CENTRO UNIVERSITÁRIO CHRISTUS

SISTEMAS DE INFORMAÇÃO

PEDRO HENRIQUE OLIVEIRA LIMA

ESTUDO COMPARATIVO SOBRE AS FUNCIONALIDADES DE INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA DAS FERRAMENTAS JENKINS E GITLAB

FORTALEZA

2021

PEDRO HENRIQUE OLIVEIRA LIMA

ESTUDO COMPARATIVO SOBRE AS FUNCIONALIDADES DE INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA DAS FERRAMENTAS JENKINS E GITLAB

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Sistemas de Informação do Centro Universitário Christus, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Orientadora: Profª. Ms. Amanda Oliveira de Sousa

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Centro Universitário Christus - Unichristus
Gerada automaticamente pelo Sistema de Elaboração de Ficha Catalográfica do
Centro Universitário Christus - Unichristus, com dados fornecidos pelo(a) autor(a)

L732e Lima, Pedro Henrique Oliveira.
ESTUDO COMPARATIVO SOBRE AS FUNCIONALIDADES DE INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA DAS FERRAMENTAS JENKINS E GITLAB / Pedro Henrique Oliveira Lima. - 2021.
55 f. : il. color.

Trabalho de Conclusão de Curso (Graduação) - Centro Universitário Christus - Unichristus, Curso de Sistemas de Informação, Fortaleza, 2021.
Orientação: Profa. Ma. Amanda Oliveira de Sousa.

1. Integração Contínua. 2. Entrega Contínua. 3. Jenkins. 4. GitLab. 5. Pipelines. I. Título.

CDD 005

PEDRO HENRIQUE OLIVEIRA LIMA

ESTUDO COMPARATIVO SOBRE AS FUNCIONALIDADES DE INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA DAS FERRAMENTAS JENKINS E GITLAB

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Sistemas de Informação do Centro Universitário Christus, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Profª. Ms. Amanda Oliveira de Sousa (Orientadora)
Centro Universitário Christus (Unichristus)

Prof. Ms. Euristenho Queiroz de Oliveira Júnior
Centro Universitário Christus (Unichristus)

Prof. Ms. Tiago Guimarães Sombra
Centro Universitário Christus (Unichristus)

AGRADECIMENTOS

Primeiramente, agradecer a Deus pela saúde e força para superar as dificuldades apresentadas durante a construção do trabalho.

Aos meus pais, Sandra Maria e Carlos José, pelo apoio, incentivo, ensinamentos e sacrifícios durante toda minha jornada.

Aos companheiros de trabalho e amigos Diego Mello e Marcelo Costa pela paciência, incentivo e suporte prestados durante a construção da obra.

A Professora Amanda Oliveira, pela paciência, conhecimento fornecido e disponibilidade prestada durante a orientação.

A todos que participaram de maneira direta ou indireta da minha formação, o meu muito obrigado.

"O sucesso é construído de 99% de fracasso."

(Soichiro Honda)

RESUMO

Como consequência da evolução tecnológica o nível de complexidade das soluções de *softwares* e a demanda gerada por seus clientes estão aumentando, nesse contexto, o *DevOps* se apresenta como solução para otimizar os processos de desenvolvimento e entrega *softwares*, tornando-os mais rápidos. O termo *DevOps* foi criado por Patrick Debois em 2009 e consiste na ideia de união de dois segmentos que são: Desenvolvimento *(Dev)* e Operações *(Ops)*. Para a união desses segmentos é necessário um conjunto de práticas com a finalidade agilizar o processo de desenvolvimento e entrega dos *softwares*, dentre essas práticas temos: a Integração Contínua e a Entrega Contínua. A Integração Contínua (CI) volta-se para o lado do desenvolvimento, propõe a integração de novas funcionalidades ou correção de erros várias vezes ao versionador de código fonte, com o objetivo de melhorar a aplicação. A Entrega Contínua (CD) propõe a entrega do artefato no ambiente do usuário final de maneira mais rápida e automatizada, mas para que seja possível o uso dessas práticas, podemos destacar os *pipelines* de CI/CD que são importantes, pois são eles os responsáveis pela automação de processos repetíveis. Nesse contexto, ferramentas são essenciais para a execução dos processos que envolvem o *DevOps* e consequentemente a CI e a CD. Dentre as ferramentas existentes no mercado, O *Jenkins* e o *GitLab* ganham destaque, pois são as mais utilizadas segundo o *Google Trends*. O objetivo desse trabalho é realizar uma comparação das ferramentas *Jenkins* e *GitLab* nos quesitos CI e CD com a finalidade de apresentar qual ferramenta pode ser utilizada em determinados contextos. Para realizar a comparação foram levantados cinco critérios de avaliação e também serão implementados seus *pipelines* na prática.

Palavras-chave: *DevOps*; Integração Contínua; Entrega Contínua; *Pipelines*; *Jenkins*; *GitLab*.

ABSTRACT

As a result of technological evolution, the level of complexity of software solutions and the demand generated by its customers are increasing, in this context, DevOps presents itself as a solution to optimize software development and delivery processes, making them faster. The term DevOps was created by Patrick Debois in 2009 and consists of the idea of joining two segments that are: Development (Dev) and Operations (Ops). For the union of these segments, a set of practices is necessary in order to streamline the software development and delivery process, among these practices we have: Continuous Integration and Continuous Delivery. CI turns to the development side, proposes the integration of new features or correction of errors several times to the source code versioner, with the objective of improving the application. CD proposes the delivery of the artifact in the end-user environment in a faster and more automated way, but in order to use these practices, we can highlight the CI/CD pipelines that are important, as they are responsible for process automation repeatable. In this context, tools are essential for the execution of processes that involve DevOps and, consequently, CI and CD. Among the tools on the market, Jenkins and GitLab are highlighted, as they are the most used according to Google Trends. The objective of this work is to carry out a comparison of the Jenkins and GitLab tools in terms of CI and CD in order to present which tool can be used in certain contexts. To carry out the comparison, five evaluation criteria were raised and their pipelines will also be implemented in practice.

Keywords: DevOps; Continuous Integration; Continuous Delivery; Pipelines; Jenkins; GitLab.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CD Entrega Contínua
CI Integração Contínua

SUMÁRIO

1 INTRODUÇÃO

As organizações do âmbito da tecnologia da informação, especialmente do mercado de desenvolvimento de *software*, estão enfrentando grandes desafios, dentre eles destaca-se, a necessidade de adequar constantemente os seus sistemas para atender as demandas e mudanças geradas por seus clientes, fornecendo soluções de *softwares* de forma cada vez mais rápida, ágil, multiplataforma e carregando consigo um alto índice de disponibilidade e confiabilidade (HUMBLE; MOLESKY, 2011).

Com a missão de endereçar tais necessidades, as empresas estão ficando cada vez mais sobrecarregadas, pois, além dos sistemas desenvolvidos por elas serem mais complexos, entregá-los com mais rapidez exige a simplificação e automatização de alguns processos, pois os processos manuais geram gargalos e consequentemente atrasos no desenvolvimento e na entrega dos produtos de *software* (HUMBLE; MOLESKY, 2011).

O formato em que as organizações mais tradicionais trabalham para garantir o desenvolvimento e entrega de suas aplicações, parte de dois segmentos historicamente mais separados, são eles: I) o segmento de desenvolvimento *(Dev)*, responsável pela criação, testes e manutenção do código e geração de *releases* e II) o segmento de operações *(Ops)*, que arca com a parte de implantação e monitoramento da aplicação no ambiente de produção do cliente. Como consequência dessa separação, o processo de implantação ou atualização de um sistema se torna bastante traumático, pois dependendo das possíveis limitações existentes, pode ocorrer um atraso do processo, como limitações de *hardware*, ou seja, no momento em que a aplicação necessitar de algum recurso além do existente, até que esse problema seja resolvido, o processo de implementação ou atualização pode levar meses para ser concluído (VALENTE, 2020).

Portanto, para endereçar os desafios e solucionar os problemas gerados pela separação destes segmentos *(i.e., Dev e Ops)*, surgiu a ideia de aproximá-los e assim garantir mais rapidez na entrega e no monitoramento das aplicações, bem como para prevenir possíveis falhas. A união desses setores ficou conhecida como um movimento chamado de *DevOps*, e passou a ser promovido e adotado como uma cultura em diversas organizações (VALENTE, 2020) (HUMBLE; MOLESKY, 2011).

Em uma abordagem *DevOps*, diversas práticas são fundamentais para a automatização de processos repetíveis, dentre essas práticas destacam-se, a CI que está vinculada mais aos desenvolvedores, contudo devido a essa prática se permite a acelerar o processo de desenvolvimento. A CI ataca principalmente os processos de interações com o versionador

de código fonte dos projetos, permitindo *commits* e *merges* várias vezes ao dia, eliminando a necessidade de requerer um dia específico para realizá-los. Além disso, inclui-se a criação de testes automatizados de verificação e validação do que foi desenvolvido. Outra prática em destaque é a CD que funciona como uma extensão da CI, a qual é direcionada ao setor de operações, fornecendo soluções e uso de ferramentas que automatizem e otimizem o tempo na parte de entrega do *software* no ambiente do cliente, seja o de validação ou o ambiente produtivo (PUROHIT, 2020) (PITTET, 2021).

Para fazer o uso das práticas de Integração Contínua e Entrega Contínua, torna-se necessário o uso de ferramentas que possam auxiliar os processos de desenvolvimento e entrega do *software*, tornando-os mais ágeis. Entre as ferramentas disponíveis para a apoiar os processos de CI/CD, este estudo destaca duas ferramentas populares no mercado, a saber: *Jenkins* e o *GitLab*. O *Jenkins* é uma ferramenta de automação em código aberto, que possui diversas funcionalidades relacionadas a construção, teste e entrega de *software*, oferecendo *plugins* que auxiliam nessas funcionalidades (JENKINS, 2021a). O *GitLab* é uma plataforma dedicada ao *DevOps*, capaz de unir os processos em uma única aplicação, pois além de ser um versionador de código baseado em *Git*[1]. , é referência em processos que são populares nas práticas *DevOps*, como *pipeline* CI/CD, que constrói, testa e entrega o *software*(GITLAB, 2021b).

Para apresentar indícios da popularidade das ferramentas que são discutidas no presente trabalho, foi verificado no *Google Trends* [2] que as ferramentas *Jenkins* e *GitLab*, as quais estão no topo das mais utilizadas em todo o mundo entre Setembro e Novembro de 2021, se comparadas com outras ferramentas, como o *Travis CI* e *Azure DevOps Server*.

Segundo Hüttermann (2012) quanto mais ferramentas estiverem no leque de possibilidades de escolha da organização, maior serão as chances de selecionar a mais adequada para determinados contextos. Com base nisso, trazendo para a perspectiva das ferramentas da CI/CD não é diferente, pois é importante também conhecer as ferramentas para determinados processos (e.g., Integração Contínua e Entrega contínua, etc.) para a adoção das ferramentas que melhor atendem as necessidades do processo como um todo. Portanto, ao adotar uma nova ferramenta as equipes devem se perguntar: Como avaliar e escolher uma ferramenta que dá suporte aos processos de CI/CD no contexto de seu projeto?

---

[1] *Git* é uma ferramenta de controle de versão capaz de lidar com pequenos e grandes projetos. Disponível em: https://git-scm.com/. Acesso em: 17 de Outubro de 2021

[2] *Google Trends* é uma ferramenta que fornece acesso as amostras de pesquisas reais do *Google*. Disponível em: https://support.google.com/trends/answer/4365533?hl=pt. Acesso em: 17 de Outubro de 2021

1.1 Objetivo

O objetivo do presente trabalho é apresentar a presença ou ausência de funcionalidades importantes relacionadas a Integração Continua e Entrega Contínua das duas ferramentas mais populares no mercado *(e.g., Jenkins e GitLab)* por meio de um estudo comparativo.

1.2 Objetivos Específicos

- Apresentar o conceito de *DevOps*, Integração Contínua e Integra Contínua;
- Apresentar as ferramentas *Jenkins* e *GitLab*;
- Definir e apresentar os critérios comparativos;
- Realizar a avaliação de ambas as ferramentas;
- Analisar e apresentar os resultados obtidos da comparação.

1.3 Justificativa

O *DevOps* é proposto para otimizar o tempo dentro do ciclo de vida do desenvolvimento da aplicação, fornecendo métodos e ferramentas desde a fase de desenvolvimento até a fase de entrega do produto para o cliente final. Com base nisso, o *DevOps* é utilizado amplamente por diversas equipes envolvidas no ciclo de desenvolvimento da aplicação, pois através dele é possível garantir a entrega antecipada e contínua do *software*, fazendo uso de ferramentas responsáveis por gerenciar *pipelines* CI/CD (REDHAT, 2021). Portanto, o presente trabalho é destinado à profissionais e estudiosos da área de *DevOps*, principalmente relacionados aos segmentos de Integração Contínua e Entrega Continua, servindo como consulta para auxiliar na escolha entre as ferramentas *Jenkins* e *GitLab* em um determinado contexto.

1.4 Estrutura

O presente trabalho está estruturado da seguinte forma: O Capítulo 2 apresentamos o que é o *DevOps*, seu ciclo, seus benefícios e também as práticas de Integração e Entrega Contínua e a importância de seus *pipelines* e ferramentas. Ainda no Capítulo 2, é descrito o que é o *Jenkins* e *GitLab* e suas características. No Capítulo 3 apresentamos os trabalhos correlacionados com o presente trabalho, com a finalidade de mostrar suas principais diferenças e semelhanças. No Capítulo 4 apresentamos a definição dos critério e a avaliação das ferramentas

Jenkins e *GitLab*, expondo suas semelhanças e diferenças de forma comparativa. No Capítulo 6 são expostos os resultados da comparação baseando-se nos critérios adotados. Pro fim, no capítulo 7, apresentamos as conclusões do presente trabalho e novas oportunidade de trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os conceitos base do que serão estudados neste trabalho, como a definição de *DevOps*, suas principais características, fluxos de trabalho e seu ciclo de vida. Ao decorrer deste, também são apresentados os conceitos de CI e CD e como funcionam dentro de uma organização, destacando as principais ferramentas relacionadas existentes no mercado, em especial, as ferramentas *Jenkins* e *GitLab*.

2.1 Separação Desenvolvimento e Operações

As organizações no mercado de desenvolvimento de *software* que trabalham de maneira mais tradicional, que não implementam culturas que prezam por agilidade como o *DevOps*, são constituídas historicamente com a separação dos seguintes seguimentos (HÜTTERMANN, 2012):

- **Desenvolvimento**: Equipe que é composta pelo time de programadores, aqueles que escrevem e mantém o código da aplicação;
- **Testes**: Equipe responsável por uma parte específica do projeto, entram em ação logo após o time de desenvolvimento encerrar suas atividades e cuidam principalmente da validação e verificação do código escrito pelo time de desenvolvedores;
- **Operações**: Equipe responsável pela última etapa do ciclo de um software, os principais papéis pertencentes a esse time são: administradores de sistemas, banco de dados, redes, dentre outros administradores que tem como principal objetivo instalar e manter o servidor onde o sistema está implementado e o sistema propriamente dito. Historicamente, a equipe de Operações é a que trabalha de maneira mais afastada que as demais equipes.

Segundo Hüttermann (2012), a utilização de modelos tradicionais e com times segregados pode causar barreiras organizacionais e culturais tais como:

- **Equipes Isoladas**: Times com essa característica, tendem a defender e buscar interesses individuais ou somente do grupo no qual o individuo é pertencente, ao invés de defender interesses do projeto como um todo;
- **Falta de uma linguagem única entre os times**: Equipes separadas, tendem a possuir termos cujo o entendimento é somente de um grupo específico. O ideal segundo Hüttermann (2012), é a criação de uma linguagem única e compartilhada entre os times, inclusive entre

os clientes da organização, pois com a adição de uma linguagem única, é possível trabalhar de maneira mais harmônica, sem termos exclusivos entre os setores e incluir ainda mais o cliente para o interior da organização;

- **Ausência de entendimento entre as equipes**: São equipes que tendem a olhar com desconfiança o que pessoas de outros times fazem, por medo que atividade realizada possa afetar o modo de trabalho adotado por elas. Por consequência, o compartilhamento de conhecimento fica cada vez mais difícil de ser realizado, poisas pessoas ficam acomodadas.

Nesse caso, a adoção de uma cultura que visa melhorar a relação desses times poderia otimizar o trabalho das mesmas e orientando-as a alcançar um objetivo em comum, bem como auxilia na propagação do conhecimento na organização como um todo.

2.2 *DevOps*

Segundo Ebert *et al.* (2016), o termo *DevOps* representa o desenvolvimento e a implementação cada vez mais rápida e flexível, unindo dois segmentos da tecnologia da informação tradicionalmente mais afastados. Sua origem foi uma consequência da implementação de práticas ágeis no desenvolvimento de software, e portanto compartilha de um pensamento bastante conhecido, existente a mais de um século, chamado *Lean* (EBERT *et al.*, 2016). Tal pensamento tem como principal filosofia a eliminação de itens indesejados ou processos desnecessários, a fim de torná-los mais ágeis possíveis e consequentemente evitar desperdício de recursos. (RAVICHANDRAN *et al.*, 2016).

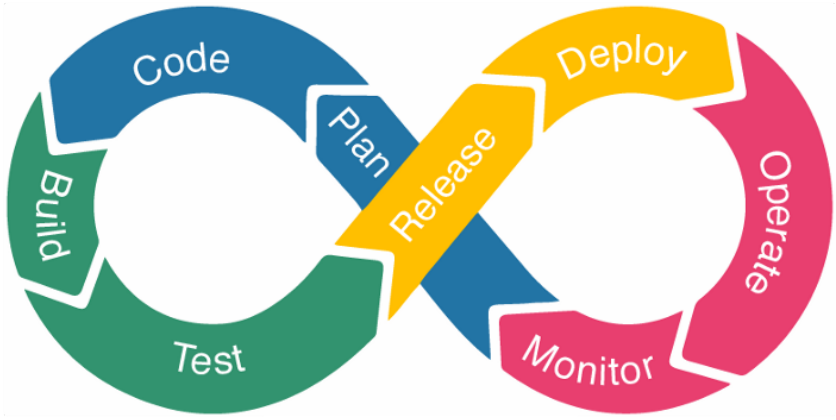
Sendo uma junção dos termos Desenvolvimento e Operações, o termo *DevOps* foi criado em 2009 por Patrick Debois, que o definiu como um conjunto de práticas, metodologias, técnicas e mentalidade para contribuir para automação e integração entre as atividades relacionadas ao desenvolvimento e operações. Nesse sentido, entre os benefícios proporcionados pela prática do *DevOps*, pode-se destacar a redução do ciclo do *software*, graças a implementação de testes automatizados, atualizações contínuas e *feedbacks* mais rápidos (CHATTERJEE, 2021).

2.3 O Ciclo *DevOps*

O ciclo *DevOps* possui uma série de etapas, cada uma delas significa recursos, procedimentos e ferramentas distintas, mas, quando unidas, servem para acelerar os processos

desde o desenvolvimento até a utilização do usuário final.

Figura 1 – *DevOps Life Cycle.*



Fonte: Yıldırım (2019)

A Figura 1 retrata como é composto o ciclo *DevOps*, mostrando todas as etapas e como os processos, tanto na parte de desenvolvimento quanto na parte de operações, podem se unir e trabalhar de maneira harmônica, ajudando na aceleração do desenvolvimento e na entrega do *software*. Bass *et al.* (2015), em seu livro, destacou que o ciclo *DevOps* é definido principalmente pelos seguintes processos que também estão ilustrados na Figura 1:

1. **Planejamento:** Se caracteriza pela coleta de requisitos dos clientes, interações e atualizações;
2. **Desenvolvimento**: Tudo que é levantado de requisitos na etapa de Planejamento é transformado em código por meio de uma determinada linguagem de programação escolhida pela equipe de desenvolvimento. Nessa etapa, é bastante comum que diversas interações sejam realizadas no versionador de código-fonte. Ainda nesta etapa também são realizados diversos testes unitários no momento da compilação para a construção dos artefatos;
3. **Verificação**: Processo onde verifica-se o código através de testes automatizados, sejam eles durante ou após o desenvolvimento;
4. **Teste**: Etapa de verificação da qualidade do artefato gerado, ou seja, se o que foi produzido atende todos os requisitos solicitados pelo cliente. Nessa etapa se caracteriza por um lançamento de uma versão no ambiente de teste do cliente com o objetivo de ser validada pelo mesmo;
5. **Implementação**: Após o cliente validar a aplicação no ambiente de testes, a etapa visa a publicação do *software* no ambiente produtivo;
6. **Operação**: Esta etapa visa melhorar o ambiente onde a aplicação está implementada

em produção, nesse sentido, procedimentos como planejamento de escalonamento das máquinas automaticamente, planejar como aplicação vai ser executada, juntamente com seu módulos e dependências são importantes para manter o sistema sempre disponível;

7. **Monitoramento**: Etapa de coleta de *logs* para geração de relatórios de performance, com o intuito de melhorar continuamente a aplicação e assim prevenindo e combatendo o mais rápido possível o número de falhas que podem ser geradas pelo *software*. Esse processo é importante, pois é nele que o planejamento terá como base do que será posto para ser executado no próximo ciclo.

2.4 Implementação do *DevOps*

Um dos maiores desafios para as empresas é buscar um padrão para implementar o *DevOps*. Alguns profissionais relatam que é intencional essa maneira de agir, dessa forma o *DevOps* se adapta a qualquer tipo de ambiente. Apesar de não ter um padrão definido, é necessário ter o entendimento dessa metodologia que pode ser dividida em cinco (5) componentes (WIEDEMANN *et al.*, 2019):

- **O Componente Cultural**: Fazer com que o ambiente organizacional prese primeiramente pela confiança entre os seus integrantes, vontade de aprender, melhorar constantemente e manter a mente aberta à mudanças vindas do desenvolvimento ou operações;
- **Componente de Automação**: Se baseia na criação de *pipelines* de automação, principalmente em processos de Integração e Entrega Contínua;
- **Componente de Simplificação**: Tem o intuito de melhorar e otimizar os processos, tanto de desenvolvimento quanto em operações, evitando gargalos que possam dificultá-los, melhorando a eficiência dos mesmos;
- **Componente de Medição**: Trata das medições e monitoramento das métricas do sistema e também os seus indicadores de desempenho;
- **Componente de Compartilhamento de Conhecimento**: Membros devem se comunicar e compartilhar conhecimento de maneira proativa, para além do setores onde estão inseridos.

Logo após a apresentação da metodologia, parti-se para as práticas *DevOps*, que irão impactar de maneira positiva a cultura da organização, principalmente em processos de desenvolvimento e entrega de um *software*. No entanto, por mais que essas práticas sejam

positivas, não existe um manual de como aplicá-las, pois serão aplicadas de acordo com o perfil e contexto de cada organização (TOTVS, 2021). As práticas do *DevOps* são bem populares, entre elas destacam-se a Integração Contínua, Entrega Contínua, Infraestrutura como código, Microsserviços e Centralização de *Logs* (AWS, 2021). No corrente são evidenciados os processos de Integração e Entrega Contínua.

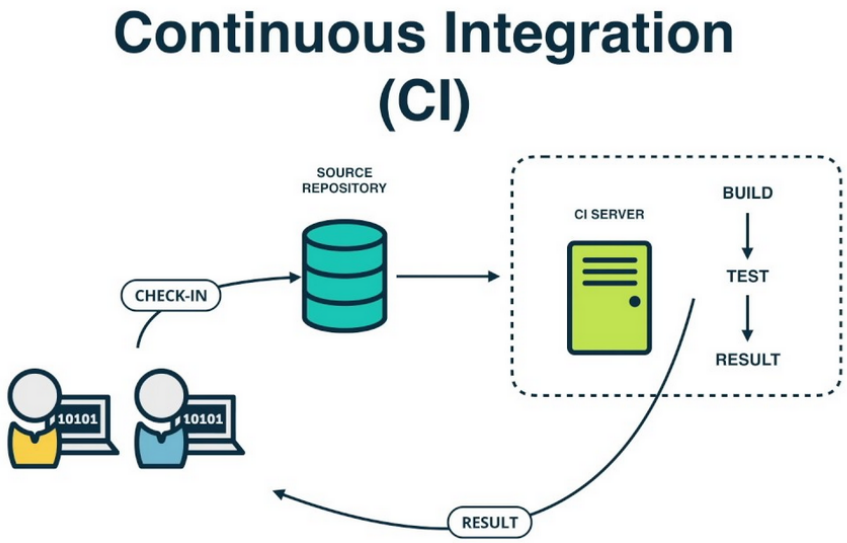
2.4.1 Integração Contínua (CI)

Antes da Integração Contínua, era bem comum os desenvolvedores perderem bastante tempo para confirmar suas alterações na *branch master* do projeto, esse procedimento deveria ser combinado entre eles para que não ocorressem muitos problemas. Como consequência dessa prática, além da perda de tempo devido a dificuldade de mesclar o código, era o acúmulo de *bugs* sem correção ao longo do desenvolvimento do sistema, que atrasavam a entrega do *software* para o cliente (MOHAMMAD, 2016).

Como solução deste problema surgiu a Integração Contínua, que é uma prática bastante conhecida no segmento *DevOps*, e tem sua origem ligada a metodologia ágil *Extremme Programming* (XP) (MEYER, 2014). A Integração Contínua é definida como uma prática concebida no *DevOps* destinada ao desenvolvimento, que visa várias vezes durante o dia a integração dos trabalhos realizados pelos desenvolvedores de forma rápida e ágil. Antes que cada integração seja realizada, é efetuado um processo de construção, o qual trabalha também com os testes automatizados, e assim, acaba facilitando a detecção de erros na integração de forma mais eficiente. Equipes de desenvolvimento destacam a redução significativa de erros, graças ao processo de Integração Contínua (FOWLER; FOEMMEL, 2006).

A Figura 2 apresenta como é realizado o processo de Integração Contínua no desenvolvimento, que tem como sua base o compartilhamento de um repositório central, pois nele possui uma linha principal chamada de *master*, este deve ser bastante protegida e pouco alterada, contudo se a alteração for realizada diretamente nela, é recomentado que seja de pequeno impacto, pois a partir da linha principal é o que vai ser entregue para o ambiente do cliente. As alterações de maior relevância no projeto, devem serem feitas em ramificações baseadas a partir dela, que são chamadas de *branches* (MEYER, 2014).

Figura 2 – Ciclo de Integração Contínua.



Fonte: PrimeControl (2019)

Após realizadas as alterações no código, o servidor de Integração Contínua é encarregado de capturar essas alterações e realizar o *build* e os testes automatizados da aplicação. Nesse passo, o aplicativo é construído e também são executados os testes definidos pelos desenvolvedores ou analistas de teste para saber o funcionamento da aplicação, no final, serão apresentados os resultados de suas alterações, se foi aprovado ou não nos testes definidos, em algumas configurações o artefato já pode ser entregue após os testes no ambiente do cliente, tanto de homologação, quanto o de produção (MOHAMMAD, 2016). É de grande importância que esse processo seja contínuo e repetível e altamente automatizado em todo o ciclo de desenvolvimento (VIRMANI, 2015).

Com todos os benefícios da Integração Contínua, existem algumas hábitos que dificultam a ampla difusão do processo, tais como: a não inclusão de todas as alterações para o repositório, devido a Integração Contínua iniciar sua execução a partir de um repositório, ou seja, incluir todas as alterações no repositório é de fundamental importância para a base do processo; falta de correção imediata do código caso ocorra algumas falhas nos testes, porque se o código não for corrigindo rapidamente podem ocorrer atrasos no projeto; Adoção de testes a cada interação com o repositório, testes mais completos são importantes, mas teste mais rápidos a partir das interações, geralmente testes unitários, fornecem um reposta mais rápida para o desenvolvedor (CLOUD, 2021).

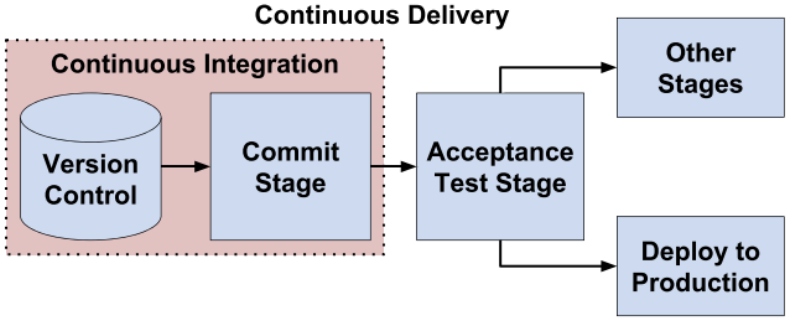
2.4.2 Entrega Contínua (CD)

Humble e Farley (2010) explicam em seu livro que, atualmente os aplicativos modernos possuem um alto grau de complexidade e por isso muitas organizações optam por entregá-los ao cliente final manualmente, separando por etapas, cada uma dessas etapas fica de responsabilidade a um individuo ou equipe. Por se tratar de um procedimento manual, a entrega fica sujeita a erros humanos, ocorrendo falhas na implantação. Empresas que optam por essa forma de entrega, enfrentam alguns problemas, tais como (HUMBLE; FARLEY, 2010):

- Extensas e árduas documentações devido ao processo ser fragmentado e engessado, possuindo várias ferramentas para cada etapa de entrega;
- Confiar em testes manuais, para certificar que a aplicação está funcionando;
- Contato extremo com a equipe de desenvolvedores para explicar o porque que a aplicação não foi implantada;
- Correções de problemas durante a implantação do aplicativo;
- Versões que podem gerar falhas não previstas, muitas vezes tendo que ser revertidas;
- Muito tempo de espera após implantação ter sido realizada, verificando se o ambiente está operando correntemente, ou caso não esteja, custando muito tempo tentando descobrir a falha.

Criada em 2010 e oriunda da Engenharia de *Software*, a Entrega Contínua tem o papel de solucionar os problemas com a entrega dele para produção do cliente (CHEN, 2015). Esses problemas são solucionados graças a capacidade da Entrega Contínua fornecer e tornar o processo de entrega do *software* mais simplificada, repetível, confiável, previsível e automatizada. Os principais benefícios trazidos por essa prática *DevOps*, além de tornar essa etapa mais eficiente, é a satisfação do usuário final, devido a correção de *bugs* e falhas da aplicação ser fornecidas rapidamente no ambiente produtivo (HUMBLE; FARLEY, 2010).

Figura 3 – Relação entre Integração Contínua e Entrega Contínua.



Fonte: Laukkanen *et al.* (2017)

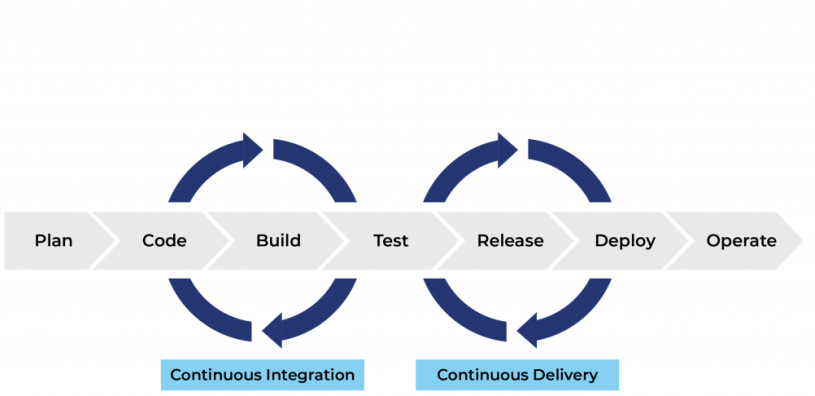
Na Figura 3, é apresentada a relação entre a Integração Contínua e a Entrega Contínua. A Entrega Contínua é uma extensão da Integração Contínua dedicada na geração de novas versões para serem implementadas em um repositório, e logo após ser implementado no ambiente do usuário final. (ITKONEN *et al.*, 2016).

Existe uma diferença entre os termos, CD de *Continuos Delivery* e CD de *Continuos Deployment*, apesar de ambos serem automatizados, a Entrega Contínua permite a cada interação feita no repositório, a *release* já é destinada ao repositório, mas ainda necessita que a implementação seja manual no ambiente do usuário final, diferentemente da Implementação Contínua, que necessita que o *release* seja implementado rapidamente em produção. (VALENTE, 2020).

2.4.3 Pipeline CI/CD

Um *pipeline* CI/CD é uma prática da Integração e Entrega Contínua que consiste em uma serie de passos para a disponibilização de um software, que auxiliam desde o desenvolvimento até a entrega dele para o cliente (REDHAT, 2020).

Figura 4 – *Pipeline CI/CD*.



Fonte: Dimikj (2020)

A Figura 4 apresenta a capacidade de um *pipeline* para realiza nos quesitos da Integração e Entrega Contínua, para assim desenvolver e entregar a aplicação para o usuário. Os passos que o *pipeline* deve conter são (REDHAT, 2020):

1. **Construção do código**: Esta etapa consiste na construção realizada após um *commit* no repositório de código central;
2. **Teste**: Esta etapa consiste na execução dos testes automatizados elaborados pela equipe responsável;
3. **Lançamento**: Esta etapa consiste no envio do artefato a um repositório para ser imple-

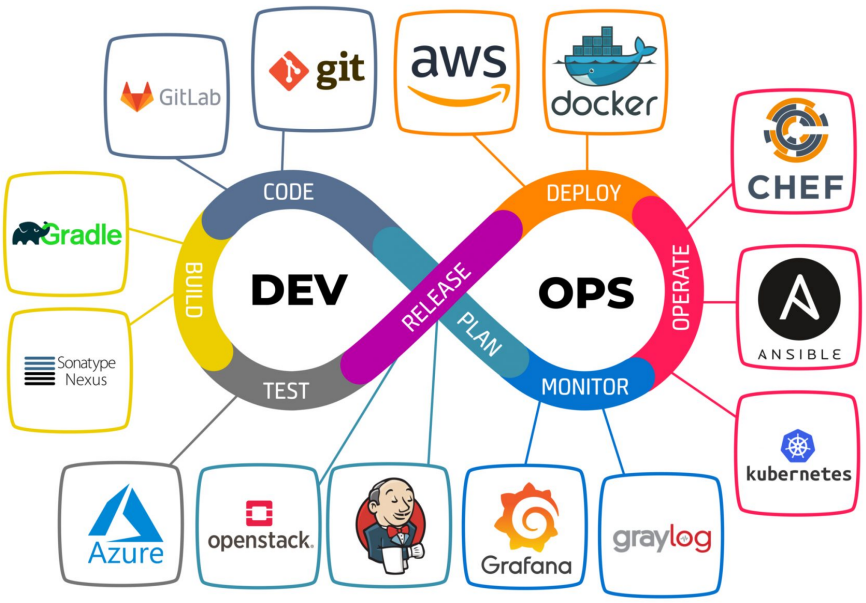
mentado no ambiente produtivo ou ambiente de qualidade do cliente;

4. **Implementação**: Esta etapa consiste na implementação de fato do artefato que está no repositório, nos ambiente de produção ou homologação do usuário final;
5. **Validação**: Esta etapa consiste na verificação do ambiente aonde está implementado o artefato.

2.5 Ferramentas

Para o *DevOps*, escolher a melhor ferramenta para cada processo do ciclo de vida é muito importante, pois cada uma delas assumem o papel de suporte para manter as atividades em pleno funcionamento. Além disso, manter os processos desenhados, alinhados e simplificados ao máximo, principalmente com ferramentas ligadas a automação ajudam a manter ainda mais tal funcionamento.(HÜTTERMANN, 2012). A Figura 5 mostra algumas ferramentas existentes no mercado em cada processo do ciclo de vida do *DevOps*.

Figura 5 – Ferramentas *DevOps*.



Fonte: Gobbi (2019)

Segundo Hüttermann (2012) quanto mais ferramentas estiverem no kit da organização, maior é a possibilidade de escolha adequada para as atividades do *DevOps*. É de extrema importância que os processos sejam automatizados do começo ao fim, ou seja, desde o desenvolvimento até a entrega para o usuário final. Portanto, é necessário que a empresa adote (HÜTTERMANN, 2012):

- *Scripts* de automatização de construção da aplicação;
- *Scripts* de testes automatizados;
- *Scripts* de implantação do aplicativo;
- *Scripts* de aceitação da aplicação no ambiente do cliente;
- *Scripts* que possam configurar a aplicação para diversos ambientes;
- *Scripts* que possam controlar e monitorar a aplicação no ambiente do cliente.

No que se refere as ferramentas que evolvem diretamente a Integração e Entrega Contínua, o corrente trabalho apresenta duas ferramentas de destaque, que são: *Jenkins* e *Gitlab*.

2.5.1 Jenkins

O *Jenkins* é uma ferramenta de automação em código aberto, capaz de abranger todas as atividades relacionadas a construção, testes, entrega ou implementação de *softwares* (JENKINS, 2021a). Ele é provavelmente umas das ferramentas mais populares quando se pensa em Integração Contínua e Entrega Contínua (RITI, 2018).

- **Características e Funcionalidades da ferramenta**

A ferramenta *Jenkins* é extensível e simples devido a sua interface de fácil interação, seja para desenvolvedores ou usuários do setor de operações, pois se trata de uma *software* de automação completo. Ela também é capaz de construir projetos de *Ant* e *Maven*, dentre outras tipos de projetos. Além disso, é compatível com diversos repositórios de código fonte como, *Git*, *Subversion*, *CVS* e *Mercurial*. O conceito que a ferramenta também aborda de *plugins*, que são recursos adicionados a mais, que podem ser moldados de acordo com o método de desenvolvimento e de entrega adotado pela organização o torna mais atrativo para realizar as atividades, a ferramenta disponibiliza muitos *plugins* e de diversos tipos, que são (SONI, 2015):

- Ferramentas de Gerenciamento de código;
- Gatilhos para construção;
- Ferramentas de Construção;
- Ferramentas de Notificação;
- Ferramentas Pós Construção;
- Relatórios de Construção;
- Ferramentas de customização da interface gráfica.

Jenkins também é uma ferramenta de fácil instalação, funcionando em diversos tipos de sistemas operacionais como: *Windows, MacOS, Linux Ubuntu, RadHat, OpenSUSE* e *Docker*, além disso possui um pacote genérico em Java. Sua Comunidade é bem extensa e possui várias formas de colaboração, seja na parte de desenvolvimento de novas funcionalidades ou apenas doação de dinheiro para o projeto (JENKINS, 2021b).

2.5.2 GitLab

O *GitLab* é uma plataforma *DevOps*, que unifica diversas funcionalidades em um único aplicativo e consegue abranger todo o ciclo de desenvolvimento de *software*, o que o diferencia bastante dos seus concorrentes. Com seu uso, é possível criar vários fluxos de trabalho de maneira simples, fazendo com que a organização deixe de usar diversas ferramentas diferentes para cada fluxo de trabalho (GITLAB, 2021c).

- ***Características e Funcionalidades da ferramenta***

O *GitLab* possui uma grande comunidade contendo mais de três mil colaboradores e mais de 2 milhões de usuários na plataforma, portanto, o suporte é bem robusto, contendo diversos conteúdos de configurações (GITLAB, 2021c). Além disso, fornece a capacidade de otimizar o tempo de desenvolvimento e entrega do *software* tornando esse fluxo mais rápido, eficiente e seguro. Uma das características que mais se destacam na plataforma é sua facilidade de uso da sua interface gráfica que é bastante amigável e simples, sendo capaz de gerenciar facilmente as permissões e demais outras configurações. Além do mais, possui recursos já inclusos a sua interface para facilitar a documentação da aplicação que nele está sendo desenvolvida (HETHEY, 2013).

O *GitLab* CI/CD é uma sub ferramenta do *GitLab* que faz de forma automatizada a construção, teste e implementação das aplicações no ambiente de homologação ou produção do cliente (AREFEEN; SCHILLER, 2019). Para que o *Gitlab* consiga executar todos os processos desde o *build* até a entrega, é utilizado o *GitLab Runner*, que é um aplicativo destinado totalmente à execução de *pipelines* dentro dos projetos (GITLAB, 2021a).

3 TRABALHOS CORRELATOS

Este Capítulo apresenta os trabalhos relacionados a área de *DevOps*, Integração e Entrega Contínua e suas ferramentas que estão, concomitantemente, correlacionados com o presente trabalho. Serão apresentados os principais pontos de convergência e divergência entre os estudos correlatos.

Mohammad (2016) tem o objetivo em sua obra de apresentar os benefícios de se adotar a Integração Contínua como uma prática importante para auxiliar os desenvolvedores na implantação de mudanças no código do *software* de forma cada vez mais rápidas. Toda interação que é feita no repositório de código-fonte, passa por testes automatizados, que visam garantir a integridade do código desenvolvido. Dessa forma, a Integração Contínua contribui também para o processo de confiabilidade do aplicativo. Antes da Integração Contínua era difícil de desenvolver, pois cada membro do time trabalhava separadamente e as alterações poderiam ser realizadas somente quando a *branch master* estivesse finalizada. Como consequência era muito custoso o processo de mesclar o código contribuindo para o acúmulo de *bugs* e erros no sistema. O autor também apresenta brevemente as características com o objetivo de expor as particularidades das seguinte ferramentas de Integração Contínua: *GitLab CI, Jenkins, TravisCI, TeamCity, Bamboo, CircleCI* e *Buddy*. Nesta obra o autor conclui que a Integração Contínua contribui para o aumento da produtividade do desenvolvimento e que a prática possui excelentes ferramentas disponíveis para uso. O presente trabalho apresenta através de um implementação as funcionalidades tanto de Integração Contínua quanto de Entrega Contínua das ferramentas *Jenkins* e *GitLab*, visando mostrar como ambas trabalham nesses aspectos.

Révész e Pataki (2021), apresentam a importância de ferramentas de Integração Contínua na Engenharia de *Software*, reforçando que elas são fundamentais para o desenvolvimento, pois ajudam a mesclar o código mais facilmente, otimizando a resolução de *bugs* e tornando-as mais rápidas. Nesse trabalho também são apresentadas as soluções *Azure DevOps, GitLab CI* e *Jenkins*, porém, seu estudo prioriza o *Jenkins*, destacando-a como uma das ferramentas mais utilizadas no quesito Integração Contínua e objetivando somente melhorar as *pipelines* escritas mais abrangentes do *Jenkins* tornando-as mais legíveis. A obra é concluída com a solução de *pipeline* desenvolvida pelo autor No trabalho proposto, não será implementado somente as *pipelines* codificadas do *Jenkins*, pois serão explorados os *pipelines* utilizando a interface da ferramenta, além disso, serão implantadas as *pipelines* do *GitLab*, com a finalidade de compará-las.

Glein *et al.* (2019) em sua obra, trata do objetivo melhorar o desenvolvimento e testes

de *software* e *firmware* para *chips Field-Programmable Gate Array* (FGPA) através da Integração Contínua. Nesse trabalho é enfatizado a importância da Integração Contínua para a descoberta e correção de erros mais rapidamente com a ajuda de testes automatizados. Além disso, no estudo também é informado que com a Integração Contínua as compilações de *firmwares* se mostraram mais facilitadas no sentido na reprodução dos resultados serem mais claras, conseguindo rastrear em que parte do código foi causadora do erro apresentado. Sendo assim, Glein *et al.* (2019) concluem que as funcionalidades do *GitLab* CI/CD e que foram importantes para agilizar os processos de desenvolvimento dos *softwares* e *firmwares*. Entretanto, os autores não apresentaram uma implementação mais detalhada de como esse processo é configurado, exibindo somente os status da *pipeline* criada. O presente trabalho implementa de forma detalhada e configuração as *pipelines* do *GitLab* e do *Jenkins*.

4 METODOLOGIA

Este capítulo apresenta o procedimento metodológico utilizado como base para a execução das atividades do trabalho. A técnicas de pesquisa realizadas para elaboração da obra são do tipo bibliográfica, por fazer uso de fontes secundárias de dados como artigos, livros e monografias. Este também é uma pesquisa do tipo documental, por fazer uso de fontes primárias como documentações privadas de projetos (MARCONI; LAKATOS, 2003).

O presente estudo se propõe a avaliar e comparar, mediante a critérios que serão levantados ao decorrer do trabalho, as funcionalidades das ferramentas *Jenkins* e *Gitlab* no tocante a Integração e Entrega Contínua. Este estudo também faz uso de ferramentas que auxiliam no procedimento de implementação dos *pipelines*, a saber: *GitLab Runner*[1], *Nexus Repository Manager*[2] e *GitHub*[3].

A Coleta de dados possui característica documental, e além disso, os dados serão extraídos a medida em que as ferramentas forem executadas. Os resultados obtidos serão analisados de maneira qualitativa, com o intuito de verificar o comportamento de cada ferramenta em determinado critério levantado.

[1] Ferramenta que executa *jobs* no *GitLab* CI/CD. Disponível em: https://docs.gitlab.com/runner/. Acesso em: 1 de Novembro de 2021

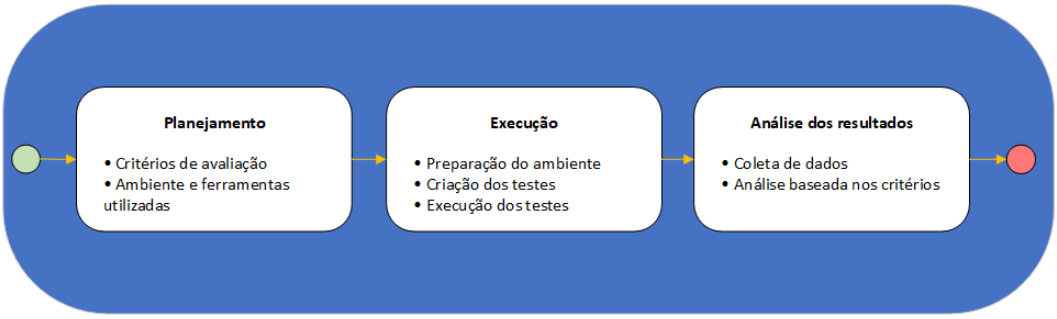
[2] Repositório de Artefatos Privado. Disponível em: https://www.sonatype.com/products/repository-oss?topnav=true. Acesso em: 1 de Novembro de 2021

[3] Repositório de artefatos na *web*. Disponível em: https://github.com/. Acesso em: 1 de Novembro de 2021

5 CONDUÇÃO DO ESTUDO

Este capítulo apresenta o formato de como esse estudo é conduzido. A Figura 6 representa graficamente o fluxo de cada etapa para a execução do presente trabalho, este fluxo é composto por três etapas, a saber: planejamento, execução e análise dos resultados obtidos.

Figura 6 – Fluxo de etapas realizadas para a execução do trabalho.

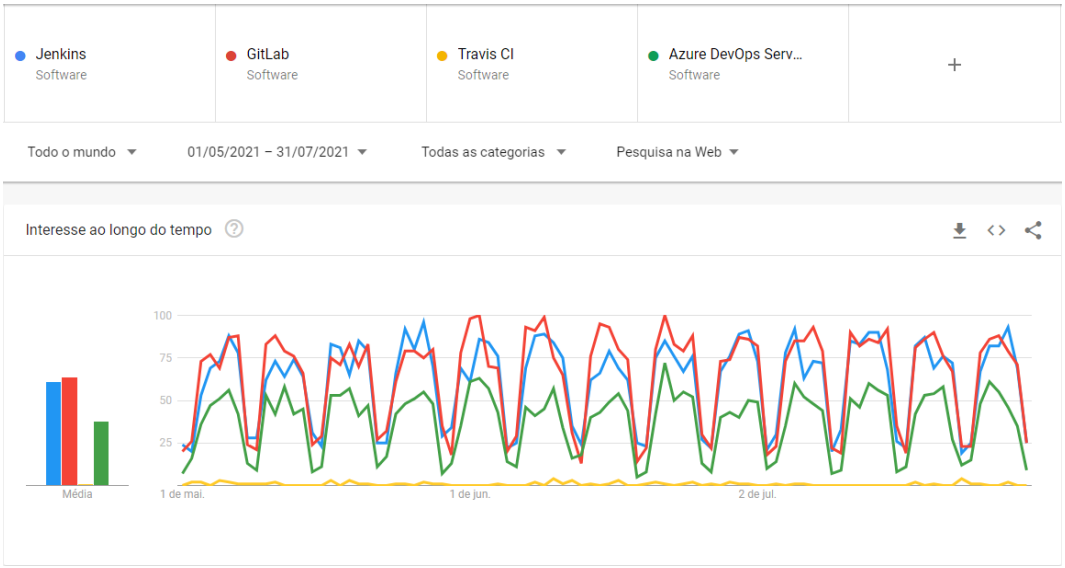


Fonte: Elaborada pelo Autor.

5.1 Planejamento

Para a construção deste trabalho foram selecionadas as ferramentas *Jenkins* e *GitLab* a fim de comparar aspectos de funcionamento, incluindo o processo de configuração do *pipeline* de Integração e Entrega Contínua. A escolha das ferramentas foram baseadas em uma busca realizada na plataforma *Google Trends* com o objetivo de saber quais, dentre as ferramentas de Integração e Entrega Contínua, são as mais utilizadas do mercado a nível global.

Figura 7 – Comparação entre as ferramentas de CI/CD.



Fonte: Trends (2021)

A Figura 7 apresenta a comparação realizada na plataforma *Google Trends*. As ferramentas de CI e CD que foram os alvos dessa busca são: *Jenkins, GitLab, TravisCI* e *Azure DevOps Server*. A comparação é realizada a nível global no período de três meses, entre o dia 1 de Maio de 2021 a 31 de Julho de 2021.

Na definição dos critérios adotados para realizar a comparação, foram utilizadas documentações oficiais, livros e artigos relacionados a área de *DevOps*. Os artigos e livros utilizados para foram pesquisados na plataforma Google Acadêmico. A busca foi efetuada com palavras-chave relacionadas a área de *DevOps*, com foco em ferramentas de CI/CD, a saber: *Pipelines CI/CD, Integrated DevOps Tools, DevOps Tools e Continuous Integration Tools*. Nesse processo de busca não foi feita filtragem por data.

Softwares livres são importantes para a comunidade de desenvolvedores e consumidores, devido a sua alta capacidade de customização, por conta da fácil acessibilidade ao código-fonte da aplicação e também por conta do custo zero para a adquiri-los (GNU, 2021). Com base nisso, o primeiro critério foi levantado para a composição do presente estudo e teve seu nome definido como: Ferramenta Gratuita.

Arachchi e Perera (2018), em sua obra, apresentam a importância que os *pipelines* de Integração e Entrega Contínua possuem para o desenvolvimento atualmente, deixando os processos mais ágeis e otimizados, desde o desenvolvimento até a entrega do *software*. Dessa forma, o segundo critério foi levantado para a composição do presente estudo e teve seu nome definido como: Suporte completo aos *pipelines*.

As integrações entre as ferramentas são importantes para manter o ambiente sempre otimizado e flexível, dessa forma, não há sobrecarga e nem centralização das cargas de trabalho em uma única ferramenta do ambiente. Além disso, devido a flexibilidade, a ferramenta possui uma maior probabilidade de de se integrar a qualquer outra ferramenta pertencente ao leque de possibilidades em cada processo do ciclo *DevOps* (BOBROV *et al.*, 2020). Com base nisso, o terceiro critério foi levantado para composição do presente trabalho e teve seu nome definido como: Ferramentas Integradas.

O *Software as a Service* (Sistema como um Serviço) é um tipo que arquitetura da computação em nuvem que contribui para o aumento da acessibilidade e para a ausência de preocupação, por parte do usuário final, com ralação a processos relacionados infraestrutura e com versão da aplicação que está sendo utilizada (SALESFORCE, 2021). Com base nisso, o quarto critério foi levantado para a composição o presente estudo e teve seu nome definido como:

Suporte a *Software as a Service* (SaaS).

Ter conhecimento de diferentes possibilidades de configuração dos *pipelines* é interessante, pois a configuração se torna mais flexível para os usuários que farão uso da ferramenta. Dessa forma, o quinto critério foi levantado, com caráter adicional, para composição do presente trabalho e teve seu nome definido como: Formatos de configuração do *pipeline*.

Para fins comparativos entre as duas ferramentas, os critérios definidos são:

- **C1 - Ferramenta gratuita**: *Softwares* gratuitos são mais acessíveis para os usuários e estão constantemente sendo atualizados pela comunidade (GNU, 2021). Esse critério verifica se *Jenkins* e *GitLab* são gratuitos;
- **C2 - Suporte completo aos *pipelines***: Esta métrica verifica se a ferramenta oferece um *pipeline* completo, iniciando com a construção do projeto, seguindo com os testes, e finalizando com a implementação no ambiente do usuário final. Esses passos são importantes, pois a aumentam da eficiência no desenvolvimento do projeto. (ARACHCHI; PERERA, 2018). Esse critério medirá se nos *pipelines* do *Jenkins* e o *Gitlab* possuem campos separados para *buid, test* e *deploy*;
- **C3 - Ferramentas Integradas**: Quanto maior a capacidade em que a ferramenta tem de se integrar a outras ferramentas, melhor para a organização, pois otimiza os processos envolvendo o *DevOps*, não sobrecarregando uma só ferramenta. E também, adequando-se a qualquer ambiente devido alta probabilidade de integração com outras ferramentas pertencentes ao leque de opções (BOBROV *et al.*, 2020). Esse critério se *Jenkins* e *GitLab* são capazes de se integrar com outras ferramentas;
- **C4 - Suporte a *Software as a Service* (SaaS)**: Ferramentas que possui a arquitetura *Software as a Service* (SaaS) pode trazer uma série de benefícios para quem à utiliza, dentre elas: Acessibilidade e rápidas atualizações de sistemas (SALESFORCE, 2021). Esse critério verifica se *Jenkins* e *GitLab* oferecem planos em *Software as a Service* (SaaS);
- **C5 - Formatos de configuração do *pipeline***: Baseando-se na importância dos *pipelines*, a ferramenta que traz diferentes formatos para a configuração dos seus *pipelines* pode atrair uma maior quantidade de usuários para seu uso. Esse critério implica em quantos formatos de configuração *Jenkins* e *GitLab* possui para seus *pipelines*.

O desenvolvimento deste estudo utiliza quatro máquinas virtuais com as seguintes ferramentas instaladas: *Jenkins, GitLab, GitLab Runner* e *Nexus Repository Manager*. *Jenkins* e *GitLab* serão as ferramentas alvo da comparação. O *GitLab Runner* é a ferramenta que torna

possível a execução das *pipelines* no *GitLab*. O *Nexus Repository Manager* será utilizado para simular o ambiente do cliente em produção onde será implementado o artefato gerado pelos *pipelines*. O *Nexus* foi selecionado, pois é uma ferramenta gratuita e de rápida configuração. O artefato gerado pelos *pipelines* é baseado em um projeto *Java* mais o gerenciador de dependências *Maven*, que simula uma biblioteca que vai ser implementada no *Nexus*, que representa o ambiente no cliente nesse estudo. Para testes com o *Jenkins*, com o objetivo de separar as ferramentas, não será utilizado o *GitLab* como versionador de código-fonte, ao invés dele é utilizado o *GitHub*[1].

5.2 Execução

A primeira etapa da execução, trata-se de uma implementação de uma simulação de como o *Jenkins* é configurado para executar seus *pipelines*. Nesse momento, são feitas duas configurações que possuem a mesma finalidade, mas em formatos diferentes fornecidos pela ferramenta, que são: I) Configuração do *pipeline* declarativo e II) Configuração do *pipeline* utilizando sua própria interface do *Jenkins*. A segunda etapa da implementação das ferramentas será com a configuração do *pipeline* com a mesma finalidade no *GitLab*. Para a execução da simulação no presente estudo, as aplicações escolhidas farão uso dos seguintes recursos de *hardware* e *software* disponíveis:

Jenkins versão 2.303.2, *GitLab Runner* versão 14.4.0 e *Nexus Repository Manager* versão 3.35.0, cada um dos *softwares* foi executado em máquinas virtuais separadas, porém com as mesmas configurações listadas a seguir:

- Sistema Operacional: *Ubuntu Server* versão 20.04.3 *Long-Term Support* (LTS);
- Processadores: 2 processadores virtuais *Intel Xeon*;
- *Random Access Memory* (RAM): 2GB.

A maquina virtual utilizada para executar o *GitLab Omnibus Community Edition* (CE) versão 14.4.0, possui uma configuração superior em relação as demais, devido a aplicação utilizar mais recursos da máquina virtual. As configurações são:

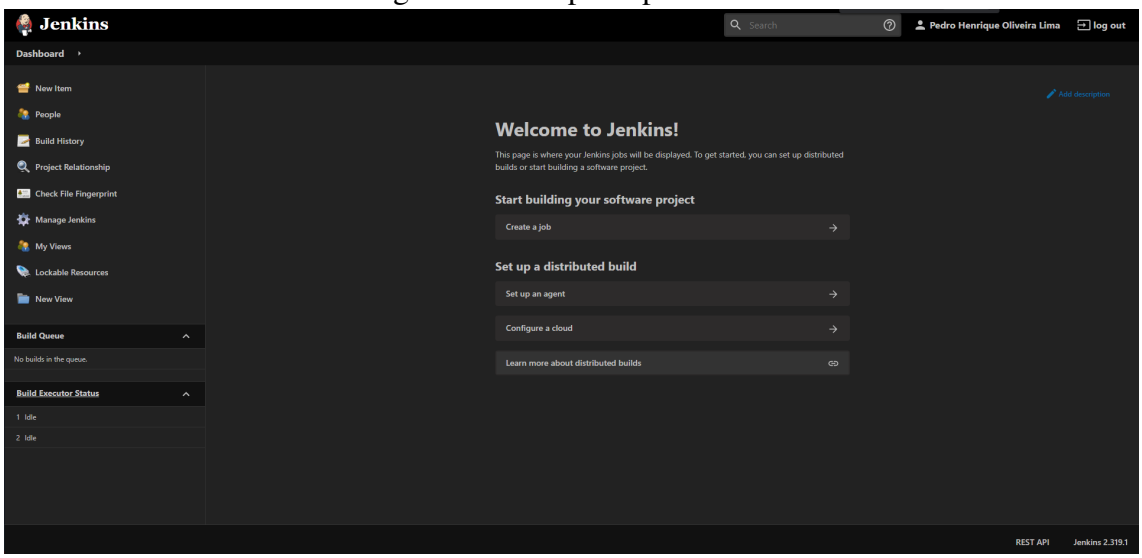
- Sistema Operacional: *Ubuntu Server* versão 20.04.3 *Long-Term Support* (LTS);
- Processadores: 8 processadores virtuais *Intel Xeon*;
- *Random Access Memory* (RAM): 4GB.

[1] https://github.com/pedrooliveira0997/projetoteste

5.2.1 Jenkins

O *Jenkins* é uma ferramenta gratuita e de código aberto, portanto, cumpri o critério relacionado a *softwares* gratuitos apresentados na seção 5.1. Além disso, não possui suporte a *Software as a Service* (SaaS), sendo somente possível o *Self-Hosted* (Auto-Hospedado), ou seja, o custo da ferramenta estão relacionados somente com a infraestrutura que o aloca, nesse caso, não corresponde ao critério relacionado ao *Software as a Service* (SaaS) apresentado na seção 5.1.

Figura 8 – Tela principal *Jenkins*.



Fonte: Elaborada pelo Autor.

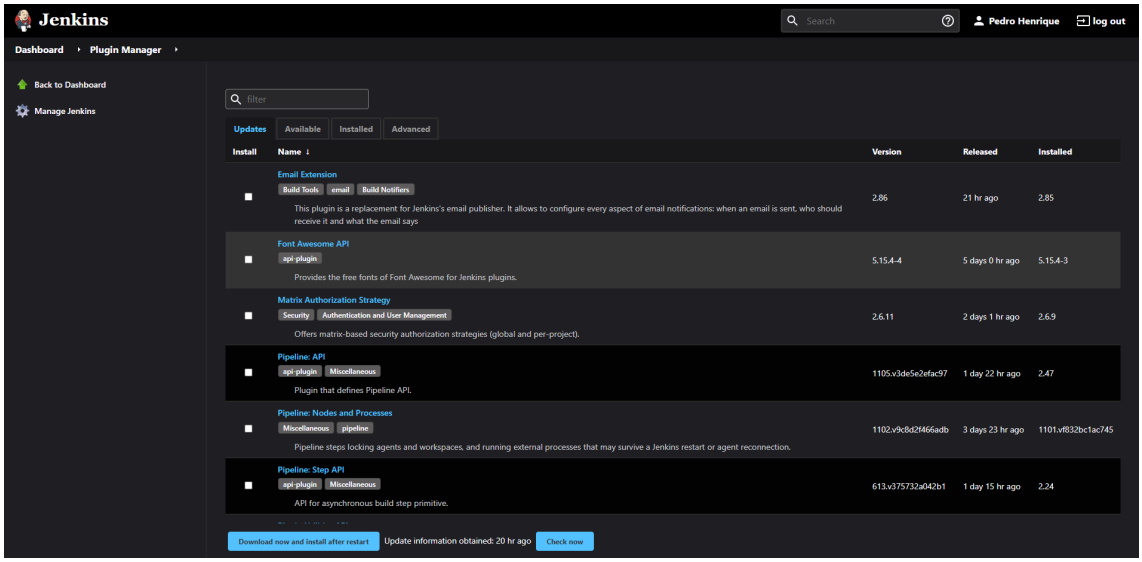
A Figura 8 apresenta a tela inicial do *software Jenkins*, em que é possível observar uma série de configurações que podem ser feitas, como: Criação de novos *jobs*, tela de usuários, histórico de *builds*, *builds* na fila de execução, *builds* em execução e configurações gerais.

- **Configuração Inicial**

Antes da construção dos *pipelines* é necessário realizar uma pré configuração referente a aplicação que vai ser construída, testada e implantada. Para realizar a configuração, o *Jenkins* faz a utilização de *plugins* que são serviços que permitem agregar várias funcionalidades. Existem diversos tipos de *plugins* disponíveis para o *Jenkins*, cada um deles responsáveis por atividades específicas. Os *plugins* são utilizados também para integrar o *Jenkins* com outras ferramentas que fazem parte do ciclo, com essa possibilidade de integração, o *Jenkins* cumpre

o critério relacionado a integração com outras ferramentas na seção 5.1, por exemplo: *Plugins* de notificação ao final da execução do *pipeline*, *plugins* de envio de artefatos via *Secure Copy* (SCP) e *Plugins* referente a provedores de computação na nuvem como *Amazon Web Services* (AWS) e *Microsoft Azure* (JENKINS, 2021c).

Figura 9 – Tela gerenciamento de *plugins* do *Jenkins*.



Fonte: Elaborada pelo Autor.

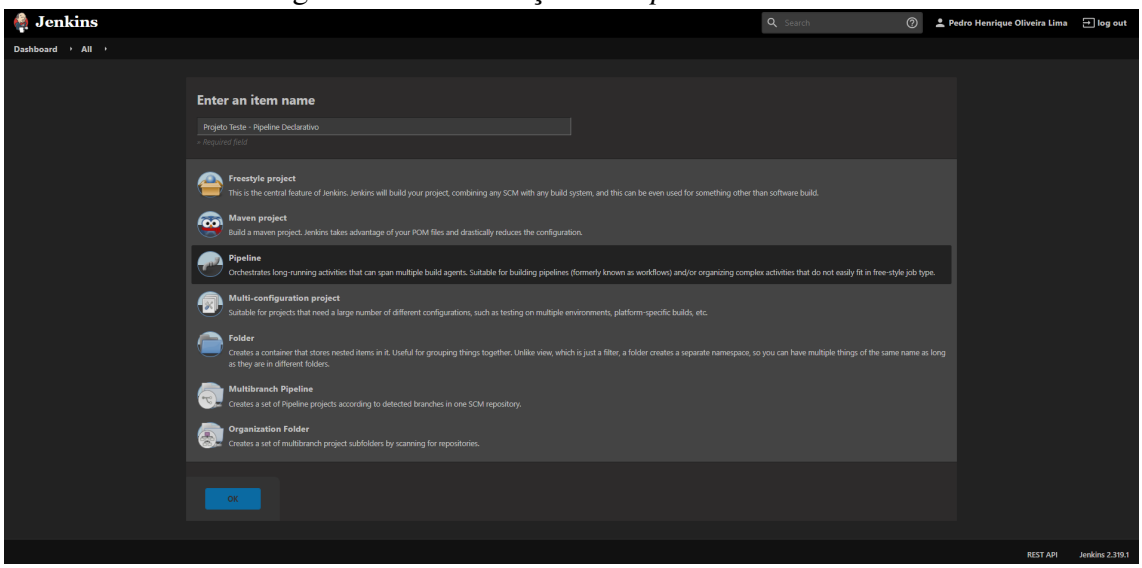
A Figura 9 apresenta a tela de configuração de *Plugins* do *Jenkins* localizada em ***Manage Jenkins*** → ***Manage Plugins***. Como a aplicação teste faz uso do *Maven* e é implantada em um repositório no *Nexus*, são instalados os seguintes *plugins* referente a essas funcionalidades: *Maven Integration* e *Nexus Artifact Uploader*. Logo após a configuração desses *plugins* a ferramenta está apta para a configuração dos *pipelines*.

- ***Pipeline* Declarativo**

O *Jenkins* fornece a configuração de seus *pipelines* através de um modo declarativo, que é construído através de linhas de código separadas por estágios, cada estágio representando uma fase que será executada no *pipeline*.

A Figura 10 apresenta a tela que dispõe as *features* para escolher a criação do *Pipeline* Declarativo no *Jenkins* localizada em ***New Item*** na tela principal. Em seguida, tem-se um campo para nomear, e por fim, deve-se selecionar ***Pipeline*** e a ferramenta já permite que o usuário inicie a escrita do *pipeline*.

Figura 10 – Tela criação do *Pipeline* Declarativo.



Fonte: Elaborada pelo Autor.

A Figura 11 apresenta o *pipeline* declarativo no *Jenkins*. Ele é iniciado através do bloco chamado *Pipeline*, declarado na Linha 1 e encerrado na Linha 49. A Linha 2 especifica em qual agente deve ser executado o *pipeline*, agentes são os locais em que o *jobs* são executados. Nessa configuração, não foi definida a preferência de execução, portanto, a tarefa será executada no próprio servidor da ferramenta. Da Linha 3 a Linha 6, são declaradas as ferramentas utilizadas para a execução, nesse caso *Maven* e *Java*.

Ainda na Figura 11, são apresentados os estágios do *pipeline* que estão declarados da Linha 7 a Linha 48. Esses estágios percorrem desde a consulta ao repositório, construção, executando os testes configurados e implementando no ambiente do cliente, que está sendo representado pelo *Nexus*.

Percorrendo a Figura 11, da Linha 8 a Linha 14, o estágio de consulta do repositório do projeto teste é iniciado, projeto este localizado no *GitHub*. Esse estágio consulta o repositório e tem como objetivo a verificar se houve alterações ocorridas no código-fonte da aplicação, possibilitando que novas construções possam ser disparadas. A ferramenta *Jenkins*, além de oferecer suporte ao *Git*, é capaz capturar alterações de outros de controladores de versão, por exemplo, *Subversion* (SVN), *Mercurial* e *CVS*.

Declarado da Linha 15 a Linha 21 na Figura 11, o estágio de Construção é responsável pela compilação do projeto, no caso do projeto teste, o *Maven* é a ferramenta utilizada para realizar esse procedimento, devido o projeto ser baseado nele. O *Jenkins* oferece suporte a ferramentas além do *Maven*, como *Gradle Build Tool* e *Ant Build*.

Figura 11 – *Pipeline* Declarativo do *Jenkins*.

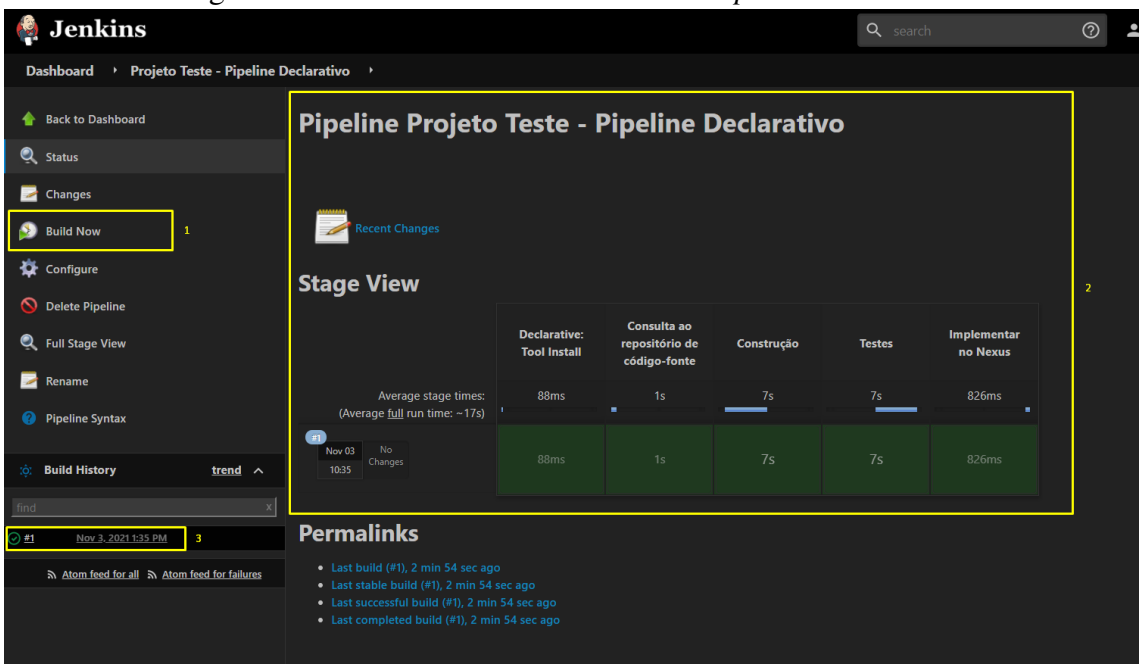
```
pipeline {
    agent any
    tools {
        maven "Maven 3.8.3"
        jdk 'Java 8'
    }
    stages {
        stage("Consulta ao repositório de código-fonte") {
            steps {
                script {
                    git 'https://github.com/pedrooliveira0997/projetoteste.git';
                }
            }
        }
        stage("Construção") {
            steps {
                script {
                    sh "mvn clean package -DskipTests=true"
                }
            }
        }
        stage("Testes") {
            steps {
                script {
                    sh "mvn test"
                }
            }
        }
        stage("Implementar no Nexus") {
            steps {
                nexusArtifactUploader artifacts: [
                    [
                        artifactId: 'projetoteste',
                        classifier: '',
                        file: 'target/projetoteste-1.0.0.jar',
                        type: 'jar'
                    ]
                ],
                credentialsId: 'nexus-producao',
                groupId: 'com.projetoteste',
                nexusUrl: '172.16.21.122:8081',
                nexusVersion: 'nexus3',
                protocol: 'http',
                repository: 'maven-repo-libs',
                version: '1.0.0'
            }
        }
    }
}
```

Fonte: Elaborada pelo Autor.

Da Linha 22 a Linha 28 na Figura 11 é declarado o estágio de Testes. Nesse caso, o artefato é testado antes de ir para o repositório *Nexus* do cliente. Nesse Projeto, estão sendo utilizado os testes padrões realizados pelo *Maven*, que são apenas testes de execução simples.

Por fim, ainda na Figura 11 considerando as linhas 29 a 46, é executado o estágio de implementação do artefato no repositório do cliente. Nesse momento, a implementação do artefato só é realizada se aprovada nos testes executados pelo próprio *Maven*.

Figura 12 – Tela *Status* e Visão Geral do *Pipeline* Declarativo.



Fonte: Elaborada pelo Autor.

A Figura 12 apresenta a interface que procede a configuração do *pipeline*. No destaque 1 da imagem, o botão ***Build Now*** é responsável pela execução completa do *pipeline*. Quando a execução é finalizada, será apresentado o destaque de número 2 da imagem, que indica o *status* e a visão geral do *job*, nele está contido o tempo gasto em cada estágio e se foi executado com sucesso. Para a indicação sucesso, a cor verde é utilizada, caso contrário, será utilizado a cor vermelho. Para informações mais detalhadas do *job*, como *logs* da execução, o destaque 3 da imagem apresenta essa função. Portanto a ferramenta conta com um suporte total aos *pipelines* contendo construção, teste e implementação, além disso, apresenta a uma visão geral e os *status* contribuindo para o 1º critério da seção 5.1.

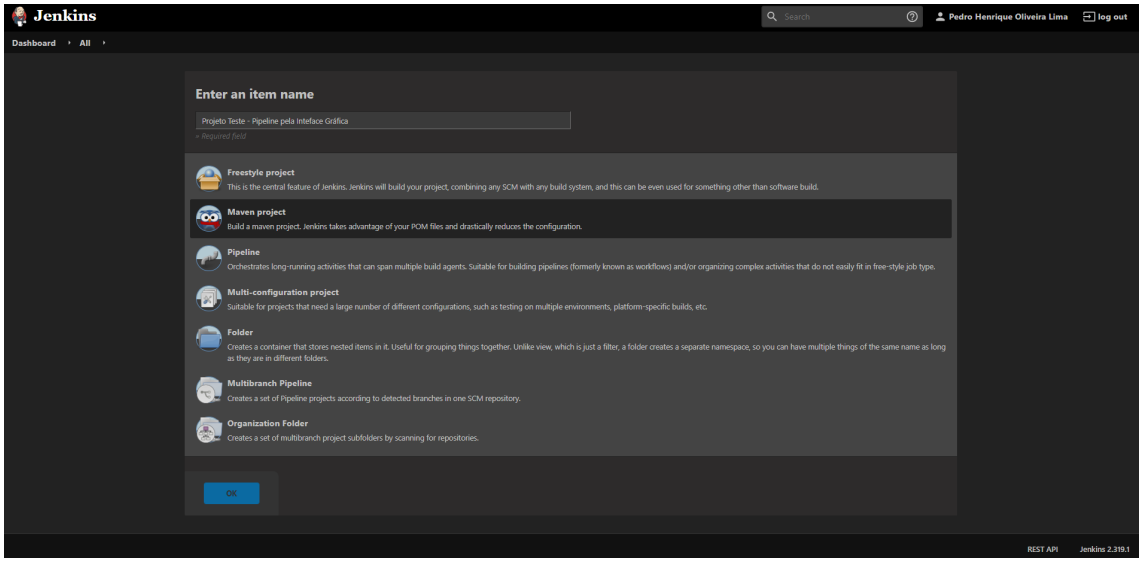
- **P*Pipeline* Utilizando a Interface Gráfica**

Além de oferecer a configuração de um *pipeline* de maneira declarativa ou em linha de código, o *Jenkins* também fornece a configuração de um *pipeline* utilizado sua própria interface gráfica. Essa forma de configurar os *pipelines* segue o mesmo principio da forma declarativa, ou seja, dividida em estágios que são etapas que são executadas pelas ferramenta.

Assim como no *pipeline* declarativo, as etapas que serão executadas até o o ambiente simulado do cliente serão: I) leitura do versionador de código para procurar por alterações no mesmo, II) construção e testes da aplicação através do *Maven*, e por fim, III) implementação no

ambiente simulado do cliente. A principal diferença entre os estágios desse *pipeline* e o *pipeline* por linha de código é a adição de mais um passo presente na configuração, que são ações de elementos pré *build* da aplicação.

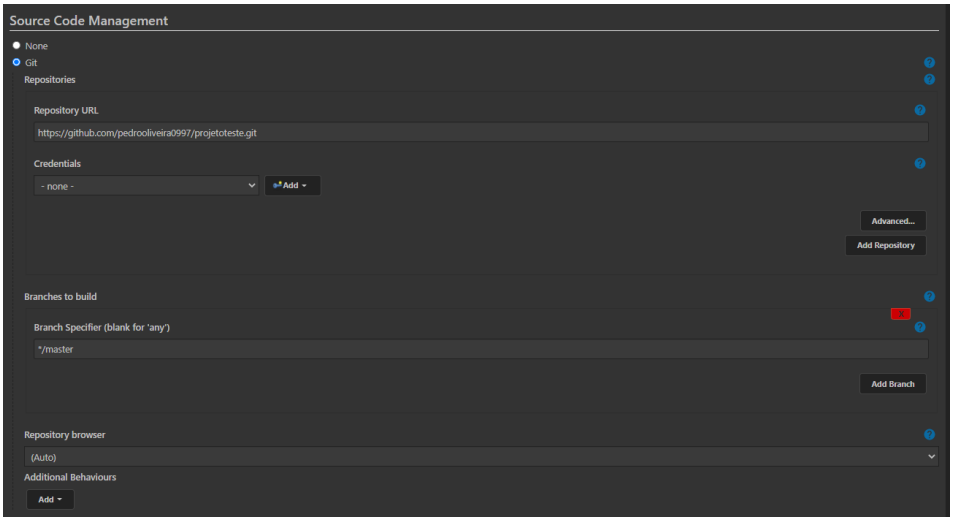
Figura 13 – Tela criação no *pipeline* pela interface gráfica.



Fonte: Elaborada pelo Autor.

A Figura 13 apresenta a tela destinada a realização da criação do *pipeline* pela interface gráfica no *Jenkins*, que está localizada no menu ***New Item*** na tela principal. Logo, é solicitada a nomeação do projeto. Com a instalação do *plugin* do *Maven*, é disponibilizada uma *feature* que oferece um campo exclusivo de configuração para esse tipo arquitetura de projeto através da interface gráfica. Nesse caso, é selecionado a opção **Maven Project**, e em seguida, a ferramenta já permite que o usuário inicie o desenvolvimento do *pipeline* pela interface gráfica.

Figura 14 – Tela de consulta ao versionador no *pipeline* pela interface gráfica.



Fonte: Elaborada pelo Autor.

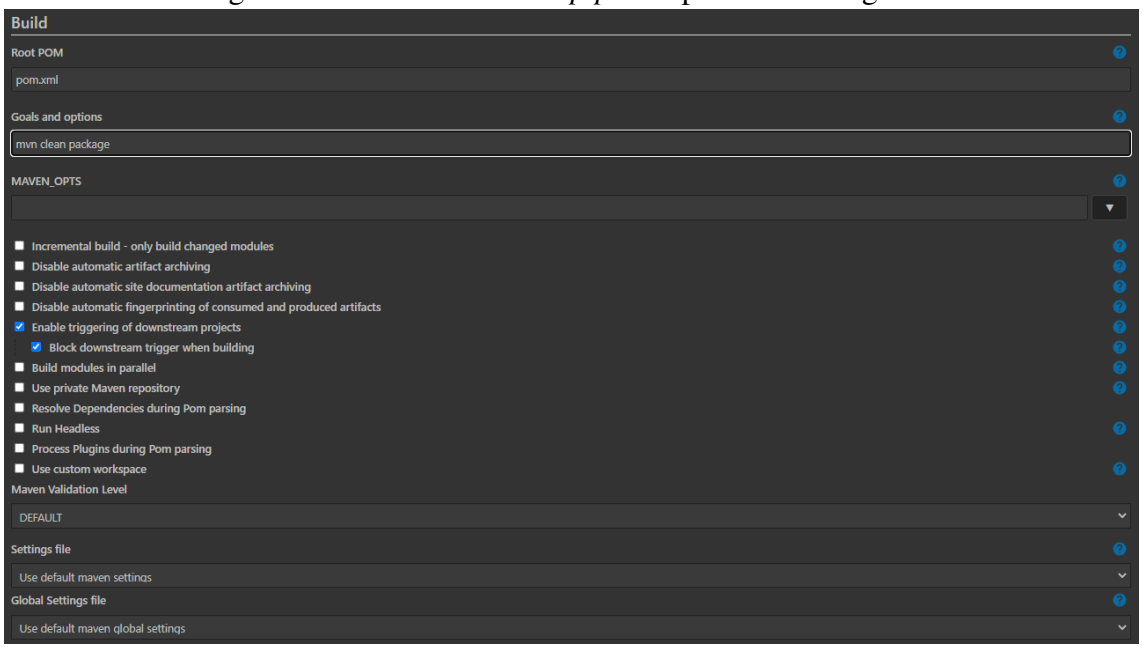
A Figura 14 mostra o primeiro estágio do *pipeline* pela interface gráfica, que é a consulta ao versionador do código-fonte. Nesse caso, o *GitHub*. Esse passo tem o mesmo funcionamento do *pipeline* declarativo, ou seja, verificar as alterações realizadas no código-fonte da aplicação para fazer a construção da mesma.

Figura 15 – Tela de pré-*build* do *pipeline* pela interface gráfica.

Fonte: Elaborada pelo Autor.

A Figura 15 apresenta a alguns estágios pré-*build* do *pipeline*, neles estão contidos, construções periódicas, consultas automáticas ao versionador do código-fonte em busca de alterações ou execução do algum comando seja via *Shell* ou *Batch*, todas essas configurações são realizadas antes da *build* do projeto.

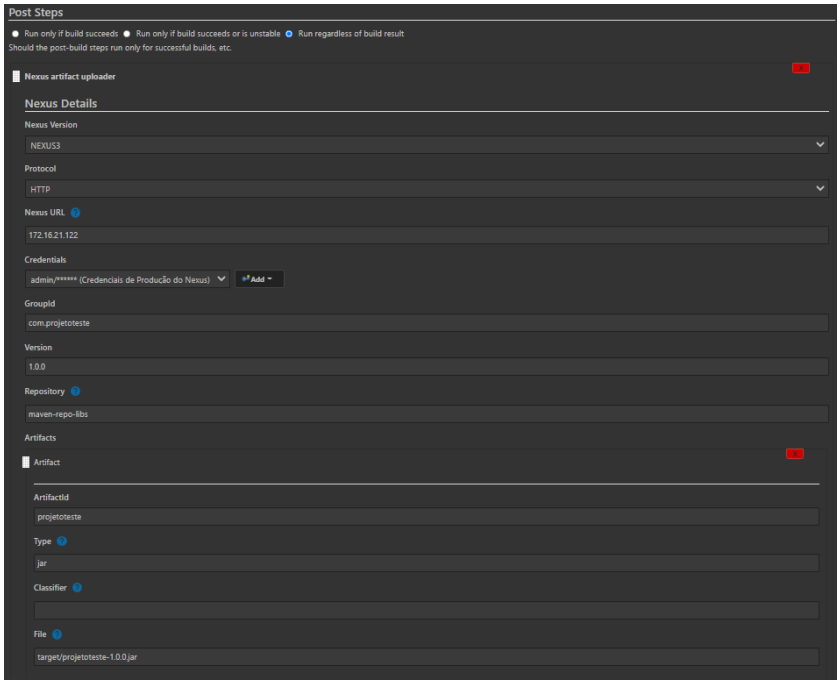
Figura 16 – Tela de *build* do *pipeline* pela interface gráfica.



Fonte: Elaborada pelo Autor.

A Figura 16 apresenta o estágio de *build* do *pipeline*, nele estão contidos configurações referentes ao *Maven*, como: Variáveis, *Goals* ou parâmetros, execução da construção em segundo plano, dentre outras funcionalidades. Nesse estágio, já está incluído o estágio de teste do código.

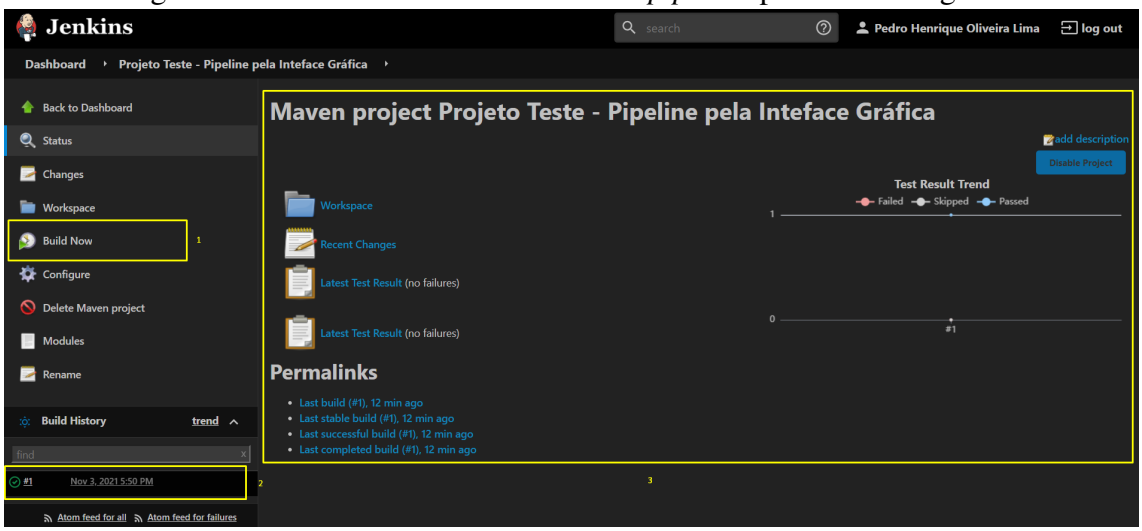
Figura 17 – Tela para implementar no *Nexus* do *pipeline* pela interface gráfica.



Fonte: Elaborada pelo Autor.

A Figura 17 apresenta estagio referente a implementação do artefato no ambiente do cliente. Assim como no *pipeline* declarativo, este passo só é realizado se projeto for aprovado no estagio de testes executado anteriormente.

Figura 18 – Tela *Status* e Visão Geral do *pipeline* pela interface gráfica.



Fonte: Elaborada pelo Autor.

A Figura 18 apresenta interface quando o *pipeline* tem sua configuração finalizada. No destaque 1 na imagem, o botão ***Build Now*** é responsável pela execução completa do *Pipeline*. Quando a execução é finalizada, será apresentado o destaque de número 2 na imagem, esse destaque indica o status e a visão geral do *job*. Nele, estão contidos a consulta ao *Workspace* do projeto, histórico de testes executados e ultimas mudanças no projeto. Para informações mais detalhadas do *job*, como *logs* da execução, o destaque 3 da imagem apresenta essa função.

5.2.2 GitLab

O *GitLab* é uma ferramenta gratuita e de código aberto, porém, somente a versão *Community Edition* (CE) possui o benefício da gratuidade. A versão *Enterprise Edition* (EE) é paga, e como vantagem, essa versão oferece mais recursos do que a versão *Community Edition* (CE). Portanto, por possuir cobranças para usar a versão *Enterprise Edition* (EE), o *GitLab* atente parcialmente o critério de softwares gratuitos descrito na seção 5.1.

O *GitLab* está disponível em *Software as a Service* (SaaS) oferecendo seus serviços completamente na nuvem, atendendo o critério *Software as a Service* mencionado na seção 5.1. No segmento *Software as a Service*, o *GitLab* também oferece planos gratuitos, porém, com recursos limitados com relação a versão paga. A versão utilizada no presente trabalho é a *Omnibus Community Edition* (CE), essa versão e a *Self-Hosted* ou seja, instalada localmente e os custos relacionados são somente da Infraestrutura no qual vai hospeda-lo.

O *Gitlab* é baseado e *Git* e tem como uma grande vantagem a união de várias ferramentas em uma única, além de possuir muitas integrações, dentre elas: *Docker, Amazon Web Services* (AWS) *Microsoft Azure* e *Kubernetes*. É possível entregar o *Gitlab* com outras ferramentas de Entrega Contínua até mesmo com o próprio *Jenkins* (GITLAB, 2021).

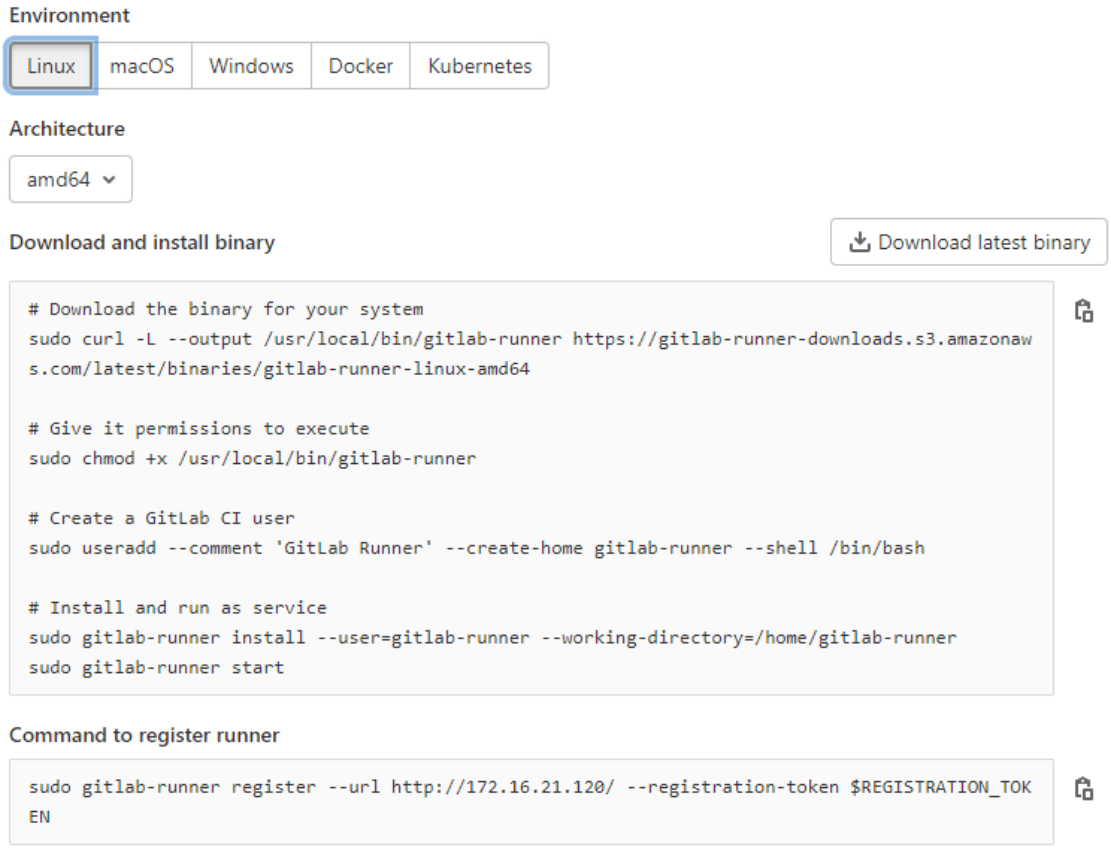
Portanto no quesito integração, o *GitLab* atende ao critério relacionado à ferramentas integradas descrito seção 5.1. Entretanto, por ser baseado em *Git*, não há suporte para outros versionadores, nesse contexto, organizações que possuem versionadores diferentes do *GitLab*, não conseguem fazer uso da ferramenta, sendo necessário a migração dos projetos para o *GitLab*.

- **Configuração Inicial**

Para que o *GitLab* possa executar os seus *pipelines*, é necessário a configuração do *GitLab Runner*. Essa aplicação deve ser instalada obrigatoriamente em uma máquina separada

de onde está instalada o *Gitlab*, para não causar conflito entre as ferramentas.

Figura 19 – Instalação do *GitLab Runner*.



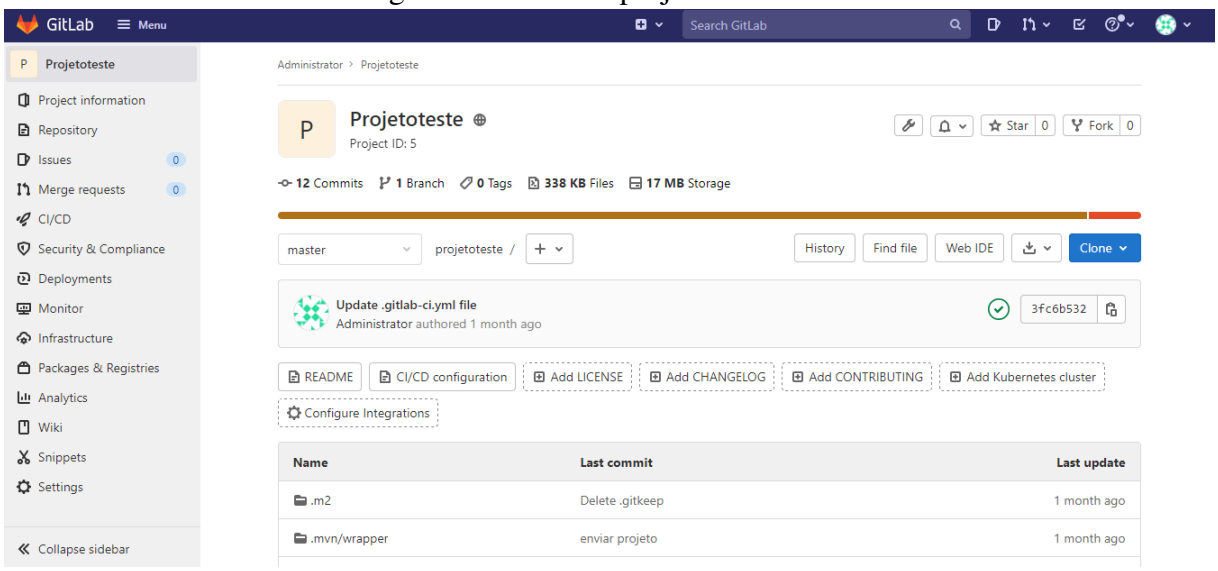
Fonte: Elaborada pelo Autor.

A Figura 19 apresenta o passos utilizados para instalar e registrar o *GitLab Runner* na instância do *GitLab*. Esse passos foram elaborados de acordo com a máquina virtual destinada ao *GitLab Runner*.

- ***Pipeline* GitLab**

Para iniciar o *pipeline*, o projeto utilizado como teste, alocado no *GitHub*, foi transferido para o *Gitlab*, pois este somente manipula projetos alocados nele, ou seja, não aceita outros versionadores de código. A Figura 20 apresenta a tela do projeto na interface gráfica do *GitLab*. Nos submenus à esquerda da tela, são apresentadas algumas funcionalidades do projeto, para criar o *pipeline* o menu ***CI/CD*** deve ser acionado e em seguida o Botão ***Editor***

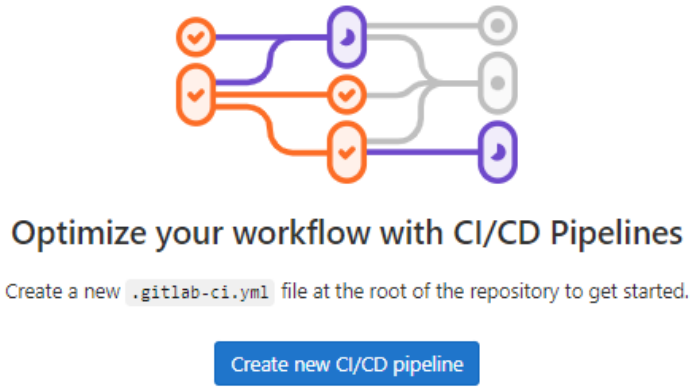
Figura 20 – Tela do projeto no *GitLab*.



Fonte: Elaborada pelo Autor.

O arquivo do *pipeline* criado no *GitLab* é chamado *.gitlab-ci.yml*, assim como no *Jenkins*, o arquivo é estruturado em estágios, cada estágio é responsável por uma determinada atividade do *pipeline*. A Figura 21 apresenta a tela para realizar a criação do arquivo *.gitlab-ci.yml*.

Figura 21 – Tela de criação do *pipeline* no *GitLab*.



Fonte: Elaborada pelo Autor.

Com a criação do arquivo, se torna possível começar a desenvolver o código. A Figura 22 apresenta o *pipeline* proposto na versão do *Gitlab*. A Linha 1 da A Figura 22 é informado a imagem *Docker* por padrão o *GitLab Runner* executa essa ferramenta, como o projeto é em *Maven*, a imagem utilizada é a do próprio *Maven*.

Da Linha 3 a Linha 5 na Figura 22, são apresentadas as variáveis do *Maven* utilizadas no *pipeline*. A etapa de armazenamento do *cache* gerado na construção do projeto são guardados no diretórios declarados nas Linha 7 a 10. Os estágios utilizados no *pipeline* são declarados na Linha 12 a Linha 15. Nesse *pipeline* serão utilizados respectivamente os estágios: *build, test* e *deploy*.

Figura 22 – *Pipeline GitLab*.

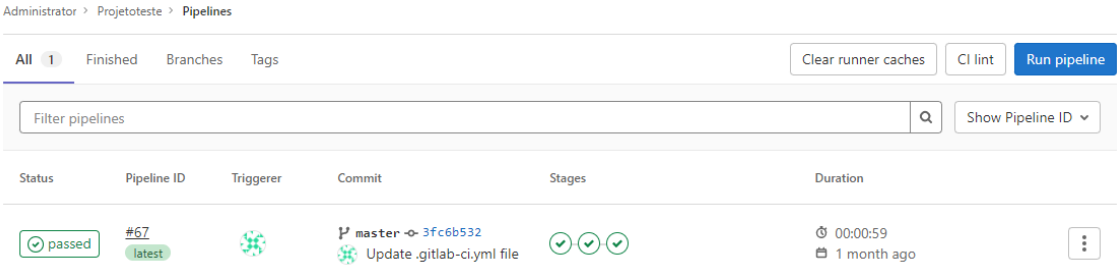
```
1  image: maven:3.8
2
3  variables:
4    MAVEN_CLI_OPTS: "-s .m2/settings.xml --batch-mode"
5    MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"
6
7  cache:
8    paths:
9    - .m2/repository/
10   - target/
11
12 stages:
13   - build
14   - test
15   - deploy
16
17 Construção:
18   stage: build
19   script:
20     - mvn $MAVEN_CLI_OPTS clean package -DskipTests=true
21   artifacts:
22     paths:
23       - target/*.jar
24
25 Testes:
26   stage: test
27   script:
28     - mvn $MAVEN_CLI_OPTS test
29
30 Implantar no Nexus:
31   stage: deploy
32   script:
33     - mvn $MAVEN_CLI_OPTS deploy:deploy-file
34         -Dmaven.test.skip=true
35         -DgroupId=com.projetoteste
36         -DartifactId=projetoteste
37         -Dversion=1.0.0 -Dpackaging=jar
38         -Dfile=target/projetoteste-1.0.0.jar
39         -DgeneratePom=true -DrepositoryId=nexus
40         -Durl=http://172.16.21.122:8081/repository/maven-repo-libs/
```

Fonte: Elaborada pelo Autor.

Na Figura 22, as Linhas 17 a 23, são responsáveis pela construção do projeto, nessa etapa o projeto é compilado e empacotado. A etapa de testes declarada nas Linha 25 a 28, fará somente testes padrão do *Maven*. Por fim, a etapa de *deploy* fará com que o artefato seja implementado para o *Nexus* no ambiente de produção simulado do cliente.

Após o desenvolvimento do *pipeline* será necessário realizar um *commit* do arquivo *.gitlab-ci.yml* no repositório, em seguida, fica disponível uma tela de *Status* e visão geral do *pipeline*, indicando execução, *logs* e se o projeto foi executado com sucesso ou não.

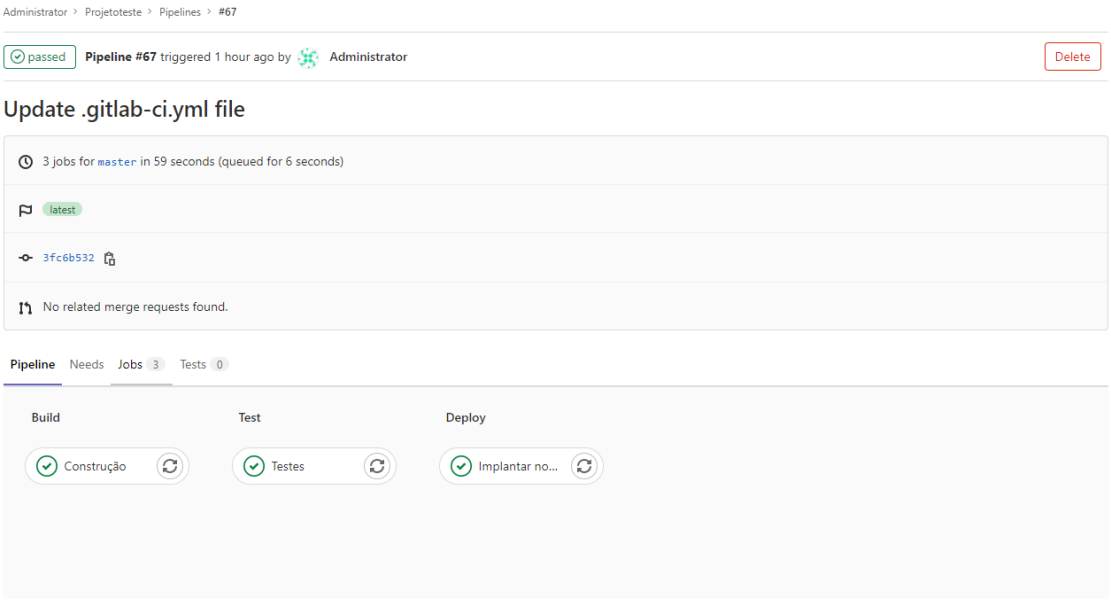
Figura 23 – Tela de *Status* e Visão Geral do *Pipeline GitLab*.



Fonte: Elaborada pelo Autor.

A Figura 23 apresenta a primeira tela de *status* e visão geral do *GitLab*, para saber mais detalhes sobre o *pipeline*, o botão ***passed*** irá redirecionar para uma tela contendo mais detalhes a respeito da execução, como: tempo de execução e *jobs* executados representados pela na Figura 24.

Figura 24 – Detalhes *Status* e Visão Geral do *Pipeline GitLab*.



Fonte: Elaborada pelo Autor.

Dessa forma, a ferramenta conta com um suporte total aos *pipelines* contendo: construção, teste e implementação, além disso, apresenta a uma visão geral e os *status*, contribuindo para o entendimento do 1º critério, definido na seção 5.1.

6 RESULTADOS

Neste Capítulo são apresentados os resultados obtidos a partir da comparação mediante aos critérios levantados. O Capítulo é dividido em duas seções, nomeadas respectivamente: Análise por critério e Sumarização dos resultados. A primeira seção deste capítulo, expõe uma análise realizada sobre o comportamento das ferramentas mediante a cada critério definido na seção 5.1 do presente estudo. A segunda seção deste capítulo, expõe um resumo dos resultados obtidos, apontando as vantagens e desvantagens de cada ferramenta.

6.1 Análise por Critério

6.1.1 C1 - Ferramenta Gratuita

Esse critério leva em consideração se as ferramentas fornecem acesso gratuito aos seus usuários. Nesse caso, tanto o *Jenkins* quanto o *GitLab* fornecem tal benefício. Porém, apenas o *Jenkins* disponibiliza acesso 100% gratuito e em código aberto. O *GitLab* oferece plano gratuito e o código aberto somente na versão *Community Edition* (CE), para ter acesso a recursos mais avançados do sistema é necessário realizar a assinatura da versão *Enterprise Edition* (EE).

6.1.2 C2 - Suporte completo aos pipelines

Esse critério leva em consideração se as ferramentas possuem as principais estruturas de um *pipeline* que são *build, test* e *deploy*. Esse critério também observa se as ferramentas também se as ferramentas possuem painéis de *status* e visão geral dos resultados. Nesse sentido, tanto o *Jenkins* quanto o *GitLab* possuem esse suporte. A seção 5.2.1, comprova que a ferramenta *Jenkins* contém as estruturas que compõem um *pipeline*, e também apresenta a tela de *status* e visão geral dos *pipelines* expostas também na seção 5.2.1, portanto *Jenkins* atende o critério C2.

A seção 5.2.2 mostra que o *GitLab* assim como o *Jenkins*, contém a estrutura completa de um *pipeline* e também possui uma tela contendo o *status* e a visão geral dos seus *pipelines*, se enquadrando também no critério C2.

6.1.3 C3 - Ferramentas Integradas

Esse critério verifica se as ferramentas possuem métodos para se integrar com outras ferramentas. Nesse caso, a ferramenta *Jenkins* possui uma arquitetura de *plugins* que são

serviços que podem ser adicionados ou removidos do aplicativo. Cada *plugin* representa uma funcionalidade específica, sendo essa funcionalidade do próprio *Jenkins* ou de uma aplicação terceira. No *GitLab* igualmente como no *Jenkins*, é possível realizar as integrações com outras aplicações, como ferramentas de monitoramento de tarefas como *Jira*, ferramentas gerenciadores de *containers* como *Docker* dentre várias outras ferramentas. Entretanto, somente o *Jenkins* tem a capacidade de se integrar com outros versionadores de código, o *GitLab* por ser em *Git*, não é possível utilizar suas ferramentas de CI e CD com outros versionadores. Portanto ambas as ferramentas se enquadram no critério C3.

6.1.4 C4 - Suporte a Software as a Service (SaaS)

Esse critério se concentra em atestar se as ferramentas possuem suporte a *Software as a Service* (SaaS) em seu uso. Durante a execução da ferramenta *Jenkins*, foi observado que ela não possui suporte a *Software as a Service* (SaaS) tendo seus custos relacionados da forma de instalação *Self-Hosted*. Apesar do o *GitLab* ter sido executado na versão *Self-Hosted* no presente estudo. A ferramenta, além do planos *Self-Hosted* também oferece suporte *Software as a Service* (SaaS) em seus planos de aquisição. Portanto, somente o *GitLab* atende o critério C4.

6.1.5 C5 - Formatos de configuração do pipeline

Esse critério leva em consideração as diferentes possibilidades de configuração do *pipeline*. O *Jenkins* possui duas opções para a configuração, que são: *Pipeline* Declarativo e o *Pipeline* utilizando a interface gráfica. O primeiro método, é escrito através de linhas de código, essa opção se enquadra para usuários preferem deixar seus *pipelines* escritos em um único arquivo de texto. Nesse formato de configuração, o painel de *status* e visão geral é segmentado de acordo com o estágio do *pipeline*, assim como mostra a Figura 12 na seção 5.2.1. O segundo método, é a configuração *pipeline* pela própria interface gráfica do *Jenkins*, tendo o mesmo funcionamento do *pipeline* declarativo. Esse formato de configuração, é mais intuitivo, pois cada estágio é apresentado de maneira mais visual para o usuário, assim como é mostrado também na seção 5.2.1.

O *GitLab* possui somente a configuração por linha de código, apresentado na seção 5.2.2. Este método, é bem semelhante ao arquivo no modo declarativo do *Jenkins*. Nele é onde configura-se o arquivo de *pipeline* chamado de *.gitlab-ci.yml* e fica localizado na raiz do projeto. Esse arquivo contém todos os estágios que o *GitLab Runner* vai executar.

6.2 Sumarização dos resultados

A Tabela 1 apresenta os resultados obtidos decorrentes da comparação entre as ferramentas de Integração e Entrega Contínua *Jenkins* e *GitLab*, fazendo o uso da implementação de ambas.

Tabela 1 – Resultado da Comparação entre as ferramentas *Jenkins* e *GitLab*.

ID	**Critério**	***Jenkins***	***GitLab***
C1	Ferramenta gratuita	Sim	Parcialmente
C2	Suporte completo aos *pipelines*	Sim	Sim
C3	Ferramentas integradas	Sim	Sim
C4	Suporte a *Software as a Service* (SaaS)	Não	Sim
C5	Formatos de configuração dos *pipelines*	2 Formatos	1 Formato

Fonte: Elaborada pelo Autor.

Do ponto de vista do *Jenkins*, os resultados obtidos na execução deste estudo foram: I) A ferramenta satisfaz o primeiro critério sendo totalmente gratuito. II) A ferramenta satisfaz o segundo critério, possuindo suporte total aos *pipelines* e tela de *status* e visão geral dos mesmos. III) A ferramenta satisfaz o terceiro critério, possuindo várias formas de integração com outras ferramentas. IV) A ferramenta não satisfaz o quarto critério, pois não oferece suporte a *Software as a Service* (SaaS). V) A ferramenta *Jenkins* apresenta dois formatos de configuração de seus *pipelines*: O *Pipeline* Declarativo e o *Pipeline* utilizado a interface gráfica.

Do ponto de vista do *GitLab*, os resultados obtidos na execução deste estudo foram: I) A ferramenta satisfaz parcialmente o primeiro critério, pois oferece a gratuidade somente a versão *Community Edition*, os demais planos são todos cobrados para o uso. II) A ferramenta satisfaz o segundo critério, possuindo suporte total aos *pipelines* e tela de *status* e visão geral dos mesmos. III) A ferramenta satisfaz o terceiro critério possuindo várias formas de integração com outras ferramentas. porém não é possível a integração com outros versionadores de código. IV) A ferramenta satisfaz o quarto critério, pois oferece suporte a *Software as a Service* (SaaS) em seu portfólio V) A ferramenta apresenta somente um formato de configuração para seus *pipelines*, que o formato por linhas de código.

7 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho apresentou o conceito de *DevOps*, sua filosofia, suas principais práticas e seu ciclo de funcionamento. Também foi discutido como as organizações que não adotam as práticas *DevOps* podem ter perdas, como: O atraso no desenvolvimento e entrega dos *softwares* para seus clientes. Foi levantado também os benefícios que uma organização ganha ao adotar a cultura *DevOps*. Além disso, foi apresentado as principais ferramentas e a importância delas para a continuidade do ciclo *DevOps*. Nesse contexto, esta monografia se concentrou em avaliar ferramentas que dão suporte as práticas de Integração Contínua e Entrega Contínua. Esta obra é destinada à profissionais da área *DevOps*, principalmente ligados a Integração e Entrega Contínua.

Este trabalho de conclusão pretendeu realizar um estudo comparativo entre as ferramentas *Jenkins* e *GitLab* no quesito Integração e Entrega Contínua baseando-se em cinco critérios. O primeiro critério verificou se as ferramentas oferecem gratuidade em seu uso. O segundo critério analisou se ambas as ferramenta fornecem todas as etapas de um *pipeline* como *build, test* e *deploy*, inclusive, se possuem telas de *status* visão geral dos *pipelines* executados. O terceiro critério verificou se se ambas as ferramentas eram capazes de se integrar a outras para a continuidade do ciclo *DevOps*. O quarto critério analisou se ambas as ferramentas oferecem suporte a *Software as a Service* (SaaS). Por fim, o quinto e ultimo critério, apresentou os formatos de configuração de cada ferramenta disponibiliza.

Com base nesses resultados, conclui-se que tanto *Jenkins* quanto o *GitLab* são ótimas ferramentas para desempenhar as atividade de Integração e Entrega Contínua, porém, para organizações que possuem baixo orçamento e não querem gastar muito com *software*, o *Jenkins* torna-se a melhor opção por ser 100% gratuito, apesar do *GitLab* também oferecer recursos gratuitos, mas, esses recursos são limitados. Para organizações que possuem versionadores diferentes do *GitLab*, o uso da ferramenta se torna inviável, pois seu *pipelines* funcionam somente com os projetos de dentro do *GitLab*. Nesse cenário, o *Jenkins* leva uma vantagem, pois é capaz de realizar integrações com outros versionadores para desempenhar suas atividades. Em um cenário em que a organização não quer se preocupar com obrigações relacionadas a infraestrutura o *GitLab* apresenta uma vantagem sobre o *Jenkins*, pois oferece suporte a *Software as a Service* (SaaS), no caso do *Jenkins* para seu uso é necessário ter esse cuidado, mesmo sendo alocado em uma nuvem do tipo *Infrastructure as a Service* (Infraestrutura como um Serviço) o cuidado ainda vai existir, pois nesse forma de alocação ainda é necessário cuidados relacionados

com a infraestrutura como a versão do *Jenkins* e o sistema operacional que o suporta. Para usuários que gostam elaborar os *pipelines* por linhas de código, tanto o *GitLab* quanto o *Jenkins* são capazes de suprir essa característica, porém, para usuários que preferem configurar tudo por interface gráfica, somente o *Jenkins* é capaz de suprir essa preferência.

Como trabalhos futuros, sugere-se realizar estudos incluindo outras ferramentas, adicionando mais critérios de comparação relacionado a performance das aplicações tendo como parâmetro o tempo de execução de *pipelines*.

REFERÊNCIAS

ARACHCHI, S.; PERERA, I. Continuous integration and continuous delivery pipeline automation for agile software project management. In: IEEE. **2018 Moratuwa Engineering Research Conference (MERCon)**. [S.l.], 2018. p. 156–161.

AREFEEN, M. S.; SCHILLER, M. Continuous integration using gitlab. **Undergraduate Research in Natural and Clinical Science and Technology Journal**, v. 3, p. 1–6, 2019.

AWS. **O que é o DevOps?** 2021. <https://aws.amazon.com/pt/devops/what-is-devops/>. Acesso em 20 de Setembro de 2021.

BASS, L.; WEBER, I.; ZHU, L. **DevOps: A software architect's perspective**. [S.l.]: Addison-Wesley Professional, 2015.

BOBROV, E.; BUCCHIARONE, A.; CAPOZUCCA, A.; GUELFI, N.; MAZZARA, M.; NAUMCHEV, A.; SAFINA, L. Devops and its philosophy: Education matters! In: **Microservices**. [S.l.]: Springer, 2020. p. 349–361.

CHATTERJEE, R. Security in devops and automation. In: **Red Hat and IT Security**. [S.l.]: Springer, 2021. p. 65–104.

CHEN, L. Continuous delivery: Huge benefits, but challenges too. **IEEE software**, IEEE, v. 32, n. 2, p. 50–54, 2015.

CLOUD, G. **Tecnologia do DevOps: integração contínua**. 2021. <https://cloud.google.com/architecture/devops/devops-tech-continuous-integration?hl=pt-br>. Acesso em 21 de Setembro de 2021.

DIMIKJ, V. **Create a CI/CD pipeline with GitHub Actions**. 2020. <https://www.north-47.com/knowledge-base/create-a-ci-cd-pipeline-with-github-actions/>. Acesso em 22 de Setembro de 2021.

EBERT, C.; GALLARDO, G.; HERNANTES, J.; SERRANO, N. Devops. **Ieee Software**, IEEE, v. 33, n. 3, p. 94–100, 2016.

FOWLER, M.; FOEMMEL, M. **Continuous integration**. 2006.

GITLAB. **GitLab integrations**. 2021. <https://docs.gitlab.com/ee/integration/>. Acesso em 03 de Novembro de 2021.

GITLAB. **GitLab Runner**. 2021. <https://docs.gitlab.com/runner/>. Acesso em 18 de Outubro de 2021.

GITLAB. **Simplify your workflow with GitLab**. 2021. <https://about.gitlab.com/stages-devops-lifecycle/>. Acesso em 30 de Agosto de 2021.

GITLAB. **What is GitLab?** 2021. <https://about.gitlab.com/what-is-gitlab/>. Acesso em 17 de Outubro de 2021.

GLEIN, R.; PERLOFF, A.; ULMER, K. Continuous integration of fpga designs for cms. In: **Topical Workshop on Electronics for Particle Physics TWEPP2019**. [S.l.: s.n.], 2019. v. 2, p. 6.

GNU. **What is Free Software?** 2021. <https://www.gnu.org/philosophy/free-sw.html>. Acesso em 01 de Novembro de 2021.

GOBBI, R. **Qual o nível de maturidade DevOps da sua empresa?** 2019. <https://blog.4linux.com.br/qual-o-nivel-de-maturidade-devops-da-sua-empresa/>. Acesso em 22 de Setembro de 2021.

HETHEY, J. M. **GitLab Repository Management**. [S.l.]: Packt Publishing, 2013.

HUMBLE, J.; FARLEY, D. **Continuous delivery: reliable software releases through build, test, and deployment automation**. [S.l.]: Pearson Education, 2010.

HUMBLE, J.; MOLESKY, J. Why enterprises must adopt devops to enable continuous delivery. **Cutter IT Journal**, v. 24, n. 8, p. 6, 2011.

HÜTTERMANN, M. **DevOps for developers**. [S.l.]: Apress, 2012.

ITKONEN, J.; UDD, R.; LASSENIUS, C.; LEHTONEN, T. *et al.* Perceived benefits of adopting continuous delivery practices. In: **ESEM**. [S.l.: s.n.], 2016. p. 42–1.

JENKINS. **Jenkins User Documentation**. 2021. <https://www.jenkins.io/doc/>. Acesso em 30 de Agosto de 2021.

JENKINS. **Participate and Contribute**. 2021. <https://www.jenkins.io/participate/>. Acesso em 04 de Dezembro de 2021.

JENKINS. **Plugins Index**. 2021. <https://plugins.jenkins.io/>. Acesso em 02 de Novembro de 2021.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. Problems, causes and solutions when adopting continuous delivery—a systematic literature review. **Information and Software Technology**, Elsevier, v. 82, p. 55–79, 2017.

MARCONI, M. d. A.; LAKATOS, E. M. **Fundamentos de metodologia científica**. [S.l.]: 5. ed.-São Paulo: Atlas, 2003. 174-183 p.

MEYER, M. Continuous integration and its tools. **IEEE software**, IEEE, v. 31, n. 3, p. 14–16, 2014.

MOHAMMAD, S. M. Continuous integration and automation. **International Journal of Creative Research Thoughts (IJCRT), ISSN**, p. 2320–2882, 2016.

PITTET, S. **Integração contínua vs. entrega contínua vs. implementação contínuat**. 2021. <https://www.atlassian.com/br/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>. Acesso em 30 de Agosto de 2021.

PRIMECONTROL. **Integração Contínua, Entrega Contínua e Implantação Contínua**. 2019. <https://www.primecontrol.com.br/integracao-continua-entrega-continua-e-implantacao-continua/>. Acesso em 30 de Setembro de 2021.

PUROHIT, K. Executing devops & ci/cd, reduce in manual dependency. **IJSDR**, v. 5, n. 6, p. 511–515, 2020.

RAVICHANDRAN, A.; TAYLOR, K.; WATERHOUSE, P. Devops in the ascendency. In: **DevOps for Digital Leaders**. [S.l.]: Springer, 2016. p. 3–14.

REDHAT. **O que é um pipeline de CI/CD?** 2020. <https://www.redhat.com/pt-br/topics/devops/what-cicd-pipeline>. Acesso em 22 de Setembro de 2021.

REDHAT. **Introdução ao DevOps**. 2021. <https://www.redhat.com/pt-br/topics/devops>. Acesso em 01 de Dezembro de 2021.

RÉVÉSZ, A.; PATAKI, N. Visualisation of jenkins pipelines. **Acta Cybernetica**, University of Szeged, 2021.

RITI, P. **Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes**. [S.l.]: Springer, 2018.

SALESFORCE. **SaaS: o que é Software as a Service?** 2021. <https://www.salesforce.com/br/saas/>. Acesso em 07 de Novembro de 2021.

SONI, M. **Jenkins Essentials**. [S.l.]: Packt Publishing, 2015.

TOTVS, E. **DevOps: Conceito, objetivo e dicas para aplicar a metodologia**. 2021. <https://www.totvs.com/blog/developers/metodologia-devops/>. Acesso em 20 de Setembro de 2021.

TRENDS, G. **Comparar Ferramentas**. 2021. <https://trends.google.com.br/trends/?geo=BR>.

VALENTE, M. T. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**. 1. ed. [S.l.]: Independente, 2020. 408 p.

VIRMANI, M. Understanding devops & bridging the gap from continuous integration to continuous delivery. In: IEEE. **Fifth international conference on the innovative computing technology (intech 2015)**. [S.l.], 2015. p. 78–82.

WIEDEMANN, A.; FORSGREN, N.; WIESCHE, M.; GEWALD, H.; KRCMAR, H. Research for practice: the devops phenomenon. **Communications of the ACM**, ACM New York, NY, USA, v. 62, n. 8, p. 44–49, 2019.

YILDIRIM, A. **DevOps Lifecycle: Continuous Integration and Development**. 2019. <https://medium.com/@yildirimabdrhm/devops-lifecycle-continuous-integration-and-development-e7851a9c059d>. Acesso em 12 de Setembro de 2021.