Pontifícia Universidade Católica de Goiás

Campus Goiânia

Graduação em Engenharia da computação

Haryston Daniel Martins Campos

DevOps e Computação em Nuvem: Automação Inteligente de Pipelines CI/CD

Goiânia

2025

Haryston Daniel Martins Campos

DEVOPS E COMPUTAÇÃO EM NUVEM: AUTOMAÇÃO INTELIGENTE DE PIPELINES CI/CD

Trabalho de Conclusão de Curso apresentado à Escola politécnica e de Artes, da Pontíficia Universidade de Goiàs como parte dos requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Prof. Dr. Clarimar José Coelho

Goiânia

2025

Haryston Daniel Martins Campos

DEVOPS E COMPUTAÇÃO EM NUVEM: AUTOMAÇÃO INTELIGENTE DE PIPELINES CI/CD

Trabalho de Conclusão de Curso apresentado à Escola politécnica e de Artes, da Pontíficia Universidade de Goiàs como parte dos requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Aprovado em xx de Dezembro de 2025

BANCA EXAMINADORA

Prof. Dr. Clarimar José Coelho
Orientador/PUC Goiás

Professor
Prof. Mr. Daniel Correa Da Silva/PUC Goiás

Professor
Prof. Dr. Ernersto Fonseca Veiga/PUC Goiás

A minha família e amigos. Especialmente minha mãe Maria José e meu pai Ariston Pereira, minhas maiores inspirações.

AGRADECIMENTOS

Agradecer primeiramente a Deus pela oportunidade.

Agradeço a minha família e amigos por todo apoio durante esta jornada.

Ao ao meu orientador, Dr. Clarimar José Coelho pelo suporte.

Serei eternamente grato por todos os momentos que tivemos na construção desta tese de conclusão de curso.

Agradeço aos meus colegas de trabalho pelo auxilio e conselhos nesta jornada.

Gratidão a todos os funcionários desta universidade, pois de forma direta ou indireta foram muito importantes nesta jornada.

Não espere o futuro mudar tua vida, porque o futuro será a consequência do presente. (Racionais MC´s)

RESUMO

Este trabalho apresenta o desenvolvimento de uma solução prática de automação de pipelines de Integração Contínua e Entrega Contínua em ambiente de computação em nuvem, demonstrando como a adoção de práticas DevOps melhora a eficiência e a confiabilidade no processo de entrega de software. O estudo define claramente o problema, analisa conceitos fundamentais e aplica uma metodologia baseada em revisão bibliográfica, análise comparativa de ferramentas, implementação experimental e validação de resultados. A pesquisa utiliza recursos acessíveis, integra tecnologias amplamente adotadas no mercado e demonstra que processos automatizados reduzem falhas, padronizam ambientes e aumentam a velocidade de entrega. A implementação inclui a criação de um pipeline funcional, a configuração de uma infraestrutura em nuvem e a publicação automatizada de imagens em um repositório remoto. A análise das execuções confirma que a solução entregue é estável, segura e reprodutível. O trabalho também valida a proposta por meio da comparação com um caso real de uso em produção, fortalecendo a relevância prática da automação apresentada. Os resultados mostram ganhos concretos de desempenho, redução de erros e maior rastreabilidade das etapas de desenvolvimento.

Palavras chaves: DevOps; Integração Contínua; Entrega Contínua; CI/CD; Computação em Nuvem; Automação; Docker; GitHub Actions.

ABSTRACT

This work presents the development of a practical automation solution for Continuous Integration and Continuous Delivery pipelines in a cloud computing environment, demonstrating how the adoption of DevOps practices enhances efficiency and reliability in software delivery processes. The study defines the problem, analyzes essential concepts, and applies a methodology based on literature review, comparative analysis of tools, experimental implementation, and results validation. The research employs accessible resources, integrates widely adopted technologies, and shows that automated processes reduce failures, standardize environments, and accelerate delivery. The implementation includes the creation of a functional pipeline, the configuration of a cloud infrastructure, and the automated publication of container images in a remote repository. The analysis of pipeline executions confirms that the delivered solution is stable, secure, and reproducible. The work also validates the proposal through comparison with a real production case, reinforcing the practical relevance of the implemented automation. The results demonstrate concrete improvements in performance, error reduction, and traceability across development stages.
Keywords: DevOps; Continuous Integration; Continuous Delivery; CI/CD; Cloud Computing; Automation; Docker; GitHub Actions.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecnologia da Informação
CI	*Continuous Integration* (Integração Contínua)
CD	*Continuous Delivery* (Entrega Contínua)
NIST	*National Institute of Standards and Technology* (Instituto Nacional de Padrões e Tecnologia dos Estados Unidos)
LTS	*Long Term Support (Suporte de Longo Prazo)*
VPC	*Virtual Private Cloud (Nuvem Privada Virtual)*
CPU	*Central Processing Unit (Unidade Central de Processamento)*
SSD	*Solid State Drive* (Unidade de Estado Sólido)
GB	*Gigabyte (Unidade* de medida de armazenamento)
RAM	*Random Access Memory* (Memória de Acesso Aleatório)
SSH	*Secure Shell* (Protocolo Seguro de Comunicação)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HTTPS	*Hypertext Transfer Protocol Secure* (Protocolo de Transferência de Hipertexto Seguro)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
OCI	*Oracle Cloud Infrastucture* (Interface de Programação de Aplicações)
IaaS	*Infrastructure as a Service* (Infraestrutura como Serviço)
PaaS	*Platform as a Service* (Plataforma como Serviço)
SaaS	*Software as a Service* (Software como Serviço)
REST	*Representational State Transfer* (Transferência de Estado Representacional)

SUMÁRIO

1 INTRODUÇÃO

DevOps é um conjunto de práticas que integra o desenvolvimento de *software* (Dev) e operações de TI (Ops), com foco em automação, colaboração e entrega contínua de valor (Kim *et al.*, 2016; Bass; Weber; Zhu, 2015). Computação em nuvem é um modelo de fornecimento de recursos de TI (armazenamento, processamento, redes e aplicações) via internet sob demanda, geralmente em esquema de pagamento por uso, com escalabilidade e elasticidade (Mell; Grance, 2011; Armbrust *et al.*, 2010). A automação inteligente se refere ao uso de ferramentas de automação, muitas vezes associadas a inteligência artificial,*machine learning* ou orquestração avançada, para reduzir intervenção humana, aumentar eficiência e minimizar erros em processos (Jeston; Nelis, 2014; Gartner, Inc., 2023).

A transformação digital tem impulsionado uma demanda crescente por sistemas ágeis, escaláveis e resilientes. Nesse contexto, empresas e instituições precisam entregar software de forma mais rápida e segura, mantendo a qualidade e reduzindo falhas. A abordagem DevOps surge como resposta a esse desafio, ao promover integração contínua entre equipes de desenvolvimento e operações, aliada à automação de processos críticos (Kim *et al.*, 2016; Bass; Weber; Zhu, 2015). A computação em nuvem *(Cloud Computing)* consolida-se como a principal infraestrutura tecnológica para atender tais demandas, oferecendo elasticidade, escalabilidade sob demanda e redução de custos operacionais (Mell; Grance, 2011; Armbrust *et al.*, 2010). A convergência entre DevOps e nuvem amplia as possibilidades de automação, tornando os pipelines de Integração e Entrega Contínua mais inteligentes, eficientes e adaptáveis. Do ponto de vista social e econômico, pesquisas nessa área são relevantes porque contribuem para a redução do tempo de entrega de serviços digitais essenciais em setores como saúde, transporte, finanças e educação. Permitem que *startups* e pequenas empresas utilizem soluções escaláveis de baixo custo, aumentando competitividade e inovação. Promovem confiabilidade e segurança digital, aspectos críticos para a sociedade em um cenário de crescente dependência de sistemas online. Assim, investigar automação inteligente de pipelines CI/CD em nuvem não é apenas um tema de interesse tecnológico, mas também uma contribuição significativa para a sociedade, ao possibilitar serviços digitais mais ágeis, seguros e acessíveis.

O conceito de integração contínua (CI) remonta ao final da década de 1990, quando práticas de desenvolvimento ágil começaram a difundir-se em resposta às limitações do modelo tradicional em cascata. A proposta central era integrar mudanças de código de forma frequente, automatizando testes e garantindo maior confiabilidade no processo de desenvolvimento (Fowler; Foemmel, 2006). Na sequência, evoluiu-se para o conceito de entrega contínua (*Continuous Delivery*, CD), consolidado na obra de (Humble; Farley,

2010), que apresentou práticas e ferramentas para automatizar desde a construção do *software* até sua disponibilização em ambientes de produção. Esse marco deu origem ao movimento de padrões de automação em pipelines, que se tornaram essenciais para reduzir tempo de entrega e aumentar a qualidade. Paralelamente, a computação em nuvem consolidou-se como paradigma tecnológico a partir da década de 2000, com definições formais propostas pelo Instituto Nacional de Padrões e Tecnologia NIST (Mell; Grance, 2011) e análises de impacto apresentadas por (Armbrust *et al.*, 2010). O NIST é uma agência governamental dos Estados Unidos vinculada ao Departamento de Comércio, cuja missão é desenvolver padrões, métricas e diretrizes técnicas de referência para a indústria e a sociedade. No campo da tecnologia da informação, o NIST desempenha papel central na padronização e definição de conceitos-chave, oferecendo documentos de referência amplamente adotados por governos, empresas e instituições de pesquisa. No contexto da computação em nuvem, o NIST publicou em 2011 o documento *The NIST Definition of Cloud Computing* , elaborado por (Mell; Grance, 2011), que se tornou referência mundial. Esse relatório sistematizou as características essenciais da nuvem, definidas como: Autoatendimento sob demanda *(On-demand self-service)*, Acesso amplo à rede *(Broad network access)*, Agrupamento de recursos *(Resource pooling)*, Elasticidade rápida *(Rapid elasticity)*, Serviço mensurado *(Measured service)*. Além dessas características, o NIST também classificou os modelos de serviço em: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS). Descreveu os modelos de implantação (nuvem privada, pública, comunitária e híbrida). A contribuição do NIST foi fundamental para criar uma linguagem comum e padronizada que permitiu a expansão da nuvem em escala global, garantindo interoperabilidade, segurança e confiabilidade. Desde então, pesquisadores e profissionais têm utilizado essas definições como base para o desenvolvimento de arquiteturas e soluções em ambientes de nuvem. Assim, o trabalho de (Mell; Grance, 2011) não apenas estabeleceu um marco conceitual, mas também orienta até hoje práticas de DevOps e automação de pipelines CI/CD em ambientes de nuvem, sendo uma das referências mais relevantes para este estudo.

A nuvem trouxe elasticidade, escalabilidade e modelos de serviço (IaaS, PaaS e SaaS) que viabilizaram a implementação massiva de pipelines automatizados em escala global. O movimento DevOps, sistematizado a partir de 2009 e popularizado por obras como *The DevOps Handbook* (Kim *et al.*, 2016) e *DevOps: A Software Architect's Perspective* (Bass; Weber; Zhu, 2015), estabeleceu a integração entre desenvolvimento e operações, com foco em colaboração, automação e monitoramento contínuo. Essa abordagem permitiu alinhar práticas de CI/CD à infraestrutura de nuvem, ampliando eficiência e confiabilidade. Entre os trabalhos mais relevantes para esta pesquisa destacam-se: (Fowler; Foemmel, 2006), que estabeleceu fundamentos da Integração Contínua. (Humble; Farley, 2010), que sistematizaram a Entrega Contínua.(Armbrust *et al.*, 2010) e (Mell; Grance, 2011), que formalizaram o modelo de computação em nuvem. (Kim *et al.*, 2016) e (Bass; Weber;

Zhu, 2015), que consolidaram o movimento DevOps como prática organizacional. Apesar dos avanços, pesquisadores e profissionais ainda enfrentam dificuldades recorrentes. Complexidade na integração de ferramentas heterogêneas (CI/CD, monitoramento, orquestração de containers, etc.). Segurança em pipelines de nuvem, especialmente em ambientes multi-tentantes. Equilíbrio entre automação e governança, já que a rapidez de entrega deve ser acompanhada de conformidade e rastreabilidade. Escalabilidade inteligente, pois pipelines precisam adaptar-se a diferentes contextos de carga e disponibilidade de recursos. Esses desafios inspiram esse Trabalho de Conclusão de Curso, que busca contribuir para a discussão e desenvolvimento de estratégias de automação inteligente de pipelines CI/CD em ambientes de nuvem, alinhadas às práticas DevOps.

Neste trabalho, foi proposta e executada uma pesquisa aplicada para desenvolver e implementar uma solução prática de automação de pipelines de CI/CD em ambiente de nuvem, utilizando práticas de DevOps com o objetivo central de melhorar a eficiência e a confiabilidade do processo de desenvolvimento e entrega de *software*. A implementação seguiu uma abordagem metodológica que integrou pesquisa teórica e aplicação prática, permitindo não apenas o estudo dos conceitos fundamentais, mas também sua validação através de um caso real. Para atingir o objetivo geral, foram estabelecidos e cumpridos objetivos específicos inter-relacionados. Inicialmente, realizou-se um estudo aprofundado dos conceitos e práticas de DevOps, fundamentado em trabalhos seminais como (Kim *et al.*, 2016) e (Bass; Weber; Zhu, 2015), que forneceram a base teórica necessária para compreender os aspectos culturais, técnicos e organizacionais desta abordagem. Em seguida, procedeu-se à análise comparativa de ferramentas de automação de CI/CD, incluindo *Jenkins*, *GitLab CI* e *GitHub Actions*, avaliando critérios como facilidade de implementação, integração com ambientes de nuvem, custos e suporte à comunidade. A etapa de implementação contemplou a configuração de um ambiente em nuvem na *DigitalOcean*, selecionada por oferecer um equilíbrio entre custo e funcionalidades para o escopo deste trabalho, seguida do desenvolvimento de um pipeline completo de CI/CD para uma aplicação web de exemplo.

Este trabalho apresenta vantagens significativas em relação a outros já publicados na área, principalmente pela sua abordagem prática e acessível. Diferente de estudos que focam em implementações empresariais de grande porte, esta pesquisa mostra que é possível obter benefícios tangíveis da automação de CI/CD mesmo com recursos limitados, tornando-se particularmente relevante para pequenas e médias empresas brasileiras. Como principais conclusões, observou-se que a adoção de práticas DevOps contribui para fortalecer a colaboração entre equipes, padronizar processos e promover uma cultura de aprendizado contínuo. Assim, o trabalho reafirma a importância da automação e da integração entre desenvolvimento e operações como pilares para a construção de sistemas modernos, escaláveis e sustentáveis.

2 MATERIAIS E MÉTODOS

2.1 Definição do problema

O cerne da problemática abordada neste trabalho reside na persistência de processos manuais e fragmentados no desenvolvimento e entrega de software, que se manifesta através de desafios operacionais críticos identificados tanto na literatura especializada quanto na prática profissional contemporânea. Conforme demonstrado pelo relatório da *International Business Machines* (IBM) (IBM Security, 2025), as organizações que dependem de processos manuais e não adotam práticas modernas de automação enfrentam custos médios de violação de dados significativamente superiores e tempos de recuperação mais prolongados. O estudo revela que o custo médio global de uma violação atingiu USD 4,44 milhões em 2025, enquanto organizações que utilizam extensivamente ferramentas de Inteligência Artificial (IA) e automação em segurança conseguiram reduzir esse custo em aproximadamente USD 1,9 milhão, além de encurtar o ciclo de vida da violação em 108 dias. A falta de automação e governança adequada também está associada a violações envolvendo IA inabilitáveis, que acrescentam em média USD 200 mil ao custo total. Os métodos tradicionais de *deploy*, realizados por meio de configurações manuais em servidores físicos ou máquinas virtuais, apresentam uma série de limitações como tempo excessivo de execução, inconsistência entre ambientes, custos elevados, vulnerabilidades de segurança e períodos de inatividade, esses problemas são evidenciados pelo relatório *Accelerate: State of DevOps* 2024 (Forsgren *et al.*, 2024), que destacam a correlação direta entre automação e melhores resultados em desempenho organizacional.

2.2 Abordagem metodológica

Este trabalho adota a metodologia definida por (Pressman, 2016) para projetos de engenharia de software. A metodologia aplicada segue o modelo iterativo e incremental que permite o desenvolvimento evolutivo de soluções através de ciclos repetitivos de planejamento, implementação, teste e avaliação. Esta escolha metodológica justifica-se pela natureza complexa e dinâmica dos sistemas de automação de CI/CD, que exigem ajustes contínuos e refinamentos progressivos durante o processo de desenvolvimento.

O trabalho foi estruturado em quatro fases principais inter-relacionadas, cada uma incorporando os elementos essenciais do processo de engenharia de software: comunicação, planejamento, modelagem, construção e entrega. A primeira fase, de revisão bibliográfica sistemática, seguiu o padrão de processo de software adaptativo, onde a compreensão do problema e a coleta de requisitos foram realizadas através da análise abrangente de fontes

acadêmicas e técnicas. Esta etapa incluiu o estudo de artigos científicos indexados nas principais bases de dados, documentação oficial das ferramentas, relatórios de indústria reconhecidos e estudos de caso relevantes, permitindo o estabelecimento de uma base teórica sólida para o desenvolvimento da solução.

A segunda fase envolveu a análise comparativa de ferramentas disponíveis no mercado. Esta análise permitiu a seleção consciente do *stack* tecnológico mais adequado aos objetivos da pesquisa. Nesta fase revelou-se fundamental para embasar a seleção tecnológica do projeto, considerando que a escolha adequada de ferramentas constitui um dos pilares para o sucesso da implementação de práticas DevOps.

A terceira fase desenvolveu-se a implementação prática, incorporou os princípios do modelo de construção iterativa, onde a solução foi desenvolvida em ciclos incrementais com *feedback* contínuo. Esta etapa incluiu a configuração de ambientes seguindo as práticas de infraestrutura como código, desenvolvimento de *pipelines* de CI/CD aplicando os princípios de arquitetura de *software*, implementação de monitoramento baseado em métricas e execução de testes de validação sistemáticos. A abordagem iterativa permitiu a identificação e correção precoce de problemas, reduzindo riscos e garantindo a qualidade do produto final

A quarta e última fase dedicou-se à avaliação sistemática, aplicou os princípios de garantia de qualidade de software e validação. Esta etapa envolveu a coleta e análise quantitativa de métricas de desempenho, comparação com o estado anterior do processo de desenvolvimento, e documentação abrangente dos aprendizados obtidos. A avaliação seguiu os critérios de qualidade estabelecidos por (Pressman, 2016), incluindo confiabilidade, eficiência, usabilidade, manutenibilidade e portabilidade, garantindo assim uma validação robusta da solução implementada.

2.3 Ferramentas e tecnologias utilizadas

A seleção do *stack* tecnológico foi guiada por critérios rigorosos de relevância técnica, adoção no mercado, relação custo-benefício e adequação ao contexto de pesquisa acadêmica. O ecossistema implementado integrou múltiplas ferramentas especializadas. A maturidade tecnológica e a adoção pelo mercado emergiram como fatores críticos, uma vez que ferramentas consolidadas oferecem maior estabilidade e suporte comunitário. Paralelamente, a análise de custos mostrou-se imperativa no contexto acadêmico, onde restrições orçamentárias frequentemente determinam a viabilidade de implementação. Neste aspecto, constatou-se que soluções *open-source* ou com modelos *freemium* apresentam vantagens significativas para projetos de pesquisa.

Docker foi empregado para orquestração de containers, aproveitando sua tecnologia de virtualização leve que empacota aplicações e suas dependências em unidades isoladas. Esta escolha fundamenta-se na capacidade do Docker de garantir consistência absoluta entre

ambientes através de imagens imutáveis, seguindo as melhores práticas documentadas por (Turnbull, 2014). A implementação incluiu a criação de Dockerfile otimizado, configuração de redes entre containers e utilização do Docker Hub como repositório centralizado de imagens.

A arquitetura do Docker, ilustrada na Figura 1, demonstra claramente as camadas que compõem esse ecossistema, partindo da infraestrutura física ou virtual até as aplicações em execução nos containers.

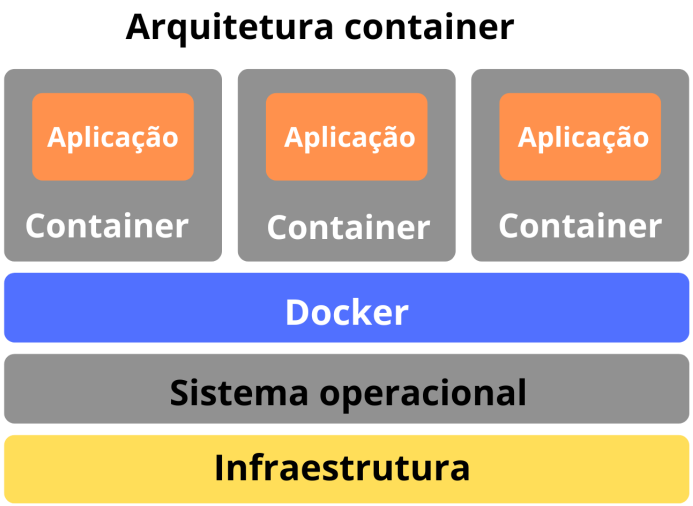

Figura 1 – Arquitetura de containerização

Na base da arquitetura encontra-se a infraestrutura, representando o hardware físico ou ambiente de nuvem que fornece os recursos computacionais necessários. Sobre esta camada reside o sistema operacional linux, que no contexto deste trabalho foi o Ubuntu 24.04 LTS, responsável por gerenciar os recursos da infraestrutura e fornecer os serviços básicos para a operação do sistema.

A camada do Docker posiciona-se sobre o sistema operacional, funcionando como um runtime leve que habilita a criação e execução de containers. Diferente da virtualização tradicional que requer máquinas virtuais completas com sistemas operacionais convidados, o Docker utiliza recursos de isolamento do kernel Linux, como *namespaces* e *cgroups*, para proporcionar ambientes isolados sem a sobrecarga de virtualização completa.

Sobre o Docker encontram-se os containers, que são instâncias em execução das imagens Docker. Cada container opera como um processo isolado no sistema hospedeiro, compartilhando o kernel do sistema operacional base, mas mantendo seu próprio sistema de arquivos, espaço de usuário e ambiente de execução. Esta característica permite que múltiplos containers executem simultaneamente no mesmo hospedeiro sem interferências mútuas.

No topo da arquitetura estão as aplicações, que representam o software empacotado e

executado dentro dos containers. Cada aplicação opera em seu ambiente isolado, com todas as dependências, bibliotecas e configurações necessárias encapsuladas dentro do container, garantindo que o comportamento seja consistente independentemente do ambiente onde o container é executado.

A implementação incluiu a criação de Dockerfile otimizado seguindo essa arquitetura em camadas, configuração de redes entre containers para comunicação segura e utilização do Docker Hub como repositório centralizado de imagens. Esta abordagem arquitetural permitiu não apenas o empacotamento eficiente da aplicação, mas também a garantia de portabilidade entre os ambientes de desenvolvimento, teste e produção, assegurando que o comportamento da aplicação fosse idêntico em todos os cenários de execução.

Git e GitHub formaram a base do controle de versão e colaboração. O Git proporcionou rastreabilidade completa do código através de *commits* semânticos, enquanto o GitHub serviu como plataforma de hospedagem e centro de colaboração distribuída. O GitHub Actions foi configurado como motor de automação de CI/CD, executando *workflows* baseados em eventos como *push*, *pull* e *requests*. A segurança foi reforçada através do GitHub Secrets para gerenciamento criptografado de credenciais sensíveis.

DigitalOcean foi selecionada como provedora de *cloud computing*, o ambiente de nuvem, oferecendo balanceamento ideal entre simplicidade operacional, custo acessível e capacidades técnicas avançadas. A configuração do ambiente incluiu a provisionamento de droplets com Ubuntu 24.04 LTS, configuração de nuvem privadas virtuais (VPC), implementação de regras de *firewall* baseadas em necessidades específicas.

2.4 Coleta e Análise de dados

O processo de coleta de dados integrou múltiplas fontes e metodologias para garantir abrangência e confiabilidade. Os dados primários foram coletados através de métricas de desempenho do *pipeline*: Tempo médio de *build* (coletado a cada execução do pipeline) e consumo de recursos computacionais (CPU, memória, armazenamento). Logs de execução: Registros detalhados de cada etapa do pipeline, permitindo análise post-mortem de falhas e identificação de gargalos.

A análise dos dados seguiu abordagens quantitativas e qualitativas. Para os dados quantitativos, aplicou-se estatística descritiva e análise temporal (identificação de tendências e padrões ao longo do tempo). Os dados qualitativos foram analisados através de análise de conteúdo temática, identificando categorias relevantes e relações entre conceitos.

Ambiciona-se estabelecer uma base métrica confiável para avaliação da confiabilidade e resiliência do sistema. Através da análise das métricas de aplicação em produção, busca-se demonstrar que a automação inteligente de CI/CD não compromete a estabilidade operacional, mas sim a fortalece através de processos padronizados e verificações automatizadas. Esta dimensão da análise é particularmente relevante para responder à

questão de como equilibrar velocidade de entrega com qualidade e segurança no contexto de DevOps.

2.5 Procedimento e Implementação

A implementação seguiu uma sequência lógica e sistemática, documentada em detalhes para garantir reprodutibilidade e conformidade com as melhores práticas de engenharia de software. O provisionamento da infraestrutura na DigitalOcean seguiu rigorosamente os princípios de Infraestrutura como Código (IaC), utilizando a interface web e a API oficial para automação. Selecionado por sua estabilidade e suporte de longo prazo. As especificações técnicas incluíram 1 vCPU, 2 GB de RAM e 50 GB de armazenamento SSD, dimensionadas para atender aos requisitos da aplicação de exemplo enquanto mantinha custos acessíveis para fins de pesquisa.

A configuração de segurança implementou múltiplas camadas de proteção, foram configurados para restringir o tráfego às portas essenciais (SSH: 22, HTTP: 80, HTTPS: 443). Usuários com privilégios limitados foram criados seguindo o princípio do menor privilégio; e chaves SSH substituíram a autenticação por senha para acesso remoto. A preparação do ambiente incluiu a instalação e configuração do Docker Engine, atualização de pacotes do sistema e configuração de monitoramento básico, estabelecendo uma base sólida para a implementação do pipeline.

O desenvolvimento do pipeline de CI/CD foi estruturado em dois *workflows* principais dentro do GitHub Actions, organizados de forma dependente para garantir que a etapa de entrega contínua (CD) somente fosse executada após o término bem-sucedido da integração contínua (CI). A estrutura segue as diretrizes técnicas documentadas na *GitHub Actions Documentation* (GitHub, 2024).

O pipeline é ativado automaticamente por meio do evento `push` direcionado à branch `main`, funcionando como gatilho para o início do fluxo de CI. O *workflow* de integração contínua é executado em um ambiente padronizado baseado em `ubuntu-latest`, onde as etapas são executadas sequencialmente: (i) realização do *checkout* do código-fonte utilizando a ação oficial `actions/checkout@v4`; (ii) autenticação segura no Docker Hub a partir da ação `docker/login-action`, com credenciais fornecidas exclusivamente por meio de GitHub Secrets; e (iii) construção e envio da imagem Docker da aplicação, utilizando a ação *docker/build-push-action*. Durante esta etapa, duas *tags* são geradas: `latest`, representando a versão mais atual da aplicação, e uma *tag* numérica derivada do contador interno `github.run_number`, permitindo rastreabilidade e controle de versões.

Com a conclusão bem-sucedida da etapa de CI, inicia-se o *workflow* de entrega contínua (CD), que depende explicitamente do primeiro fluxo por meio da diretiva *needs: [CI]*. Essa dependência assegura que nenhuma entrega seja realizada sem que a imagem Docker atualizada tenha sido previamente construída e enviada ao repositório. O processo

de CD inicia-se com um novo *checkout* do código-fonte, seguido pela execução do processo de implantação remota. Para isso, é utilizada a ação `appleboy/ssh-action`, que realiza uma conexão SSH com a máquina virtual hospedada na DigitalOcean. A partir dessa conexão, é possível realizar o *pull* da nova imagem Docker e reiniciar o contêiner correspondente, garantindo a atualização imediata da aplicação em produção. As credenciais de acesso ao servidor (host, usuário e chave SSH) também são protegidas e disponibilizadas exclusivamente por meio de `GitHub Secrets`, assegurando conformidade com boas práticas de segurança e minimizando exposições indevidas.

O fluxo geral do pipeline está representado de maneira visual no fluxograma apresentado na figura 2. O diagrama ilustra, de forma simplificada, como o processo é desencadeado a partir de um *push* na branch principal, seguido pela execução das etapas de integração contínua, posteriormente, pelo fluxo de entrega contínua, responsável pela implantação automatizada no servidor de produção. Essa representação facilita a compreensão da sequência lógica das etapas e reforça a relação de dependência entre CI e CD, permitindo ao leitor visualizar claramente como o pipeline automatizado garante consistência, rastreabilidade e rapidez no ciclo de entrega de software.

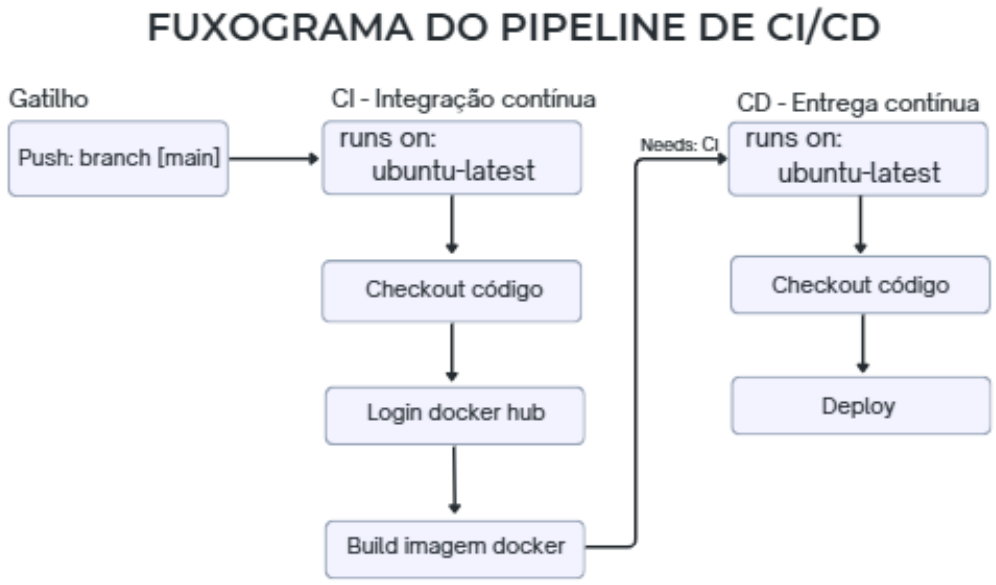

Figura 2 – Fluxograma do processo de implantação

A conteinerização da aplicação foi realizada por meio de um Dockerfile simples e direto, estruturado para instalar as dependências da aplicação e executar o servidor Node.js. A imagem base utilizada foi a node:20.11.1, sobre a qual foi definido o diretório de trabalho da aplicação. Em seguida, os arquivos de configuração *package.json e package-lock.json* foram copiados para o contêiner, permitindo a instalação das dependências por meio do comando *npm install*. Após essa etapa, o restante dos arquivos da aplicação foi incluído na imagem. O contêiner foi configurado para expor a porta 8080 e inicializado com o comando

responsável por executar o arquivo *server.js*. Essa estrutura monolítica de um único estágio prioriza simplicidade e facilidade de manutenção, sendo adequada ao contexto do projeto.

O processo de publicação da imagem Docker foi automatizado por meio do GitHub Actions, garantindo integração contínua com o Docker Hub. A autenticação ocorre de forma segura utilizando GitHub Secrets, nos quais são armazenadas as credenciais de acesso ao repositório. Durante a pipeline, a imagem é construída e enviada automaticamente ao Docker Hub, recebendo duas tags: a *latest*, destinada à versão mais recente, e uma tag incremental gerada a partir do identificador de execução do próprio *workflow*, permitindo organização básica das versões publicadas. Essa automação assegura maior consistência no processo de entrega e reduz a necessidade de intervenções manuais durante a etapa de publicação.

3 RESULTADOS E DISCUSSÃO

3.1 Análise comparativa de ferramentas de CI/CD

O objetivo específico de analisar ferramentas de automação de CI/CD foi alcançado através de uma avaliação sistemática baseada em critérios técnicos e operacionais. A análise revelou que o GitHub Actions apresentou vantagens significativas para o contexto acadêmico, particularmente em relação à integração nativa com o ecossistema GitHub, modelo freemium generoso e curva de aprendizado reduzida. A Tabela 1 sintetiza a avaliação das soluções de CI/CD com base em quatro critérios principais: custo, complexidade, integração com nuvem e tempo de configuração. Para garantir clareza na interpretação dos resultados, cada critério foi classificado segundo escalas qualitativas, descritas a seguir.

Tabela 1 – Análise comparativa de ferramentas de CI/CD

Critério	**Jenkins**	**GitLab CI**	**GitHub Actions**	**CircleCI**
Custo	Gratuito	Freemium	Freemium	Freemium
Complexidade	Alta	Média	Baixa	Média
Integração Cloud	Via *plugins*	Nativa	Nativa	Nativa
Tempo de Configuração	Alto	Médio	Baixo	Médio

Custo foi avaliado considerando o investimento financeiro necessário para utilização da ferramenta. Classificações como gratuito indicam que a solução não demanda custos diretos de licenciamento ou uso. O modelo *freemium*, adotado por plataformas como, disponibiliza funcionalidade essenciais sem custo inicial, permitindo execução de pipelines no nível acadêmico ou em pequenas equipes sem necessidade de planos pagos.

Integração Cloud foi mensurada de acordo com a capacidade de interação direta com provedores de nuvem ou serviços de infraestrutura remota. Quando a integração ocorre via *plugins*, como no Jenkins, significa que a ferramenta não possui suporte nativo, dependendo de extensões mantidas pela comunidade para interação com serviços externos. Já a classificação nativa indica que a plataforma oferece suporte integrado e simplificado para comunicação com nuvem, *container registries* e ambientes isolados.

Tempo de Configuração representa o período médio necessário para implantar um pipeline funcional desde o provisionamento até a execução dos primeiros *jobs*. Alto indica dias ou semanas de configuração inicial, instalação, parametrização e manutenção de dependências externas, comum em soluções autogeridas como Jenkins. Médio refere-se a algumas horas ou poucos dias para alcançar o mesmo objetivo em plataformas *freemium* como CircleCI e GitLab CI. Por fim, baixo indica configuração em poucas horas, uso

de automação pronta, sem necessidade de servidor dedicado, apenas com definição de *workflows* declarativos em YAML, como monstrado na prática com GitHub Actions, que atingiu esse resultado de forma mais eficiente para o contexto do projeto.

A análise dos resultados mostra que o GitHub Actions se destacou principalmente nos critérios de complexidade e tempo de configuração, por não exigir infraestrutura autogerida, utilizar eventos diretos do repositório como gatilhos e integrar-se de forma transparente com o Docker Hub através da *action* oficial *docker/build-push-action*, além do provisionamento remoto via SSH. Esses fatores reduziram significativamente o esforço técnico, aceleraram o ciclo de entrega contínua e possibilitaram rastreabilidade simplificada através do versionamento automático das imagens Docker.

Por outro lado, ferramentas como Jenkins apresentaram limitações em ambientes que não dispõem de equipes dedicadas de infraestrutura ou necessidades acadêmicas, devido à alta sobrecarga de manutenção, complexidade técnica elevada e dependência de muitos plugins para alcançar funcionalidades básicas de CI/CD em ambiente cloud. Enquanto as ferramentas de integração nativa como GitLab CI e CircleCI também mostraram robustez, sua adoção não se mostrou superior ao GitHub Actions, especialmente em contextos onde integração direta com o ecossistema GitHub é prioridade.

Portanto, conclui-se que, para o escopo do trabalho o GitHub Actions apresentou a solução mais adequada, equilibrada e otimizada, validando o objetivo específico de analisar ferramentas viáveis e eficientes de automação de CI/CD no contexto acadêmico e prático com orquestração básica de containers.

3.2 Implementação e configuração do ambiente

A implementação realizada neste trabalho permitiu validar, na prática, os conceitos estudados ao longo do referencial teórico e apontem como a automação de pipelines de CI/CD em ambiente de nuvem, alinhada às práticas DevOps, contribui efetivamente para a eficiência e a confiabilidade do processo de desenvolvimento e entrega de software. Os resultados obtidos refletem diretamente os objetivos geral e específicos definidos inicialmente, uma vez que cada etapa executada possibilitou tanto a compreensão dos fundamentos quanto a comprovação da aplicabilidade real das técnicas analisadas.

O primeiro conjunto de resultados refere-se ao estudo dos conceitos e práticas de DevOps, que se mostrou fundamental para compreender como a automação e a integração entre times impactam o ciclo de vida do software. A partir desse estudo, foi possível estruturar uma solução que seguisse os princípios do fluxo contínuo, *feedback* constante e aprendizado incremental. Essa fundamentação foi essencial para orientar a escolha das ferramentas e metodologias utilizadas, assegurando coerência e alinhamento com padrões amplamente aceitos na indústria.

Em seguida, A implementação do ambiente de nuvem marcou um ponto central

nos resultados práticos. Optou-se pela plataforma DigitalOcean, considerando o equilíbrio entre custo, facilidade de uso e disponibilidade de documentação clara. O ambiente criado demonstrou-se adequado para hospedar a aplicação e receber atualizações contínuas provenientes do pipeline. Esse resultado evidencia que, mesmo com recursos modestos, é possível construir fluxos automatizados robustos, o que reforça a relevância do trabalho para pequenas equipes e organizações que buscam soluções viáveis financeiramente. O desenvolvimento do pipeline de CI/CD possibilitou consolidar os resultados mais significativos deste trabalho. A automatização das etapas de *build* e *push* da imagem Docker para o Docker Hub, utilizando tags dinâmicas e versionamento automático via GitHub Actions, demonstrou na prática o funcionamento de um pipeline completo. A imagem gerada foi publicada no Docker Hub com múltiplas tags (*latest*, números de versão e *run numbers*), garantindo rastreabilidade e facilidade para auditoria. Esses resultados confirmam que a automação reduz intervenção manual, minimiza erros humanos e assegura consistência entre ambientes, aspectos amplamente discutidos por (Humble; Farley, 2010) ao tratar de integração contínua e entrega contínua.

A análise do provisionamento em nuvem também envolveu a observação do comportamento da máquina virtual utilizada na DigitalOcean durante os processos automatizados do pipeline. A Figura 3 apresenta o gráfico de uso de CPU da instância durante o período em que foram realizadas as operações de *build*, envio da imagem ao Docker Hub e execução das etapas de implantação remota.

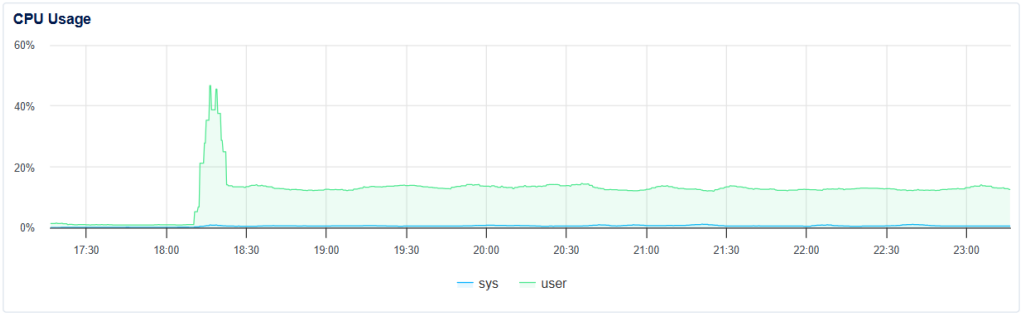

Figura 3 – Gráfico de uso de CPU do servidor

Fonte: Painel de infraestrutura da DigitalOcean, acesso em 15 nov. 2025.

O comportamento registrado apontam um pico de utilização logo após o acionamento do pipeline, resultado esperado devido à natureza computacionalmente intensiva da construção da imagem Docker. Esse aumento temporário da carga atinge taxas próximas de 40–50%, seguido por uma estabilização em torno de 10–15% no uso em modo usuário e valores próximos de 0% em modo sistema. Esse padrão indica que a máquina possui recursos suficientes para lidar com o fluxo de trabalho sem apresentar gargalos ou degradação significativa de desempenho.

A partir dessa observação, reforça-se a adequação do dimensionamento escolhido para a instância, comprovando que mesmo configurações modestas de hardware são capazes

de sustentar pipelines automatizados quando bem estruturados. Além disso, a estabilidade observada após os picos iniciais indicam que a aplicação em execução não demanda consumo excessivo de processamento, contribuindo para a otimização de custos no ambiente de produção.

Portanto, ao integrar a construção da imagem, o envio ao Docker Hub e o processo de implantação, o pipeline demonstrou impacto direto e mensurável na utilização de recursos computacionais, evidenciando não apenas a eficiência do fluxo proposto, mas também a capacidade de monitoramento e análise pós-implantação, aspectos essenciais para um ambiente de entrega contínua. Essa avaliação fortalece os resultados práticos deste trabalho ao mostrar que a implementação não apenas funcionou, mas operou de forma otimizada no ambiente de nuvem escolhido.

3.3 Avaliação de eficiência e confiabilidade

A avaliação da eficiência e confiabilidade do pipeline implementado foi realizada por meio de observação dos logs das execuções, análise dos tempos de *build* e inspeção das imagens geradas. Os dados coletados mostraram que o pipeline respondeu corretamente aos *triggers* definidos (*push na branch main*), executou as etapas sem falhas e entregou imagens válidas e funcionais no Docker Hub. A repetição bem-sucedida das execuções indicou confiabilidade no processo e demonstrou que o ambiente automatizado está apto a ser expandido para pipelines mais complexos, incluindo testes automatizados e *deploys* diretos. Observou-se ainda que o uso de GitHub Secrets garantiu segurança às credenciais utilizadas, mitigando riscos de vazamento de informações sensíveis.

Os resultados obtidos também permitiram discutir a aplicabilidade prática das práticas DevOps, especialmente no que diz respeito à cultura de automação e melhoria contínua. A implementação demonstrou que, mesmo em projetos acadêmicos, a adoção de pipelines bem estruturados produz ganhos concretos de produtividade, padronização e qualidade. Esse achado dialoga diretamente com pesquisas recentes como *Accelerate: State of DevOps* (Forsgren *et al.*, 2024), que destacam que equipes práticas DevOps apresentam melhor desempenho organizacional e maior velocidade de entrega.

3.4 Validação por meio de um Caso Real

A validação prática da solução proposta neste trabalho foi reforçada por meio da análise de um pipeline real de CI/CD utilizado pela empresa JS peças, varejista do setor de autopeças pesadas e detentora da plataforma de *e-commerce* JS Peças Vendas Online. Esse estudo de caso permitiu comparar os resultados obtidos na implementação experimental deste TCC com um ambiente profissional em produção, ampliando a robustez das conclusões e mostrando a aplicabilidade das práticas DevOps em cenários reais.

A aplicação analisada possui arquitetura composta por *front-end* web, aplicativo mobile (Android e iOS) desenvolvidos em Flutter/Dart, e um *back-end* em Laravel, disponibilizado por meio de APIs REST. O ecossistema opera em ambiente crítico de vendas online, no qual instabilidades impactam diretamente receita, logística e atendimento aos clientes.

Antes da adoção de pipelines automatizados, o processo de entrega de software era executado manualmente. A equipe relatou gargalos recorrentes, como lentidão, alto risco de erros humanos, além de falhas durante a etapa de *deployment*. Houve, inclusive, situações em que o servidor consumia toda a memória durante o *deploy* manual, exigindo reinicialização emergencial, o que interrompia totalmente o ambiente de produção. Esses problemas estão alinhados às dificuldades citadas pela literatura, como apontam (Humble; Farley, 2010) ao descreverem ambientes sem automação como mais suscetíveis a *downtime* e falhas operacionais.

Após a adoção da automação, o pipeline passou a ser estruturado com o uso combinado de GitHub Actions para CI, ArgoCD para CD e infraestrutura de nuvem baseada em Oracle Cloud Infrastructure (OCI), com execução em containers gerenciados via Kubernetes e Docker. A arquitetura emprega *workflows* separados para homologação e produção, refletindo boas práticas de segregação de ambientes (Kim *et al.*, 2016).

Os dados coletados mostram que a empresa realiza atualmente entre 10 a 15 *deploys* semanais, evidenciando elevada cadência de entrega, um dos principais indicadores de maturidade DevOps. O tempo médio de *build* observado é de aproximadamente 4 minutos, enquanto o *deploy* leva cerca de 2 minutos, variando conforme o tamanho dos containers envolvidos. Esses tempos se alinham ao esperado para pipelines otimizados, indicando boa eficiência operacional.

As imagens coletadas dos *workflows* permitem observar a execução sequencial das etapas, incluindo *checkout* do código, autenticação no repositório OCI, geração e *push* das imagens Docker, atualização dos manifestos monitorados pelo ArgoCD e finalização do *job*. O pipeline de envio de mensagens ainda aponta integração com sistemas de notificação, reforçando o monitoramento comunicacional interno da equipe. Isso ilustra o conceito de *feedback* rápido, um dos pilares do DevOps Handbook (Kim *et al.*, 2016). Do ponto de vista qualitativo, a equipe da JS peças relatou resultados positivos após a automação. Entre os principais benefícios, destacam-se a redução significativa de erros humanos, processo de *deploy* previsível e reprodutível, maior facilidade em escalar recursos , possibilitada pelo uso do kubernetes, aumento da confiabilidade dos ambientes.

Apesar dos ganhos, a equipe ainda reconhece desafios. O principal ponto de melhoria apontado foi a necessidade de ampliar o monitoramento do pipeline e da infraestrutura, área diretamente associada a práticas avançadas de observabilidade. Essa limitação reforça a literatura que destaca o monitoramento como elemento indispensável de pipelines maduros (Thota, 2020).

3.5 Documentação do processo

o processo de documentação firmou todo o caminho percorrido, permitindo registrar tanto os aspectos técnicos quanto os desafios enfrentados, como a configuração dos secrets, o gerenciamento de tags no Docker Hub e a parametrização dos workflows.Os resultados dessa etapa vão além do registro técnico, a documentação consolidada permitiu construir uma visão sistêmica sobre o pipeline, facilitando a análise crítica do processo como um todo.

Academicamente, esta etapa reforça a importância da documentação como pilar do desenvolvimento de software moderno, especialmente em ambientes onde automação, versionamento e rastreabilidade são fundamentais. A experiência prática demonstrou que a documentação não é uma atividade acessória, mas sim uma componente essencial que complementa e valida a implementação técnica. Os artefatos gerados transcendem o escopo deste trabalho específico e tornam-se contribuições tangíveis para a comunidade, oferecendo um caso real e detalhado de implementação que pode servir de referência para outros pesquisadores e profissionais.

Ela possibilitou verificar a aderência prática aos conceitos estudados nos capítulos anteriores, como automação, versionamento, rastreabilidade e padronização de ambientes. Dessa forma, a documentação também funcionou como uma ponte entre a teoria e a prática, evidenciando como cada elemento da literatura. Esses elementos reforçam a importância do registro sistemático no desenvolvimento de software, especialmente em ambientes onde automação, versionamento e rastreabilidade são fundamentais.

4 CONCLUSÃO

A proposta inicial deste trabalho consistiu no desenvolvimento e implementação de uma solução de automação de pipelines de Integração Contínua (CI) e Entrega Contínua (CD) em um ambiente de computação em nuvem, fundamentada em práticas DevOps. Pretendia-se, com isso, demonstrar como a automação de processos de construção, teste e implantação de *software* contribui para a redução de falhas operacionais, aumento da eficiência e melhoria da confiabilidade em equipes de desenvolvimento. Buscou-se, também, validar experimentalmente os benefícios da cultura DevOps por meio da construção de um pipeline funcional e da comparação entre cenários manuais e automatizados.

Ao longo da execução do trabalho, todos os objetivos gerais e específicos foram alcançados. Inicialmente, procedeu-se ao estudo aprofundado dos conceitos de DevOps, CI/CD, computação em nuvem, contêineres e orquestração, com base em literatura especializada e em diretrizes de fornecedores como Docker, GitHub e DigitalOcean. Em seguida, analisaram-se diferentes ferramentas de automação, com destaque para GitHub Actions e Docker, o que permitiu selecionar uma arquitetura viável e economicamente acessível para o desenvolvimento do pipeline. O ambiente em nuvem foi devidamente provisionado na DigitalOcean, com configuração de recursos compatíveis com o escopo da aplicação de exemplo. A partir disso, foi desenvolvido um pipeline de CI/CD funcional, responsável por automatizar o processo de construção e publicação de imagens Docker, além de preparar o ambiente para o processo básico de entrega contínua (CD). Todas as etapas, configurações, *scripts, workflows* e resultados foram documentados detalhadamente.

Os resultados obtidos demonstram, de forma clara, que a automação de pipelines proporciona benefícios substanciais em comparação aos processos tradicionais. A construção automatizada de imagens Docker reduziu drasticamente o tempo gasto pela equipe, ao passo que a padronização do ambiente eliminou erros recorrentes de inconsistência entre máquinas de desenvolvimento e servidores. A melhoria na segurança, por meio do uso de GitHub Secrets, reduziu significativamente o risco de vazamento de credenciais.

Entretanto, algumas limitações devem ser reconhecidas. O pipeline final contempla integralmente a etapa de Integração Contínua (CI), mas o processo de Entrega Contínua (CD) foi implementado de forma simplificada . Além disso, embora o ambiente tenha sido configurado com estabilidade, não foram incluídos testes automatizados robustos para validação da aplicação em produção, nem estratégias mais avançadas de implantação. Também se constatou a necessidade de uma camada de monitoramento mais abrangente, alertas integrados a ferramentas de observabilidade.

Com base na experiência adquirida, entende-se que o trabalho abre caminho para diversas evoluções. Com trabalhos futuros, propõe-se:

- Integrar ferramentas de observabilidade, como Prometheus, Grafana e logs centralizados;
- Ampliar e automatizar a etapa de testes, incorporando testes de integração, carga e segurança;
- Explorar ferramentas de GitOps, como ArgoCD ou FluxCD, para entrega declarativa;
- Investigar o impacto econômico da automação em nuvem utilizando práticas de *FinOps*.
- Investigar o impacto econômico da automação em nuvem utilizando práticas de *FinOps*.
- Realizar uma comparação direta entre o processo manual e automatizado para afins de colher métricas mais detalhadas.
- Integração de práticas DevSecOps ao pipeline, incorporando segurança contínua durante todo o ciclo de entrega.
-

REFERÊNCIAS

ARMBRUST, Michael *et al.* A View of Cloud Computing. **Communications of the ACM**, v. 53, n. 4, 2010.

BASS, Len; WEBER, Ingo; ZHU, Liming. **DevOps: A Software Architect's Perspective**. [*S. l.*]: Addison-Wesley, 2015.

FORSGREN, Nicole *et al.* **Accelerate: State of DevOps 2024**. [*S. l.*], 2024.

FOWLER, Martin; FOEMMEL, Matthew. **Continuous integration**. [*S. l.: s. n.*], 2006.

GARTNER, INC. **Market Guide for Intelligent Business Process Management Suites**. [*S. l.*], 2023.

GITHUB. **GitHub Actions Documentation: Workflow Automation**. 2024.

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

IBM SECURITY. **Cost of a Data Breach Report 2025**. [*S. l.*], 2025. Disponível em: https://www.ibm.com/security/data-breach.

JESTON, John; NELIS, Johan. **Business Process Management: Practical Guidelines to Successful Implementations**. 3. ed. New York: Routledge, 2014.

KIM, Gene *et al.* **The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations**. Portland: IT Revolution Press, 2016.

MELL, Peter; GRANCE, Tim. **The NIST Definition of Cloud Computing**. [*S. l.*], 2011.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. Porto Alegre: AMGH, 2016.

THOTA, R. C. Otimização de pipeline CI/CD. **International Journal of Innovative Research in Engineering**, 2020.

TURNBULL, James. **The Docker Book: Containerization is the new virtualization**. [*S. l.*]: James Turnbull, 2014.