Unidade 02

Disciplina: Arquiteturas para Entrega Contínua e DevOps

Professor(a): Marco Mendes

Introdução a Automação de Builds

Na prática DevOps, um ***build*** é um código executável gerado a partir de instruções automatizadas e precisas de compilação a partir do código fonte e que atenda a um critério mínimo de testes.

A gestão de builds (ou *Build Management*) é prática essencial para garantir que os executáveis da arquitetura sejam gerados de forma consistente, em base diária. Esta prática busca evitar o problema comum do código funcionar na máquina do desenvolvedor e ao memo tempo quebrar o ambiente de produção.

A automação de builds externaliza todas as dependências de bibliotecas e configurações feitas dentro de uma IDE para um script específico e que possa ser consistentemente movida entre máquinas. Embora a automação de builds, na sua definição formal, lide apenas com a construção de um build, a prática comum de mercado é que builds devam executar um conjunto mínimo de testes de unidade automatizados para estabelecerem confiabilidade mínima ao executável sendo produzido.

Esta prática lida ainda com o estabelecimento de repositórios confiáveis de bibliotecas, o que garante governança técnica sobre o conjunto de versões específicas de bibliotecas que estejam sendo usadas para montar uma aplicação.

Observe que esta definição tem duas implicações práticas:

1. O código executável precisa ser gerado a partir de um arquivo chamado de definição de *build* para que possa ser compilado em qualquer ambiente. Isso evita a síndrome do desenvolvedor lambão que diz para você – "Na minha máquina compila".

2. O código executável deve passar com sucesso por uma suíte de testes automatizados para garantir sua estabilidade mínima (chamados de *smoke tests*). E isso evita a segunda síndrome do seu colega de trabalho, o desenvolvedor lambão, que diz para você – "Na minha máquina funciona".

Depois que a automação dos builds acontece, ela pode ser programada para ser executada em base diária ou até mesmos várias vezes por dia. Quando esta maturidade for alcançada, podemos avançar para que ela seja executada continuamente, i.e., toda vez que um um *commit* acontecer em um código fonte.

É esperado que esta prática faça pelo menos o seguinte conjunto de passos:

- promova a recompilação do código fonte do projeto;
- execute as suítes de teste de unidade automatizados do projeto;
- gere o build do produto;
- crie um novo rótulo para o build;
- gere defeitos automatizados para o time se o build falhou por algum motivo.

A prática da integração continua promove as seguintes vantagens:

- detectar erros no momento que os mesmos acontecem;
- buscar um ambiente de gestão de configuração minimamente estável de forma continuada;
- estabelecer uma mudança cultural no paradigma de desenvolvimento, através de feedbacks contínuos para o time de desenvolvimento da estabilidade do build.

Com essas definições apontadas, vamos definir uma escala de maturidade de automação de Builds.

1. **Maturidade 1 – Inicial** – Aqui não existe uma cultura de builds. A compilação está acoplada às IDES como o Eclipse ou Visual Studio e é totalmente dependente destes ambientes. É comum que códigos compilem ou rodem em máquinas específicas apenas. Neste estágio também não observamos uma suíte de automação de testes que buscam garantir a qualidade do *build*.

2. **Maturidade 2 – Consciente** – Aqui a cultura de builds começa a ser instalada. Ferramentas como o Make, Maven, Gradle, Rake, Nuget e MSBuild começam a ser utilizadas pelo time para eliminar dependências de máquina na compilação. Isso permite que o código seja compilado com facilidade em qualquer máquina. Neste nível o time também introduz um suíte mínima de testes de fumaça (*smoke tests*) para garantir a estabilidade do *build*.

3. **Maturidade 3 – Gerenciado** – Aqui os builds começam a ser executados em intervalos regulares em ambientes dedicados (ambientes de integração). Ferramentas como o Jenkins, GitLab, IBM Racional Team Concert, Ansible, Microsoft TFS ou Microsoft VSTS entram em cena para baixar do repositório os códigos fontes, compilar e testar os *builds*. Falhas na geração dos *builds* geram defeitos automatizados para o time de desenvolvimento. Neste nível também observamos uma maior

cobertura das suítes de testes (desejável maior que 20% do código fonte) e verificação automatizada das métricas de qualidade de código.

4. **Maturidade 4 – Avançado** – Aqui temos um incremento na frequência de execução do *build*. Os *builds* começam a ser executados várias vezes por dia e operam até mesmo em ambientes paralelos com objetivos distintos (ex. Ambiente de build para testes de performance e ambiente de Build para testes funcionais). A cobertura do código fonte é agora expressiva e normalmente passa de 50% do código fonte e inclui testes funcionais e outros tipos de teste como performance, usabilidade ou segurança.

5. **Maturidade 5 – Melhoria Contínua –** Aqui o time está tão avançado que ele começa a experimentar a prática da integração contínua (*continuous integration*), que formalmente definida é o processo de geração de *builds* disparado por modificações no código fonte. Em outros times, existem também políticas automatizadas que impedem o *commit* de código fonte que provoque quebra nos *builds* (padrão Gated Commit).

Podemos entender esta escala de maturidade a partir de dois fatores: frequência de execução e número de processos de qualidade executados. Um time com baixa maturidade não executa *builds* com frequência e não possui processos automatizados de qualidade que garantam o *build* gerado. Já um time de alta maturidade faz isso várias vezes por dia e cobre muitos processos técnicos tais como: testes de unidade, testes de tela, testes de integração, testes de segurança, testes de usabilidade, testes de performance, verificação da qualidade do código e até mesmo conformidade arquitetural a padrões pré-estabelecidos.

Naturalmente, este é um assunto denso e requer disciplina e tempo para ser implementado. Mas os resultados são notáveis e pagam, com juros, o esforço investido.

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.