Jenkins: uma apresentação geral

Gabryela Barros, Matheus Amorim, Samuel Nascimento, Victor Assis

Engenharia de Software – Universidade Católica do Salvador (UCSal)
Salvador – BA – Brasil

Resumo. *Este trabalho tem como objetivo explicar as características e os benefícios da ferramenta Jenkins, bem como indicar quando e porque utilizá-lo em um projeto.*

1. Introdução

A dinâmica utilizada na construção de software passou por mudanças ao longo dos anos. Algo que era comum há cinco anos no processo de desenvolvimento, hoje pode ser considerado obsoleto. Um grande exemplo é comparar as metodologias tradicionais com a metodologia ágil. Esta última é praticada de diferentes formas, dependendo do time e das diretrizes da empresa.

Com base nas mudanças das práticas de desenvolvimento, novas maneiras de trabalhar foram surgindo, e hoje é comum que integrantes de um time não estejam fisicamente próximos (prática presente na metodologia ágil). Com isso, surgiu a necessidade de utilizar ferramentas de apoio no desenvolvimento remoto (para facilitar o versionamento e manter o histórico do que foi produzido) e a integração de código dos membros constituintes do time.

A integração contínua, junto a um sistema de controle centralizado de versão, possibilita que os desenvolvedores de uma equipe de produção de software integrem o seu projeto frequentemente e tenham acesso a versões anteriores do mesmo e o registro dos seus respectivos autores.

Diante disso, o presente trabalho busca explanar alguns aspectos sobre o Jenkins, mostrando o papel fundamental que essa ferramenta exerce no auxílio à integração contínua.

2. OBJETIVOS

O principal objetivo deste trabalho é explicar, de forma clara, o funcionamento da ferramenta Jenkins e a sua real colaboração para o desenvolvimento de um software.

Para alcançar o objetivo, é necessário que alguns conceitos sejam explicados previamente. São eles: testes automatizados, build e deploy.

3. MÉTODOS/DESENVOLVIMENTO

É comum que tarefas repetitivas ocorram durante as etapas de desenvolvimento de um software. A energia utilizada para realizar atividades repetidas demandam tempo e custo a um projeto e, algumas delas, poderiam ser facilmente automatizadas. São os caso das fases de testes, builds e deploys. A curto prazo essa mudança de paradigma pode ser considerado dispendioso, mas a longo prazo traz muitos benefícios para o time, pois há ganho de produtividade, confiabilidade e integridade em tais tarefas. Mas afinal, o que são essas fases e para que elas servem?

Os testes é uma das soluções encontradas pelos membros de um time para garantir a qualidade de um produto. Eles podem ser de vários tipos e, em sua maioria, certificam se há

conformidade nos requisitos, como também se a solução implementada está de acordo com os tais requisitos. Com a automatização dos testes é possível criar rotinas periódicas para sua execução sem interferência humana, tornando esta tarefa mais frequente, ao passo que o retorno sobre possíveis falhas sejam reconhecidos com mais agilidade. Porém, não são todos os casos que necessitam de testes automatizados. É preciso que o time faça uma análise crítica para decidir se vale a pena automatizar os testes.

A etapa de build consiste em tornar a aplicação apta a ser publicada, ou seja, após a execução dos testes e verificação da conformidade, o build de um projeto prepara os artefatos para serem utilizados em um cenário real.

Já a etapa de deploy consiste em publicar no servidor de aplicação o pacote que foi gerado na etapa de build.

O processo de construção de um software passa por todas essas fases: codificação, commit, testes, build e deploy. Após a entrega do produto, há possibilidade de modificações (evolução de software) que podem ser corretivas, ou simplesmente para melhorar ou introduzir alguma funcionalidade. O processo de construção de software volta ao seu início e se repete quantas vezes forem necessárias. Agora imagine todo esse processo realizado manualmente e refeito quantas vezes forem necessárias, é no mínimo dispendioso. Volta à questão do gasto de energia com tarefas repetitivas.

Dados esses conceitos, o Jenkins atua na orquestração e organização dessas atividades tornando-as automatizadas, sem a interferência humana depois de um setup (configuração prévia). Além disso, os códigos são testados e buildado constantemente. Outra função do Jenkins é prevenir e alertar os desenvolvedores de qualquer problema encontrado após alguma atualização.

O Jenkins é uma ferramenta open source disponível para integração contínua de projetos, seja ele desenvolvimento ou infraestrutura.

4. RESULTADOS E DISCUSSÕES

Os resultados encontrados no presente estudo sugerem que a integração contínua é uma das principais soluções para problemas como colisões de código e desperdício de tempo no processo de desenvolvimento do software. Com o auxílio de ferramentas como o Jenkins esse processo se torna mais eficaz e fácil de ser controlado.

Por ser uma ferramenta de integração contínua, o Jenkins apresenta diversos benefícios para equipe de desenvolvedores, pois a ferramenta torna possível a construção do projeto de forma automática acompanhada da execução dos testes disponíveis, a fim de alertar antecipadamente os membros da equipe de possíveis erros no código.

5. CONCLUSÃO

Neste trabalho dissertamos em favor do uso de ferramentas que prestam auxílio a integração contínua no processo de desenvolvimento de software, mais especificamente, o Jenkins. Além de ser uma ferramenta open source, em versões atuais encontra-se apta à integrações com outras ferramentas através de plugins existentes na própria aplicação. Tornando-se útil para quaisquer tratamentos, testes e versionamento que a equipe de desenvolvedores necessitar para o projeto. Por fim, fica claro neste trabalho a contribuição positiva do uso dessa ferramenta.

Na sequência deste documento, um How To foi disponibilizado para auxiliar o download e instalação do Jenkins no Windows 10, acompanhado dos passos necessários para criação e gerenciamento de um novo job na ferramenta.

Referências

GOMES, Fabio. Integração contínua: introdução ao assunto. DevMedia. Disponível em <https://www.devmedia.com.br/integracao-continua-uma-introducao-ao-assunto/28002#ixzz2St6M3Y7F> . Acesso em: 23. Maio. 2018.

GARCIA, Diego. SVN e Jenkins: Integração contínua. DevMedia. Disponível em <https://www.devmedia.com.br/svn-e-jenkins-integracao-continua/30833>. Acesso em: 23. Maio. 2018.

JENKINS. Jenkins. Disponível em <https://jenkins.io/>. Acesso em: 23. Maio. 2018.

WIKIVERSIDADE. Github - Jenkins. Wikiversity. Disponível em <https://pt.wikiversity.org/wiki/Github_-_Jenkins>. Acesso em: 23. Maio. 2018.

RODER. Larissa. Blog DB1. A importância da Integração Contínua, utilizando Jenkins e SonarQube. Disponível em <https://blog.db1.com.br/importancia-da-integracao-continua-utilizando-jenkins-e-sonarqube/>. Acesso em: 23. Maio. 2018.