Deyvison Luiz Guedes Dantas

Proposta de um Ambiente para o Desenvolvimento de Softwares com Integração Contínua: um Estudo de Caso

Natal – RN

Dezembro de 2023

Deyvison Luiz Guedes Dantas

Proposta de um Ambiente para o Desenvolvimento de Softwares com Integração Contínua: um Estudo de Caso

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Prof. Dr. Carlos Manuel Dias Viegas

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Engenharia de Computação e Automação – DCA

Curso de Engenharia de Computação

Natal – RN

Dezembro de 2023

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Dantas, Deyvison Luiz Guedes.
Proposta de um ambiente para o desenvolvimento de softwares com integração contínua: um estudo de caso / Deyvison Luiz Guedes Dantas. - 2023.
39 f.: il.

Monografia (graduação) - Universidade Federal do Rio Grande do Norte, Curso de Engenharia de Computação, Natal, RN, 2023.
Orientação: Prof. Dr. Carlos Manuel Dias Viegas.

1. Ambiente de desenvolvimento de software - Monografia. 2. Infraestrutura de TI - Monografia. 3. Integração contínua - Monografia. 4. Contêineres Docker - Monografia. I. Viegas, Carlos Manuel Dias. II. Título.

RN/UF/BCZM CDU 004.4

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

Deyvison Luiz Guedes Dantas

Proposta de um Ambiente para o Desenvolvimento de Softwares com Integração Contínua: um Estudo de Caso

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Prof. Dr. Carlos Manuel Dias Viegas

Trabalho aprovado. Natal – RN, 11 de Dezembro de 2023:

Prof. Dr. Carlos Manuel Dias Viegas - Orientador
DCA-UFRN

Prof. Dr. Ivanovitch Medeiros Dantas Da Silva
DCA-UFRN

Prof. Dr. Eduardo De Lucena Falcão
DCA-UFRN

Natal – RN
Dezembro de 2023

Dedico a minha esposa Esther Laryssa Lima Correia Dantas, a minha filha Elisa Correia Dantas e aos meus pais Luiz Batista Dantas e Maria Eronice Guedes Dantas

AGRADECIMENTOS

A Deus, dono de todo o conhecimento, que me deu forças e ânimo nesse processo, e sempre age de misericórdia comigo, mesmo sem merecer. Que seja tudo para honrar o Seu Nome.

A minha esposa Esther, que foi paciente, esteve ao meu lado, viveu comigo as dificuldades e sempre me motivou a persistir no curso e me formar.

Aos meu pais, homem e mulher trabalhadores, que sempre me cuidaram, amaram e incentivaram a estudar, proporcionando o melhor para isso.

Ao professor Viegas, que é uma referência de docente na UFRN, por sua orientação e zelo ao ensinar, sua paciência e compreensão, estando sempre disposto a ajudar.

Aos colegas de faculdade, que sofreram junto comigo durante o curso e me ajudaram nas disciplinas.

A Centro de Tecnologia da Informação e colegas da Câmara Municipal de Paramirim, que possibilitou a infraestrutura para este trabalho e abriu espaço para o desenvolvimento da ideia.

"Tudo tem o seu tempo determinado,
e há tempo para todo o propósito debaixo do céu"
(Bíblia Sagrada, Eclesiastes 3:1)

RESUMO

Partindo da demanda de construção de sistemas de tecnologia da informação para melhoria de processos e trabalhos internos em instituições públicas e seus interesses no desenvolvimento de *softwares* próprios, é essencial que existam ambientes computacionais e infraestruturas que possibilitem tal fluxo de desenvolvimento de maneira eficiente. Esses ambientes são hospedados em servidores, seja em *data centers* locais ou em servidores na nuvem, os quais são configurados com base nos requisitos específicos do desenvolvimento e nas características da aplicação a ser desenvolvida. Dentre as práticas de desenvolvimento, destaca-se a Integração Contínua (CI), originária da cultura DevOps, que tem sido amplamente adotada com o intuito de aprimorar o fluxo de integração, testes, correções e atualizações de versões de uma aplicação. Isso é alcançado por meio de automações, proporcionando agilidade e eficiência no processo de entrega e desenvolvimento de *software*. Compreendendo a necessidade do desenvolvimento de *software* em instituições públicas e a importância da adoção de boas práticas em seu processo, este trabalho tem como objetivo elaborar uma proposta de ambiente para o desenvolvimento de *softwares* com a prática da Integração Contínua, a qual poderá ser implementado em instituições públicas. Para ilustrar a viabilidade dessa proposta, será apresentado um estudo de caso.

Palavras-chaves: Ambiente de desenvolvimento de *software*; Infraestrutura de TI; Integração Contínua; Contêineres Docker.

ABSTRACT

Based on the demand for building information technology systems to improve processes and internal work in public institutions and their interests in developing their own software, it is essential that there are computing environments and infrastructures that enable such a flow of development in an efficient manner. These environments are hosted on servers, either in local data centers or on cloud servers, which are configured based on the specific development requirements and the characteristics of the application to be developed. Among the development practices, Continuous Integration (CI) stands out, originating from the DevOps culture, which has been widely adopted with the aim of improving the flow of integration, testing, corrections and version updates of an application. This is achieved through automation, providing agility and efficiency in the software delivery and development process. Understanding the need for software development in public institutions and the importance of adopting good practices in its process, this work aims to develop a proposal for an environment for software development with the practice of Continuous Integration, which can be implemented in public institutions. To illustrate the feasibility of this proposal, a case study will be presented.

Keywords: Software development environment; IT infrastructure; Continuous integration; Docker Containers.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AWS	*Amazon Web Services*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CMP	*Câmara Municipal de Parnamirim*
HTML	*HyperText Markup Language*
PHP	*PHP: Hypertext Preprocessor*
SaaS	*Software as a Service*
SO	*Sistema Operacional*
VM	*Virtual Machine*
YAML	*YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo é apresentado o contexto em que esse trabalho está inserido, apresentando o conceito de DevOps, o objetivo, a motivação e sua estrutura.

1.1 Contextualização

A informatização de processos em empresas e instituições tem avançado muito nos últimos anos, todos os sistemas de computadores utilizados para a modernização passam pelo processo de desenvolvimento, que acontece em diversas etapas até a disponibilização ao usuário final. Todo esse processo acontece por completo em ambientes específicos, como ambiente de desenvolvimento, de testes, homologação e produção (HOSTINGER, 2023). Este último é onde as versões finais do sistemas são disponibilizados para o acesso por parte dos usuários finais.

Os ambientes do processo de desenvolvimento são montados em servidores, em nuvem, ou localmente em *data centers* próprios. Esses ambientes demandam uma infraestrutura específica que atenda as necessidades dos sistemas que serão desenvolvidos ali. Um ambiente apropriado é importante para o desenvolvimento de aplicações, ambientes que sejam seguros e protegidos, com acessos e permissões controladas, que também que sejam estáveis e eficientes para o propósito.

1.2 DevOps

O DevOps (junção das palavras "desenvolvimento"e "operação") é um modelo que combina práticas, filosofias e ferramentas que aumentam a capacidade de uma empresa desenvolvedora de *software* em distribuir seus serviços em alta velocidade. É uma prática de engenharia de *software* para unir o desenvolvimento com a operação, de forma a integrar essas áreas e alcançar uma maior qualidade nas entregas. Quando essas áreas não estão integradas, é possível que falhe a comunicação entre as equipes, resultando em atrasos, retrabalhos e baixa qualidade (GUEDES, 2018).

Atualmente, o DevOps ganha destaque por causa dos processos de integração e entrega contínua. Processos que auxiliam na padronização de ambientes de desenvolvimento, homologação e produção, além do gerenciamento e controle sobre o ambiente e infraestrutura (GUEDES, 2018).

O processo de integração contínua, que faz parte das as práticas de DevOps, que vem sendo adotado por muitas empresas no fluxo de desenvolvimento de *softwares* tem

como principais objetivos encontrar e investigar *bugs* rapidamente, melhorar a qualidade do *software* e reduzir o tempo que leva para validar e lançar novas atualizações de *software* (AWS, 2023).

1.3 Objetivo

O presente trabalho tem como objetivo geral propor a criação de um ambiente de desenvolvimento que incorpore a integração contínua em seu fluxo, especialmente voltado para instituições públicas que pretendem iniciar o desenvolvimento de *softwares* por meio da implementação de soluções de tecnologia da informação. Essa abordagem visa não apenas modernizar os processos internos dessas instituições, mas também otimizar a entrega de soluções tecnológicas, promovendo eficiência, segurança e agilidade no desenvolvimento e implementação de *softwares*.

Os objetivos específicos do trabalho são:

- Montar uma infraestrutura para um ambiente de desenvolvimento;
- Projetar o fluxo e a integração entre as ferramentas que comporão o ambiente;
- Criar ambiente com os serviços em contêineres Docker;
- Desenvolver um *pipeline* que executará a a Integração contínua e
- Criar um modelo que possa ser aplicado em outras instituição que tenham interesse.

1.4 Motivação

Como servidor público da CMP - Câmara Municipal de Parnamirim - observando o interesse, o planejamento e trabalhos iniciais a respeito do desenvolvimento de *softwares* por meio do Centro de Tecnologia da Informação da instituição para atender necessidades dos trabalhos legislativos, houve a motivação de criação de um ambiente que proporcionasse o desenvolvimento de sistemas com a prática da integração contínua.

O desenvolvimento de softwares por parte da própria instituição traz benefícios, como maior controle e domínio sobre os sistemas utilizados, evita transtornos de migração de um sistema para outro, quando nas licitações de sistemas, seja necessário a troca dos sistemas utilizados, e a redução de despesas, já que não será mais necessário pagar por softwares de terceiros, pois a instituição produzirá seus próprios sistemas.

A motivação da implementação da integração contínua no ambiente, vem da cultura DevOps, que visa um bom fluxo de desenvolvimento e a entrega rápida de lançamentos de

versões, correções e atualizações das aplicações entre outras práticas, fazendo com que o processo de desenvolvimento aconteça de forma mais eficiente.

1.5 Estrutura do Trabalho

Além desta Introdução, este trabalho está dividido em mais 4 capítulos. O Capítulo 2 trata da fundamentação teórica, apresentando as ferramentas que são utilizadas no trabalho. No capítulo 3, é descrito o desenvolvimento, mostrando como cada etapa e elemento necessário foi instalado e configurado. O capítulo 4 apresenta o teste realizado e a validação da proposta por meio do estudo de caso. Por fim, no capítulo 5, conta com as considerações finais e a exposição de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados trabalhos relacionados e os conceitos e detalhes das ferramentas utilizadas para o desenvolvimento do trabalho.

2.1 Trabalhos relacionados

Nesta sessão, são apresentados trabalhos relacionados aos temas de DevOps, integração e entrega Contínua.

Buffa (2021) apresentou um estudo prático sobre a implementação e viabilização da cultura DevOps para pequenas empresas e *startups* em seu trabalho, explorando apenas ferramentas *open source*, objetivando gerar soluções para suprir as demandas no processo de desenvolvimento de *software*. Como estudo de caso, foi utilizado a implementação de um simples sistema de votação. O autor relatou dificuldades no desenvolvimento do trabalho, como para realizar a configuração da ferramenta de orquestração de contêineres Kubernetes, porém destacou a importância de utilizar a cultura DevOps e ferramentas de código aberto de forma a facilitar a manutenção de uma aplicação computacional, cumprido os objetivos do trabalho de mostrar a viabilidade da implantação do DevOps em pequenas empresas e *startups*.

Lima (2021) em seu trabalho de conclusão de curso, tinha como objetivo apresentar um comparativo nos quesitos de CI/CD entre duas ferramentas bastante utilizadas, o Jenkins e GitLab, com a finalidade de mostrar qual pode ser usada em contextos específicos. A comparação usou critérios de avaliação e a implantação de *pipelines*. O autor adotou como critérios o (1) fato da ferramenta ser gratuita, (2) o suporte completo aos *pipelines*, (3) as ferramentas integradas, (4) o suporte a SaaS (*Software as a Service*) e (5) os formatos de configuração do *pipeline*. No trabalho foi concluído que ambas as ferramentas são ótimas para desempenhar as atividades propostas, mas que o Jenkins leva vantagem para organizações com baixo orçamento, empresas que usam o controle de versão de código diferente do GitLab e para usuário que preferem realizar configuração por interface gráfica. Já o GitLab levou vantagem a respeito de organizações que não querem se preocupar com obrigações relacionadas a infraestrutura. E tanto o Jenkins, quanto o GitLab atendem a característica de usuários que preferem elaborarem a *pipelines* por linhas de código.

Silva (2020) propôs desenvolver uma metodologia mais ágil em comparação com tradicionais operações de criação de infraestrutura, utilizando a metodologia GitOps - que usa repositórios Git como única fonte de informações para oferecer infraestrutura como código - como forma de automatizar tais processos. O trabalho explicou a cultura DevOps e

seus conceitos, como fundamento para a implantação do GitOps, demonstrando o processo desde seu planejamento até a criação da infraestrutura de uma aplicação. A conclusão obtida pelo autor é que o processo realizado através da integração e entrega contínua (CI/CD) manteve um histórico único de alterações no sistema de controle de versão e permitiu a redução de tempo na resolução de conflitos durante o ciclo de desenvolvimento da infraestrutura e da aplicação.

2.2 Contêineres e Docker

A seguir são apresentados os conceitos de contêineres e do Docker.

2.2.1 Contêineres

O contêiner é o conjunto de uma aplicação e suas respectivas dependências de forma isolada, que compartilham o o *kernel* do sistema operacional do *host*, seja uma máquina física ou virtual (FERNANDO, 2023). Todos os arquivos necessários para sua execução são disponibilizados por meio de uma imagem individual (REDHAT, 2022). Os contêineres tem similaridade com as máquinas virtuais, porém são mais integrados ao sistema operacional e mais leves, que por compartilhar o kernel, é proporcionado um melhor desempenho por conta do gerenciamento único dos recursos.

A imagem de um contêiner é bem sucinta, nela há apenas o necessário para que a execução da aplicação funcione, isso faz com que haja um custo operacional menor, quando comparado com a aplicação sendo executada nativamente no sistema operacional (FERNANDO, 2023), um dos motivos é por causa do compartilhamento dos recursos com a máquina *host*.

Quando são utilizadas máquina virtuais, é virtualizado todo o seu *hardware*, utilizando mais recursos da máquina *host*. Isso não ocorre quando são utilizados contêineres, pois os recursos do *host* são compartilhados, dessa forma, aumenta a capacidade de executar mais contêineres em apenas um *host* em relação a se executar em máquinas virtuais (FERNANDO, 2023). Na Figura 1, é possível ver a diferença entre quando as aplicações são em máquinas virtuais e em contêineres, respectivamente.

Um ponto importante da utilização de contêineres é a sua portabilidade. Um contêiner pode ser executado em qualquer ambiente - no nosso caso, que tenha o Docker instalado - independente de qual ele foi criado, seja Linux, Windows ou MacOS, sem ser necessário atenção com as dependências, pois já estão todas presentes nele (FERNANDO, 2023).

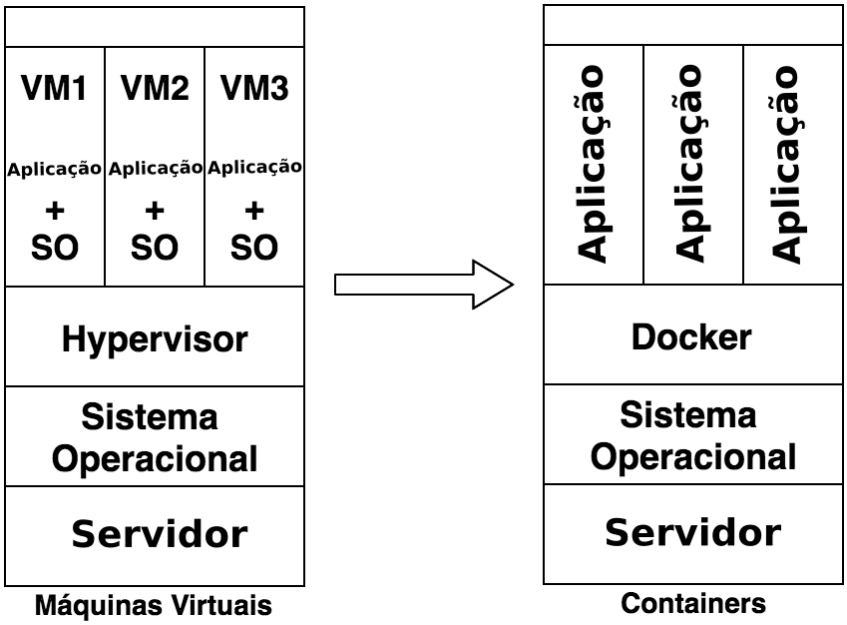

Figura 1 – Comparação da execução de aplicações em máquinas virtuais e contêineres

Fonte: Adaptado de FERNANDO, Jeferson. DESCOMPLICANDO O DOCKER, 2023.

2.2.2 Docker

Para o gerenciamento dos contêineres existe algumas plataformas, entre elas o Docker, o qual é usado neste trabalho. O Docker é uma plataforma *open source*, que permite o desenvolvimento, implantação, execução a atualização e administração de ambientes isolados, possibilitando o empacotamento das aplicações em contêineres, facilitando a criação, implantação, cópia e migração de um ambiente para outro com maior flexibilidade (GUEDES, 2018).

O Docker possui características de ser leve e rápido, sendo uma alternativa viável e econômica, tendo um bom desempenho em ambientes de alta densidade para implantações de pequeno e médio porte (DOCKER DOCS, 2023), oferecendo ferramentas ideais para os fluxos de trabalho de integração contínua, que serão utilizadas neste trabalho.

Algumas ferramentas do Docker são importantes serem mencionadas, como o Dockerfile, que é um arquivo de texto simples, contendo instruções sobre como a imagem de um contêiner Docker é criada. O Dockerfile automatiza o processo de criação da imagem, consistindo em uma lista de instruções em Interface de linha de comando (CLI - command-line interface) que são executadas pelo Docker para montar a imagem (IBM, 2023).

As imagens do Docker contêm um o código-fonte da aplicação, além de todas as ferramentas, bibliotecas e estrutura que o código precisa para ser executado como um contêiner. Uma imagem ao ser ser executada se torna uma instância do contêiner (IBM, 2023). As imagens podem ser armazenadas em repositórios de imagens, como no Docker

Hub, o repositório público de imagens do Docker, que contém milhares de imagens, ou em um Docker Registry, que pode ser instalado localmente para o armazenamento de imagens privadas de forma gratuita.

Outra ferramenta do Docker bastante utilizada para a orquestração de contêineres é o Docker Compose, que gerencia aplicações de vários contêineres, através da criação de um arquivo YAML, especificando quais serviços estarão incluídos nas aplicações. Com um único comando o Docker Compose pode implementar e executar contêineres. O Docker Compose também pode ser usado para definir volumes persistentes para área de armazenamento, especificar nós de base e documentar e configurar estruturas de serviço (IBM, 2023).

2.3 Integração Contínua

A Integração Contínua (*Continuous integration*), conhecida por sua sigla em inglês CI, é uma prática no desenvolvimento de softwares, especialmente na cultura DevOps, adotada por empresas de diversos portes no desenvolvimento de *software*, que leva os desenvolvedores de software a introduzir pequenas alterações no código-fonte constantemente nos repositórios de controle de versão. É um método de desenvolvimento em que os desenvolvedores geralmente incorporam código em um sistema totalmente integrado, que com a construção automatizada e os testes automatizados podem então verificar qualquer integração (MOHAMMAD, 2016).

Diversas vantagens são oferecidas com a integração contínua como a identificação rápida de problemas de integração, garantia de uma melhor qualidade dos códigos, redução de problemas no *deploy* e diminuição de erros por falhas humanas. Os principais objetivos da integração contínua são encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software (AWS, 2023).

O fluxo da integração contínua começa quando desenvolvedores enviam suas alterações no código da aplicação a um repositório de controle de versão, como o Git. Quando seu *commit* chega ao repositório, um *pipeline* é disparado e iniciado testes sobre o código, sendo validados, o *pipeline* irá pra o estágio de construção e após de *deploy*. O *deploy* contínuo amplia o processo de integração através da entrega de todas as alterações feitas ao um ambiente de teste ou um ambiente de produção (MOHAMMAD, 2016) .

Para implementação da integração contínua há diversas ferramentas disponíveis como o Jenkins, *software open source* feito em Java, bastante utilizado para integração contínua, que permite a construção rápida e testes automatizados; o TeamCity, uma solução empresarial versátil, mas com limitações em seu plano gratuito; o Bamboo, uma ferramenta da Atlassian, que conta com uma interface de arrastar e soltar; o Buddy,

ferramenta desenvolvida para operar com projetos que utilizam Bitbucket e GitHub; e o GitLab CI, plataforma *open source* de integração contínua, que possui facilidade de instalar e configurar projetos hospedados no GitLab, implementando testes e construção (MOHAMMAD, 2016). Este último será o adotado para este trabalho.

2.4 Git, GitLab e GitLab CI/CD

A seguir, são apresentados os conceitos das ferramentas Git, Gitlab e GitLab CI/CD.

2.4.1 Git

O Git é um sistema *open source* de controle de versão de códigos, utilizado largamente em projetos de desenvolvimento de *software* (ATLASSIAN, 2023), em que é possível a contribuição em um projeto por diversos desenvolvedores simultaneamente, de forma controlada e totalmente organizada. Existem algumas ferramentas que fornecem funcionalidades aplicadas ao Git como o GitHub e GitLab, por exemplo. A seguir serão apresentados alguns conceitos e comandos do Git, importantes para o entendimento do desenvolvimento deste trabalho.

git add: Comando que adiciona os arquivos alterados informando que estão preparados para ser enviados ao repositório, mas ainda não são enviados de fato.

git commit -m "mensagem": Comando que realiza o *commit*, o qual cria uma revisão dos arquivos adicionados (com o *git add*)com um número e um comentário.

git push: Comando que publica e envia ao repositório os *commits* com as alterações já realizados, sendo necessário a autenticação.

git pull: Comando que traz todas as modificações dos aquivos que estão no repositório para seu projeto localmente.

2.4.2 GitLab e GitLab CI/CD

O GitLab é uma plataforma *open source*, utilizada por muitas empresas de desenvolvimento de *software*, que combina vários recursos, incluindo gerenciamento de código, rastreamento de bugs, controle de versão e outras funções. É uma plataforma completa para equipes de desenvolvimento de software que buscam uma solução integrada para gerenciar todo o seu fluxo de trabalho, desde a codificação até a entrega em produção (MATOS, 2023).

As vantagens para o uso do GitLab, que contribuiu para ser escolhido para este trabalho são em permitir a instalação em servidores locais e possuir uma ferramenta

interna de Integração e Entrega contínua, o GitLab CI/CD.

O GitLab CI/CD é uma ferramenta interna do GitLab, em que é permitido descrever todos os passos de integração e implantação contínua em um arquivo, o *.gitlab-ci.yaml*, armazenado no próprio repositório, sendo possível implementar todo o fluxo de trabalho, desde a codificação até a implantação em produção, em um único lugar. Com o GitLab CI/CD, pode-se automatizar tarefas de testes, construção, *deploy* entre outras possibilidades, que torna o processo de desenvolvimento rápido, eficiente e confiável. Além disso, pode-se visualizar o histórico e o progresso de todas as suas entregas diretamente na interface do GitLab (MATOS, 2023).

3 DESENVOLVIMENTO

Neste capítulo é apresentando o desenvolvimento do trabalho, descrevendo a forma em que a proposta foi elaborada e construída e o estudo de caso. A infraestrutura utilizada para o desenvolvimento e testes da proposta foi um servidor Linux Ubuntu Server 20.04 disponível na CMP para testes e ambientes de trabalho com sistema operacionais Windows 11 e Ubuntu 22.04.

3.1 Estudo de caso

Como estudo caso da proposta, para sua validação, será usado uma aplicação em Wordpress, que é um Sistema de Gestão de Conteúdos (*Content Management System - CMS*) *open source*, voltado, em sua grande maioria, para a criação de *sites* e *blogs*. É uma das ferramentas mais utilizadas para conteúdo na internet e bastante disseminada, pois tem código aberto, é fácil de utilizar e possui versatilidade.

O Wordpress é baseado na linguagem de programação PHP (*PHP: Hypertext Preprocessor*), uma linguagem de programação interpretada, sendo uma das primeiras em que era possível inserção em documentos HTML (*HyperText Markup Language*). Seu código é interpretado no lado do servidor, que gera a página web a ser vizualizada no lado do Cliente.

Como banco de dados, o Wordpress utiliza o MySQL, um sistema de gerencimento de banco de dados (SGBD) que utiliza a linguagem SQL (Structured Query Language) como interface. O MySQL possui fácil integração com PHP e é um dos sistemas de gerenciamento de bancos de dados mais populares atualmente, sendo utilizado por grandes empresas de tecnologia.

3.2 Arquitetura da solução

A proposta do ambiente de desenvolvimento foi construída em um servidor Linux com o Docker instalado, onde foi instalado um serviço de banco de dados Mysql, para uso das aplicações, um repositório de códigos e ferramenta de versionamento Gitlab, o Gitlab-runner, para a execução das pipelines da entrega contínua e um *registry* local, para o armazenamento de imagens Docker, além do serviço do Wordpress - que é nossa aplicação de estudo de caso. Todos esses serviços foram executados por meio de contêineres Docker, promovendo a infraestrutura necessária para o desenvolvimento de uma aplicação utilizando Wordpress, como o desenvolvimento de outros possíveis softwares.

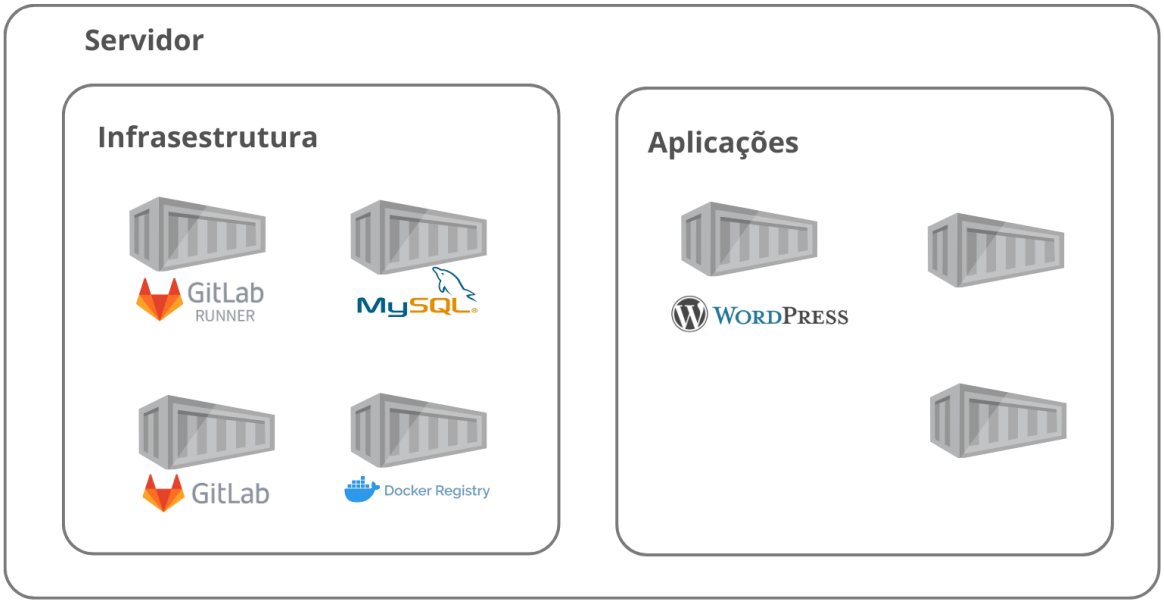

Figura 2 – Infraestrutura do ambiente proposta

Fonte: Elaborada pela autor

De forma geral, este trabalho propõe a construção de um ambiente de desenvolvimento e além da montagem do ambiente, a definição de um fluxo para integração contínua (CI) no desenvolvimento de *softwares*. Na Figura 3 podemos visualizar o fluxo, que parte do desenvolvedor quando realiza um *commit* no projeto da aplicação, o enviando ao repositório do projeto no Gitlab. No Gitlab, por meio do Gitlab CI/CD é disparado o *pipeline* que é executado pelo Gitlab-runner para realizar a integração contínua. Por sua vez, o Gitlab-runner constrói imagens Docker, as armazenam no *registry* e em um estágio subsequente as buscam para realizar o *deploy* da aplicação (Wordpress, que é nosso estudo de caso), a tornando disponível para acesso por usuários, desenvolvedores e analistas de qualidade. A aplicação em funcionamento realiza consultas ao banco de dados MySQL presente no a infraestrutura do ambiente. Dessa forma, os testes, construção de imagens e deploys são automatizados no ambiente de desenvolvimento.

Nas próximas seções, é mostrado como cada elemento da infraestrutura foi instalado e configurado, assim como o *pipeline* para a integração contínua foi construído.

3.3 Docker

Inicialmente, foi realizada a instalação do Docker no servidor, que será o gerenciador de contêineres, permitindo a criação e execução dos mesmos. Sua instalação no servidor foi realizada a partir de sua documentação oficial, de acordo as especificações necessárias para o sistema operacional do host.

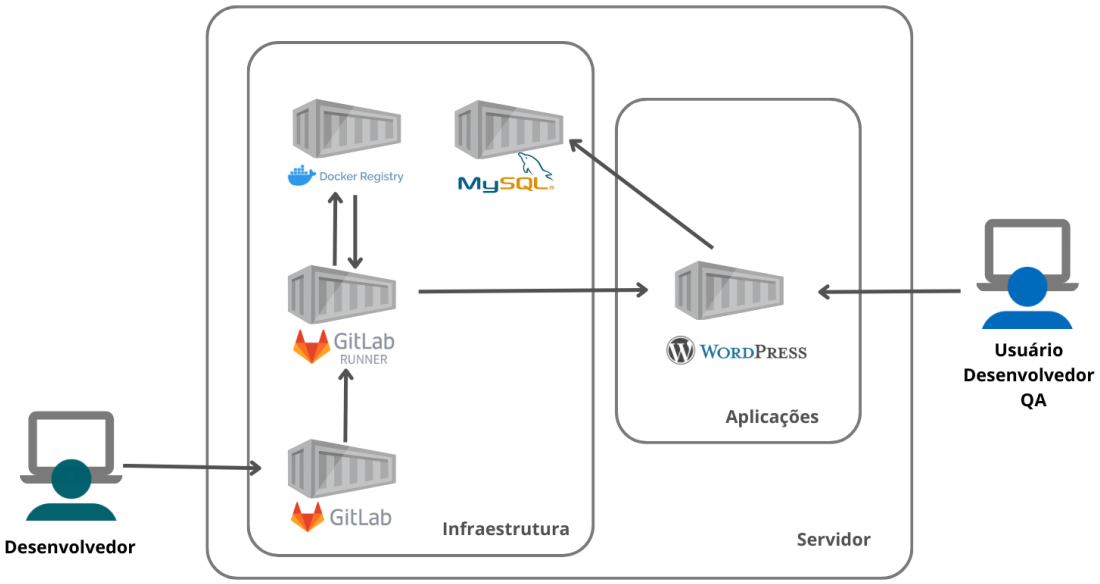

Figura 3 – Fluxo da proposta

Fonte: Elaborada pela autor

Para a criação e execução dos contêineres foi utilizado a ferramenta *docker-compose*, em que é possível definir características dos serviços, que serão executados em contêineres, como imagem, portas, dependências, local dos volumes, rede entre outras configurações, em um arquivo YAML. No trabalho, todos os serviços da infraestrutura (Gitlab, Giltab-runner, Registry e Mysql) foram definidos em um único arquivo *docker-compose.yaml*.

3.4 Gitlab

O Gitlab foi escolhido para ser o repositório de códigos da proposta do ambiente de desenvolvimento. O Gitlab é uma ferramenta *Open Source*, que pode ser instalada em servidores locais, é usada por diversas empresas de desenvolvimento de *software* atualmente. O Gitlab possui uma ferramenta interna para integração e entrega contínua, o Gitlab CI/CD, que será utilizado para a proposta de integração contínua deste trabalho.

O contêiner que executará o Gitlab foi criado a partir da uma imagem oficial, definido no arquivo *docker-compose.yaml*. Após inicializado e em execução, por meio de sua interface web foi definida a senha do usuário administrador, feitas outras configurações e criado o projeto da aplicação Wordpress que será o estudo de caso. Na Figura 4 vemos o Gitlab em execução e o projeto da aplicação criado, já estando disponível para acesso e interação por parte dos desenvolvedores.

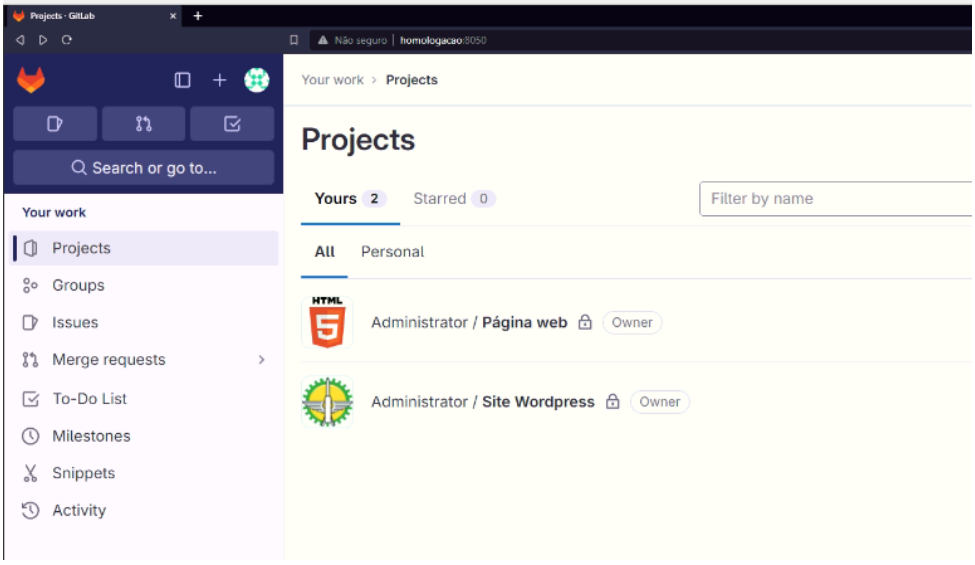

Figura 4 – Gitlab em execução

Fonte: Elaborada pela autor.

3.5 Gitlab-Runner

O Gitlab-runner é a ferramenta que é utilizada para a execução dos *jobs* definidos no *pipeline* no Gitlab CI/CD que realizará a integração contínua. A sua instalação pode ser feita diretamente em um servidor ou por meio de *containeres* Docker, esta segunda opção foi a adotada neste trabalho com a criação de um arquivo Dockerfile a partir da sua imagem Docker oficial.

No Dockerfile é definido também a instalação do Docker, que servirá para construir imagens, enviá-las ao *registry* e executar os *containeres* no ato do *deploy*. Vale destacar que o Docker que é executado dentro do container tem conexão e tem dependência com o docker instalado no host em que o container está sendo executado. Para execução do Gitlab-runner foi ultilizado o *docker-compose* que constrói a imagem, a partir do Dockerfile, e executa o container com base na imagem construída.

3.5.1 Registro

Com o Gitlab-runner em execução, é necessário criar e registrar um (ou mais) *runners* no Gitlab CI/CD, para que o pipeline definido no através do arquivo *.gitlab-ci.yaml Gitlab* seja executado. O registro é feito acessando o container, em modo texto, informando um nome, o endereço do repositório Gitlab, um *token* fornecido na área de configurações do Gitlab CI/CD e uma etiqueta(*tag*), que servirá para identificar o *runner* em que um determinado job será executado. Além disso, também é especificado o *excutor*, que determina o ambiente em que os *jobs* são executados, que pode ser *ssh*, *shell*, Docker, Virtualbox entre outros. Para este trabalho foi escolhido o ambiente *shell*, que permitirá

executar comandos Linux. Após o registro ser feito, é possível visualizar o *runner* na área de configuração do Gitlab CI/CD do repositório como apresentado na Figura 5.

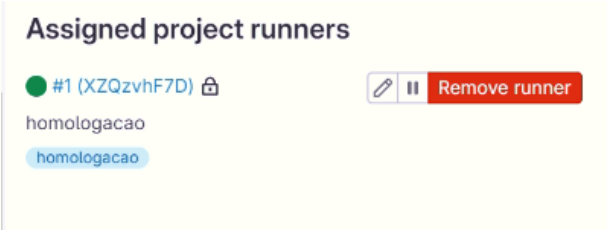

Figura 5 – Runner ativo no Gitlab CI/CD.

Fonte: Elaborada pela autor.

3.5.2 Par de chaves ssh

Uma solução para que o pipeline realize o *deploy* é copiar arquivos da aplicação para o servidor em que a aplicação será executada, fazer *pull* da imagem Docker da referida aplicação construída anteriormente e criar um contêiner que a portará, o executando a partir da imagem. Para realizar a cópia e poder executar comandos comandos no servidor em que estará a aplicação é possível fazer utilizando o protocolo *ssh*, que permite acesso remoto de um servidor a outro.

No caso da pipeline, para o *runner* acessar o servidor de forma automática, sem a necessidade de inserção de credenciais para o acesso é necessário configurar um par de chaves ssh entre o contêiner do Gitlab-runner, em que o *runner* opera, e o servidor onde será executado a aplicação. Em nosso caso, o servidor em que a aplicação será executada é o próprio *host* onde estão sendo executados os contêineres Docker da infraestrutura proposta. Criados o par de chaves ssh, a pipeline consegue acessar remotamente de forma direta o servidor para realizar o deploy da aplicação.

3.6 Registry

Para armazenamento das imagens das aplicações construídas é necessário um repositório de imagens Docker, um *registry* Docker. O Docker possui um *registry* público, o Docker Hub - de onde puxamos as imagens para os nossos containers da infraestrura, por exemplo - disponível, porém a opção para ter imagens privadas lá não está inclusa em seu plano gratuito. Ter um *registry* local é uma boa alternativa para armazenar imagens privadas de forma gratuita, proporcionando maior segurança.

Neste trabalho, o *registry* foi instalado via contêiner Docker no próprio *host* em que os demais serviços são executados, mas poderia ser instalado em um outro servidor específico para esse serviço também. Para a criação e execução do contêiner do *registry*

foi utilizado sua imagem oficial e mantido sua porta padrão 5000, definidos no arquivo *docker-compose.yaml*. Após inicializado, já está disponível para *pulls* e *pushs* de imagens.

3.7 Banco de Dados MySQL

O banco de dados em um ambiente de desenvolvimento é um elemento importante e crucial para armazenamento de dados das aplicações. O sistema de gerenciamento de banco de dados escolhido para este trabalho foi o MySQL, seu serviço foi criado e executado a partir da sua imagem Docker oficial, definida no arquivo *docker-compose.yaml* junto com as configurações de nome de usuário, senha e porta (neste caso a sua porta padrão 3306), além da definição de seu volume. Com o contêiner inicializado é possível conectar no banco de dados como na Figura 6.

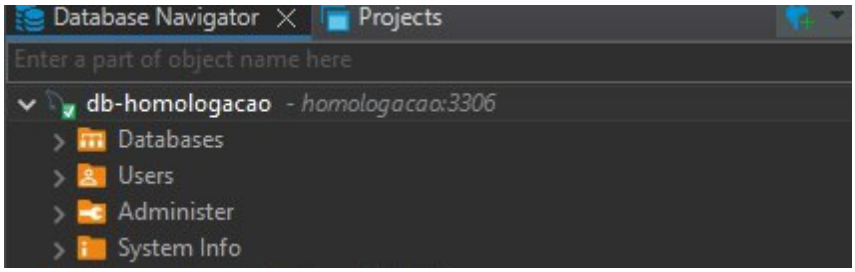

Figura 6 – Banco Mysql acessível.

Fonte: Elaborada pela autor.

3.8 Aplicação Wordpress

O estudo de caso para a proposta apresentada é uma aplicação Wordpress, desenvolvida na linguagem PHP que utiliza banco de dados MySQL. A aplicação será executada por meio de um contêiner Docker, desta vez utilizando um Dockerfile, com a imagem base oficial do Wordpress. Usar o Dockerfile permite a personalização do Wordpress, uma vez que embarcará na imagem construída os códigos desenvolvidos pelos programadores da instituição.

O Dockerfile da imagem é inserido no diretório do projeto da nossa aplicação Wordpress, nele é feita a cópia dos arquivos do projeto da aplicação para a imagem, dessa forma, o código desenvolvido estará na imagem que será construída e quando o contêiner com o serviço do Wordpress for criado e executado estará disponível a versão desenvolvida.

Para execução do contêiner com o serviço é usado o *docker-compose*, a partir da imagem construída com Dockerfile. No arquivo *docker-compose.yaml*, também inserido do diretório do projeto, é definindo a porta em que o serviço estará disponível, neste caso a

8080, e especificado, através de variáveis de ambiente, as credenciais (endereço, usuário, senha e nome do banco de dados) para acesso ao banco de dados do ambiente.

A construção da imagem a partir das configurações definidas no Dockerfile e execução do contêiner por meio do *docker-compose*, deverá acontecer de forma automatizada por meio do *pipeline* que implementará a integração contínua da aplicação. A Figura 7 mostra a aplicação Wordpress disponível para acesso na porta 8080 após construção da imagem e inicialização do contêiner com o *docker-compose*.

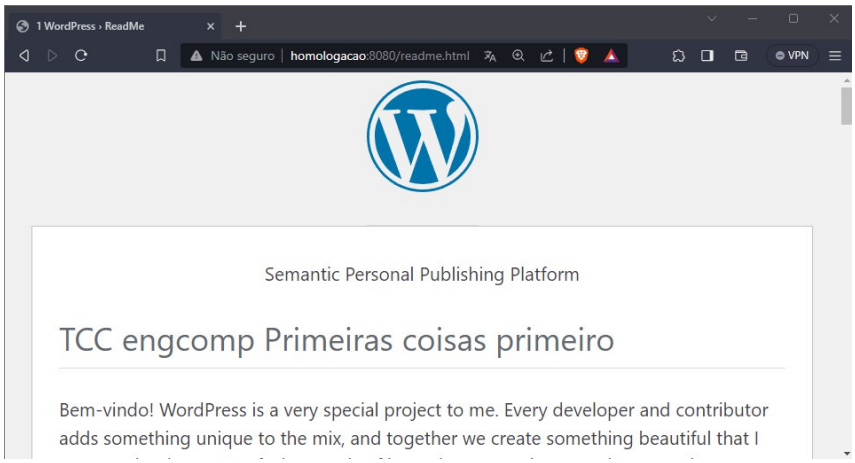

Figura 7 – Wordpress em execução

Fonte: Elaborada pela autor

3.9 Pipeline para a integração contínua

Após a montagem e definição da infraestrutura do ambiente e aplicação já configurada, é criado o *pipeline* que implementará a integração contínua da aplicação, automatizando o estágios de teste, construção da imagem e *deploy* da aplicação quando o desenvolvedor subir seu código para o repositório no Gitlab por meio de um *git push*.

O *pipeline* é definido no arquivo oculto *.gitlab-ci* inserido também no diretório do projeto. Este arquivo presente no repositório é identificado pelo Gitlab CI/CD e através do *gitlab-runner* executará os *jobs* definidos.

O *pipeline* criado é formado por estágios e em cada estágio são definidos *jobs*, que são tarefas a serem executadas. Na proposta do ambiente, a pipeline conta com três estágios: teste, *build* e *deploy*, como visto na Figura 8, e os respectivos *jobs teste-job*, *build-job* e *deploy-job*. No estágio de testes pode ser realizados testes unitários, de integração, entre outros, conforme o interesse do desenvolvedor, no estágio de *build* é realizado a construção de imagens da aplicação e no estágio de *deploy* é realizado o *deploy* da aplicação para sua disponibilização.

```
stages:
  - teste
  - build
  - deploy
```

Figura 8 – Estágios do pipeline

Fonte: Elaborada pela autor

No estágio de *build*, o *job build-job* realiza a construção da imagem com o código da aplicação Wordpress que está no repositório a partir do Dockerfile, e em seguida, envia a imagem para armazenamento no *registry* por meio do comando *docker push*, como mostrado na Figura 9.

```
build-job:
  stage: build
  script:
    - echo "Fazendo build da imagem..."
    - docker build -f Dockerfile -t homologacao:5000/site-wordpress .
    - docker push homologacao:5000/site-wordpress
    - echo "Build da imagem realizado!"
  tags:
    - homologacao
```

Figura 9 – Job de build

Fonte: Elaborada pela autor

No estágio de *deploy*, o *job deploy-job* realiza o deploy da aplicação no servidor em que a aplicação será executada, no caso deste trabalho, o *deploy* é feito no mesmo servidor. O acesso ao servidor para realizar o *deploy* é feito pelo protocolo *ssh*, que permite acesso remoto. O *job* inicia fazendo o *pull* da imagem da aplicação armazenada no *registry* que foi construída e armazenada lá anteriormente, em seguida copia os arquivos do projeto no repositório para o servidor de destino e então executa o *docker-compose* que irá parar a versão antiga da aplicação e iniciar a nova a partir da imagem construída, tornando a nova versão da aplicação Wordpress disponível para acesso. As instruções do *job* são mostradas na Figura 10.

O *pipeline* só será finalizado se todos os seus estágios foram concluídos, caso algum *job* falhe e apresente erro, o *pipeline* inteiro falha. Caso sejam executados com sucesso todos os *jobs*, o *pipeline* é concluído.

```
deploy-job:
  stage: deploy
  script:
    - echo "Iniciando deploy da aplicação..."
    - ssh gitlab-runner@homologacao 'docker pull homologacao:5000/site-wordpress' #pull da imagem do re
    - ssh gitlab-runner@homologacao 'rm -rf *'
    - scp -r /home/gitlab-runner/builds/XZQzvhF7/0/root/site-wordpress/* gitlab-runner@homologacao:~/
    - ssh gitlab-runner@homologacao 'docker-compose down && docker-compose up -d'
    - echo "Deploy realizado com sucesso! Acesse homologacao:8080 para acesso ao site"
  tags:
    - homologacao
```

Figura 10 – Job de deploy

Fonte: Elaborada pela autor

4 RESULTADOS

Nesse capitulo são apresentados os resultados da construção do ambiente e testes realizados para validar a proposta do ambiente de desenvolvimento e a integração contínua.

4.1 Validação da proposta

Após a montagem do ambiente e instalação de cada ferramenta por meio de contêineres Docker o ambiente se apresentou funcionando e estável, com seus serviços em execução e acessíveis como visto na Figura 11.

```
COMMAND                  CREATED       STATUS                 PORTS                                                  NAMES
"docker-entrypoint.s…"   2 weeks ago   Up 2 weeks             0.0.0.0:8080->80/tcp, :::8080->80/tcp                  wordpress
"/assets/wrapper"        2 weeks ago   Up 2 weeks (healthy)   22/tcp, 443/tcp, 0.0.0.0:8050->80/tcp, :::8050->80/tcp gitlab
"docker-entrypoint.s…"   2 weeks ago   Up 2 weeks             0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql
"/usr/bin/dumb-init …"   2 weeks ago   Up 2 weeks             0.0.0.0:8093->8093/tcp, :::8093->8093/tcp              gitlab-runner
"/entrypoint.sh /etc…"   2 weeks ago   Up 2 weeks             0.0.0.0:5000->5000/tcp, :::5000->5000/tcp              registry
```

Figura 11 – Containers Docker em execução

Fonte: Elaborada pela autor

Para a validação da proposta, será simulado uma alteração feita no código do projeto da aplicação Wordpress, que quando enviado ao repositório *git* por um *git push*, disparará o *pipeline* que construirá a imagem Docker da aplicação com a alteração e fará o *deploy* disponibilizando a aplicação ao usuário.

O arquivo a ser alterado será o arquivo ***readme.html*** presente na raiz do diretório do projeto, disponível no endereço *http://homologacao:8080/readme.html*, como mostrado na Figura 12.

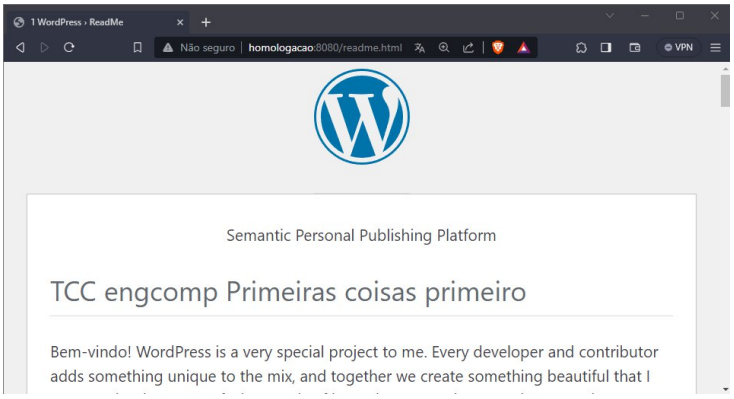

Figura 12 – Página readme.html

Fonte: Elaborada pela autor

Iniciando o teste, foi feito uma alteração no código no arquivo da página *readme.html*, mostrada na Figura 13. Com a alteração salva, foi feito o *commit* e o *git push* para o repositório.

```
</h1>
<p style="text-align: center">Semantic Personal Publishing Platform</p>

<h2> Alteração com deploy feito automaizado.  TCC engcomp Primeiras coisas primeiro
   Desenvolvedor > Gitlab > Gitlab-runner > Registry > Wordpress
</h2>
<p>Bem-vindo!
```

Figura 13 – Trecho do código alterado

Fonte: Elaborada pela autor

Após o *commit* no repositório, o *pipeline* foi disparado e executou os *jobs* dos estágios de teste, *build* e *deploy*, todos concluídos com sucesso, como mostrado a Figura 14. Na Figura 15 podemos ver os *jobs* de cada estágio concluídos.

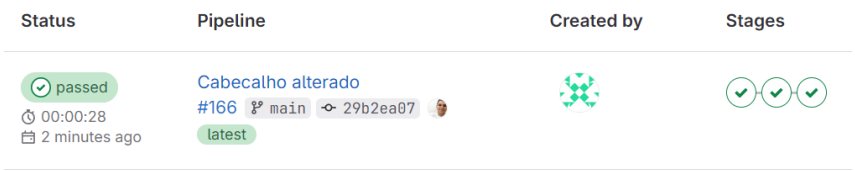

Figura 14 – Pipeline concluído

Fonte: Elaborada pela autor

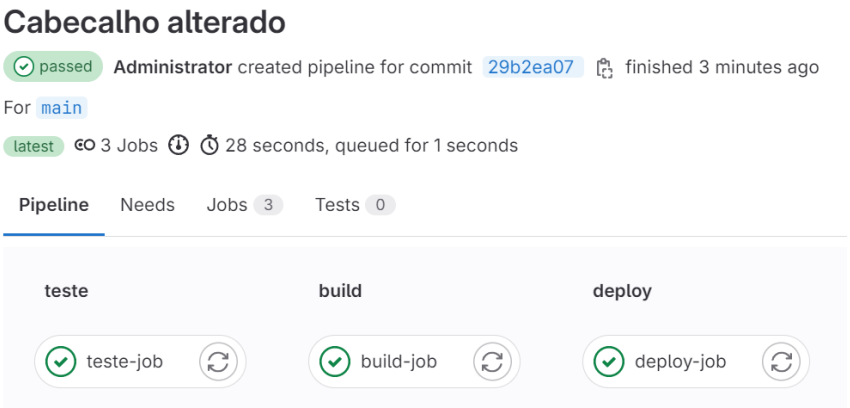

Figura 15 – Jobs concluídos

Fonte: Elaborada pela autor

Após a execução do *pipeline*, ao acessar ao endereço *http://homologacao:8080/readme.html* da página que teve seu código alterado, é possível notar que a alteração foi feita na aplicação e já está acessível, como podemos ver na Figura 16. Caso haja falha em algum *job* ou estágio, o *pipeline* falha por completo, não realizando a integração e a entrega contínua.

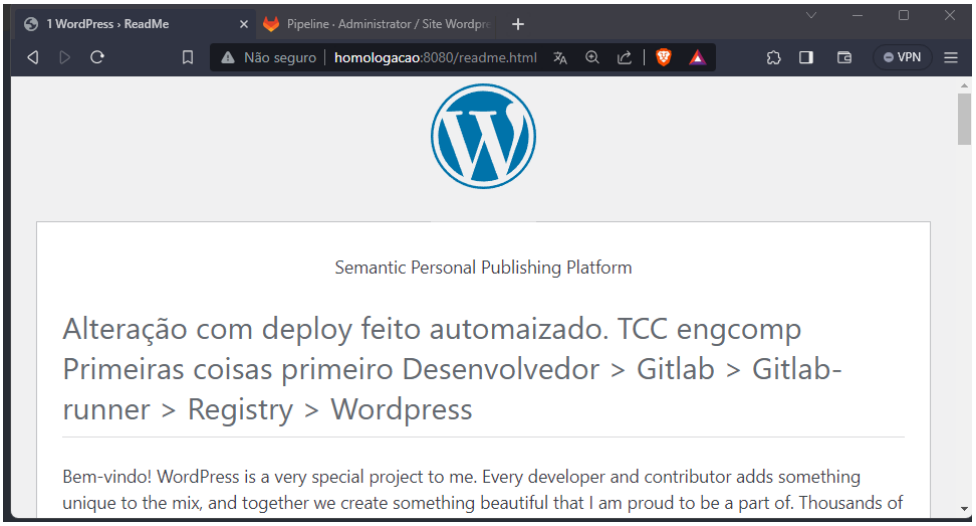

Figura 16 – Página com código alterado

Fonte: Elaborada pela autor

O correto funcionamento do *pipeline*, executando os *jobs* definidos sem erros ou falhas e aplicando a alteração no Wordpress de forma automatizada, valida a proposta apresentada e toda a infraestrutura do ambiente de desenvolvimento criada. Cada elemento incluindo o Gitlab, Gitlab-runner, *registry* e o Banco de dados MySQL, desempenham um papel essencial no fluxo para que a integração contínua seja realizada e as novas versões da a aplicação sejam disponibilizadas.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a elaboração de uma proposta de um ambiente destinado ao desenvolvimento de *software* com foco na integração contínua, apresentando um estudo de caso, como sugestão a instituições públicas que desejam e/ou estão desenvolvendo *softwares*, concentrando-se em um ambiente que atendas às demandas específicas da instituição, garantindo um fluxo contínuo de integração.

Analisando os resultados obtidos na proposta, nota-se o funcionamento correto de toda a infraestrutura do ambiente de desenvolvimento e em todos os seus componentes, como também no fluxo de desenvolvimento com tarefas automatizadas por meio do *pipeline*. O estudo de caso realizado através de uma aplicação Wordpress mostrou-se satisfatório, atendendo as expectativas da proposta da criação do ambiente de desenvolvimento com a integração contínua.

5.1 Trabalhos futuros

Os testes, que fazem parte do fluxo de desenvolvimento de *software*, nesse trabalho foram apenas ilustrados no *pipeline* sem que fossem realizados de fatos, desta maneira, um trabalho futuro a ser desenvolvido é a criação e aplicação de testes, conforme a necessidade dos desenvolvedores, a serem especificados no estágio de testes do *pipeline*.

Um próximo passo a ser dado é realizar testes a partir de um projeto real que está sendo desenvolvido, como sugestão o Portal da Câmara Municipal de Parnamirim. Sendo necessário a sua containerização, seguindo o processo realizado neste trabalho de criação de Dockerfile para a construção de imagens específicas do portal.

Um ponto importante que não foi abordado neste trabalho é a criação de uma rotina de *backups* do ambiente, desde os volumes e configurações dos contêineres até *snapshots* do servidor que está sendo executado o ambiente. Isso irá garantir que se por algum motivo, o ambiente ou algum elemento seja perdido, haverá cópia, tornando-se possível a reconstrução sem ou poucos prejuízos.

A implementação da proposta apresentada neste trabalho em uma instituição para uso como ambiente de desenvolvimento de *software* com integração contínua, é um trabalho a ser executado, que tornará efetiva a utilização da solução apresentada, no seu desenvolvimento de *software*.

Por fim, com os quatro pontos anteriores efetuados, é interessante um trabalho de elaboração do ambiente de produção para hospedagem dos *softwares* desenvolvidos, que

estarão em pleno funcionamento e acessíveis aos usuários finais, também sendo prezado pela entrega contínua da aplicação. Para o ambiente de produção, é necessário atenção maior na segurança da informação, com processos mais rígidos de proteção ao ambiente.

REFERÊNCIAS

TEKZOOOM. O QUE É AMBIENTE DE HOMOLOGAÇÃO, DESENVOLVIMENTO E PRODUÇÃO. YouTube, 2022. Disponível em: https://www.youtube.com/watch?v=gKl95IfxGXk. Acesso em 30 de setembro de 2023.

FERNANDO, J. DESCOMPLICANDO O DOCKER, 2023. Disponível em: https://livro.descomplicandodocker.com.br/. Acesso em: 04 de dezembro de 2023.

DANTAS, D. L. G. APRIMORAMENTO DO PROCESSO DE DEVOPS COM INTEGRACÃO CONTÍNUA DA JUSTIÇA FEDERAL DO RIO GRANDE DO NORTE. Natal, 2021.

DSP (AMBIENTES PARA DESENVOLVIMENTO DE SOFTWARE) // DICIONÁRIO DO PROGRAMADOR. Youtube, 2021. Disponível em: https://www.youtube.com/watch?v=uBqCqU5gKbM. Acesso em 30 de setembro de 2023.

EXPLICANDO SOBRE AMBIENTES DE SISTEMAS - DESENVOLVIMENTO/HOMOLOGAÇÃO/PRODUÇÃO. YouTube, 2022. Disponívem em: https://www.youtube.com/watch?v=ahcGQ2kdw5g. Acesso em 30 de setembro de 2023.

QUANTOS AMBIENTES DE TESTE PRECISO? DESCUBRA. YouTube, 20220. Disponível em: https://www.youtube.com/watch?v=50iG1K7Gwfo. Acesso em 30 de setembro de 2023.

INSTALL DOCKER ENGINE ON UBUNTU. Docker Docs. Disponível em: https://docs.docker.com/engine/install/ubuntu. Acesso em 11 de setembro de 2023.

GITLAB DOCKER IMAGES. GitLab Docs. Disponível em: https://docs.gitlab.com/ee/install/docker.html. Acesso em 11 de setembro de 2023.

GITLAB RUNNER. Gitlab Docs. Disponível em: https://docs.gitlab.com/runner. Acesso em 15 de setembro de 2023.

REGISTERING RUNNERS. GitLab Docs. Disponível em: https://docs.gitlab.com/runner/register. Acesso em 15 de setembro de 2023.

TUTORIAL: CREATE, REGISTER, AND RUN YOUR OWN PROJECT RUNNER. GitLab Docs. Disponível em: https://docs.gitlab.com/ee/tutorials/create-register-first-runner. Acesso em 15 de setembro de 2023.

INSTALL GITLAB RUNNER MANUALLY ON GNU/LINUX. GitLab Docs. Dispinível em: https://docs.gitlab.com/runner/install/linux-manually.html. Acesso em 15 de setembro de 2023.

COMO CONFIGURAR A AUTENTICAÇÃO BASEADA EM CHAVES SSH EM UM SERVIDOR LINUX. DigitalOcean. Disponível em: https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server-pt. Acessem em 09 de novembro de 2023.

GERANDO E USANDO CHAVES SSH PARA AUTENTICAÇÃO DE HOST REMOTO. IBM Cloud. Disponível em: https://cloud.ibm.com/docs/ssh-keys?topic=ssh-keys-generating-and-using-ssh-keys-for-remote-host-authenticationlocale=pt-BR. Acesso em 09 de novembro de 2023.

REGISTRY. Docker docs. Disponível em: https://docs.docker.com/registry/deploying. Acesso em 09 de novembro de 2023.

WORDPRESS. Docker hub. Disponível em: https://hub.docker.com/-/wordpress. Acesso em 26 de outubro de 2023.

GITLAB/GITLAB-RUNNER. Docker hub. Disponível em: https://hub.docker.com/r/gitlab/gitlab-runner/dockerfile. Acesso em 26 de outubro de 2023.

CRIANDO O SEU DOCKER REGISTRY | DESCOMPLICANDO O DOCKER V1 - PARTE 13. YouTube, 2017. Disponível em: https://www.youtube.com/watch?v=ndoF1VUFnWY. Acesso em 07 de novembro de 2023.

ROSSETO, A. DOCKER: COMO CRIAR E EXECUTAR UM REGISTRY PRIVADO. Medium, 2020. Disponível em: https://alexandrerosseto.medium.com/docker-como-criar-e-executar-um-registry-privado-e595ae0b1d4f. Acesso em 07 de novembro de 2023.

DOCKER REPOSITORY SERVER GAVE HTTP RESPONSE TO HTTPS CLIENT. Stack Overflow. Disponível em: https://stackoverflow.com/questions/49674004/docker-repository-server-gave-http-response-to-https-client/54190375. Acesso em 07 de novembro de 2023.

O QUE É UM AMBIENTE DE DESENVOLVIMENTO? COMO ELE DIFERE DE UM AMBIENTE DE DESENVOLVIMENTO INTEGRADO (IDE)?. Hostinger Tutoriais. Disponível em: https://www.hostinger.com.br/tutoriais/ambiente-de-desenvolvimento. Acesso em 14 de novembro de 2023.

YAML. Wikipédia, A enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/YAML. Acesso em 27 de novembro de 2023.

O QUE SIGNIFICA INTEGRAÇÃO CONTÍNUA?. AWS. Disponível em: https://aws.amazon.com/pt/devops/continuous-integration. Acesso em 01 de dezembro de 2023.

O QUE É UM CONTAINER LINUX?. RedHat, 2022. Disponível em: https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container. Acesso em 04 de dezembro de 2023.

DOCKER OVERVIEW. Docker docs. Disponível em: https://docs.docker.com/get-started/overview/ Acesso em 04 de dezembro de 2023.

GUEDES, M. NO FINAL DAS CONTAS: O QUE É O DOCKER E COMO ELE FUNCIONA?. TreinaWeb, 2018. Disponível em: https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona. Acesso em 04 de dezembro de 2023.

O QUE É O DOCKER?. IBM Cloud. Disponível em: https://www.ibm.com/br-pt/topics/docker Acesso em 04 de dezembro de 2023.

O QUE É O GIT. Atlassian. Disponível em: https://www.atlassian.com/br/git/tutorials/what-is-git Acesso em 04 de dezembro de 2023.

WORDPRESS. Wikipédia, A enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/WordPress Acesso em 04 de dezembro de 2023.

PHP. Wikipédia, A enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/PHP Acesso em 04 de dezembro de 2023.

MYSQL. Wikipédia, A enciclopédia livre. https://pt.wikipedia.org/wiki/MySQL Acesso em 04 de dezembro de 2023.

MOHAMMAD, S. M. CONTINUOUS INTEGRATION AND AUTOMATION. International Journal of Creative Research Thoughts (IJCRT), ISSN, 2016.

MATOS, L. ENTENDA O GITLAB CI/CD: UMA INTRODUÇÃO PARA INICIANTES. Medium, 2023. Disponível em: https://medium.com/@luccapessoa/entenda-o-gitlab-ci-cd-5c738b6049ef. Acesso em 05 de dezembro de 2023.

GUEDES, M. AFINAL, O QUE É DEVOPS?. TreinaWeb, 2018. Disponível em: https://www.treinaweb.com.br/blog/afinal-o-que-e-devops. Acesso em 05 de dezembro de 2023.

O QUE É O GITOPS?. Red Hat, 2023. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-gitops. Acesso em 04 de dezembro de 2023.

LIMA, P. H. O. ESTUDO COMPARATIVO SOBRE AS FUNCIONALIDADES DE INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA DAS FERRAMENTAS JENKINS E GITLAB. Trabalho de Conclução de Curso -Centro Universitário Christus - Unichristus, Curso de Sistemas de Informação, Fortaleza, 2021.

SILVA. R. R. V. GITOPS: UMA NOVA PROPOSTA PARA A INFRAESTRUTURA. Trabalho de Conclusão de Curso - Departamento de Informática e Estatistica - Universidade Federal de Santa Catarina, Curso de Sistemas de Informação, Florianópolis, 2020.

BUFFA, L. H. TORNANDO ACESSÍVEL A CULTURA DEVOPS A PEQUENAS EMPRESAS E STARTUPS. Trabalho de Conclusão de Curso - Escola Politécnica - Pontifícia Universidade Católica De Goiás, Graduação em Engenharia ee Computação, Goiânia, 2021.