Sumario

DevOps - Prática em Desenvolvimento e Operações

A criação de um software é um processo dinâmico que envolve diversas etapas interligadas. Desde a concepção da ideia até a entrega final ao usuário, cada fase desempenha um papel crucial no sucesso do projeto.

Inicialmente, ocorre a compreensão profunda das necessidades do cliente. É nesse momento que se define o escopo do projeto, ou seja, quais funcionalidades o software deverá ter. Essa etapa é fundamental para garantir que o produto final atenda às expectativas do usuário.

Com o escopo definido, a equipe de desenvolvimento inicia a fase de planejamento. Aqui, são traçadas as estratégias para a construção do software, como a escolha da tecnologia a ser utilizada, a definição da arquitetura do sistema e a divisão das tarefas entre os membros da equipe.

Em seguida, vem a implementação, que consiste na escrita do código-fonte. É nessa etapa que a ideia se transforma em realidade, dando vida às funcionalidades do software. Os desenvolvedores utilizam linguagens de programação específicas para criar os componentes que compõem o sistema.

Concomitantemente à implementação, ocorre a fase de teste. Os testes são essenciais para garantir que o software funcione corretamente e atenda aos requisitos definidos no início do projeto. Existem diversos tipos de testes, como testes unitários, de integração e de sistema, cada um com um objetivo específico.

Após a fase de testes, o software está pronto para ser implantado. Essa etapa consiste em colocar o software em produção, ou seja, torná-lo disponível para os usuários. A implantação pode ser feita de forma gradual ou completa, dependendo da complexidade do sistema e da estratégia adotada pela equipe.

A jornada de um software não termina com a implantação. A fase de manutenção é crucial para garantir que o sistema continue funcionando corretamente ao longo do tempo. Nessa etapa são realizadas correções de bugs, implementações de novas funcionalidades e adaptações para atender às novas necessidades dos usuários.

É importante ressaltar que essas etapas não são lineares, mas sim iterativas. Ou seja, é comum retornar a etapas anteriores para realizar ajustes ou adicionar novas funcionalidades. Além disso, a metodologia utilizada para o desenvolvimento do software pode influenciar como essas etapas são organizadas e executadas.

Software de Prateleira vs. SaaS: Uma Análise Detalhada

Ao escolher uma solução de software para sua empresa, você se depara com duas opções principais: o software de prateleira e o SaaS (Software as a Service). Cada uma dessas opções possui suas particularidades, vantagens e desvantagens, e a escolha ideal dependerá das necessidades e características específicas do seu negócio.

Software de Prateleira é como um produto pronto para o consumo, desenvolvido para atender a um público amplo. É adquirido por meio de licenças e instalado localmente na sua empresa. A grande vantagem desse modelo é a possibilidade de ter um controle total sobre o software, podendo personalizá-lo conforme as suas necessidades. No entanto, a instalação e manutenção são de responsabilidade da sua equipe, o que pode gerar custos adicionais e demandar conhecimento técnico. Além disso, as atualizações do software geralmente são menos frequentes e podem exigir um processo de instalação mais complexo.

SaaS, por sua vez, é um modelo de software hospedado na nuvem e acessado através da internet. Você paga uma assinatura mensal ou anual para utilizar o software, e o fornecedor se encarrega de toda a infraestrutura e

manutenção. A principal vantagem do SaaS é a sua flexibilidade, pois você pode acessar o software de qualquer lugar com uma conexão à internet. Além disso, as atualizações são automáticas e frequentes, garantindo que você sempre tenha acesso à versão mais recente do software. No entanto, a dependência de um fornecedor externo pode ser um ponto negativo para algumas empresas, especialmente em relação à segurança dos dados.

O que é DevOps?

DevOps é um termo que vem da junção das abreviações das palavras Development e Operations fazendo referência a Desenvolvimento e Operações para criar equipes multidisciplinares que utilizam práticas e ferramentas compartilhadas e eficientes.

DevOps é a união de pessoas, processos e produtos para habilitar a entrega contínua do valor para os usuários finais. As práticas essenciais de DevOps incluem planejamento ágil, integração contínua, entrega contínua e monitoramento de aplicativos.

DevOps é a ponte entre desenvolvimento e operações, cultura - pratica - ferramentas.

What is DevOps? - In Simple English

DevOps é uma abordagem cultural e técnica que visa unir as equipes de desenvolvimento de software (Dev) e operações (Ops), com o objetivo de entregar software de forma mais rápida, confiável e frequente. Essa união promove uma colaboração mais estreita entre as equipes, automatizando processos e quebrando silos, resultando em um ciclo de vida de desenvolvimento de software mais eficiente.

Pontos Principais

- Produzir um software envolve diversas disciplinas (análise, arquitetura, programação, testes, distribuição, implantação e operação).
- A produção de software tem de lidar com os desafios relacionados a trabalhar em equipe e dentro de limites de custo e prazo.
- DevOps expressa uma forma de pensar orientada a fazer uma entrega contínua do produto de software.

Gerenciamento de Código Fonte

O gerenciamento de código fonte, também conhecido como controle de versão, é uma prática fundamental no desenvolvimento de software. Ele consiste em um conjunto de ferramentas e técnicas utilizadas para rastrear e controlar as alterações feitas no código ao longo do tempo. Essa prática é essencial para garantir a colaboração eficiente entre desenvolvedores, facilitar a recuperação de versões anteriores do código e permitir um desenvolvimento mais organizado e seguro.

Como funciona o gerenciamento de código fonte?

Repositório: Um repositório é um local centralizado onde o código fonte é armazenado. Ele funciona como um banco de dados que armazena todas as versões do código.

- Commits: Um commit é uma salvaguarda das alterações feitas no código. Cada commit inclui uma mensagem descrevendo as mudanças realizadas.

- Branches: Branches são ramificações do código principal, permitindo que os desenvolvedores trabalhem em novas funcionalidades de forma isolada, sem afetar o código principal.
- Merges: O processo de combinar as alterações de uma branch com o código principal é chamado de merge.

Ferramentas populares para gerenciamento de código fonte

- Git: O Git é um sistema de controle de versão distribuído, amplamente utilizado e considerado o padrão da indústria. Ele oferece grande flexibilidade e permite trabalhar de forma offline.
- SVN (Subversion): O SVN é um sistema de controle de versão centralizado, mais simples de usar que o Git, mas menos flexível.
- Mercurial: O Mercurial é outro sistema de controle de versão distribuído, similar ao Git, mas com algumas diferenças em sua arquitetura.

Benefícios do uso de ferramentas de gerenciamento de código fonte

- Melhora a qualidade do código: Ao facilitar a revisão por pares e o histórico de alterações, o gerenciamento de código fonte contribui para a produção de código mais limpo e com menos erros.
- Aumenta a produtividade: Ao automatizar tarefas repetitivas e facilitar a colaboração, as ferramentas de gerenciamento de código fonte aumentam a produtividade da equipe de desenvolvimento.
- Reduz riscos: Ao manter um histórico completo do código, é possível restaurar versões anteriores em caso de problemas, reduzindo o risco de perda de dados.

O que é o Git?

O Git é um sistema de controle de versão distribuído amplamente utilizado por desenvolvedores de software para gerenciar e acompanhar mudanças no código-fonte de projetos. Criado por Linus Torvalds, o mesmo criador do Linux, o Git foi projetado para ser rápido, eficiente e suportar fluxos de trabalho não lineares e colaborativos.

Principais características do Git

- Distribuído: Cada desenvolvedor tem uma cópia completa do repositório, incluindo todo o histórico de mudanças. Isso permite trabalhar offline e faz do Git uma ferramenta resiliente contra falhas.
- Rastreamento de mudanças: O Git mantém um histórico completo de todas as alterações feitas em um projeto. Isso inclui adições, modificações e exclusões de arquivos, assim como informações sobre quem fez as mudanças e quando foram feitas.
- Branches e merges: O Git facilita a criação de branches (ramificações) para que desenvolvedores possam trabalhar em novas funcionalidades ou correções de bugs sem afetar o código principal. Depois, é possível mesclar (merge) as mudanças de volta ao branch principal.
- Velocidade e eficiência: O Git é otimizado para operações rápidas, como commit, diff, branch e merge, tornando-o uma escolha popular para projetos de todos os tamanhos.
- Colaboração: O Git é amplamente utilizado em plataformas de hospedagem de código como GitHub, GitLab e Bitbucket, que oferecem ferramentas adicionais para revisão de código, integração contínua e gerenciamento de projetos.

Quais são os benefícios do Git?

- Controle de versão preciso: O Git rastreia cada alteração feita no código, permitindo que você volte para qualquer versão anterior com facilidade. Isso é essencial para identificar a causa de bugs, recuperar arquivos perdidos e acompanhar a evolução do projeto.
- Colaboração eficiente: O Git facilita o trabalho em equipe, permitindo que vários desenvolvedores trabalhem simultaneamente em um mesmo projeto sem conflitos. Cada desenvolvedor possui uma cópia local do repositório, o que torna a colaboração mais ágil e flexível.
- Ramificação simplificada: Com o Git, você pode criar branches (ramificações) do código principal para trabalhar em novas funcionalidades de forma isolada. Isso permite experimentar novas ideias sem afetar o código principal e facilita a revisão de código por outros desenvolvedores.
- Distribuído: Ao contrário de outros sistemas de controle de versão centralizados, o Git é distribuído. Isso significa que cada desenvolvedor possui uma cópia completa do repositório, o que torna o sistema mais robusto e resistente a falhas.
- Flexibilidade: O Git oferece uma grande variedade de comandos e ferramentas para personalizar o fluxo de trabalho. Você pode adaptar o Git às suas necessidades específicas e ás do seu projeto.
- Comunidade ativa: O Git possui uma comunidade de usuários muito grande e ativa, o que significa que você encontrará facilmente ajuda e recursos online.
- Integração com outras ferramentas: O Git se integra facilmente com outras ferramentas de desenvolvimento, como editores de código, plataformas de CI/CD e ferramentas de gerenciamento de projetos.

O que é um Branch?

Um branch (ou ramificação) no Git é uma linha independente de desenvolvimento dentro de um repositório. Ele permite que você trabalhe em novas funcionalidades, correções de bugs ou experimentos sem afetar o código principal do projeto.

Um branch no Git pode ser imaginado como um ramo que se separa do tronco principal de um projeto. Em termos mais técnicos, é uma linha de desenvolvimento independente do código-fonte.

Criação e Uso

- Criar um Branch: Você pode criar um branch com o comando ***git branch nome-do-branch.***
- Trocar para um Branch: Para trabalhar em um branch específico, você pode usar ***git checkout nome-do-branch ou git switch nome-do-branch.***
- Mesclar Branches: Uma vez que o trabalho no branch está concluído, ele pode ser mesclado de volta ao branch principal usando ***git merge nome-do-branch.***

O que é um Commit?

Um commit no Git é um registro de alterações feitas no código. Ele captura o estado atual do projeto em um ponto específico no tempo. Cada commit tem um identificador único (um hash SHA-1) e geralmente inclui uma mensagem descrevendo as mudanças feitas, quem as fez e quando.

Como realizar um commit

```
cd caminho/do/repositorio        # Navega até o diretório do repositório
git status                       # Verifica o status dos arquivos
git add .                        # Adiciona todas as mudanças ao staging
git commit -m "Descrição clara das alterações"  # Cria o commit com uma mensagem
```

git log # Verifica o histórico de commits

Certifique-se de estar no diretório correto:

- Use cd caminho/do/repositorio para navegar até o diretório do seu projeto, onde o repositório Git está inicializado.

Verifique o status do repositório:

- Execute git status para ver o status atual do seu repositório. Isso mostrará quais arquivos foram modificados, adicionados ou excluídos, e se há mudanças que precisam ser registradas.

Antes de fazer um commit, você precisa adicionar as mudanças ao staging area (área de preparação), o que significa que você está selecionando quais mudanças serão incluídas no próximo commit.

Adicionar arquivos específicos:

- Use git add nome-do-arquivo para adicionar um arquivo específico.

Adicionar todos os arquivos modificados:

- Use git add . para adicionar todos os arquivos modificados e novos ao staging.

Uma vez que as mudanças estejam na área de preparação, você pode criar o commit.

Criar um commit com uma mensagem descritiva:

- Use git commit -m "Mensagem do commit" para fazer o commit. A mensagem deve descrever de forma clara o que foi alterado, como "Corrige bug na função de login" ou "Adiciona nova funcionalidade de busca".

Após fazer o commit, você pode verificar se ele foi criado corretamente:

Verifique o log dos commits:

- Execute git log para ver uma lista de todos os commits no repositório. O commit mais recente deve aparecer no topo.

Diferença entre Commit e Branch

- Commit: Representa uma mudança específica no código. Pense nele como uma "foto" do projeto em um determinado momento. Vários commits juntos formam o histórico de mudanças de um branch.
- Branch: É uma linha de desenvolvimento que pode conter múltiplos commits. Ele aponta para o commit mais recente de uma sequência de commits. Quando você faz um novo commit em um branch, esse branch é atualizado para apontar para o novo commit.

Commit: É um ponto específico no histórico do seu projeto.
Branch: É uma linha de desenvolvimento que contém uma série de commits.

Playlist - Git na Prática:

https://youtube.com/playlist?list=PLSbD5F_Z_s7b5TJF80zb5dQojao9UQLxL&si=-uwUCpe4M_enOOE8

Pontos Principais

- Descobrir o que é o Git, que é um sistema de controle de versão de código aberto (VCS) distribuído que permite armazenar código, rastrear histórico de revisão, mesclar alterações de código e reverter para versões de código anteriores;
- Perceber que o Git tem vários benefícios importantes, como: Controle de histórico de alterações, trabalho em equipe, melhoria da velocidade e da produtividade da equipe, disponibilidade e Redundância e a popularidade do Git; e
- Entender como funciona um branch no Git que é um leve ponteiro móvel para um desses commits;

Gerenciamento de Código Fonte Distribuído

O gerenciamento de código fonte distribuído é uma abordagem para controlar e acompanhar as mudanças em um projeto de software, onde cada desenvolvedor possui uma cópia completa do repositório. Isso significa que cada equipe tem a liberdade de trabalhar de forma independente, sem depender de um servidor central.

Git

O Git é o sistema de controle de versão distribuído mais popular e amplamente utilizado hoje em dia. Ele permite que os desenvolvedores:

- Trabalhem offline: Como cada desenvolvedor possui uma cópia completa do repositório, eles podem trabalhar mesmo sem conexão com a internet.
- Crie branches facilmente: Branches são como ramificações do código principal, permitindo que os desenvolvedores trabalhem em novas funcionalidades de forma isolada.
- Revertam mudanças: Se algo der errado, é fácil voltar para uma versão anterior do código.
- Colaborem de forma eficiente: O Git facilita a colaboração entre equipes, permitindo que os desenvolvedores mesclem suas alterações e resolvam conflitos.

Conceitos-chave no Git

- Repositório: Um diretório que contém todos os arquivos do projeto, incluindo o histórico de versões.
- Commit: Um instantâneo do projeto em um determinado momento. Cada commit registra as alterações feitas desde o último commit.
- Branch: Uma linha de desenvolvimento independente que se ramifica do código principal.
- Merge: A ação de combinar duas ou mais branches em uma única.
- Pull Request: Uma solicitação para que as alterações de um branch sejam mescladas em outro branch, geralmente o branch principal.

Vantagens do Git

- Desempenho: O Git é extremamente rápido, especialmente para operações locais.
- Flexibilidade: O Git oferece uma grande variedade de ferramentas e workflows para atender às necessidades de diferentes projetos.

- Comunidade: O Git possui uma comunidade grande e ativa, o que significa que há muita documentação, tutoriais e suporte disponíveis.

Como o Git funciona

- Clonagem: O desenvolvedor cria uma cópia local do repositório remoto.
- Modificação: O desenvolvedor faz alterações nos arquivos.
- Commit: As alterações são registradas em um novo commit.
- Push: O desenvolvedor envia suas alterações para o repositório remoto.

Commit

Um commit no Git é como um instantâneo do seu projeto em um determinado momento. É como tirar uma foto de todos os seus arquivos e salvar essa imagem em um histórico. Cada commit registra as alterações feitas desde o último commit, criando uma linha do tempo do desenvolvimento do seu projeto.

Por que os commits são importantes

- Histórico: Os commits permitem que você acompanhe a evolução do seu projeto ao longo do tempo. Você pode ver quais alterações foram feitas, quem as fez e quando.
- Versões: Cada commit representa uma versão específica do seu projeto. Isso significa que você pode voltar a uma versão anterior se precisar.
- Colaboração: Os commits facilitam a colaboração em equipe, pois cada desenvolvedor pode fazer suas próprias alterações e depois mesclá-las com o trabalho dos outros.
- Backup: Os commits servem como um backup seguro do seu projeto. Mesmo que você perca arquivos localmente, você pode recuperá-los a partir dos commits.

Como fazer um commit

- Adicionar as alterações: Use o comando git add <arquivo> para adicionar os arquivos modificados à área de staging.
- Criar o commit: Use o comando git commit -m "Mensagem do commit" para criar um novo commit. A mensagem do commit deve descrever as alterações que você fez.

Exemplo:

```
git add meu_arquivo.py
git commit -m "Adicionando nova função de cálculo"
```

Branch

Um branch (ou ramificação) no Git é uma linha independente de desenvolvimento dentro de um repositório. Ele permite que você trabalhe em novas funcionalidades, correções de bugs ou experimentos de forma isolada, sem afetar o código principal do projeto. Isso é extremamente útil em ambientes de desenvolvimento colaborativo, onde diferentes membros da equipe podem trabalhar simultaneamente em diferentes partes do projeto.

Conceitos-Chave Sobre Branches

Isolamento

- Cada branch é um espaço separado para desenvolvimento. As alterações feitas em um branch específico não afetam outros branches até que sejam mescladas (merged).
- Isso permite que desenvolvedores experimentem e desenvolvam novas funcionalidades sem interferir no código que está em produção ou em outros trabalhos em andamento.

Branch Principal (main ou master)

- O branch principal de um repositório, frequentemente chamado de main (ou master em versões mais antigas), contém o código estável que está em produção ou pronto para ser lançado.
- Em muitos fluxos de trabalho, o código que passa pelos testes e revisão é eventualmente integrado ao main.

Branches de Funcionalidade

- Um branch pode ser criado para trabalhar em uma nova funcionalidade ou corrigir um bug. Esses branches geralmente são temporários e, uma vez que o trabalho é concluído e testado, as mudanças são mescladas de volta ao branch principal.
- Nomes comuns para esses branches incluem feature/nova-funcionalidade, bugfix/corrigir-erro, etc.

Criação e Alternância entre Branches

- Você pode criar um branch com o comando git branch nome-do-branch.
- Para começar a trabalhar em um branch específico, use o comando git checkout nome-do-branch ou git switch nome-do-branch (a partir das versões mais recentes do Git).

Mesclagem (Merge)

- Uma vez que o trabalho em um branch está completo, as mudanças feitas nesse branch podem ser mescladas de volta ao branch principal ou a outro branch.
- O comando git merge nome-do-branch é usado para integrar as mudanças de um branch a outro.
- Durante a mesclagem, pode haver conflitos se as mesmas partes do código foram alteradas em diferentes branches. O Git oferece ferramentas para resolver esses conflitos manualmente.

Branching Strategy:

- Git Flow: Um fluxo de trabalho popular que utiliza branches de desenvolvimento (develop), branches de funcionalidades (feature), branches de release (release), e branches de hotfixes (hotfix).
- GitHub Flow: Uma abordagem mais simples que geralmente envolve o branch main e branches de funcionalidades que são mesclados por meio de pull requests.
- Trunk-Based Development: Envolve trabalhar diretamente em um branch principal, com pequenas ramificações de curta duração para commits frequentes.

Vantagens de Usar Branches

- Desenvolvimento Paralelo: Permite que múltiplas funcionalidades ou correções sejam desenvolvidas em paralelo sem interferências.
- Histórico Organizado: Ajuda a manter o histórico de commits mais organizado e fácil de seguir.

- Colaboração Facilitada: Desenvolvedores podem colaborar de forma mais eficaz, sem se preocupar com interferências em suas mudanças.

Merge

Em termos simples, um merge no Git é a ação de combinar as alterações de um branch em outro. É como juntar dois galhos de uma árvore em um só. Essa ação é fundamental para que diferentes desenvolvedores possam trabalhar em partes diferentes do código e, posteriormente, unificar suas contribuições.

Tipos de Merge

- Merge Automático (Fast-Forward): Ocorre quando o branch de destino não tem novos commits desde a criação do branch que está sendo mesclado. Nesse caso, o Git simplesmente "avança" o ponteiro do branch de destino para o commit mais recente do branch a ser mesclado, sem criar um novo commit de mesclagem. Exemplo: Se o branch main não recebeu novos commits enquanto você estava trabalhando no branch feature, o Git pode avançar o main para o mesmo commit final de feature sem criar um novo commit.
- Merge de Três Vias (Three-Way Merge): Esse tipo de merge ocorre quando há commits novos tanto no branch de origem quanto no branch de destino. O Git usa o último commit comum entre os dois branches como ponto de referência e cria um novo commit que combina as mudanças. Exemplo: Se o branch main recebeu commits enquanto você trabalhava no branch feature, o Git cria um novo commit de mesclagem que integra as mudanças de ambos os branches.

Por que fazer um merge?

- Integrar mudanças: Quando um desenvolvedor termina de trabalhar em uma nova funcionalidade em um branch específico, ele pode fazer um merge para integrar essas mudanças no branch principal ou em outro branch relacionado.
- Resolver conflitos: Às vezes, quando dois desenvolvedores fazem alterações nos mesmos arquivos, o Git não consegue automaticamente determinar qual versão deve ser mantida. Nesses casos, ocorrem conflitos de merge que precisam ser resolvidos manualmente.
- Atualizar branches: É comum que os branches precisem ser atualizados com as últimas alterações do branch principal para evitar divergências significativas.

Como fazer um merge?

O comando básico para realizar um merge é git merge <nome_do_branch>.

- Selecione o branch de destino: Use o comando git checkout <nome_do_branch> para mudar para o branch onde você deseja fazer o merge.
- Execute o merge: Use o comando git merge <nome_do_branch_fonte> para mesclar o branch de origem no branch de destino.

Exemplo:

```
# Mudar para o branch principal
git checkout main
```

Mesclar o branch 'feature' no branch principal
git merge feature

Conflitos de merge

Às vezes, o Git não consegue automaticamente integrar as mudanças porque os mesmos arquivos foram modificados de maneiras conflitantes em ambos os branches. Isso resulta em um conflito de merge.
Quando um conflito ocorre, o Git interrompe o processo de merge e marca os arquivos em conflito, permitindo que você resolva manualmente.

Abra os arquivos em conflito e veja as marcações que o Git adicionou (<<<<<<<, =======, >>>>>>>). Essas marcações indicam onde o conflito está e quais são as mudanças conflitantes.

Resolvendo conflitos

- Edite os arquivos: Abra os arquivos com conflito em um editor de texto e remova os marcadores de conflito.
- Escolha as alterações: Decida quais alterações você deseja manter e remova as demais.
- Adicione as alterações: Use o comando git add para adicionar as alterações resolvidas.
- Commit: Faça um novo commit para registrar a resolução do conflito.

Dicas para evitar conflitos

- Faça merges frequentes: Merges menores são mais fáceis de gerenciar do que merges grandes.
- Rebase com cautela: O rebase pode ser útil para limpar o histórico, mas use-o com cuidado, pois pode criar problemas se o branch que você está rebasando já foi compartilhado.
- Use ferramentas visuais: Existem diversas ferramentas visuais que facilitam a visualização e resolução de conflitos.

Fundamentos de Git

Apostila em pt-BR sobre fundamentos e conceitos do Git.

Capitulo 2 - Fundamentos de Git
https://git-scm.com/book/pt-br/v2/Fundamentos-de-Git-Obtendo-um-Reposit%C3%B3rio-Git

Capitulo 3 - Branches no Git
https://git-scm.com/book/pt-br/v2/Branches-no-Git-Branches-em-poucas-palavras