Contents

Continuous Delivery (Entrega ou Delivery Contínuo)

Segundo a definição publicada no artigo "O que significa distribuição contínua?" publicado na Amazon, *Continuous Delivery é uma prática de desenvolvimento de software DevOps em que alterações de código são criadas, testadas e preparadas automaticamente para liberação para produção* Essa definição é interessante pois reformça a idéia de que todo o concieot DevOps é baseado em culturas, implementadas a partir de práticas que utilizam ferramentas.

Essa ideia pode ser analisada como uma evolução ou o próximo passo após a integração continua, ou seja, uma vez que o processo de integração e teste automatizado de código seja implementado é possível gerar de forma automática a versão ou artefato pronto para entrada em produção. Dessa forma cada alteração de código criada, é testada e enviada para um ou vários estágios subsequentes de validação para que por fim seja aprovada e implementada em produção.

Continuos Deployment (Deploy Contínuo)

É comum que no começo a terminologia seja confusa principalmente devido a um terceiro elemento (Além de ci e cd) que engloba pŕaticas devops para entrega de código, esse elemento é chamado de Deployment Contínuo (Se eu tentar abreviar chamaria de cd ai ficaria mais confuso ainda =D), a diferença entre delivery e deploy é sútil e se dá apenas no processo de entrega em produção, no modelo conhecido como "Contínuos Delivery" descrito acima existe um processo manual de aprovação, já no Contínuos Deployment a aprovação é automática, o que faz deste o estágio final de um modelo 100% ágil e consequentemente o mais díficil de ser alcançado.

A imagem abaixo foi retirada do paper já citado da amazon e ilusta bem os três modelos e as respectivas diferenças entre eles:

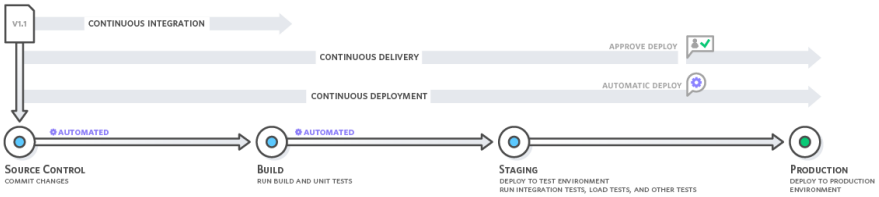

Figure 1: alt tag

Continuos Integration (Integração Contínua)

O termo "continuos integration" será um dos pontos bases a serem explicadas neste conteúdo, ele refere-se a uma prática adotada por times no processo de desenvolvimento de software onde um repositório é utilizado para centralizar o armazenamento de código. É partir deste repositório que testes de validação e integração de aplicações serão executados e também é a partir deste mesmo repositório que os binários e fontes de código para entrega de aplicações em produção serão obtidos;

Um dos principais objetivos por trás deste modelo é centralizar dados de forma que possamos encontrar e investigar bugs mais rapidamente o que se enquadra perfeitamente na filosofia "Fail Fast, Fail Often", melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações.

Dica:

No whitepapper O que significa integração contínua? a amazon descreve este modelo de forma sucinta sendo um bom começo para se entender o assunto. # Deployment Pipeline

Uma das bases mais importantes na implementação de metodologias ágeis e de uma cultura devops é compreender e evoluir o processo de delivery ou mais especificamente os mecanismos que estão entre a concepção da idéia e a entrega do código em produção, dentro dessa lógica a implementação de um pipeline é um assunto sobre o qual temos temos de trocar algumas idéias.

Conceitualmente um "pipeline" ou "deployment pipeline" é a implementação do processo de build, teste e deploy de aplicações, este processo é distinto em cada organização pois depende de vários fatores, mas o princípio básico envolvido é sempre o mesmo, entender isso é o primeiro passo para começar a entender como uma empresa/organização funciona e como as etapas referentes a entrega de código podem ser melhoradas.

O exemplo abaixo na figura 1.1 foi extraído do Livro de Jez Humble e demonstra um exemplo de um pipeline:

Figure 2: alt tag

Figura 1.1: Exemplo de um modelo simples de pipeline, partindo do commit de código e chegando até a liberação do release da Aplicação.

Um pipeline descreve o modelo usado no processo de deploy, este por exemplo inicia-se no commit de código, cada mudança feita no código, configurações de ambiente ou bases de dados gera um gatilho e consequentemente um processo, a próxima etapa deste processo considerando o modelo descrito será a criação de um binário, o que chamamos de "empacotamento", a partir dai ainda serão executados testes de aceite, testes de carga e finalmente a liberação deste suposto binario, o que em nosso exemplo seria a nossa nova release ou o que chamamos de "release candidate" dai a famosa nomenclatura "rc".

Se a nossa querida rc passar com glamour por cada um dos testes do pipeline ela poderá ser implementada (Não precisa de nenhum glamour, é s passar mesmo que fica de bom tamanho).

Essa é a essência e função básica de um modelo de pipeline, definir o método utilizado no processo de deployment de uma aplicação, o que está intimamente ligado a idéia de Delivery Continuo e Integração Continua de código.

Introdução

"The most important problem that we face as software professionals is this: If somebody thinks of a good idea, how do we deliver it to users as quickly as possible?"

O trecho acima refere-se ao primeiro paragráfo do primeiro capítulo do Livro "Continuos Delivery" dos Authores Jez Humble e David Farley, o livro é uma das melhores referências escritas até então sobre Delivery Continuos e todas as práticas que compõem essa prática, o que desenvolvedores, engenheiros e afins passaram a chamar de devops ou "cultura devops" se preferir, é exatamente sobre essa cultura e sobre essas práticas que trataremos neste material, a idéia

Figure 3: alt tag

é abordar o tema de forma prática com exemplos e contextos baseados em conteúdos já desenvolvidos sobre o assunto e utilizando ferramentas e plataformas que tornem isso possível.

Importante:

Boa parte dos textos são baseados e utilizam elementos dessa bibliografia, fica a indicação desse conteúdo que meio que funciona como uma biblia sobre o asssunto.

Terminologia e conceitos básicos

Para começarmos a aprofundar um pouco as coisas alguns conceitos precisam ser estabelecidos, alguns deles possuem capítulos próprios e serão destrinchados a frente porém ainda assim neste primeiro momento vale uma aprensentação para facilitar :

- Deployment Pipeline
- Continuous Integration
- Continuous Delivery e Continuous Deployment

Por que entregar código em produção é tão complicado (Será que é mesmo?)

Agora que já esclarecemos os três conceitos acima podemos começar a discutir o problema, ao falar em devops é muito comum que tentemos identificar e resolver alguns problemas que de forma direta ou indireta comprometem o tempo necessário para delivery de código em produção, neste momento chamaremos estes problemas de "Anti Patterns", listarei dois deles abaixo:

1. Entrega Manual de Software

Qualquer aplicação de médio porte desenvolvida hoje em dia vem recheada de componentes e depêndencias que a tornam complexa, isso sem falar na criticidade geralmente relacionada ao ambiente e sua disponibilidade, o que torna a coisa toda bem difícil de se manipular, entregar releases de aplicaçes que reunam essas caracteristicas manualmente em muitos casos não a melhor das alternativas mas ainda assim é o que muitas empresas fazem, considerando essas entregas manuais fica bem fácil identificar possíveis efeitos colaterais:

- A entrega manual de software está sujeita a julgamentos e tomada de decisões que podem variar entre cada delivery;
- Essas decisões estão naturalmente sujeitas a erros humanos;
- Mesmo que decisões corretas sejam tomadas alterações na ordem em que as coisas são feitas geram diferenças no resultado final (E isso quase nunca algom bom);

Baseado nesses sintomas vamos aos resultados esperados:

- Testes manuais geralmente são necessários, afinal se o processo não foi automatizado como garantir que terá sempre o mesmo resultado se não validando o que foi feito?;
- É comum que o processo seja segregado entre times, e nesses casos é igualmente comum que o time de desenvolviemnto tenha contato limitado ou não tenha qualquer contato ou acesso ao ambiente de produção;
- Também tendem a ocorrer correções manuais nos processos de release de código, o que as vezes ocorre no meio do delivery;
- Os ***Environments***(ambientes) que suportam a aplicação tendem a divergir entre si, (diferentes versões de bibliotecas, arquivos de configuração etc).

2. Entrega de código em produção somente após completar o esforço de desenvolvimento

A caracteristica marcante deste **“Anti pattern”* é que o primeiro deploy de uma determinada aplicação só será feita após o desenvolvimento ser completamente finalizado.

> OK... isso pode parecer algo bom mas não é (com muita boa vontade ser algo aceitável) ainda assim definitivamente não é uma prática ligada à cultura devops, tem em mente que esse primeiro deploy não precisa ser necessáriamente em produção, ambientes de staging, development, homologação estão inclusos.

O problema desse modelo é que geralmente o primeiro contato dos times de operações com a aplicação é retardado até o momento da entrega no ambiente, ainda que o mesmo time responsável pela aplicação seja o time que irá operacionalizar e manter o ambiente existe um retardo e neste caso seria no contato desse time com o ambiente. Essa lógica vai contra os principais modelos discutidos atualmente cujo foco é performance acabndo por inviabilizar metodologias como "Baby Steps" e entregas em "Small Batchs".

As dependências relacionadas ao ambiente também se tornam um problema aqui, os times tendem a definir instaladores, artefatos, arquivos de configuração, bases de dados sem que esses componentes sejam devidamente testados nos ambientes onde efetivamente serão aplicados na entrega do software.

> Tem outra probleminha ai... Humble e Farley descrevem em sua obra uma situação eminente relacionada a este **"Anti Pattern"*:
>
> **Em muitos modelos de delivery manual, quando ocorre a entrega ou deploy nos ambientes de homologação, uma equipe é montada para executar esse processo. Às vezes, essa equipe possui todas as habilidades necessárias, mas muitas vezes em organizações muito grandes as responsabilidades de implantação são divididas entre vários grupos. DBAs, equipes de middleware, equipes da web e outros, ajudam a implementar a versão mais recente do aplicativo. Considerando que em geral as várias etapas envolvidas nunca foram testadas juntas no ambiente, não é incomum que elas apresentem erros. Também ocorre da documentação e scripts/automação relacionada fazerem suposições sobre a versão ou configuração do ambiente de destino de forma errada, fazendo com que a implantação falhe... *

E se ao invés disso...

E se ao invés disso comessemos a integrar nossos componentes? Por exemplo desenvolver um modelo onde os testes são integrados a aplicação e onde testes específicos são utilizados para validar a comunicação com o ambiente e obter métricas de funcionamento, o que alías chamamos de testes de integração.

O processo de delivery naturalmente também seria mudado, nesse novo layout ele seria comum, diário, automatizado e com o tempo se tornaria normal a ponto de que a qualquer entrega um novo release já poderia ser gerado para ser aplicado em produção, para isso os ambientes deveriam ser homogêneos, e o time deveria ser capaz de recriar estes ambientes a qualquer momento conforme suas necessidades. Finalmente para que esse nível de intregação funcione teriamos de nos certificar de que todos os times envolvidos no processo de entrega de software, (desde a equipe de criação aos times de Q.A. e claro os desenvolvedores) trabalhem juntos desde o início do projeto.

Esse "E se" trás parte da lógica por trás do que chamamos de **Cultura DevOps**, a partir desse formato a imagem abaixo passa a fazer muito sentido:

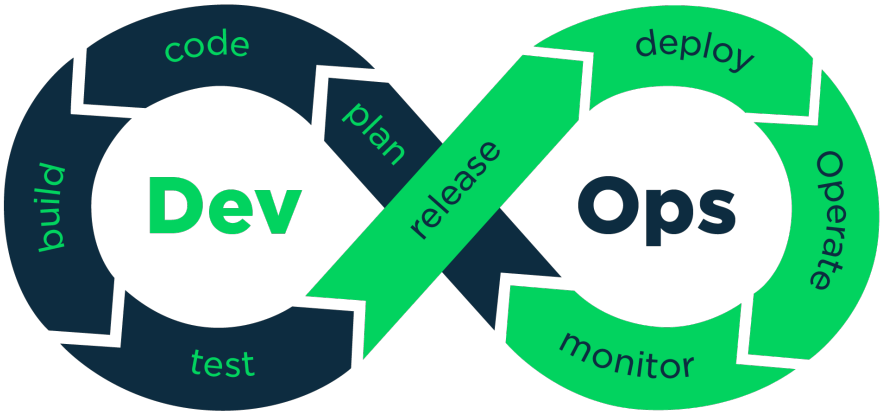

Figure 4: alt tag

Concluindo

No trecho acima foram descritas duas das muitas barreiras para que um time ou empresa adote uma cultura de desenvolvimento ágil, boa parte dessa questão está relacionada a velocidade com a qual o time em questão consegue entregar seu código em produção. Quanto a isso vale mais um trecho da bibliografia de J. Humble:

**... A velocidade é essencial porque existe um custo de oportunidade associado à não entrega de software. Você só pode começar a obter um retorno sobre seu investimento assim que seu software for lançado... *

Entregar software rápidamente também é importante porque permitirá que você verifique se seus recursos e correções de erros são realmente úteis. A entidade a julgar isso será sempre o usuário, Você pode fazer hipóteses sobre quais recursos e correções de bugs serão úteis, no entanto, até que estejam nas mãos do usuário essa hipóteses permanecem como hipóteses, sendo assim é vital minimizar o tempo desse ciclo para que o feedback efetivo possa ser recebido.

Tal como no livro do autor nesse material um dos objetivos principais é discutir e apresentar formas de reduzir o tempo de entrega, seja para um **bugfix* ou um recurso ou pata toda uma plataforma a ser disponibilizá-da para os usuários.

Material de Referência:

Conforme já citado umas 10 vezes muito de nosso materia lé baseado na obra "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" (O nome é enorme, eu sei..), dos autores Jez Humble David Farley publicado pela editora Pearson, o livro pode ser encontrado na Amazon, Google ou até em outros tipos de fontes mais duvidosas...

- Livro, Continuous Delivery: Reliable Software...

Além do material base existem ótimas boas referências em Português sobre o assunto, essa aqui foi escrita pelo Guto Carvalho da Instruct:

- O que é DevOps afinal?

Free Software, Hell Yeah!