ANÁLISE E OTIMIZAÇÃO DE UM FLUXO AUTOMATIZADO PARA IMPLEMENTAÇÃO DE *SOFTWARE*

Christian Gabriel Bernini Alves da Silva

christian.silva.15@fatec.sp.gov.br

Prof. Me. Mario Marques

mario.marques@fatec.sp.gov.br

Fatec Itapetininga - SP

RESUMO: Através de um estudo de caso em uma empresa do ramo financeiro e, a partir do levantamento do fluxo de atuação da equipe responsável por colaborar com a transformação digital da instituição, bem como da documentação do processo atual de implementação de *software*, seus requisitos são compreendidos e identificados e um fluxo alternativo é proposto, buscando alinhar-se às melhores práticas de mercado inspiradas na literatura de referência. A adoção de uma ferramenta de gerência de configuração é analisada, junto aos benefícios e prejuízos da automatização nas atividades diárias da equipe.

Palavras-chave: Automatização, Desenvolvimento de *software*, DevOps.

ABSTRACT: Through a case study in a financial institution, by mapping the workflow of the team supporting its digital transformation, the current software delivery process was documented, its requirements gathered and understood, and an alternative flow is proposed, seeking alignment with best practices in the market and inspired by the referenced literature. The adoption of a configuration management tool is analyzed, with the benefits and downsides of automation for the team's daily activities.

Keywords: Automation, Software Development, DevOps.

1 INTRODUÇÃO

O Manifesto Ágil em seu primeiro princípio, enuncia: "Nossa maior prioridade é satisfazer o cliente por meio da entrega adiantada e contínua de *software* de valor". (BECK; et al, 2001). Entrega essa que, para ser contínua, necessita ser consistente, controlada e reprodutível. (HUMBLE; FARLEY, 2011)

Tal prática busca harmonizar os ganhos, alcançados com o desenvolvimento de software através de modelos Ágeis e versáteis, às necessidades da infraestrutura da empresa e da área de operações de TI, utilizando-se de gerência de configuração e automatização, entre outras ferramentas e técnicas, com o objetivo de efetuar uma

implementação de *software* eficiente e de baixo risco.

Através de uma compreensão das necessidades, em reuniões e entrevistas realizadas com os clientes, aqui compreendidos como integrantes de *squads*[1] ágeis, iniciou-se a construção de ferramentas que tornassem transparente o provisionamento de infraestrutura e seu acesso por tais clientes.

Entende-se que ferramentas são aceleradores de cultura e não a cultura em si, de forma que a utilização e a adaptação das mesmas são contínuas, e visam atender as necessidades dos reais envolvidos com o processo de implementação de *software* desde o ambiente de desenvolvimento até o de produção: os colaboradores.

Todos os modelos aqui discutidos partem da premissa de que a Integração Contínua (CI) no projeto em questão já existe, porém não avaliaremos seu grau de maturidade. Durante o projeto foi percebida uma carência em como tais artefatos, gerados pelo processo de CI, são promovidos nos ambientes e customizados de acordo com necessidades específicas da aplicação.

A infraestrutura suportada pelas automatizações consiste de uma arquitetura de micro serviços[2], de tecnologias diversas, implementada em uma nuvem privada[3] através da solução OpenShift, do fornecedor RedHat. (REDHAT, 2017)

Os serviços foram desenhados e implementados originalmente para terem seu acesso direto via API aos desenvolvedores após a fase de CI, ou por interface gráfica através da ferramenta Rundeck, responsável por disponibilizar tais acessos e concentrar as automatizações para a implementação e manutenção da infraestrutura suportada.

Este artigo busca retratar, analisando um cenário real, como a utilização de práticas ágeis, alinhadas à cultura DevOps, fornecem a velocidade desejada na implementação de *software* em produção, demonstrar a evolução das ferramentas e apresentar um fluxo alternativo ao atual, como melhoria a ser oferecida ao cliente, cabendo à liderança da empresa a autorização para sua implementação.

2 METODOLOGIA

Através de entrevista com os recursos da Equipe de Engenharia DevOps, e análise dos casos de uso já existentes na ferramenta Rundeck, foi decidido, ao longo das reuniões de equipe, quais processos poderiam ser otimizados ou automatizados. Uma vez definidos os processos, foram levantados os requisitos funcionais e não funcionais para a realização da tarefa.

[1] Equipes compactas e multidisciplinares.

[2] Arquitetura de *software* onde uma aplicação é construída a partir de vários pequenos serviços que implementam uma funcionalidade de negócio, cada um isolado em seu processo e comunicando-se geralmente por HTTP ou através de uma API.

[3] Modelo onde os recursos computacionais são contratados como serviços tendo caraterísticas como elasticidade, segurança e escalabilidade garantidas por padrão. Por ser privada, é implementada pela empresa que a possui.

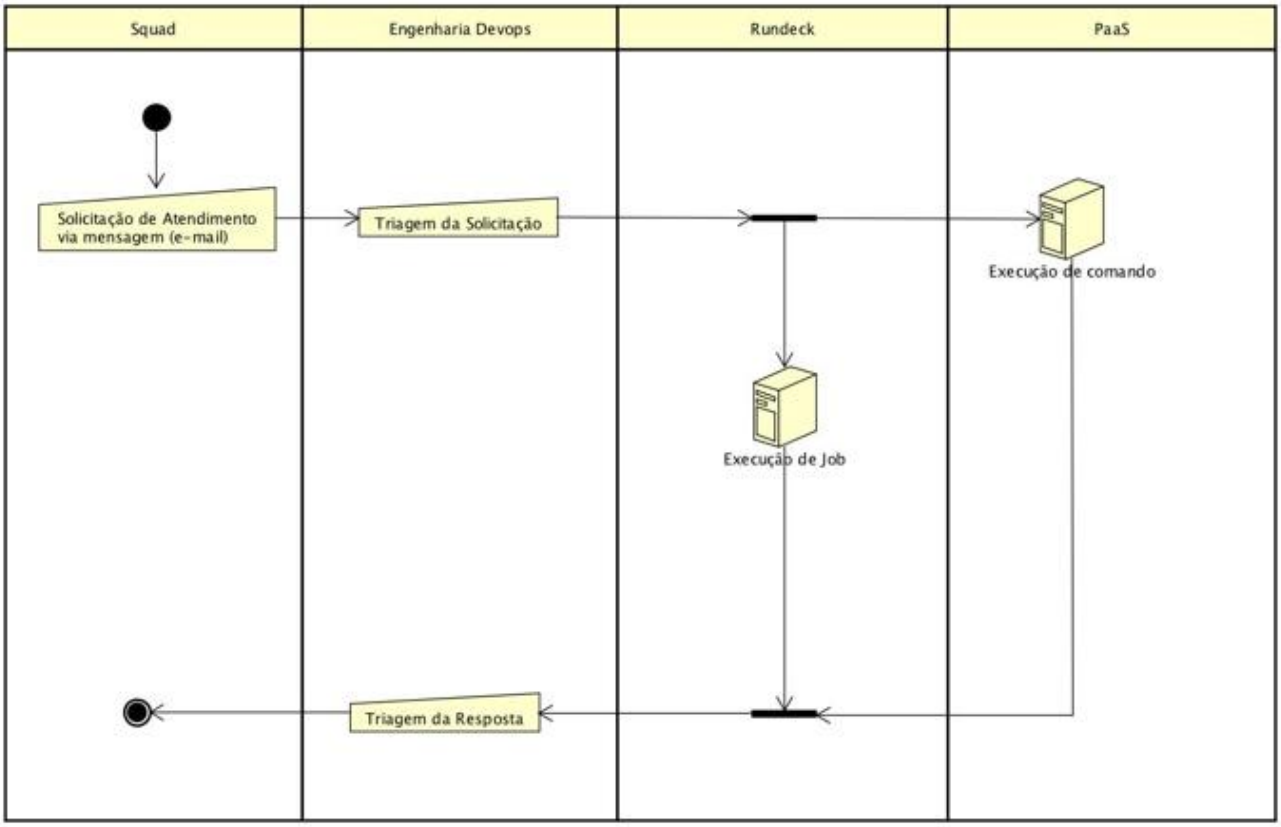

Figura 1 Fluxo de atendimento squads x equipe Engenharia DevOps Fonte: Próprio autor

A proposta apresentada busca seguir a boa prática, como descrito por Humble e Farley (2011), de manter as configurações de cada ambiente separadas, em um controle de versão, com o arquivo correto sendo recuperado através de uma variável de ambiente passada ao *script* de implementação, indicando qual o entorno deverá ser utilizado (desenvolvimento, homologação e produção).

Para todas as modelagens, foi utilizada a UML. Inicialmente foi elaborado um fluxograma da área (Figura 1) para compreender a melhor maneira de implementar a automatização dos requisitos.

3 REFERENCIAL TEÓRICO

Os conceitos aqui definidos, quando utilizados, serão referenciados por seu acrônimo, à maneira como se faz no meio profissional.

3.1 DEVOPS

Cunhado em 2009, o termo DevOps descreve uma abordagem onde as equipes de desenvolvimento e operações têm metas compartilhadas e utilizam-se das práticas de CI de forma a facilitar a implementação do *software* (KIM; et al, 2016). Hoje, em um modelo mais maduro, compreende-se por DevOps um conjunto de práticas e cultura sobre quatro pilares: colaboração, afinidade, ferramentas (automatização), tudo isso em escala. (DAVIS; DANIELS, 2016)

A cultura DevOps busca delegar a rotinas automatizadas todo o trabalho repetitivo durante o ciclo de vida de uma aplicação, buscando prover mais autonomia e agilidade aos membros de uma equipe, também conhecida por *squad*, composta por não mais que dez integrantes, de maneira a evitar o aumento de complexidade de um *software* como previsto pela Lei de Conway, segundo Kim; et al. (2016 *apud* CONWAY, 1968), onde é exposto que uma organização, ao desenhar um sistema, irá produzir na estrutura de tal desenho uma cópia da estrutura de comunicação de tal organização.

3.2 INTEGRAÇÃO CONTÍNUA (CI)

É o processo de integrar o código na *branch*[4] principal, frequentemente, durante o dia. Uma série de testes automatizados é disparada contra a nova versão assim que a mesma passa a fazer parte da *branch*. Através de entregas pequenas, porém constantes, torna-se muito mais fácil de encontrar, resolver e minimizar regressões causadas por alguma mudança no código. (DAVIS; DANIELS, 2016)

Caso uma *build*[5] quebre após algum código entrar na *branch*, o *feedback* é imediato e a solução tende a ser mais rápida. Uma ferramenta amplamente utilizada para CI e utilizada no projeto, disponibilizada como *software* livre, é o Jenkins.

[4] Ramificação em um sistema de controle de versões, a *branch mestre* ou principal é onde se encontra o código apto à produção.
[5] Processo pelo qual um novo artefato da aplicação é gerado.

3.3 ENTREGA CONTÍNUA (CD)

Também conhecida por CD, ou *Continuous Delivery*, consiste na utilização de *pipelines*[6] de implantação, com alto grau de automatização tanto de testes como de entrega e uso adequado de gerência de configuração, possibilitando entregas apertando apenas um botão ou dependente de apenas um estímulo, quando necessário, em qualquer ambiente, seja ele teste, desenvolvimento ou produção). (HUMBLE; FARLEY, 2011)

O CD é visto como uma evolução natural do CI, de forma a abranger não só o processo de desenvolvimento, mas garantir uma visão holística de todo o ciclo de vida do *software*. A ferramenta escolhida para centralizar as automatizações é o Rundeck, também disponível como *software* livre.

4 RESULTADOS E DISCUSSÃO

A área de Engenharia DevOps, consiste em uma equipe responsável por otimizar, viabilizar e propor melhores práticas de CD, além de disseminar a cultura DevOps no cliente, uma multinacional do ramo financeiro.

Além disso a equipe se tornou responsável, após o início do projeto, pela configuração e provisionamento dos *pipelines*

[6] Assim chamados pois são análogos aos fluxos de processamento paralelo observados na arquitetura de processadores, que dividem as tarefas de maneira a otimizar o tempo de execução.

de implementação, criação dos ambientes na solução de plataforma como serviço (PaaS), o *software* OpenShift, em interação direta com os *squads* de desenvolvimento e sua liderança imediata.

O estudo de caso também buscou explorar o aspecto humano do que é hoje chamado DevOps e, erroneamente, interpretado como a aplicação de, somente, automatização e novas tecnologias. Davis e Daniels (2016) entendem que nem tudo pode ou sequer deve ser automatizado e, frequentemente, quanto mais complexas se tornam as automatizações, maior a necessidade de intervenção humana para manter ou analisar problemas percebidos nas mesmas. Deve-se levar em conta, também, a necessidade de evitar o *deskilling*, processo que se dá quando há um número tão grande de automatizações em vigor que as pessoas podem esquecer como devem realizar tal procedimento e sua experiência em tal área diminui com o tempo. (DAVIS; DANIELS, 2016)

Através da utilização do *software* Rundeck, a equipe de Engenharia DevOps pôde disponibilizar, via API e pela própria ferramenta através de uma interface *web*, um *pipeline* de implementação que garantisse aos desenvolvedores autonomia nos ambientes de Desenvolvimento e Homologação enquanto a equipe de Operações pudesse gerir o ambiente de produção. As automatizações implementadas via Rundeck são (Figura 2):

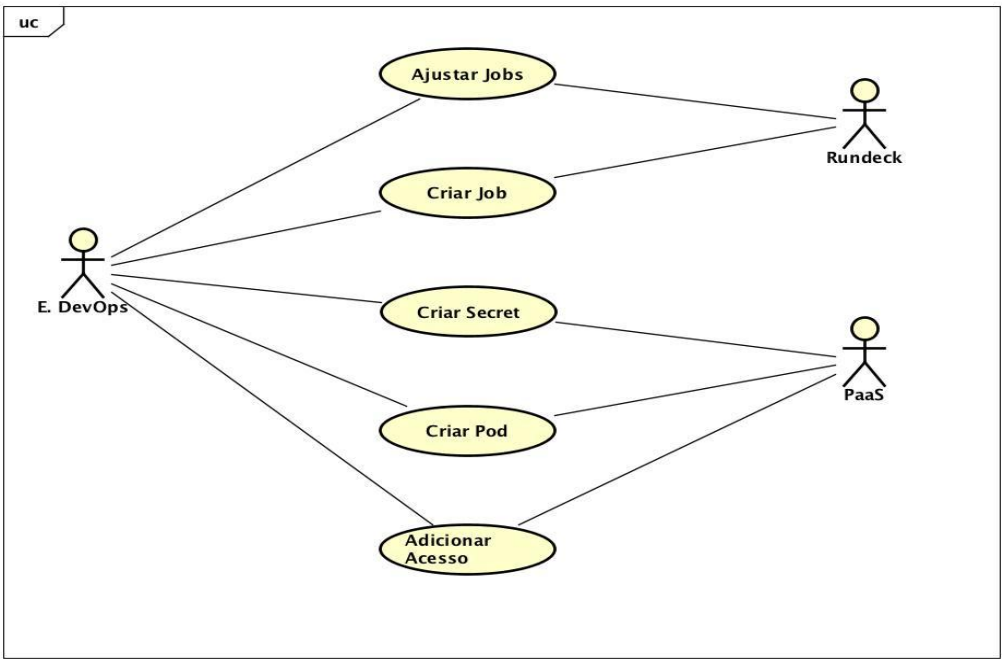

Figura 2: Processos automatizados via Rundeck Fonte: Próprio autor

1. Criação de *secrets*[7], para configuração de usuário/senha na plataforma Openshift;
2. Criação automática dos *jobs*[8] de implementação e contingência de aplicações;
3. Liberação de acesso na plataforma Openshift.

Durante a interação com os *squads* de desenvolvimento, foi percebido que a criação e customização de aplicações nos projetos na plataforma OpenShift eram solicitações constantes à equipe de Engenharia DevOps e para isso foram entregues duas novas automatizações:

4. Criador de aplicações na plataforma OpenShift, já também criando os *jobs* de implementação e contingência;
5. Atualizador de *jobs* na ferramenta Rundeck.

A entrega mais recente, o Atualizador de *jobs*, surgiu da necessidade de adequar os *jobs* já existentes às constantes evoluções do ambiente, visto que a ferramenta em si não fornece uma opção de edição em lote com customização suficiente. É válido pontuar que todas as automatizações dependem de alguma entrada de dados, seja ela manual ou oriunda da plataforma de CI.

Em tempo de projeto foi aprendido que, através de estudo da literatura referenciada e das boas práticas de mercado descritas na mesma, a utilização mais rigorosa da disciplina de gerência de configuração diminuiria os acionamentos à equipe de Engenharia DevOps, pelo fato de garantir que toda informação relacionada à área de CD estivesse sob um controle de versão e mudanças nos processos fossem automaticamente documentadas e validadas. Tal deficiência já causava impacto na equipe devido à quantidade de acionamentos feitos a cada nova implementação, quando era percebido que, uma vez que a configuração do ambiente não estava persistida em controle de versão assim como o código da aplicação, não havia maneira fácil e ou transparente de validar que o ambiente funcionaria até que fosse realizada a referida implementação.

[7] Arquivos com conteúdo cifrado, uma funcionalidade do OpenShift para disponibilizar senhas e configurações sensíveis nos seus ambientes.

[8] Sequências de instruções ou comandos reunidos, geralmente em um arquivo de *script*, para um propósito específico, como por exemplo realizar um backup.

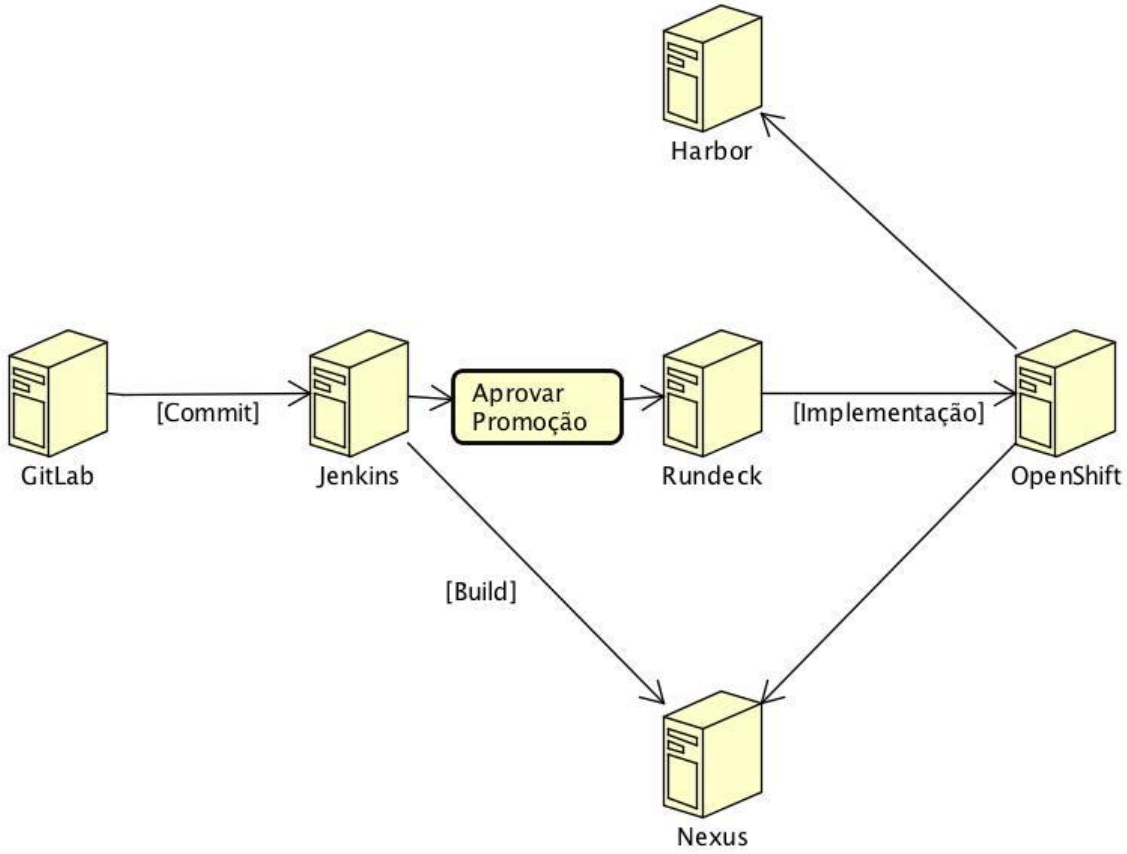

Figura 3: Fluxo Atual do Sistema Fonte: Próprio autor

Somente no mês de agosto de 2017, os *jobs* implementados na ferramenta Rundeck foram executados mais de 10 mil vezes (mais de 20 mil se consideradas as execuções automáticas, valor extraído diretamente da função de relatório da ferramenta), permitindo dezenas de *squads* entregar *software* em produção sem necessidade de intervenção manual ou acionamento de alguma equipe externa ao processo, exceto a validação humana para a promoção em produção. Toda essa infraestrutura (Figura 3) é hoje administrada e provisionada por uma equipe de 5 (cinco) engenheiros de sistemas.

Um cenário alternativo (Figura 4) foi proposto, de maneira a aumentar o nível de automatização e reduzir o tempo de entrega da infraestrutura. O diagrama apresenta um fluxo onde toda a gerência de configuração passa a ser automática, através da customização na ferramenta Ansible e o controle de promoção passa a ser diretamente pela plataforma Jenkins de CI. A possibilidade de configuração automática através de um ponto central dá a garantia e a visibilidade de que as necessidades de segurança estejam presentes e sejam facilmente auditáveis, além de minimizar o erro humano durante a configuração e ou provisionamento do ambiente.

O modelo busca respeitar o conceito de infraestrutura como código (FOWLER, 2016), garantindo que toda a configuração do ambiente é definida e persistida, em código, num sistema de controle de versão, tratada com o mesmo cuidado que o código da aplicação, e de infraestrutura imutável (MORRIS, 2013), a qual consiste em provisionar servidores que uma vez instanciados não sofrem nenhuma alteração visto que são atômicos, a partir da soma de uma imagem base, configuração automaticamente gerenciada e dados, amarrados antes da subida do mesmo, de maneira a garantir maior confiabilidade no processo de entrega e diminuir o atrito entre as áreas envolvidas uma vez que qualquer engenheiro do *squad* passa a ter mais autonomia sobre a infraestrutura do seu projeto e o acionamento direto de recursos da equipe de Engenharia DevOps deixa de acontecer, visto que uma plataforma de autosserviço passa a ser a principal interface com os clientes internos.

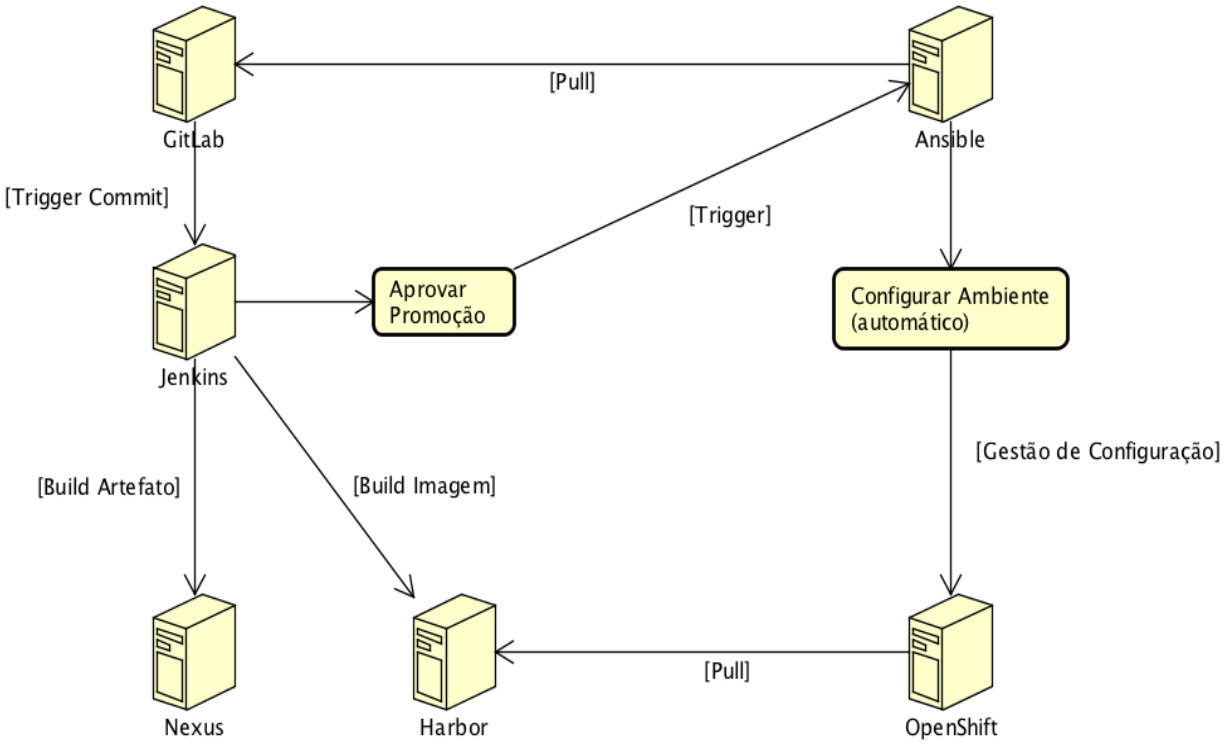

Figura 4: Fluxo alternativo proposto Fonte: Próprio autor

5 CONSIDERAÇÕES FINAIS

Implementar *pipelines* para requisitos não funcionais se mostrou essencial para garantir o cumprimento do nível de serviço, SLA, com o cliente além de permitir que a equipe de Engenharia DevOps pudesse desenhar, propor e implementar uma solução mais robusta de CD e colaborar na transformação digital desejada pelo cliente, alinhado à cultura DevOps.

Observou-se que, apesar da utilização de ferramentas comuns a outras empresas, as soluções de CD são exclusivas e não um modelo exato de como tudo deve ser feito. As práticas que sustentam essa abordagem são agnósticas à tecnologia, de forma que a escolha das mesmas se alinha aos objetivos de negócio, familiaridade com o uso e modelo de suporte com o fornecedor.

Os objetivos com o fluxo alternativo proposto são o de centralizar a gerência de configuração e automatizá-la de maneira a garantir cobertura de testes sobre a mesma e eliminar a ocorrência de incidentes nos ambientes produtivos em razão da ausência de configuração ou de falta de validação da mesma.

REFERÊNCIAS BIBLIOGRÁFICAS

DAVIS, J.; DANIEL, K. Efffective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. 1.ed. California: O'REILLY, 2016. 410p.

HUMBLE, J.; FARLEY, D. Entrega Contínua: como entregar *software* de forma rápida e confiável. 1.ed. Rio Grande do Sul: BOOKMAN, 2014. 464p.

KIM, G. et al. The DevOps Handbook: How to create world-class agility, reliability, and security in technology organizations. Oregon: IT REVOLUTION PRESS, 2016. 480p.

BECK, K. et al. Manifesto for agile software development. Disponível em: <https://agilemanifesto.org> acesso em setembro de 2017.

MORRIS, K. ImmutableServer. Disponível em: < https://martinfowler.com/bliki/ImmutableServer.html> acesso em novembro de 2017.

FOWLER, M. InfrastructureAsCode. Disponível em: < https://martinfowler.com/bliki/InfrastructureAsCode.html> acesso em novembro de 2017.

CONWAY, M. How Do Committees Invent?. Disponível em: <http://www.melconway.com/Home/pdf/committees.pdf> acesso em novembro de 2017.

RED HAT, OpenShift Blog. How to Avoid Cloud Vendor Lock-In when Evaluatin PaaS. Disponível em: <https://blog.openshift.com/how-to-avoid-cloud-vendor-lock-in-when-evaluating-a-paas/> acesso em outubro de 2017.