UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE E INOVAÇÃO

MARIEL MADLENE DOS SANTOS

Mapeamento do Processo e Melhorias do Pipeline de Entrega Contínua de um Software Embarcado

Monografia de Conclusão de Curso apresentada como requisito parcial para a obtenção do grau de Especialista em Engenharia de Software e Inovação

Orientador: Prof. Dra. Ingrid Nunes

Porto Alegre
2021

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Santos, Mariel Madlene dos

Mapeamento do Processo e Melhorias do Pipeline de Entrega Contínua de um Software Embarcado / Mariel Madlene dos Santos. – Porto Alegre: PPGC da UFRGS, 2021.

44 f.: il.

Monografia (especialização) – Universidade Federal do Rio Grande do Sul. Curso de Especialização em Engenharia de Software e Inovação, Porto Alegre, BR–RS, 2021. Orientador: Ingrid Nunes.

1. DevOps. 2. Entrega Contínua. 3. Pipeline. 4. Software Embarcado. I. Nunes, Ingrid. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Profª. Patricia Pranke
Pró-Reitor de Pós-Graduação: Prof. Júlio Otávio Jardim Barcellos
Diretora do Instituto de Informática: Profª. Carla Maria Dal Sasso Freitas
Coordenadora do Curso: Profª. Karin Becker
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Primeiramente, agradeço à Deus pela minha saúde e por ter concluído mais uma etapa importante em minha vida em um momento atípico que todos nós estamos vivendo devido a pandemia do novo coronavírus.

Agradeço à minha família e aos meus amigos por todo apoio, incentivo e compreensão nos momentos em que precisei me ausentar.

RESUMO

Cada vez mais as organizações visam realizar uma entrega de software contínua e eficaz a fim de entregar valores aos seus usuários. No entanto, atingir esse objetivo requer o uso de práticas que acelerem o desenvolvimento e mudanças culturais dentro das organizações que foquem não somente na entrega rápida, mas também na qualidade do software que está sendo entregue. Para isso, a adoção do movimento *development and operations* (DevOps) tem se destacado e se tornado indispensável no desenvolvimento de software. DevOps é um movimento cultural que através de alguns princípios, práticas e ferramentas, visa auxiliar organizações a entregarem software com maior confiança e rapidez. O presente trabalho visa apresentar como as práticas de DevOps podem contribuir para que haja uma entrega de software contínua e confiável através do mapeamento do processo do pipeline de entrega contínua de um projeto de software embarcado de uma empresa de Telecomunicações. Com base no processo mapeado, são sugeridas propostas de melhorias para o pipeline, o que inclui uma melhoria contínua que norteia as práticas de DevOps.

Palavras-chave: DevOps. Entrega Contínua. Pipeline. Software Embarcado.

Process Mapping and Continuous Delivery Pipeline Improvements for Embedded Software

ABSTRACT

More and more organizations aim to carry out continuous and effective software delivery to deliver value to their users. However, achieving this goal requires using practices that accelerate development and cultural change within organizations that focus not only on rapid delivery but also on the quality of the software being delivered. For this, the adoption of the *development and operations* (DevOps) movement has become prominent and indispensable in software development. DevOps is a cultural movement that, through some principles, practices, and tools, aims to help organizations deliver software with greater confidence and speed. This paper aims to present how DevOps practices can contribute to continuous and reliable software delivery by mapping the continuous delivery pipeline process of an embedded software project of a Telecommunications company. Based on the mapped process, proposals for improvements to the pipeline are suggested, which include continuous improvement that guides DevOps practices.

Keywords: DevOps, Continuous Delivery, Pipeline, Embedded Software.

LISTA DE ABREVIATURAS E SIGLAS

ARM	Advanced RISC Machine
CLI	Command-line Interface
CD	Continuous Delivery
CI	Continuous Integration
DEV	Desenvolvimento
XP	Extreme Programming
KVM	Kernel-based Virtual Machine
OPS	Operações
P&D	Pesquisa e Desenvolvimento
USB	Universal Serial Bus
VM	Virtual Machine
VMM	Virtual Machine Monitor

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A entrega de software sempre foi vista como um desafio na maioria das organizações, independente da metodologia utilizada para a entrega de software (HUMBLE; FARLEY, 2014). Existem problemas tanto no tempo em que se leva para colocar uma implementação em produção quanto nos inúmeros processos manuais que precisam ser seguidos para que tal implementação aconteça. De certa forma, isso acaba por gerar insegurança na equipe responsável devido ao risco de erro humano. Além disso, há problemas de comunicação entre equipes, geralmente enfrentados pela equipe de desenvolvimento com a equipe de operações. Isso acaba, muitas vezes, afetando o tempo em que as entregas de software são disponibilizadas aos clientes.

Com as metodologias ágeis (HUMBLE; FARLEY, 2014) sendo mais visíveis pelo mundo, veio o advento do movimento *development and operations* (DevOps). DevOps (SATO, 2014) é um movimento cultural que, através de um conjunto de práticas e ferramentas, auxilia as organizações a entregarem software com mais frequência e com qualidade. Além disto, a cultura visa quebrar quaisquer barreiras existentes entre as equipes e principalmente encorajar a comunicação entre elas. Isso reflete positivamente nas entregas, entre outros benefícios oriundos das práticas de DevOps, pois se tornam mais frequentes e confiáveis, uma vez que as equipes encontram-se mais unidas e colaborativas (HUMBLE; FARLEY, 2014).

A entrega de software de sistemas embarcados no âmbito das telecomunicações, em particular, é um grande desafio, pois há vários fatores em seu desenvolvimento que exigem uma maior atenção. Esses fatores estão relacionados aos inúmeros tipos de protocolos de comunicação, montagens dos ambientes de testes, comunicação entre vários equipamentos etc. Tudo isso é bastante trabalhoso, demanda tempo, esforços físicos, testes e requer também uma certa qualidade e estabilidade.

As práticas DevOps ainda são adotadas de forma exploratória. Assim, o conjunto de práticas e ferramentas adotadas acaba não sendo documentado de forma adequada, o que resulta em uma dispersão de conhecimento que pode se perder. Além disso, uma vez que se tenha uma visão de processo resultante da adoção de DevOps, é possível identificar melhorias.

Dessa forma, neste trabalho, propomos a realização da análise das práticas de DevOps adotadas em uma empresa de telecomunicações para que se possa entender e documentar como as práticas estão sendo adotadas. Também visa-se identificar decisões

não ótimas que podem ser melhoradas.

A Empresa em questão atua com o desenvolvimento de produtos e soluções tecnológicas para o setor de telecomunicações, desde a sua fabricação até a sua comercialização. O foco principal da Empresa está direcionado diretamente para o desenvolvimento de equipamentos de redes, tais como switches e roteadores. Para tal, a Empresa conta com uma equipe de Pesquisa e Desenvolvimento (P&D) para o desenvolvimento próprio de um sistema embarcado para esses equipamentos.

A cultura DevOps, já adotada há alguns anos, veio para fazer diferença na Empresa. E uma das práticas de DevOps que possibilitou o sucesso na entrega de software foi a prática de integração contínua e o uso do pipeline de entrega contínua. Através do uso de ferramentas apropriadas, automatização e visibilidade dos processos, foi possível ter um desenvolvimento frequente de produtos estáveis e também incentivou as equipes a terem autonomia para evoluirem a qualidade tanto do processo quanto dos produtos, culminando no aumento da confiança dos clientes.

Um pipeline é composto por uma série de etapas que são executadas a fim de disponibilizar uma nova versão de software. Todavia, ter um pipeline de entrega contínua exige um cuidado para mantê-lo atualizado de acordo com os requisitos de negócios e buscando, sempre que possível, por melhorias. Isto, inclusive, também faz parte das práticas de DevOps, a melhoria contínua.

O presente trabalho busca propor melhorias no processo de entrega contínua da Empresa de Telecomunicações, apresentando como é realizado o pipeline de entrega contínua e mostrando seus benefícios. Desta forma, este trabalho busca contribuir com a Empresa através das melhorias propostas e na documentação para melhor entendimento do processo. Isto será útil quando houver novos colaboradores e também como contribuição para novos profissionais da área, mostrando como funciona um pipeline de entrega contínua para sistemas embarcados.

Este trabalho está estruturado da seguinte forma.

- O **Capítulo 2** apresenta a fundamentação teórica necessária para o entendimento do trabalho, bem como apresenta o levantamento das ferramentas utilizadas no processo de entrega contínua de um software embarcado.

- O **Capítulo 3** apresenta um contexto geral sobre a empresa-alvo deste trabalho e também apresenta os passos realizados para o mapeamento do processo do pipeline de entrega contínua de um software embarcado.

- O **Capítulo 4** apresenta a documentação do pipeline de entrega contínua.
- O **Capítulo 5** apresenta as propostas de melhorias para o pipeline de entrega contínua de um projeto de software embarcado.
- O **Capítulo 6** apresenta a conclusão deste trabalho e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para que possamos compreender como funciona um pipeline de entrega contínua para sistemas embarcados, é necessário primeiro entendermos os conceitos de assuntos relacionados como as práticas de DevOps, sistemas embarcados e ferramentas que são utilizadas durante o processo de entrega contínua. Neste capítulo são abordados esses assuntos.

2.1 DevOps

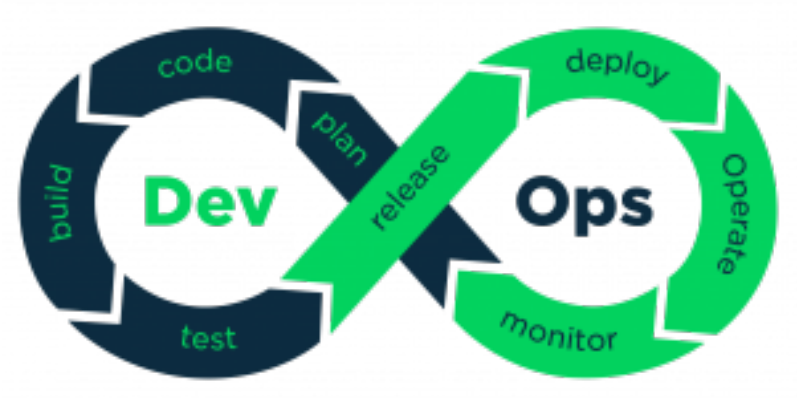
O advento do DevOps veio após a adoção em larga escala das metodologias ágeis. DevOps é um movimento cultural que visa criar uma cultura e uma melhora na colaboração e comunicação entre as equipes de desenvolvimento (Dev) e de operações (Ops). Através dessa colaboração é possível aumentar, com segurança, a estabilidade e robustez do ambiente de produção, o que se dá através dos elementos essenciais que a cultura DevOps aborda, tais como: código, compilação, testes automatizados, planejamento, operação, implantação, monitoramento e lançamento de versões. Na Figura 2.1 é possível visualizar como esses elementos estão ligados às equipes de desenvolvimento e de operações (SATO, 2014).

Assim como a cultura das metodologias ágeis romperam algumas barreiras entre clientes, arquitetos, desenvolvedores, analistas de negócios e de qualidade, a cultura DevOps veio para ampliar essa colaboração, incluindo infraestrutura, gerenciamento de lançamento, suporte, segurança, entre outras funções nas operações de TI (MINUDEL; MORRIS, 2014).

2.1.1 Pilares do DevOps

Seguir os pilares do DevOps é fundamental para que a organização e as equipes tenham sucesso na adoção da cultura desse movimento. Sato (2014) ressalta que o movimento cultural DevOps não é apenas a utilização de ferramentas, de modo que para definir o que é o DevOps, Willis (WILLIS, 2010) criou o acrônimo CAMS e posteriormente Humble adicionou Lean para completar esse acrônimo, resultando em CALMS (SATO, 2014). O significado de cada uma das letras que compõem o acrônimo é apresentado a

Figura 2.1: Ciclo DevOps



Fonte: (LEE, 2018)

seguir.

- **(C)ultura:** Cultura é a base do DevOps, em que pessoas e processos de comunicação são mais importantes do que as ferramentas utilizadas na realização das atividades.

- **(A)utomação:** A ideia é automatizar todas as tarefas repetitivas que forem possíveis de se automatizar, deixando assim, os humanos livres para que possam desenvolver as tarefas criativas. Tarefas automatizadas são mais confiáveis devido a sua baixa taxa de erro humano.

- **(L)ean (pensamento enxuto):** A metodologia Lean agrega em vários fatores na cultura DevOps, tais como: melhoria contínua, qualidade, eliminação de desperdícios, otimização do todo e o respeito às pessoas.

- **(M)edição:** É importante medir tudo o que for possível em relação a dados e pessoas, pois só assim é possível saber se há evolução. Alguns exemplos do que pode ser medido são as métricas de desempenho, processos, negócios.

- **(S)haring (compartilhamento):** Tão importante quanto a colaboração entre as equipes são os compartilhamentos de ideias e conhecimentos. Isso é essencial para se ter sucesso com DevOps.

Para Sato (2014), o movimento DevOps tem como objetivo focar nas práticas de automação de diversas atividades que possam sempre melhorar a qualidade do código que está em produção. Algumas dessas atividades são: compilação de código, criação e configuração de ambientes e infraestrutura (para testes, homologação ou produção), desempe-

nho, deploy, testes automatizados, monitoramento, segurança, auditoria, gerenciamento de logs e métricas, entre outros.

2.1.2 Práticas DevOps

Seguir as práticas de DevOps é essencial para que a adoção da cultura DevOps seja implementada com sucesso nas organizações. Dentre as práticas abordadas pela cultura, estão o uso de ferramentas e abordagens diferentes para a realização de processos como um todo. A seguir são apresentadas as práticas de *Integração Contínua*, *Entrega Contínua* e *Pipeline de Entrega Contínua* que compõem a cultura DevOps.

2.1.2.1 Integração Contínua

Segundo Sato (2014), a Integração Contínua (*Continuous Integration - CI*) é uma prática originada da metodologia ágil Programação Extrema (*Extreme Programming - XP*). A prática de integração contínua vem se destacando e sendo cada vez mais utilizada no desenvolvimento de software de equipes ágeis (SATO, 2014). A integração contínua tem como objetivo manter o software sempre em estado funcional. Essa prática de desenvolvimento requer que, a cada mudança realizada na aplicação, haja a recompilação e a execução de uma bateria de testes automatizados (testes unitários, testes de componentes e testes de aceitação), garantindo que a alteração não tenha quebrado a aplicação ou comprometido sua qualidade (HUMBLE; FARLEY, 2014).

Alguns dos benefícios de se utilizar a integração contínua são os feedbacks imediatos, os ganhos de custos e de tempo. Isso se torna possível devido aos defeitos que são encontrados mais cedo, sendo consequentemente mais fáceis de se corrigir devido a rastreabilidade de cada entrega (HUMBLE; FARLEY, 2014).

Para se ter uma integração contínua eficaz, Humble and Farley (2014) ressaltam a importância de se ter times profissionais que sigam os princípios que regem a integração contínua.

2.1.2.2 Entrega Contínua

Entrega Contínua (*Continuous Delivery - CD*) é um conjunto de práticas em Engenharia de Software que visa revolucionar o processo de entrega de software tornando o caminho entre a ideia e o valor de negócio criado mais curto e mais seguro, resultando em

entregas frequentes e confiáveis (HUMBLE; FARLEY, 2014). Um sistema só gera valor e/ou lucro quando já se encontra nas mãos de seus usuários (HUMBLE; FARLEY, 2014). Entrega Contínua é uma extensão natural da prática de integração contínua (SATO, 2014).

Jez Humble diz o seguinte sobre entrega contínua:

Entrega contínua significa garantir que seu software esteja sempre pronto para a produção ao longo de todo o seu ciclo de vida - que qualquer construção possa ser potencialmente liberada para os usuários com o toque de um botão, usando um processo totalmente automatizado em questão de segundos ou minutos (HUMBLE, 2010).

Segundo Minudel and Morris (2014), os benefícios de se utilizar a prática de entrega contínua são muitos, entre os listados a seguir.

- **Riscos reduzidos:** Os riscos são reduzidos como consequência das entregas serem frequentes e de escopo pequeno. Isso beneficia a organização, uma vez que se torna possível verificar rapidamente que o produto certo está sendo construído da maneira certa. Os defeitos são detectados mais cedo, tornando-se mais fáceis e baratos de se corrigir.

- **Redução de desperdício:** Os desperdícios são reduzidos drasticamente quando processos manuais são automatizados. Isto porque, quando a execução de testes, a implantação, a infraestrutura e a configuração são automatizados, a probabilidade de erro se torna muito baixa, reduzindo o custo que haveria com correções e retrabalhos.

- **Maior qualidade:** Como as entregas ocorrem mais cedo e com frequência, lições são aprendidas e informações são descobertas durante esse processo de modo que podem ser utilizadas imediatamente assim que for necessário. Logo, os defeitos podem ser melhor controlados, aumentando a qualidade interna e externa dos produtos entregues. Isto ajuda a aumentar o foco no valor do negócio, o que realmente importa para os clientes e usuários.

- **Maior resiliência:** Há sempre algum plano de remediação e correção que define como se recuperar de erros específicos e bugs em caso de ocorrência destes em produção. Esses planos são procedimentos automatizados e testados para que, quando for necessário utilizá-los, não se tenha surpresas desagradáveis.

- **Maior capacidade de resposta:** Com o tempo de espera reduzido, é reduzido também o tempo de se fazer uma mudança, reagir a novas circunstâncias ou a algum

evento inesperado.

- **Maior inovação:** Com a capacidade de lançar entregas com antecedência e frequência, surge a oportunidade de explorar o mercado, o que permite obter a reação dos usuários ao produto, o valor e a adequação à finalidade do produto e, assim, descobrir novas oportunidades de negócios.

2.1.2.3 Pipeline de Entrega Contínua

Antes de falarmos sobre o pipeline de entrega contínua, é importante esclarecer a diferença de entrega contínua e implantação contínua, que muitas vezes são confundidas. Segundo Sato (2014), a *implantação contínua (deploy contínuo ou Continuous Deployment) é a prática de colocar em produção todo commit bem sucedido e geralmente significa fazer diversos deploys em produção por dia; entrega contínua é a prática onde qualquer commit pode ir para produção a qualquer momento, porém a decisão de quando isto acontece é uma opção de negócio.*

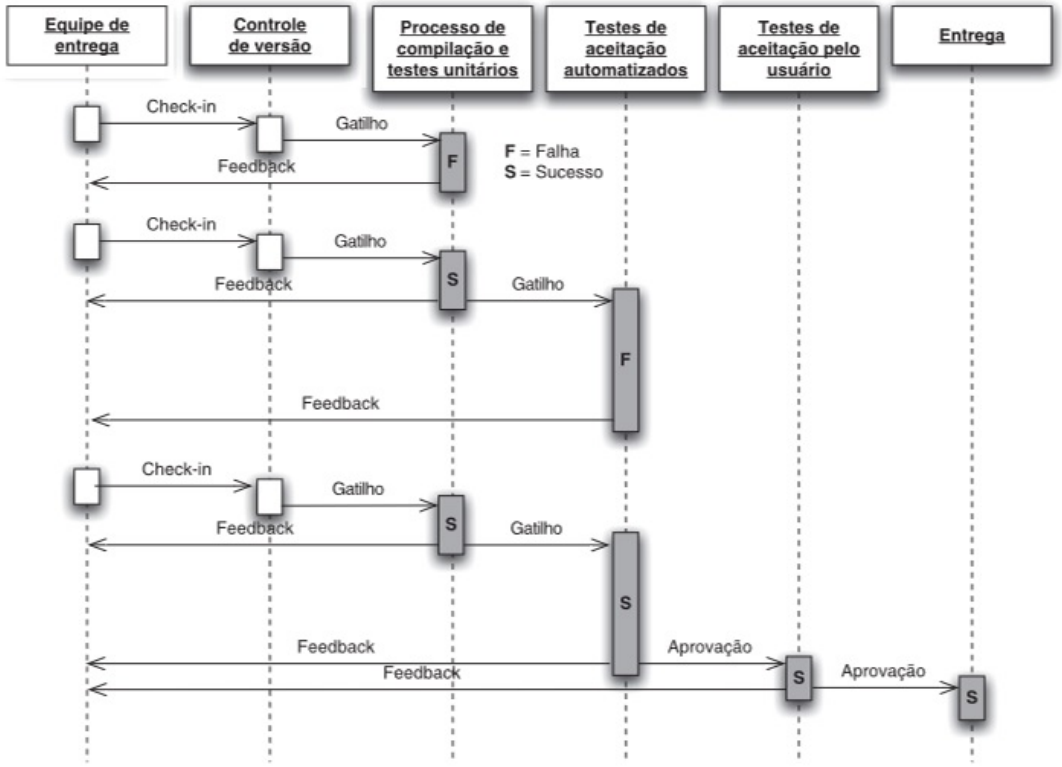
Um pipeline de entrega contínua é a automação de todo o processo que uma mudança no software percorre até chegar ao usuário final. Esse processo envolve desde a compilação do software até os variados estágios de testes e implantação. Muitas vezes para seguir esse processo, são necessários o envolvimento e a colaboração de várias equipes.

O pipeline de entrega contínua visa beneficiar todos os envolvidos nesse processo, pois permite que eles vejam e controlem o passo a passo da mudança. Assim, com essa visibilidade, as equipes conseguem atuar o quanto antes em caso de falhas e/ou problemas (HUMBLE; FARLEY, 2014). Para melhor entender como funciona um pipeline de entrega contínua, Humble and Farley (2014) utilizaram um diagrama de sequência, mostrado na Figura 2.2, para ilustrar cada etapa. No entanto, cada organização pode implementar e configurar seu pipeline de entrega contínua de acordo com as suas necessidades.

2.2 Sistemas Embarcados

Um sistema embarcado, também conhecido como sistema embutido, é um sistema que executa um conjunto de tarefas, cujos requisitos são bem específicos e definidos. Esse tipo de sistema é completamente encapsulado no dispositivo o qual ele controla e, por isso, não pode ter suas tarefas alteradas durante o seu uso (OSSADA; MARTINS, 2010).

Figura 2.2: Diagrama de Sequência de um Pipeline de Entrega Contínua



Fonte: (HUMBLE; FARLEY, 2014)

Podemos definir que sistemas embarcados são aplicações desenvolvidas para interagir diretamente com o hardware, conhecido também como desenvolvimento de firmware. Para isso, há a necessidade de se ter conhecimentos profundos necessários do hardware para o qual o sistema está sendo desenvolvido. Na maioria das vezes, sistemas embarcados não possuem interface gráfica ou interação com o usuário. Geralmente a comunicação com os sistemas embarcados dá-se através do uso de protocolos de comunicação como ethernet e USB (FRõES; WEBER, 2020).

A base dos sistemas embarcados são os microcontroladores e os microprocessadores (REIS, 2015). Há algumas características importantes que devemos citar sobre os sistemas embarcados, como por exemplo: baixo consumo de energia, baixo custo (chips que custam um valor muito baixo), confiabilidade e segurança, operações em tempo real, baixo custo de resposta, operações especializadas (execução de tarefas específicas), entre outras. Esses sistemas fazem muito uso da área de eletrônica (FRõES; WEBER, 2020).

Os sistemas embarcados não são recentes, o primeiro sistema embarcado reconhecido foi no sistema aeroespacial Apollo Guidance Computer, desenvolvido por Charles Stark Draper. E esse sistema foi utilizado pela NASA para levar o homem à Lua nos anos 60 e 70 (FRõES; WEBER, 2020). Desde essa época, os sistemas embarcados vêm

crescendo e evoluindo, e atualmente estão largamente presentes em nosso dia a dia. Podemos encontrá-los em smartphones, automóveis, switches, roteadores, brinquedos, caixas eletrônicos, eletrodomésticos (micro-ondas, lavadoras), robótica, entre outros (FRõES; WEBER, 2020). Desde então, os sistemas embarcados se tornaram indispensáveis para a vida moderna (REIS, 2015).

Todavia, realizar entregas frequentes de sistemas embarcados também requer uma atenção maior devido ao uso de ferramentas específicas para o seu desenvolvimento e a complexidade para a execução de testes. Entretanto, utilizar um pipeline de entrega contínua torna todo esse processo mais simples e fácil de gerenciar. Na seção seguinte, são apresentadas ferramentas utilizadas no pipeline de entrega contínua para o desenvolvimento de sistemas embarcados.

2.3 Ferramentas utilizadas no pipeline de entrega contínua

Segundo Humble and Farley (2014), é essencial o uso de ferramentas que contribuem para a adoção da prática de entrega contínua. Sem essas ferramentas, não há como garantir que a entrega de software seja contínua e confiável. No entanto, não é necessário fazer uso da melhor ferramenta ou a considerada mais utilizada no que tange o desenvolvimento de software. Entretanto, é necessário que as ferramentas escolhidas atendam aos requisitos que regem os princípios de entrega contínua. É considerado como requisito o uso de ferramentas que contemplem: *sistema de controle de versão*, *servidor de entrega contínua*, *gerência de configuração de ambientes* e *gerência de testes*.

A seguir são apresentados os conceitos de cada um desses requisitos e também como as ferramentas utilizadas por esses requisitos auxiliam no desenvolvimento e na qualidade de software embarcado.

2.3.1 Sistema de Controle de Versão

Para que o desenvolvimento de software seja produtivo e o trabalho em equipe seja colaborativo, é importante que a organização trabalhe com algum tipo de sistema de controle de versão. Um sistema de controle de versão é uma ferramenta utilizada para o armazenamento de código fonte dos desenvolvedores e dos projetos.

A utilização desta ferramenta torna-se importante dado que, com ela, podemos fa-

zer o versionamento do código de desenvolvimento de forma que nenhuma alteração seja perdida e que desenvolvedores de diferentes equipes possam trabalhar simultaneamente sem prejudicar o trabalho um do outro. Falar em sistema de controle de versão nos dias de hoje já é algo bem natural devido aos seus inúmeros benefícios e exigências para o mercado de trabalho. A expressão *máquinas do tempo* e *robôs de integração* utilizada por Aquiles and Ferreira (2014) nos ajuda a entender melhor como esses sistemas funcionam e seus benefícios.

A *máquina do tempo* se refere às alterações dos arquivos de códigos, ou seja, nos permite acompanhar a evolução e as mudanças do arquivo desde sua primeira versão, nos permite também identificar os autores dos códigos, exclusão de trabalhos/códigos obsoletos e o mais importante, nos permite voltar a versões anteriores em casos de desastres.

Já os *robôs de integração* se referem a forma como serão mescladas as informações/alterações dos diversos desenvolvedores da equipe ou até mesmo entre equipes. Em outras palavras, os robôs de integração nada mais são que a forma como um sistema de controle de versão funciona quando há pessoas mexendo em arquivos simultaneamente, de como resolver os conflitos de informações divergentes, de como saber quem está realizando tal trabalho, etc. Esses pontos são alguns dos benefícios que um sistema de controle de versão nos fornece de forma simples, rápida, transparente e automática.

O sistema de controle de versão utilizado pela empresa-alvo é o Git[1]. O Git é bastante conhecido por ser muito simples, prático, rápido, de código aberto e de fácil aprendizagem. O motivo pelo qual o Git foi escolhido foi porque esse sistema permite o desenvolvimento distribuído, em outras palavras, não há a necessidade de se ter uma conexão com o repositório para se ter acesso ao código fonte, o que facilita muito o processo de desenvolvimento de software.

2.3.2 Servidor de Entrega Contínua

Um servidor de entrega contínua é uma ferramenta utilizada para a criação e execução de tarefas cujo foco é a automação destas. Esta ferramenta é muito utilizada nas práticas de DevOps e, desta forma, tem se tornado uma ferramenta essencial no processo de entrega contínua devido a simplicidade e benefícios que possui. Um de seus benefícios, por exemplo, é poder executar inúmeras tarefas sem ou com mínima intervenção humana. Através dessa ferramenta, é possível criar rotinas desde as mais simples até as mais com-

[1] https://git-scm.com

plexas, as quais podem auxiliar tanto no dia a dia do desenvolvimento de software quanto em operações que exigem maiores cuidados como, por exemplo, uma implementação em produção.

Para realizar todos os processos automatizados do ciclo de desenvolvimento de software da empresa-alvo, é utilizada a ferramenta Jenkins[2]. O Jenkins é um servidor de integração e entrega contínua de código aberto muito conhecido e utilizado para criação, automação e implementação de tarefas. Esta ferramenta é composta por plugins que, por sua vez, são considerados funcionalidades do Jenkins. Isto é visto como um benefício, uma vez que a escolha por funcionalidades pode ser feita sob demanda de acordo com o uso, tornando a ferramenta mais objetiva e com melhor performance.

Cada execução no Jenkins é chamada de *job*, que é um projeto configurado para a execução de uma tarefa. De modo geral, essa configuração pode ser realizada de diversas formas de acordo com a tarefa que se deseja executar. Um job pode, por exemplo, ser responsável por executar uma série de comandos que automatizam uma atividade ou copiar artefatos entre servidores. Além disso, o Jenkins torna-se útil quando permite abstrair a conexão com outras máquinas e por ser compatível com a maioria das ferramentas de projetos de desenvolvimento de software.

2.3.3 Ferramenta de Compilação

A utilização de uma ferramenta automatizada de compilação é muito relevante no processo de desenvolvimento de software devido a sua facilidade em gerir as necessidades de dependências nos projetos. Em outras palavras, é uma ferramenta que permite criar uma modelagem de rede de dependências. Nesta modelagem, são considerados os passos necessários e a ordem de execução destes para realizar uma dada tarefa.

Segundo Humble and Farley (2014), existem basicamente dois tipos de ferramentas de compilação: a primeira é *orientada a tarefa* e a segunda é *orientada a produto*. A ferramenta orientada a tarefa (por exemplo Ant, NAnt e MsBuild) refere-se a execução de um conjunto de tarefas. Já a ferramenta orientada a produto (por exemplo Make) representa os processos em termos de produto gerado, o que resulta em um executável.

Para desenvolver o próprio sistema operacional para as diferentes plataformas de hardware e emuladores para os equipamentos, é utilizada a compilação cruzada. Esta é, por sua vez, uma forma de utilizar um compilador que permite gerar um código executável

[2]https://www.jenkins.io

para uma plataforma/ambiente alvo que não suporta um processo de compilação. No caso da empresa-alvo, o processo de compilação cruzada é realizado através da ferramenta Buildroot.

Buildroot[3] é utilizado para desenvolver sistemas embarcados Linux através de compilação cruzada, e faz uso da ferramenta orientada a produto Make. A ferramenta Buildroot é muito utilizada no mundo de sistemas embarcados, uma vez que permite a construção de uma gama de ferramentas para a compilação cruzada, o que simplifica e automatiza muitos processos. Além disso, ela é conhecida por ter suporte a vários processadores, tais como PowerPC e ARM, permitindo o desenvolvimento de muitos projetos embarcados (ORG, 2021).

2.3.4 Configuração de Ambiente

A configuração de ambiente é mais uma etapa essencial para o processo de entrega contínua de software. O conceito de ambiente no desenvolvimento de software compreende um espaço alocado, que pode ser tanto físico quanto virtual, apropriado para a execução de aplicações.

A configuração de um ambiente, comumente conhecido como provisionamento de ambiente, consiste em preparar um espaço com os recursos específicos que uma aplicação necessita para sua execução, recursos os quais envolvem a configuração de hardware (como quantidade de memória, armazenamentos, interfaces de rede, etc.), configuração de sistema operacional (plataforma suportada pela aplicação/serviço) e configuração de middleware (como servidores web, banco de dados, sistemas de mensagens etc.). É recomendado que todos esses requisitos sejam configurados e gerenciados através de ferramentas de automação. Dessa forma, o uso de virtualização ao realizar as automações de provisionamento de ambientes é de extrema importância (HUMBLE; FARLEY, 2014).

A virtualização é uma tecnologia bastante conhecida e utilizada atualmente no que tange a configuração de ambientes. Virtualização significa poder executar uma aplicação ou serviço virtualmente ou em nuvem. No entanto, para habilitar uma virtualização é necessário ter ao menos uma máquina física. Existem duas principais abordagens para trabalhar com virtualização, uma baseada em *máquinas virtuais* e outra em *containers*.

Uma máquina virtual (*Virtual Machine* – VM) consiste em emular um ambiente computacional sobre outro ambiente computacional. Por outro lado, um container con-

[3] https://buildroot.org

siste em isolar parte do sistema operacional. As máquinas virtuais utilizam um *Hypervisor* para o seu gerenciamento. Um Hypervisor, conhecido também como Monitor de Máquina Virtual (*Virtual Machine Monitor* - VMM), é utilizado para compartilhar os recursos de hardware de uma máquina física com a(s) máquina(s) virtual(is). Já os containers utilizam um mecanismo mais leve cujo objetivo é apenas compartilhar um único sistema operacional (RAHO et al., 2015). Ambas as abordagens possuem vantagens e desvantagens, portanto, é importante avaliar os requisitos do meio no qual serão aplicadas. Essas abordagens são vistas em ferramentas como as apresentadas a seguir.

- **Docker:** O Docker[4] é uma ferramenta utilizada no gerenciamento de containers. Um container é basicamente um ambiente isolado do sistema operacional que permite a execução de uma aplicação.

- **KVM:** KVM[5] (Kernel-based Virtual Machine), como o próprio nome informa, é uma Máquina Virtual baseada em kernel, ou seja, sua virtualização já é integrada ao sistema operacional Linux.

A necessidade de se ter sempre o mesmo ambiente com a mesma configuração e a mesma disponibilidade para diferentes cenários (como produção, homologação ou testes) faz com que o uso da tecnologia de virtualização seja cada vez mais presente no desenvolvimento de software. Seu uso nos processos de integração e entrega contínua se faz indispensável devido à facilidade de provisionar e gerenciar os ambientes.

2.3.5 Gerenciamento de Testes

Para termos uma entrega de sucesso, é necessário que a sua qualidade seja de sucesso também. Para tal, a qualidade da entrega se dá através dos variados níveis de testes executados sob ela, tais como: unitário, componente, regressão, aceitação etc. Para Humble and Farley (2014), o termo *testar* significa uma atividade a ser exercida e executada continuamente por todos os integrantes do time.

No entanto, selecionar quais os tipos de testes que serão executados no pipeline de entrega contínua é uma tarefa que exige muito cuidado e atenção. A importância desses testes faz total diferença na qualidade e confiabilidade da entrega, sendo assim, a criticidade dos testes precisa ser de um nível bastante elevado.

[4]https://www.docker.com
[5]https://www.linux-kvm.org

Segundo Humble and Farley (2014), o teste fundamental para ser executado em um pipeline de entrega contínua são os testes de aceitação. Através dos testes de aceitação podemos garantir que estamos testando os critérios de aceitação do negócio da aplicação, em outras palavras, estará sendo validado que o código entregue gera funcionalidade de valor para os usuários. Logo, é necessário que haja uma estratégia eficiente desses testes. Por outro lado, os demais testes devem ser executados em outras rotinas configuradas no servidor de integração contínua.

De modo geral, a gerência e a organização dos testes é um fator essencial para que a qualidade da entrega seja atendida. Neste processo, é recomendado que os testes sejam totalmente automatizados a fim de facilitar a sua manutenção, suas dependências e execuções em todos os ambientes necessários, tais como desenvolvimento, homologação e produção.

Para realizar a gerência e execução dos testes no pipeline de entrega contínua da empresa-alvo é utilizada a ferramenta Robot Framework[6]. O Robot Framework permite realizar a escrita, execução e análise dos resultados dos testes. O uso desta ferramenta foi efetivado por ser de código aberto, compatível com o servidor Jenkins, independente de sistema operacional e de fácil aprendizagem, o que permite aos times ter uma maior liberdade para criar automações personalizadas.

2.4 Considerações Finais

Nos dias atuais sabemos o quão essencial é entregar um software com qualidade e com frequência para os clientes. No entanto, estas atividades não são simples de serem realizadas, visto que envolvem uma série de fatores que vão desde o seu planejamento até o seu desenvolvimento. Para auxiliar as organizações em todos esses processos que tangem o desenvolvimento e qualidade de software, foi apresentado neste capítulo como a cultura DevOps pode ser benéfica para as organizações que a adotam. Dentre os benefícios apresentados, está o uso de um pipeline de entrega contínua.

Assim, para o melhor entendimento, foram apresentadas as ferramentas utilizadas em um pipeline de entrega contínua de um software embarcado. No entanto, para se ter um pipeline de entrega contínua de sucesso, é necessário entender como o software funciona e, para tanto, é de extrema importância que exista uma documentação a respeito dele. No próximo capítulo será apresentada uma forma de documentação de alta abstração

[6]https://robotframework.org

através do mapeamento do processo que visa facilitar o gerenciamento e evolução de um projeto de software.

3 PROCEDIMENTO DE MAPEAMENTO DO PROCESSO

Neste capítulo é apresentada a visão geral de desenvolvimento da empresa-alvo e os passos realizados no mapeamento do processo do pipeline de entrega contínua de um software embarcado.

3.1 Visão Geral sobre a Empresa-alvo

O presente trabalho é realizado sob o pipeline de entrega contínua de uma empresa de telecomunicações que tem como foco o desenvolvimento de equipamentos tais como switches e roteadores.

Cada modelo de equipamento leva ao desenvolvimento de uma plataforma cuja numeração é interna e sigilosa. O processo de desenvolvimento da Empresa é composto por equipes, de forma que cada equipe é responsável por um módulo do sistema embarcado principal. Neste contexto, o desenvolvimento de software é chamado de desenvolvimento de sistema embarcado.

Quando uma alteração é realizada no código fonte do sistema embarcado, é necessário atualizar os equipamentos. Este processo é chamado de atualização de firmware e se dá através de uma imagem de sistema operacional. Um firmware nada mais é que um sistema de baixo nível utilizado para controlar o hardware do equipamento, ou seja, a atualização de firmware é comumente conhecida como atualizar o sistema operacional do equipamento. Entretando, uma vez que os equipamentos tenham sido atualizados, é necessário garantir que eles continuem em estado funcional, o que é realizado através da execução de uma bateria de testes. Todo o processo de atualizar o firmware dos equipamentos, e a consequente execução de testes, era realizado de forma manual, fazendo com que uma entrega levasse em média cinco semanas para ser disponibilizada no servidor oficial, quando bem sucedida. Em caso de falha, por outro lado, poderia levar meses a mais para realizar uma entrega.

Com a adoção das prática de DevOps e a implementação do pipeline de entrega contínua foi possível melhorar o processo de desenvolvimento e a entrega de software na Empresa. Entre os inúmeros benefícios obtidos após a adoção do movimento DevOps estão a automação dos processos, o rápido feedback, a segurança, a maior confiabilidade da entrega e a transparência para as equipes do processo como um todo.

3.2 Procedimento para Mapeamento do Processo

Neste trabalho é realizado um procedimento cujo objetivo é criar uma documentação do pipeline de entrega contínua de um projeto de software embarcado, de modo que se obtenha o conhecimento necessário de seu funcionamento para propor sugestões de melhorias. Os passos deste procedimento são listados a seguir.

- **Levantamento das ferramentas utilizadas:** Realização do levantamento das ferramentas que são utilizadas no processo de entrega contínua a fim de apresentar o contexto e finalidade delas, conforme descrito na Seção 2.3.
- **Identificação e documentação das etapas do pipeline:** Realização da identificação das etapas que compõem o pipeline, bem como o papel e regras de cada uma. Descrição do processo que é realizado em cada etapa do pipeline. E, por fim, a apresentação de um diagrama de sequência para melhor entendimento do processo realizado no pipeline.
- **Análise e propostas de melhorias no pipeline:** Realização de uma análise de cada etapa do pipeline de entrega contínua a partir da documentação desenvolvida. Esta análise visa propor melhorias no que tange os princípios da entrega contínua, tais como entregas frequentes, qualidade da entrega e melhorias contínuas.

3.3 Considerações Finais

Conforme o contexto apresentado neste capítulo, fazer uso do mapeamento do processo pode trazer muitos benefícios, além de ajudar na diminuição de desperdícios de tempo. Uma vez que se conhece como o software funciona, pode-se evitar muitos contratempos e investir em melhorias como qualidade, expansão, manutenção, inovação e em novas implementações ou reuso do software.

Além disso, a existência de uma documentação sucinta sobre o sistema, e com certo nível de abstração, facilita o seu suporte. No entanto, é necessário que a documentação esteja sempre atualizada.

4 RESULTADOS

Neste capítulo é apresentada a documentação realizada sobre o pipeline de entrega contínua, conforme o procedimento descrito no capítulo anterior. Por fim, um diagrama de sequência é fornecido para ilustrar os procedimentos de cada etapa que compõem este pipeline.

4.1 Estrutura do Pipeline

A estrutura de um pipeline de entrega contínua é composta por etapas sequenciais e/ou paralelas para as quais só é permitido passar para a próxima etapa se na primeira todos os requisitos impostos foram executados com êxito. Desta forma, a entrega torna-se mais confiável e segura. Contudo, para que isso aconteça, é necessário que os critérios/requisitos estabelecidos para cada etapa sejam claros e rigorosos (HUMBLE; FARLEY, 2014).

Para compreendermos a estrutura do pipeline de entrega contínua da Empresa, precisamos entender como funciona o processo de integração contínua dela. O projeto para o qual o pipeline foi definido é desenvolvido por módulos e conta com mais de seis equipes distribuídas. Os módulos, por sua vez, são considerados funcionalidades dos equipamentos de telecomunicações. Todo o código fonte do projeto está inserido no sistema de controle de versão - Git.

O processo de integração contínua é realizado quando o desenvolvedor envia sua modificação para o Gerrit[1], no qual é realizada a revisão em duas etapas. A primeira etapa é a revisão automática que se dá através da integração com a ferramenta Jenkins, na qual é executada uma série de validações de acordo com o módulo em questão. Essas validações são análises de código, documentação do módulo, verificação de dependências entre módulos, etc. E a segunda etapa é manual, em que é necessária a revisão de algum outro membro da equipe a fim de aprovar a modificação. Após esse procedimento, a modificação estará apta para passar para o processo de entrega contínua.

Por outro lado, o processo de entrega contínua é realizado pelo pipeline, no qual o procedimento é unir todos os módulos do projeto ao sistema de compilação, criar uma imagem de firmware para os equipamentos e posteriormente executar os testes de aceitação e integração.

[1] Ferramenta vinculada ao Git para gerenciar a revisão de código.

É importante ressaltar que todas as etapas presentes no pipeline foram definidas pelo time de automações juntamente com o responsável de cada equipe, definindo assim os critérios e regras para cada etapa.

4.2 Descrição das Etapas do Pipeline

Atualmente o pipeline utilizado no processo de entrega contínua é composto por seis etapas, nas quais são executados processos como verificação de commits, construção de imagens, provisionamento de VMs, execução de testes de aceitação e integração e consolidação de artefatos. Todos esses processos são executados através de jobs que, por sua vez, executam scripts nas linguagens de programação python, shell script e comandos de CLI (command-line interface).

Através do diagrama de sequência apresentado na Figura 4.1, podemos ter uma visão geral sobre o fluxo de todas as etapas que compõem o pipeline de entrega contínua da empresa-alvo. A seguir é apresentada a documentação dessas etapas.

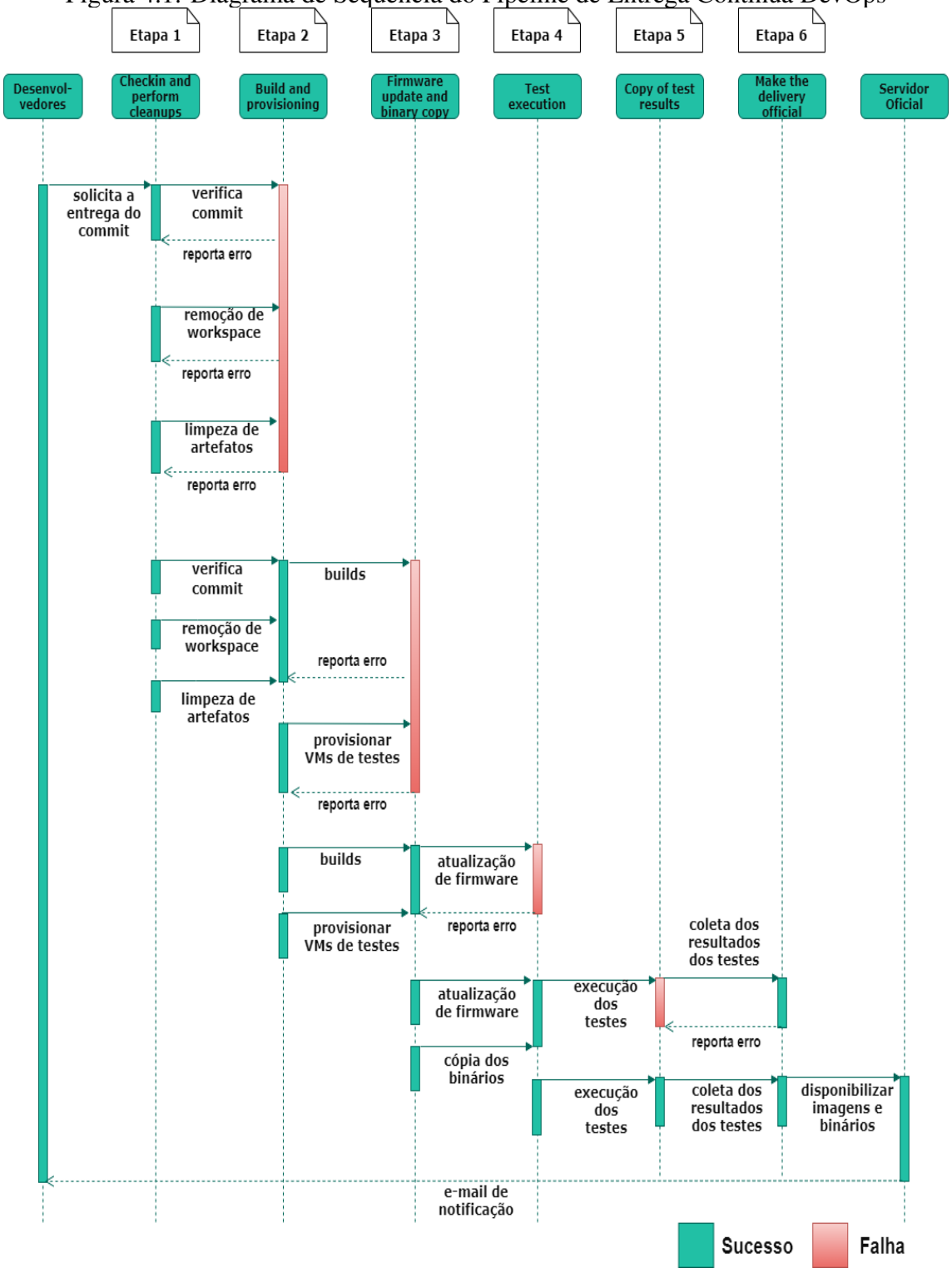
4.2.1 Etapa 1 - Checkin and Perform Cleanups

O processo de *Checkin* consiste em validar os commits que foram submetidos para o processo de entrega contínua. Essas validações verificam se os commits foram revisados, aprovados/reprovados ou se, até mesmo, já foram integrados. Se estiver tudo certo com os commits, é realizado um merge com o repositório principal (buildroot) e o resultado é arquivado em um arquivo de extensão .tar.gz para ser utilizado nas etapas seguintes.

Já o processo de *Perform Cleanups* é o processo de realizar limpezas. É realizada a remoção do workspace (espaço de trabalho do job) e dos resultados de artefatos dos testes da execução de entrega contínua anterior.

Os processos citados acima sobre as validações e as limpezas são executados por jobs independentes e em paralelos. Entretanto, se algum deles falhar, a execução da entrega é interrompida e não passa para a próxima etapa.

Figura 4.1: Diagrama de Sequência do Pipeline de Entrega Contínua DevOps



Fonte: Autora

4.2.2 Etapa 2 - Build and Provisioning

O processo de *Build* consiste em gerar as imagens de firmware para cada plataforma presente no pipeline de entrega contínua. A criação dessas imagens é chamada de build ou compilação. Os builds, por sua vez, são executados em máquinas comuns com poucos recursos de hardware através de um container Docker para realizar a compilação da imagem. É importante ressaltar que os builds são executados um por vez na máquina, de modo a não comprometer o desempenho dela (essas compilações costumam ser pesadas) e para otimizar o tempo de compilação. Sendo assim, todos os builds são executados em paralelo.

Já o processo de *Provisioning* consiste em preparar os ambientes que executam os testes através da criação e do provisionamento das VMs de testes, sendo um ambiente específico para cada plataforma. Todos os processos nesta etapa são executados em paralelo, entretanto, se houver alguma falha, a próxima etapa não é executada.

4.2.3 Etapa 3 - Firmware Update and Binary Copy

O processo de *Firmware Update* consiste em atualizar os equipamentos de testes a partir da imagem gerada na Etapa 2. Essa imagem é considerada uma imagem de sistema operacional, de modo que é necessária a instalação dessa imagem nos equipamentos para que possam ser executados os testes posteriormente. É importante ressaltar que esses ambientes de testes são compostos por máquinas físicas (computadores) e dispositivos auxiliares que estão distribuídos em diferentes racks e datacenters pela empresa, sendo o acesso a eles feito através da rede.

Já o processo de *Binary Copy* consiste em copiar as imagens e binários gerados até o momento em um diretório específico no servidor. Este procedimento é utilizado para depuração em casos de falhas da entrega. Desta forma, é possível analisar e executar os artefatos localmente sem precisar gerá-los novamente. Todos os processos nesta etapa são executados em paralelo, entretanto, se houver alguma falha, a próxima etapa não é executada.

4.2.4 Etapa 4 - Test Execution

Esta etapa consiste em executar os testes de aceitação e integração nos ambientes de testes configurados nas etapas anteriores.

4.2.5 Etapa 5 - Copy of Test Results

O processo de *Copy of test results* consiste em coletar os resultados dos testes e disponibilizá-los em um lugar visível para o job da entrega em questão. Esta etapa é executada independente dos resultados dos testes executados na etapa anterior, no entanto, se houver alguma falha na Etapa 4, a próxima etapa não é executada.

4.2.6 Etapa 6 - Make the Delivery Official

Esta etapa consiste em disponibilizar no servidor oficial as imagens e binários da entrega corrente e enviar a tag do commit para o repositório principal (buildroot). Por fim, é enviado um e-mail para todos os desenvolvedores oficializando a entrega do commit.

4.3 Tempo de Execução das Etapas

Para complementar as informações do pipeline de entrega contínua que foram descritas na seção anterior, é importante ressaltar o tempo que cada uma das etapas leva para ser executada. Na Tabela 4.1 é apresentada uma visão geral sobre as etapas, assim como o tempo total de execução delas.

Entretanto, para que a entrega contínua cumpra o seu princípio de feedback rápido, é importante estabelecer um tempo limite para a sua execução. No contexto descrito na seção anterior, existe uma configuração de timeout para algumas etapas, a qual se faz necessária devido a inúmeras situações que podem ocorrer e dificultar o feedback rápido, como por exemplo: instabilidade de rede, comunicação com os equipamentos de testes, tempo de respostas dos processos, entre outros fatores. Atualmente, a entrega contínua tem levado em média um tempo total de 2 horas e 20 minutos de execução. No entanto, esse tempo pode variar devido a algumas regras no processo das Etapas 2, 3 e 4, conforme descritas a seguir.

Tabela 4.1: Resumo das Etapas do Pipeline de Entrega Contínua

Etapas	**Resumo**	**Tempo Total de Execução**
1 - Checkin and Perform Cleanups	Nesta etapa é realizada a validação do commit que os desenvolvedores submeteram para o processo de entrega contínua e, em paralelo, é realizada a limpeza dos workspaces (espaço de trabalho do job) das execuções anteriores. Entretanto, para que o pipeline continue executando, é necessário que todos os processos desta etapa sejam concluídos com sucesso.	De 2 a 3 minutos
2 - Build and Provisioning	Nesta etapa é realizada a construção da imagem de firmware dos equipamentos. Para cada modelo de equipamento é gerada uma imagem de sistema operacional. Em paralelo, é realizada a configuração inicial desses equipamentos que posteriormente irão receber essa imagem. Nesta etapa é necessário que todas as execuções sejam concluídas com sucesso para que o pipeline continue para a próxima etapa.	De 50 minutos a 1 hora
3 - Firmware Update and Binary Copy	Nesta etapa é realizado o restante da configuração dos equipamentos, incluindo a atualização destes a partir da imagem gerada na Etapa 2. Em paralelo, é realizada a cópia de todos os binários gerados até o momento para um diretório específico de um servidor, a fim de facilitar a investigação em casos de falhas. No entanto, se algum job nesta etapa falhar, a execução do pipeline é interrompida.	De 13 a 15 minutos
4 - Test Execution	Nesta etapa é realizada a execução dos testes de aceitação e integração nos ambientes de testes que foram configurados nas etapas anteriores.	1 hora
5 - Copy of Test Results	Nesta etapa são coletados os resultados dos testes dos equipamentos e disponibilizados no job do pipeline que está em execução. Esta etapa é executada independente dos resultados obtidos na etapa anterior.	25 segundos
6 - Make the Delivery Official	Nesta etapa são coletados todos os binários e imagens gerados durante a execução e disponibilizados no servidor oficial de desenvolvimento. Posteriormente, os desenvolvedores são notificados sobre a conclusão da entrega. No entanto, esta etapa só é executada se todos os testes executados na Etapa 4 foram concluídos com sucesso.	De 8 a 10 minutos

Fonte: Autora

- **Etapa 2:** Nesta etapa o tempo pode variar bastante devido o processo de compilação de várias plataformas diferentes, sendo umas consideradas mais leves e outras mais pesadas. Normalmente a compilação é baseada em binários e artefatos já existentes, de forma que a plataforma é sempre executada na mesma máquina a fim de facilitar suas dependências, download de pacotes, etc. No entanto, existe uma configuração que permite escolher se a compilação será do zero ou não, o que implica uma variação de tempo de 40 minutos a 80 minutos a mais do normal.

- **Etapa 3:** Nesta etapa é configurado um timeout de 60 minutos para a atualização de firmware. Geralmente a atualização leva em torno de 15 minutos, podendo levar menos tempo em algumas máquinas. Entretanto, esse tempo pode atingir 60 minutos em casos de travamento da máquina ou indisponibilidade de rede, afetando assim o tempo total da entrega.

- **Etapa 4:** Nesta etapa é configurado um timeout de 1 hora e 30 minutos para a execução dos testes, entretanto, os times procuram escrever os testes com duração de 1 hora.

4.4 Considerações Finais

A documentação de um processo é de grande valor para dar visibilidade de como e porque funcionam de tal forma. Além de ajudar uma pessoa que necessite conhecer o processo, é útil para realizar análises, identificar gargalos, modificações, melhorias, inovação dentre outros fatores que envolvem o desenvolvimento de software. Vale ressaltar que é importante manter a documentação sempre atualizada para que ela não perca seu valor.

5 PROPOSTAS DE MELHORIAS

Neste capítulo são apresentadas as propostas de melhorias para o pipeline de entrega contínua de um software embarcado. O objetivo é remover possíveis gargalos que possam estar impedindo os desenvolvedores de obterem o feedback mais rápido.

1. *Melhoria para reduzir as ocorrências de falhas dos testes:*

   Contexto:

   Os testes executados no pipeline de entrega contínua necessitam de um ambiente mais robusto devido serem ambientes que executam testes automáticos para software embarcado. Esses ambientes são compostos por configurações automatizadas e configurações manuais. As configurações manuais consistem em preparar o ambiente fisicamente, onde é necessário disponibilizar a máquina já configurada com os requisitos básicos (placa de rede, memória, etc.) e também disponibilizar os dispositivos auxiliares que serão ligados a ela. Os dispositivos auxiliares são utilizados para simular uma conexão de rede entre os equipamentos. Já as configurações automatizadas consistem em preparar os ambientes virtuais, como provisionar a VM e atualizar o equipamento com a imagem de firmware gerada na Etapa 2 do pipeline. Após a preparação do ambiente como um todo, são executados os testes automáticos.

   Problema:

   Muitas vezes os testes falham por resquícios de execuções anteriores ou até mesmo por alguma intermitência no ambiente. Com isso, a execução da entrega resulta em falha e consequentemente atrasa a entrega atual e as demais entregas presentes na fila do pipeline. Contudo, o problema só aumenta, pois geralmente essas ocorrências necessitam do envolvimento da equipe de DevOps para parar a execução do pipeline (o que permite que nenhuma outra entrega execute) e também das equipes responsáveis pelos equipamentos para analisarem o teste que falhou e identificar a causa, o que geralmente se resolve com a reinicialização dos equipamentos.

   Proposta:

   Devido a complexidade que há no desenvolvimento de software embarcado e as peculiaridades do que são os equipamentos de testes, há poucas chances de garantir que a reinicialização dos equipamentos via *soft reset* seja suficiente para que o

equipamento volte ao seu estado normal. Neste caso, é necessário que esse tipo de evento seja realizado a nível de sistema operacional, em outras palavras, é garantir que esses eventos executem antes que a aplicação inicie. Todavia, existem alguns equipamentos que permitem ligar ou desligar dispositivos que são conectados a eles (equipamentos no qual a empresa já possui). Entretanto, necessitam de uma interação para realizar os comandos.

Uma sugestão de melhoria está relacionada à criação de um script que se conecte a este equipamento e através de comandos intereja com o mesmo. Para tal, num primeiro momento, será necessário a colaboração de algum desenvolvedor com conhecimentos adequados para realizar a criação do script. Em um segundo momento, será necessário que as equipes conectem seus equipamentos de testes a esses dispositivos. E por fim, dentro do escopo das automações do DevOps, esse script pode ser adicionado no pipeline de entrega contínua para ser executado de forma automática antes de iniciar os testes, garantindo assim, que todos os equipamentos de testes sejam reinicializados via *hard reset*, em outras palavras, seria executada a limpeza das memórias, caches entre outras configurações realizadas em execuções anteriores, resultando assim, que o equipamento volte ao seu estado normal e não interfira nas execuções de testes posteriores.

2. *Melhoria para reduzir o tempo de compilação:*

   Contexto:

   Para o desenvolvimento de software embarcado são utilizadas as linguagens de baixo nível C/C++ e arquivos de compilação. Além destes, são utilizados códigos e bibliotecas de terceiros, para os quais são mantidos uma cópia no sistema de controle de versão da empresa como uma forma de não precisar baixá-los de fora a cada uso e também para garantir que seja sempre o mesmo a ser utilizado. Toda a estrutura do desenvolvimento e suas dependências são definidas em módulos e salvos no sistema de controle de versão.

   Problema:

   Existem módulos que utilizam pacotes e binários que não são modificados com frequencia, como por exemplo os códigos e bibliotecas de terceiros. Mesmo não havendo modificações, a cada execução do pipeline de entrega, é recompilado todo o código fonte juntamente com suas depedências, sendo assim, acaba tornando a compilação mais lenta por se tratar de mais um passo a passo a ser compilado.

Proposta:

Levantar uma ação com o responsável de cada time de desenvolvimento para alterar a receita de compilação dos módulos que utilizam pacotes e binários que não são modificados, colocando-os na receita de forma que sejam baixados diretamente do servidor de artefatos ou do sistema de controle de versão ao invés de recompilá-los a cada execução. Além dessa ação, pode ser verificado com os arquitetos do projeto para analisarem alguma abordagem que otimize a compilação realizada pela ferramenta buildroot. Esta ação irá otimizar as compilações e consequentemente reduzirá o tempo total da execução da entrega.

3. *Melhoria para o processo de entrega:*

   É importante ressaltar que o pipeline de entrega contínua utilizado na empresa foi desenvolvido com o objetivo de atender integralmente o setor de Pesquisa & Desenvolvimento (P&D). Logo, o cliente final do pipeline de entrega não é necessariamente o cliente final da empresa, mas sim os clientes internos (outros setores) do setor de P&D. Em outras palavras, o conceito de entrega contínua, comumente conhecido por sempre ter uma versão candidata disponível ao cliente final, é utilizado pela empresa de forma que o cliente final seja, na verdade, um cliente interno, de modo que a entrega realizada no pipeline é apenas uma parte que compõe um pacote maior de artefatos que serão entregues às partes interessadas. No entanto, a criação do pacote de artefatos é um processo ainda composto por uma série de etapas manuais, e parte disso se deve ao fato de haver mais de um destino possível para a disponibilização de uma nova versão de firmware.

   Por se tratar de uma empresa que atua tanto no desenvolvimento de software embarcado quanto na fabricação e comercialização de equipamentos de telecomunicações, existem muitos processos que seguem uma hierarquia rígida em termos de permissões, setores e pessoas. Por isso, a presença de procedimentos manuais se mantém pelas peculiaridades dessas etapas finais e também, em parte, por questões de segurança que foram assumidas no início do projeto, quando este não estava totalmente maduro. Por esta razão, o setor de P&D acaba sendo a primeira parte da entrega.

   Contexto:

   Uma melhoria importante a ser avaliada está voltada ao processo de entrega. Atualmente, o pipeline de entrega contínua utilizado é um pipeline semi-automático.

Ao final de cada execução com sucesso, as imagens de firmware resultantes são disponibilizadas em um servidor de acesso restrito às equipes de P&D. Além destas equipes, há outras, de outros setores, que também necessitam utilizar essas imagens. Por exemplo, pode ser necessário atualizar a versão de firmware em equipamentos que possam estar na planta de um cliente (em produção), em desenvolvimento na linha de montagem, finalizados no estoque, em laboratórios sob testes específicos ou em demonstrações/treinamentos a clientes.

Contudo, o processo de disponibilização das imagens para os demais setores interessados não é totalmente automatizado, dependendo de passos manuais e de uma pessoa com permissões específicas para realizá-lo. No entanto, esse procedimento é realizado apenas ao final de cada versão, o que leva aproximadamente seis meses de desenvolvimento.

Conforme apresentado no Capítulo 4, a automação está presente apenas nos processos de compilação das imagens, no provisionamento dos ambientes virtuais e na execução dos testes. Entretanto, é possível diminuir as intervenções manuais presentes nos procedimentos finais das entregas, automatizando algumas das etapas e agilizando a maior parte dos subprocessos iniciados após a entrega da imagem de firmware no pipeline.

Problema:

Atualmente o pipeline de entrega é disparado de modo manual, quando um desenvolvedor submete o seu commit através de um job no Jenkins. Ao final de uma execução bem sucedida, os artefatos gerados (imagens e binários) são disponibilizados em um servidor de acesso comum ao setor de P&D. Esta é a primeira parte da entrega.

A segunda parte, executada apenas quando uma versão é considerada finalizada, consiste em oficializar o lançamento da versão para os demais setores que a utilizam. Para tanto, é executado um script que obtém todas as imagens das plataformas (referenciadas pelos modelos dos equipamentos) e as disponibilizam no diretório correspondente a cada plataforma no servidor oficial do projeto. No entanto, esse procedimento requer a realização de alguns passos manuais antes da execução do script como, por exemplo, a edição de um arquivo para alterar uma numeração que identifica que está sendo gerada uma nova versão e também indicação da tag (vinda do Git) sob a qual a geração do firmware está baseada.

Proposta:

Dado o contexto atual, as mudanças que gerariam melhorias na entrega do software estão relacionadas à forma como o pipeline é executado e como o firmware final é disponibilizado para os clientes internos (outros setores). A seguir, estão listadas as sugestões de melhoria.

- Disparar o pipeline automaticamente a cada commit no módulo principal de desenvolvimento (buildroot). Desta forma, removeria a etapa em que o desenvolvedor executa um job para submeter seu commit ao processo de entrega no pipeline. Mas isso se torna viável apenas se as melhorias 2 e 4 fossem implementadas com sucesso, pois ambas visam diminuir o tempo gasto por cada ciclo do pipeline, uma vez que o número de commits por dia poderia gerar um gargalo.
- Adicionar uma etapa no pipeline (ou, até mesmo, uma etapa que apenas disparasse a execução de um outro job) para gerar o pacote de artefatos automaticamente, disponibilizando-o no servidor de acesso comum aos demais setores interessados.

Em resumo, teria-se sempre uma versão candidata para os clientes internos e os processos manuais, que hoje tomam um certo tempo das pessoas responsáveis, seriam eliminados. Além disso, poderiam ser adicionados critérios para a liberação do pacote de artefatos como, por exemplo, liberar apenas para determinados setores (uma vez que o servidor possui uma hierarquia de locais e permissões bem definida). Assim, isso não comprometeria o desenvolvimento de software e não teria o risco de algum setor disponibilizar a versão a algum cliente externo sem esta ser decretada como final. No entanto, ao ser decretada a finalização da versão, seria necessário informar de alguma forma que todos os setores envolvidos poderiam já receber o pacote de artefatos.

4. *Melhorias para reduzir o tempo total de execução do pipeline de entrega:*

   - **Provisionamento das VMs:**

     Contexto: O ambiente de testes é considerado uma etapa complexa devido a necessidade de configurar e manipular as topologias de redes que envolvem máquinas físicas (computadores), máquinas virtuais (Virtual Machine - VM), equipamentos de rede sob teste e dispositivos auxiliares. Todos os elementos

constituintes desses ambientes são configurados antes dos testes e destruídos ao final do pipeline, com o objetivo de eliminar resquícios de execuções anteriores que podem influenciar nos resultados.

Problema: As VMs são utilizadas para isolar a execução dos testes que verificam os pacotes de rede e também para simular e controlar o tráfego de dados entre os equipamentos testados. Entretanto, as VMs são obtidas a partir de imagens naturalmente pesadas e lentas (dado que encapsulam um sistema operacional inteiro), o que leva a uma parcela de tempo considerável do tempo total do pipeline.

Proposta: Uma sugestão de melhoria está relacionada à troca das VMs por containers Docker. Isso tem uma grande chance de melhorar significativamente o tempo de provisionamento dos ambientes, uma vez que manipular um container (que encapsula um processo) é comumente mais rápido que manipular uma VM (RAINA, 2017). No entanto, é necessário que os arquitetos e especialistas responsáveis pelo firmware verifiquem o quão viável é realizar essa troca e se a utilização de containers Docker atendem a todos os requisitos necessários para a execução dos testes. Sabe-se que a troca de ferramentas não é uma tarefa simples, ainda mais quando as ferramentas utilizam estruturas diferentes e partem de paradigmas distintos. Por isso, é importante ressaltar que a utilização das VMs, no início do projeto, se deu pela sua capacidade em lidar com o tráfego de rede gerado e analisado durante os testes, o que inicialmente não foi possível com containers Docker. Contudo, cerca de sete anos após o início do projeto, houve uma série de evoluções dos containers e, por este motivo, esta sugestão de melhoria torna-se importante de ser avaliada novamente pelos responsáveis.

- **Artefatos:**

  Contexto: Ao final de cada execução bem sucedida do pipeline são gerados artefatos tais como binários e imagens de firmware, os quais são copiados para servidores específicos de uso do projeto. Cada plataforma tem seus binários e imagem em particular, o que resulta em muitos artefatos no geral. Todavia, sabe-se que é natural para o projeto que o número de plataformas cresça com a sua evolução.

  Problema: Conforme aumenta o número de plataformas em desenvolvimento no projeto, aumenta-se também o número de artefatos gerados e há o conse-

quente aumento de tempo para as operações de cópia de arquivos pela rede. Analisando o histórico de execuções do pipeline, é possível observar que o tempo de execução da Etapa 6 era de três minutos no início do projeto. Atualmente, a mesma etapa está levando cerca de dez minutos. Este tempo tende a aumentar conforme novas plataformas forem sendo agregadas ao projeto.

Proposta: A sugestão de melhoria é reduzir o tempo de cópia dos artefatos executados na Etapa 6 do pipeline. Para otimizar o tempo dessa cópia de arquivos, ao invés de executar apenas um job nessa etapa, levando a cópias sequenciais, pode-se criar um job para cada plataforma. Assim, as cópias dos artefatos poderiam ser paralelizadas, diminuindo o tempo total das cópias por consequência. Esta melhoria poderia ser realizada dentro do escopo das automações do DevOps. Outra abordagem que poderia ser realizada também é diminuir o tamanho das imagens de firmware, mas como uma ação a ser analisada pelos arquitetos do software.

- **Testes:**

  Contexto: A Etapa 4 do pipeline, em que há a execução dos testes de aceitação e integração, tem a duração aproximada de uma hora. Contudo, vale lembrar que este tempo não inclui o provisionamento dos ambientes de teste. Assim, o tempo de provisionamento somado ao tempo de testes introduz uma parcela de tempo de cerca de duas horas no pipeline.

  Problema: Conforme a equipe e os testes que esta desenvolve, o timebox de uma hora da Etapa 4 pode ou não ser totalmente utilizada dentro dos seus jobs. Porém, se pelo menos um ambiente utilizar o *timebox* inteiro, todos os demais ficam presos a esse tempo, visto que todos são executados em paralelo.

  Proposta: A sugestão de melhoria é propor aos times responsáveis pelos testes que realizem uma análise destes e avaliem a diminuição do *timebox* da Etapa 4, reduzindo o tempo de execução dos testes. A seguir, algumas sugestões para serem avaliadas.

  - verificar se não há algum teste defasado (sem otimizações) ou desnecessário (optando somente pelos testes essenciais);
  - identificar os testes que podem ter paralelismo (isto é, aqueles que não interferem um no outro);
  - e por fim, verificar se há a possibilidade de extrair configurações que os

testes fazem e encapsular isso no ambiente na forma de imagens criadas fora do pipeline (imagens das VMs ou Docker), buscando realizar o mínimo de etapas de configuração/provisionamento durante os testes, mantendo apenas as dinâmicas essenciais.

5.1 Considerações Finais

Este capítulo apresentou o conceito de como a entrega contínua é realizada na empresa e também apresentou algumas propostas de melhorias para o pipeline de entrega contínua de um software embarcado. Essas melhorias podem ser realizadas de médio a longo prazo. O principal efeito que elas causam são a diminuição do tempo total de execução do pipeline e na eliminação de alguns processos manuais. Algumas destas melhorias podem ser realizadas dentro do escopo de automações do DevOps e outras necessitam do auxílio dos arquitetos e analistas responsáveis pelo projeto. Todavia, as sugestões de melhorias visam obter um feedback mais rápido da entrega, tanto para as falhas quanto para o sucesso.

6 CONCLUSÃO

Com a visibilidade dos métodos ágeis crescendo no mercado, a entrega de software ganhou uma exigência de ser cada vez mais frequente e com qualidade agregada. Com isso, o movimento DevOps vem sendo adotado por inúmeras organizações para auxiliar nesse processo devido seus benefícios no que tange o desenvolvimento de software. Sabemos que não é fácil disponibilizar um software ou parte de um software continuamente e com um alto nível de qualidade. No entanto, as práticas de DevOps como a integração e entrega contínua são essenciais para realizar todo esse processo de forma simples e objetiva.

O trabalho apresentou o quão importante é ter e manter uma documentação atualizada e disponível do processo de entrega contínua. A documentação serve tanto para deixar registrado como o processo funciona e assim auxiliar no entendimento de qualquer pessoa interessada quanto para melhorar os procedimentos que envolvem a entrega contínua. Para tanto, este trabalho teve como foco a criação da documentação do pipeline de entrega contínua de um software embarcado e, posteriormente, uma análise desta documentação para propor algumas melhorias.

A documentação realizada sob o pipeline foi criada com alto nível de abstração, cujo objetivo é disponibilizá-la de forma clara para todos da equipe de P&D da empresa-alvo. Desta forma, a documentação foi realizada em três partes, sendo uma delas, as propostas de melhorias. Inicialmente foram identificadas as etapas que compõem o pipeline de entrega contínua, bem como suas configurações e estrutura. Consequentemente, a segunda parte foi documentar esse levantamento e por último foi realizada uma análise e proposto melhorias. Para trabalhos futuros, as propostas de melhorias foram apresentadas para a equipe de automações da empresa-alvo e após serem avaliadas e aprovadas, foi possível identificar a viabilidade de serem aplicadas a médio e longo prazo.

Em tempo, as lições aprendidas durante o trabalho foram que quanto mais rápido for o feedback, melhor será o tempo hábil para a correção da falha. Contudo, para que isso aconteça, é necessário que os processos estejam em constante melhoria.

REFERÊNCIAS

AQUILES, A.; FERREIRA, R. **Controlando versões com Git e Github**. [S.l.]: Casa do Código, 2014.

FRõES, G.; WEBER, V. **Sistemas Embarcados (Embedded Systems)**. 2020. Acesso em: 01-02-2021. Available from Internet: <https://www.youtube.com/watch?v=XppU8kKpa6I>.

HUMBLE, J. **Continuous Delivery vs Continuous Deployment**. 2010. Acesso em: 23-01-2021. Available from Internet: <https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>.

HUMBLE, J.; FARLEY, D. **Entrega Contínua: Como Entregar Software de Forma Rápida e Confiável**. [S.l.]: Bookman, 2014.

LEE, K. S. **DevOps, O Que É? Metodologia, Benefícios E Ferramentas!** 2018. Acesso em: 19-01-2021. Available from Internet: <https://mundodevops.com/blog/o-que-e-devops/>.

MINUDEL, L.; MORRIS, K. **Continuous Delivery Overview**. 2014. Acesso em: 24-01-2021. Available from Internet: <https://www.infoq.com/minibooks/continuous-delivery-overview>.

ORG, B. **The Buildroot user manual**. 2021. Acesso em: 29-05-2021. Available from Internet: <https://buildroot.org/downloads/manual/manual.html#_about_buildroot>.

OSSADA, J. C.; MARTINS, L. E. G. Um estudo de campo sobre o estado da prática da elicitação de requisitos em sistemas embarcados. 2010.

RAHO, M. et al. Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing. In: **2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)**. [S.l.: s.n.], 2015. p. 1–8.

RAINA, A. S. **DOCKER VS. VIRTUAL MACHINE: WHERE ARE THE DIFFERENCES?** 2017. Acesso em: 19-07-2021. Available from Internet: <https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/>.

REIS, F. dos. **Introdução aos Sistemas Embarcados**. 2015. Acesso em: 01-02-2021. Available from Internet: <https://www.youtube.com/watch?v=1I3QKMzSXUM>.

SATO, D. **DevOps na prática: entrega de software confiável e automatizada**. [S.l.]: Casa do Código, 2014.

WILLIS, J. **What Devops Means to Me**. 2010. Acesso em: 09-07-2021. Available from Internet: <https://blog.chef.io/what-devops-means-to-me>.