**Unidade 02**

Disciplina: Arquiteturas para Entrega Contínua e DevOps

Professor(a): Marco Mendes

# Introdução a Automação de Releases

A automação das liberações (*Release Automation*) é uma prática que buscar garantir que o processo de promoção do executável para os ambientes de testes, homologação e produção sejam automatizados e assim tornados consistentes. Isso é importante porque em muitas organizações é difícil colocar um produto em ambiente de produção. A demora para acesso aos ambientes, alto número de passos manuais, a complexidade e a dificuldade de analisar os impactos são comuns. Isso gera atritos, longas demoras e desgates entre times de QA, desenvolvimento e operações. E no fim isso também provoca erros nos ambientes de produção.

Esta prática tem como principais benefícios:

- reduzir o tempo para entregar um novo *build* em ambiente de produção através da automação da instalação e configuração de ferramentas e componentes arquiteturais;
- reduzir erros em implantação causadas por parâmetros específicos que não foram corretamente configurados pelos times de desenvolvimento e operações;
- minimizar a fricção entre os times de desenvolvimento, QA e operações;
- prover confiabilidade e segurança no processo de implantar aplicações.

Um ponto de atenção é que a automação de releases não deve recompilar a aplicação. Para garantir consistência, ela deve garantir que o mesmo executável que foi montado na máquina do desenvolvedor (mesmo conjunto de bits) esteja operando em outros ambientes. Ou seja, a automação de releases faz a movimentação dos executáveis entre os ambientes e modifica apenas os parâmetros da aplicação e variáveis de ambiente. Este processo pode também envolver a montagem de máquinas virtuais em tempo de execução do script.

**Continuous Delivery – Entrega Contínua**

Depois que um processo mínimo de integração e implantação estiver em curso, podemos avançar também para um processo contínuo de implantação nos ambientes intermediários (testes ou homologação). Este processo pode ser controlado por uma ferramenta e

normalmente é ativado quando um novo build foi gerado pela ferramenta de automação de builds.

Em linhas gerais, a prática da implantação contínua garante que mesmo conjunto de bits do build é replicado no ambiente do desenvolvedor para ambientes controlados de desenvolvimento e homologação, garantindo a consistência do build em outros ambientes.

Em ambientes rigidamente governados onde existam processos ITIL, DBAs e times independentes de QA, naturalmente irão haver ciclo de pré-aprovações ou pós-aprovações para que as promoções de ambiente ocorram. O fato importante a notar na cultura DevOps é que o ser humano envolvido não copia arquivos ou parametriza aplicações. Ele examina a requisição, o build, parâmetros e as suas evidências de teste. Ao aprovar a solicitação de promoção, um robô irá fazer a movimentação dos builds e a parametrização apropriada da aplicação. Se ele não autorizar a promoção, um defeito é aberto para o solicitante no canal apropriado.

**Continuous Deployment – Implantação Contínua**

Em termos simples, a implantação continua é o processo de **entrega contínua em ambiente de produção**. Ela é normalmente ativada por uma ferramenta automatizada quando um novo build for publicado com sucesso no ambiente de homologação. A entrega contínua normalmente envolve:

- a requisição de aprovação de implantação para o responsável pelo ambiente de produção;
- a cópia dos arquivos do ambiente de homologação para o ambiente de produção;
- a verificação do estado da aplicação disponibilizada em produção.

Observe que a entrega contínua não significa que o ambiente de produção é modificado a todo instante. Apenas empresas de serviços de Internet podem e devem fazer isso. A entrega contínua implica que o ambiente de produção pode ser alterado se um novo build estiver disponível e as aprovações necessárias foram dadas para a promoção do build.

A entrega contínua envolve tipicamente a execução de *smoke tests*, que são testes de sanidade da aplicação. Estes testes verificam minimamente a estabilidade do build colocado em produção e testam cenários funcionais mínimos.

**Princípios da Automação de Liberações**

Quando estamos discutindo liberações, estamos falando em como mover com velocidade e segurança um build nos diversos ambientes da sua empresa (exemplo – testes, homologação e produção).

**Os princípios orientadores da gestão de releases**

1. Construa qualidade contínua
2. Trabalhe em pequenos lotes

3. Os computadores executam tarefas repetitivas, as pessoas resolvem problemas
4. Procure melhoria contínua, obsessivamente
5. Todos devem ser responsáveis

**Construa qualidade contínua**

Edwards Deming, figura-chave na história do movimento Lean e influenciador do movimento DevOps, ofereceu 14 princípios fundamentais para a gestão. O terceiro princípio diz – *"Eliminar a dependência da inspeção para alcançar a qualidade. Elimine a necessidade de inspeção em massa, construindo qualidade no produto em primeiro lugar ".*

É muito mais barato solucionar problemas e defeitos se os encontrarmos imediatamente – idealmente antes de serem verificados no controle de versão, executando testes automatizados localmente. Encontrar defeitos através da inspeção (como o teste manual) é demorado e caro.

Criar e evoluir loops de feedback para detectar problemas o mais cedo possível é essencial na entrega contínua. Se encontrarmos um problema em nossos testes exploratórios, não devemos apenas corrigi-lo, mas depois perguntar: como poderíamos ter captado o problema com um teste automatizado de aceitação? Quando um teste de aceitação falha, devemos perguntar: Poderíamos ter escrito um teste de unidade para capturar esse problema?

**Trabalhe em pequenos lotes**

Em abordagens tradicionais para o desenvolvimento de software as transferências de desenvolvimento para teste ou teste para produção de TI consistem em lançamentos completos: semanas ou meses de trabalho por equipes e fábricas com muitas pessoas. E a razão pela qual muitos times trabalham em grandes lotes é devido ao grande custo fixo de entregar as mudanças.

Na entrega contínua tomamos a abordagem oposta e buscamos obter todas as mudanças no controle de versão até o final da liberação, obtendo feedback o mais rápido possível. Um dos principais objetivos da entrega contínua é mudar a economia do processo de entrega de software para tornar economicamente viável trabalhar em pequenos lotes para que possamos obter os muitos benefícios dessa abordagem.

Trabalhar em pequenos lotes tem muitos benefícios. Isso reduz o tempo necessário para obter feedback sobre o nosso trabalho, facilita a triagem e a reação a problemas e aumenta a eficiência e a motivação.

**Computadores executam tarefas repetitivas, as pessoas resolvem problemas**

Uma das primeiras idéias filosóficas da tradição de Toyota é o jidoka, às vezes traduzido como "automação com um toque humano". O objetivo é que os computadores executem tarefas simples e repetitivas, como testes de regressão para que os humanos possam se concentrar na resolução de problemas. Assim, computadores e pessoas se complementam.

Muitas pessoas se preocupam que a automação os afastará de um emprego. Este não é o objetivo. Nunca haverá escassez de trabalho em uma empresa de sucesso. Em vez disso, as pessoas são liberadas do trabalho de trabalho despreocupado para se concentrar em atividades de maior valor. Isso também tem o benefício de melhorar a qualidade, uma vez que os seres humanos estão mais propensos a erros ao executar tarefas insensatas.

**Procure melhoria contínua, obsessivamente**

A melhoria contínua, ou kaizen em japonês, é outra idéia-chave do movimento Lean. Taiichi Ohno, uma figura-chave da história da empresa Toyota, disse uma vez,

*"As oportunidades de Kaizen são infinitas. Não pense que você tenha feito as coisas melhores do que antes e esteja à vontade … Isso seria como o aluno que se torna orgulhoso porque superou seu mestre duas vezes em três na esgrima. Uma vez que você escolhe as idéias kaizen, é importante ter a atitude correta em nosso trabalho diário porque depois de uma ideia Kaizen existe uma outra ideia Kaizen a ser descoberta."*

Não trate a transformação como um projeto que será iniciado e depois concluído para que possamos retornar ao negócio como de costume. As melhores organizações são aquelas em que todos tratam o trabalho de melhoria como parte essencial do seu trabalho diário e onde ninguém está satisfeito com o status quo.

**Todos devem ser responsáveis**

Em organizações de alto desempenho, nada é "problema de outra pessoa". Os desenvolvedores são responsáveis pela qualidade e estabilidade do software que eles criam. As equipes de operações são responsáveis por ajudar os desenvolvedores a desenvolver qualidade. Todos trabalham juntos para atingir os objetivos de nível organizacional, em vez de otimizar o que é melhor para sua equipe ou departamento.

Quando as pessoas fazem otimizações locais que reduzem o desempenho geral da organização, muitas vezes são devido a problemas sistêmicos como sistemas de gerenciamento pobres, ciclos orçamentários anuais ou incentivos que recompensam os comportamentos errados. Um exemplo clássico é recompensar os desenvolvedores por aumentar sua velocidade ou escrever mais código ou recompensar testadores com base no número de erros que eles encontram.

A maioria das pessoas quer fazer o que é certo, mas eles vão adaptar seu comportamento com base em como eles são recompensados. Portanto, é muito importante criar loops de feedback rápido das coisas que realmente importam: como os clientes reagem ao que construímos e o impacto em nossa organização.

Estabelecido esta base conceitual, vamos apresentar aqui uma escala de maturidade para a gestão de releases.

1. **Maturidade 1 – Inicial** – Aqui não existe uma cultura de gestão de releases. A entrega de um release nos ambientes de testes, homologação e produção é sempre feita de forma manual e sem o auxílio de ferramentas. Neste nível observamos que os desenvolvedores precisam trocar manualmente os apontamentos de banco de

dados e editar arquivos de configuração (exemplos – WebConfig em .NET ou web.xml e Java). Muitas vezes observamos rebotes e atritos com o time de infraestrutura e erros em produção devido a falhas humanas.

2. **Maturidade 2 – Consciente** – Aqui a cultura de releases começa a ser instalada. Ferramentas como Ansible ou Microsoft PowerShell começam a ser utilizadas para automatizar os passos de publicação. Embora o processo ainda exija intervenção humana, os passos estão documentados em scripts que podem ser disparados sob demanda.

3. **Maturidade 3 – Gerenciado** – Aqui os releases começam a ser publicados em intervalos regulares em ambientes controlados. Ferramentas como o Jenkins, GitLab, IBM Racional Team Concert, Microsoft TFS ou Microsoft VSTS entram em cena para pegar um build que já tenha sido produzido e mover este build para os ambientes de testes, homologação e produção. Neste nível informações sobre senhas de produção não são mais conhecidas pelo time de desenvolvidos. Os arquivos com estas informações são usados pelos ambientes de gestão de release de forma transparente. E aqui também existem smoke tests automatizados que verificam se o ambiente de testes, homologação ou produção não foram quebrados.

4. **Maturidade 4 – Avançado** – Aqui temos um incremento na frequência de implantação de *builds* e também na capacidade de desfazer publicações automaticamente. Os *builds* começam a ser implantados com frequência diária em ambientes de testes e homologação e as publicações em produção ocorrem conforme os calendários acordados com áreas de negócio. E um aspecto crítico observado neste nível também é a capacidade de desfazermos publicações caso alguma instalabilidade tenha sido observada em ambiente de produção. Esta capacidade de desfazer publicações, obviamente, é controlada por ferramentas para evitar erros humanos.

5. **Maturidade 5 – Melhoria Contínua –** Aqui o time está tão avançado que começa a experimentar a prática da implantação contínua (*continuous deployment*) e entrega contínua (*continuous delivery*). A primeira prática lida com a capacidade de publicar automaticamente builds nos ambientes de testes e homologação toda vez que um novo build tiver sido gerado pelo time. E a segunda prática lida com a capacidade de entregar builds em produção de forma automatizada e com governança toda vez que um build tiver sido aprovado no ambiente de homologação. Aqui vemos também os times experimentando com testes e ambientes canários (também chamados de testes A/B).

Observe que esta escala de qualidade trabalha com dois fatores: o esforço humano necessário para a publicação de builds e a frequência de publicação. Times pouco maduros tem muita dificuldade para fazer publicações e normalmente dependem de intervenções humanas para publicar uma aplicação. Já times com maturidade alta tem o seu processo totalmente controlado por robôs, que não ficam cansados, e conseguem gerar publicações com grande frequência.

**Recursos de Aprendizado**

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution.

Kim, G., Behr, K., & Spafford, K. (2014). *The phoenix project: A novel about IT, DevOps, and helping your business win*. IT Revolution.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education.