Uma proposta de licenciamento *open source* e criação de um *pipeline* para evolução da *WebSTAMP*

Maria Luiza Fernandes[1], Otávio Augusto Faria[1], Rodrigo Martins Pagliares[1]

[1] DCC/Unifal-MG Universidade Federal de Alfenas, Alfenas, Brasil.

maria.luiza@sou.unifal-mg.edu.br, otavio.faria@sou.unifal-mg.edu.br, rodrigo.pagliares@unifal-mg.edu.br

Abstract. *Currently, open source software projects use DevOps practices aiming at the integration and continuous delivery of new features to bring agility to software development, delivery, and deployment processes. WebSTAMP is a closed source tool for hazard analysis, in constant evolution, expected to become an open source tool. The software development process used by developers to integrate and deliver new WebSTAMP functionality prescribes a sequence of manual steps that demands a lot of time and effort from the development team, not benefiting from the use of DevOps practices, principles and values. Because it is a manual activity, the integration and delivery of new features is error-prone. Therefore, it is essential to automate the sequence of steps to reduce errors and facilitate the process of integration and delivery of WebSTAMP features. We propose in this work: (i) an open source license for WebSTAMP and (ii) the creation of a pipeline, consisting of development, integration and continuous delivery workflows, which automates the steps performed manually for the evolution of WebSTAMP. We carried out a study on open source licenses for use in WebSTAMP and a comparative study between tools for creating pipelines. Based on the studies conducted, we decided on the Creative Commons license and opted for using Github Actions to implement the pipeline. With the results obtained, we can conclude that (i) an open source license has the potential to facilitate the participation of new collaborators in the evolution of WebSTAMP and (ii) the implementation of a pipeline automates the sequence of steps for integration and delivery of WebSTAMP functionalities.*

Resumo. *Atualmente, projetos de software open source utilizam práticas de DevOps objetivando a integração e entrega contínuas de novas funcionalidades para trazer agilidade nos processos de desenvolvimento, entrega e implantação de software. A WebSTAMP é uma ferramenta de código fechado para análise de hazards (situações de perigo), em constante evolução, com previsão de se tornar uma ferramenta open source. O processo de desenvolvimento de software usado por desenvolvedores para a integração e entrega de novas funcionalidades da WebSTAMP prescreve uma sequência de passos manuais que demanda muito tempo e esforço do time de desenvolvimento, não se beneficiando do uso de práticas, princípios e valores DevOps. Por ser uma atividade manual, a integração e entrega de novas funcionalidades é propensa a erros. Por isso, torna-se fundamental automatizar a sequência de passos para reduzir erros e facilitar o processo de*

integração e entrega de funcionalidades da WebSTAMP. Nós propomos neste trabalho: (i) uma licença open source para a WebSTAMP e (ii) a criação de um pipeline, constituído por workflows de desenvolvimento, integração e entrega contínuas, que automatiza os passos realizados manualmente para a evolução da WebSTAMP. Realizamos um estudo sobre licenças open source para utilização na WebSTAMP e um estudo comparativo entre ferramentas para criação de pipelines. A partir dos estudos conduzidos, decidimos pela licença Creative Commons e optamos pelo uso do GitHub Actions para implementação do pipeline. Com os resultados obtidos, podemos concluir que (i) uma licença open source tem o potencial de facilitar a participação de novos colaboradores na evolução da WebSTAMP e (ii) a implementação de um pipeline automatiza a sequência de passos para integração e entrega de funcionalidades da WebSTAMP.

1. Introdução

Atualmente, projetos de *software* de código aberto (*open source*) utilizam práticas de *DevOps* [KIM et al. 2016] objetivando a integração contínua de novas funcionalidades para trazer agilidade no processo de desenvolvimento, eliminando barreiras entre times de desenvolvimento (*Dev*) e de operação (*Ops*) do *software* no ambiente de produção.

Apesar da tendência de uso de práticas *DevOps* em projetos *open source* e dos benefícios de *DevOps* descritos na literatura [KIM et al. 2016], ainda é comum nos depararmos com projetos de desenvolvimento de *software* que não se beneficiam das vantagens preconizadas por suas práticas, princípios e valores.

A *WebSTAMP* [SOUZA et al. 2019] é um exemplo de ferramenta *web* que não utiliza *DevOps* em seu processo de desenvolvimento. Trata-se de uma ferramenta de código fechado, em constante evolução (com inclusão de novas funcionalidades e manutenção de funcionalidades existentes), e com previsão de se tornar uma ferramenta *open source*.

A *WebSTAMP* tem como objetivo auxiliar analistas de *safety* (segurança) na confecção de análises de *hazards* (situações de perigo). Um *hazard* é um estado ou conjunto de condições de um sistema que, em combinação com um conjunto particular de piores condições ambientais, pode ocasionar perdas (acidentes) inaceitáveis do ponto de vista de um *stakeholder*, tais como perda de vidas, econômica ou ambiental [LEVESON e THOMAS 2018]. Pista escorregadia em uma rodovia (devido a óleo, aquaplanagem, entre outros) é um exemplo de *hazard*, já que em combinação com outros fatores tais como veículos em alta velocidade e falta de sinalização, pode ocasionar perdas inaceitáveis.

O processo de desenvolvimento, integração e entrega de *software* usado atualmente para evolução da *WebSTAMP* prescreve uma sequência manual de passos que demanda muito tempo e esforço do time de desenvolvimento. A sequência de passos constitui fluxos de trabalho (*workflows*) desempenhados pelos desenvolvedores e mantenedores da *WebSTAMP*.

Iniciar as dependências de *software* necessárias para o desenvolvimento, executar manualmente os testes e, implantar (*deploy*) no ambiente de produção as novas funcionalidades ou manutenção de existentes, são exemplos de passos presentes nos

workflows para desenvolvimento, integração e entrega contínuas de funcionalidades para a *WebSTAMP*.

Dessa forma, acredita-se que automatizando esses passos, o esforço e tempo necessários para evolução da *WebSTAMP* serão reduzidos e, ao mesmo tempo, aumentará a qualidade do código entregue no ambiente de produção, devido à execução e verificação de cobertura de testes de forma automatizada.

Este trabalho tem como objetivos (i) a proposta de uma licença *open source* para a *WebSTAMP* e (ii) a criação de um *pipeline*, constituído por *workflows* de desenvolvimento, integração e entrega contínuas, que automatiza passos realizados manualmente para a evolução da *WebSTAMP*, como executar testes, analisar estaticamente o código, realizar o cálculo da cobertura de testes, e implantar a *WebSTAMP* em um ambiente de produção.

Com relação ao primeiro objetivo (licenciamento *open source*), a *WebSTAMP* possui alguns requisitos, citados na Seção 2, para que um colaborador externo ao time de desenvolvimento possa utilizar seu código. Existem diversas licenças *open source* que apresentam características que podem ou não atender os requisitos estipulados pela equipe atual de gestão e desenvolvimento (mantenedores) da *WebSTAMP*. Desta forma, foram analisadas sete licenças *open source* a fim de escolher a melhor opção para inclusão na *WebSTAMP* respeitando os requisitos definidos pelos mantenedores.

Devido ao desejo de mudança de licenciamento da *WebSTAMP* de proprietário para *open source*, foi necessária a escolha de um *workflow* para colaboração [ABILDSKOV 2020]. Optamos por utilizar o *GitHub* como a plataforma base deste *workflow* tendo em vista que o *GitHub* é o local proposto para armazenamento do repositório *open source* com o código-fonte da *WebSTAMP.* O Acesso ao código-fonte foi gentilmente cedido aos autores deste trabalho por seus mantenedores para validação dos resultados.

Os seguintes *workflows* foram analisados: *Centralized Workflow*, com apenas uma *branch* principal (*main*); *Feature Branch Workflow*, em que cada alteração é criada uma *branch*; *Gitflow Workflow*, o qual possui seis *branches* sendo que cada uma delas tem um propósito específico estipulado; e *Forking Workflow*, em que é feito uma cópia independente do repositório principal para o repositório do colaborador e o mantenedor decide se as alterações serão aceitas no repositório principal.

De forma a atender o segundo objetivo deste trabalho (criação de um pipeline), apresentamos um estudo comparativo entre tecnologias para criação de *pipelines* de integração contínua: *Jenkins* [SMART 2011], *Travis CI* [BELLER et al. 2017] e *Github Actions* [BELMONT 2018]. Além disso, também conduzimos um estudo comparativo entre plataformas de computação em nuvem: *Google Cloud* [HUNTER e PORTER 2018], *Amazon Web Services* [WITTIG M. e WITTIG A. 2018], *Azure* [COPELAND et al. 2015], e *Heroku* [MIDDLETON e SCHNEEMAN 2013], para hospedagem e entrega contínua da *WebSTAMP*.

Tendo em vista os pontos analisados, para distribuição em código aberto da *WebSTAMP* sugerimos a licença *Creative Commons BY-NC-SA.* Além disso, para a construção do *pipeline,* escolhemos a ferramenta *Github Actions* e como plataforma para *deployment* a *Heroku Cloud.*

Este trabalho está organizado da seguinte maneira: na Seção 2 é apresentada uma discussão sobre *DevOps* e sua relação com este trabalho; a Seção 3 apresenta trabalhos relacionados; uma revisão da literatura acerca de licenças de *software open source* é apresentada na Seção 4; a proposta de um *pipeline* para desenvolvimento, integração e entrega contínua para a *WebSTAMP* é apresentada na Seção 5; a Seção 6 apresenta discussões com relação ao desenvolvimento do presente trabalho; a Seção 7 apresenta as conclusões e trabalhos futuros; e por fim, apresentam-se as referências bibliográficas.

2. *DevOps*

O termo *DevOps* refere-se a uma cultura que prega a colaboração entre todas as partes envolvidas no desenvolvimento e operação de um *software,* visando tornar o desenvolvimento mais veloz e aumentar a qualidade da operação de um sistema.

As organizações que exercem os princípios e práticas de *DevOps* obtêm melhora na produtividade, nos tempos de entrega, na qualidade do produto e na satisfação do cliente [KIM et al. 2016]. Além disso, implementar essas práticas em uma empresa traz vantagens competitivas pelo fato de permitir a entrega mais rápida de valor para os clientes [KIM et al. 2014].

Para que as práticas de *DevOps* consigam fornecer um desenvolvimento veloz é necessário automatizar as atividades [KRISHNAN 2019]. Isso é possível com a construção e definição de um *pipeline* constituído por etapas repetíveis e automatizadas que permitam com que a entrega e evolução do *software* seja contínua de forma consistente e confiável.

Já para que se obtenha uma maior qualidade na operação do *software,* realiza-se a execução automatizada de testes que minimizam as chances de *bugs* no sistema. Por conta disso, o sistema oferece maior confiabilidade e usabilidade para o usuário na operação do mesmo.

3. Trabalhos Relacionados

LIN et al. (2006) apresentam um estudo comparativo entre as onze licenças mais comumente utilizadas em projetos *open source*. Os autores trazem o modelo de licenciamento *Creative Commons* como base de comparação para as características das onze licenças analisadas a fim de deixar clara as suas semelhanças e diferenças.

KAPITSAKI et al. (2015) apresentam um estudo investigativo sobre licenças para sistemas de *software* de código aberto. Os autores fazem uma análise comparativa das características das licenças e apresentam uma visão geral crítica sobre cada uma.

ZHU et al. (2016) discutem acerca do conceito de *DevOps* e suas práticas. O *DevOps* tem como um dos seus principais objetivos facilitar a alteração no código-fonte, desde a codificação na máquina do desenvolvedor até a implantação (*deployment*) em um ambiente de produção. Os autores discutem mais a fundo sobre o tema, e também, trazem como discussão o uso do *DevOps* em arquiteturas de *software* baseadas em microsserviços e suas ferramentas.

MOHAMMAD (2016) apresenta um estudo sobre integração contínua e automação. O autor traz pontos que mostram a relação entre a importância da automação e o conjunto de práticas *DevOps*, em especial a prática de integração contínua.

MORRIS et al. (2017) apresentam uma aplicação do *Docker* para implantação e teste em um *software* de astronomia. O *Docker* é uma ferramenta que faz conteinerização, uma forma de virtualização em nível de sistema operacional. Os autores apresentam a aplicação do *Docker* em dois projetos, um mais simples e outro serviço mais complexo para exemplificar como o *Docker* funciona, bem como suas vantagens. Vale ressaltar que os desenvolvedores da *WebSTAMP* utilizam a tecnologia *Docker* para desenvolvimento e testes em ambientes locais (ambientes sem necessidade de conexão com a *Internet*).

BUNYAKIATI e SAMMAPUN (2019) apresentam *workflows* típicos utilizados em projetos de desenvolvimento de *software open source*. Os autores investigam o impacto da adoção das ações prescritas no *GitHub Actions* e apresentam uma visão geral de como as ações influenciam na quantidade de *Pull Requests* rejeitados nos repositórios.

LOPES (2020) apresenta uma aplicação de integração contínua em um sistema de auxílio à tomada de decisão. No trabalho são automatizados passos que eram realizados manualmente utilizando as ferramentas *Travis CI*, *CircleCI* [SOCHAT 2018] e *Jenkins*. A autora conclui que a automação dos passos realizados utilizando as ferramentas citadas diminui o esforço dos administradores e, portanto, se mostra eficaz para integração contínua de alterações no sistema.

KHAN et al. (2020) apresentam conceitos sobre as práticas de entrega rápida, testes, e construção e implantação contínuas com a aplicação de técnicas de *pipeline* de *DevOps*. Os autores mostram, como resultado, passos para automatizar essas práticas utilizando um *pipeline* de *DevOps* em nuvem.

KINSMAN et al. (2021) usam a ferramenta *GitHub Actions* a fim de mensurar alguns indicadores de atividade (por exemplo, o número mensal de *pull requests* que foram unidas à *branch* principal do repositório) para concluir se a ferramenta é uma boa alternativa para criação de *pipelines*. Os autores concluem que apesar de *GitHub Actions* ser uma nova ferramenta, até o momento há uma percepção positiva sobre ela e também afirmam que com a sua adoção o número de *Pull Requests* rejeitados é maior, indicando que o número de alterações não desejadas adicionadas no código é menor com o uso do *GitHub Actions*.

4. Licenças de Software *Open Source*

Analisamos sete licenças de *software,* escolhidas dentre as onze mais utilizadas pela comunidade segundo a literatura [LIN et al. 2006], levando em conta os requisitos definidos pelos mantenedores com objetivo de escolher a melhor opção para migração do código-fonte atual da *WebSTAMP*, fechado, proprietário, para um modelo de código-fonte aberto (*open source*).

A análise foi feita respeitando os requisitos definidos em reunião com os mantenedores da *WebSTAMP*. O principal requisito levantado foi de não permitir a comercialização do

projeto modificado. O Quadro 4.1 descreve as licenças analisadas e suas principais características.

A coluna *"Uso Comercial"* indica a permissão ou não de uso comercial após alteração do código-fonte de origem. A coluna "*Garantia"* indica se há suporte para possíveis erros ao utilizar o código-fonte. A coluna "*Documentação das modificações"* diz se é necessário documentar o que foi alterado.

A coluna "*Distribuição na mesma licença"* indica se é necessário distribuir novas alterações na mesma licença do código-fonte inicial. E por fim, a coluna "*Divulgar código-fonte*" indica a necessidade do colaborador de divulgar ou não o código-fonte quando o material licenciado for distribuído.

Definir uma licença *open source* para a *WebSTAMP* significa dizer que os direitos de propriedade intelectual do código-fonte estão disponíveis para domínio público, permitindo uso e alteração de forma gratuita, sem necessidade de se adquirir uma licença paga.

Com base na análise das licenças, sugerimos a adoção da *Creative Commons - CC BY-NC-SA* [COMMONS 2009] cujas características apresentadas no Quadro 4.1 vão ao encontro com o que nos foi solicitado pelos mantenedores da *WebSTAMP* (documentação das modificações, distribuição na mesma licença e divulgação do código-fonte).

Quadro 4.1. Comparativo de licenças de *software* para uso na *WebSTAMP*

ID	Licença	Uso Comercial	Garantia	Documentação das modificações	Distribuição na mesma licença	Divulgar código-fonte
1	GNU GPLv3	Sim	Não	Sim	Sim	Sim
2	MIT	Sim	Não	Não	Não	Não
3	Mozilla Public 2.0	Sim	Não	Não	Sim	Sim
4	Apache 2.0	Sim	Não	Sim	Não	Não
5	CC BY-NC-SA	Não	Não	Sim	Sim	Sim
6	CC BY-NC	Não	Não	Sim	Não	Sim
7	CC BY-ND	Sim	Não	Sim	Não	Sim

5. Um *pipeline* para evolução da *WebSTAMP*

A Figura 5.1 apresenta o *pipeline* constituído de *workflows* de *desenvolvimento*, *integração* e *entrega contínuas* para facilitar a colaboração entre desenvolvedores e evolução da *WebSTAMP*.

Conforme pode ser visto na figura (parte superior esquerda), um *Colaborador* pode ser um membro do time de desenvolvimento da *WebSTAMP* ou um desenvolvedor não

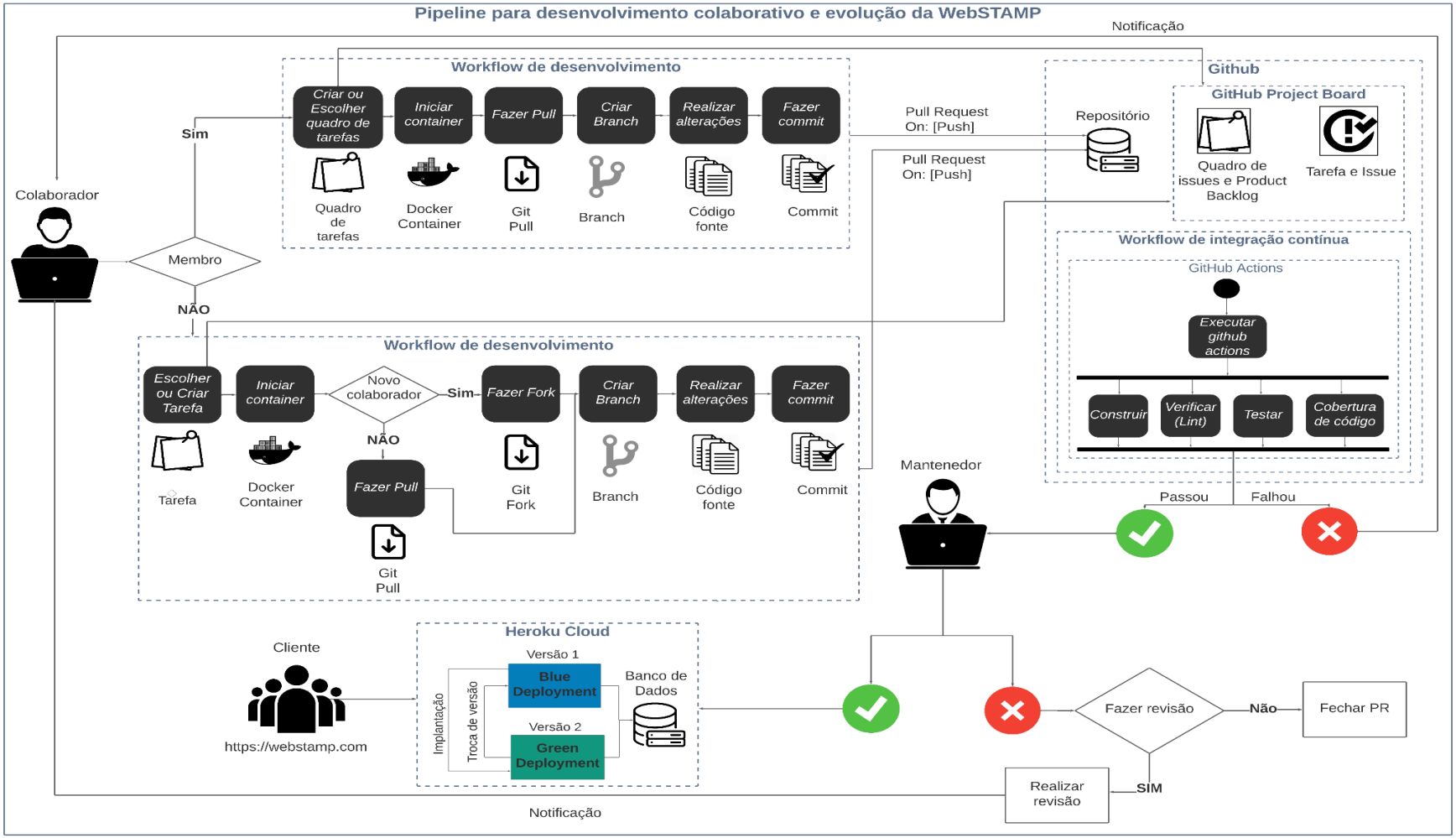

Figura 5.1. *Pipeline* para desenvolvimento colaborativo e evolução da *WebSTAMP*

vinculado ao time de desenvolvimento, mas que pretenda colaborar com a evolução da *WebSTAMP* (a contribuição de pessoas não diretamente vinculadas ao time de desenvolvimento é característica comum em projetos de desenvolvimento de *software open source* [VON KROGH 2003]). A sequência de passos para os dois tipos de *Colaboradores* se diferencia apenas no *workflow* de desenvolvimento, que é o primeiro conjunto de passos que constitui o *pipeline*.

Quando o *Colaborador* é membro da *WebSTAMP*, o *workflow* de desenvolvimento é iniciado pelo passo *Criar ou Escolher quadro de tarefas*, que tem como artefato de saída o *Quadro de tarefas* (visível na parte direita da figura - *GitHub Project Board*) que diz quais tarefas os membros têm para fazer para evolução da *WebSTAMP*. Neste passo propomos o uso do *GitHub Project Board* [BEER 2018] que contém tarefas e questões (*issues*) criadas no repositório do projeto. Uma *issue* é qualquer problema encontrado no código-fonte do *software*, tal como um *bug*, solicitação de alteração, ausência de documentação, implementação de novo requisito, entre outros. Uma *issue* é normalmente associada a uma prioridade (por exemplo, alta, média, baixa) [TAN et al. 2020].

Após criar ou escolher o *Quadro de tarefas*, é necessário executar o *container*, que inicia as dependências de *software* necessárias para executar o projeto, tendo como artefato de saída o *Docker Container*. Um *Docker Container* é uma unidade de *software* que engloba código e dependências de *software* de forma a permitir a rápida execução do *software* independente do sistema operacional usado pelo desenvolvedor.

No próximo passo do workflow de desenvolvimento, o ambiente de desenvolvimento local é atualizado via comando pull [PIPINELLIS 2018]. O comando pull permite obter do repositório remoto no GitHub a última versão do código-fonte da WebSTAMP. O artefato de saída para este passo é a versão mais recente do código-fonte da WebSTAMP na máquina local do desenvolvedor.

Com o ambiente de desenvolvimento local atualizado, é criada uma ramificação (*branch*), que é uma cópia da *branch* principal do projeto, criada para tornar mais segura as alterações no código-fonte original, evitando que defeitos sejam introduzidos inadvertidamente por desenvolvedores durante evolução do código da WebSTAMP. Conforme visto na figura, o artefato produzido por este passo é uma nova *branch*.

Tendo o ambiente local configurado, são realizadas as alterações no código, que tem como artefato de saída o código-fonte com as modificações. Por fim, é feito *commits* na *branch* local. *Commit* é o termo usado quando adicionam-se as alterações realizadas no código no repositório local. Os *commits* realizados formam o artefato produzido neste passo do *workflows*.

O *workflow* de desenvolvimento é basicamente o mesmo quando o colaborador não é membro da WebSTAMP, porém se diferencia no primeiro passo em que não é criado um Quadro de tarefas e sim criada uma nova tarefa ou escolhida uma tarefa existente no GitHub Project Board.

Além disso, também se diferencia no passo de realizar o Git pull, já que para não membros que ainda não são colaboradores é feito o Git fork, que faz uma cópia do repositório da WebSTAMP para o repositório do colaborador. Quando o não membro já

é um colaborador externo (já foi realizado o Git Fork do repositório) é feito o Git pull, sendo que para este caso é necessário ser configurado o sincronismo do repositório remoto do Colaborador com o repositório da WebSTAMP para que novas atualizações sejam integradas.

Após realizar os passos do *workflow* de desenvolvimento, é criado um *Pull Request*, que basicamente envia para o repositório remoto as atualizações locais do *Colaborador*. O *Pull Request* dispara um evento no servidor do *GitHub* que inicia a execução do *workflow* de integração contínua (parte central à direita na figura) que executa os passos descritos no *workflow*, via *GitHub Actions*.

O *workflow* de integração contínua é constituído pelos seguintes passos, cuja execução não necessariamente se dá de forma sequencial: (i) *Construir* (*build*), no qual são construídas todas as dependências de *software* necessárias para execução do projeto; (ii) *Verificar* (*lint*), passo em que é feita uma análise sintática de padrões de código nas alterações realizadas pelo *Colaborador*; (iii) *Testar*, passo para a execução de testes automatizados que testam se as alterações realizadas não prejudicam a qualidade do código; (iv) *Analisar cobertura de código*, no qual é criado e gerado um relatório de cobertura de código com informações sobre a porcentagem do código-fonte (linhas de código, número de métodos, número de classes, entre outros) cobertos por testes automatizados.

Se houver alguma falha na execução do *workflow* de integração contínua, o *Colaborador* recebe uma notificação e, após sanar as falhas identificadas, na notificação inicia novamente a execução do *pipeline*. Caso contrário (ausência de falhas), o *Pull Request* é enviado para o *Mantenedor*, que é responsável por gerenciar o repositório, analisar as alterações feitas pelo *Colaborador* e decidir se as alterações no código devem ser integradas ou não na *branch* principal no repositório da *WebSTAMP*.

Caso o *Mantenedor* não aprove as alterações, ele deve tomar a decisão de encerrar o *Pull Request* ou solicitar revisões nas alterações sugeridas. Se o Mantenedor optar pela não revisão, o Pull Request é fechado. Se ele decidir por uma revisão, o Colaborador é notificado com as alterações descritas pelo Mantenedor para que o seu código seja aceito.

O *deployment* da *WebSTAMP* é feito via *workflow* de entrega contínua (parte inferior esquerda na Figura 5.1) por meio do *Heroku*, utilizando a técnica de *Blue-Green Deployment* [YANG et al. 2018]. Ao se usar a técnica *Blue-Green Deployment*, requisições do *Cliente*, usuário final de serviços da *WebSTAMP*, são redirecionadas temporariamente para o ambiente de testes (*green*), até que as novas funcionalidades sejam implementadas no ambiente de produção real (*blue*), isso faz com que o ambiente não fique indisponível para o cliente enquanto novas atualizações são implantadas.

Dessa forma, as alterações aceitas pelo *Mantenedor* são adicionadas no repositório central do *GitHub* e é feito o *deployment* por meio do serviço *Preboot* [COUTERMARSH 2014] do *Heroku*, chamado de implantação verde (parte inferior da figura). Após a conclusão da implantação, existem duas versões simultâneas da aplicação *online*, nomeadas *blue* e *green* respectivamente.

Quando um usuário acessa a *WebSTAMP*, uma instância do *Dyno* [MIDDLETON e SCHNEEMAN 2013] é criada para atender as requisições *web* realizadas. O serviço Dyno é uma abstração criada pelo *Heroku* para escalar aplicações *web* em containers

Linux [SEO et al. 2014] que no escopo da *WebSTAMP* armazenam os *docker containers* [TURNBULL 2014]. Os *docker containers* redirecionam as requisições dos clientes para o sistema implantado no ambiente de produção (*Blue* ou *Green* na parte inferior esquerda da Figura 5.1).

6. Discussões

A proposta de mudança da *WebSTAMP* para uma licença *open source* torna necessária a escolha de um *workflow* para colaboração no *GitHub*. Conforme discutido na Seção 1, os seguintes *workflows* foram analisados: *Centralized Workflow*, *Feature Branch Workflow*, *Gitflow Workflow*, e *Forking Workflow*.

Dentre os *workflows* analisados, para não membros do time de desenvolvimento da *WebSTAMP*, foi escolhido para utilização no desenvolvimento o *Forking Workflow*, pois, é o mais recomendado para projetos *open source* [STĂNCIULESCU et al. 2015], permitindo que o mantenedor possa aceitar apenas modificações que realmente agregam valor no projeto. Já para membros da *WebSTAMP*, escolhemos o *Feature Branch Workflow*, tendo em vista que neste *workflow* os desenvolvedores possuem acesso direto ao repositório principal da *WebSTAMP*, sem necessidade de fazer um *fork*.

Existem diversas ferramentas *(Jenkins, Travis CI, GitHub Actions)* para gerenciamento de *pipelines CI/CD (continuous integration/continuous delivery)*, com características e aplicabilidades diferentes que influenciam na escolha para cada projeto. Para a *WebSTAMP* o principal objetivo foi utilizar ferramentas que facilitassem a integração com o *GitHub*, que disponibilizassem recursos de *cloud* e que ao mesmo tempo fosse possível construir *workflows* alinhados com os conceitos de *DevOps.*

Seguindo os objetivos do *pipeline CI/CD* da *WebSTAMP*, realizamos um estudo comparativo entre *Jenkins*, *Travis CI* e *GitHub Actions*. Todas as ferramentas citadas são adequadas, com planos gratuitos e com diferenciais pontuais, descritos na sequência, que foram cruciais para a escolha de acordo com o escopo atual do projeto.

Jenkins é uma ferramenta gratuita, *open source,* possui integração com o *GitHub* e exige uma configuração elaborada que inclui servidores próprios. Em contrapartida o *Travis CI*, também possui integração com o *GitHub* e é baseado em nuvem. Ele é gratuito para a maioria dos casos, com exceção de empresas que para utilizá-lo precisam assinar o plano empresarial.

Apesar de todos os diferenciais do *Jenkins* e do *Travis CI*, para o escopo atual da *WebSTAMP* optamos pelo *GitHub Actions*, pois, como é uma ferramenta distribuída pelo próprio *GitHub*, sua integração com o *GitHub* é praticamente instantânea, além de ser gerenciável e fácil de implantar por meio de arquivos de configuração *YAML* [BEN-KIKI et al. 2009].

Além das ferramentas citadas, há diversas plataformas (*AWS*, *Azure*, *Google Cloud*) de computação em nuvem (*clouds*) [TAURION 2009] que oferecem *SaaS (Software as Service)* [KULKARNI 2012] que providenciam a infraestrutura desejada de forma

automática e rápida. No contexto de *CI/CD* estes serviços podem ser escolhidos para a construção de um *pipeline* fim-a-fim de uma aplicação.

Três serviços de nuvem bastante populares para construção de um *pipeline* são: *AWS CodePipeline* [DALBHANJAN 2015], *Azure DevOps* [ROSSBERG 2019] e *Cloud Build* [KRISHNAN and GONZALEZ 2015]. Todos disponibilizam uma *interface* que permite configurar cada etapa de um *pipeline CI/CD* da aplicação, incluindo etapas que vão desde o *build* até o seu *deployment*. No geral, estes serviços apresentam propósitos similares e atendem bem os seus objetivos.

Estes serviços são alternativas para as plataformas anteriormente citadas: *Jenkins*, *Travis CI* e *Github Actions*. A configuração da infraestrutura necessária para construção do *pipeline* é simplificada nesses serviços disponibilizados pelas *clouds.* Por conta disso, há custo financeiro relacionado, um fator para que o *Github Actions* (gratuito) tenha sido a ferramenta escolhida para a implementação do *pipeline CI/CD* na *WebSTAMP*.

Quando realizado o *deployment* de um sistema, para que novas funcionalidades sejam adicionadas em ambiente de produção, o sistema em produção fica inativo enquanto é atualizado. Para contornar este problema foram analisadas técnicas de *deployment* (*Blue-Green* e *Canary Deployment*) alinhadas com a prática de entrega contínua em *DevOps - CD* (Continuous *Delivery)*.

O *Blue-Green Deployment* consiste na ideia de dois ambientes. Quando o ambiente de produção é atualizado, os clientes são direcionados momentaneamente para um segundo ambiente idêntico ao ambiente de produção antes das atualizações. Dessa forma, o sistema não fica inativo em nenhum momento visto que os clientes são direcionados para um outro local enquanto o ambiente de produção está sendo atualizado.

O *Canary Deployment* [RUDRABHATLA 2020] também visa impedir com que o sistema fique inativo durante o *deployment*. Porém, ao contrário do *Blue-Green Deployment*, não possui dois ambientes, mas sim uma subdivisão de um mesmo ambiente, consistindo na ideia deste mesmo ambiente ser atualizado parcialmente. Logo, apenas algumas partes do ambiente de produção ficarão inativas e os clientes poderão acessar as partes ativas conforme o ambiente é atualizado.

O *Blue-Green deployment* pode ser implementado por meio do recurso de *Preboot* do *Heroku*, porém o mesmo possui algumas limitações quando comparado a um serviço específico de *Blue-Green deployment* ofertado em *clouds*. Podem ocorrer situações em que há duas versões da base de código rodando lado-a-lado, por exemplo, os *Dynos* da *web* (contêineres do *Heroku* responsáveis por receber as requisições de clientes *web*) podem estar executando uma versão do código diferente do que os *Dynos* de execução (contêineres do *Heroku* responsáveis por executar as requisições recebidas pelos *Dynos* da *web*) e isso pode levar a problemas de compatibilidade.

Embora o *pipeline* proposto na Figura 5.1 inclua a técnica *Blue-Green deployment*, no contexto e momento atuais da *WebSTAMP,* o *deployment* em um ambiente de produção comum (sem uso de *Blue-Green* ou *Canary Deployment*) é a melhor opção. Dentre as razões destacam-se, o número ainda reduzido de usuários e a existência de custo financeiro necessário para implementar as técnicas de *Blue-Green* e *Canary Deployment*.

7. Conclusões e Trabalhos Futuros

A definição de um *pipeline CI/CD* para uma ferramenta não é uma tarefa simples, visto que para defini-lo é necessário analisar cautelosamente cada etapa do ciclo de desenvolvimento da aplicação. Para isso é necessário estudar e escolher conceitos, ferramentas e serviços que melhor se adequam no contexto da aplicação para serem aderidos ao *pipeline.*

O serviço *Preboot* realiza configurações automáticas para o *deployment* da aplicação, não sendo necessário a utilização de uma das práticas de *DevOps* - IaC (*Infrastructure as Code*) [MORRIS 2016], que tem como objetivo automatizar o provisionamento da infraestrutura de recursos de *hardware* e *software*. Dessa forma, não há a necessidade do uso de ferramentas como a *Terraform* [BRIKMAN 2019].

Conforme discutido na Seção 6, o serviço de *Preboot* do *Heroku* apresenta limitações. Desta forma, ao invés de utilizar este serviço, uma das alternativas para se aderir futuramente, é realizar o *deployment* utilizando serviços de *clouds,* como *AWS CodePipeline*, *Azure DevOps* e *Cloud Build*.

Com os resultados obtidos, podemos concluir que a implementação do *pipeline* proposto neste trabalho automatiza a sequência de passos para integração e entrega de novas funcionalidades, facilitando a participação de novos colaboradores na evolução da *WebSTAMP*. Além disso, a definição de uma licença *open source* viabiliza a distribuição da aplicação alinhado com os requisitos (não comercialização, divulgação do código-fonte, entre outros) estipulados pelos mantenedores da WebSTAMP.

Referências

ABILDSKOV, J. (2020). Collaboration in Git. Practical Git. Apress, Berkeley, CA. 83-106.

BEER, B. (2018). Introducing GitHub: A non-technical guide. O'Reilly Media, Inc.

BELLER, M., GOUSIOS, G., ZAIDMAN, A. (2017). Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), p. 447-450.

BELMONT, J. (2018). Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI. Packt Publishing Ltd.

BEN-KIKI, O., EVANS, C., INGERSON, B. (2009). Yaml ain't markup language (yaml™) version 1.1. Working Draft 2008-05 11.

BRIKMAN, Y. (2019). Terraform: up & running: writing infrastructure as code. O'Reilly Media.

BUNYAKIATI, P., SAMMAPUN, U. (2019). Open-Source Projects and their Collaborative Development Workflows. ArXiv, https://arxiv.org/abs/2103.12224, September.

COMMONS, L. C. (2009). LE LICENZE CREATIVE COMMONS.

COPELAND, M., SOH, J., PUCA, A. (2015). Microsoft Azure. New York, NY, USA:: Apress.

COUTERMARSH, M., (2014). Heroku Cookbook. Packt Publishing Ltd.

DALBHANJAN, P. (2015). Overview of deployment options on AWS. Amazon Whitepapers.

HUNTER, T., PORTER, S. (2018). Google Cloud Platform for developers: build highly scalable cloud solutions with the power of Google Cloud Platform. Packt Publishing Ltd.

KAPITSAKI, G. M.; TSELIKAS, N. D., FOUKARAKIS, I. E. (2015). An insight into license tools for open source software systems. Journal of Systems and Software, v.102, p. 72-87, 2015.

KHAN, M. O., JUMANI, A. K., FARHAN, W. A. (2020). Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud. Indian Journal of Science and Technology, v.13, n.05, p.552-575.

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.

KINSMAN, T., WESSEL, M., GEROSA, M. A., TREUDE, C. (2021). How Do Software Developers Use GitHub Actions to Automate Their Workflows?. IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), p.420-431, https://ieeexplore.ieee.org/document/9463074, September.

KRISHNAN, K. (2019). Building Big Data Applications. Krish Krishnan.

KRISHNAN, S. P. T., GONZALEZ, J. L.. (2015). Building your next big thing with Google Cloud Platform: A guide for developers and enterprise architects. USA: Apress.

KULKARNI, G. (2012). Cloud computing-software as service. International Journal of Cloud Computing and Services Science 1.1.

LEVESON, N. G., THOMAS, J. P. (2018). STPA handbook. Cambridge, MA, USA.

LIN, Y., KO, T., CHUANG, T., LIN, K. (2006). Open Source Licenses and the Creative Commons Framework: License Selection and Comparison. J. Inf. Sci. Eng, v.22, p.1-17.

LOPES, M. D. (2020). Aplicação da integração contínua em um sistema de auxílio à tomada de decisão. 50f. Dissertação (Mestrado em Computação Aplicada) - Universidade de Passo Fundo, Passo Fundo, RS, 2020.

MIDDLETON, N., SCHNEEMAN, R. (2013). Heroku: up and running: effortless application deployment and scaling. O'Reilly Media, Inc.

MOHAMMAD, S. M. (2016). Continuous Integration and Automation. International Journal of Creative Research Thoughts (IJCRT), v.4, n.3, p.938-945, https://ssrn.com/abstract=3655567, September.

MORRIS, D., VOUTSINAS, S., HAMBLY, N. C. (2017). Mann, Robert (2017). Use of Docker for deployment and testing of astronomy software. Astronomy and computing, v.20, p. 105-119.

MORRIS, K. (2016). Infrastructure as code: managing servers in the cloud. " O'Reilly Media, Inc.".

PIPINELLIS, A. (2018). GitHub Essentials: Unleash the power of collaborative development workflows using GitHub. Packt Publishing Ltd.

ROSSBERG, J. (2019). An overview of Azure devops. Agile Project Management with Azure DevOps.

RUDRABHATLA, C. (2020). Comparison of zero downtime based deployment techniques in public cloud infrastructure. 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC). IEEE.

STĂNCIULESCU, Ş., SCHULZE, S., WĄSOWSKI, A. Forked and integrated variants in an open-source firmware project. IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015.

SEO, K. T., HWANG, H. S., MOON, I. Y., KWON, O. Y., KIM, B. J. (2014). Performance comparison analysis of linux container and virtual machine for building cloud. Advanced Science and Technology Letters, v. 66, n. 105-111, p. 2.

SMART, J. F. (2011). Jenkins: The Definitive Guide: Continuous Integration for the Masses. O'Reilly Media, Inc.

SOCHAT, V. (2018). Containershare: Open Source Registry to build, test, deploy with CircleCI. Journal of Open Source Software.

SOUZA, F. G., PEREIRA, D. P., PAGLIARES, R. M., NADJM-TEHRANI, S., HIRATA, C. M. (2019). WebSTAMP: a Web Application for STPA & STPA-Sec. In MATEC Web of Conferences, v.273, p.2010.

TAN, X., ZHOU, M., SUN, Z. (2020). A first look at good first issues on GitHub. Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

TAURION, C. (2009). Cloud computing-computação em nuvem. Brasport.

TURNBULL, J. (2014). The Docker Book: Containerization is the new virtualization. James Turnbull.

VON KROGH, G. Open-source software development. MIT Sloan Management Review, 2003.

WITTIG, M., WITTIG, A. (2018). Amazon web services in action. Simon and Schuster.

YANG, B., SAILER, A., JAIN, S., TOMALA-REYES, A., SINGH, M., RAMNATH, A. (2018). Service discovery based blue-green deployment technique in cloud native environments. 2018 IEEE International Conference on Services Computing (SCC).

ZHU, L., BASS, L., CHAMPLIN-SCHARFF, G. (2016). DevOps and Its Practices. IEEE Software, v.33, n.3, p.32-34, 2016, https://ieeexplore.ieee.org/abstract/document/7458765, September.