerização em DevOps

A essência do DevOps consiste em processos leves e reproduzíveis que automatizam o processo de desenvolvimento de software. No entanto, os aplicativos modernos estão cada vez mais complexos, especialmente à medida que se expandem e incluem muitos serviços diferentes.

Contêineres ajudam a simplificar essa complexidade por meio da maior padronização e repetibilidade, e isso se converte em um SDLC mais rápido, de maior qualidade e mais eficiente. O GitHub fornece ferramentas que ajudam as empresas a adotar e gerenciar contêineres em suas práticas de DevOps. Por meio dessa experiência, o GitHub identificou as principais áreas que as organizações precisam levar em consideração para integrar contêineres com sucesso no SDLC.

Os benefícios da conteinerização incluem:

- **Portabilidade:** até mesmo diferenças aparentemente pequenas no ambiente subjacente podem afetar a forma como o código é executado. É por isso que a frase "Funciona no meu computador" raramente faz sentido e, muitas vezes, é uma piada. É também por isso que a expressão "Escreva uma vez, execute em qualquer lugar" tem sido uma meta recorrente para pessoas que buscam melhorar as práticas de desenvolvimento de software. Contêineres ajudam as organizações a fazer isso, agrupando todos os componentes necessários para um aplicativo em ambientes consistentes e portáteis que facilitam a padronização do desempenho desse aplicativo.

- **Escalabilidade:** os contêineres podem ser implantados e configurados para funcionarem uns com os outros em uma arquitetura de sistema maior por meio do uso de ferramentas de gerenciamento de orquestração, como o Kubernetes. Essas ferramentas também podem ser usadas para automatizar o provisionamento de novos ambientes em contêineres, de forma que eles sejam dimensionados com base na demanda em tempo real. Isso significa que ambientes em contêineres adequadamente configurados podem ser rapidamente ampliados (ou reduzidos) com pouca ou nenhuma intervenção humana.

- **Independente da nuvem:** quando configurados para portabilidade, os contêineres podem ser executados em qualquer lugar, seja um laptop, um servidor bare metal ou uma plataforma de provedor de nuvem. E como os contêineres abstraem as diferenças subjacentes da plataforma, eles reduzem o risco de bloqueio da plataforma. Você também pode usar contêineres para executar aplicativos em várias plataformas de nuvem e mudar de um provedor para outro.

- **Integração ao pipeline de DevOps:** as plataformas de conteinerização geralmente são projetadas para serem inseridas em fluxos de trabalho de automação maiores. Isso as torna ideais para DevOps, em que ferramentas de CI/CD podem criar e destruir contêineres automaticamente para tarefas como testes ou até mesmo implantação em produção.

- **Uso eficiente dos recursos do sistema:** ao contrário das máquinas virtuais, os contêineres costumam ser mais eficientes e exigem menos sobrecarga. Normalmente, não há um hipervisor ou sistema operacional adicional que seja nativo de um contêiner. Em vez disso, as ferramentas de contêiner fornecem estrutura suficiente para tornar cada contêiner um ambiente independente que aproveita os recursos compartilhados do sistema host sempre que possível, e isso inclui o sistema operacional subjacente.

- **Facilite os lançamentos de software mais rápidos:** os contêineres podem ser usados para simplificar aplicativos maiores e mais complexos, dividindo suas bases de código subjacentes em processos de runtime menores que funcionam em conjunto. Isso ajuda as organizações a acelerar cada etapa do SDLC, pois permite que os profissionais se concentrem em uma parte específica de um aplicativo, em vez de trabalharem com toda a base de código mais ampla.

- **Flexibilidade:** os contêineres trazem uma flexibilidade inerente ao SDLC, permitindo que as organizações provisionem rapidamente mais recursos de computação para atender à demanda em tempo real. Eles também são frequentemente usados para criar redundâncias com o objetivo de promover a favorecer maior confiabilidade e aumentar o tempo de atividade dos aplicativos.

- **Maior confiabilidade e segurança do aplicativo:** ao tornarem o ambiente do aplicativo parte do pipeline de DevOps, os contêineres estão sujeitos à mesma garantia de qualidade que o restante do aplicativo. E, embora os contêineres funcionem juntos, o ambiente isolado fornecido por um contêiner facilita a prevenção de problemas em uma parte do aplicativo, impedindo que eles afetem o sistema como um todo.

Os contêineres oferecem uma maneira moderna de desenvolver software com eficiência e em escala. Eles fornecem flexibilidade e capacidade de repetição que combinam bem com práticas fundamentais de DevOps. Na próxima seção, exploraremos como os contêineres podem aprimorar o processo de CI/CD.

Como os contêineres funcionam em CI/CD

Um pipeline de CI/CD pode ser considerado a esteira transportadora que impulsiona o fluxo de trabalho do DevOps. Para ser eficiente, um pipeline de CI/CD deve equilibrar velocidade com precisão. Sem velocidade, um fluxo de CI/CD corre o risco de gerar atrasos à medida que os commits são feitos com mais rapidez do que conseguem passar pelo pipeline. Sem rigor, as pessoas perderão a confiança no pipeline de CI/CD à medida que os problemas entrarem em produção.

Veja como a conteinerização aprimora os dois aspectos do CI/CD em fases principais:

1. **Integração:** ao usar contêineres, você não precisa começar do zero ao integrar alterações de código à base de código mais ampla. Você pode criar um contêiner base que já contenha as dependências do aplicativo e modificá-lo durante a fase de integração.
2. **Teste:** os contêineres podem ser rapidamente provisionados e desativados conforme necessário. Em vez de precisar manter manualmente ambientes de teste explícitos ou aguardar scripts de configuração para criar um ambiente, um contêiner pode ser provisionado e implantado automaticamente em grande escala. Dessa maneira, os testes são executados com mais rapidez e com menos necessidade de intervenção humana para criar ambientes de teste.
3. **Versão:** após a aprovação de todos os testes, a fase de compilação de um pipeline de CI/CD resulta em uma imagem de contêiner que é então armazenada em um registro de contêineres. Quando essa imagem existe, grande parte do trabalho que normalmente seria realizada nas fases de lançamento e implantação já está concluída. Ferramentas de orquestração, como o Kubernetes, então se encarregam de gerenciar onde esses contêineres são implantados e como interagem.

O uso de contêineres em um fluxo de trabalho de CI/CD de DevOps em prol de previsibilidade e escalabilidade reflete princípios fundamentais de DevOps. Embora o uso de contêineres não seja um requisito absoluto do DevOps, as organizações que usam contêineres tendem a funcionar melhor no modelo de maturidade de DevOps. Os contêineres são vistos como um ajuste natural para reduzir o atrito do desenvolvimento de aplicativos modernos, permitindo maior consistência e repetibilidade. É essa consistência e repetibilidade, juntamente com a confiabilidade, que é essencial para qualquer prática de DevOps bem-sucedida.

Segurança no DevOps (DevSecOps)

O DevOps transformou a maneira como muitas organizações compilam e enviam software. Mas, até recentemente, um aspecto do SDLC permanecia fora do DevOps: a segurança. O DevSecOps busca corrigir isso integrando a segurança no SDLC da mesma maneira que DevOps prioriza a qualidade, velocidade e colaboração profunda em todas as fases do desenvolvimento de software.

O DevSecOps busca integrar a segurança em todas as etapas do SDLC. Isso significa, idealmente, que testes relacionados à segurança (automatizados ou não) acontecem em todas as fases, desde a codificação até o merge de branches e compilação e implantações, inclusive na operação do software de produção. Além disso, o DevSecOps avança a ideia de que todos trabalhando em um produto são responsáveis por sua segurança. Isso ajuda as equipes a detectarem vulnerabilidades antes delas chegarem na produção e diminui a necessidade de revisões de segurança manuais em uma fase posterior, o que pode atrasar os lançamentos de software.

Organizações que adotam o DevSecOps tipicamente encontram vantagens que incluem:

- **Risco reduzido de violações de dados:** o DevSecOps busca tornar o código seguro por design. Uma combinação de práticas culturais de código seguro, ambientes de desenvolvedor seguros e testes automatizados de segurança em todo o SDLC ajuda a reduzir a chance de vulnerabilidades de segurança ou falhas chegando ao software em produção.
- **Maior conformidade:** os praticantes de DevSecOps geralmente usam a automação para aplicar o compliance de código e integrar as ferramentas de aplicação de política diretamente na pipeline de CI/CD.
- **Maior confiança nas dependências:** o stack de tecnologia moderno depende muito de códigos de terceiros, muitas vezes vindos de repositórios de pacotes públicos. Praticantes de DevSecOps frequentemente usam ferramentas e testes automatizados para identificar problemas em potencial antes de um lançamento de software.
- **O valor chega aos usuários finais com mais rapidez:** ao criar uma cultura de segurança em primeiro lugar e aplicar verificações automatizadas, o DevSecOps reduz a necessidade de revisões de segurança distintas que atrasam as implantações de código.

Criar uma prática de DevSecOps de sucesso requer a integração da segurança em todas as fases do SDLC. Isso varia de uma organização para outra, e, muitas vezes, indústrias diferentes terão regulamentos diferentes que devem ser seguidos. Com o DevSecOps, incorporar segurança em cada fase do SDLC não significa implementar controles onerosos, o que pode retardar o desenvolvimento de software. Muito pelo contrário. A segurança em práticas efetivas de DevSecOps torna-se parte do lançamento em si, levando a implantações mais rápidas e seguras.

Boas práticas

O DevSecOps argumenta que a segurança precisa fazer parte de todo o SDLC. Independentemente de sua organização já praticar DevOps ou se ela está pensando em como adotar uma cultura de DevOps, apresentamos aqui algumas melhores práticas de base que você precisa para estabelecer uma prática de DevSecOps:

- **Criar uma cultura DevSecOps:** o sucesso do DevSecOps depende de todos assumirem a responsabilidade pela segurança. Isso significa que todas as pessoas no SDLC escrevem código, compilam, testam e configuram aplicativos e configurações de infraestrutura de maneira defensiva. Assim como em DevOps, o DevSecOps floresce em uma cultura aberta onde cada indivíduo trabalha junto para criar o melhor e mais seguro produto possível.

- **Colocar a segurança dentro do produto:** o DevSecOps busca colocar a segurança dentro do produto desde as fases de planejamento inicial até o código implementado no nível de produção. Isso significa que o trabalho de segurança é planejado junto com o trabalho de recursos, e que os profissionais recebem conhecimentos de segurança e fazem testes em todas as fases do seu trabalho de desenvolvimento. O objetivo é fazer com que a segurança faça parte do dia a dia de trabalho da sua equipe.

- **Construir uma prática de modelagem de ameaças:** geralmente as sementes das vulnerabilidades de segurança são plantadas antes que a linha de código seja escrita. Modele ameaças em potencial durante a fase de planejamento e projete sua infraestrutura e a arquitetura do aplicativo para mitigar esses problemas. Testes de penetração periódicos, onde uma pessoa de confiança tenta entrar no seu sistema, podem ajudar a desvendar fraquezas que a modelagem de ameaças pode deixar passar.

- **Automatize para ter velocidade e segurança:** os testes automatizados são usados em todo o SDLC para garantir que as verificações de segurança certas aconteçam na hora certa. Isso dá às pessoas mais tempo para se concentrar em construir o produto principal, enquanto garante que os requisitos de segurança são atendidos.

- **Planeje pontos de verificação de segurança no desenvolvimento de produtos:** identifique pontos de transição no SDLC onde o perfil de risco é alterado. Isso pode ser um ponto no qual o desenvolvedor faz merge do seu código no branch principal, o que pode aumentar o potencial para aquele código ser executado nas máquinas de colegas e eventualmente chegar na produção. Nesse caso, abrir um efetuar pull pode ser um bom evento acionador para verificações automatizadas de segurança, junto com as escalações manuais adequadas.

- **Aborde as falhas de segurança como oportunidades de aprendizagem:** com base em uma cultura de melhoria contínua de DevOps, uma prática de DevSecOps bem-sucedida busca transformar incidentes de segurança em oportunidades de aprendizado. Isso pode ser realizado utilizando logs de auditoria, construindo relatórios de incidente e modelando comportamentos nocivos para melhorar as ferramentas, testes e processos, para proteger ainda mais suas aplicações e sistemas.

- **Acompanhe as dependências:** entender e mitigar as ameaças em potencial das dependências é essencial para a segurança do seu produto. Aplique a mesma modelagem de ameaças e teste automatizado para suas dependências e para seu código interno. O GitHub identificou e compartilhou detalhes de dezenas de milhares de ameaças em software de código aberto, ajudando as organizações e desenvolvedores a serem mais conscientes e evitarem vulnerabilidades.

- **Crie seus recursos de análise e relatórios:** o monitoramento contínuo é parte essencial da prática de DevSecOps, e inclui alertas em tempo real, análises do sistema e monitoramento proativo de ameaças. Ao medir todos os aspectos das suas aplicações e do seu pipeline de DevSecOps, você pode criar um ponto em comum para entender a integridade da aplicação. Painéis de relatório e alertas destacam os problemas logo. Quando um problema ocorre, a telemetria configurada, que pode ser, por exemplo, um registro em relatório no nível da aplicação, oferece insights para a resolução do incidente e a análise de causa raiz.

Criar uma cultura DevSecOps começa fazendo com que a segurança seja responsabilidade de todos. Isso pode ser uma grande mudança para muitas organizações. Tradicionalmente, a segurança é algo que os desenvolvedores deixam nas mãos dos profissionais de segurança. Os conflitos eram frequentes, pois as equipes de engenharia enxergavam as práticas de segurança como um impedimento para o envio rápido de software. Em casos mais extremos, a segurança era apenas um carimbo de borracha no processo.

O DevSecOps busca mudar fundamentalmente essa percepção, fazendo com que a segurança seja tão essencial ao SDLC quanto a escrita de código, execução de testes e serviços de configuração. Assim como o modelo de DevOps reuniu desenvolvedores e equipes de operação, o DevSecOps coloca a segurança na vanguarda. Cada novo recurso ou correção começa com a consideração de suas implicações de segurança. As políticas de segurança e compliance são aplicadas por todos os testes automatizados. Para organizações modernas, o DevSecOps torna-se apenas "DevOps": a segurança é integrada em todos os aspectos do fluxo de trabalho do SDLC.

Planejamento, ferramentas e recursos de DevOps

Automação, monitoramento contínuo e comentários contínuos são partes essenciais do modelo de DevOps.

Como termo genérico, ferramentas de DevOps incluem qualquer aplicativo que automatiza processos dentro do SDLC, melhora a colaboração organizacional e implementa monitoramento e alertas. As organizações geralmente investem na criação de uma "ferramentas e cadeia de DevOps", ou coleção de ferramentas para uso em suas práticas de DevOps, a fim de abordar cada estágio do SDLC.

Uma toolchain de DevOps é um elemento essencial de qualquer prática de DevOps, ajudando as organizações a aplicar automação ao SDLC e melhorar sua capacidade de fornecer software de alta qualidade com mais rapidez. É também um dos aspectos mais tangíveis do DevOps.

Algumas organizações investirão em uma plataforma "tudo em um" para criar sua toolchain de DevOps. Outras integrarão diferentes soluções de ponta para criar uma toolchain. Porém, fundamentalmente, não existe uma abordagem única para o DevOps ou a criação de uma toolchain de DevOps.

> Nossa filosofia é criar automação e um excelente DevOps para a empresa que você será amanhã."
>
> **Todd O'Connor**
> Engenheiro sênior de SCM da Adobe

Guia para ferramentas

Cada fase do pipeline DevOps tem considerações exclusivas que uma ou muitas ferramentas podem ajudar a resolver. Nas próximas seções, as oito fases do pipeline de DevOps são reintroduzidas, juntamente com várias considerações a serem consideradas ao selecionar ferramentas. Uma única ferramenta pode ser responsável por várias fases do pipeline de DevOps. Por outro lado, uma única fase no pipeline de DevOps pode ser representada por várias ferramentas. Independentemente das ferramentas que estão sendo usadas, as organizações mais bem-sucedidas terão fluxo coeso e integração entre cada uma das fases no pipeline de DevOps.

Nesta seção, veremos como as ferramentas de DevOps podem moldar seu planejamento estratégico, comunicação e o roteiro para o futuro.

Ferramentas de planejamento e colaboração para DevOps

Em grande parte, o DevOps busca reunir equipes anteriormente em pontos de isolamento em todas as fases do SDLC, e isso começa na fase de planejamento. Desde aplicativos de chat até ferramentas de gerenciamento de projetos, há várias ferramentas que as organizações podem implementar em suas cadeias de ferramentas de DevOps para melhor alinhar e incentivar a colaboração em uma organização durante as fases de planejamento.

Em geral, as ferramentas de planejamento e colaboração de DevOps geralmente se dividem em dois grupos:

- **Planejamento de produtos e roteiro:** ter um local centralizado para planejar, monitorar e gerenciar o trabalho é um recurso fundamental para qualquer equipe de desenvolvimento moderna e também para as organizações de DevOps. As melhores ferramentas ajudam as organizações a criar planos, sprints e roadmaps, ao mesmo tempo em que podem atribuir e acompanhar o trabalho desde os planos iniciais até o produto final entregue. Precisa de um exemplo? Experimente analisar nossos próprios planos públicos de roadmap de produtos, que criamos usando projetos no GitHub.

- **Comunicação da equipe:** manter a comunicação durante todo o processo de planejamento é essencial para estimular a colaboração, e ter um registro preservado das conversas que levaram a uma determinada decisão pode ser extremamente útil. Ferramentas como o GitHub Discussions, aplicativos de chat e rastreadores de issues que permitem conversas em equipe são fundamentais aqui. O GitHub fornece aplicativos para ajudar sua equipe a se integrar ao Slack ou ao Microsoft Teams. As melhores ferramentas também se integrarão ao planejamento do seu projeto. Isso significa que você pode transformar uma discussão em um trabalho executável ou transformar uma ideia em uma discussão se for necessário um diálogo adicional antes que o trabalho possa começar.

DevOps build tools

Depois que os desenvolvedores fazem commit das alterações de código em um repositório central, começa a fase de compilação, o que significa usar o controle de versão para criar repositórios compartilhados, provisionar ambientes de desenvolvimento e integrar código, entre outras coisas.

Nessa fase, as organizações normalmente podem aproveitar as seguintes ferramentas de DevOps:

- **Controle de versão e origem:** o objetivo de um sistema de controle de versão é registrar alterações em arquivos automaticamente e preservar registros de versões anteriores desses arquivos, que possibilitaria reversões, referências históricas e várias branches de código, permitindo que os desenvolvedores codifiquem e trabalhem em paralelo de maneira colaborativa. Plataformas como o GitHub oferecem controle de versão e controle do código-fonte com recursos como pull requests, que permitem que desenvolvedores individuais recebam avaliações sobre as alterações de código propostas antes que estas sejam integradas à branch principal do código. As melhores plataformas de controle de versão e controle do código-fonte se integram à sua toolchain de DevOps mais ampla e permitem que as equipes de produto colaborem em todo o SDLC.

- **Ambientes de desenvolvimento pré-produção:** em uma prática de DevOps, os desenvolvedores precisam utilizar ambientes virtuais que espelhem a produção da maneira mais próxima possível. Esses ambientes são idênticos entre si e fáceis de provisionar, de forma que todos os desenvolvedores possam criar e testar rapidamente alterações de código em ambientes consistentes. As organizações geralmente utilizam registros e plataformas de conteinerização, como o GitHub Packages, para criar ambientes padronizados de pré-produção para equipes de desenvolvimento. O ideal é que essas plataformas se integrem à solução de controle do código-fonte de forma que, quando um membro da equipe fizer commit de um novo código, isso acione o provisionamento automatizado de um ambiente de pré-produção.

- **IDEs baseados em nuvem:** IDEs baseados em nuvem oferecem ambientes de desenvolvimento abrangentes que são pré-configurados e podem ser rapidamente provisionados. Essas são uma ferramenta cada vez mais popular no DevSecOps (e nos círculos de desenvolvimento de forma mais ampla), pois ajudam a padronizar os ambientes de desenvolvedores, incluindo configurações de segurança entre todas as máquinas. E, como são gerenciados centralmente, os IDEs baseados na nuvem também mantêm o código fora do computador de um desenvolvedor individual, o que pode melhorar a segurança geral do desenvolvimento. Ferramentas como o GitHub Codespaces também apresentam integrações profundas nas principais plataformas de DevOps. Isso pode melhorar as velocidades de desenvolvimento ao encurtar o tempo necessário para ativar um ambiente de desenvolvedor, reduzindo assim a necessidade de esperar pela execução de compilações e testes localmente.

- **IaC:** o aumento da infraestrutura em nuvem, ou **Infraestrutura como Serviço** (**IaaS**), simplificou o provisionamento rápido de recursos para atender à demanda em tempo real. Ele também introduziu uma necessidade entre as organizações de gerenciar uma infraestrutura complexa baseada em nuvem em escala. A IaC baseia-se nas melhores práticas de DevOps para provisionar e gerenciar recursos de infraestrutura em nuvem de um sistema de controle de versão como o GitHub por meio de arquivos YAML. Esses arquivos especificam uma automação de fluxo de trabalho de CI/CD que é acionada por um evento, como uma pull request, um commit de código ou um code merge. Quando esse evento acontece, o fluxo de trabalho automatiza o provisionamento e o gerenciamento dos recursos da infraestrutura de nuvem. Ferramentas como o GitHub Actions oferecem esse tipo de integração, o que facilita o gerenciamento da infraestrutura do seu repositório com a CI/CD.

Ferramentas de CI de DevOps

A CI é a base de qualquer prática de DevOps e combina a prática cultural de commits frequentes de código com automação para integrar esse código com sucesso e criar compilações.

- **CI:** como prática, a CI geralmente envolve fazer commit de várias alterações de código por dia em um repositório compartilhado e usar automação para integrar essas mudanças, bem como aplicar uma série de testes automatizados

à base de código com merge para garantir sua estabilidade e preparar essa base de código para implantação. Esse nível de automação exige uma integração profunda entre uma solução de controle de versão e a plataforma de CI/CD em geral, permitindo que as organizações de DevOps criem pipelines de CI/CD acionados por um commit de código. Quando estiver procurando uma boa solução de CI, certifique-se de que ela se integre facilmente à sua solução de controle de versão. Essa integração é essencial para garantir que você consiga criar um pipeline automatizado que comece assim que suas equipes de desenvolvimento fizerem commit das alterações no código. Um bom exemplo desse nível de integração é a plataforma GitHub, que apresenta CI/CD nativa de plataforma por meio do GitHub Actions e também inclui várias integrações pré-criadas para serviços de CI/CD de terceiros. Você também deve garantir que qualquer plataforma de CI/CD escolhida possa aplicar testes automaticamente em todas as fases do SDLC e inclua suporte nativo para plataformas de conteinerização.

- **Teste automatizado:** ferramentas de teste automatizadas são uma parte essencial de qualquer toolchain de DevOps. A maioria das plataformas oferecerá testes automatizados como um recurso que simplifica a incorporação de testes automatizados às principais partes do pipeline, por exemplo, após o merge de uma alteração de código com a branch principal. O objetivo é ter uma estratégia de testes abrangente com testes unitários básicos, testes de integração e testes de aceitação que sejam aplicados em pontos-chave do SDLC. As melhores ferramentas de testes se integram perfeitamente à sua plataforma de CI/CD, ou fazem parte dela, e oferecem cobertura de código e visualização de testes incorporadas. Você também deve procurar plataformas de testes que possibilitem recursos de testes de criação de matrizes ou que permitam testar compilações simultaneamente em vários sistemas operacionais e versões de runtime. Também é uma boa prática garantir que a solução de teste automatizado de sua escolha acompanhe monitoramento e alertas que se integrem ao aplicativo de chat de sua escolha. Isso significa que, se algo falhar, você poderá receber rapidamente uma notificação e trabalhar para corrigir qualquer problema subjacente. Ferramentas como o GitHub Actions, por exemplo, podem ser usadas para enviar alertas a aplicativos de chat quando um teste falha, possibilitando uma correção mais rápida.

- **Pacotes:** depois que as alterações no código passam por todos os testes em um pipeline de CI/CD, elas são empacotadas em unidades de código independentes e preparadas para implantação. As organizações de DevOps normalmente utilizam um gerenciador de pacotes, como o GitHub Packages, para facilitar a entrega de pacotes de software em um repositório compartilhado em preparação para um lançamento. Gerenciadores de pacotes ajudam a eliminar a necessidade de instalações manuais e ajudam a agrupar dependências de código em um determinado projeto. Existem diferentes gerenciadores de pacotes para diferentes bibliotecas de código, mas o ideal é procurar uma solução que se integre ao seu sistema de controle de versão e à sua plataforma de CI/CD.

Ferramentas de CD de DevOps

A CD se baseia na CI/CD, eliminando a necessidade de intervenção humana ao lançar o software. Em vez disso, uma prática de CD aplica automação a cada estágio do SDLC. Isso significa que, se uma alteração de código passar em todos os testes automatizados, ela será implantada em produção. Essas ferramentas oferecem suporte à CD:

- **Implantação automatizada:** as implantações automatizadas são uma parte essencial da CD e têm uma toolchain que oferece suporte à implantação automatizada. Esses recursos geralmente estão presentes na maioria das plataformas de CI/CD. No entanto, não existe uma abordagem única para criar um pipeline de CD e ele não funcionará da mesma forma com todos os aplicativos ou ambientes. Se você decidir investir em CD, procure plataformas com suporte imediato ao desenvolvimento e ao gerenciamento de vários ambientes. É importante ressaltar que você precisa de uma solução que ajude a proteger contra a "deriva do servidor" ou as diferenças entre ambientes de desenvolvimento, pré-produção e produção. Você também deve considerar uma plataforma que ofereça suporte a implantações azuis/verdes, permitindo migrar lentamente o tráfego de uma versão antiga de um aplicativo para uma nova versão a fim de garantir sua estabilidade na produção. No GitHub, fornecemos painéis de implantação e exibições de visualizações de CI/CD como parte da nossa ferramenta nativa de CI/CD, o GitHub Actions, e consideramos esses recursos essenciais para qualquer toolchain de CD. Isso visa dar às organizações de DevOps visibilidade total sobre diferentes branches de código, resultados de testes automatizados, logs de auditoria e implantações contínuas à medida que eles acontecem.
- **Gerenciamento de configuração:** o gerenciamento de configuração é um processo em que as equipes de tecnologia gerenciam as diferentes configurações ambientais necessárias na infraestrutura principal e nos sistemas de aplicativos ao longo da vida útil do produto. Também é algo que costuma sert emparelhado com CI/CD e controle de versionamento por meio da automação. Assim como um pipeline de CI/CD aplica a automação a todo o SDLC, as ferramentas de gerenciamento de configuração aplicam automaticamente as alterações de configuração em resposta a eventos baseados em acionadores. Esses fluxos de trabalho automatizados podem ser usados para orquestrar e gerenciar clusters de contêineres com plataformas. Repositórios e issues do GitHub facilitam o trabalho dos profissionais de TI com sistemas que produzem arquivos de configuração baseados em texto para IaC e **Configuração como Código** (**CaC**).

Ferramentas de testes contínuos

Em uma prática de DevOps, os testes não param na CI/CD. Pelo contrário, eles são uma prática contínua que se estende por todo o SDLC. E, o mais importante: o DevOps busca substituir equipes de controle de qualidade em pontos de isolamento por uma prática de testes contínuos que aproveita a automação e as estratégias holísticas de testes em todo o SDLC.

Cada organização de DevOps projetará sua própria estratégia de testes contínuos de acordo com as necessidades. O GitHub Actions fornece automação de fluxo de trabalho relacionada a testes e dá suporte a um rico conjunto de ferramentas de teste comerciais e de código aberto. Cada estratégia de teste contínuo aproveitará uma combinação dos seguintes tipos de testes em todo o SDLC:

- **Testes unitários:** testes de unidade são uma maneira de testar pequenas unidades de código para verificar se elas estão estruturadas corretamente com componentes isolados. Eles também são os testes mais fáceis de criar e os mais rápidos de executar, o que os torna um teste fundamental para automatizar em qualquer prática de testes contínuos.

- **Testes de integração:** após o commit de alterações de código em um repositório, os testes de integração garantem a estabilidade da compilação e que a base de código continue funcionando com sucesso. Esses testes são usados para identificar defeitos que surgem quando é feito o merge de diferentes processos de aplicativos e unidades de código. Testes de integração geralmente são automatizados para começar assim que as alterações de código são confirmadas em uma base de código e testam a interação de várias partes de um aplicativo.

- **Testes completos e de regressão:** com base em testes de integração, testes de regressão e ponta a ponta são aplicados depois que uma base de código é empacotada e preparada em um ambiente de pré-produção. Esses testes são usados para verificar se algum defeito antigo, bug ou problema foi reintroduzido por alterações no código. Testes de regressão são comumente usados antes e depois das implantações, para garantir que um aplicativo funcione conforme o esperado e que não contenha problemas identificados anteriormente.

- **Testes de produção:** após a implantação de um aplicativo, testes em nível de produção monitoram a integridade e a estabilidade desse aplicativo e identificam quaisquer problemas antes que eles causem dor de cabeça para os usuários finais. É importante ressaltar que esses testes ajudam as organizações a identificar possíveis problemas em um ambiente de produção com tráfego de usuários em tempo real que não pode ser totalmente replicado em um ambiente de pré-produção.

Ferramentas de monitoramento contínuo e operações de DevOps

Uma prática bem-sucedida de DevOps abrange todas as etapas do SDLC, e isso também inclui o software em nível de produção. Isso significa que as empresas precisam investir em operações principais e ferramentas de monitoramento contínuo para avaliar o desempenho dos aplicativos e da infraestrutura. Se usadas corretamente, essas ferramentas podem ajudar a identificar continuamente possíveis problemas em todo o SDLC:

- **Monitoramento de aplicativos e infraestrutura:** o monitoramento de aplicativos e infraestrutura é um componente essencial de uma prática de monitoramento contínuo de sucesso. As melhores ferramentas oferecem monitoramento automatizado 24 horas por dia, 7 dias por semana, da integridade do aplicativo e da infraestrutura e fornecem alertas aos profissionais de DevOps quando algo dá errado, bem como visibilidade sobre qual pode ser o problema subjacente. O ideal é monitorar a integridade dos aplicativos em ambientes de produção e pré-produção a fim de rastrear quaisquer problemas ou áreas do processo e melhorar o desempenho geral. Isso também vale para sua infraestrutura subjacente, em que o monitoramento pode levar a insights sobre como melhorar sua IaC e políticas de gerenciamento de configuração. Tente procurar uma ferramenta que se integre à sua ferramenta de controle de versão e aos seus aplicativos de chat, para que você possa enviar alertas imediatamente às pessoas certas e criar issues para delinear o escopo do trabalho em busca de uma solução.

- **Logs de auditoria:** a auditoria é uma parte central de uma prática eficaz de operações e monitoramento contínuos, além de resolver quaisquer incidentes, se e quando eles acontecerem. Logs fornecem aos profissionais de DevOps um registro do que aconteceu, onde aconteceu e quando aconteceu, e podem ser essenciais para criar modelos comportamentais que geraram um problema e melhorar a integridade dos aplicativos e da infraestrutura. Procure ferramentas de DevOps que tenham logs dinâmicos e períodos de retenção de auditoria para empoderar suas equipes com as informações de que elas precisam para melhorar os principais serviços e o desempenho dos aplicativos.

- **Rastreamento de incidentes e alterações:** o objetivo principal do DevOps é ajudar as organizações a enviar softwares de alta qualidade com mais rapidez por meio de colaboração e automação profundas. E isso significa que rastrear incidentes e alterações à medida que eles surgem e compartilhá-los com as pessoas certas é fundamental. Para criar uma toolchain de DevOps de sucesso, você deve incorporar ferramentas que revelem incidentes e mudanças na sua plataforma principal de DevOps e repositórios compartilhados. Quanto mais centralizados você puder manter todos os relatórios sobre incidentes e alterações, melhor. O objetivo é criar uma única fonte confiável que facilite a identificação e a correção de problemas.

- **Comentários contínuos:** um elemento essencial do DevOps, o feedback contínuo é uma prática que se concentra em rastrear o comportamento dos usuários e o feedback dos clientes sobre seus principais produtos e em criar dados acionáveis para embasar futuros investimentos em novos recursos e atualizações do sistema. Isso pode incluir dados de pesquisas do NPS sobre como os usuários estão navegando em seu produto. Também pode incluir o rastreamento e a modelagem do comportamento dos usuários no próprio produto. Para criar uma prática de feedback contínuo, você deve identificar as principais áreas do seu produto e até mesmo fora dele, em locais como mídias sociais e avaliações, onde seja possível identificar comportamentos inesperados dos usuários e pontos problemáticos dos clientes. Procure ferramentas que permitam modelar e analisar os comportamentos dos usuários. Você também pode considerar ferramentas de escuta social, que podem ser usadas para rastrear padrões históricos em mídias sociais e sites de avaliação.

Ferramentas de segurança e DevSecOps

À medida que o DevOps evoluiu como prática, ele ressaltou a necessidade de superar as abordagens mais tradicionais de segurança, que geralmente eram isoladas do SDLC principal. Para garantir o envio de códigos de alta qualidade, é importante tornar a segurança uma parte essencial da prática de DevOps. Essa prática é comumente chamada de DevSecOps, que busca integrar a segurança em cada fase do SDLC e torná-la uma parte essencial dos pipelines de CI/CD.

As empresas que investem em DevOps geralmente encontram a necessidade de investir também na criação de uma prática de DevSecOps para garantir a segurança do software. Isso normalmente envolve várias ferramentas que ajudam as organizações a modelar possíveis ameaças e aplicar testes de segurança automatizados às principais fases do SDLC. Embora as organizações geralmente tentem usar ferramentas individuais para criar uma solução, produtos integrados, como o GitHub Advanced Security, podem reduzir o atrito de levar o DevSecOps para suas equipes. Ao complementarem sua toolchain de DevOps com ferramentas DevSecOps, as empresas geralmente buscam as seguintes soluções:

- **Modelagem de ameaças:** aqui está algo óbvio: é muito mais fácil encontrar vulnerabilidades de segurança e possíveis pontos fracos quando você está desenvolvendo um software, e não depois de lançá-lo. A modelagem de ameaças é uma prática que os profissionais de DevSecOps adotam desde as fases iniciais de planejamento do SDLC para antecipar quaisquer problemas e desenvolver planos para resolvê-los. Atualmente, as organizações de DevSecOps também investem em ferramentas de modelagem de ameaças que aproveitam estratégias de automação e monitoramento para identificar proativamente ameaças e esforços de mitigação. As melhores ferramentas pesquisam ameaças de aplicativos e infraestrutura e rastreiam automaticamente alterações na base de código subjacente e na arquitetura da infraestrutura. Procure soluções que possam se integrar à sua toolchain principal de DevOps para fornecer atualizações às pessoas relevantes da sua equipe e mostrar pontuações de avaliação de riscos em todo o SDLC.

- **Painéis de segurança:** ter uma visão única do seu perfil de segurança, incluindo riscos potenciais, cobertura de testes, alertas e muito mais, é essencial para qualquer prática de DevSecOps. Painéis de segurança geralmente são usados para reunir e detalhar todas as informações de segurança relevantes e fornecem uma maneira rápida de fazer a triagem de problemas e atribuir tarefas. No GitHub, incluímos uma página de visão geral da segurança com o GitHub Advanced Security para ajudar a mostrar categorias de riscos em projetos e repositórios, além de detalhes de alertas. O ideal é procurar ferramentas que se integrem à sua toolchain de segurança DevSecOps mais ampla e ofereçam uma visão única do seu perfil de segurança.

- **Teste estático de segurança de aplicação (SAST):** ferramentas de SAST são usadas para avaliar o código antes que ele seja executado, a fim de identificar possíveis riscos ou vulnerabilidades de segurança. É importante ressaltar que essas ferramentas não precisam de um sistema em execução para serem executadas, mas podem ser executadas em uma base de código estática. As melhores ferramentas se integram diretamente em um repositório compartilhado e pesquisam quaisquer vulnerabilidades de segurança, fazem análises de dependências, verificam qualquer senha ou segredo confidencial e identificam erros de codificação antes que eles entrem em produção. Essas ferramentas também facilitam a localização, a triagem e a priorização de correções para quaisquer problemas na sua base de código. O ideal é procurar uma solução que se integre ao seu repositório e que possa ser automatizada para criar issues com base em análises. No GitHub, por exemplo, temos o Dependabot, uma ferramenta de SAST que analisa todas as dependências em busca de vulnerabilidades de segurança conhecidas, além de ser integrada diretamente a todos os repositórios na plataforma.

- **Teste dinâmico de segurança de aplicação (DAST):** DAST são usados para imitar ataques mal-intencionados em um aplicativo e encontrar possíveis vulnerabilidades que possam colocar em risco sua segurança no mundo real. Ferramentas de DAST normalmente analisam aplicativos em ambientes de pré-produção para ajudar os profissionais de DevSecOps a identificar possíveis falhas de segurança antes da implantação no ambiente de produção. Essas falhas geralmente incluem problemas subjacentes que os invasores podem explorar para executar ataques de injeção de SQL e ataques de **cross-site scripting** (**XSS**), entre outras coisas. As melhores ferramentas de DAST se integram à sua plataforma de CI/CD preferida, para que você possa automatizar sua implantação dentro do SDLC mais amplo.

- **Teste interativo de segurança de aplicação (IAST):** soluções de IAST são usadas para identificar e traçar o perfil de riscos e vulnerabilidades na execução de aplicativos, geralmente no início do SDLC, antes de um lançamento. Essas soluções aproveitam a instrumentação de software para monitorar e coletar informações em ambientes de pré-produção por meio de testes manuais e automatizados. As melhores soluções de IAST incluirão ferramentas de **análise de composição de software** (**SCA**) para identificar quaisquer vulnerabilidades de componentes de código aberto.

- **Varredura de imagens de contêineres:** devido às suas arquiteturas leves, os contêineres simplificaram para as organizações de DevOps as tarefas de criar, testar, implantar e atualizar aplicativos de maneira rápida e flexível. Porém, ambientes de contêineres de grande escala também apresentam riscos de segurança devido ao número de áreas de superfície e ao potencial de vulnerabilidades. Para mitigar quaisquer riscos, os profissionais de DevSecOps utilizam ferramentas de varredura de contêineres para identificar problemas no registro de contêineres, examinar clusters de contêineres em runtime e evitar que vulnerabilidades entrem no ambiente de produção. Procure ferramentas que possam ser integradas ao seu pipeline de CI/CD e automatizadas para execução em pontos específicos do seu SDLC antes da implantação, incluindo as fases de compilação, integração e empacotamento.

Ferramentas de monitoramento

O monitoramento é parte essencial de uma prática de DevOps bem-sucedida e uma forma crítica de entender e detectar possíveis problemas antes que cheguem à fase de produção, além de identificar qualquer imprevisto que possa surgir na produção.

Monitoramento contínuo

O monitoramento costumava ser um processo oneroso até pouco tempo atrás. As ferramentas consumiam recursos preciosos do sistema e exigiam intervenção manual. Além disso, os dados fornecidos por essas ferramentas normalmente levavam tempo para serem analisados e administrados. Consequentemente, era comum as organizações monitorarem apenas processos críticos, como problemas de codificação e desempenho em nível de produção.

Hoje, coletar dados é muito mais fácil devido às ferramentas mais avançadas. Por outro lado, a quantidade de dados também aumentou drasticamente. Isso significa que, atualmente, as organizações precisam determinar a melhor forma de gerenciar, interpretar e administrar volumes muito maiores de dados.

O monitoramento contínuo é uma prática que busca sanar esse problema, integrando o monitoramento em todas as partes do SDLC. Seu principal objetivo é possibilitar a detecção rápida de possíveis problemas e fornecer feedback em tempo real.

Uma prática de monitoramento contínuo utilizará uma série de ferramentas e uma série automatizada de testes para avaliar o novo código e o desempenho da produção de uma aplicação, bem como sua infraestrutura subjacente. O objetivo principal é gerar uma visão automatizada 360 graus de todos os sistemas e garantir que as pessoas certas saibam quando e onde intervir.

As melhores práticas de monitoramento contínuo costumam priorizar a coleta do máximo de dados possível para auditar sistemas em sua totalidade e analisar possíveis problemas operacionais, bem como riscos de conformidade e segurança.

Implementação de ferramentas de monitoramento em sua prática de DevOps

Assim como a transição para o DevOps em si, estabelecer uma estratégia de monitoramento de DevOps bem-sucedida requer uma combinação de cultura, processos e ferramentas. E, embora você possa se inspirar no modo como outras organizações gerenciam o monitoramento, o modelo assertivo que você adota será orientado pelas necessidades únicas de sua organização e seu SDLC.

Há muitos frameworks que oferecem orientações sobre quais dados capturar. Entretanto, saber onde implementar o monitoramento é uma questão de otimização. Quais perguntas você precisa responder? Que dados você precisa para obter essas respostas? Como você agirá com base nesses dados? Quem deve estar envolvido?

Capacidades que você deve buscar em ferramentas de monitoramento de DevOps

Há muitas opções de ferramentas para ajudá-lo a incorporar o monitoramento em sua prática de DevOps. Escolher os produtos certos depende do formato de seu SDLC e da infraestrutura de sua aplicação. Porém, existem duas questões iniciais principais que você deve considerar ao avaliar as ferramentas de monitoramento:

É acionável? A ferramenta se integra novamente ao seu pipeline de DevOps e com suas outras ferramentas para viabilizar a automação de ações e alertas com base em seus dados?

Ela proporciona algo novo? Gerar mais dados é fácil, mas esse aumento demandam atenção, ocupa espaço de armazenamento e precisa ser mantido. Em vez de ferramentas que tragam ganhos irrelevantes, priorize aquelas que abram novos caminhos de monitoramento.

Há várias áreas em que o monitoramento deve ser implementado em uma prática de DevOps, e algumas são mais óbvias do que outras. As ferramentas para monitorar a infraestrutura e a rede são usadas para entender como as restrições, como memória e CPU, estão afetando seu aplicativo. Na camada do aplicativo, as ferramentas de **monitoramento do desempenho da aplicação** (**APM**) são usadas para mostrar sinais sobre o desempenho do seu aplicativo. Essas ferramentas fornecem insights sobre como otimizar melhor seu aplicativo. O GitHub tem recursos para monitorar fluxos de trabalho de CI/CD com o GitHub Actions, agregar descobertas de segurança com o GitHub Advanced Security e fornecer métricas de velocidade do desenvolvedor com insights organizacionais.

O uso de ferramentas geralmente é o aspecto mais visível de uma prática de DevOps. É uma manifestação prática da cultura e dos processos de DevOps que influenciam cada fase do SDLC. Em DevOps, as ferramentas geralmente são usadas para aplicar a automação sempre que possível, criar ciclos de feedback e liberar recursos organizacionais. Ao incentivar os ciclos de feedback por meio de monitoramento automatizado e ferramentas de relatório, o DevOps ajuda as equipes a criar softwares mais resilientes. Quando surgem problemas, a automação e as ferramentas de DevOps ajudam a implantar as correções para a produção com mais rapidez do que nas práticas de desenvolvimento de software tradicionais.

O DevOps não é implementado simplesmente comprando um conjunto de ferramentas, mas ferramentas abertas e colaborativas por natureza podem promover princípios de DevOps. As ferramentas de DevOps de última geração, juntamente com uma cultura de DevOps de alto funcionamento, podem fornecer às organizações uma enorme vantagem competitiva.

Conclusão: DevOps como uma estrutura para agregar valor

Se você pedir a 10 pessoas que definam DevOps, provavelmente receberá pelo menos cinco respostas diferentes.

Algumas pessoas podem se concentrar na implementação prática do DevOps (CI/CD, automação de testes e assim por diante) e chamam isso de processo. Outros podem chamar o DevOps de uma metodologia com um conjunto de processos que funcionam juntos sob uma filosofia coerente. Porém, essas duas definições ignoram o ponto principal: o DevOps consiste em um conjunto de práticas que são adaptáveis a cada empresa que as adota.

É melhor entender o DevOps como uma estrutura para pensar em como agregar valor por meio de software. Ele é mais do que uma única metodologia ou coleção de processos. É fundamentalmente um conjunto de práticas, tanto culturais quanto tecnológicas.

As ferramentas geralmente são o aspecto mais visível do DevOps, mas o DevOps não é uma única ferramenta. Uma transformação de DevOps começa com uma mudança cultural cuja finalidade é alterar a forma como pensamos sobre o conhecimento e a responsabilidade. As informações são a moeda de um pipeline de DevOps. A consideração principal é se essas informações podem fluir livremente entre as fases, o que é facilitado pela cultura e pelo processo tanto quanto pela tecnologia.

Assim que a base de uma cultura de DevOps de alto funcionamento estiver em vigor, as ferramentas poderão influenciar absolutamente o sucesso geral de uma prática de DevOps em uma organização. A melhor cultura de DevOps, com ciclos contínuos de feedback de aprendizagem, requer ferramentas capazes de permitir o fluxo coeso de uma fase de DevOps para a próxima. A combinação de pessoas, processos, práticas culturais e tecnologias que trabalham juntas é o indicador de uma prática de DevOps bem-sucedida. Somente quando cada pilar estiver funcionando em uníssono é possível alcançar os benefícios do DevOps de entrega mais rápida, maior qualidade e produtos mais escaláveis.

Recursos

- O que é o Modelo de DevOps? Explorando práticas fundamentais no DevOps
- Fundamentos do DevOps: definição dos princípios de DevOps
- Devemos pensar no DevOps como uma metodologia?
- O que é um pipeline de DevOps? Um guia completo
- Os fundamentos da integração contínua em DevOps
- Os fundamentos da implantação contínua no DevOps
- O que é conteinerização?
- Explicando DevSecOps
- Um guia sobre ferramentas e toolchain para automação de DevOps
- Ferramentas de monitoramento de DevOps: Automatizando seus processos de monitoramento de DevOps

Crie sua prática de DevOps no GitHub

O GitHub é uma plataforma integrada que leva as empresas da ideia ao planejamento e à produção, combinando uma experiência de desenvolvedor focada com uma infraestrutura poderosa e totalmente gerenciada de desenvolvimento, automação e teste.

O conjunto abrangente de ferramentas do GitHub reúne todo o pipeline de DevOps em um único pacote de ferramentas. A fim de auxiliar no planejamento, o GitHub Issues and Projects fornece uma abordagem inovadora e centrada no desenvolvedor para o gerenciamento do trabalho. Depois que a ideia for planejada, os desenvolvedores podem começar a trabalhar no código em um contêiner de desenvolvimento isolado idêntico ao de seus colegas de trabalho com o GitHub Codespaces. Quando o recurso está pronto para ser revisado, as pull requests no GitHub permitem que os desenvolvedores colaborem e recebam feedback em tempo real. O GitHub Actions é a plataforma de automação usada para CI, CD e automatização de tudo que é possível. O GitHub Packages é usado para armazenar, gerenciar e distribuir pacotes de software. Para manter seu código seguro e os segredos fora do controle de origem sem interromper o fluxo dos desenvolvedores, aproveite o pacote de ferramentas do GitHub Advanced Security. Ao usar o GitHub e seus recursos, cada fase do pipeline de DevOps pode ser aprimorada.

Como a maior e mais avançada plataforma de desenvolvimento do mundo, o GitHub ajuda milhões de desenvolvedores e empresas a colaborar, criar e entregar com mais rapidez. E, com milhares de integrações de DevOps, você pode criar com suas ferramentas já conhecidas de longa data ou descobrir novas. Obtenha a plataforma completa para desenvolvedores hoje mesmo e junte-se a mais de 83 milhões de desenvolvedores e 4 milhões organizações que desenvolvem software no GitHub.

Tem dúvidas sobre o GitHub Enterprise?

Nós podemos ajudar.

Visite nossa página do GitHub Enterprise
ou conecte-se com nossa equipe de vendas.