Modelos, Métodos e Técnicas de Engenharia de Software

Prof.: Sônia A. Santana

una

Introdução

- Seja o seguinte cenário:
  - Usamos um processo para implementar um sistema
  - Os seu requisitos foram definidos e implementados
  - O projeto e arquitetura estão consolidados
  - Diversos testes foram implementados
  - E diversas refatorações já foram realizadas

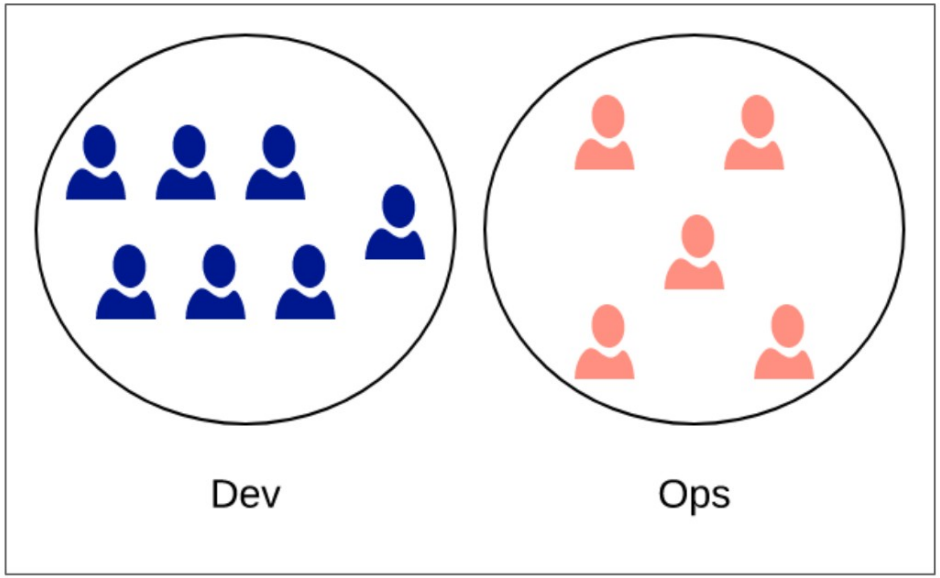
Introdução

Falta agora a última fase:

implantar o sistema (colocá-lo em produção)

Introdução

Antigamente, implantação era sempre traumática



Dois silos independentes, com pouquíssima comunicação

Ops = administradores de sistema, suporte, sysadmin, pessoal de IT, etc

Problemas

- Urgência em implantar um novo sistema (pelos ops)
- Desconhecimento das especificidades da plataforma utilizada no desenvolvimento (pelos ops) ou
- Desconhecimento das especificidades da plataforma utilizada na produção (pelos devps)

Resultado

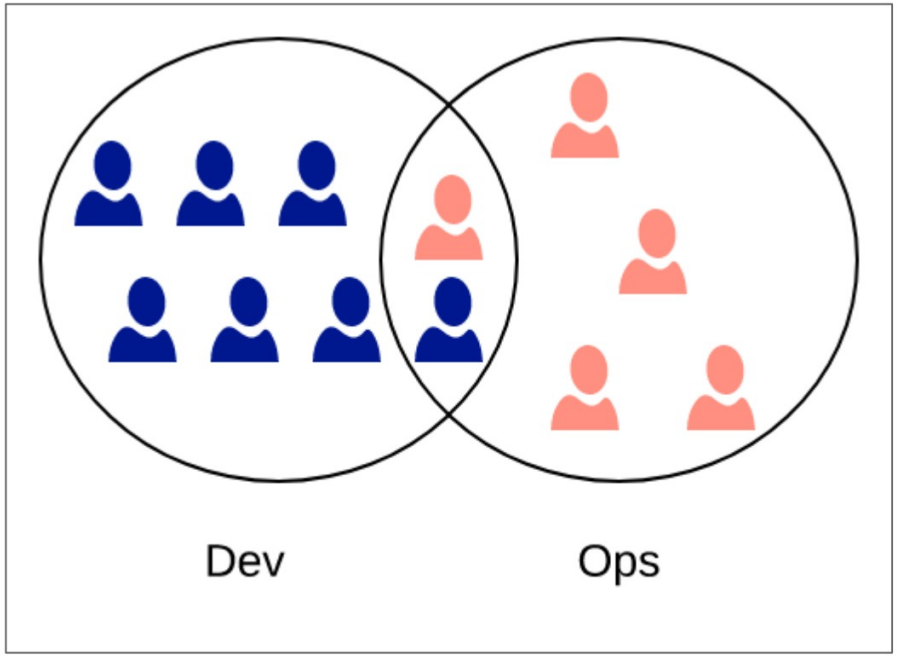
- Atraso ou cancelamento da implantação ou abandono do sistema, em alguns casos.

Consequências

- Falta de hardware para executar o novo sistema ou a nova funcionalidade
- Problemas de desempenho
- Incompatibilidades com o banco de dados de produção
- Vulnerabilidades de segurança

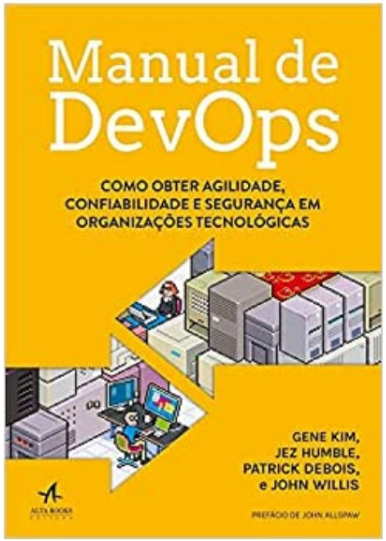
Introdução

Ideia central de DevOps: aproximar Dev e Ops



Frequentemente, Devs e Ops sentam juntos para discutir questões sobre a entrega do sistema

Introdução



Imagine um mundo no qual POs, devs, QA, "pessoal da TI" e de segurança trabalhem juntos ...

Introdução

Objetivo: passagem de bastão bem sucedida!

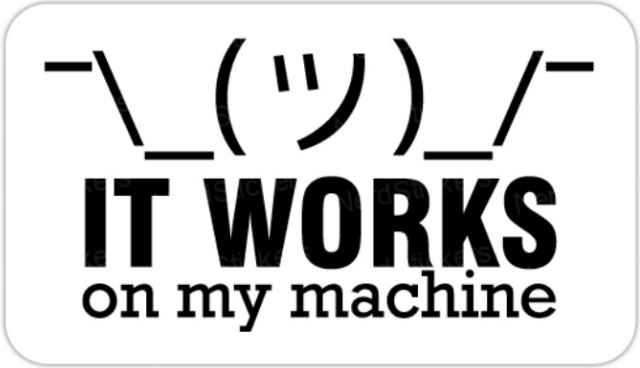
(isto é, deve começar o quanto antes; ser automatizada, etc)

Ops

Dev

Introdução

Objetivo: acabar com uma das frases mais famosas em Engenharia de Software



Introdução

Objetivo: acabar com o jogo-de-empurra

Dev: o problema não é meu código, mas seu servidor

Ops: o problema não é meu servidor, mas o seu código

DevOps

- Não é um cargo; mas um conjunto de princípios e práticas
- Ou um movimento, ou cultura
- Termo surgiu ~2009

DevOps

- "O DevOps é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.
- O DevOps permite que funções anteriormente isoladas – desenvolvimento, operações de TI, engenharia da qualidade e segurança – atuem de forma coordenada e colaborativa para gerar produtos melhores e mais confiáveis. Ao adotar uma cultura de DevOps em conjunto com as práticas e ferramentas de DevOps, as equipes ganham a capacidade de responder melhor às necessidades dos clientes, aumentar a confiança nos aplicativos que constroem e cumprir as metas empresariais mais rapidamente."

(https://azure.microsoft.com/pt-br/overview/what-is-devops/)

Princípios de DevOps

Baseado nos princípios para entrega de software do livro: "*Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, de Humble e Farley, 2010.

1. Crie um processo repetível e confiável para entrega de software: colocar um software em produção deve ser tão simples como apertar um botão.
2. Automatize tudo que for possível (pré-requisito do princípio anterior):
   a. todos os passos para entrega de um software devem ser automáticos, incluindo seu build, a execução dos testes, a configuração e ativação dos servidores e da rede, a carga do banco de dados, etc.
3. Mantenha tudo em um sistema de controle de versões:
   a. não apenas a todo o código fonte, mas também arquivos e scripts de administração do sistema, documentação, páginas Web, arquivos de dados, etc.
   b. deve ser simples restaurar e voltar o sistema para um estado anterior.

Princípios de DevOps

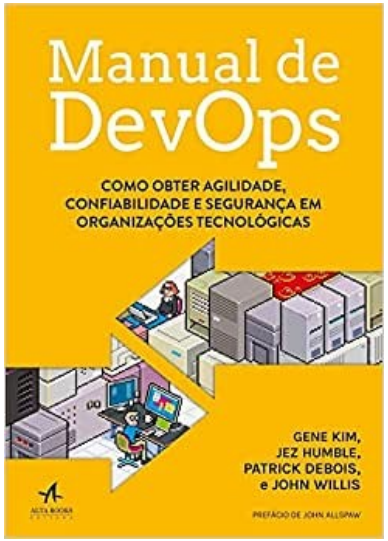
Baseado nos princípios para entrega de software do livro: "*Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, de Humble e Farley, 2010.

4. Se um passo causa dor, execute-o com mais frequência e o quanto antes:
    a. antecipar os problemas antes que eles se acumulem e as soluções fiquem complicadas (**Integração Contínua**)
5. Concluído significa pronto para entrega:
    a. Não pode haver pendências como: a implementação ainda não foi testada com dados reais, ela ainda não foi documentada, ela ainda não foi integrada com o sistema X, etc.
6. Todos são responsáveis pela entrega do software:
    a. não admite-se mais que os times de desenvolvimento e de operações trabalhem em silos independentes e troquem informações apenas na véspera de uma implantação.

Princípios de DevOps

- Resumindo....
  - Aproximar Devs e Ops, desde o início do projeto
  - Adotar princípios ágeis na fase de implantação
  - Transformar implantações em um não-evento
  - Implantar sistemas (partes dele) todos os dias
  - Automatização do processo de implantação

Princípios de DevOps



- Em vez de iniciar as implantações à meia-noite de sexta-feira e passar o fim de semana trabalhando para concluí-las, as implantações ocorrem em qualquer dia útil, quando todos estão na empresa e sem que os clientes percebam ...

Pilares do DevOps

Fonte: https://blog.estabil.is/calms-conceito-devops/

Pilares do DevOps

- Cultura:
  - Para além de ferramentas, o que deve existir na organização é uma cultura que englobe a definição, produção, versionamento, entrega e que mantêm funcionando seu produto/software/aplicação.
  - É preciso oferecer um ambiente no qual se possa "errar sem julgamento" (oferecer segurança).

Pilares do DevOps

- Automação:
  - Aqui as ferramenta podem falar mais alto. A ideia é automatizar ao máximo todos os fluxos e processos (principalmente os repetitivos), com foco em reduzir erros/falhas humanas.
  - Surge jargão “pipeline”, que é um fluxo programado de ações e entregas onde uma vez iniciado ele termina com um artefato/entrega pronto. Pipelines podem ser 100% automatizados ou requererem ações humanas em alguns pontos.

Pilares do DevOps

- LEAN IT:
  - A parte *zen* do DevOps.
  - Aqui o LEAN IT trará conceitos e práticas onde a cultura DevOps irá se apoiar e prosperar. Ele possui 5 princípios:
    - Especificar/Definir Valor
    - Identificar e mapear o fluxo
    - Criar um fluxo contínuo
    - Produção puxada (puxada do cliente)
    - Buscar a perfeição

Pilares do DevOps

- LEAN Software Development:
- 7 princípios:
  - Eliminar o desperdício/Lixo
  - Trabalho parcial / features extras / reaprendizado / troca de tarefas / esperas / defeitos / gerenciamento
  - Ampliar o aprendizado / conhecimento
  - Decidir o mais tarde possível
  - Entregue o mais rápido possível
  - Empoderar / fortalecer o time
  - Otimizar o todo
  - Buscar a perfeição / melhoria contínua

Pilares do DevOps

- Medição:
- É importante para a execução do DevOps poder medir e registrar toda a execução, seja para:
  - Medir o fluxo
  - Monitorar o fluxo
  - Corrigir/ajustar problemas e desvios o mais rápido possível
  - Implementar melhorias onde possível

Pilares do DevOps

- MEDIÇÕES:
  - Quanto tempo levou para uma feature ser entregue, do desenvolvimento à produção?
  - Qual a frequência de erros e falhas?
  - Qual o tempo de recuperação após uma falha?
  - Qual o tempo de correção de um erro?
  - Qual a carga do sistema no momento?
  - Qual os picos de acessos, carga ou latência do sistema?
  - O que realmente é usado no sistema?

Pilares do DevOps

- Compartilhamento / Share:
  - É onde o DevOps avança e se estabelece.
  - O conceito é descentralizar o conhecimento com o objetivo de reduzir as dependências e gargalos, gerando autossuficiência na equipe, reduzindo ao máximo a existência de peças insubstituíveis.
  - Se compartilha:
    - Conhecimentos e experiências
    - Sucessos e fracassos

Práticas de DevOps

- Controle de Versões
- Integração Contínua (CI)
- Deployment Contínuo (CD)

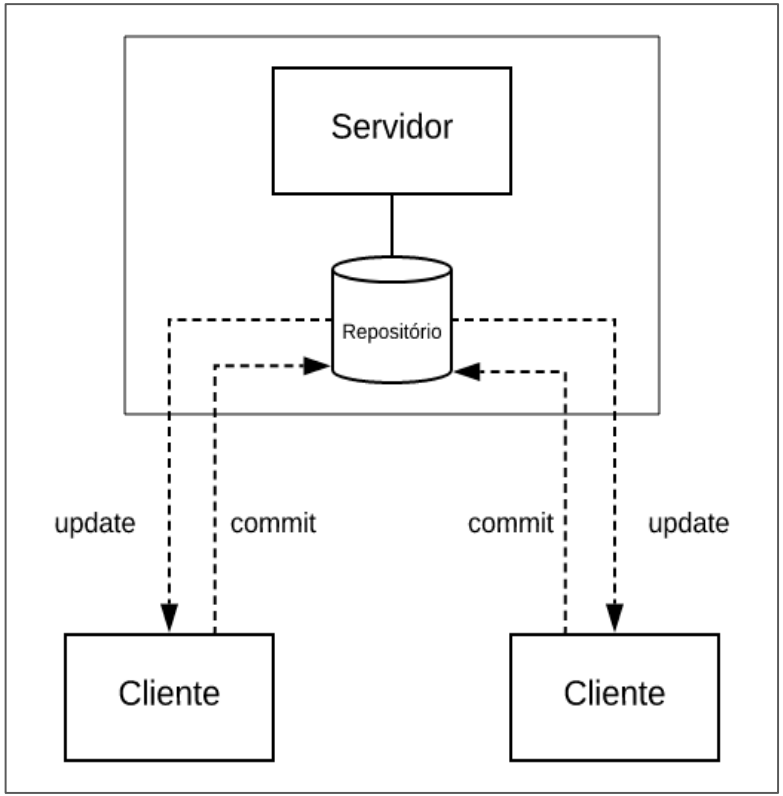
CONTROLE DE VERSÕES

Controle de Versões

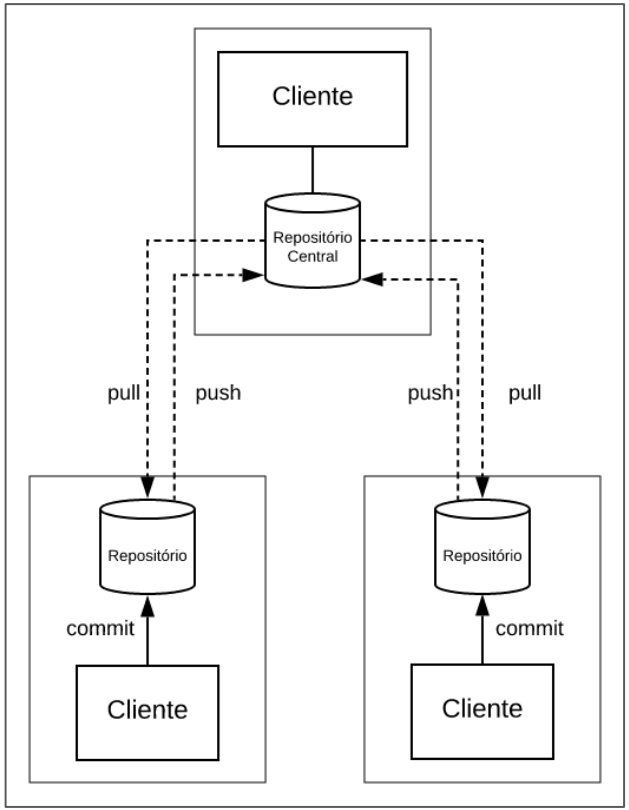
- Fundamental para desenvolvimento colaborativo
- Armazena "última" versão do sistema (fonte da verdade)
- Permite recuperar versões anteriores

Tipos de Controle de Versões

CENTRALIZADO (CVN, CVS)



DISTRIBUÍDO (GIT)

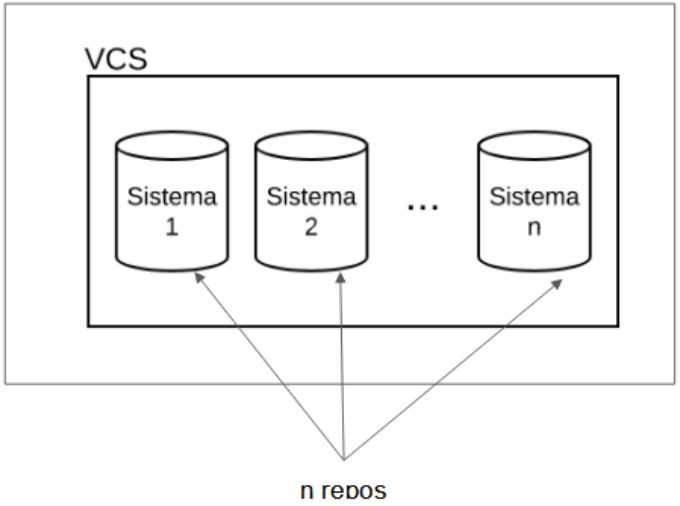


Vantagens do Controle de Versões Distribuídos

- Pode-se fazer commits com mais frequência
- Commits são mais rápidos
- Pode-se trabalhar off-line
- Arquiteturas alternativas: P2P, hierárquica, etc

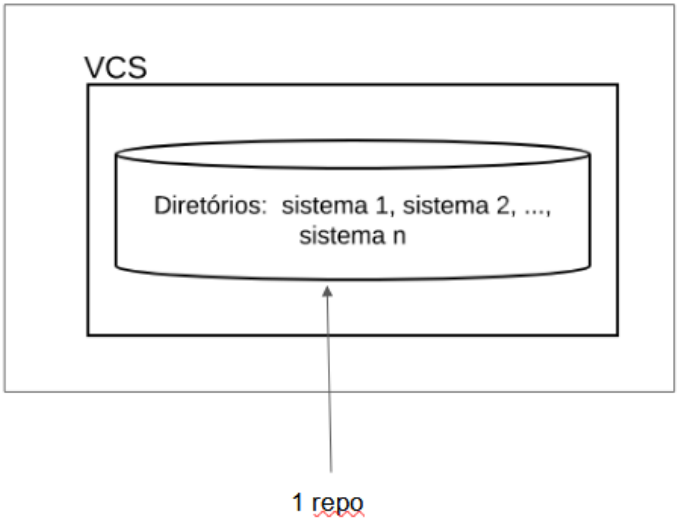
Multirepos x Monorepos

MULTIREPOS



+ comum;
Gerencia um repositório por projeto ou sistema.

MONOREPOS



Gerencia apenas um único repositório;
Projetos são diretórios desse repositório.
Usado por grandes empresas, p. ex.: Google.

Vantagens do Monorepos

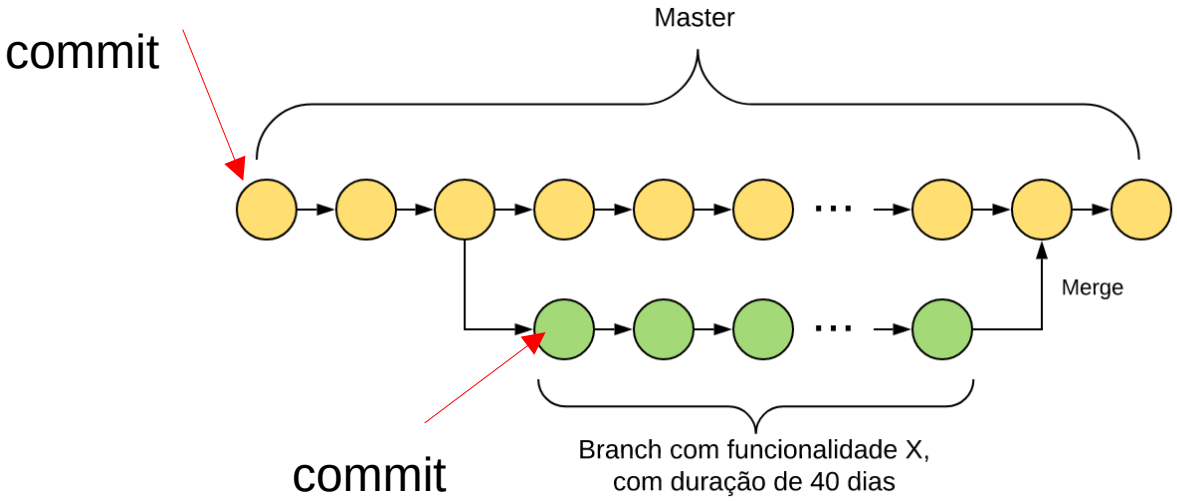
- Uma única fonte de "verdade";
- Incentivam reúso de código;
- Mudanças são atômicas (1 commit pode alterar n sistemas);
- Facilitam refactorings em larga escala;

Integração Contínua (CI)

- Motivação : Problema que levou à proposta
  - Branches ou mais especificamente Branches de funcionalidades (feature branches)
    - Podem levar meses para se integrar à linha principal (main)
    - Merge pode ocorrer conflitos de integração ou conflitos de merge

Integração Contínua (CI)

- Exemplo: Antigamente Feature branches eram comuns



40 dias

merge hell

integration hell

Termos usados para descrever os problemas que ocorrem durante a integração de branches de funcionalidades.

Integração Contínua (CI)

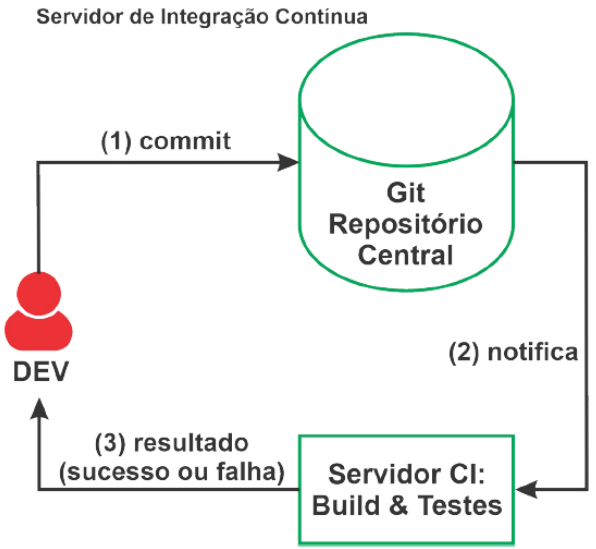
- O que é Integração Contínua (Continuous Integration ou CI)?
  - Prática de desenvolvimento proposta por Extreme Programming (XP)
  - "Se uma tarefa causa dor, não podemos deixar que ela acumule"... devemos quebrá-la em subtarefas
  - CI recomenda integrar o código de forma frequente

Boas Práticas para Uso de CI

- **Build Automatizado**
  - Build: compilação de todos os arquivos de um sistema para gerar uma versão executável.
  - Automatizado: sem nenhum passo manual
- **Testes Automatizados**
  - garantir que o sistema continua com o comportamento esperado após o novo build, executando testes automaticamente
- **Servidores de Integração Contínua**
  - responsável por executar o build automatizado e os testes sempre que houver mudanças

Servidores de Integração Contínua (CI)

- **Builds e testes automatizados devem ser executados com frequência**
- **Se possível após cada novo commit realizado no master**



circleci | Travis CI | Jenkins | GitHub Actions → monitoram a atividade no seu repositório

Integração Contínua (CI)

- **Evitar a integração de código com problemas**
  - Quando o build falha, costuma-se dizer que ele "quebrou"
  - Alguns motivos: o desenvolvedor esqueceu de realizar o commit de algum arquivo, dependências incorretas (biblioteca de versão anterior), etc…
- **Parar tudo o que está fazendo e providenciar a correção (ou reverter o código para a versão anterior)**
- **Somente avançar para próxima tarefa de programação após receber o resultado do servidor de CI**

Desenvolvimento Baseado Trunk

- Já que merges podem gerar conflitos, TBD defende:
  - Não usar mais branches de funcionalidades
  - Implementá-las diretamente no branch principal
  - Branch principal = trunk, master, main
  - Ex.: Google, Meta

Desenvolvimento Baseado Trunk

Quase todo desenvolvimento ocorre no HEAD do repositório. Isso ajuda a identificar problemas de integração mais cedo e minimiza o esforço para realização de merges.

Todos engenheiros trabalham em um único branch... o que torna o desenvolvimento mais rápido, pois não dispende-se esforço na integração de branches de longa duração no trunk.

Deployment Contínuo (CD)

- Próximo passo:
  - Commits/merge são realizados com frequência (CI)
  - E entram imediatamente em produção (CD)
  - Objetivo: experimentação e feedback!

Deployment Contínuo (CD)

- Exemplo:

Se não houver usuários "passando o mouse" sobre um novo elemento da UI, um experimento pode mover esse elemento para um novo local na tela.

Se todos os experimentos mostrarem falta de interesse, o elemento será removido do produto

Fonte: The Top 10 Adages in Continuous Deployment, IEEE Software 2017

Deployment Contínuo (CD)

- Fluxo de Trabalho quando usa CD
  - **O desenvolvedor desenvolve e testa na sua máquina local**
  - **Ele realiza um commit e o servidor de CI executa novamente um build e os testes de unidade**
  - **Algumas vezes no dia, o servidor de CI realiza testes mais exaustivos com os novos commits que ainda não entraram em produção**
  - **Se todos os testes passarem, os commits entram imediatamente em produção → CD !**

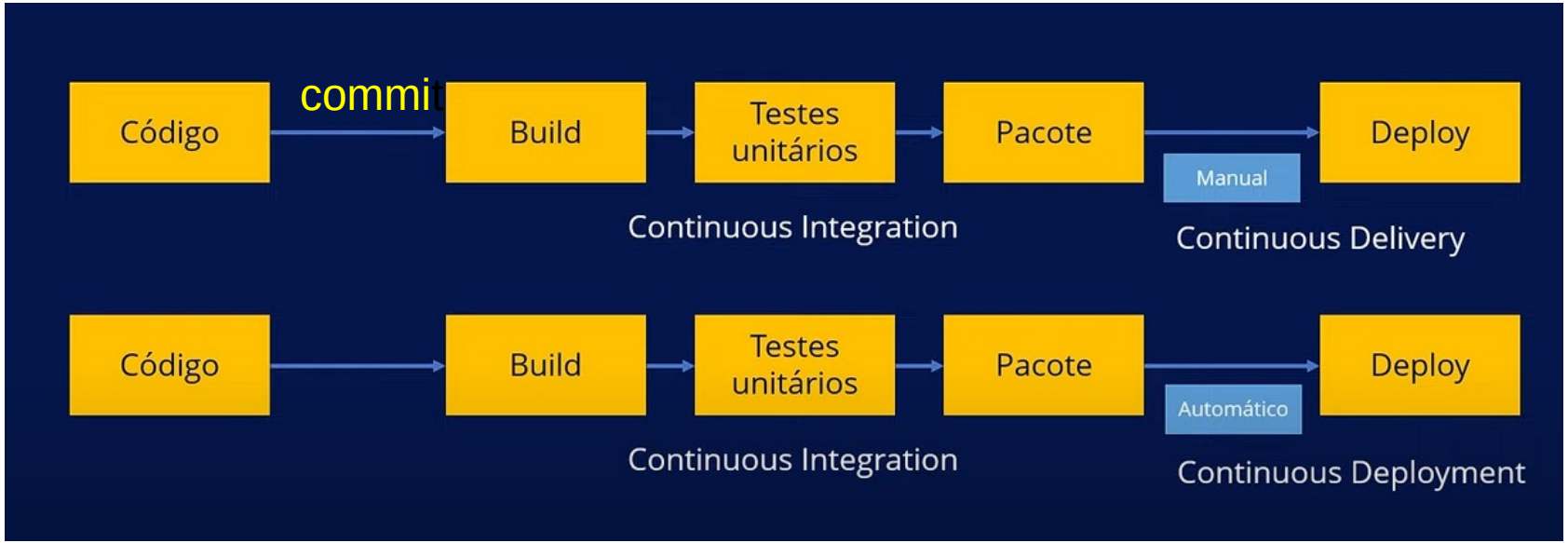
Vantagens do CD

- **redução no tempo de entrega de novas funcionalidades, que são liberadas assim que ficam prontas, diminuindo o intervalo entre releases e o número de releases.**
- **CD torna novas entregas um não-evento (não existe mais um dia D ou um deadline para entrega de novas releases) e evita que a perda de um deadline atrase a entrega de uma funcionalidade por meses.**
- **melhora a motivação dos desenvolvedores, que rapidamente recebem retorno — vindo de usuários reais — sobre o sucesso ou não de suas tarefas.**
- **favorece a experimentação e um estilo de desenvolvimento orientado por dados e feedback dos usuários.**

Entrega Contínua (CD/EC)

- Continuous Delivery
  - **Deployment Contínuo (CD) não é recomendável para certos tipos de sistemas**
    - desktop, app móvel ou embarcada que precisa de instalação para ser atualizada.
  - **Entrega Contínua (EC)**
    - Todo commit pode entrar em produção imediatamente
    - Gerente de projetos ou de releases, por exemplo, toma a decisão sobre quando os commits, de fato, serão liberados para os usuários finais

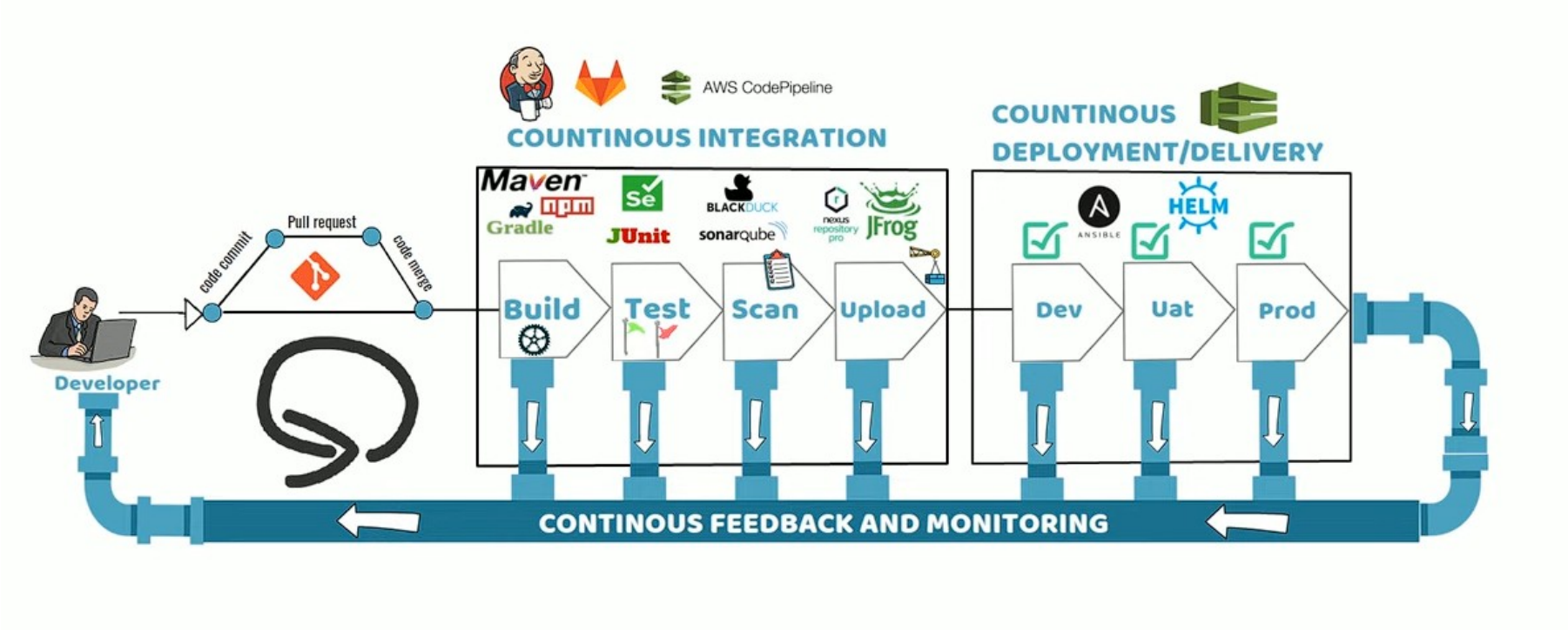
Fluxo de Trabalho



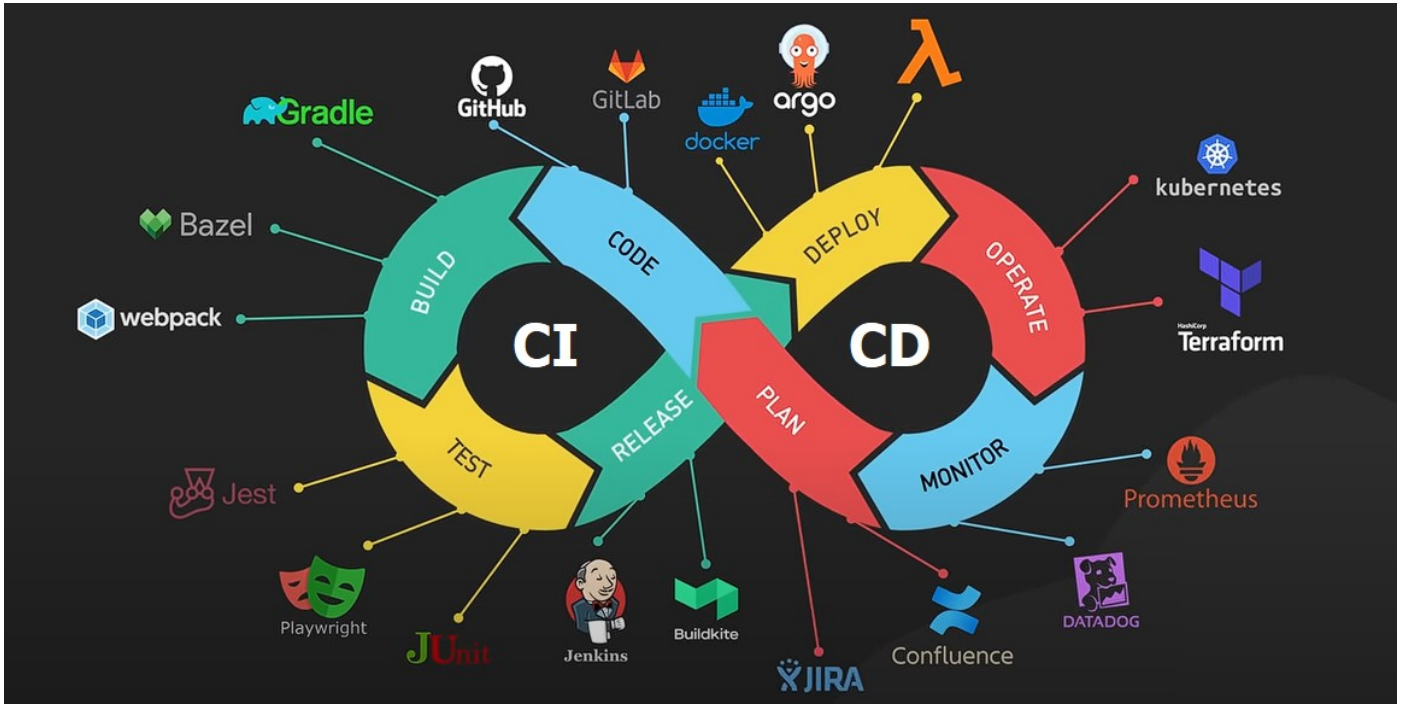
Diferença entre CD e EC

- **Diferença entre CD e EC**
  - **Deployment** é o processo de liberar uma nova versão de um sistema para seus usuários.
  - **Delivery** é o processo de liberar uma nova versão de um sistema para ser objeto de deployment.
  - Quando adota-se Deployment Contínuo (CD), ambos os processos são automáticos e contínuos. Porém, com Entrega Contínua (EC), a entrega é realizada com frequência, mas o deployment depende de uma autorização manual.

Detalhando a Pipeline de CI/CD



Ferramentas para CI/CD



Referências

- Material elaborado a partir de:
- Capítulo 10 do livro online: Engenharia de Software Moderna - Princípios e Práticas para Desenvolvimento de Software com Produtividade, autor: Marco Tulio Valente, disponível em: https://engsoftmoderna.info/
- Notações de aula (slides) disponibilizados pelo prof. Jean da UniRitter