Avaliação de soluções de CI/CD: Uma Análise Comparativa entre GitHub Actions e GitLab CI

Lucas P. Graciano[1], **Gustavo K. Kakinohana**[1]

[1]FACOM - Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 15.064 – 91.501-970 – Campo Grande – MS – Brazil

{lucas.graciano, gustavo.kikey}@ufms.br

Abstract. *This article presents a comparative analysis between the Continuous Integration platforms GitHub Actions and GitLab CI, with the aim of determining which one is best suited to meet the needs of software development teams. The research involved collecting opinions from development professionals and executing pipelines in both environments. The results revealed significant nuances between the two solutions, highlighting crucial factors for choosing the most appropriate platform. In conclusion, this study provides valuable guidance for development teams as they face the challenge of selecting the CI platform that best meets their needs, promoting efficiency, quality, and agile deliveries.*

Resumo. *Este artigo apresenta uma análise comparativa entre as plataformas de Integração Contínua GitHub Actions e GitLab CI, com o objetivo de determinar qual delas é mais adequada para atender às necessidades das equipes de desenvolvimento de software. A pesquisa envolveu a coleta de opiniões de profissionais da área de desenvolvimento e a execução de pipelines em ambos os ambientes. Os resultados revelaram nuances significativas entre as duas soluções, destacando fatores cruciais para a escolha da plataforma mais adequada. Em conclusão, este estudo fornece orientações valiosas para equipes de desenvolvimento ao enfrentar o desafio de selecionar a plataforma de CI que melhor atenda às suas necessidades, promovendo eficiência, qualidade e entregas ágeis.*

1. Introdução

A Integração Contínua (CI) é uma prática central da engenharia de software moderna, revolucionando a forma como as equipes de desenvolvimento concebem, implementam e entregam software [Rehkopf 2023]. Com a demanda incessante por inovação e a crescente complexidade dos projetos, a CI se tornou uma prática indispensável para organizações que buscam manter um ciclo de desenvolvimento eficiente e a qualidade do software [Augustino 2023].

A CI permite que as equipes integrem e testem seu código de forma contínua, identificando problemas precocemente e possibilitando lançamentos mais confiáveis e frequentes. A CI é amplamente adotada em empresas de todos os setores e tamanhos, tornando-se uma abordagem fundamental para manter a competitividade no mercado [Rehkopf 2023].

A CI é a prática de integrar o código de um projeto continuamente, submetendo-o a testes automatizados à medida que é desenvolvido. Isso garante a detecção precoce de

erros e a manutenção da qualidade do software em todos os estágios do desenvolvimento [Augustino 2023]. Junto com essa prática, também temos a Entrega Contínua (CD), que é um método que busca automatizar o ciclo completo de disponibilização de software, desde a compilação até sua implementação em ambiente de produção.

Nesse sentido, uma análise comparativa das plataformas GitHub Actions e GitLab CI se torna necessária uma vez que, na área de CI/CD, essas são as plataformas mais recorrentes. Ambas as plataformas se destacam em suas capacidades de automação de pipelines de CI/CD e têm uma base de usuários considerável.

No cenário atual, GitHub Actions e GitLab CI são concorrentes diretos no mercado de Integração Contínua. Ambas as plataformas são altamente competitivas e continuamente aprimoram suas funcionalidades para atender às demandas das equipes de desenvolvimento. Essa concorrência resulta em inovações constantes e em uma ampla gama de recursos para os usuários.

No entanto, apesar da abundância de opções de CI disponíveis, a escolha entre o GitHub Actions e o GitLab CI pode ser desafiadora. As nuances das características de cada plataforma podem impactar significativamente a eficiência das equipes de desenvolvimento. Portanto, a questão central deste estudo é a seguinte:

Qual das plataformas de Integração Contínua, GitHub Actions ou GitLab CI, é mais apropriada para atender às necessidades específicas das equipes de desenvolvimento de software?

Para responder a essa pergunta, este estudo concentra-se na comparação detalhada de recursos, desempenho e usabilidade entre o GitHub Actions e o GitLab CI. O objetivo é fornecer orientações sólidas e imparciais para auxiliar as equipes de desenvolvimento a tomar decisões informadas ao escolher a plataforma de CI mais adequada para seus projetos, promovendo eficiência, qualidade e entrega rápida de software.

A princípio, o estudo teve um foco na análise comparativa das plataformas de Integração Contínua GitHub Actions e GitLab CI, ambas conhecidas por suas capacidades de automação de pipelines de CI/CD e com uma base de usuários considerável. A metodologia adotada envolverá uma abordagem abrangente para avaliar essas duas soluções com base em critérios diversos, incluindo recursos, desempenho e usabilidade.

Para alcançar os objetivos apresentados, foi realizada uma pesquisa de campo, coletando opiniões de profissionais da área de desenvolvimento de software. Essas opiniões forneceram dicas valiosas sobre a preferência dos usuários em relação às duas plataformas. Adicionalmente, conduzimos testes práticos executando pipelines em ambos os ambientes para avaliar o desempenho e a eficiência das plataformas em um cenário real.

Como resultado, uma análise comparativa entre duas das principais plataformas de CI, o GitHub Actions e o GitLab CI é apresentada. Os insumos obtidos revelam informações importantes para as equipes de desenvolvimento de software que buscam escolher a plataforma de CI mais apropriada para suas necessidades.

Em suma, as conclusões deste estudo não apenas destacam as vantagens oferecidas por cada plataforma, mas também enfatizam a importância de considerar fatores como a complexidade do projeto, recursos disponíveis e curva de aprendizado ao fazer essa escolha. Esses resultados fornecem uma base para tomada de decisão e orientação na

escolha da plataforma de CI mais adequada.

2. Referencial teórico

2.1. Integração Contínua

A Integração Contínua (CI) é uma prática fundamental no desenvolvimento de software que envolve a integração regular e automatizada do código produzido por diferentes membros da equipe. Ela desempenha um papel crucial no contexto DevOps e na entrega contínua de software [Augustino 2023].

A CI é uma prática fundamental no desenvolvimento de software que envolve a integração regular e automatizada do código produzido por diferentes membros da equipe de desenvolvimento em um repositório central. Essa prática é essencial para evitar a fragmentação do código, detectar conflitos precocemente e garantir a qualidade do software durante todo o ciclo de desenvolvimento [Augustino 2023].

A CI desempenha um papel crítico na entrega de software confiável e de alta qualidade. Ela envolve a integração contínua das alterações no código-fonte, permitindo que a equipe de desenvolvimento detecte problemas de integração rapidamente e garanta que o software esteja sempre em um estado funcional. A CI também promove a colaboração entre os membros da equipe, uma vez que todos trabalham com base em um código centralizado e atualizado regularmente [Paul et al. 2007].

[Paul et al. 2007] ressaltam que a Integração Contínua contribui para a redução de conflitos de integração, identificação precoce de erros e a manutenção da estabilidade do projeto. Somado a isso, ela apoia a entrega contínua de software, garantindo que cada alteração seja integrada e testada de maneira consistente.

A CI é composto por etapas cruciais que são executadas de forma contínua, garantindo a qualidade do software. [Paul et al. 2007] fornecem informações valiosos sobre como esses processos funcionam de maneira coordenada:

Compilação: A compilação é o processo de transformar o código-fonte em um formato executável. Na CI, a compilação ocorre sempre que um desenvolvedor integra suas alterações no repositório central. Isso garante que o código seja compilado regularmente, o que é vital para identificar erros de compilação precocemente. Quando a compilação falha, os desenvolvedores são notificados imediatamente, permitindo correções imediatas [Paul et al. 2007].

Testes:A fase de teste na CI é fundamental para garantir a qualidade do software. Os testes automatizados, incluindo testes unitários e de integração, são executados sempre que uma alteração é integrada. Isso ajuda a identificar problemas de lógica, erros de programação e problemas de integração. Os testes automatizados garantem que o software continue funcionando conforme o esperado, mesmo com mudanças contínuas no código [Paul et al. 2007].

2.2. Entrega Contínua

Entrega Contínua (CD) é uma prática que visa automatizar todo o processo de entrega de software, desde a compilação até a implantação em produção. Ela se diferencia da CI por ser uma prática mais abrangente e que envolve todo o processo de entrega do software. A CI se concentra apenas na compilação e nos testes unitários do código-fonte [Hat 2023].

A CD, por outro lado, inclui a automação de testes funcionais, testes de aceitação e implantação em produção. Isso permite que as equipes de desenvolvimento possam entregar software com mais rapidez e eficiência. A Entrega Contínua tem como objetivo principal a automação do processo de entrega de software, desde a compilação até a implantação em produção. Isso inclui a automação de testes funcionais, testes de aceitação e implantação em produção. A automação dos pipelines de CD é fundamental para garantir a eficiência e a confiabilidade do processo de entrega. Com a automação dos testes e validações, é possível garantir que o software esteja sempre pronto para ser implantado em produção [Institute 2023].

Em complemento a isso, as estratégias de entrega permitem que as equipes possam implantar novas versões do software sem interromper o acesso dos usuários finais ao sistema.

Os objetivos da CD incluem a automação de implantação, testes e validações, além de garantir que o software esteja sempre pronto para ser implantado em produção [Hat 2023]. A automação dos pipelines é fundamental para garantir a eficiência e a confiabilidade do processo de entrega. Com a automação dos testes e validações, é possível garantir que o software esteja sempre pronto para ser implantado em produção.

A CD tem como objetivo principal a automação do processo de entrega de software, desde a compilação até a implantação em produção. Isso inclui a automação de testes funcionais, testes de aceitação e implantação em produção. A automação dos pipelines de CD é fundamental para garantir a eficiência e a confiabilidade do processo de entrega. Com a automação dos testes e validações, é possível garantir que o software esteja sempre pronto para ser implantado em produção [Hat 2023].

A automação de pipelines de CD é um dos pilares que possibilita a eficiência e a consistência desse processo [Hat 2023]. Um pipeline de CD é composto por uma série de etapas automatizadas que conduzem o software desde a fase de desenvolvimento até a implantação em produção. A automação assegura que todas as etapas do pipeline sejam executadas sem a necessidade de intervenção manual, reduzindo consideravelmente o risco de erros humanos e garantindo que o software seja entregue de forma confiável. Os pipelines de CD geralmente incluem as seguintes etapas:

Compilação: A primeira etapa envolve a compilação do código-fonte, onde o código é transformado em um pacote executável.

Testes automatizados: Após a compilação, uma bateria de testes automatizados é executada. Isso abrange desde testes unitários e testes de aceitação e testes de desempenho. A automação dos testes garante que o software seja constantemente validado em busca de erros e regressões.

Implantação em ambientes de teste e homologação: O software é implantado automaticamente em ambientes de teste e homologação, permitindo uma validação completa antes de chegar à produção.

Implantação em produção: Uma vez que o software tenha passado por todas as etapas de teste com sucesso, ele é implantado em um ambiente de produção de maneira automática. Isso pode ser feito em diferentes servidores e *data centers* para garantir alta disponibilidade.

Monitoramento e *feedback* contínuo: O *pipeline* de CD é projetado para monitorar continuamente o software em produção. Isso inclui a detecção de problemas em tempo real, garantindo que qualquer erro seja rapidamente identificado e corrigido.

A automação dos pipelines de CD é essencial para garantir a eficiência e a rapidez do processo de entrega, pois elimina atrasos decorrentes de intervenções manuais e minimiza erros humanos [Fitzgerald et al. 2020]. Isso possibilita que as equipes de desenvolvimento entregam software de alta qualidade de forma consistente e previsível, beneficiando tanto os desenvolvedores quanto os usuários finais.

2.3. GitHub Actions

O GitHub Actions™ é uma plataforma de automação nativa da plataforma GitHub, lançada em 2018. Desde então, a ferramenta evoluiu para oferecer aos desenvolvedores capacidades de automação poderosas e integração contínua/implantação contínua (CI/CD) diretamente ao lado do código-fonte hospedado no GitHub. Sua proposta é simplificar os fluxos de trabalho com automação flexível e fornecer funcionalidades de CI/CD acessíveis e prontas para uso, construídas por desenvolvedores, para desenvolvedores [GitHub 2023a].

A princípio, o GitHub Actions permite que os desenvolvedores automatizam tarefas de desenvolvimento, como a compilação de código, testes, implantação e muito mais, diretamente no ecossistema do GitHub [GitHub 2023a]. Uma de suas vantagens distintivas é a integração nativa e a proximidade com o repositório, o que simplifica a configuração e a execução de fluxos de trabalho de automação.

Os *workflows* são essenciais para a automação de tarefas de desenvolvimento e integração contínua. Eles representam processos automatizados configuráveis, projetados para simplificar e otimizar o ciclo de desenvolvimento de software [GitHub 2023b]. Esses fluxos são altamente customizáveis e podem ser adaptados para atender às necessidades específicas de um projeto. Eles são definidos por meio de arquivos de formato YAML, que residem no diretório .github/*workflows* dentro do seu repositório.

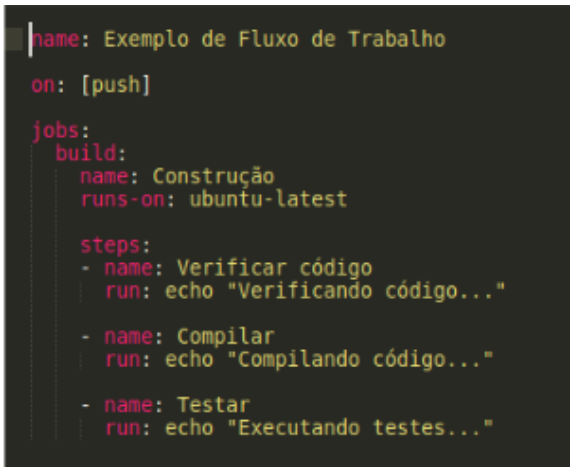

Figure 1. Exemplo de Workflow

A figura 1 apresenta o exemplo na qual criamos um arquivo YAML para um workflow chamado “Exemplo de Fluxo de Trabalho”. O fluxo de trabalho é acionado pelo evento de “*push*” em seu repositório. Ele consiste em um único trabalho chamado “Construção” que roda na última versão do ambiente Ubuntu disponível.

O trabalho é dividido em três etapas, cada uma realizando uma tarefa específica. No exemplo da Figura 1, os *steps* simulam a verificação do código, a compilação e a execução de testes respectivamente. Em uma situação realista, devemos substituir os comandos ”*echo*” por comandos reais para verificar, compilar e testar seu código.

Como apresentado na Figura 1, é possível controlar quais tipos de eventos iriam acionar o *workflow*.. Por exemplo, você pode configurar um *workflow* para ser executado automaticamente sempre que alguém envia uma solicitação *pull* ou quando ocorrem pedidos de integração entre as ramificações. Esses eventos acionadores comumente chamados de triggers, garantem que a automação seja realizada apenas quando necessário, economizando recursos e tempo.

Essa funcionalidade também permite uma configuração para execução manual, o que pode vir a ser muito produtivo em situações onde a intervenção humana é necessária, como uma situação de *deploy* em produção a qual é necessário colher autorizações dos responsáveis.

Adicionalmente, o GitHub Actions permite a definição de vários *workflows* em um único repositório. Isso é valioso quando diferentes partes de um projeto têm requisitos de automação distintos. Cada fluxo é capaz de realizar um conjunto diferente de tarefas, permitindo que você crie processos automatizados específicos para compilação, testes, implantação, entre outros [GitHub 2023b].

Sendo assim, podemos dizer que os *workflows* no GitHub Actions fornecem uma flexibilidade notável para automatizar e gerenciar todas as etapas do ciclo de desenvolvimento de software. Eles podem ser facilmente configurados, personalizados para atender às necessidades específicas do seu projeto e são acionados por eventos ou de forma manual, proporcionando um controle preciso sobre a automação em seu ambiente de desenvolvimento.

O GitHub Actions™ oferece uma integração contínua altamente eficiente com repositórios hospedados no GitHub. Isso permite a configuração de fluxos de trabalho que lidam com a construção e teste de cada solicitação pull enviada ao seu repositório, além de automatizar a implantação das solicitações pull mescladas no ambiente de produção. A ferramenta vai além do DevOps tradicional, permitindo que você execute fluxos de trabalho quando outros eventos ocorrem em seu repositório [GitHub 2023b].

Essa flexibilidade possibilita a automação de tarefas que vão desde testes automatizados até a implantação contínua, proporcionando um ambiente altamente controlado e eficiente para o desenvolvimento de software [GitHub 2023b]. Em suma, o GitHub Actions se destaca como uma solução robusta para automatizar fluxos de trabalho de desenvolvimento e CI/CD, estreitamente integrada com o GitHub, proporcionando aos desenvolvedores uma ferramenta poderosa e versátil para seus projetos.

2.4. GitHub Actions

O GitLab CI/CD faz parte do ecossistema GitLab, que é uma plataforma abrangente de desenvolvimento de software. Essa plataforma oferece um ambiente centralizado para o gerenciamento de repositórios de código, rastreamento de problemas, colaboração em projetos e automação de processos de entrega de software. O GitLab CI/CD é uma extensão fundamental deste ecossistema, fornecendo soluções nativas para integração contínua e entrega contínua [GitLab 2023a].

Essa ferramenta foi projetada com uma ênfase especial na integração perfeita com o GitLab. Muito semelhante ao GitHub Actions, ela permite que você automatize processos diretamente em seu ambiente de desenvolvimento, possibilitando a capacidade de gerenciar tudo em um único lugar, desde o código-fonte até a implantação em produção, uma grande vantagem em relação a outras ferramentas de CI/CD como Jenkins™ e Travis CI™. Isso simplifica o processo de desenvolvimento e economiza tempo ao evitar a necessidade de integrar várias ferramentas de terceiros. Além disso, as *pipelines* do GitLab CI são altamente avançadas e customizáveis, oferecendo ambiente isolado, revisões de código e rastreamento de métricas, tornando-o uma solução completa para automação de desenvolvimento de ponta a ponta [GitLab 2023b].

No GitLab CI/CD, as *pipelines* são uma representação visual dos processos de automação. Elas são definidas por meio de um arquivo chamado .gitlab-ci.yml, que reside no diretório do seu repositório GitLab [GitLab, 2023]. Este arquivo YAML descreve como os estágios de pipeline são configurados e quais ações devem ser executadas em cada estágio. Cada estágio é composto por um conjunto de trabalhos, que podem ser executados em paralelo ou sequencialmente.

Os estágios comumente conhecidos como *jobs*, são totalmente personalizáveis e permitem a configuração de tarefas específicas, como compilação, testes, análise de código, implantação e muito mais. Isso possibilita a criação de *pipelines* adaptadas às necessidades específicas do seu projeto [GitLab, 2023]. Também, as *pipelines* podem ser acionadas automaticamente em resposta a eventos, como push de código, ou serem agendadas para execução em horários específicos. Os resultados de cada job são registrados, permitindo que você acompanhe o progresso e identifique possíveis problemas rapidamente.

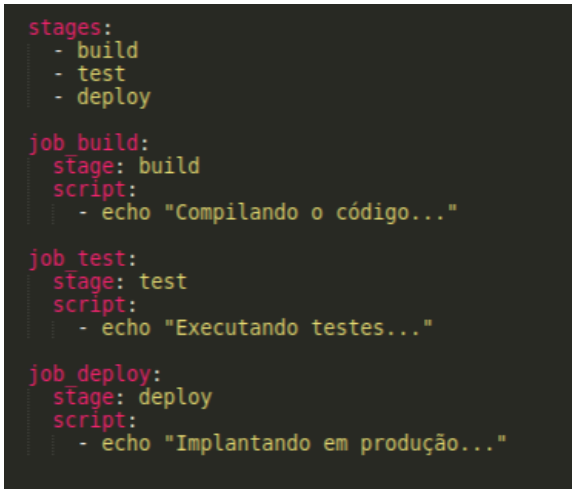

Figure 2. Exemplo de pipeline

O exemplo da Figura 2 demonstra a definição de um arquivo .gitlab-ci.yml que configura uma pipeline com três estágios: "*build*", "*test*" e "*deploy*". Cada estágio contém um único trabalho que executa tarefas simuladas. Na prática, deveríamos substituir os comandos "echo" por comandos reais, como compilação, testes e implantação do código.

A integração do GitLab CI/CD com os repositórios GitLab é uma das principais vantagens desta solução. Ela permite que você configure e gerencie pipelines diretamente a partir do ambiente do GitLab. Ao criar um arquivo .gitlab-ci.yml (Figura 2) no seu repositório e configurar os estágios, você pode automatizar tarefas de desenvolvimento, integração e entrega de forma eficaz.

As esteiras podem ser acionadas automaticamente em resposta a eventos, como push de código para o repositório, ou podem ser agendadas para execução em momentos específicos, o que é útil para tarefas de rotina. Isso proporciona controle total sobre o processo de automação, garantindo que as ações apropriadas sejam executadas quando necessário.

Um destaque é que o GitLab CI/CD oferece a capacidade de criar ambientes isolados para implantação e testes. Isso é essencial para cenários de desenvolvimento complexos, onde precisamos de um ambiente para desenvolvimento, homologação e produção. Isso simplifica a implantação e o teste de novos recursos e correções antes de atingir a produção, garantindo maior confiabilidade.

Portanto, podemos concluir que o GitLab CI/CD é uma solução abrangente de CI/CD que se integra perfeitamente com repositórios GitLab, permitindo a automação e o gerenciamento eficiente de fluxos de trabalho de desenvolvimento de software.

3. Trabalhos relacionados

A CI/CD é uma prática fundamental no desenvolvimento de software ágil e DevOps. As principais metodologias e ferramentas de CI/CD incluem:

Integração Contínua: O processo de integração de código novo ou alterado no repositório principal de código.

Entrega Contínua: O processo de entrega de código pronto para produção para o ambiente de produção.

Testes automatizados: O processo de execução de testes automatizados em cada nova versão de código.

Monitoramento: O processo de coleta e análise de dados de desempenho e disponibilidade do software.

Alertas: O processo de geração de notificações automáticas em caso de problemas de desempenho ou disponibilidade do software.

O Gitlab CI e o Github Actions são duas das ferramentas de CI/CD mais populares. Ambas as ferramentas oferecem um conjunto de recursos e funcionalidades que permitem automatizar o processo de CI/CD.

[Lima 2021] apresenta uma comparação entre as funcionalidades de CI e CD das ferramentas Jenkins™ e Gitlab™. De acordo com o exposto, as duas ferramentas oferecem funcionalidades semelhantes em termos de integração contínua e entrega contínua. No entanto, o Jenkins é considerado mais fácil de usar e mais extensível, enquanto o Gitlab é considerado mais barato e com melhor suporte técnico [Lima 2021].

O uso de CI/CD pode trazer diversos benefícios para o desenvolvimento de projetos Java Web, como: melhor qualidade do software, redução de custos, melhor produtividade e maior satisfação do cliente [Barbosa 2018].

Em relação a implementação de *pipelines* utilizando GitHub Actions os autores [Starling et al. 2022] investigam o impacto do uso da ferramenta em relação a *Code Quality* e *Code Review*, em repositórios *open-source* do GitHub. O trabalho conclui que o uso do Action de *Code Quality* e *Code Review* está associado a uma melhora na qualidade do código.

Seguindo a linha de análise da qualidade do código, O artigo "Garantia da Qualidade de Software com DevOps", de autoria de [Pizzaia and Malara 2022], discute a importância da garantia da qualidade de software (QA) no contexto de DevOps. Ambos apresentam uma abordagem prática para a garantia da qualidade de software com DevOps, que consiste em cinco etapas: Planejamento, desenvolvimento, execução, reportagem e avaliação.

Em relação a comparação de diferentes ferramentas, [Santos and Alcântara 2022] escreveram o artigo "Estudo e análise de *pipelines* CI/CD escaláveis de alta disponibilidade", o qual estuda e analisa as melhores práticas para a implementação de *pipelines* CI/CD escaláveis e de alta disponibilidade. Os autores esclarecem que é possível implementar *pipelines* CI/CD escaláveis e de alta disponibilidade utilizando as ferramentas e técnicas descritas no artigo.

Sendo assim, os trabalhos mencionados fornecem uma base sólida para o presente trabalho. A revisão de literatura fornece um entendimento do estado da arte da CI/CD e das ferramentas Gitlab CI e Github Actions, já os estudos de caso fornecem exemplos de como as ferramentas podem ser utilizadas em projetos reais enquanto o estudo de comparação entre Gitlab CI e Github Actions em projetos reais fornece uma visão geral das vantagens e desvantagens de cada ferramenta.

O presente trabalho irá contribuir para o estado da arte da CI/CD ao fornecer uma comparação mais abrangente entre as ferramentas Gitlab CI e Github Actions. O presente trabalho irá considerar um conjunto de critérios mais amplo, incluindo funcionalidades, vantagens e desvantagens, facilidade de uso e suporte técnico. O presente trabalho também irá utilizar uma metodologia de pesquisa que permita coletar dados de forma quantitativa e qualitativa.

4. Materiais e métodos

Nossa escolha de metodologia e ferramentas reflete o objetivo de avaliar e contrastar o desempenho, usabilidade e aceitação da comunidade de desenvolvedores em relação a essas duas soluções de CI/CD.

A escolha do Gitlab CI e Github Actions é justificada pela sua ampla utilização, comunidades ativas, integração nativa com Git, flexibilidade, constante inovação e sua relevância na indústria de desenvolvimento de software. Essas ferramentas são padrões do setor e oferecem a oportunidade de avaliar seu desempenho e aceitação na prática, tornando-as escolhas ideais para a nossa pesquisa comparativa de Integração Contínua e Implantação Contínua.

Neste momento, propomos a criação de duas pipelines a partir do mesmo projeto e seguindo com as mesmas etapas, sendo uma abordagem essencial para garantir uma comparação equitativa e confiável entre esses serviços. Isso assegura que ambas as ferramentas sejam avaliadas sob as mesmas condições, eliminando qualquer viés na análise. Ao padronizar o processo podemos aplicar parâmetros de comparação consistentes, como métricas de desempenho e tempo de execução, o que torna os resultados mais facilmente comparáveis.

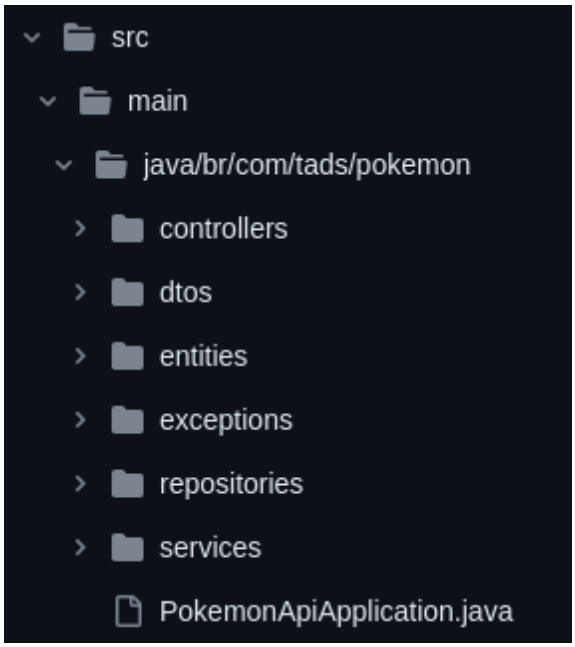

Figure 3. Arquivo de configuração YML para pipeline Gitlab CI

Seguindo essa premissa, foi decidido utilizar um projeto Maven com JDK 17 e utilizando *framework* Spring Boot. Como consta na Figura 3, trata-se de um projeto o

qual abrange uma implementação voltada para simulação de um sistema relacionado a treinadores e Pokémons, criando uma estrutura de dados abrangente com cinco entidades principais: Ability (Habilidade), Pokemon, Trainer (Treinador), Type (Tipo) e Weakness (Fraqueza). Cada uma dessas entidades é mapeada utilizando anotações JPA para persistência em um banco de dados.

O projeto envolve conceitos familiares para os fãs de Pokémon^{TM}, incluindo habilidades que podem possuir, tipos que os classificam, fraquezas que podem ser exploradas, treinadores e os próprios Pokémons, com uma série de atributos detalhados, como nome, altura, peso, saúde, ataque e defesa.

Em complemento a isso, o projeto segue as melhores práticas da programação Java, fazendo uso da ferramenta javax validation, do banco de dados em memória H2 para testes e da biblioteca Lombok para simplificar o código e melhorar a legibilidade.

Em seguida, no que diz respeito às etapas da esteira, é possível observar a partir da Figura 4 que optamos por estabelecer um fluxo uniforme para ambas as ferramentas, o qual permite um foco nas diferenças e na abordagem da solução de cada sistema de CI/CD. Dessa forma, a comparação é conduzida de forma transparente, objetiva e sólida, fornecendo uma base confiável para avaliar as capacidades de cada ferramenta no contexto do desenvolvimento de software.

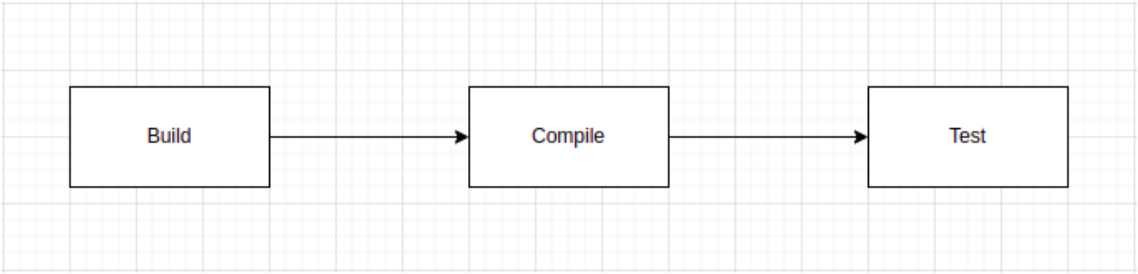

Figure 4. Fluxo das pipelines

***Build*:** A fase de build é o primeiro passo do processo de CI/CD, onde o código-fonte do projeto é preparado para compilação e empacotamento. Em ambas as ferramentas, esta fase começa com a configuração do ambiente de construção, que envolve a instalação de ferramentas e dependências necessárias. Isso garante que o ambiente esteja pronto para a compilação. Durante esta fase, o código-fonte é baixado do repositório e os artefatos de construção são gerados. Em um projeto Maven (Java), a compilação geralmente envolve a execução do comando mvn clean install, que compila o código-fonte e empacota o projeto em um formato utilizável. O resultado da fase de build é a criação de um artefato de construção, como um arquivo JAR, que contém o código compilado e pronto para ser testado e implantado.

***Compile*:** Após a fase de build, a fase de compilação entra em ação. Nesta etapa, o código-fonte compilado e os artefatos gerados na fase de build são preparados para a execução de testes. A fase de compilação não envolve a recompilação do código, uma vez que isso já foi feito na fase de build. Em vez disso, ela se concentra em configurar o ambiente de teste e garantir que as dependências e bibliotecas necessárias estejam disponíveis. No contexto de um projeto Java Maven, a fase de compile pode envolver a configuração do classpath e a organização das classes e recursos necessários para a execução dos testes.

***Test*:** A fase de teste é a etapa final das pipelines, onde os testes do projeto são executados. Isso inclui testes de unidade, testes de integração e outros tipos de testes definidos no projeto. Em ambas as ferramentas, os testes são executados em um ambiente controlado e isolado, garantindo que os resultados sejam consistentes e confiáveis. Qualquer falha nos testes é identificada e relatada. A fase de teste é fundamental para garantir a qualidade do código e identificar problemas antes que as alterações sejam implantadas em ambientes de produção.

É importante observar que, embora o fluxo das pipelines seja semelhante nas duas ferramentas, as configurações específicas e os comandos usados para cada fase podem variar dependendo das peculiaridades do Gitlab CI e Github Actions. No entanto, o objetivo geral dessas fases é o mesmo: preparar, compilar e testar o código-fonte de maneira automatizada e controlada. Em seguida iremos apresentar como as esteiras são configuradas em cada ferramenta.

4.1. GitLab CI

No que diz respeito à configuração da esteira para o GitLab CI, ela é manuseada utilizando estágios distintos. Ela começa com a etapa de "*build*" e, em seguida, passa para a etapa de "*test*" (Figura 5).

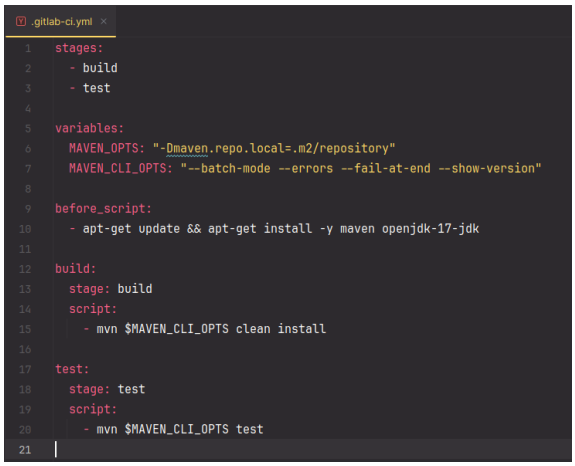

Figure 5. Arquivo de configuração YML para pipeline Gitlab CI

Na etapa de *build*, a *pipeline* configura o ambiente executando comandos para atualizar o sistema operacional e instalar o Maven e o OpenJDK 17. Isso prepara o ambiente para a compilação do projeto. O código-fonte do projeto Java, que inclui as entidades *Ability, Pokemon, Trainer, Type* e *Weakness*, é baixado do repositório Gitlab. O Maven é usado para compilar o código-fonte, aplicando o comando mvn clean install para limpar e construir o projeto. O resultado é a geração de artefatos de construção, como arquivos JAR.

Na etapa de *test*, a pipeline executa os testes do projeto Java para verificar a funcionalidade do sistema. Esses testes cobrem diversas partes do projeto, garantindo que ele funcione conforme o esperado. Métricas de cobertura de teste e resultados detalhados são coletados nesta fase.

As etapas da pipeline são executadas sequencialmente, com a etapa de *test* ocorrendo após a conclusão bem-sucedida da etapa de *build*.

4.2. GitHub Actions

Em relação a pipeline utilizando o GitHub Actions, esta é configurada como um fluxo de trabalho baseado em eventos, acionado quando ocorre um *push* no repositório como apresentado na Figura 6. No ambiente de execução, a pipeline usa a versão mais recente do Ubuntu (ubuntu-latest) como base.

A primeira etapa, chamada *Checkout Repository*, usa a ação *actions/checkout* para clonar o código-fonte do projeto a partir do repositório Github. Isso garante que o código mais recente seja disponibilizado para a pipeline.

pipeline.yml

```yaml
name: Maven CI/CD

on:
  push:
    branches:
      - '**'

jobs:
  build:
    name: Build and Test
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v3

    - name: Set up JDK 17
      uses:  actions/setup-java@v3
      with:
        java-version: 17
        distribution: 'adopt'

    - name: Build with Maven
      run: mvn clean install

    - name: Test with Maven
      run: mvn test
```

Figure 6. Arquivo de configuração YML para pipeline GitHub Actions

A etapa seguinte, chamada "Set up JDK 17," configura o ambiente Java™ 17 usando a ação "actions/setup-java."

Para a construção, a *pipeline* executa o comando "mvn -B *clean install*" para compilar o código-fonte e gerar os artefatos de construção. A opção "-B" indica que o Maven deve ser executado em modo não interativo.

Na etapa final, "*Test with Maven*", a pipeline executa os testes automatizados do projeto Java usando o comando "*mvn test*."

A priori, ambas as pipelines atingem os mesmos objetivos de compilação e teste do projeto Java, mas suas configurações e abordagens variam de acordo com a plataforma específica, seja usando stages no Gitlab CI ou *workflows* no Github Actions. As diferenças

refletem as particularidades de cada plataforma e suas respectivas maneiras de configurar e executar pipelines.

4.3. Pesquisa de campo

Em segunda análise, esse trabalho se propõe a realizar uma pesquisa de campo disponibilizada pelo formulário Google Forms. A mesma será essencial para obter informações valiosas da comunidade de TI, o que possibilitará avaliar o uso e a satisfação com as ferramentas em questão. Abaixo, apresentamos os principais aspectos da pesquisa de campo:

1. **Qual é o seu papel ou função atual na área de desenvolvimento de software?** Opções: Desenvolvimento, QA, Infraestrutura, Arquitetura, Dados, Produto, Gerência.
2. **Qual é o tamanho da sua organização?** Opções: Pequena (até 50 funcionários), Média (51-500 funcionários), Grande (mais de 500 funcionários), Não atuo no momento, Outro.
3. **Você já utilizou o GitHub Actions para automação de CI/CD?** Opções: Sim, Não.
4. **Você já utilizou o GitLab CI/CD para automação de CI/CD?** Opções: Sim, Não.
5. **Qual é o nível de satisfação com a ferramenta que você utiliza atualmente?** Opções: Muito satisfeito, Satisfeito, Neutro, Insatisfeito, Muito insatisfeito, N/A.
6. **Quais são os principais benefícios que você encontra na ferramenta que utiliza?** Opções: Facilidade de uso, Integração com outras ferramentas, Desempenho, Documentação, Suporte da comunidade, N/A, Outro.
7. **Quais são os principais desafios ou limitações que você enfrentou ao usar a ferramenta que utiliza?** Opções: Complexidade, Falta de recursos específicos, Problemas de desempenho, Falta de suporte, N/A, Outro.
8. **Em sua opinião, qual ferramenta oferece uma melhor integração com outras ferramentas ou serviços?** Opções: GitHub Actions, GitLab CI/CD, Não sei/não tenho certeza.
9. **Com base em sua experiência, qual das duas ferramentas você recomendaria para outras equipes ou organizações?** Opções: GitHub Actions, GitLab CI/CD, Outro.

Sendo assim, o público-alvo da pesquisa de campo inclui profissionais da área de desenvolvimento de software, abrangendo diversas funções, desde desenvolvedores e engenheiros de qualidade (QA) até arquitetos, gerentes e outros envolvidos em processos de CI/CD. A pesquisa visa obter informações a partir de diferentes perspectivas dentro da comunidade de desenvolvimento de software.

Ademais, espera-se que a pesquisa forneça uma visão abrangente sobre a utilização, satisfação, benefícios e desafios associados ao uso do Gitlab CI e do Github Actions. Além das expectativas, preferências e recomendações da comunidade em relação a essas ferramentas.

A princípio, a consulta é crucial para este trabalho, pois fornece dados concretos e opiniões reais da comunidade de desenvolvimento. Essas informações são valiosas para a avaliação das ferramentas em um contexto geral, ajudando a fornecer dados úteis para a comparação de ambos serviços. Além disso, a pesquisa ajuda a identificar tendências,

pontos fortes e áreas de melhoria nas ferramentas, beneficiando equipes e organizações que desejam adotar práticas de CI/CD eficazes. Sendo assim, uma pesquisa de campo será um componente essencial para o desenrolar dos resultados e da conclusão deste estudo.

5. Resultados

Neste capítulo, apresentaremos os resultados obtidos a partir da comparação das ferramentas de CI/CD GitLab CI e Github Actions, tal como os resultados da pesquisa de satisfação da comunidade a respeito dessas plataformas. Os resultados são fundamentais para avaliar o desempenho, a eficácia e as características de ambas as ferramentas em um contexto de desenvolvimento de software. Os dados serão apresentados na forma de figuras, gráficos e tabelas para proporcionar uma visão clara das diferenças observadas.

5.1. Desempenho das pipelines

Nesta seção, analisaremos o desempenho das pipelines configuradas para o Gitlab CI e o Github Actions. Os resultados são baseados na execução das esteiras utilizando o mesmo projeto Maven com etapas equivalentes para cada ferramenta.

5.2. Gitlab CI

De maneira geral, a compilação completa da esteira executada na plataforma GitLab CI demorou exatamente 2 minutos e 47 segundos, como demonstrado na Figura 7.

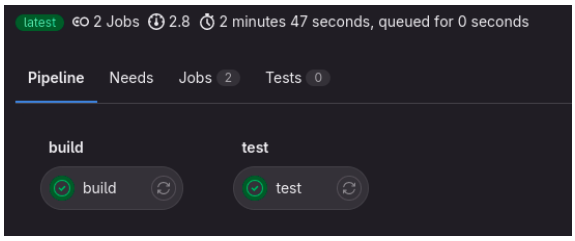

Figure 7. Resultado da pipeline com GitLab CI

Em primeira análise, a etapa de build a qual é executada em primeiro lugar teve um tempo de duração de 1 minuto e 26 segundos.

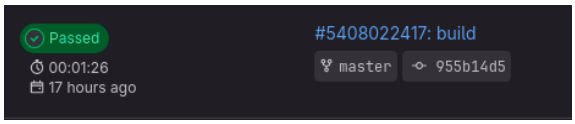

Figure 8. Tempo de duração da etapa build com GitLab CI

Ademais, acessando os logs de execução deste job é possível identificar a duração de cada passo que o GitLab realiza para a execução de um estágio de pipeline.

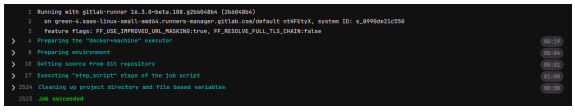

Figure 9. Log de execução do estágio de build com GitLab CI

De acordo com a Figura 9, podemos identificar que o GitLab realiza três fases preparatórias para de fato executar o comando de build definido no script de execução da pipeline. Essas etapas consistem na construção e configuração do ambiente de execução da etapa em questão, desde configurar o ambiente do docker até a cópia do repositório dentro da máquina virtual, toda essa etapa teve duração de cerca de 50 a 60 segundos.

Somado a isso, é possível analisar os logs de execução do comando de build e podemos observar que a execução do build teve uma duração de cerca de 33 segundos, como aponta a Figura 10.

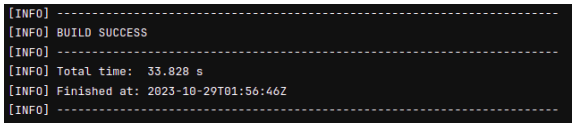

Figure 10. Resultado da execução do comando build com GitLab CI

Em seguida a etapa de test compilou em uma duração aproximada da etapa anterior, como apresenta a Figura 11 esta etapa demorou 1 minuto e 21 segundos para sua execução.

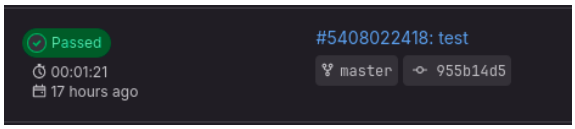

Figure 11. Tempo de duração da etapa test com GitLab CI

Sendo assim, após as fases de criação e configuração de ambiente que demoram cerca de 1 minuto, é possível identificar que a execução dos testes tem duração de cerca de 27 segundos, como demonstra a Figura a seguir.

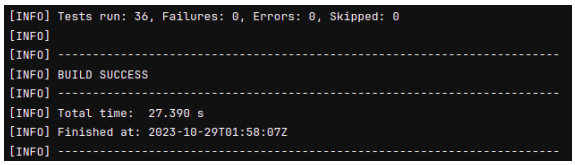

Figure 12. Resultado da execução da etapa de test com GitLab CI

5.3. Github Actions

Em relação a execução da esteira utilizando o GitHub Actions, a qual foi configurada com um único workflow de Build e Test, é possível observar pela Figura 13 que sua duração foi de 1 minuto e 43 segundos.

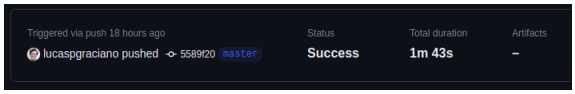

Figure 13. Resultado da pipeline com GitHub Actions

De maneira geral, como indica a Figura 14 o workflow realiza uma preparação do executor, configurando o ambiente em que será executado os comandos com o JDK

17 definido no arquivo de configuração e clonando o repositório. Essa preparação de ambiente demorou cerca de 9 segundos de duração.

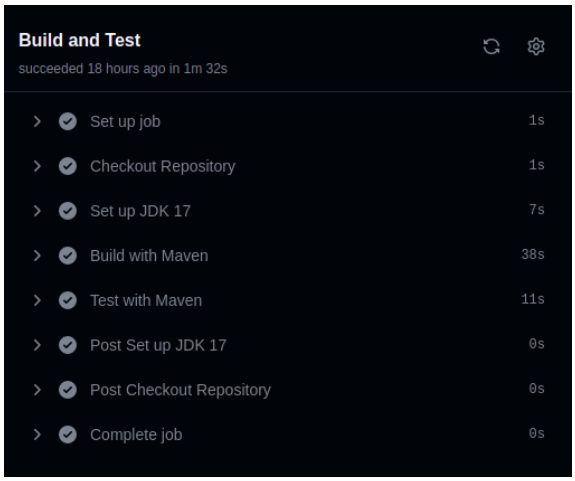

Figure 14. Detalhes do workflow Build and Test executados com GitHub Actions

No que diz respeito a execução do comando de instalação de dependências e empacotamento (Build) teve um tempo de duração de aproximadamente 36 segundos de acordo com a Figura 15.

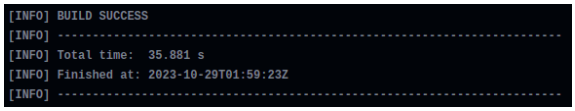

Figure 15. Resultado de execução da etapa de build com GitHub Actions

Em seguida, a etapa de execução de testes do projeto teve uma duração de apenas 9 segundos, como indica a Figura 16.

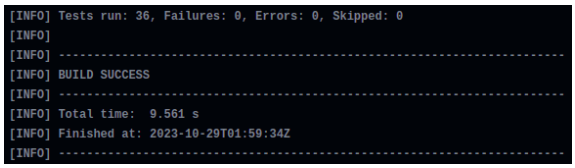

Figure 16. Resultado de execução da etapa de test com GitHub Actions

Sendo assim, a Tabela 1 apresenta uma visão geral das diferenças de tempo entre as duas execuções. O GitHub Actions teve um tempo total de execução mais curto, principalmente devido à rápida configuração de ambiente. O GitLab CI foi ligeiramente mais rápido na fase de compilação, enquanto o GitHub Actions se destacou na execução dos testes, economizando consideravelmente mais tempo nessa etapa. As diferenças podem ser relevantes dependendo das necessidades específicas do projeto.

5.4. Pesquisa de campo

A seguir, apresentaremos os resultados da pesquisa de campo realizada ao longo de duas semanas, que obteve 50 respostas, por meio do uso da plataforma Google Forms. O obje-

	GitLab CI	GitHub Actions
Configuração do ambiente	106 segundos	9 segundos
Build	34 segundos	36 segundos
Test	27 segundos	9 segundos

Table 1. Tempos de execução de cada etapa das pipelines

tivo da pesquisa foi avaliar as percepções e experiências dos usuários em relação às ferramentas de Integração Contínua e Implantação Contínua (CI/CD) Gitlab CI e GitHub Actions. A pesquisa envolveu participantes com diversos perfis e funções na área de desenvolvimento de software, sendo a maioria dos participantes do estado do Mato Grosso do Sul e São Paulo e coletou dados significativos que permitiram uma análise abrangente das percepções dos usuários em relação às ferramentas. Abaixo, destacamos os resultados-chave da pesquisa:

5.4.1. Qual é o seu papel ou função atual na área de desenvolvimento de software?

A maioria dos participantes desempenha funções relacionadas ao desenvolvimento de software, com algumas exceções em arquitetura, dados, gerência e outras áreas.

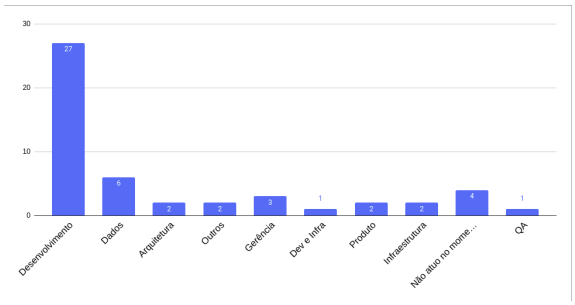

Figure 17. Resultado da questão 1

De acordo com a Figura 17, a quantidade mais significativa de uso de CI/CD reside no domínio de desenvolvimento seguido pela área de dados. Isso mostra a importância da automatização para os profissionais que atuam diretamente com o desenvolvimento de software.

5.4.2. Qual é o tamanho da sua organização?

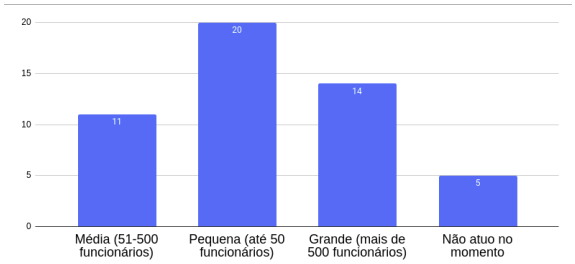

Figure 18. Resultados da questão 2

A pesquisa incluiu uma variedade de tamanhos de organizações, desde pequenas até grandes, com uma presença notável em organizações de pequeno porte.

5.4.3. Você já utilizou o GitHub Actions para automação de CI/CD?

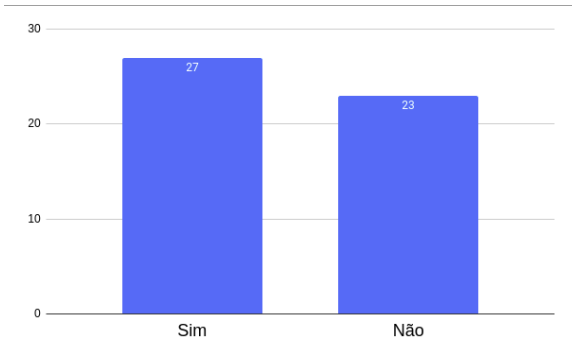

Figure 19. Resultados da questão 3

A partir do resultado da Figura 19 podemos concluir que a popularidade do GitHub Actions é grande, visto que esta ferramenta possui uma comunidade altamente ativa.

5.4.4. Você já utilizou o GitLab CI/CD para automação de CI/CD?

Os resultados revelam que metade das pessoas têm experiência com o GitLab CI/CD. Sendo assim, a Figura 20 apresenta detalhes de uma popularidade menor da ferramenta em relação a sua concorrente (Figura 19).

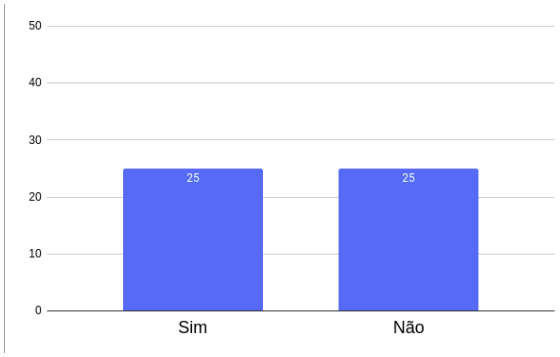

Figure 20. Resultados da questão 4

5.4.5. Qual é o nível de satisfação com a ferramenta que você utiliza atualmente?

A maioria dos participantes expressou satisfação com a ferramenta que utilizam atualmente, com níveis variados de satisfação. Conforme aponta a Figura 21, os entrevistados estão satisfeitos com as soluções de CI/CD fornecidas pelo GitHub Actions e GitLab Actions, posto isso é notável a qualidade da ferramenta e a satisfação da comunidade de desenvolvimento de software a respeito dessas soluções de Integração de código.

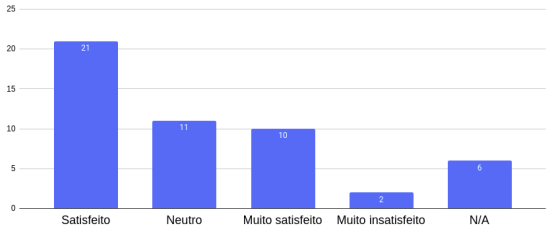

Figure 21. Resultados da questão 5

5.4.6. Quais são os principais benefícios que você encontra na ferramenta que utiliza?

Facilidade de uso foi citada como um benefício significativo, juntamente com integração com outras ferramentas, desempenho, documentação e suporte da comunidade.

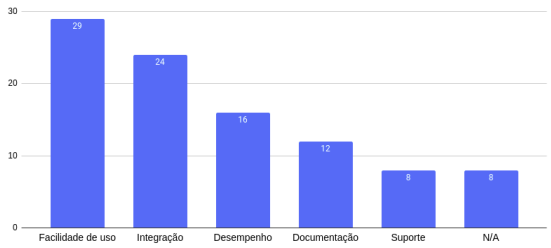

Figure 22. Resultados da questão 6

Desta forma, as respostas do Figura 22 só reforçam a satisfação que os entrevistados possuem com relação ao uso das ferramentas de CI/CD abordadas nesse estudo.

5.4.7. Quais são os principais desafios ou limitações que você enfrentou ao usar a ferramenta que utiliza?

Segundo a Figura 23, os desafios incluíram complexidade, falta de recursos específicos, problemas de desempenho e falta de suporte, sendo a complexidade a resposta mais recorrente dos entrevistados. Sendo assim, deram uma pequena preferência para o GitHub Actions, com algumas pessoas indicando que não tinham certeza.A partir dessa informação, é possível identificar uma preferência dos entrevistados pela solução proposta pelo GitHub Actions.

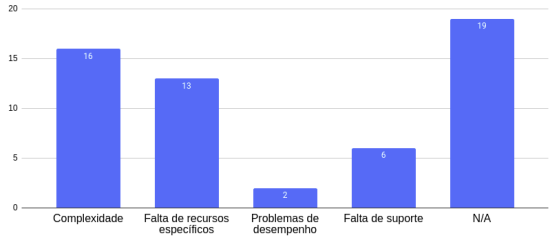

Figure 23. Resultados da questão 7

5.4.8. Em sua opinião, qual ferramenta oferece uma melhor integração com outras ferramentas ou serviços?

As opiniões deram uma pequena preferência para o GitHub Actions, com algumas pessoas indicando que não tinham certeza. A partir da Figura 24, é possível identificar uma preferência dos entrevistados pela solução proposta pelo GitHub Actions.

5.4.9. Com base em sua experiência, qual das duas ferramentas você recomendaria para outras equipes ou organizações?

De acordo com o Figura 25, os entrevistados demonstraram uma preferência significativa pelo GitHub Actions, com 34 recomendações, em contraste com 16 para o GitLab CI/CD. Esse dado enfatiza a forte preferência dos participantes por usar o GitHub Actions.

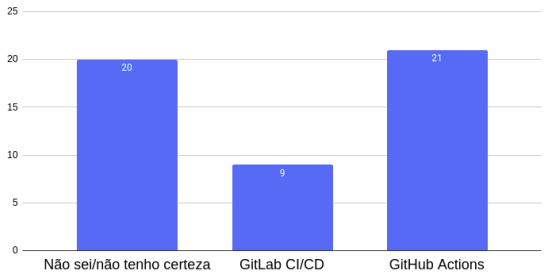

Figure 24. Resultados da questão 8

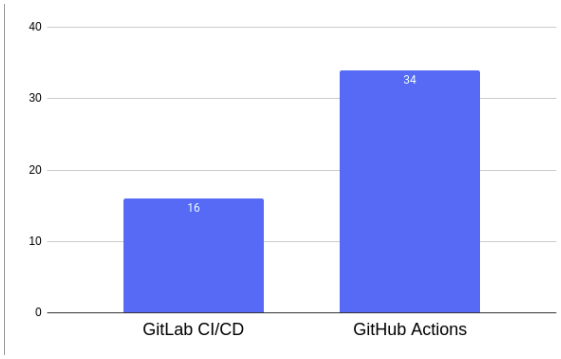

Figure 25. Resultados da questão 9

A pesquisa abrangeu opiniões variadas sobre o GitHub Actions e o GitLab CI/CD, destacando preferências e desafios. No geral, o GitHub Actions foi fortemente preferido devido à facilidade de uso, desempenho e integração com outras ferramentas. Apesar de alguns participantes apontarem desafios, como complexidade e falta de suporte, é notável uma inclinação para o GitHub Actions. Essa preferência evidente destaca a popularidade do GitHub Actions entre os profissionais de desenvolvimento, fornecendo resultados valiosos para as conclusões finais do estudo.

6. Conclusão

Este estudo se concentrou na análise comparativa das plataformas de Integração Contínua GitHub Actions e GitLab CI, com o objetivo de fornecer informações valiosas para equipes de desenvolvimento de software na escolha da plataforma mais adequada às suas necessidades. A Integração Contínua desempenha um papel crucial na engenharia de software moderna, garantindo a qualidade do código e facilitando lançamentos frequentes e confiáveis.

A pesquisa de campo, que envolveu a coleta de opiniões de profissionais da área, revelou uma preferência notável pelo GitHub Actions, com 34 recomendações em comparação com 16 para o GitLab CI. Os principais pontos positivos destacados para o GitHub Actions incluíram facilidade de uso, desempenho, integração com outras ferramentas, documentação e suporte da comunidade. No entanto, alguns participantes também mencionaram que a curva de aprendizado pode ser um pouco mais longa em comparação com o GitLab CI.

Os resultados obtidos nas execuções das pipelines mostraram que o GitHub Ac-

tions foi mais eficiente em termos de tempo, concluindo as tarefas em 1 minuto e 43 segundos, em comparação com o GitLab CI, que levou 2 minutos e 47 segundos. Essa diferença de tempo pode ser significativa em ambientes de desenvolvimento que buscam otimizar seus processos.

	GitLab CI	GitHub Actions
Tempo de execução	167 segundos	103 segundos
Já utilizaram	25	27
Recomendaram	16	34

Table 2. Resultados

No entanto, o estudo também identificou que a escolha entre essas plataformas pode depender das necessidades e preferências específicas da equipe. Ambas as plataformas são competitivas e continuamente aprimoram suas funcionalidades, o que significa que a escolha pode variar dependendo do contexto do projeto.

Os próximos passos incluem a continuação da observação das tendências no mercado de Integração Contínua, à medida que novas atualizações e recursos são lançados pelas plataformas. Além disso, a expansão do estudo para incluir outras plataformas de CI/CD pode oferecer uma visão mais abrangente das opções disponíveis para equipes de desenvolvimento. Em última análise, este trabalho serve como um guia valioso para equipes que buscam otimizar sua prática de Integração Contínua, promovendo eficiência, qualidade e entrega de software mais rápida e confiável.

References

Augustino, L. (2023). Benefícios da integração contínua no desenvolvimento de software. `https://www.monitoratec.com.br/blog/integracao-continua/`. Acessado em: 28-09-2023.

Barbosa, H. A. B. (2018). Análise das vantagens do uso de ci/cd no desenvolvimento de um projeto java web.

Fitzgerald, B., Stol, K.-J., and O'Sullivan, D. (2020). Continuous software engineering: A roadmap and agenda. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*.

GitHub (2023a). Github action documentation. `https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions`. Acessado em: 09-10-2023.

GitHub (2023b). Github resources: Automation with github actions. `https://resources.github.com/devops/tools/automation/actions/`. Acessado em: 11-10-2023.

GitLab (2023a). Gitlab ci/cd documentation. `https://docs.gitlab.com/ee/ci/`. Acessado em: 13-10-2023.

GitLab (2023b). Gitlab devops lifecycle - continuous integration. `https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/`. Acessado em: 13-10-2023.

Hat, R. (2023). Introdução a devops. https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd. Acessado em: 28-09-2023.

Institute, D. (2023). The continuous delivery playbook. https://www.devopsinstitute.com/a-continuous-delivery-playbook/. Acessado em: 28-09-2023.

Lima, P. H. O. (2021). Estudo comparativo sobre as funcionalidades de integração contínua e entrega contínua das ferramentas jenkins e gitlab.

Paul, D., Steve, M., and Andrew, G. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional.

Pizzaia, V. H. and Malara, R. D. (2022). Garantia da qualidade de software com devops. RECIMA21-Revista Científica Multidisciplinar.

Rehkopf, M. (2023). What is continuous integration? https://www.atlassian.com/continuous-delivery/continuous-integration. Acessado em: 28-09-2023.

Santos, M. C. and Alcântara, R. M. d. (2022). Estudo e análise de pipelines ci/cd escaláveis de alta disponibilidade.

Starling, P. H. C., d Santos, V. A., and Gerais, M. (2022). Github actions: Uma análise do impacto de actions de code quality e code review em repositórios open-source do github.