Eficiência operacional em entrega de software

O que nós chamamos de “build” é a compilação de código que escrevemos em alguma linguagem de programação para que este código se torne em uma aplicação utilizável de fato, como um Word ou Excel que todos nós estamos acostumados a utilizar no dia a dia.

Agora falando especificamente do código-fonte em si e desse fluxo de “build” até que coloquemos a aplicação pronta no servidor, muita coisa mudou de alguns anos pra cá.

Especificamente falando da plataforma distribuída (ou seja, não será comentado mainframe), antigamente nós colocávamos o código-fonte em algum lugar compartilhado (uma pasta compartilhada na rede) ou versionávamos em algum software de controle de código como o Visual SourceSafe da Microsoft ou o IBM Rational ClearCase, ferramentas antigas que não possibilitavam o real trabalho colaborativo com várias pessoas mexendo no código ao mesmo tempo. Naquela época apenas 1 pessoa poderia fazer o “check-in” do código atualizado por vez e era comum haver uma enorme confusão e necessidade de organização por uma pessoa bem experiente para que tudo desse certo.

Além do código-fonte, boa parte das vezes o “build” era feito na máquina do desenvolvedor, manualmente pelo próprio desenvolvedor e o produto desse “build” que era a aplicação pronta era colocado numa pasta de rede ou enviado por e-mail para a pessoa que tinha acesso ao servidor para que este atualizasse a versão com a nova aplicação. Ou o próprio desenvolvedor às vezes tinha lá os “acessos de Deus” e entrava no servidor e colocava a nova aplicação “na mão”, manualmente, e

seja o que Deus quiser né, pois descobria-se na hora se esta implantação tinha dado certo ou errado.

Obviamente que todo esse processo manual ou até "artesanal" trazia riscos de erro humano, além de ser algo repetitivo e que consumia tempo e dinheiro, ou quando alguém perdia código-fonte não devidamente versionado ou esquecia de efetuar um backup da versão da aplicação que está sofrendo atualização. Enfim, são vários os motivos pelo qual este processo de criação de código até a entrega do mesmo evoluiu até o que temos de mais moderno: Continuous Integration, Continuous Delivery e Continuous Deployment.

Os chamados SCM – Source Control Management ou traduzido Gestão de Controle de Fonte evoluíram como ferramentas que permitem a colaboração entre várias pessoas ao mesmo tempo, não só isso mas através dessas ferramentas conseguimos criar fluxos automáticos em que você salva um código-fonte mais atualizado e o "build" é disparado em um servidor, não sendo necessário fazer isso manualmente e nem na máquina do desenvolvedor.

Este fluxo de "build" automatizado já testa a aplicação nova e até pode disparar a implantação em servidores, sem a necessidade de uma pessoa fazer manualmente e protegendo o servidor, uma vez que humanos não precisem ter acesso ao servidor. Obviamente que um fluxo automatizado a este nível permite voltar a uma versão anterior em caso de problemas, também tudo automático, por muitas vezes sem a necessidade de intervenção humana, trazendo mais segurança ao processo como um todo e retirando tempo e dinheiro dispendido com pessoas para este processo.

Essa parte do fluxo como um todo em que o desenvolvedor envia o código-fonte alterado, faz o "build" (compilação) e executa os testes unitários automatizados é o que chamamos de Continuous Integration (Integração Contínua), é através desse processo que o desenvolvedor tem a entrega de código de forma colaborativa em paralelo com outros desenvolvedores e em que há o feedback rápido, tanto no momento de "build" se o mesmo falhar por qualquer erro de codificação, quanto no

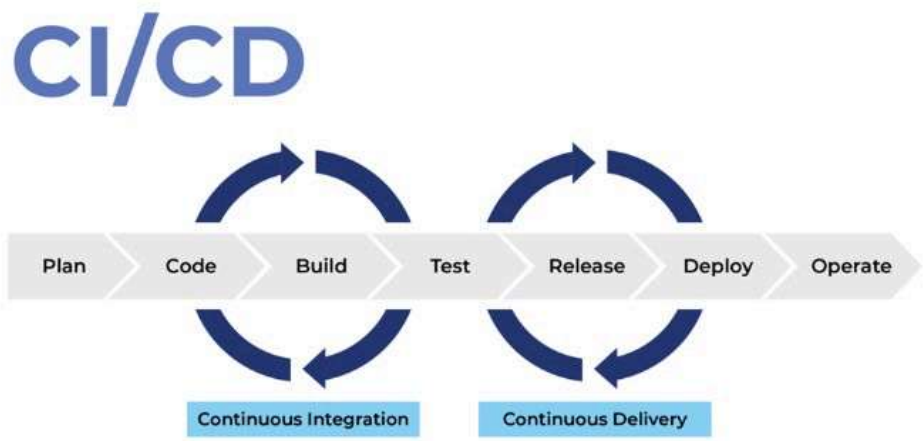
momento de testes unitários quando testa-se os fluxos de cada parte desenvolvida, em relação às regras de negócio codificadas na aplicação. É claro que não é só o ferramental e a tecnologia que determinam a adoção de Continuous Integration, mas é uma cultura onde o dia a dia do desenvolvedor permite um modelo ágil de trabalho com entregas frequentes (ao menos 1x ao dia), errar rápido para corrigir rápido.

A próxima etapa refere-se ao Continous Delivery (Entrega Contínua), onde o intuito é sempre deixar a próxima versão pronta para ir para a produção, então aqui a aplicação fica pronta em ambiente de testes para que sejam feitos os testes integrados de negócio (automatizados ou não). Aplicação devidamente testada funcionalmente significa que a aplicação estará pronta para ser entregue para o ambiente de produção, mais uma vez o modelo de trabalho ágil permite um fluxo de entregas pequenas e constantes.

A última etapa refere-se ao Continuous Deployment (Implantação Contínua), que é a implantação de fato no ambiente de produção, dada alguma estratégia de implementação que possa permitir eventual rollback (voltar para uma versão anterior estável) em caso de problemas. Todo esse fluxo deverá permitir a implantação de versões todos os dias, várias vezes ao dia, impactando o mínimo a disponibilidade dos produtos e serviços aos clientes.

Nesta aula vemos que ferramentas como o Gitlab nos permitem a colaboração entre desenvolvedores em times de entregas ágeis com controle de código-fonte organizado e automatizado, de forma a ajudar líderes técnicos a organizar as entregas à partir de código-fonte e dos desenvolvedores, ter um fluxo de CI/CD (Continuous Integration/Continuous Delivery) nesta mesma ferramenta facilita e acelera as entregas dos times, através do Gitlab Pipeline em que é possível escrever um script com uma espécie de "receita" que diz o que fazer com o código-fonte à cada atualização: se é para efetuar testes unitários, fazer o build, determinar como entregar a aplicação, onde entregar a aplicação, entre muitas possibilidades.

Podemos utilizar a tecnologia de containers através do Docker para garantir implantação contínua através dessas pipelines, subindo versões novas aproveitando a velocidade do modelo de containers oferecido pela ferramenta Docker e implantar em praticamente qualquer lugar, uma vez que o Docker é compatível com grande parte dos sistemas operacionais disponibilizados no mercado, tanto num modelo On-Premise, como em "nuvem privada" ou "nuvem pública".



Atividade Extra

Uma vez que iremos abordar constantemente o Gitlab CI/CD, é recomendado o estudo da documentação oficial do Gitlab (que está em inglês): https://docs.gitlab.com/ee/ci/pipelines/

Referência Bibliográfica

- GRANGEIRO, Fernando de Jesus. CI/CD utilizando Fastlane e Circle CI Parte 1, 23 de janeiro de 2020. Disponível em: https://medium.com/android-dev-br/ci-cd-utilizando-fastlane-e-circle-ci-parte-1-e339f3cf833a. Acesso em: 21 de maio de 2022.

Ir para exercício