Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Implementação de um pipeline CI/CD para o Projeto SisGera

Rafael Nepomuceno Siqueira Campos

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Euler Horta Marinho

COORIENTAÇÃO:
Silvandro Sergio Martins Oliveira

Agosto, 2025
João Monlevade–MG

Rafael Nepomuceno Siqueira Campos

Implementação de um pipeline CI/CD para o Projeto SisGera

Orientador: Euler Horta Marinho

Coorientador: Silvandro Sergio Martins Oliveira

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Agosto de 2025

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C198i Campos, Rafael Nepomuceno Siqueira.
Implementação de um pipeline CI/CD para o projeto SisGera. [manuscrito] / Rafael Nepomuceno Siqueira Campos. - 2025.
83 f.: il.: color., tab..

Orientador: Prof. Dr. Euler Horta Marinho.
Coorientador: Silvandro Sergio Martins Oliveira.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Sistemas de Informação .

1. Análise de sistemas. 2. Engenharia de software. 3. GitHub (Programa de computador). 4. Software - Confiabilidade. 5. Software - Desenvolvimento. 6. Software - Testes. I. Marinho, Euler Horta. II. Oliveira, Silvandro Sergio Martins. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004.41

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Rafael Nepomuceno Siqueira Campos

Implementação de um Pipeline CI/CD para o Projeto Sisgera

Monografia apresentada ao Curso de Sistemas de Informação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação

Aprovada em 28 de agosto de 2025

Membros da banca

Dr. Euler Horta Marinho - Orientador - Universidade Federal de Ouro Preto
Silvandro Sergio Martins de Oliveira - Coorientador - Autoglass
Dr. Fernando Bernardes de Oliveira - Universidade Federal de Ouro Preto
Dr. Igor Muzetti Pereira - Universidade Federal de Ouro Preto

Euler Horta Marinho, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 09/09/2025

Documento assinado eletronicamente por **Euler Horta Marinho**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 09/09/2025, às 13:35, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0975664** e o código CRC **AA5E2175**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Dedico este trabalho à minha família, pelo amor incondicional e incentivo constante, aos meus amigos, por cada palavra de apoio e companhia nos momentos difíceis, e aos meus professores, cujos ensinamentos foram essenciais para minha trajetória até aqui.

Agradecimentos

A conclusão desta jornada acadêmica só foi possível graças ao apoio, incentivo e amor que recebi das pessoas essenciais em minha vida.

Aos meus pais, meus pilares, expresso minha mais profunda gratidão. Seus ensinamentos, valores e, sobretudo, seu sacrifício e confiança incondicional, foram os alicerces que me permitiram chegar até aqui e moldaram quem eu sou.

Ao meu irmão, um agradecimento especial por sua companhia, por compartilhar sua sabedoria e por ser meu porto seguro nos momentos de maior desafio. Sua presença foi fundamental.

Aos meus amigos, pela parceria e lealdade constantes. Vocês tornaram a caminhada mais leve, compartilhando não apenas as alegrias, mas também oferecendo ombro e ânimo nas dificuldades. A jornada acadêmica não teria sido a mesma sem nossas conversas, risadas e apoio mútuo.

À Giovanna, cuja presença foi parte essencial desta etapa. Agradeço por sua compreensão e paciência nos momentos em que precisei estar ausente, pelo incentivo constante e pelo carinho que sempre me trouxe equilíbrio e força.

Aos meus mestres e professores, meu reconhecimento pela dedicação em transmitir seus valiosos conhecimentos, que foram cruciais para expandir meus horizontes intelectuais e me preparar para os desafios da vida acadêmica e profissional.

Aos meus orientadores, Euler e Silvandro, minha sincera admiração e gratidão. Agradeço pela paciência, pela excelência na orientação, pela amizade e pela oportunidade de contribuir com um projeto tão incrível. O rigor acadêmico e o incentivo de vocês foram determinantes para o meu crescimento.

A cada um de vocês, que desempenhou um papel único e insubstituível nesta trajetória, meu muito obrigado.

"I'm not a prophet or a stone age man
Just a mortal with potential of a superman"

— David Bowie (1947 – 2016)

Resumo

Este trabalho apresenta a implementação de um *pipeline* de Integração Contínua / Implantação Contínua (CI/CD) para o Sistema de Gerenciamento e Registro de Atividades (SisGera), uma aplicação utilizada por corporações de bombeiros voluntários em Minas Gerais. O objetivo principal foi modernizar o processo de entrega de software, anteriormente realizado de maneira manual, propenso a falhas e com risco de indisponibilidade. Após a análise do sistema e a criação de um ambiente de testes com infraestrutura semelhante à de produção, adotou-se a ferramenta GitHub Actions, devido à sua integração nativa com o repositório do SisGera, à gratuidade e à facilidade de uso. A estratégia de implantação escolhida foi o *Atomic Deployment*, garantindo confiabilidade e capacidade de reversão em caso de falhas. A *pipeline* foi dividida em três etapas: Integração Contínua (CI), Implantação Contínua (CD) e *rollback*. Além disso, o projeto incluiu a reformulação da documentação do SisGera, introduzindo um histórico de contribuições passadas, uma descrição geral do sistema, o modelo de gerenciamento de *branches* utilizado, a recomendação de práticas como a adoção do *Conventional Commits* e orientações para a preparação do ambiente de desenvolvimento local.

Palavras-chaves: SisGera. DevOps. Integração Contínua. Implantação Contínua.

Abstract

This work presents the implementation of a Continuous Integration / Continuous Deployment (CI/CD) pipeline for the SisGera, an application used by volunteer firefighter organizations in the state of Minas Gerais, Brazil. The main objective was to modernize the software delivery process, which was previously carried out manually, being prone to errors and with a risk of unavailability. After analyzing the system and creating a test environment with infrastructure similar to the production one, the GitHub Actions tool was adopted due to its native integration with the SisGera repository, its zero cost, and ease of use. The chosen deployment strategy was Atomic Deployment, ensuring reliability and rollback capability in case of failure. The pipeline was divided into three stages: Continuous Integration (CI), Continuous Deployment (CD), and rollback. Furthermore, the project included the restructuring of the SisGera documentation, introducing a history of past contributions, a general system description, the branch management model adopted, the recommendation of practices such as the use of Conventional Commits, and guidance for preparing the local development environment.

Key-words: SisGera. DevOps. Continuous Integration. Continuous Deployment.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AWS *Amazon Web Services*

AWP Aplicação Web Progressiva

CD Implantação Contínua

CI Integração Contínua

CI/CD Integração Contínua / Implantação Contínua

SaaS *Software as a Service*

SCP *Secure Copy Protocol*

SisGera Sistema de Gerenciamento e Registro de Atividades

SQL *Structured Query Language*

SSH *Secure Shell*

TBD *Trunk-Based Development*

WHM *Web Host Manager*

YAML *YAML Ain't Markup Language*

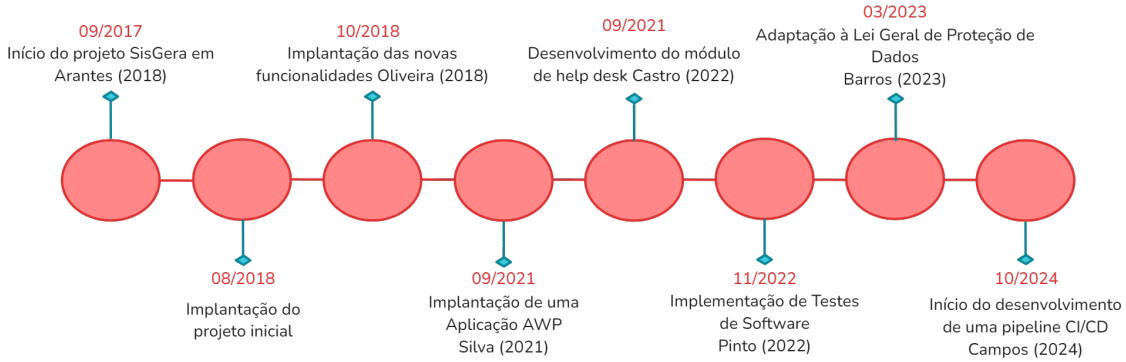
Sumário

1 Introdução

O Sistema de Gerenciamento e Registro de Atividades (SisGera) foi idealizado em 2018, durante o trabalho de conclusão de curso de Arantes (2018), que desenvolveu um sistema para o registro de ocorrências de bombeiros voluntários nas cidades de São Domingos do Prata e Barão de Cocais. Atualmente, o SisGera é utilizado em diversas cidades de Minas Gerais, como São Domingos do Prata, Barão de Cocais, Nova Era, Santa Bárbara do Leste e Cláudio. Esse projeto inicial solucionou a necessidade de substituir os registros manuais, reduzindo erros e tornando o processo mais eficiente. Desde então, o SisGera tem sido continuamente aprimorado por meio de novos trabalhos acadêmicos, cada um adicionando funcionalidades e expandindo seu escopo de aplicação.

Em sequência ao trabalho de Arantes (2018), Oliveira (2018) incorporou ao SisGera um módulo de controle administrativo, com foco na gestão de materiais e doações, atendendo às demandas organizacionais das corporações de bombeiros. Posteriormente, Silva (2021) desenvolveu uma Aplicação Web Progressiva (AWP) para o sistema, permitindo o registro de ocorrências offline, uma funcionalidade essencial para regiões com conectividade limitada. Castro (2022) introduziu um módulo de *help desk*, que facilitou a comunicação entre usuários do sistema e sua equipe de desenvolvimento, promovendo maior suporte técnico e eficiência operacional. Em continuidade, Pinto (2022) focou na melhoria do desempenho e segurança do SisGera, implementando testes de software de maneira sistemática e garantindo a preservação dos dados para futuras manutenções. Mais recentemente, Barros (2023) adaptou o sistema aos padrões da Lei Geral de Proteção de Dados (LGPD), adicionando funcionalidades de anonimização, exclusão de dados e melhorias nas políticas de privacidade, aumentando a transparência e segurança no uso dos dados coletados. A Figura 1 ilustra o acompanhamento da evolução do SisGera ao longo dos anos até o seu estado presente.

Figura 1 – Linha do tempo do projeto SisGera.



Fonte: Elaborado pelo autor.

1.1 Problema

O SisGera foi concebido como uma solução para apoiar as atividades administrativas e operacionais de corporações de bombeiros voluntários. Desde sua criação, em 2018, o sistema tem sido continuamente aprimorado, atendendo às demandas específicas de seus usuários. No entanto, o processo de desenvolvimento e implantação do SisGera ainda enfrenta desafios críticos relacionados à integração de novas funcionalidades e à entrega de atualizações de modo ágil e confiável.

Atualmente, o fluxo de trabalho do SisGera é caracterizado pela integração manual de código e pela implantação manual das atualizações no ambiente de produção e homologação. Esse processo é suscetível a erros humanos, limita a frequência de atualizações e aumenta o risco de falhas durante as implantações. Além disso, a ausência de uma automação consistente dificulta a aplicação de práticas modernas, como execução sistemática de testes automatizados e validação contínua do código, elementos essenciais para a entrega de software de alta qualidade.

Um dos problemas mais significativos é a demora gerada pelo processo manual de implantação. Durante esse intervalo, o sistema pode se tornar temporariamente indisponível, afetando diretamente a experiência dos usuários finais. No caso das corporações de bombeiros voluntários que dependem do SisGera para registrar ocorrências e gerenciar recursos em tempo real, qualquer indisponibilidade pode comprometer a captura de informações críticas, prejudicar a continuidade das operações e reduzir a eficiência no atendimento à comunidade.

Outro fator relevante é a necessidade de alinhar as práticas de desenvolvimento

do SisGera a soluções acessíveis e de baixo custo, considerando o perfil das entidades que o utilizam. A escolha da solução ideal requer uma análise detalhada que considere não apenas os custos diretos, mas também a facilidade de configuração, compatibilidade com a infraestrutura existente e os benefícios oferecidos para automação de processos.

Portanto, o problema central deste trabalho está na inexistência de um *pipeline* de CI e CD para o SisGera. A implementação desse *pipeline* permitirá uma integração mais ágil e confiável de novas funcionalidades, automatizando processos e minimizando os riscos associados a atualizações e implantações. Além disso, promoverá a manutenibilidade do sistema a longo prazo, garantindo que continue a atender às necessidades de seus usuários de maneira eficiente e segura.

1.2 Objetivos

Este trabalho busca solucionar os desafios relacionados à integração e entrega de software no SisGera por meio da implementação de um *pipeline* de CI/CD. Para isso, são definidos o objetivo geral e objetivos específicos, descritos nessa seção.

1.2.1 Objetivo Geral

O objetivo principal deste trabalho é implementar um *pipeline* de CI/CD para o SisGera.

1.2.2 Objetivos Específicos

O presente trabalho tem como objetivos específicos os seguintes tópicos:

1. Analisar e selecionar ferramentas de CI/CD apropriadas para o SisGera.
2. Automatizar a execução de testes unitários no *pipeline* CI/CD.
3. Documentar detalhadamente o processo de configuração e uso do *pipeline*.
4. Reduzir o impacto da implantação de atualizações sobre os usuários do sistema.

1.3 Organização do trabalho

Para o Capítulo 2 é reunida a fundamentação teórica, abordando conceitos de *DevOps*, práticas de CI/CD, estratégias de implantação e ferramentas de CI/CD. No

Capítulo 3 é descrito o processo de implementação do *pipeline* CI/CD, contemplando a análise do sistema, a configuração do ambiente de testes, a definição da estratégia de implantação, bem como a documentação e guia de uso elaborados, além das dificuldades ao longo do desenvolvimento. O Capítulo 4 apresenta os resultados obtidos com a implementação, incluindo evidências do funcionamento do *pipeline*, benefícios alcançados e desafios identificados. Por fim, o Capítulo 5 apresenta as considerações finais e propostas para trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais para a compreensão deste trabalho. Inicialmente, na Seção 2.1, são discutidos os principais conceitos relacionados ao DevOps. Em seguida, a Seção 2.2 introduz formalmente o conceito de CI, enquanto a Seção 2.3 aborda a Entrega e Implantação Contínuas (CD). As estratégias de implantação comumente adotadas na indústria são discutidas na Subseção 2.3.1, e as ferramentas de CI/CD mais utilizadas no mercado são apresentadas na Seção 2.4. Por fim, a Seção 2.5 traz as considerações finais deste capítulo.

2.1 DevOps

O movimento ágil impulsionou a necessidade de aplicar seus princípios também ao ambiente de infraestrutura. Essa demanda levou à convergência entre os âmbitos operacional, de planejamento e de desenvolvimento. Em 2008, Patrick Debois publicou o artigo "*Agile Infrastructure and Operations: How Infra-gile Are You* ?", no qual identificou três padrões fundamentais para integrar o ágil à infraestrutura: padrões técnicos, de projeto e operacionais. Debois expôs que a infraestrutura poderia atuar de maneira reativa e adaptativa às mudanças do negócio, além de promover maior aproximação entre os setores e a aplicação dos princípios ágeis às operações (DEBOIS, 2008).

Em 2009, a palestra intitulada "*10+ Deploys Per Day: Dev and Ops Cooperation at Flickr*", apresentada por John Allspaw e Paul Hammond na conferência Velocity, evidenciou como a colaboração eficaz entre os times de desenvolvimento e operações era essencial para alcançar entregas rápidas e confiáveis. Inspirado pela sinergia sugerida no case do Flickr, Debois organizou, ainda em 2009, a primeira conferência DevOpsDays, na Bélgica (SMITH, 2024). O evento não apenas reuniu profissionais interessados na integração entre desenvolvimento e operações, como também foi o ponto em que o termo DevOps surgiu, da junção de "*Development*" e "*Operations*".

Com o passar dos anos, o DevOps consolidou-se como uma abordagem que busca romper barreiras entre desenvolvimento (Dev) e operações (Ops), promovendo uma cultura de colaboração contínua, automação de processos e ciclos rápidos de entrega de software. Segundo Azad e Hyrynsalmi (2024), o DevOps pode ser entendido como um ciclo contínuo composto por sete fases integradas: planejamento, desenvolvimento,

construção, testes, implantação, operação e monitoramento, as quais se retroalimentam em busca de maior eficiência e confiabilidade no desenvolvimento de software (AZAD; HYRYNSALMI, 2024).

Ainda segundo Azad e Hyrynsalmi (2024), como parte fundamental de suas práticas, o DevOps se apoia em três componentes principais: Integração Contínua (*Continuous Integration*), Entrega Contínua (*Continuous Delivery*) e Implantação Contínua (*Continuous Deployment*).

- Integração Contínua: É o processo que automatiza a integração de código de múltiplos desenvolvedores em um repositório central.
- Entrega Contínua: É a abordagem que estende a CI, garantindo que o software possa ser liberado para o ambiente de produção a qualquer momento, de modo confiável e em ciclos curtos.
- Implantação Contínua: Representa um passo adiante, no qual cada alteração de código que passa pelos testes é implantada automaticamente na produção, permitindo que os clientes utilizem o resultado final de modo imediato.

Além dos aspectos técnicos, a adoção bem-sucedida do DevOps depende fortemente de fatores organizacionais. A cultura DevOps valoriza a colaboração entre equipes, a eliminação de silos, a confiança mútua e o aprendizado contínuo (KIM et al., 2021). Ele não se limita a um conjunto de práticas ou ferramentas, mas representa uma transformação cultural profunda que redefine como equipes de tecnologia colaboram, tomam decisões e entregam valor ao negócio.

2.2 Integração Contínua (CI)

A Integração Contínua é uma das práticas fundamentais do DevOps e refere-se ao processo de integrar as mudanças de código de modo contínuo em um repositório compartilhado. O objetivo da CI é identificar rapidamente erros e conflitos, reduzir o tempo de ciclo de desenvolvimento e garantir que o código esteja sempre em um estado utilizável. Isso contrasta com a prática de desenvolvedores trabalharem em *branches* independentes, apenas realizando a integração de seu código quando ele estiver completamente finalizado. Longos períodos de tempo entre as integrações significam que muitas mudanças podem ter sido feitas, aumentando a probabilidade de que algumas dessas mudanças sejam problemáticas (VALENTE, 2020).

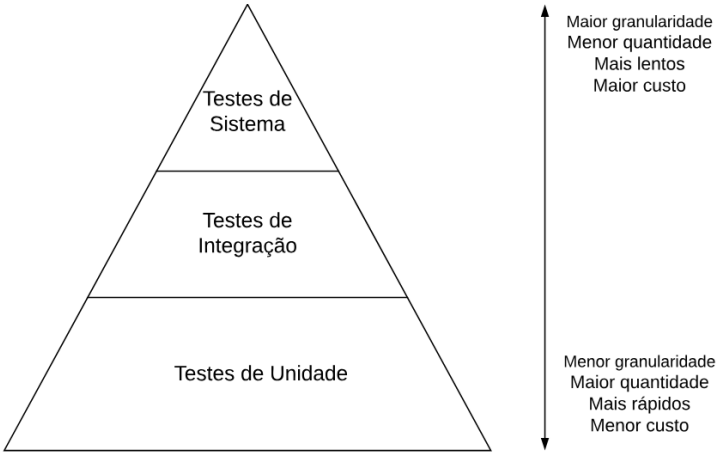
Para viabilizar essa prática, as equipes utilizam servidores de CI, que atuam como o motor de todo o processo de automação. Essa ferramenta é responsável por monitorar o repositório de código em busca de novas alterações. (VALENTE, 2020). Uma vez que uma nova alteração é detectada, o servidor de CI dá início à execução do *pipeline*, orquestrando cada uma das etapas de validação de maneira sequencial e padronizada. Além do monitoramento contínuo do repositório, um elemento essencial da Integração Contínua é o uso de scripts de build automatizados. Esses scripts orquestram todas as etapas de validação, como compilação, execução de testes, inspeção de código e até mesmo a implantação em ambientes de teste (DUVALL; MATYAS; GLOVER, 2007). A automação garante que o processo seja reproduzível e independente de ações manuais, promovendo consistência entre os ciclos de integração.

Segundo Duvall, Matyas e Glover (2007), um build vai além da simples compilação do código. Ele representa a junção de diversas atividades que asseguram que o sistema funcione como uma unidade coesa. Isso inclui não apenas testes e validações do código-fonte, mas também a integração de scripts de banco de dados versionados juntamente com a aplicação. A prática de CI também se apoia na geração de *feedback* rápido. Após cada alteração no repositório, o servidor de integração envia notificações automáticas aos membros do projeto com os resultados do build, permitindo a identificação precoce de falhas (DUVALL; MATYAS; GLOVER, 2007). As respostas imediatas contribuem para maior confiabilidade do sistema e menor custo na resolução de erros.

2.2.1 Testes Automatizados

A automação de testes é um pilar da Integração Contínua, ela permite a validação rápida e consistente do código a cada nova alteração. Uma estratégia de automação eficaz não se resume a automatizar todos os cenários de maneira indiscriminada. A pirâmide é uma metáfora visual que organiza os testes em diferentes camadas de granularidade e sugere a proporção ideal de testes em cada uma delas (VOCKE, 2018). A Figura 2 abaixo mostra a representação da pirâmide.

Figura 2 – Pirâmide de Testes.



Fonte: Valente (2020).

A premissa fundamental da Pirâmide de Testes é que uma equipe deve ter muito mais testes de baixo nível (rápidos e focados) do que testes de alto nível (lentos e abrangentes) (VOCKE, 2018). Essa estrutura visa otimizar a velocidade do *feedback* e reduzir os custos de manutenção da suíte de testes. A pirâmide é tipicamente dividida em três camadas principais:

1. Testes de Unidade.
2. Testes de Integração.
3. Testes de Sistema.

Os testes de unidade verificam partes pequenas e isoladas do código, como métodos ou classes, garantindo que produzam os resultados esperados. São rápidos, confiáveis e de fácil manutenção. Os testes de integração, ou testes de serviço, avaliam a colaboração entre componentes, como a comunicação entre uma classe de serviço e o banco de dados. Por envolverem múltiplos elementos reais do sistema, são mais lentos e exigem maior esforço de configuração (VALENTE, 2020). Já os testes de sistema, também chamados de ponta a ponta (*end-to-end*), validam o funcionamento completo da aplicação simulando a interação do usuário final. Apesar de oferecerem grande confiança sobre o funcionamento do sistema como um todo, esses testes são os mais lentos, frágeis e caros de manter. Por essa razão, devem ser implementados em menor quantidade, focando apenas nos fluxos mais críticos e importantes do sistema (VALENTE, 2020).

Uma abordagem comum para a execução desses testes de sistema é o uso de navegadores sem interface gráfica, conhecidos como headless browsers. Ferramentas como Selenium, Playwright e Puppeteer permitem simular interações reais do usuário em um navegador funcional, porém sem a sobrecarga da interface visual. Isso torna os testes significativamente mais rápidos e compatíveis com ambientes de integração contínua, como *pipelines* de CI/CD. Apesar de simplificarem a execução, os testes headless não substituem completamente os testes com interface real, sendo mais indicados para verificar lógica de fluxo e funcionalidades críticas de modo automatizado e eficiente (BROWSERSTACK, 2024).

2.2.2 Build Automatizado

O Build Automatizado é a prática que materializa os princípios da Integração Contínua, transformando o processo de compilação, teste e empacotamento de software em um fluxo orquestrado e repetível. Segundo Duvall, Matyas e Glover (2007), a construção do software deve ser uma "proposição de apertar o botão" , ou seja, um processo totalmente automatizado que pode ser iniciado com um único comando. Essa capacidade é de extrema importância para que os servidores de CI possam executar o *pipeline* sem qualquer intervenção humana.

O *build* automatizado envolve a execução de um conjunto de etapas, como compilação do código, execução de testes, inspeção do código e até mesmo a implantação em ambientes de teste, tudo sem a intervenção manual. A principal vantagem de automatizar o processo de *build* é a possibilidade de garantir que a integração de código ocorra de maneira contínua e sem falhas. Porém, um *build* pode ser mais complexo do que a simples compilação do código-fonte (DUVALL; MATYAS; GLOVER, 2007). Ele pode incluir múltiplas etapas, como a integração de bancos de dados e outras dependências, testes automatizados e verificações de qualidade de código.

De acordo com Duvall, Matyas e Glover (2007), existem diferentes tipos de *builds* que podem ser realizados, dependendo do objetivo e do estágio do ciclo de desenvolvimento. Estes builds podem ser classificados como: *Private build*, *Integration build* e *Release Build*, definidos da seguinte maneira:

- *Private build*: Realizado por um desenvolvedor antes de realizar o *commit* de suas alterações no repositório. Ele é importante para verificar se o código se integra corretamente com o código mais recente do repositório, evitando que o *commit* de código quebre o *build* de outros membros da equipe;

- *Integration build*: Realizado com o código de todos os desenvolvedores da equipe, sendo executado sempre que há alterações no repositório. O *build* de integração verifica a consistência do código de toda a equipe e geralmente envolve testes automatizados, como testes de unidade e integração;
- *Release Build*: O *build* que prepara o código para ser liberado para produção, ou para ser entregue aos clientes. Esse *build* inclui testes mais extensivos e etapas como a geração de pacotes para distribuição e a criação de instaladores;

Um aspecto fundamental do build automatizado é o *feedback* rápido. Quanto mais rapidamente o processo de integração e testes ocorrer, mais rapidamente as falhas podem ser identificadas e corrigidas.

2.2.3 Padrões de gerenciamento de branches

Projetos de software podem ser coordenados com padrões de gerenciamento de *branches* em sistemas de versionamento de código. Esses padrões auxiliam as partes envolvidas no projeto de maneira que possam manter a organização e o desenvolvimento do mesmo. Um dos padrões que mais se destacam são o "*Git-Flow*" e o "*Trunk-Based Development* (TBD)", cada um com estratégias distintas de organização e entrega de código.

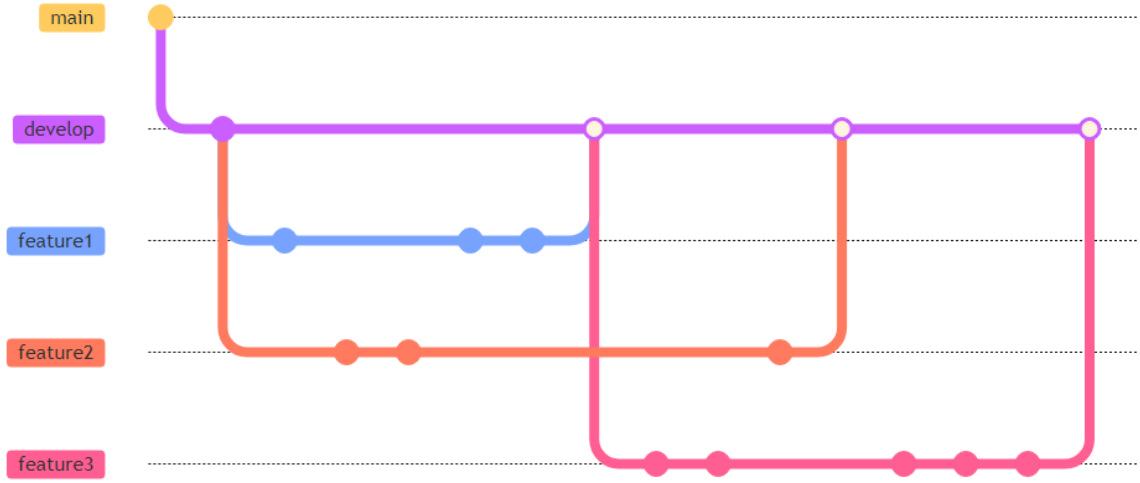
O Git-Flow é um modelo de gerenciamento de branches proposto por Vincent Driessen, que estabelece uma estrutura com papéis e ciclos de vida bem definidos (DRIESSEN, 2010). O modelo se fundamenta em duas ramificações principais: a "*main*", que deve sempre refletir o estado estável do código em produção, e a "*develop*", que serve como uma linha de base para a integração de funcionalidades concluídas (VALENTE, 2025). A partir dessas duas, são criadas ramificações auxiliares e temporárias para organizar o fluxo de desenvolvimento:

- *Feature branches*: Derivadas e reintegradas à *develop*, são usadas para desenvolver novas funcionalidades de modo isolado, sem impactar a base de código principal. A Figura 3 ilustra três branches de funcionalidade criados a partir de *develop* usando Git-Flow.
- *Release branches*: Criadas a partir da *develop* quando esta contém as funcionalidades necessárias para uma nova versão. Nestas branches, realizam-se apenas tarefas de preparação para o lançamento, como correções de bugs de última hora e ajustes

na documentação. Ao final, são mescladas tanto na *main* quanto na *develop*. Na Figura 4 é mostrado o comportamento dessa branch no Git-Flow.

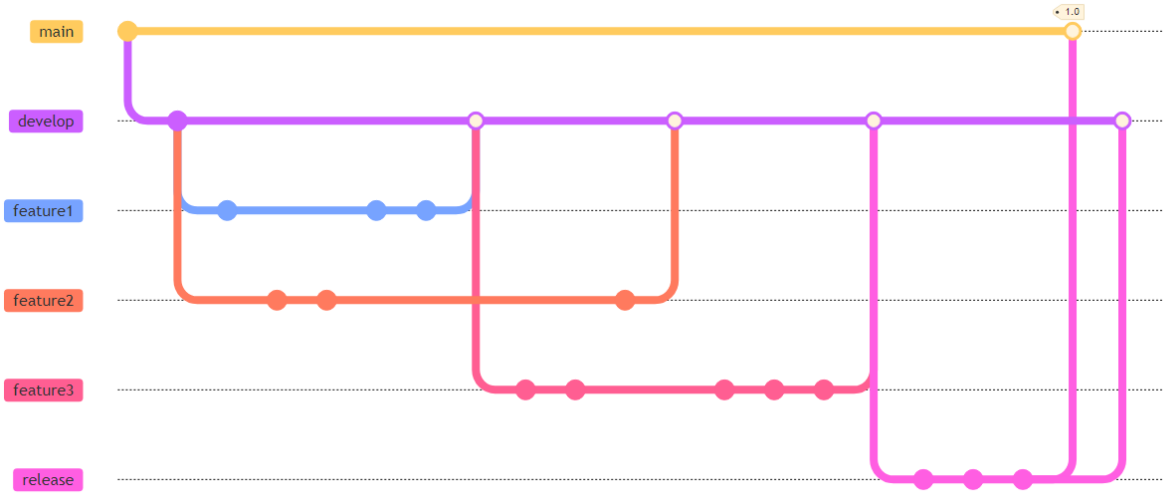
- *Hotfix branches*: Derivadas diretamente da *main*, são empregadas para corrigir falhas críticas em produção de maneira urgente. Após a correção, devem ser mescladas de volta à *main* e também à *develop* para garantir que a correção seja incorporada nos próximos lançamentos. A Figura 5 ilustra o comportamento de uma branch de *hotfix* com o Git-Flow.

Figura 3 – Comportamento de uma *feature branch*.



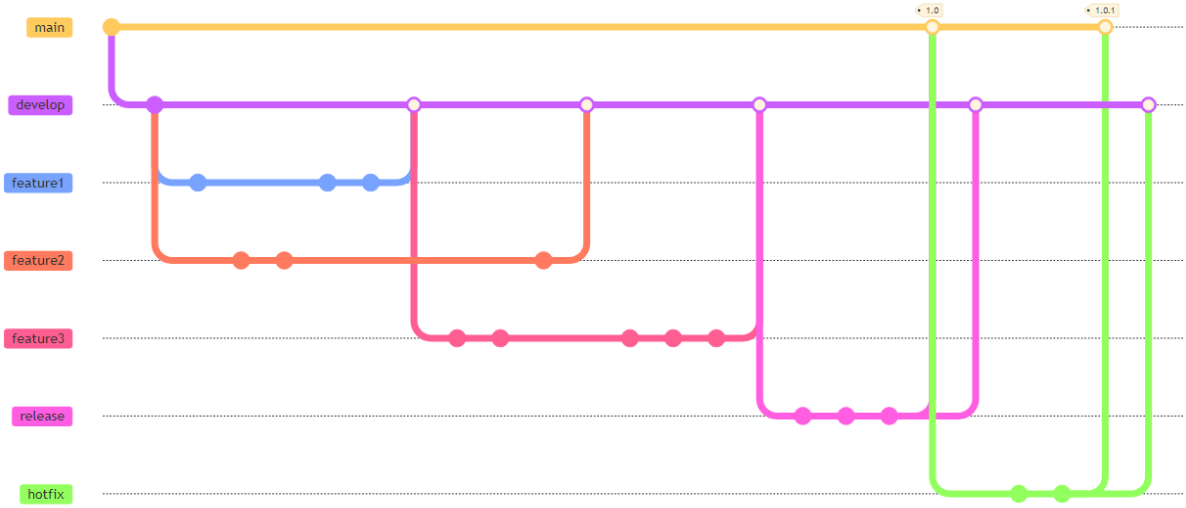
Fonte: Valente (2025).

Figura 4 – Comportamento de uma *release branch* .



Fonte: Valente (2025).

Figura 5 – Comportamento de uma *hotfix branch*.

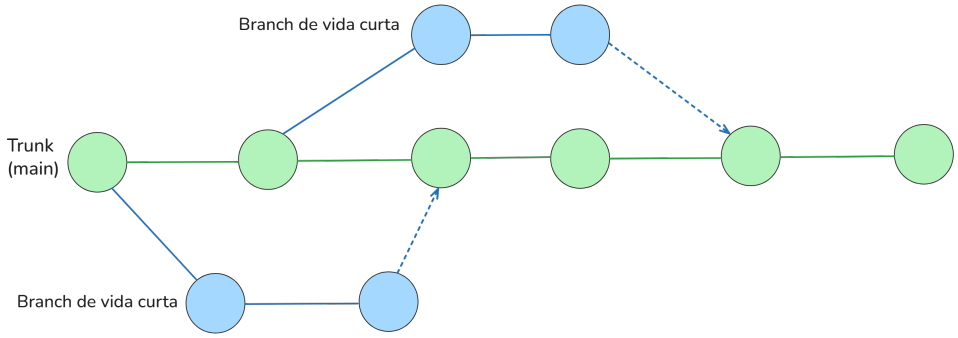


Fonte: Valente (2025).

O outro padrão de gerenciamento de branches, *Trunk-Based Development* adota uma abordagem mais enxuta e contínua. Nele, a equipe trabalha majoritariamente em um único branch principal, chamado *main* ou *trunk*. Segundo Valente (2025), os *commits* são realizados diretamente nessa ramificação, embora ainda seja possível utilizar *feature branches*, desde que sejam breves e rapidamente integradas.

Para que esse modelo funcione adequadamente, é essencial contar com uma boa cobertura de testes automatizados, especialmente de unidade e integração, a fim de evitar falhas no código principal. Além disso, o TBD facilita a adoção de práticas como CI e CD, ao promover ciclos curtos e integrações frequentes (VALENTE, 2025). A Figura 6 exemplifica como ocorre o padrão de gerenciamento de *branches* do *Trunk-Based Development*.

Figura 6 – Funcionamento do TBD.



Fonte: Elaborado pelo autor.

2.3 Implantação Contínua e Entrega Contínua (CD)

Ambos os termos "Implantação Contínua" e "Entrega Contínua" são facilmente confundidos, em inglês o seu acrônimo é o mesmo (CD), e os termos são "*Continuous Deployment*" e "*Continuos Delivery*". Embora intimamente relacionados, eles não são sinônimos. A distinção fundamental entre as duas abordagens estão resididas no tratamento da etapa final de liberação do software para o ambiente de produção.

Humble e Farley (2010) descrevem a entrega contínua como uma prática cujo objetivo é criar um processo repetível, confiável e automatizado para levar as alterações de software à produção o mais rápido possível. Para que isso seja possível, o código deve passar por um conjunto de testes automatizados e critérios de aceite, o software é preparado para produção, mas a implantação em si não é realizada automaticamente (HUMBLE; FARLEY, 2010). Ou seja, o código passa por várias etapas de verificação e testes, mas a decisão final sobre a liberação do código para produção é realizada manualmente.

A principal distinção da implantação contínua em relação a entrega contínua é que a liberação do código para produção ocorre automaticamente, sem intervenção manual.

Assim que uma alteração no código passa nos testes automatizados e é considerada estável, ela é automaticamente implantada em produção. Esse processo remove a necessidade de intervenções manuais na liberação do software para produção (HUMBLE; FARLEY, 2010).

2.3.1 Estratégias de Implantação

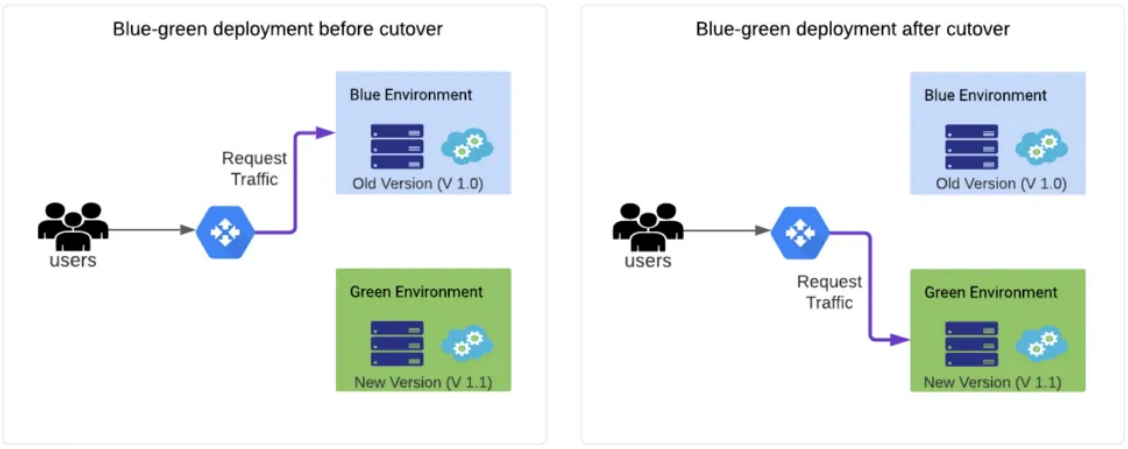
A implantação de software em produção é uma das atividades mais críticas do ciclo de vida de desenvolvimento. Uma estratégia de implantação bem definida visa reduzir os riscos, o tempo de inatividade e o estresse associado a um lançamento, garantindo que o processo seja tão confiável e repetível quanto a implantação em qualquer ambiente de teste (HUMBLE; FARLEY, 2010).

A fim de alcançar os objetivos propostos, diversas estratégias de implantação têm sido adotadas, variando em termos de complexidade e aplicabilidade, mas compartilhando o propósito comum de tornar o processo de entrega em produção mais seguro e controlado. Entre as principais abordagens, podemos citar "*Blue-green Deployment*", "*Rolling Deployment*" e "*Atomic Deployment*".

Conforme Humble e Farley (2010), o *Blue-green Deployment* constitui uma das técnicas mais eficientes para o gerenciamento de lançamentos de software. O princípio fundamental dessa abordagem é a manutenção de duas versões idênticas do ambiente de produção, denominadas "azul" (*blue*) e "verde" (*green*). Em qualquer momento, apenas uma dessas infraestruturas está ativa e recebendo o tráfego dos usuários. Por exemplo, quando o ambiente verde está ativo, os usuários são direcionados para ele.

Quando uma nova versão da aplicação é desenvolvida e validada, ela é implantada na infraestrutura que está inativa (ambiente azul). Durante este processo, o ambiente azul pode ser preparado e testado sem afetar a operação do ambiente verde que está em produção. Uma vez que a nova versão no ambiente azul esteja pronta, a comutação para a nova versão é realizada de maneira simples, alterando a configuração de um roteador para direcionar o tráfego do ambiente verde para o azul. A Figura 7 ilustra o funcionamento dessa estratégia.

Figura 7 – Estratégia Blue-green Deployment.



Fonte: Cloudops (2023).

O *Rolling Deployment* consiste em atualizar gradualmente as instâncias de uma aplicação em produção, em vez de substituí-las todas de uma vez. Esse processo envolve a atualização de um subconjunto de servidores por vez, em vez de atualizar todos simultaneamente. Durante a implantação, a aplicação pode executar versões diferentes simultaneamente, permitindo que parte do tráfego seja atendida pela versão antiga e parte pela nova (HANES, 2024). A estratégia minimiza o tempo de inatividade e reduz o impacto para os usuários finais, pois sempre há uma versão da aplicação disponível para atender às requisições. O seu funcionamento é detalhado na Figura 8.

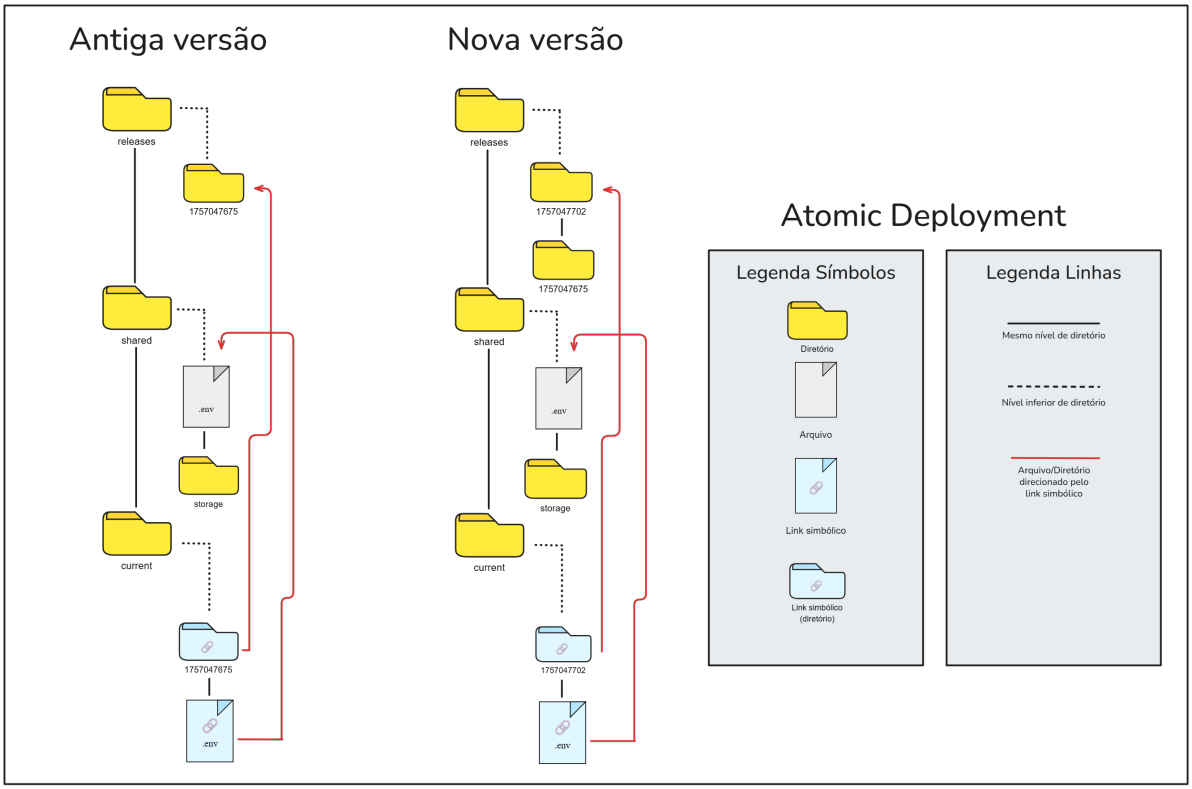
Figura 8 – Estratégia Rolling Deployment.

Fonte: Hanes (2024).

Segundo Grunwell (2019), o *Atomic Deployment* garante que a implantação de uma nova versão de uma aplicação seja bem-sucedida ou totalmente revertida, sem deixar o sistema em estado inconsistente. O objetivo é alcançar zero tempo de inatividade, proporcionando uma transição instantânea para a nova versão. A transição é gerida por meio de links simbólicos (*symlinks*), no qual o *symlink* "current" aponta para a versão ativa e é atualizado somente quando a nova versão está completamente pronta. Recursos compartilhados (*shared*) são vinculados às versões usando symlinks. Em caso de falha, o processo de reversão (*rollback*) é facilitado, pois o *symlink* current pode ser rapidamente redirecionado para uma versão anterior estável.

A Figura 9 ilustra a estrutura de diretórios da estratégia, na qual cada nova versão da aplicação, nomeada por seu *timestamp* de criação, é implantada em um diretório próprio dentro de *releases*. O mecanismo de atualização atômica consiste em redirecionar o link simbólico current para a nova versão, garantindo uma transição instantânea e mantendo o acesso a recursos persistentes no diretório shared.

Figura 9 – Estratégia *Atomic Deployment*.



Fonte: Elaborado pelo autor.

2.4 Ferramentas de CI/CD

Esta seção apresenta as principais ferramentas disponíveis no mercado utilizadas para implementação de *pipelines* de Integração Contínua / Implantação Contínua, destacando suas características e critérios relevantes para sua aplicação no SisGera. A escolha das ferramentas adequadas ao sistema deve levar em consideração fatores como custo, usabilidade, e a sustentabilidade ao longo do tempo.

2.4.1 GitLab CI/CD

O GitLab CI/CD é uma solução integrada à plataforma GitLab que permite a criação de *pipelines* para automação de processos de CI/CD. Suas configurações são realizadas por meio de arquivos *YAML Ain't Markup Language* (YAML), por padrão o arquivo principal é denominado "`gitlab-ci.yml`", armazenado diretamente no repositório do projeto (GITLAB, 2024b). A ferramenta oferece duas opções de hospedagem e gerenciamento, sendo elas por gerenciamento próprio (*self-managed*) e software como serviço (*Software as a Service* (SaaS)). Essa modalidade como serviço elimina a necessidade de gerenciar infraestrutura local e configurações próprias, preocupando apenas com planos de assinatura (GITLAB, 2024a).

Além disso, o GitLab CI/CD como SaaS oferece diferentes planos para necessidades específicas, cada um com suas características. O SisGera é um projeto que necessita evadir de custos, então a escolha de uma ferramenta acessível irá ter peso. A Tabela 1 exibe as características do plano gratuito do GitLab CI/CD.

Tabela 1 – Recursos do plano gratuito do GitLab CI/CD.

Recurso	Descrição
Minutos de CI/CD	400 minutos/mês para executores gerenciados pelo GitLab
Número de Usuários	Até 5 usuários por grupo de nível superior
Espera em fila	Execução de executores depende de entrada em filas
Atualização dos Executores	Executores são atualizados e mantidos pelo próprio GitLab

Fonte: Elaborado pelo autor.

2.4.2 GitHub Actions

O GitHub Actions é uma ferramenta integrada à plataforma GitHub que permite automatizar processos personalizados, como CI/CD. Ele é projetado para funcionar nativamente com os repositórios de código hospedados no GitHub, facilitando a automação diretamente na plataforma. No GitHub Actions, os *pipelines* de automação são chamados de "*workflows*" e são configurados por meio de arquivos no formato YAML. Esses arquivos são armazenados no diretório "`.github/workflows`", diretório padrão e obrigatório para os *workflows* (GITHUB, 2024a).

Existem diferentes planos oferecidos pelo GitHub Actions como SaaS, seu plano gratuito é bem vantajoso ao SisGera. Pelo repositório do projeto ser versionado no GitHub, a familiaridade com a plataforma é maior, além disso o plano gratuito oferece 2.000 minutos para execução de *workflows* (GITHUB, 2024b). A Tabela 1 exibe as características do plano gratuito do GitHub Actions.

Tabela 2 – Recursos do plano gratuito do GitHub Actions.

Recurso	Descrição
Minutos de CI/CD	2.000 minutos/mês para executores gerenciados pelo GitHub Actions
Número de Usuários	Ilimitado
Espera em fila	Execução de executores depende de entrada em filas
Atualização dos Executores	Executores são atualizados e mantidos pelo próprio GitHub

Fonte: Elaborado pelo autor.

2.4.3 Jenkins

O Jenkins é uma das ferramentas mais populares e amplamente utilizadas no mercado. Trata-se de uma solução *open-source* que permite a automação de tarefas em todo o ciclo de vida do desenvolvimento de software, incluindo os processos de CI/CD. Sua flexibilidade é garantida por meio de uma extensa biblioteca de plugins, que permite integrá-lo com diversas ferramentas, linguagens de programação e sistemas operacionais (JENKINS, 2024a). A elaboração de *pipelines* ocorre em arquivos "Jenkinsfile", servindo como a configuração central para gerenciar as etapas de *build*, teste, entrega e *deploy* de projetos de software (JENKINS, 2024b).

Ao contrário das outras ferramentas citadas, o Jenkins possui uma estrutura

self-managed e por isso, toda as configurações e manutenibilidade é feita pelo usuário. A complexidade envolvida para as configurações do servidor do Jenkins e das questões de infraestrutura levantam pontos negativos para sua escolha. Há também possibilidade de hospedar o servidor em uma máquina virtual de provedores de soluções em nuvem com pré-configurações, algo que facilitaria seu uso. A Tabela 3 exibe as características da ferramenta.

Tabela 3 – Recursos do Jenkins.

Recurso	Descrição
Minutos de CI/CD	Ilimitado
Número de Usuários	Ilimitado
Espera em fila	Não há espera em fila
Atualização dos Executores	Executores são atualizados pelo próprio usuário

Fonte: Elaborado pelo autor.

2.4.4 BitBucket Pipelines

O Bitbucket Pipelines é uma solução SaaS que está integrado diretamente ao Bitbucket Cloud, a plataforma de hospedagem de código-fonte baseada em nuvem da Atlassian. Ele elimina a necessidade de gerenciar infraestrutura local e atende apenas ao Bitbucket como sistema de versionamento de código (ATLASSIAN, 2024).

As *pipelines* são configurados utilizando um arquivo no formato YAML, denominado "`bitbucket-pipelines.yml`", que deve ser armazenado na raiz do repositório. Esse arquivo descreve as etapas (*steps*) e comandos (*scripts*) que compõem o *pipeline*, como tarefas de *build*, testes e implantação. Além disso, a ferramenta suporta a organização modular dos pipelines, permitindo que etapas ou configurações sejam referenciadas em outros arquivos YAML ou *scripts* externos. A Tabela 4 detalha características do plano gratuito da BitBucket Pipelines.

Tabela 4 – Recursos do plano gratuito do BitBucket Pipelines.

Recurso	Descrição
Minutos de CI/CD	50 minutos/mês para executores gerenciados pelo BitBucket
Número de Usuários	Ilimitado
Espera em fila	Execução de executores depende de entrada em filas
Atualização dos Executores	Executores são atualizados e mantidos pelo próprio BitBucket

Fonte: Elaborado pelo autor.

2.5 Considerações Finais

Este capítulo discutiu os conceitos fundamentais necessários à compreensão do presente trabalho, além de apresentar os pontos de partida para a orientação das escolhas das ferramentas e das estratégias utilizadas no desenvolvimento do *pipeline* CI/CD e no seu funcionamento.

Com diversas opções de escolhas de ferramentas que lidam com o processo de CI/CD, a opção que mais se adapta ao escopo do SisGera é o GitHub Actions. Um dos principais fatores que justificam essa escolha é a integração nativa com o GitHub, no qual o repositório do SisGera já está hospedado. Essa integração elimina a necessidade de configurações complexas ou de conectores adicionais para sincronizar o repositório, como uma ferramenta igual ao Jenkins faria.

Outro fator é que o plano gratuito oferece muitos minutos mensais, e isso atenderia bem ao projeto. Esse ponto garante que o *pipeline* possa ser executado sem custos adicionais, uma exigência importante para a sustentabilidade de um projeto que visa atender a demandas públicas e é mantido de modo colaborativo. Portanto, o GitHub Actions se apresenta como a melhor escolha para o *pipeline* do SisGera, combinando integração nativa, custo zero, simplicidade de configuração e alinhamento com as demandas do projeto.

Além dos critérios objetivos mencionados, é importante considerar a experiência do usuário com a ferramenta. Virtanen (2021) realizando um estudo comparando diversas soluções de CI/CD identificou que, embora todas cumpram adequadamente as etapas de um pipeline básico, não é possível recomendar uma única ferramenta como a melhor, visto que a eficiência de uso está diretamente relacionada à familiaridade do usuário

com a solução adotada. Isso valida ainda mais a escolha pelo GitHub Actions, já que parte das pessoas envolvidas com o SisGera já possui familiaridade com a plataforma do GitHub, o que reduz a curva de aprendizado e potencializa a produtividade.

3 Desenvolvimento

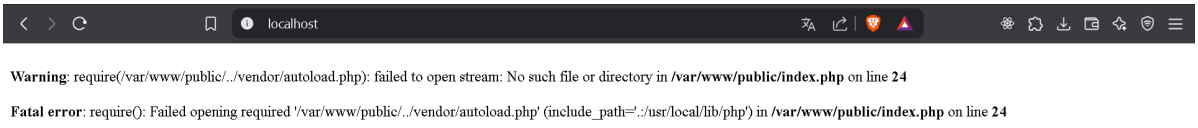
Neste capítulo, são apresentadas as etapas, decisões técnicas e soluções adotadas durante o desenvolvimento do pipeline CI/CD para o Sistema de Gerenciamento e Registro de Atividades. Primeiramente, a Seção 3.1 aborda o processo de adaptação ao sistema e a análise de suas particularidades técnicas, destacando as dificuldades encontradas e as respectivas soluções implementadas. Em seguida, a Seção 3.2 descreve a criação e configuração de um ambiente de hospedagem para fins de testes e também de um repositório auxiliar. Já na Seção 3.3 é abordado a definição da estratégia de implantação adotada. Para a Seção 3.4 é detalhado o desenvolvimento do pipeline automatizado e sua aplicação prática no ambiente real do SisGera. Por fim, a Seção 3.6 detalha a elaboração da construção do guia de uso e configuração do *pipeline*, bem como boas práticas para sustentabilidade do projeto.

3.1 Análise e Adaptação ao Sistema

Antes de realizar qualquer modificação que pudesse impactar o comportamento da aplicação, foi necessário compreender a estrutura e implementação do sistema. Essa análise do repositório foi essencial para entender suas limitações e definir as estratégias para a implementação do *pipeline* CI/CD.

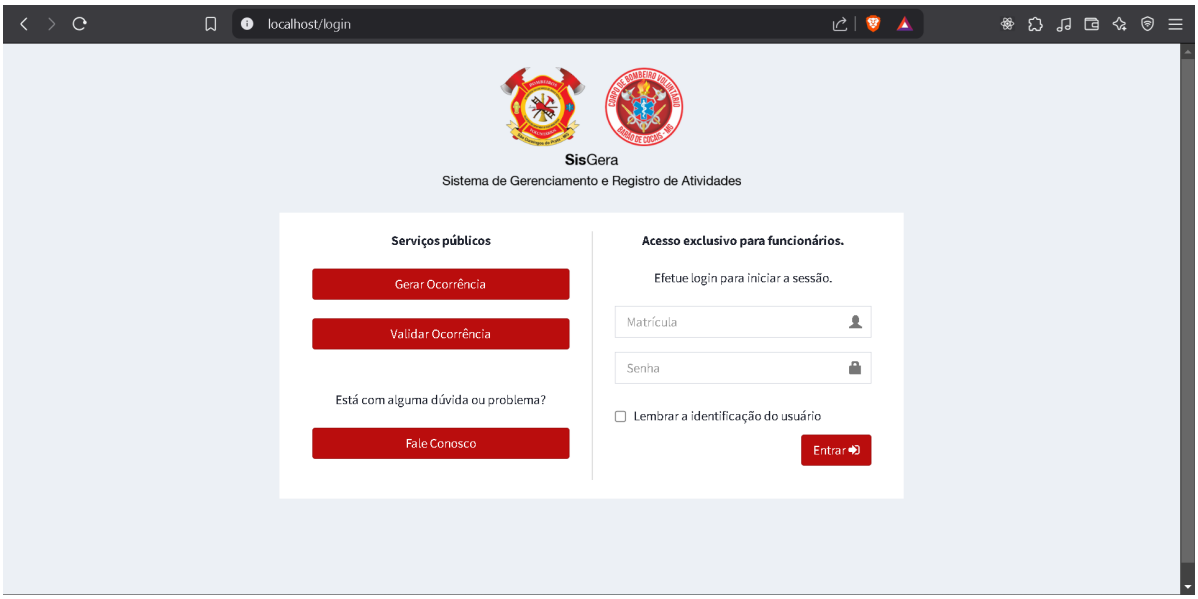
O primeiro passo consistiu na obtenção do código-fonte disponível no repositório, seguido da instalação de suas dependências e da execução do sistema. As contribuições de Barros (2023) incluíram o uso do Docker para a distribuição da aplicação do SisGera. O Docker é um *software* de código aberto de virtualização leve baseada em *containers*, que permite empacotar, distribuir e executar aplicações de modo isolado e portátil (DOCKER, 2025). Dado o benefício proporcionado pelo Docker, a primeira tentativa de execução da aplicação foi realizada utilizando o mesmo. No entanto, durante a construção da aplicação via Docker, foi identificado um problema que ao iniciar o *container*, a aplicação não encontrava o diretório “vendor”. Esse diretório é responsável por armazenar as dependências dos pacotes da aplicação.

Figura 10 – Erro apresentado ao acessar aplicação.



Fonte: Elaborado pelo autor.

Figura 11 – Carregamento da aplicação após correção.



Fonte: Elaborado pelo autor.

A Figura 10 exibe detalhadamente o erro apresentado na tentativa de acesso da aplicação. Já na Figura 11 ela exibe o carregamento da aplicação após a correção. A causa do problema estava na configuração do "Dockerfile", esse arquivo é responsável pela criação de imagens Docker. Ele define todas as instruções necessárias para construir um ambiente executável para a aplicação.

3.2 Elaboração do Ambiente de Testes

A criação de um ambiente de hospedagem similar ao ambiente real do SisGera foi necessária. Isso possibilitou compreender o ambiente que gerencia a hospedagem e também de evitar comprometer o ambiente real com qualquer modificação realizada. O

ambiente real é administrado com o cPanel. Ele oferece uma interface intuitiva para administrar domínios, e-mails, bancos de dados, arquivos e configurações do servidor, sem a necessidade de conhecimentos avançados em administração de servidores (CPANEL, 2025). Além disso, foi elaborado um repositório auxiliar da aplicação no GitHub para estudo de configurações e envio de modificações testes.

3.2.1 AWS Lightsail

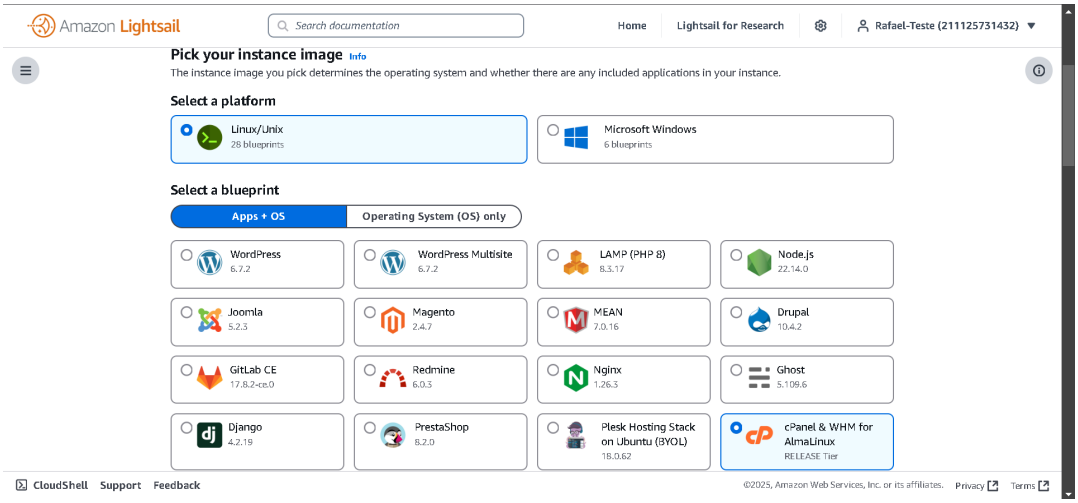
A Amazon oferece diversos serviços na sua plataforma *Amazon Web Services* (AWS). Um dos serviços que possibilitaram a criação de um ambiente isolado compondo uma hospedagem e licença do cPanel foi o *Amazon Lightsail*, uma solução que fornece servidores virtuais privados de maneira simplificada e com um custo previsível. A escolha por essa plataforma se deu principalmente pelas facilidades oferecidas no plano "*free tier*", que permitiu a implementação e teste do ambiente sem custos iniciais.

O objetivo da utilização do *Amazon Lightsail* foi criar um ambiente isolado que permitisse a implementação do pipeline CI/CD sem interferir diretamente no servidor de hospedagem do SisGera. Dessa maneira, foi possível validar as etapas do fluxo de integração e implantação contínua antes de ser realizado em produção, garantindo maior confiabilidade ao processo. Para a construção do ambiente de hospedagem, foram seguidas as seguintes etapas:

1. Provisionamento da Instância

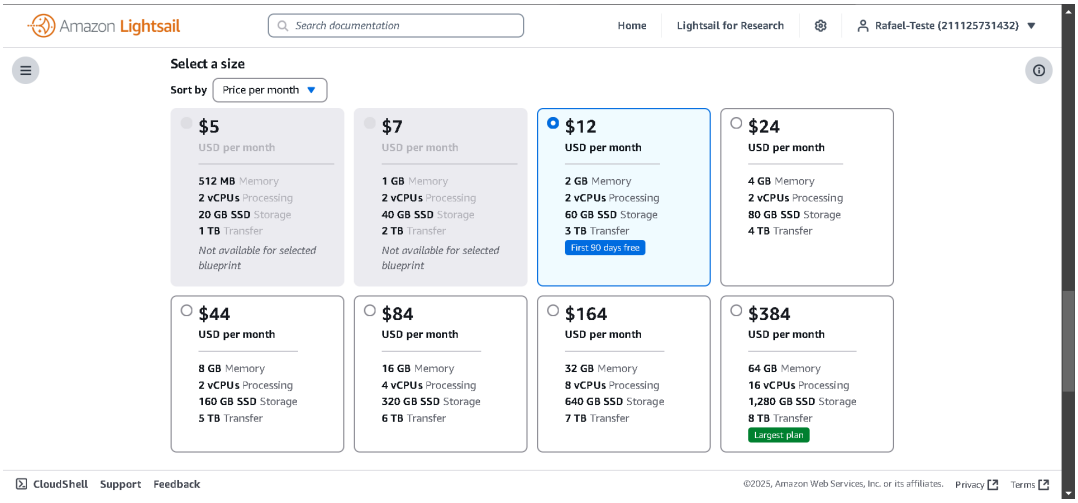
   Inicialmente, foi criada uma instância no *AWS Lightsail* com sistema operacional AlmaLinux, compatível com a instalação do cPanel e uma licença de 15 dias para seu uso. A escolha do plano levou em consideração os requisitos mínimos do cPanel, incluindo a necessidade de memória RAM, processamento e armazenamento suficientes para suportar o funcionamento do ambiente de testes. A Figura 12 e a Figura 13 exibem a seleção de configurações disponíveis para a instância.

Figura 12 – Seleção de configuração de aplicações e sistema operacional da instância.



Fonte: Elaborado pelo autor.

Figura 13 – Seleção de configuração de hardware da instância.



Fonte: Elaborado pelo autor.

2. Licenciamento e Validação

   Para garantir o pleno funcionamento do cPanel, foi necessária a ativação de uma licença válida. Essa licença permitiu a utilização completa dos recursos do painel, possibilitando a replicação do ambiente de produção.

3. Instalação e Configuração do cPanel

Com a instância provisionada, foi realizada a instalação do cPanel a partir do *script* oficial fornecido pelo cPanel e a ativação da licença de uso. Após a conclusão da instalação, a interface administrativa do *Web Host Manager* (WHM) foi acessada para a configuração inicial, incluindo ajustes de versões do PHP, MySQL, contas de usuário e permissões para acesso remoto.

4. Ajustes Finais e Testes

   Após as configurações iniciais, foram realizados testes para verificar o funcionamento correto do ambiente. Isso incluiu:

   - Validação do acesso ao cPanel e WHM;
   - Acesso do terminal do servidor via painel de usuário;
   - Verificação da versão do PHP e MySQL;
   - Criação e uso de chave *Secure Shell* (SSH);
   - Testes de *deploy* automatizado via *pipeline* CI/CD;

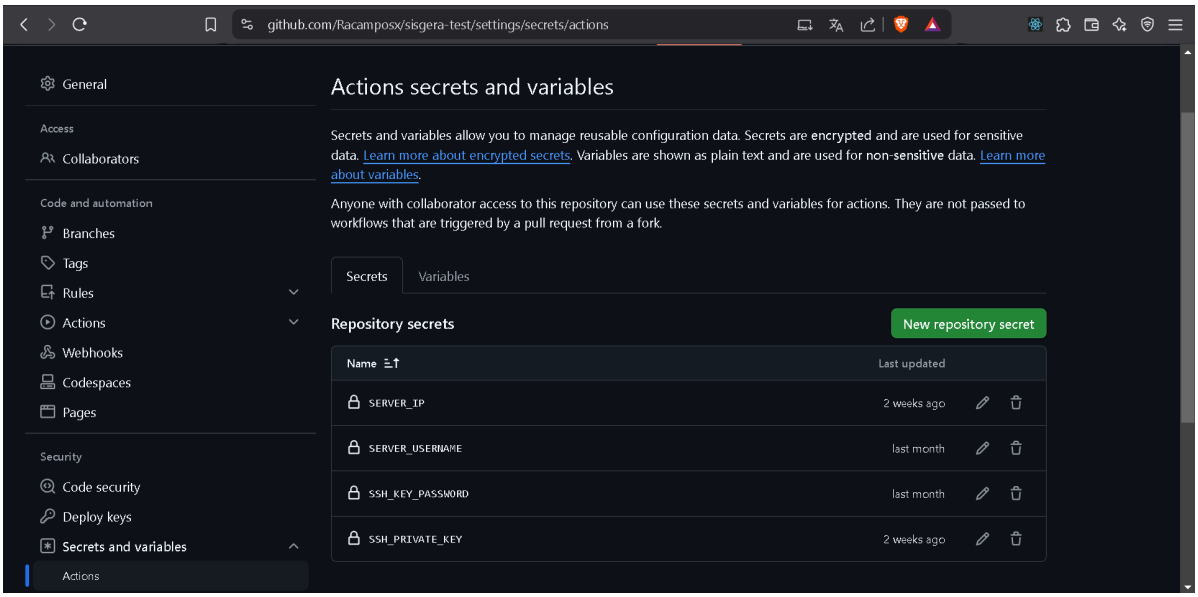
3.2.2 Repositório Auxiliar

A elaboração de um repositório auxiliar foi demandada para que configurações que envolvam o repositório não afetem o real. As configurações envolveram a criação de varáveis de segredo, chamadas comumente de "*secrets*". Elas são variáveis de ambiente protegidas usadas para armazenar informações sensíveis, como chaves de API, tokens de acesso, credenciais de banco de dados e outros dados que não devem ser expostos no código-fonte (GITHUB, 2025). Os *secrets* são criptografados e geralmente utilizados no GitHub Actions, a plataforma de automação do GitHub, no qual podem ser acessados dentro dos fluxos de automação.

A Figura 14 mostra o ambiente de armazenamento de *secrets*, bem como algumas das que foram utilizadas. Dentre essas variáveis de segredos, foram utilizadas: "SERVER_IP", "SERVER_USERNAME", "SSH_PRIVATE_KEY", "SLACK_CI_CHANNEL", "SLACK_CD_CHANNEL" e "SLACK_ROLLBACK_CHANNEL". A variável SERVER_IP armazena o endereço IP do servidor remoto para onde a aplicação será implantada, garantindo que a automação do GitHub Actions saiba para onde direcionar os arquivos. A variável SERVER_USERNAME contém o nome de usuário com permissões para acessar o servidor via SSH e executar os comandos necessários para a implantação. Já a SSH_PRIVATE_KEY contém a chave privada SSH necessária para autenticar a conexão entre o repositório e o servidor remoto de maneira segura. As

variáveis com o prefixo “SLACK” referem-se aos canais de comunicação da aplicação Slack e têm a função de configurar os mecanismos de notificação relacionados à execução do *pipeline* CI/CD. Cada variável está vinculada a um canal específico, por meio do qual são enviadas mensagens automáticas que informam o status das tarefas executadas, sinalizando eventos de sucesso ou falha ao longo do processo.

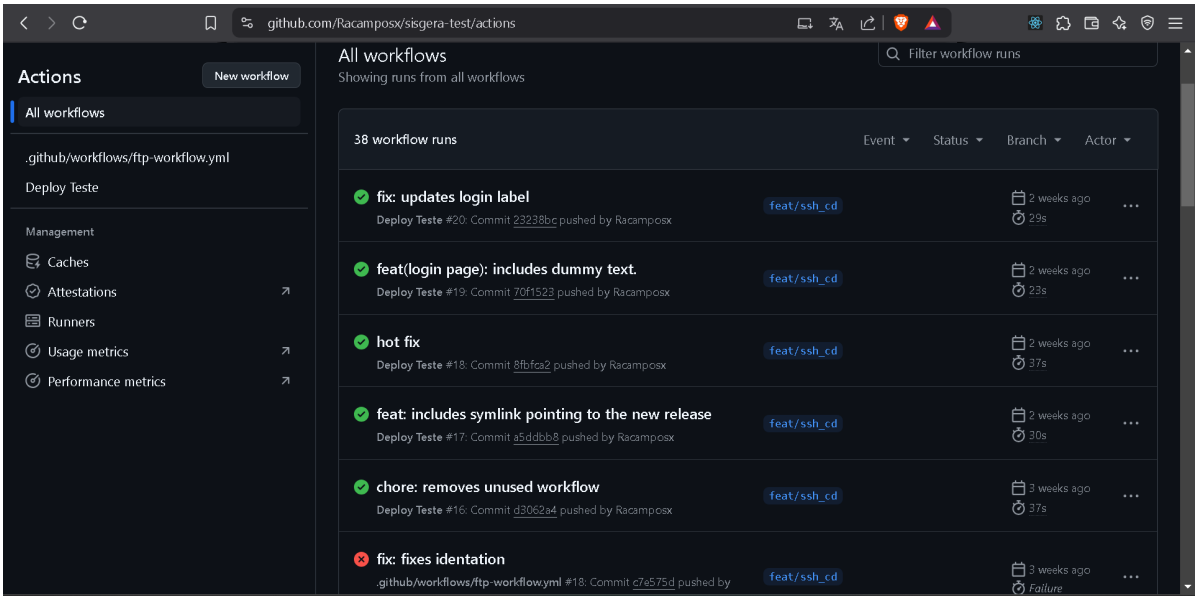
Figura 14 – Armazenamento das variáveis de segredo.



Fonte: Elaborado pelo autor.

Além da configuração dos *secrets*, foi criado um workflow no GitHub Actions para iniciar a automação da Implantação Contínua. A Figura 15 mostra o histórico de execuções desses workflows no repositório auxiliar, o que facilita a rastreabilidade das alterações e permite uma resposta mais rápida em caso de erros.

Figura 15 – Histórico de execução de *Workflows*.



Fonte: Elaborado pelo autor.

3.3 Estratégia de *Deploy* Adotada

No Capítulo 2, especificamente na Subseção 2.3.1, foram cobertas estratégias de implantação que serviram como norteadores para conduzir uma estratégia que fosse condizente com a estrutura atual do SisGera. A estratégia *Atomic Deployment* se adéqua bastante às características e limitações do ambiente de hospedagem que a aplicação se encontra, por isso ela foi adotada.

A Figura 16 ilustra a organização da estrutura de diretórios empregada para implementar essa abordagem. O diretório “releases” contém diferentes versões da aplicação, como Current e Previous, representando as versões atual e anterior, respectivamente. O diretório “shared” armazena os recursos persistentes, que não são versionados, necessários para o funcionamento contínuo da aplicação, tais como o diretório “storage”, que guarda arquivos de logs e caches, e o arquivo “.env”, que contém as variáveis de ambiente e configurações essenciais. Já o diretório “public_html” é onde os arquivos públicos da aplicação ficam acessíveis, sendo composto por links simbólicos que apontam para os arquivos da versão atualmente ativa. Esses links garantem que a transição entre versões ocorra sem necessidade de copiar ou mover arquivos manualmente.

Figura 16 – Estrutura de diretórios da estratégia de *deploy*.

Fonte: Elaborado pelo autor.

Com essa organização, sempre que uma nova versão é implantada, seus arquivos são enviados para um novo diretório dentro de "releases". Em seguida, a estrutura de links simbólicos é atualizada para apontar para essa nova versão, e os diretórios são atualizados para que a nova versão componha o nome "Current" e a antiga "Previous". Caso seja necessário reverter o processo de *deploy*, basta alterar o link simbólico para apontar para a versão anterior, sem necessidade de reprocessamento ou transferência de arquivos adicionais.

Essa estratégia adotada foi inicialmente pensada no ambiente de teste. Porém, sua composição no ambiente real precisaria ser levemente alterada, já que o SisGera possui dois ambientes. Um para homologação e outro para produção. A pasta "public_html" do servidor original compartilha o diretório "root" de cada domínio dos ambientes, ele é o diretório onde ficam armazenados os arquivos do site que serão exibidos quando o domínio for acessado. A Figura 17 representa a estrutura de diretórios do servidor do SisGera. Existe uma separação de diretórios para cada ambiente, dessa vez essa estrutura consta com o nome de cada ambiente para sua identificação.

Figura 17 – Estrutura de diretórios da estratégia adaptada ao ambiente do SisGera.

```
.
├── shared_sisgera/
│   ├── storage/
│   └── .env
├── shared_homologacao/
│   ├── storage/
│   └── .env
├── releases_sisgera/
│   ├── Current/
│   └── Previous/
├── releases_homologacao/
│   ├── Current/
│   └── Previous/
└── public_html/
    ├── index.php -> ../releases_sisgera/Current/public/index.php
    ├── css -> ../releases_sisgera/Current/public/css
    ├── js -> ../releases_sisgera/Current/public/js
    ├── dist -> ../releases_sisgera/Current/public/dist
    ├── ...
    ├── .htaccess -> ../releases_sisgera/Current/public/.htaccess
    └── homologacao/
        ├── index.php -> ../releases_homologacao/Current/public/index.php
        ├── css -> ../releases_homologacao/Current/public/css
        ├── js -> ../releases_homologacao/Current/public/js
        ├── dist -> ../releases_homologacao/Current/public/dist
        ├── ...
        └── .htaccess -> ../releases_homologacao/Current/public/.htaccess
```

Fonte: Elaborado pelo autor.

3.4 Desenvolvimento do *pipeline* CI/CD

Com o ambiente de testes e a estratégia de implantação definidos, foi possível iniciar o desenvolvimento do *pipeline* de Integração Contínua / Implantação Contínua, utilizando a ferramenta GitHub Actions. O *pipeline* desenvolvido foi segmentado em três ações principais: Integração Contínua, Implantação Contínua e mecanismo de *rollback*, cada qual com responsabilidades bem definidas e executadas em diferentes contextos de operação.

Além disso, foi implementado um ambiente de comunicação para que todas as ações do *pipeline* fossem notificadas assim que executadas. Para isso, utilizou-se o aplicativo Slack, uma plataforma de comunicação e colaboração em equipe que centraliza conversas, arquivos e ferramentas em um único ambiente, facilitando a comunicação interna e o gerenciamento de projetos.

3.4.1 Integração Contínua

A etapa de Integração Contínua é executada automaticamente sobre a *branch* "main" e "develop", sendo acionada sempre que há um *push* de código para essa ramificação via merge. Seu propósito é verificar a integridade e a qualidade do código antes da promoção para o ambiente de produção ou homologação. Durante essa etapa, diversas ações são realizadas sequencialmente:

1. Notificação inicial via Slack, sinalizando o início do processo de integração.
2. Checkout do código-fonte do repositório.
3. Configuração do ambiente PHP, com a instalação de extensões e ferramentas compatíveis com a aplicação.
4. Gerenciamento de dependências com o Composer.
5. Execução de verificação de estilo de código utilizando a ferramenta PHP-CS-Fixer.
6. Construção da aplicação via Docker.
7. Execução de migrações do banco de dados.
8. Execução dos testes unitários escritos.
9. Notificação final via Slack, informando o sucesso ou falha em uma das etapas executadas.

A automatização desses testes e validações evita que erros simples sejam propagados para os ambientes de homologação ou produção. É importante ressaltar que pelo fato do SisGera ser um projeto colaborativo por discentes, nem sempre a escrita de testes para o projeto estará presente para as novas contribuições. Portanto, além da validação do processo localmente e da ação de CI, é necessário uma validação em conjunta com os orientadores do projeto.

3.4.2 Implantação Contínua

A Implantação Contínua ocorre quando há alterações nas *branches* "develop" e "main". Na develop, somente após aprovação na CI o código é implantado no ambiente de homologação para validação das novas implementações. A main é a ramificação que

representa a versão estável da aplicação, apta a ser disponibilizada em produção. No fluxo de CD, a conexão com o servidor remoto é feita via SSH, utilizando chaves criptográficas armazenadas como variáveis de ambiente seguras (*secrets*).

Os arquivos da nova versão são transferidos para um diretório dentro do repositório remoto. Posteriormente, os links simbólicos respectivos do ambiente são atualizados para apontar para a nova versão da aplicação, conforme a estratégia *Atomic Deployment*. Notificações em tempo real sobre o sucesso ou falha da implantação são enviadas para um canal específico do Slack das operações de CD, promovendo a transparência e o acompanhamento contínuo da operação.

3.4.3 *Rollback* Manual

Complementando o processo de automação, foi desenvolvido um *workflow* específico para reverter a aplicação para a versão anterior em caso de falhas identificadas no ambiente de produção e homologação. Esse mecanismo de *rollback* representa uma salvaguarda importante no ciclo de vida da aplicação, permitindo restaurar rapidamente um estado funcional conhecido, minimizando o tempo de indisponibilidade e os impactos causados por eventuais problemas após uma atualização.

Diferentemente das etapas de CI e CD, o *rollback* não é acionado automaticamente, uma vez que sua execução está condicionada à validação e confirmação manual por parte dos envolvidos do projeto. A decisão por manter essa etapa como um gatilho manual busca evitar reversões precipitadas e dá margem para uma análise rápida dos *logs* e sintomas apresentados antes de optar pela reversão. O fluxo de *rollback*, implementado como um *workflow* separado no GitHub Actions, executa as seguintes ações de maneira automatizada:

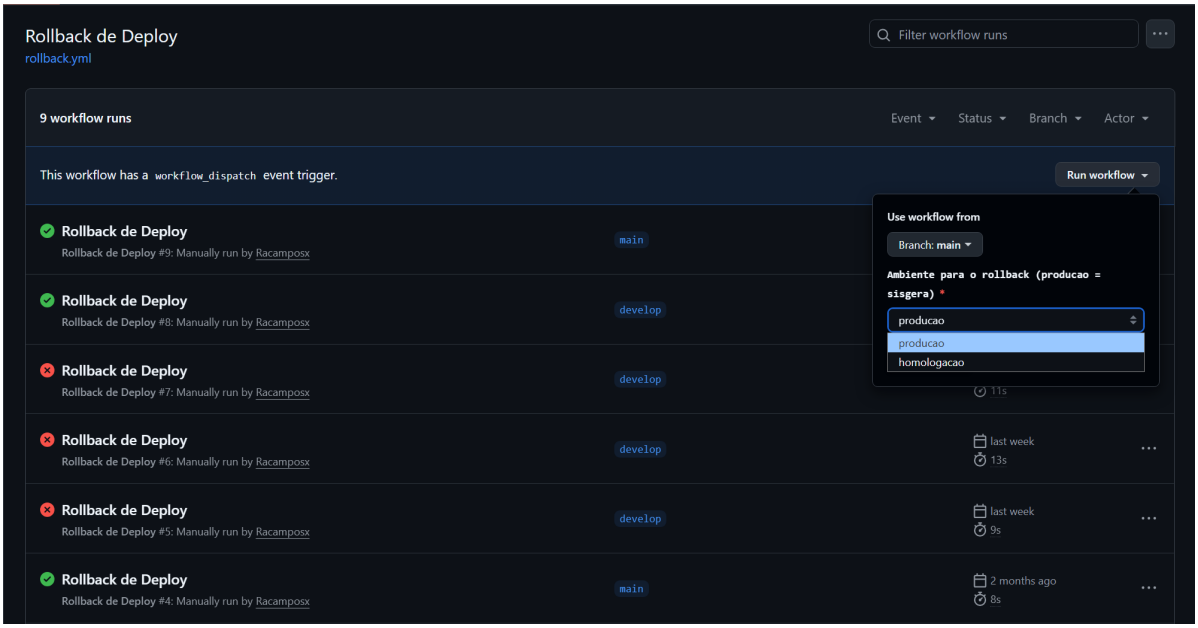
1. Estabelece uma conexão segura com o servidor remoto via SSH, utilizando o mesmo par de chaves previamente configuradas para a etapa de CD.

2. Verifica a existência de um diretório rotulado como *Previous*, que contém a versão imediatamente anterior da aplicação. Esse diretório é criado automaticamente em cada nova implantação como parte da estratégia de versionamento.

3. Atualiza os links simbólicos que definem qual versão está sendo utilizada no ambiente acionado, redirecionando-os para apontar novamente para o diretório da última versão estável.

O mecanismo garante rápida recuperação em caso de erros, sem necessidade de transferências adicionais de arquivos ou intervenção direta no ambiente. Além disso, a equipe é notificada via Slack sobre a execução do procedimento, incluindo o horário, o responsável pela ação e o resultado do processo. A Figura 18 exibe como está configurado o fluxo de *rollback*, podendo especificar em qual ambiente será executada a ação.

Figura 18 – Ação manual de *rollback* no painel de ações do GitHub Actions.



Fonte: Elaborado pelo autor.

3.5 Dificuldades observadas no SisGera

Durante o desenvolvimento do *pipeline* CI/CD, observaram-se impasses que dificultaram sua construção. O SisGera não possui um conjunto de migrações completo até o estado atual do banco de dados em produção, o que impacta diretamente o *pipeline* CI/CD. A Integração Contínua inclui uma etapa de execução das migrações, verificando a consistência da estrutura e a ausência de erros. Para a Implantação Contínua, ter migrações completas é fundamental, pois a aplicação de alterações de banco precisa ocorrer junto ao processo de implantação. Além disso, as migrações viabilizam *rollback* em caso de falhas.

Para contornar essas dificuldades, analisou-se toda a estrutura do *schema* do banco de dados hospedado. Utilizando o phpMyAdmin, ferramenta de administração MySQL, foi possível inspecionar as tabelas, suas colunas e restrições. A Figura 19 exibe

o conjunto de tabelas e registros; nota-se a ausência de uma tabela de migrações, que registraria o histórico de execuções e a evolução do banco.

Figura 19 – Menu principal do phpMyAdmin exibindo o banco de dados do SisGera.

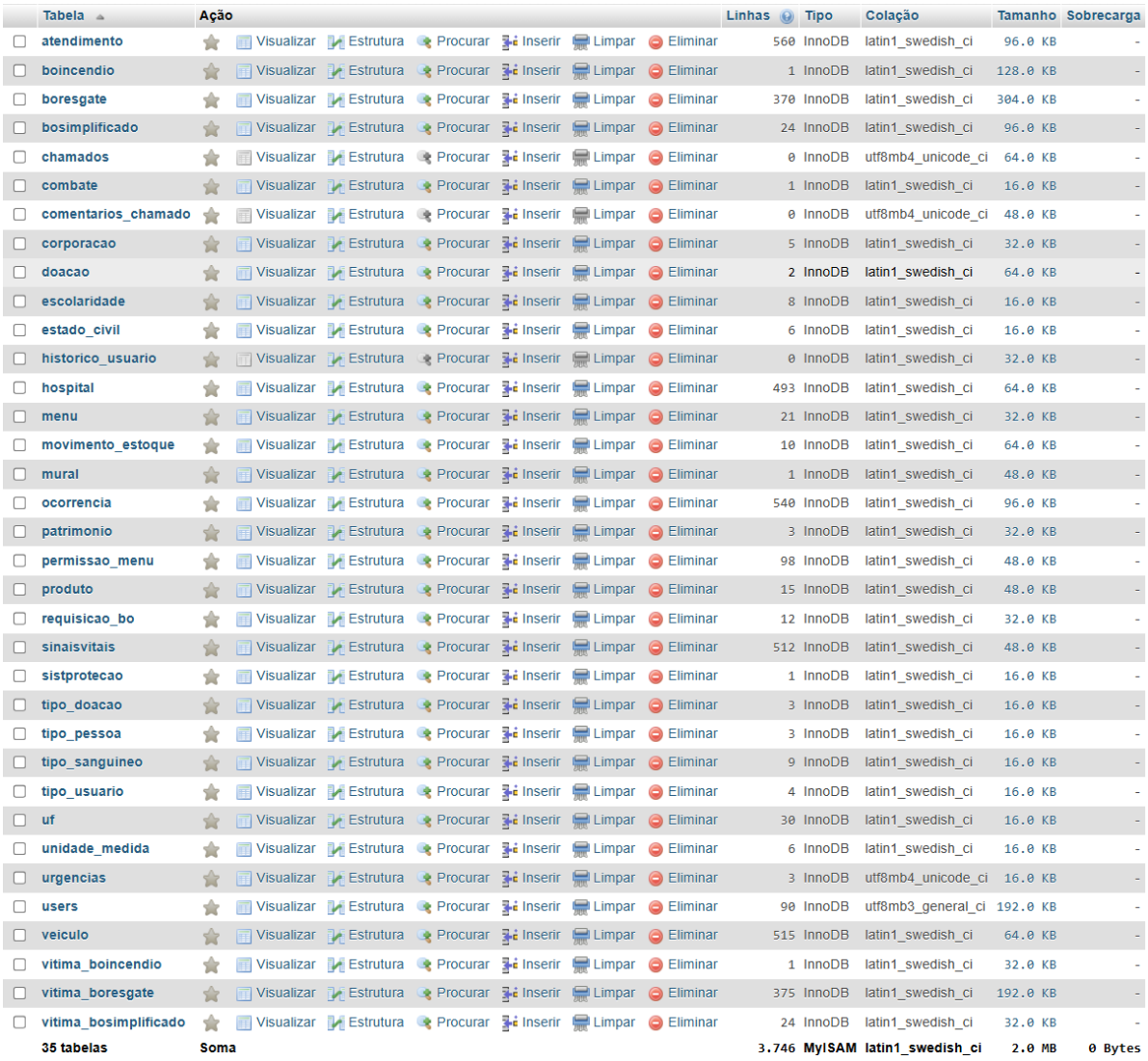
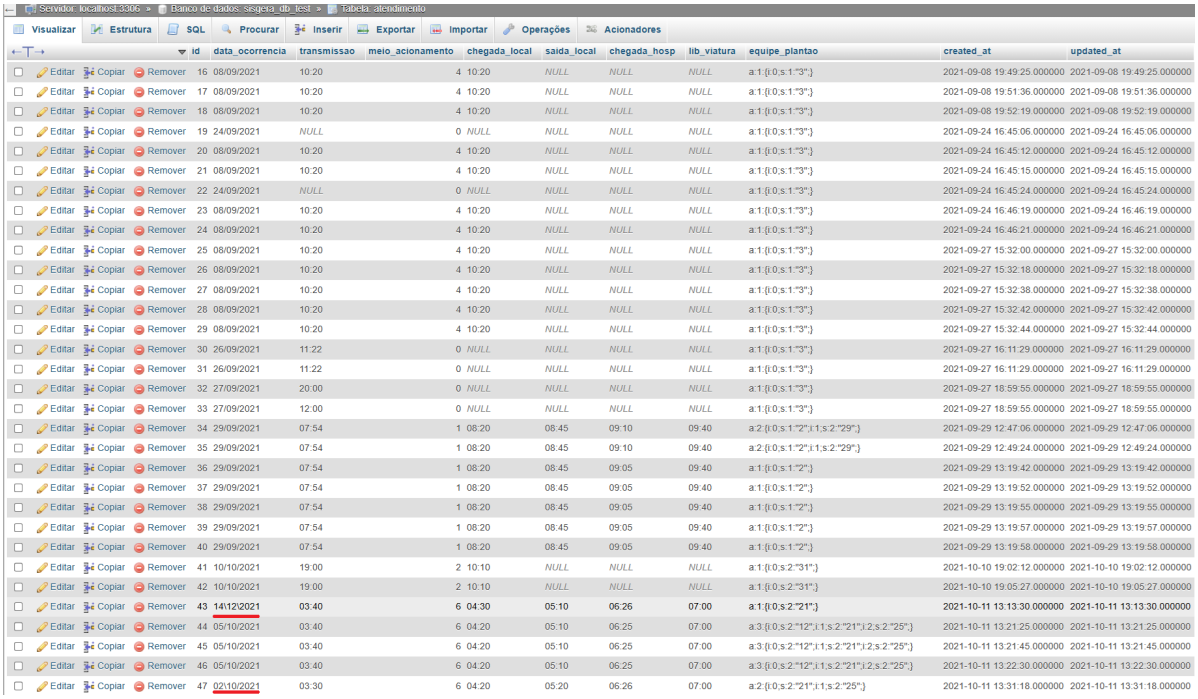


	Tabela	Ação	Linhas	Tipo	Colação	Tamanho	Sobrecarga
☐	atendimento	Visualizar Estrutura Procurar Inserir Limpar Eliminar	560	InnoDB	latin1_swedish_ci	96.0 KB	-
☐	boincendio	Visualizar Estrutura Procurar Inserir Limpar Eliminar	1	InnoDB	latin1_swedish_ci	128.0 KB	-
☐	boresgate	Visualizar Estrutura Procurar Inserir Limpar Eliminar	370	InnoDB	latin1_swedish_ci	304.0 KB	-
☐	bosimplificado	Visualizar Estrutura Procurar Inserir Limpar Eliminar	24	InnoDB	latin1_swedish_ci	96.0 KB	-
☐	chamados	Visualizar Estrutura Procurar Inserir Limpar Eliminar	0	InnoDB	utf8mb4_unicode_ci	64.0 KB	-
☐	combate	Visualizar Estrutura Procurar Inserir Limpar Eliminar	1	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	comentarios_chamado	Visualizar Estrutura Procurar Inserir Limpar Eliminar	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
☐	corporacao	Visualizar Estrutura Procurar Inserir Limpar Eliminar	5	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	doacao	Visualizar Estrutura Procurar Inserir Limpar Eliminar	2	InnoDB	latin1_swedish_ci	64.0 KB	-
☐	escolaridade	Visualizar Estrutura Procurar Inserir Limpar Eliminar	8	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	estado_civil	Visualizar Estrutura Procurar Inserir Limpar Eliminar	6	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	historico_usuario	Visualizar Estrutura Procurar Inserir Limpar Eliminar	0	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	hospital	Visualizar Estrutura Procurar Inserir Limpar Eliminar	493	InnoDB	latin1_swedish_ci	64.0 KB	-
☐	menu	Visualizar Estrutura Procurar Inserir Limpar Eliminar	21	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	movimento_estoque	Visualizar Estrutura Procurar Inserir Limpar Eliminar	10	InnoDB	latin1_swedish_ci	64.0 KB	-
☐	mural	Visualizar Estrutura Procurar Inserir Limpar Eliminar	1	InnoDB	latin1_swedish_ci	48.0 KB	-
☐	ocorrencia	Visualizar Estrutura Procurar Inserir Limpar Eliminar	540	InnoDB	latin1_swedish_ci	96.0 KB	-
☐	patrimonio	Visualizar Estrutura Procurar Inserir Limpar Eliminar	3	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	permissao_menu	Visualizar Estrutura Procurar Inserir Limpar Eliminar	98	InnoDB	latin1_swedish_ci	48.0 KB	-
☐	produto	Visualizar Estrutura Procurar Inserir Limpar Eliminar	15	InnoDB	latin1_swedish_ci	48.0 KB	-
☐	requisicao_bo	Visualizar Estrutura Procurar Inserir Limpar Eliminar	12	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	sinaisvitais	Visualizar Estrutura Procurar Inserir Limpar Eliminar	512	InnoDB	latin1_swedish_ci	48.0 KB	-
☐	sistprotecao	Visualizar Estrutura Procurar Inserir Limpar Eliminar	1	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	tipo_doacao	Visualizar Estrutura Procurar Inserir Limpar Eliminar	3	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	tipo_pessoa	Visualizar Estrutura Procurar Inserir Limpar Eliminar	3	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	tipo_sanguineo	Visualizar Estrutura Procurar Inserir Limpar Eliminar	9	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	tipo_usuario	Visualizar Estrutura Procurar Inserir Limpar Eliminar	4	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	uf	Visualizar Estrutura Procurar Inserir Limpar Eliminar	30	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	unidade_medida	Visualizar Estrutura Procurar Inserir Limpar Eliminar	6	InnoDB	latin1_swedish_ci	16.0 KB	-
☐	urgencias	Visualizar Estrutura Procurar Inserir Limpar Eliminar	3	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
☐	users	Visualizar Estrutura Procurar Inserir Limpar Eliminar	90	InnoDB	utf8mb3_general_ci	192.0 KB	-
☐	veiculo	Visualizar Estrutura Procurar Inserir Limpar Eliminar	515	InnoDB	latin1_swedish_ci	64.0 KB	-
☐	vitima_boincendio	Visualizar Estrutura Procurar Inserir Limpar Eliminar	1	InnoDB	latin1_swedish_ci	32.0 KB	-
☐	vitima_boresgate	Visualizar Estrutura Procurar Inserir Limpar Eliminar	375	InnoDB	latin1_swedish_ci	192.0 KB	-
☐	vitima_bosimplificado	Visualizar Estrutura Procurar Inserir Limpar Eliminar	24	InnoDB	latin1_swedish_ci	32.0 KB	-
	35 tabelas	Soma	3.746	MyISAM	latin1_swedish_ci	2.0 MB	0 Bytes

Fonte: Elaborado pelo autor.

Ao inspecionar individualmente cada tabela, identificaram-se falhas de design estrutural e erros de integridade referencial tanto em produção quanto em homologação. Como exemplo de falha estrutural, na tabela “atendimento” as colunas “data_ocorrencia” e “transmissao” estão definidas como “VARCHAR”. O uso desse tipo, em vez de “DATE” e “TIME”, permite a inserção de valores inválidos e é suscetível a erros. A Figura 20 e a Figura 21 exibem exemplos de dados inconsistentes com os padrões de data e hora.

Figura 20 – Inconsistência de dados da coluna “data_ocorrencia”.



Servidor: localhost:3306 » Banco de dados: segeta_db_test » Tabela: atendimento

Visualizar | Estrutura | SQL | Procurar | Inserir | Exportar | Importar | Operações | Acionadores

	id	data_ocorrencia	transmissao	meio_acionamento	chegada_local	saida_local	chegada_hosp	lib_viatura	equipe_plantao	created_at	updated_at
Editar Copiar Remover	16	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-08 19:49:25.000000	2021-09-08 19:49:25.000000
Editar Copiar Remover	17	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-08 19:51:36.000000	2021-09-08 19:51:36.000000
Editar Copiar Remover	18	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-08 19:52:19.000000	2021-09-08 19:52:19.000000
Editar Copiar Remover	19	24/09/2021	NULL	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:45:06.000000	2021-09-24 16:45:06.000000
Editar Copiar Remover	20	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:45:12.000000	2021-09-24 16:45:12.000000
Editar Copiar Remover	21	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:45:15.000000	2021-09-24 16:45:15.000000
Editar Copiar Remover	22	24/09/2021	NULL	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:45:24.000000	2021-09-24 16:45:24.000000
Editar Copiar Remover	23	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:46:19.000000	2021-09-24 16:46:19.000000
Editar Copiar Remover	24	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-24 16:46:21.000000	2021-09-24 16:46:21.000000
Editar Copiar Remover	25	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 15:32:00.000000	2021-09-27 15:32:00.000000
Editar Copiar Remover	26	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 15:32:18.000000	2021-09-27 15:32:18.000000
Editar Copiar Remover	27	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 15:32:38.000000	2021-09-27 15:32:38.000000
Editar Copiar Remover	28	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 15:32:42.000000	2021-09-27 15:32:42.000000
Editar Copiar Remover	29	08/09/2021	10:20	4	10:20	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 15:32:44.000000	2021-09-27 15:32:44.000000
Editar Copiar Remover	30	26/09/2021	11:22	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 16:11:29.000000	2021-09-27 16:11:29.000000
Editar Copiar Remover	31	26/09/2021	11:22	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 16:11:29.000000	2021-09-27 16:11:29.000000
Editar Copiar Remover	32	27/09/2021	20:00	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 18:59:55.000000	2021-09-27 18:59:55.000000
Editar Copiar Remover	33	27/09/2021	12:00	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"3";}	2021-09-27 18:59:55.000000	2021-09-27 18:59:55.000000
Editar Copiar Remover	34	29/09/2021	07:54	1	08:20	08:45	09:10	09:40	a:2:{i:0;s:1:"2";i:1;s:2:"29";}	2021-09-29 12:47:06.000000	2021-09-29 12:47:06.000000
Editar Copiar Remover	35	29/09/2021	07:54	1	08:20	08:45	09:10	09:40	a:2:{i:0;s:1:"2";i:1;s:2:"29";}	2021-09-29 12:49:24.000000	2021-09-29 12:49:24.000000
Editar Copiar Remover	36	29/09/2021	07:54	1	08:20	08:45	09:05	09:40	a:1:{i:0;s:1:"2";}	2021-09-29 13:19:42.000000	2021-09-29 13:19:42.000000
Editar Copiar Remover	37	29/09/2021	07:54	1	08:20	08:45	09:05	09:40	a:1:{i:0;s:1:"2";}	2021-09-29 13:19:52.000000	2021-09-29 13:19:52.000000
Editar Copiar Remover	38	29/09/2021	07:54	1	08:20	08:45	09:05	09:40	a:1:{i:0;s:1:"2";}	2021-09-29 13:19:55.000000	2021-09-29 13:19:55.000000
Editar Copiar Remover	39	29/09/2021	07:54	1	08:20	08:45	09:05	09:40	a:1:{i:0;s:1:"2";}	2021-09-29 13:19:57.000000	2021-09-29 13:19:57.000000
Editar Copiar Remover	40	29/09/2021	07:54	1	08:20	08:45	09:05	09:40	a:1:{i:0;s:1:"2";}	2021-09-29 13:19:58.000000	2021-09-29 13:19:58.000000
Editar Copiar Remover	41	10/10/2021	19:00	2	10:10	NULL	NULL	NULL	a:1:{i:0;s:2:"31";}	2021-10-10 19:02:12.000000	2021-10-10 19:02:12.000000
Editar Copiar Remover	42	10/10/2021	19:00	2	10:10	NULL	NULL	NULL	a:1:{i:0;s:2:"31";}	2021-10-10 19:05:27.000000	2021-10-10 19:05:27.000000
Editar Copiar Remover	43	14\12\2021	03:40	6	04:30	05:10	06:26	07:00	a:1:{i:0;s:2:"21";}	2021-10-11 13:13:30.000000	2021-10-11 13:13:30.000000
Editar Copiar Remover	44	05/10/2021	03:40	6	04:20	05:10	06:25	07:00	a:3:{i:0;s:2:"12";i:1;s:2:"21";i:2;s:2:"25";}	2021-10-11 13:21:25.000000	2021-10-11 13:21:25.000000
Editar Copiar Remover	45	05/10/2021	03:40	6	04:20	05:10	06:25	07:00	a:3:{i:0;s:2:"12";i:1;s:2:"21";i:2;s:2:"25";}	2021-10-11 13:21:45.000000	2021-10-11 13:21:45.000000
Editar Copiar Remover	46	05/10/2021	03:40	6	04:20	05:10	06:25	07:00	a:3:{i:0;s:2:"12";i:1;s:2:"21";i:2;s:2:"25";}	2021-10-11 13:22:30.000000	2021-10-11 13:22:30.000000
Editar Copiar Remover	47	02\10/2021	03:30	6	04:20	05:20	06:26	07:00	a:2:{i:0;s:2:"21";i:1;s:2:"25";}	2021-10-11 13:31:18.000000	2021-10-11 13:31:18.000000

Fonte: Elaborado pelo autor.

Figura 21 – Inconsistência de dados da coluna “transmissao”.

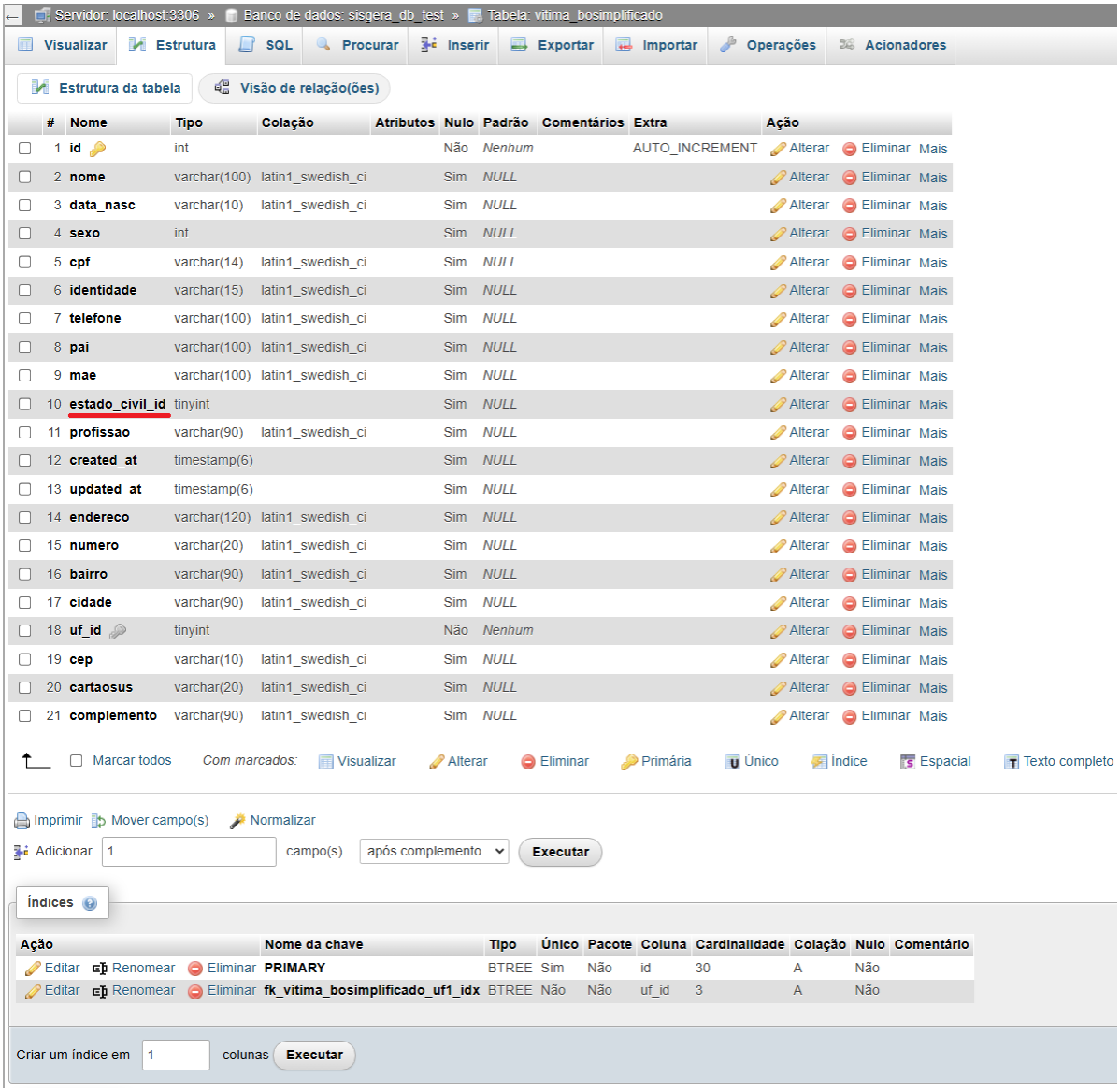
Visualizar | Estrutura | SQL | Procurar | Inserir | Exportar | Importar | Operações | Acionadores

	id	data_ocorrencia	transmissao	meio_acionamento	chegada_local	saida_local	chegada_hosp	lib_viatura	equipe_plantao	created_at	updated_at
Editar Copiar Remover	77	22/08/2019	07:00	1	07:05	07:10	08:30	10:30	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-08-22 22:27:40.708376	2019-08-29 22:11:32.907640
Editar Copiar Remover	78	26/08/2019	16:45	1	17:30	18:08	NULL	21:00	a:1:{i:0;s:1:"2";}	2019-08-28 13:03:17.171357	2021-07-27 22:12:01.505853
Editar Copiar Remover	79	29/08/2019	11:10	1	11:25	11:30	12:00	13:30	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-08-29 21:32:41.756407	2019-08-29 21:33:40.274575
Editar Copiar Remover	80	28/08/2019	18:10	1	18:30	19:20	19:36	19:50	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 22:40:20.543296	2019-08-29 22:40:20.543296
Editar Copiar Remover	81	29/08/2019	18:10	1	NULL	NULL	18:46	NULL	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 22:50:21.010438	2019-08-29 22:50:21.010438
Editar Copiar Remover	82	29/08/2019	18:10	1	NULL	NULL	18:46	NULL	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 22:50:44.345553	2019-08-29 22:50:44.345553
Editar Copiar Remover	83	29/08/2019	18:10	1	NULL	NULL	18:46	NULL	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 22:51:13.767255	2019-08-29 22:51:13.767255
Editar Copiar Remover	84	29/08/2019	18:10	1	NULL	NULL	18:46	NULL	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 22:54:35.457339	2019-08-29 22:54:35.457339
Editar Copiar Remover	85	28/08/2019	18:10	1	NULL	NULL	18:42	NULL	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:1:"5";}	2019-08-29 23:03:06.978205	2019-08-29 23:03:06.978205
Editar Copiar Remover	86	05/09/2019	06:30	1	06:40	07:20	07:50	09:30	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-10 23:08:21.479710	2019-09-10 23:08:21.479710
Editar Copiar Remover	87	30/08/2019	10:30	1	11:00	11:05	NULL	14:00	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-10 23:24:25.653008	2019-09-10 23:24:25.653008
Editar Copiar Remover	88	20/09/2019	10:45	1	11:20	11:40	12:10	12:20	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:01:45.012165	2019-09-21 13:01:45.012165
Editar Copiar Remover	89	21/09/2019	10:45	1	11:20	11:40	12:10	12:20	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:02:45.713546	2019-09-21 13:02:45.713546
Editar Copiar Remover	90	20/09/2019	10:20	1	11:20	11:40	12:00	12:10	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:28:42.348517	2019-09-21 13:28:42.348517
Editar Copiar Remover	91	21/09/2019	10:20	1	11:20	11:40	12:00	12:10	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:32:14.472108	2019-09-21 13:32:14.472108
Editar Copiar Remover	92	21/09/2019	10:20	1	11:20	11:40	12:00	12:10	a:1:{i:0;s:1:"2";}	2019-09-21 13:34:46.502524	2019-09-21 13:34:46.502524
Editar Copiar Remover	93	21/09/2019	09:50	1	10:00	10:05	NULL	13:00	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:50:23.523986	2019-09-21 13:50:23.523986
Editar Copiar Remover	94	21/09/2019	09:50	1	10:00	10:05	NULL	13:00	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-09-21 13:51:10.092993	2019-09-21 13:51:10.092993
Editar Copiar Remover	95	29/09/2019	20:40	1	20:45	20:50	21:15	22:30	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-10-19 17:29:53.802141	2019-10-19 17:29:53.802141
Editar Copiar Remover	96	27/10/2019	18:00	4	18:40	21:30	NULL	22:00	a:3:{i:0;s:1:"2";i:1;s:1:"4";i:2;s:2:"10";}	2019-10-28 01:32:21.912775	2019-10-28 01:34:02.013778
Editar Copiar Remover	97	05/11/2019	10:30	1	10:40	20:50	11:30	13:00	a:2:{i:0;s:1:"2";i:1;s:1:"4";}	2019-11-05 14:10:29.043529	2019-11-05 14:10:29.043529
Editar Copiar Remover	98	19/11/2019	NULL	0	NULL	NULL	NULL	NULL	a:1:{i:0;s:2:"14";}	2019-11-19 03:44:52.384541	2019-11-19 03:44:52.384541
Editar Copiar Remover	99	29/05/2020	17:04	1	17:10	17:35	NULL	18:00	a:3:{i:0;s:1:"9";i:1;s:2:"20";i:2;s:2:"21";}	2020-05-30 01:18:39.434818	2020-05-30 01:18:39.434818
Editar Copiar Remover	100	29/05/2020	17:04	1	17:10	17:35	NULL	18:00	a:3:{i:0;s:1:"9";i:1;s:2:"20";i:2;s:2:"21";}	2020-05-30 01:28:34.238352	2020-05-30 01:28:34.238352
Editar Copiar Remover	102	29/05/2020	NULL	1	14:50	15:00	16:29	16:40	a:2:{i:0;s:1:"2";i:1;s:2:"20";}	2020-05-30 01:50:03.315061	2020-05-30 01:50:03.315061
Editar Copiar Remover	103	11/05/2020	00:09	1	00:19	00:37	00:55	01:00	a:3:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"21";}	2020-06-02 18:50:26.557667	2020-06-02 18:50:26.557667
Editar Copiar Remover	104	03/06/2020	09:55	1	10:00	NULL	11:03	11:10	a:2:{i:0;s:1:"2";i:1;s:2:"21";}	2020-06-04 21:53:35.948194	2020-06-04 21:53:35.948194
Editar Copiar Remover	105	13/06/2020	21:00	1	21:05	21:15	21:20	21:38	a:3:{i:0;s:1:"2";i:1;s:2:"20";i:2;s:2:"21";}	2020-06-14 00:47:47.454192	2020-06-14 00:47:47.454192
Editar Copiar Remover	106	09/07/2020	15:13	1	15:40	16:55	NULL	17:20	a:2:{i:0;s:1:"2";i:1;s:2:"10";}	2020-07-09 22:04:45.939235	2020-07-09 22:04:45.939235
Editar Copiar Remover	107	18/07/2020	11:14	1	11:50	12:00	12:40	13:00	a:3:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"20";}	2020-07-19 23:48:40.077972	2020-07-19 23:48:40.077972
Editar Copiar Remover	108	19/07/2020	1_:__	1	NULL	NULL	NULL	NULL	a:1:{i:0;s:1:"2";}	2020-07-19 23:51:19.165660	2020-07-19 23:51:19.165660
Editar Copiar Remover	109	19/07/2020	13:55	1	14:15	14:45	15:00	15:45	a:3:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"20";}	2020-07-20 00:01:18.038321	2020-07-20 00:01:18.038321
Editar Copiar Remover	110	26/07/2020	22:00	1	22:05	22:30	22:34	23:00	a:3:{i:0;s:1:"9";i:1;s:2:"10";i:2;s:2:"21";}	2020-07-27 23:09:01.423285	2020-07-27 23:09:01.423285
Editar Copiar Remover	111	28/08/2020	NULL	1	20:00	20:15	20:51	21:00	a:5:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"12";i:3;s:2...	2020-08-29 23:09:26.461333	2020-08-29 23:09:26.461333
Editar Copiar Remover	112	29/08/2020	15:44	1	15:50	16:00	16:55	18:00	a:2:{i:0;s:1:"2";i:1;s:2:"21";}	2020-08-29 23:15:37.539030	2020-08-29 23:15:37.539030
Editar Copiar Remover	113	28/08/2020	NULL	1	20:00	20:15	20:51	21:00	a:5:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"12";i:3;s:2...	2020-08-29 23:25:09.872545	2020-08-29 23:59:38.238971
Editar Copiar Remover	114	28/08/2020	NULL	1	20:00	21:50	21:50	22:00	a:5:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"12";i:3;s:2...	2020-08-29 23:59:04.735631	2020-08-29 23:59:04.735631
Editar Copiar Remover	115	29/08/2020	NULL	1	13:00	13:30	14:18	15:00	a:5:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"11";i:3;s:2...	2020-08-30 00:01:25.226248	2020-08-30 00:01:25.226248
Editar Copiar Remover	116	29/08/2020	NULL	1	13:00	13:30	14:18	15:00	a:5:{i:0;s:1:"2";i:1;s:2:"10";i:2;s:2:"11";i:3;s:2...	2020-08-30 00:07:13.367476	2020-08-30 00:07:13.367476

Fonte: Elaborado pelo autor.

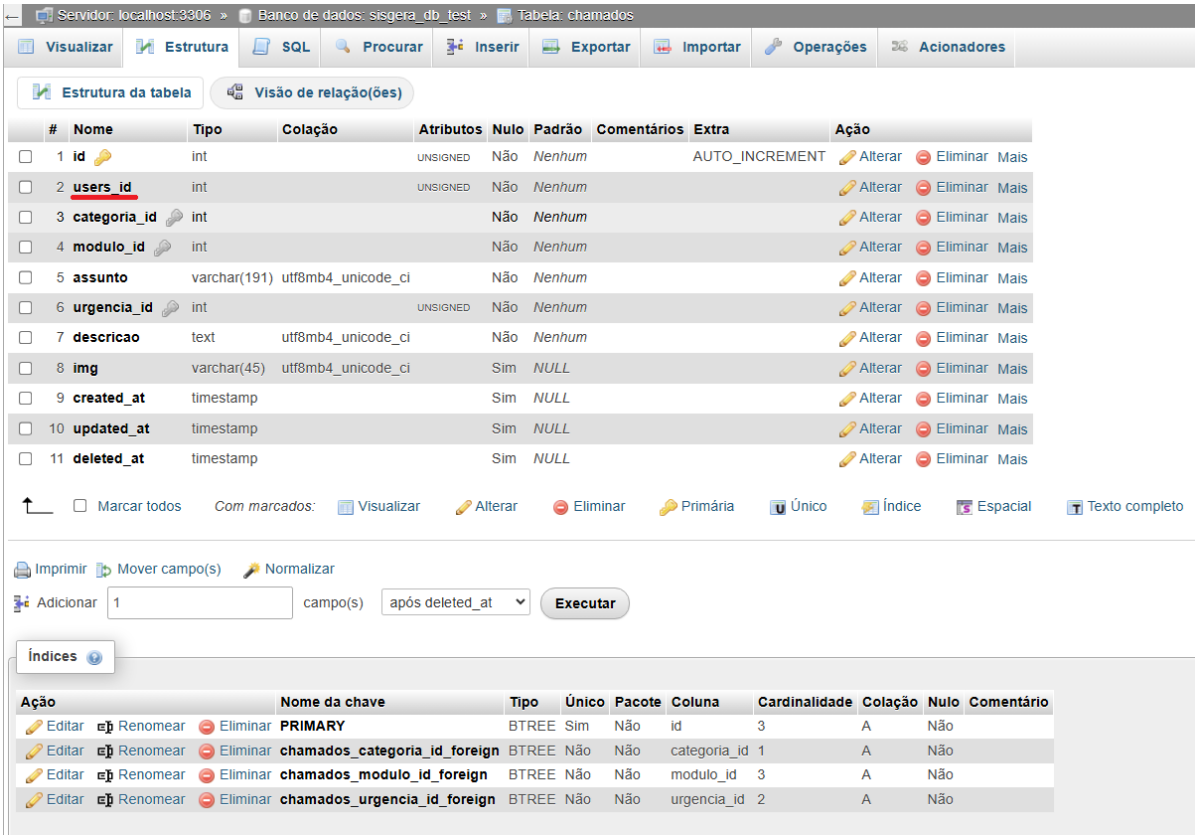
Como erros de integridade, constatou-se que as tabelas “vitima_bosimplificado”, “vitima_boincendio”, “vitima_boresgate” e “chamados” possuem colunas com identificadores que implicam relação, mas sem a criação da restrição de chave estrangeira (*Foreign Key*) que a garante. As tabelas de “vítimas” possuem a coluna “estado_civil_id” e “chamados” possui a coluna “user_id”. A Figura 22 e Figura 23 exibem a estrutura dessas tabelas e as restrições existentes.

Figura 22 – Erro de restrição de integridade na tabela “vitima_bosimplificado”.



Fonte: Elaborado pelo autor.

Figura 23 – Erro de restrição de integridade na tabela “chamados”.



Fonte: Elaborado pelo autor.

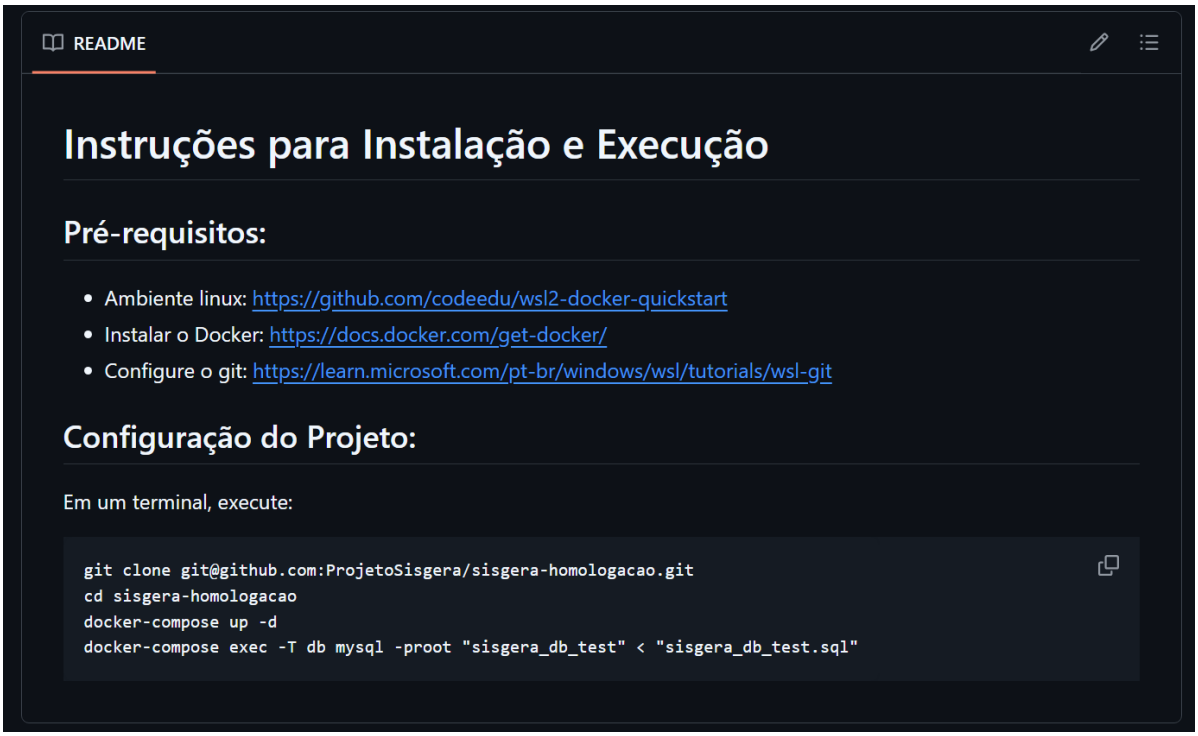
Após revisar a estrutura de ambos os ambientes (homologação e produção), foram desenvolvidos *scripts* de migração para cada tabela. Isso exigiu separar as tabelas em dois grupos: as sem dependências externas e as com dependências. Iniciou-se pelas tabelas independentes e, em seguida, migraram-se as demais. As migrações foram executadas e validadas localmente, com o devido povoamento do banco. Importa destacar que as migrações desenvolvidas reproduzem fielmente a estrutura atual, inclusive as falhas estruturais e os erros de integridade referidos. Correções não foram aplicadas neste momento, pois devem partir da representação vigente do banco de dados.

3.6 Guia de uso do *pipeline* CI/CD e de boas práticas ao projeto

Para garantir o uso eficaz do *pipeline* CI/CD implementado no projeto SisGera, foi realizada uma reestruturação da documentação do sistema. A versão anterior apresentava limitações tanto em conteúdo quanto em organização, dificultando a integração de novos

colaboradores. A Figura 24 mostra como era a antiga documentação como toda. A nova documentação foi projetada para centralizar as informações, proporcionando uma visão clara do funcionamento do projeto, de seus processos internos e do histórico de desenvolvimento.

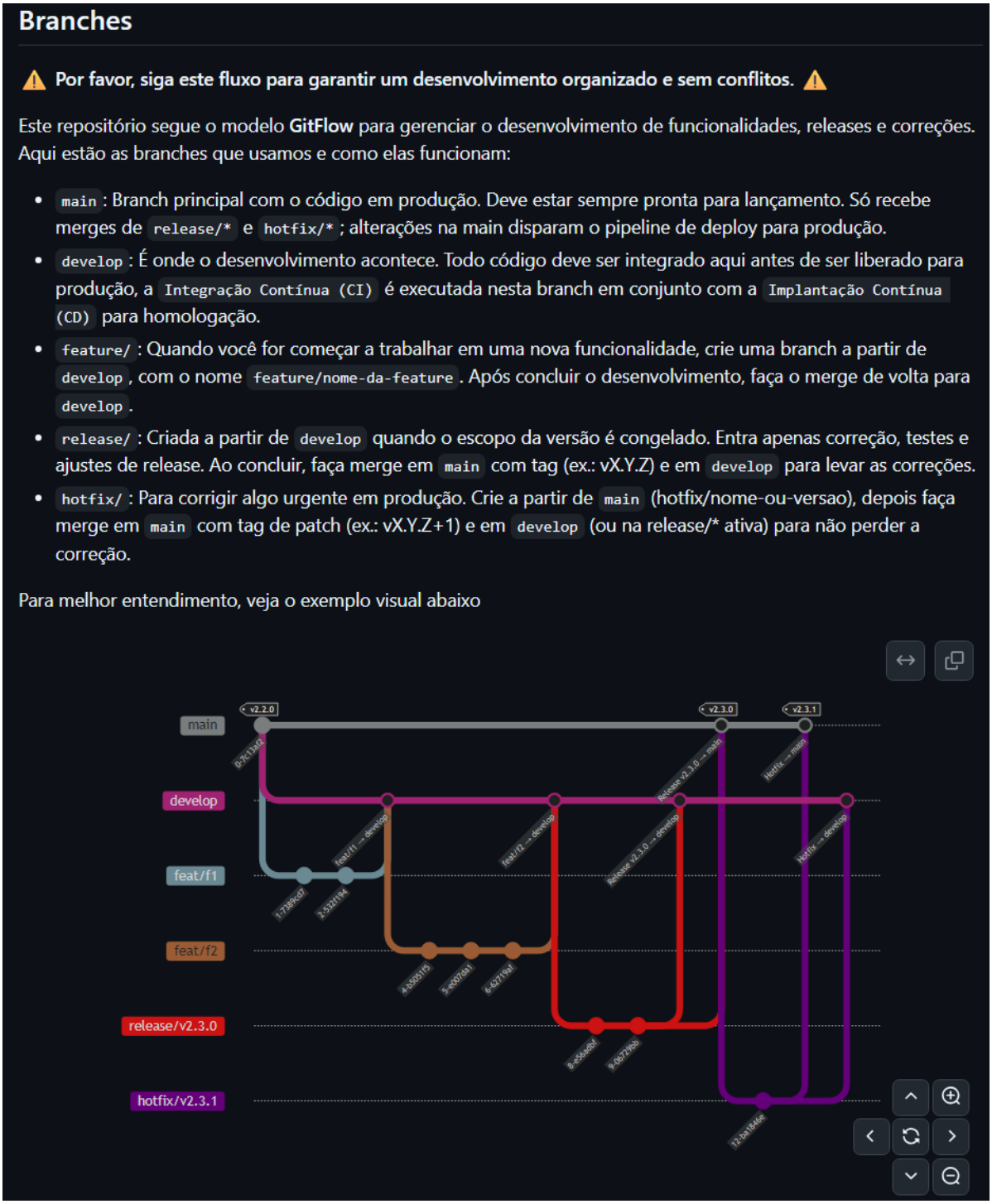
Figura 24 – Antiga documentação do SisGera.



Fonte: Elaborado pelo autor.

Sua orientação parte do modelo de gerenciamento de branches GitFlow. A escolha por este modelo se justifica pela natureza do SisGera, sendo um projeto colaborativo acadêmico que evolui por meio de contribuições bem definidas em certos intervalos de tempo. A existência de dois ambientes operacionais distintos, pode levar a separação da branch “develop” para direcionar novas implementações a serem homologadas rapidamente ao passarem pelo processo de CI. O novo código integrado entraria em processo de CD para o domínio de homologação. A Figura 25 retrata como o GitFlow é composto, seguindo suas estruturas de branches e também com exemplo visual.

Figura 25 – Introdução do GitFlow na documentação do SisGera.

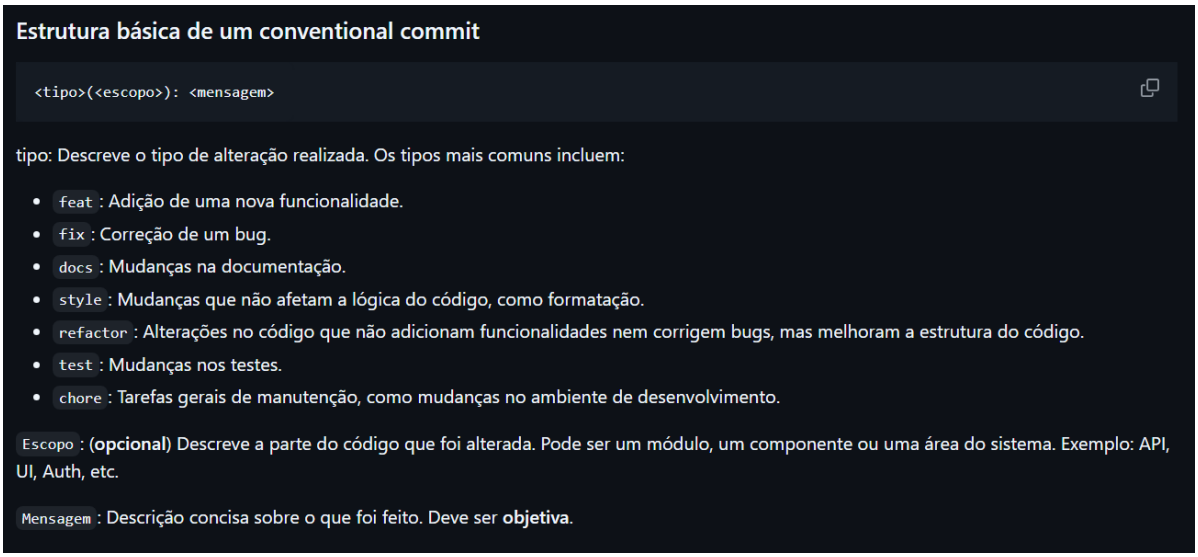


Fonte: Elaborado pelo autor.

Para além do gerenciamento de branches, a documentação institui a adoção do

padrão *Conventional Commits* como uma boa prática essencial. Essa convenção estabelece um formato estruturado para as mensagens de *commit*, composto por tipo, escopo e uma descrição concisa. A recomendação dessa prática visa criar um histórico de contribuições mais claro e organizado. A padronização não apenas facilita a compreensão da evolução do código por parte da equipe, mas também habilita a automação de tarefas importantes, como a geração de registros de alterações (*changelogs*). A Figura 26 mostra como é utilizado essa convenção e traz as definições de sua estrutura.

Figura 26 – *Conventional Commits* sendo abordado na documentação do SisGera.



Fonte: Elaborado pelo autor.

Com o intuito de acelerar a integração de novos membros à equipe, o guia também inclui instruções completas para a preparação do ambiente de desenvolvimento local. São abordadas desde a configuração inicial do Git e a clonagem do repositório até a instalação das ferramentas utilizadas no projeto. Para garantir flexibilidade, são apresentados dois métodos de execução do sistema: o servidor nativo do Laravel, chamado “Artisan”, e um ambiente em *container* utilizando Docker. Essas alternativas visam assegurar que qualquer desenvolvedor consiga reproduzir o ambiente de maneira consistente e com menor barreira técnica.

4 Resultados

Após a análise e seleção das ferramentas apresentadas no Capítulo 2, que culminou na escolha do GitHub Actions como a solução mais adequada para o SisGera, este capítulo se dedica a apresentar os resultados práticos obtidos. As observações a seguir referem-se aos fluxos documentados e às evidências coletadas ao contexto dos objetivos específicos delineados para este trabalho.

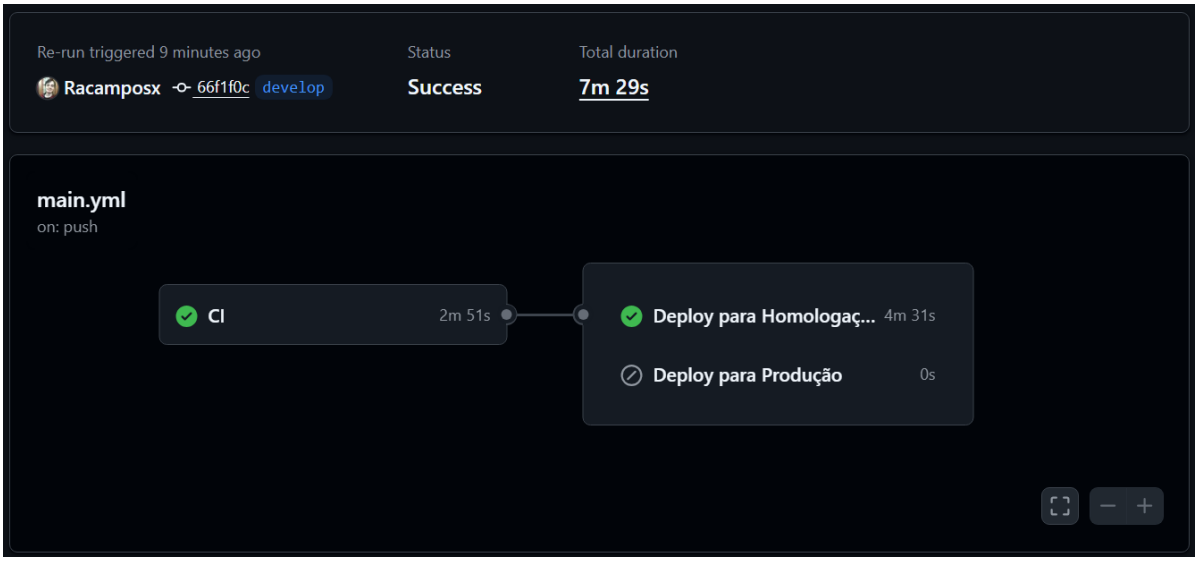
4.1 O *pipeline* CI/CD em operação

No escopo observado ao SisGera, os fluxos executados indicam automação de tarefas repetitivas, maior padronização do processo de entrega, melhoria na rastreabilidade e da comunicação entre os colaboradores envolvidos no projeto. Tais efeitos foram inferidos a partir da sequência das etapas (CI e CD) e dos registros do *pipeline*.

4.1.1 Execução da etapa de Integração Contínua

A etapa de Integração Contínua foi observada em execução quando ocorreram *merges* para as *branches* “develop” e “main”, realizando validações sequenciais para verificar a integridade do código nos casos testados. A Figura 27 ilustra um fluxo completo disparado a partir de “develop”, após o qual o código pôde ser disponibilizado ao servidor de homologação nesse cenário específico.

Figura 27 – Execução bem sucedida do fluxo a partir de “develop”.



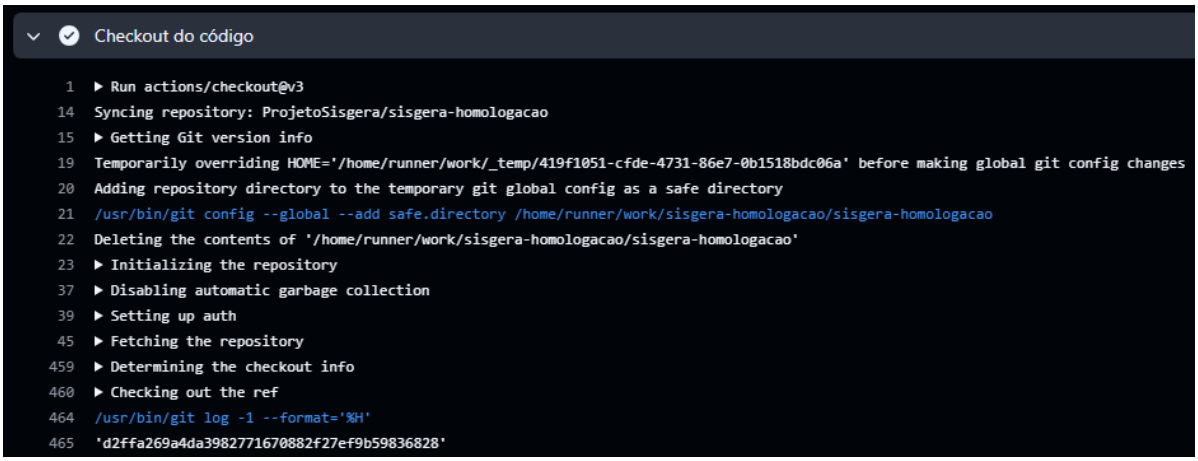
Fonte: Elaborado pelo autor.

No detalhamento dessas execuções, observa-se que ela se inicia com a preparação do executor do GitHub Actions; a Figura 28 ilustra essa preparação, bem como a obtenção de *actions* comunitárias. Em seguida, realiza-se a obtenção do código do repositório (Figura 29). Na Figura 30, configura-se na máquina de execução a versão do PHP utilizada no projeto. Depois, procede-se à validação das dependências com o Composer (Figura 31). A próxima tarefa, mostrada na Figura 32, corresponde à análise estática do código-fonte, com o objetivo de verificar aderência a padrões do PHP, como PSR-1 e PSR-2. A Figura 33 apresenta a construção da aplicação via Docker. Na sequência, executam-se os *scripts* de migração (Figura 34), verificando, neste contexto, a aplicação das alterações de esquema. Por fim, quando existentes, os testes unitários são executados automaticamente pelo *pipeline*, refletindo a cobertura atualmente disponível.

Figura 28 – Preparação do executor do GitHub Actions.

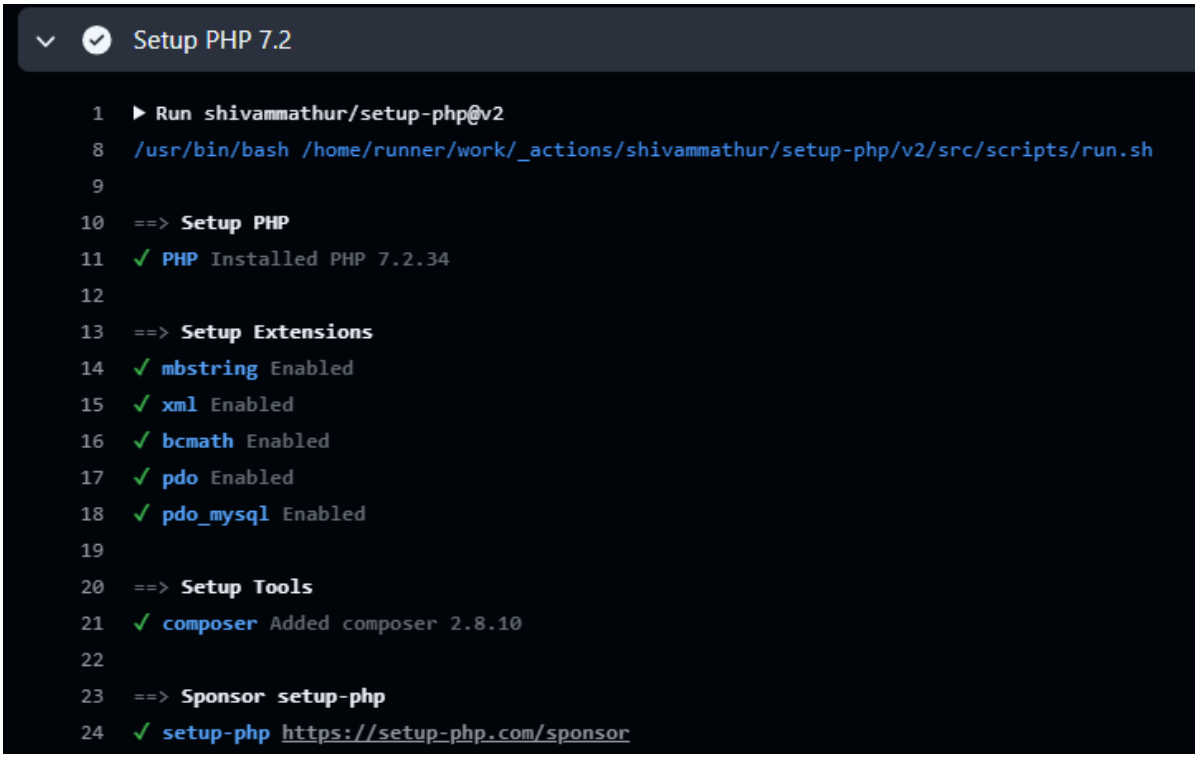
Fonte: Elaborado pelo autor.

Figura 29 – Processo de *checkout*.



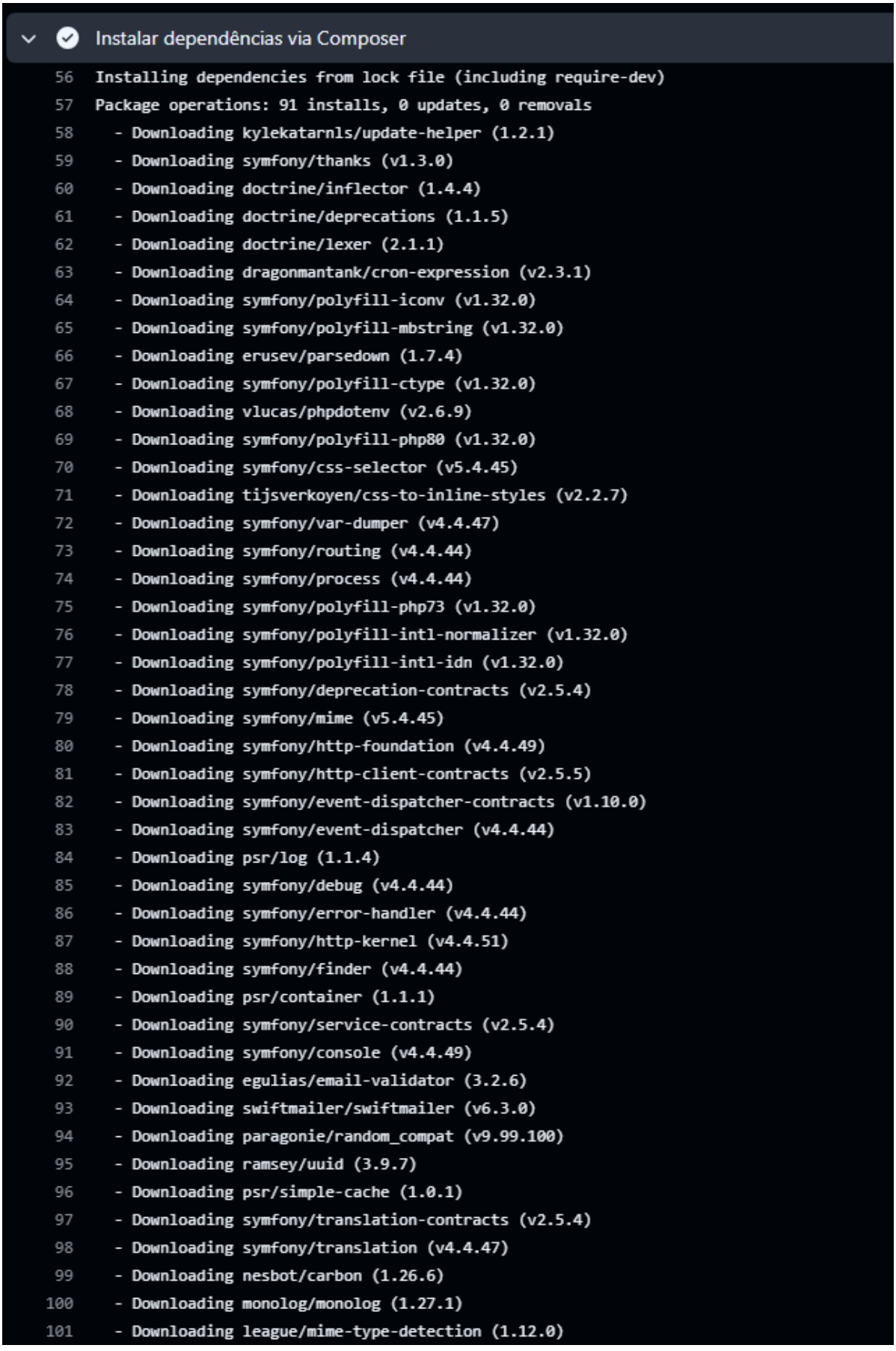
Fonte: Elaborado pelo autor.

Figura 30 – Instalação do PHP na máquina de execução.



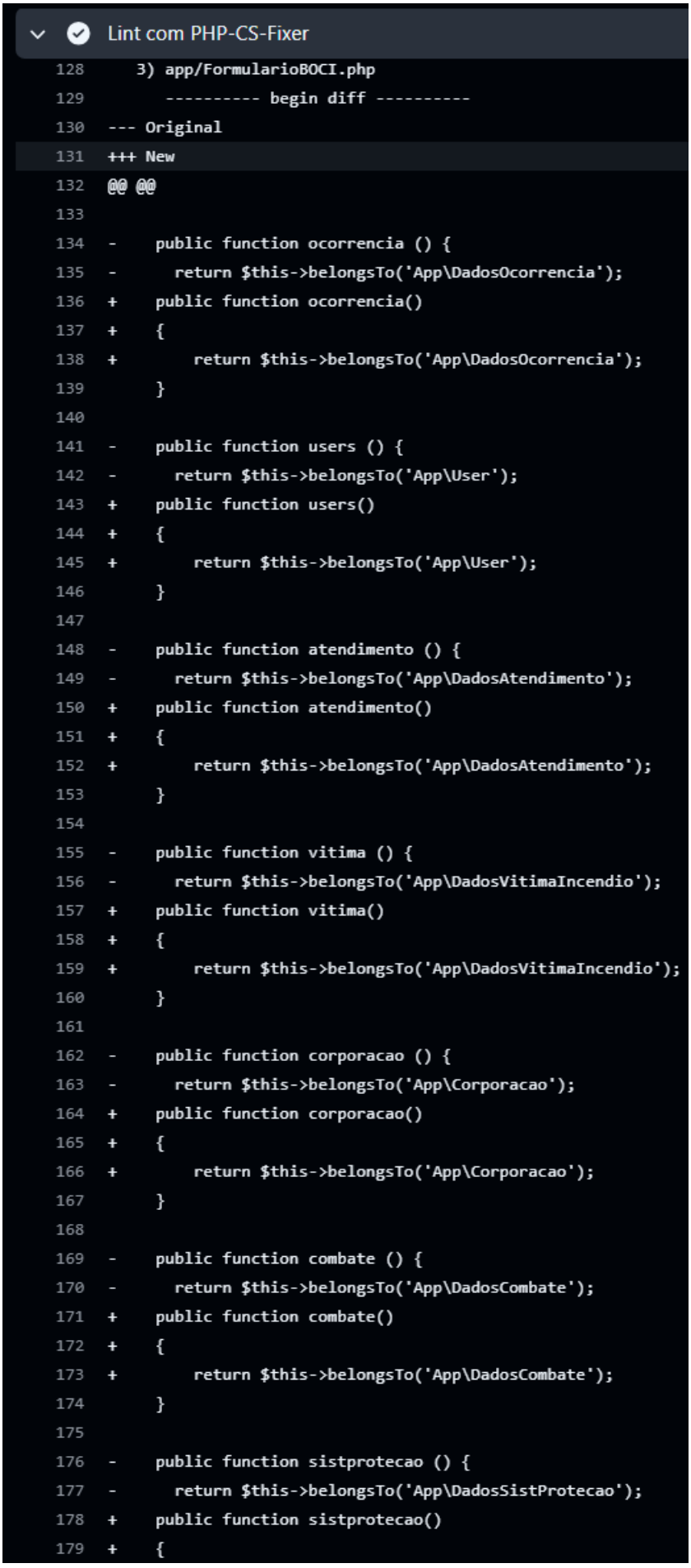
Fonte: Elaborado pelo autor.

Figura 31 – Instalação das dependências do projeto.



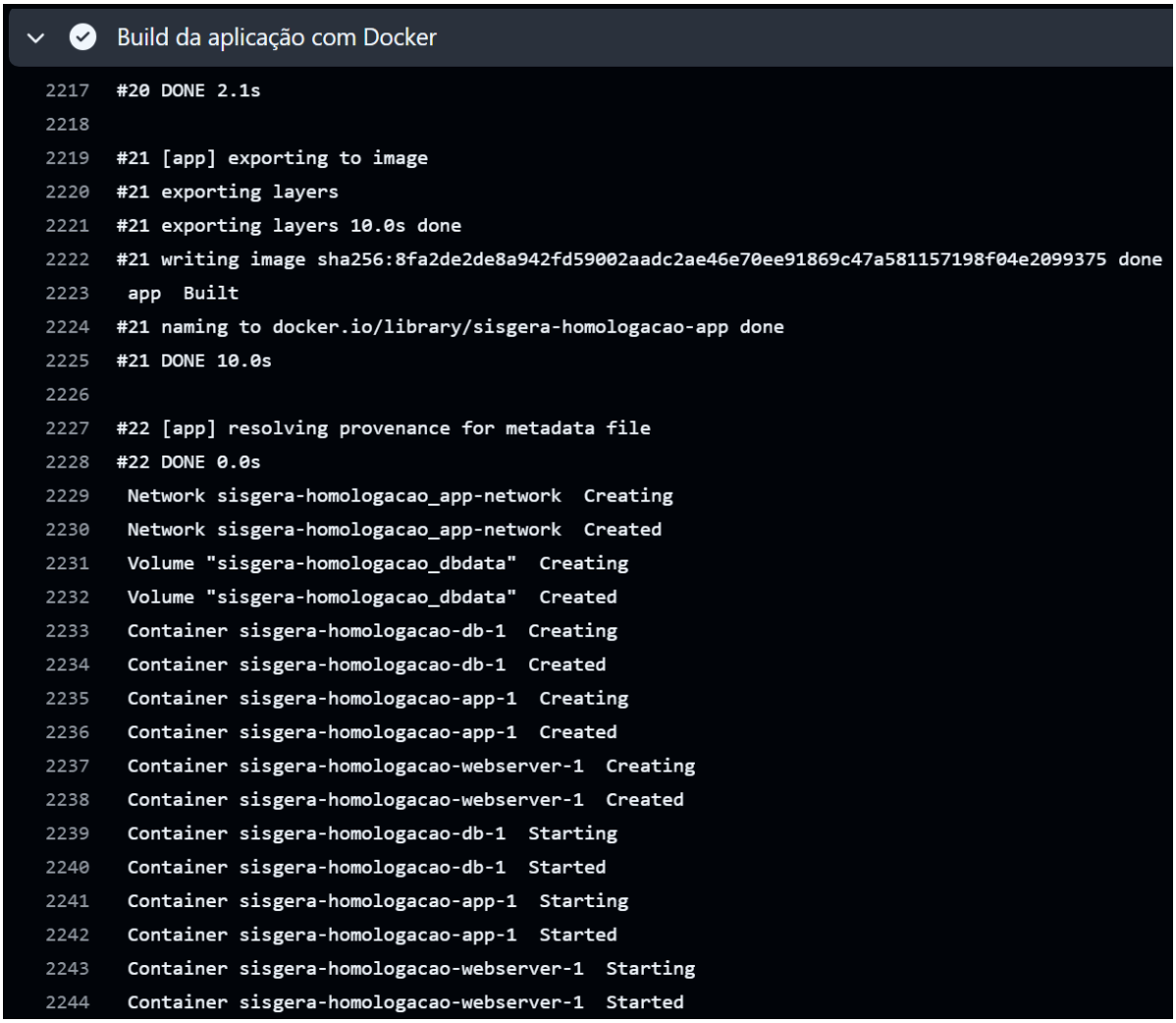
Fonte: Elaborado pelo autor.

Figura 32 – Analise estática do código fonte.



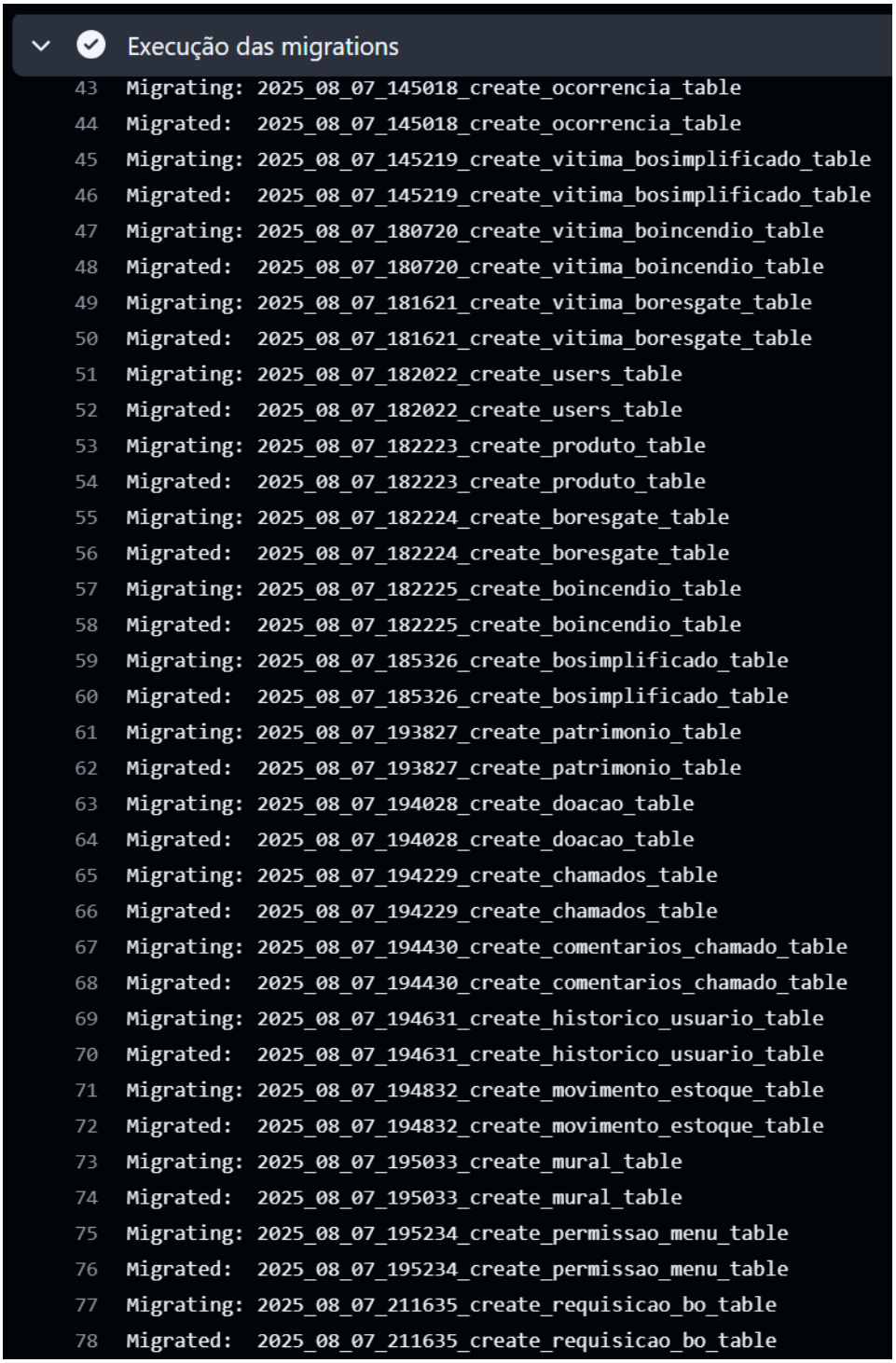
Fonte: Elaborado pelo autor.

Figura 33 – Construção da aplicação via Docker.



Fonte: Elaborado pelo autor.

Figura 34 – Execução dos scripts de migrações.



Fonte: Elaborado pelo autor.

4.1.2 Execução da etapa de Implantação Contínua

Assim como a etapa de CI é ativada quando há interações de merge para as branches de "develop" e "main", o mesmo ocorre para a etapa de CD. Seu fluxo pode ocorrer em dois ambientes, homologação e produção. A distinção em como essa etapa ocorre para ambos os ambientes está relacionada na separação de seus diretórios e a condição de ativação. A Figura 35 mostra um fluxo bem sucedido a partir de "main".

Figura 35 – Execução bem sucedida do fluxo a partir de "main".

Fonte: Elaborado pelo autor.

A etapa de CD inicia-se com a preparação do executor GitHub Actions e a instalação das *actions* comunitárias, responsáveis pelo *checkout* do código do repositório e pela configuração da conexão SSH com o servidor, conforme ilustrado na Figura 36. Em seguida, o código-fonte é obtido no servidor (Figura 37) e procede-se à compactação dos arquivos essenciais para a execução da aplicação (Figura 38). Nas etapas posteriores, a conexão SSH é configurada e o servidor é adicionado à lista de "*known hosts*", permitindo que o executor reconheça o servidor do SisGera, como mostrado nas Figura 39 e Figura 40. Após a configuração e estabelecimento da conexão SSH, é realizado a criação dos diretórios específicos para o ambiente e envio do projeto. Essas etapas são exibidas na Figura 41. Por fim, ocorre a descompactação do projeto no destino e inicia-se o *Atomic deployment*, com a atualização dos links simbólicos.

Figura 36 – Preparação do executor do GitHub Actions.

Fonte: Elaborado pelo autor.

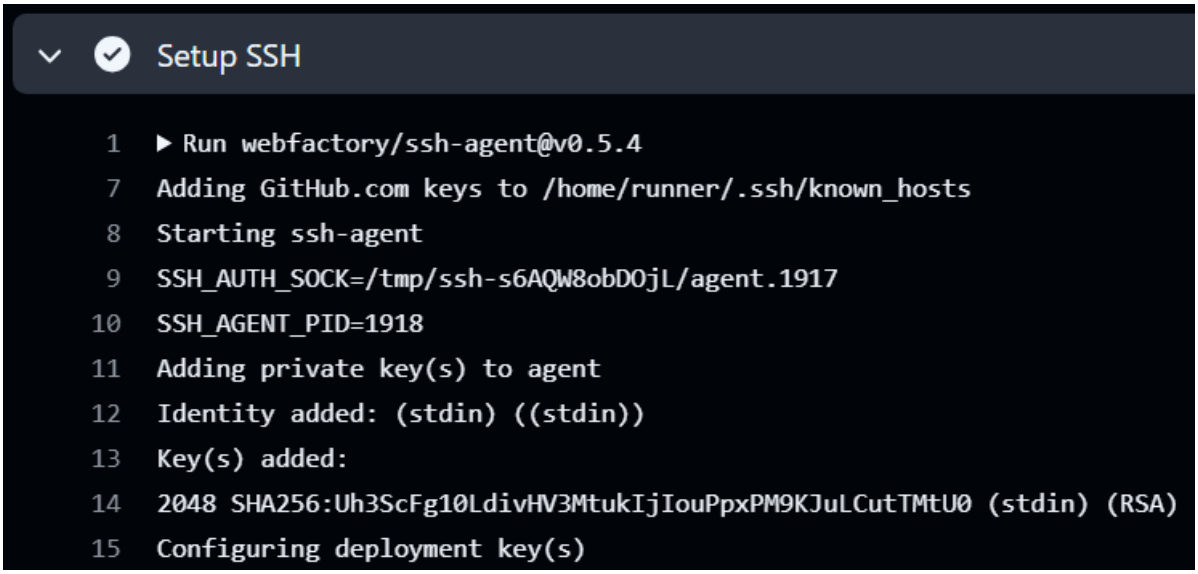
Figura 37 – Processo de *checkout*.

Fonte: Elaborado pelo autor.

Figura 38 – Compactação do projeto com arquivos essenciais.

Fonte: Elaborado pelo autor.

Figura 39 – Configuração da *action* de SSH.

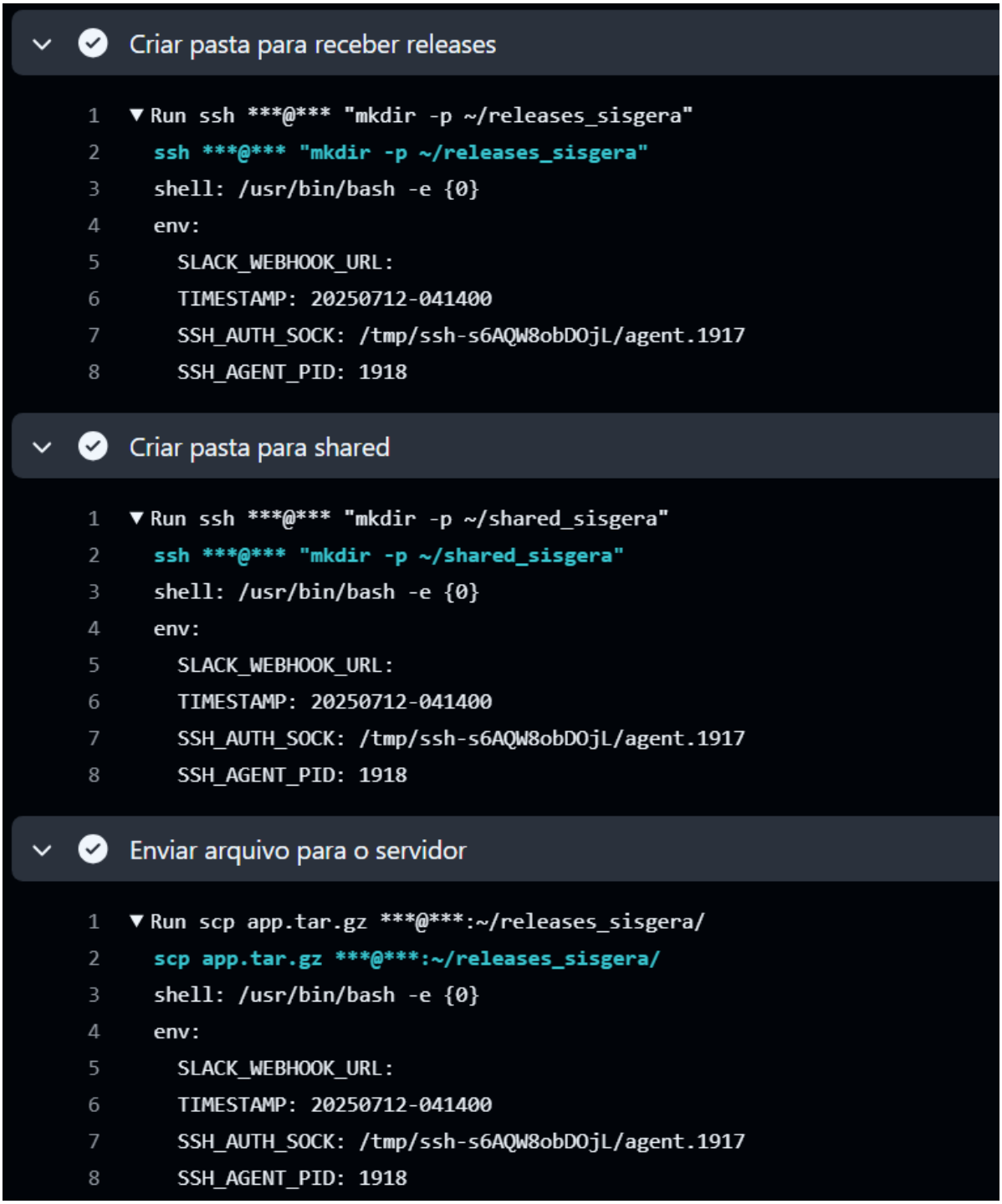


Fonte: Elaborado pelo autor.

Figura 40 – Configurando “known hosts”.

Fonte: Elaborado pelo autor.

Figura 41 – Passos conectados ao servidor: (a) criação do diretório *releases* (se necessário); (b) criação do diretório *shared* (se necessário); (c) envio do pacote por *Secure Copy Protocol* (SCP).



Fonte: Elaborado pelo autor.

Figura 42 – Descompactação e início do processo do *atomic deployment*.

Executar deploy remoto em Produção

```
1  ▼ Run ssh ***@*** << 'EOF'
2    ssh ***@*** << 'EOF'
3      cd ~/releases_homologacao
4
5      # Apagar release Previous, se existir
6      [ -d Previous ] && rm -rf Previous
7
8      # Renomear Current para Previous, se existir
9      [ -d Current ] && mv Current Previous
10
11     # Descompactar nova release
12     tar -xzf app.tar.gz
13     rm -f app.tar.gz
14
15     NEW_RELEASE=$(find . -maxdepth 1 -type d -name "Current-*" ! -name "Current" | head -n 1)
16
17     if [ -z "$NEW_RELEASE" ]; then
18       echo "Erro: nova release com prefixo Current-* não encontrada."
19       exit 1
20     fi
21
22     # Renomear para Current
23     mv "$NEW_RELEASE" Current
24
25     cd Current
26
27     # Instalar dependências
28
29     composer install --no-dev || exit -1
30
31     # Linkar .env e storage
32     ln -fs ../../shared_homologacao/.env .env
33     ln -snf ../../shared_homologacao/storage storage
34
35     # Atualizar symlinks em public_html
36     ln -fs  ~/releases_homologacao/Current/public/.htaccess        ~/public_html/homologacao/.htaccess
37     ln -nfs ~/releases_homologacao/Current/public/css              ~/public_html/homologacao/css
38     ln -nfs ~/releases_homologacao/Current/public/dist             ~/public_html/homologacao/dist
39     ln -nfs ~/releases_homologacao/Current/public/js               ~/public_html/homologacao/js
40     ln -nfs ~/releases_homologacao/Current/public/error_log        ~/public_html/homologacao/error_log
41     ln -nfs ~/releases_homologacao/Current/public/manifest         ~/public_html/homologacao/manifest
42     ln -fs  ~/releases_homologacao/Current/public/index.php        ~/public_html/homologacao/index.php
43     ln -fs  ~/releases_homologacao/Current/public/favicon.ico      ~/public_html/homologacao/favicon.ico
44     ln -fs  ~/releases_homologacao/Current/public/robots.txt       ~/public_html/homologacao/robots.txt
45     ln -fs  ~/releases_homologacao/Current/public/serviceworker.js ~/public_html/homologacao/serviceworker.js
46     ln -nfs ~/releases_homologacao/Current/public/uploads          ~/public_html/homologacao/uploads
47     ln -fs  ~/releases_homologacao/Current/public/web.config       ~/public_html/homologacao/web.config
48
49     echo "Deploy em HOMOLOGAÇÃO concluído."
50   EOF
```

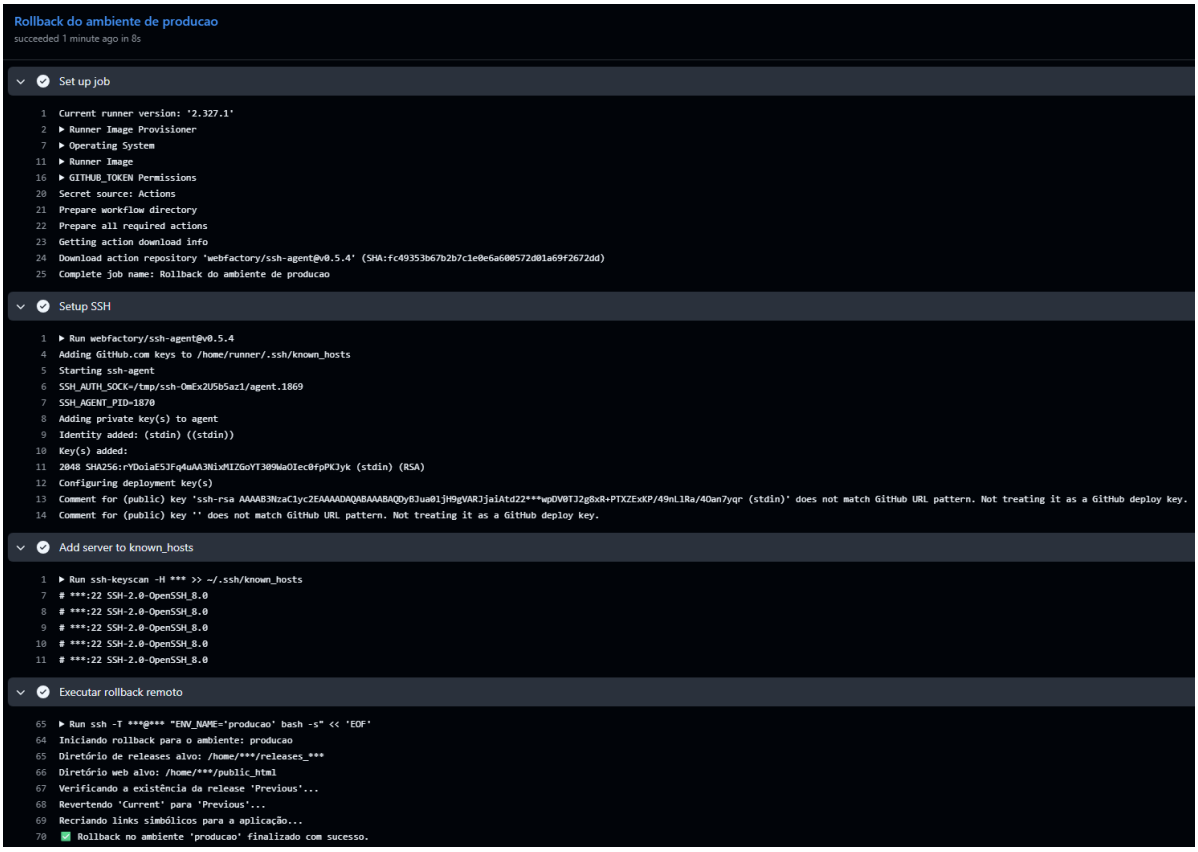
Fonte: Elaborado pelo autor.

4.1.3 Mecanismo de *rollback*

O processo de *rollback* é complementar ao desenvolvimento do *pipeline* CI/CD. O mecanismo foi concebido para sugerir a possibilidade de restaurar a versão anterior de forma célere, considerando o cenário de testes estabelecido. Para tanto, é realizada uma conexão com o servidor, a alteração dos links simbólicos para a versão estável e a exclusão da versão defeituosa. Esse processo, entretanto, não contempla alterações ao banco de dados. Como identificado anteriormente, não foi possível realizar a execução dos *scripts* de migrações para retornar ao estado anterior do banco, uma vez que grande parte da evolução do banco de dados do SisGera ocorreu via comandos de *Structured Query Language* (SQL).

Esse mecanismo tende a ser aplicável em ambos os ambientes, mas deve ser acionado apenas em situações de extrema necessidade. Seu fluxo segue o mesmo padrão para homologação e produção, diferenciando-se apenas quanto ao diretório em que o ambiente se encontra. Pelo fato de estabelecer uma conexão via SSH, verificar a existência de uma versão anterior e alterar os links simbólicos, os testes sugerem que sua execução se mostrou ágil nas condições avaliadas. A Figura 43 ilustra os processos associados.

Figura 43 – Execução do *rollback* no ambiente de produção.



Fonte: Elaborado pelo autor.

4.1.4 Notificações e comunicação via Slack

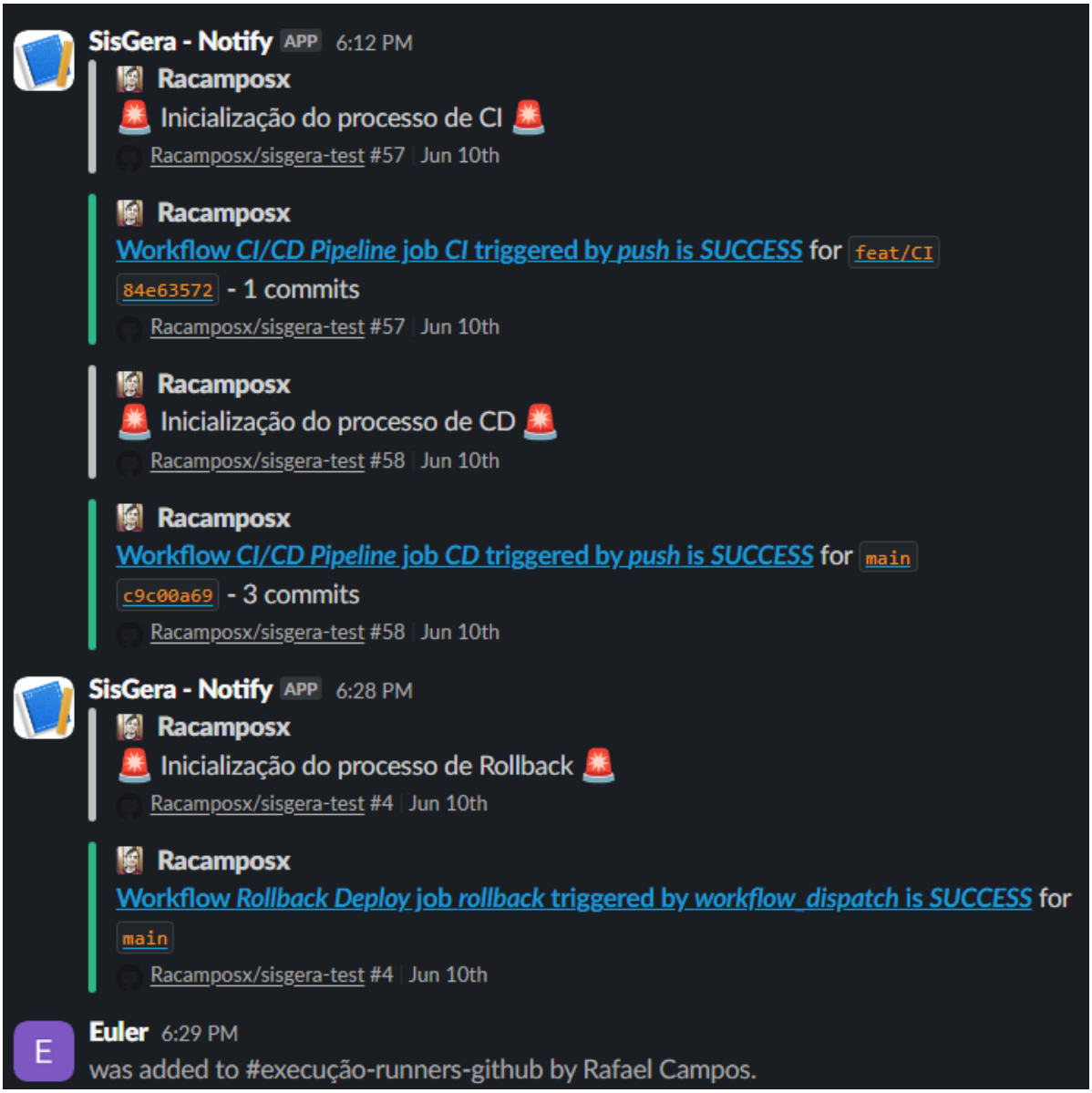
Durante a execução dos fluxos do *pipeline*, todas as etapas são acompanhadas por meio de notificações automáticas integradas ao Slack. Esse mecanismo tem como objetivo fornecer transparência em tempo real sobre o andamento dos processos de Integração Contínua, CD e *rollback*, permitindo que os colaboradores do projeto sejam informados imediatamente sobre o sucesso ou falha de cada execução.

A Figura 44 exemplifica as notificações geradas pelo sistema, indicando a inicialização e conclusão de cada etapa do *pipeline*. Essa integração contribuiu para a identificação mais rápida de falhas e sugere ganhos de eficiência no processo de manutenção do SisGera no período analisado.

Como alternativa, também foi considerada a utilização da plataforma Discord como canal de comunicação. No entanto, pela praticidade de integração nativa com ferramentas de automação, o Slack mostrou-se mais viável para atender às necessidades

do projeto.

Figura 44 – Canal de comunicação com Slack do SisGera.



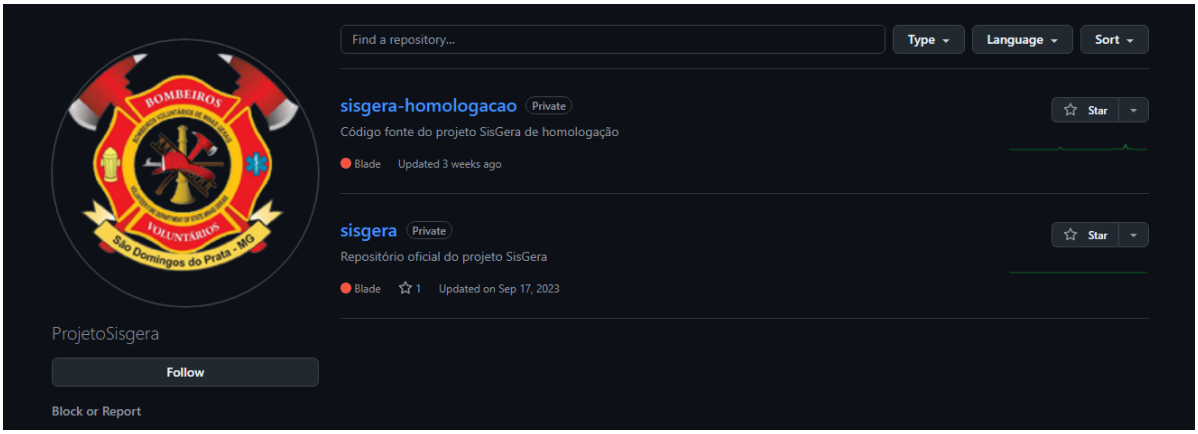
Fonte: Elaborado pelo autor.

4.2 Racionalização do repositório e governança de *branches*

O *pipeline* permitiu consolidar o projeto em um único repositório, substituindo a necessidade de separar cada ambiente por repositório. Anteriormente, a base de código era mantida em dois repositórios distintos: um para o ambiente de homologação e outro

para o de produção. Conforme ilustrado na Seção 4.1, o *pipeline* automatizado é capaz de direcionar as implantações para os ambientes corretos com base na *branch* que acionou o fluxo.

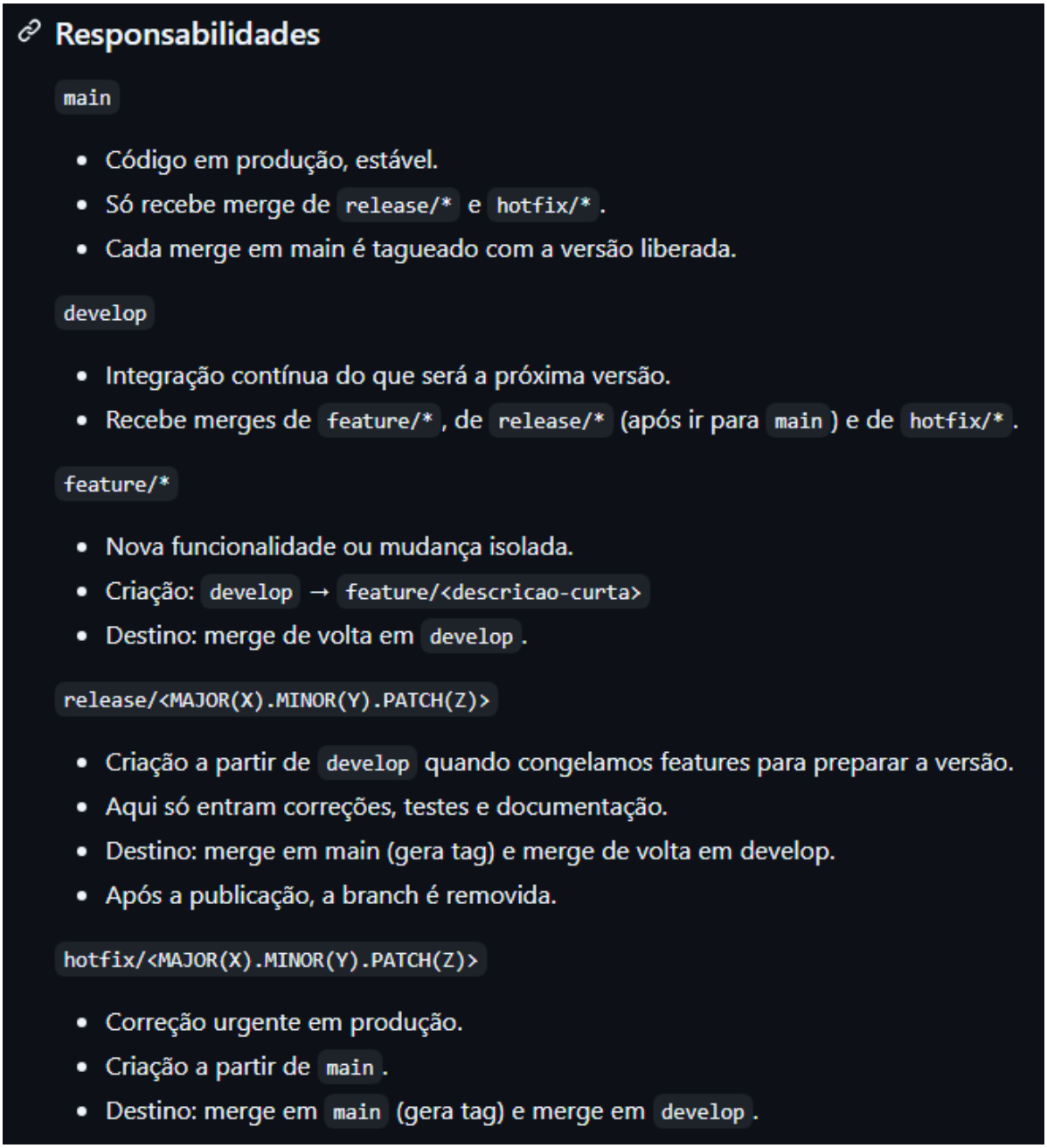
Figura 45 – Separação de repositórios do SisGera.



Fonte: Elaborado pelo autor.

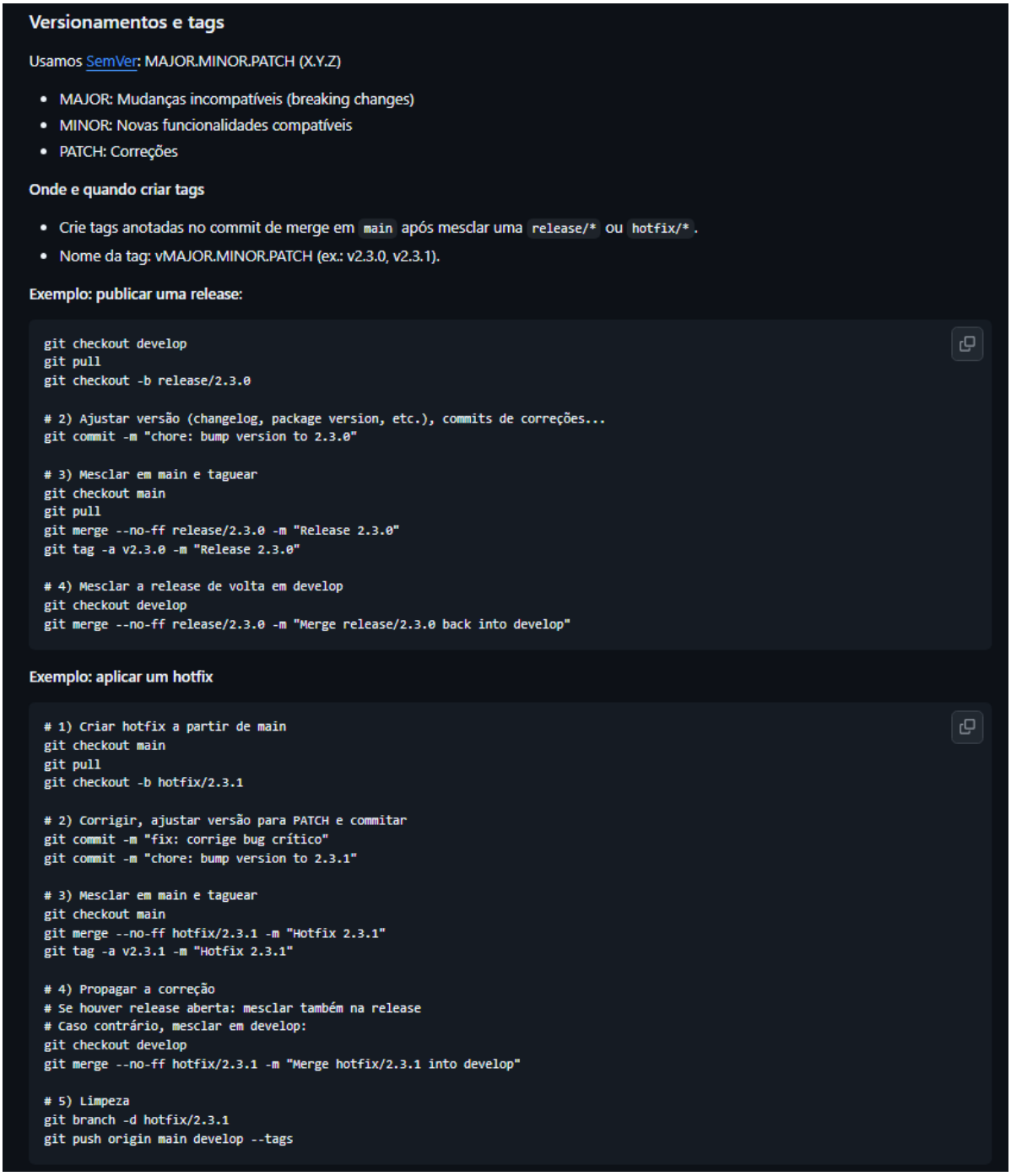
Essa mudança combinada com a adoção do GitFlow, sugere avanços de maturidade no processo de desenvolvimento no contexto observado. As responsabilidades de cada *branch* estão agora claramente definidas e documentadas, criando um fluxo de trabalho organizado, rastreável e menos suscetível a conflitos entre as contribuições dos desenvolvedores. A documentação do projeto inclui a responsabilidade de cada *branch*, como funciona a criação de tags de versão e do fluxo envolvido do *pipeline* CI/CD. Essa observação pode ser feita nas Figura 46, Figura 47 e Figura 48.

Figura 46 – Seção da documentação de responsabilidades de *branches*



Fonte: Elaborado pelo autor.

Figura 47 – Seção de criação de versão com tags



Versionamentos e tags

Usamos SemVer: MAJOR.MINOR.PATCH (X.Y.Z)

- MAJOR: Mudanças incompatíveis (breaking changes)
- MINOR: Novas funcionalidades compatíveis
- PATCH: Correções

Onde e quando criar tags

- Crie tags anotadas no commit de merge em `main` após mesclar uma `release/*` ou `hotfix/*`.
- Nome da tag: vMAJOR.MINOR.PATCH (ex.: v2.3.0, v2.3.1).

Exemplo: publicar uma release:

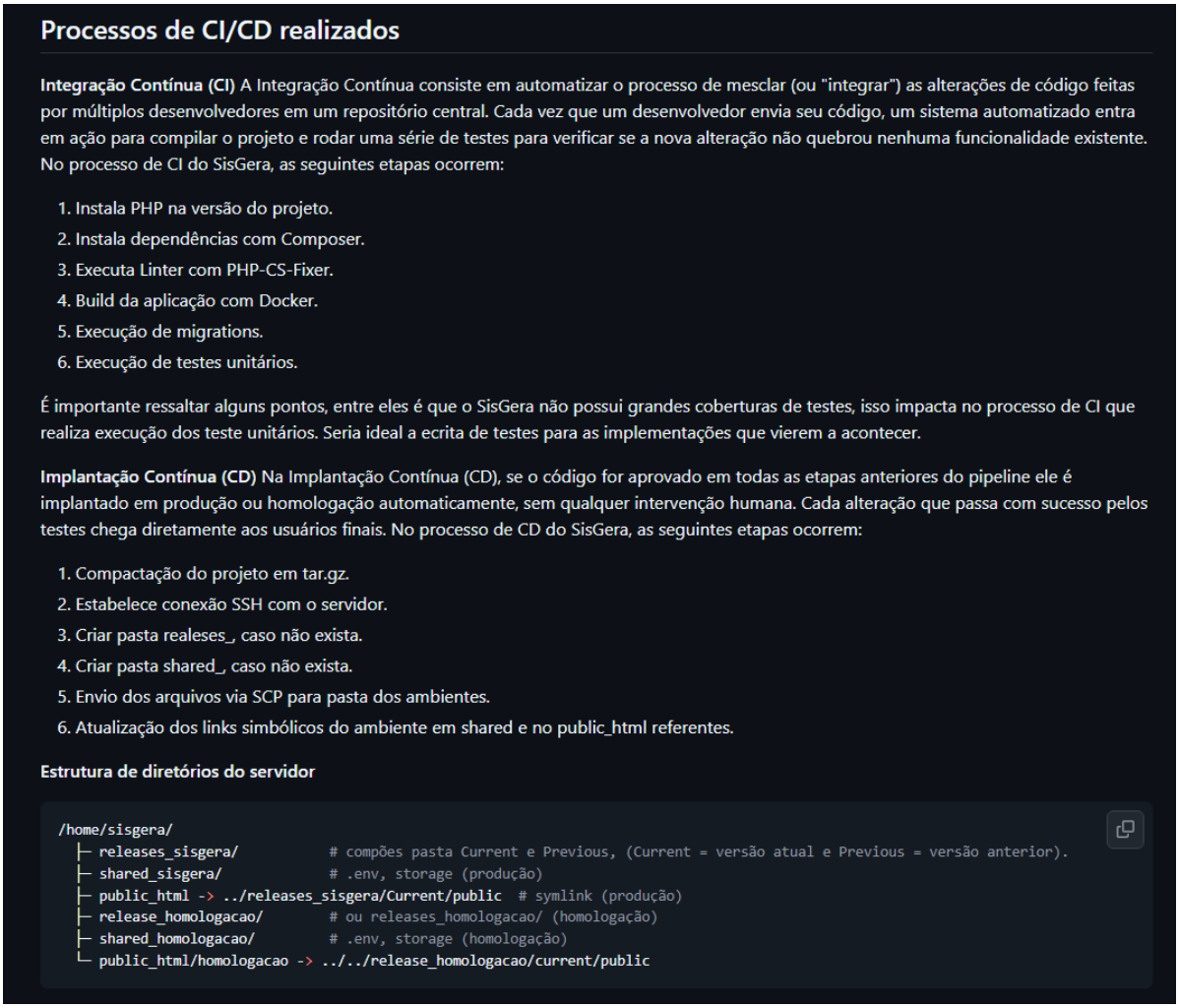
```
git checkout develop
git pull
git checkout -b release/2.3.0

# 2) Ajustar versão (changelog, package version, etc.), commits de correções...
git commit -m "chore: bump version to 2.3.0"

# 3) Mesclar em main e taguear
git checkout main
git pull
git merge --no-ff release/2.3.0 -m "Release 2.3.0"
git tag -a v2.3.0 -m "Release 2.3.0"

# 4) Mesclar a release de volta em develop
git checkout develop
git merge --no-ff release/2.3.0 -m "Merge release/2.3.0 back into develop"
```

Exemplo: aplicar um hotfix

```
# 1) Criar hotfix a partir de main
git checkout main
git pull
git checkout -b hotfix/2.3.1

# 2) Corrigir, ajustar versão para PATCH e commitar
git commit -m "fix: corrige bug crítico"
git commit -m "chore: bump version to 2.3.1"

# 3) Mesclar em main e taguear
git checkout main
git merge --no-ff hotfix/2.3.1 -m "Hotfix 2.3.1"
git tag -a v2.3.1 -m "Hotfix 2.3.1"

# 4) Propagar a correção
# Se houver release aberta: mesclar também na release
# Caso contrário, mesclar em develop:
git checkout develop
git merge --no-ff hotfix/2.3.1 -m "Merge hotfix/2.3.1 into develop"

# 5) Limpeza
git branch -d hotfix/2.3.1
git push origin main develop --tags
```

Fonte: Elaborado pelo autor.

Figura 48 – Seção descrevendo o processo de CI/CD.



Processos de CI/CD realizados

Integração Contínua (CI) A Integração Contínua consiste em automatizar o processo de mesclar (ou "integrar") as alterações de código feitas por múltiplos desenvolvedores em um repositório central. Cada vez que um desenvolvedor envia seu código, um sistema automatizado entra em ação para compilar o projeto e rodar uma série de testes para verificar se a nova alteração não quebrou nenhuma funcionalidade existente. No processo de CI do SisGera, as seguintes etapas ocorrem:

1. Instala PHP na versão do projeto.
2. Instala dependências com Composer.
3. Executa Linter com PHP-CS-Fixer.
4. Build da aplicação com Docker.
5. Execução de migrations.
6. Execução de testes unitários.

É importante ressaltar alguns pontos, entre eles é que o SisGera não possui grandes coberturas de testes, isso impacta no processo de CI que realiza execução dos teste unitários. Seria ideal a ecrita de testes para as implementações que vierem a acontecer.

Implantação Contínua (CD) Na Implantação Contínua (CD), se o código for aprovado em todas as etapas anteriores do pipeline ele é implantado em produção ou homologação automaticamente, sem qualquer intervenção humana. Cada alteração que passa com sucesso pelos testes chega diretamente aos usuários finais. No processo de CD do SisGera, as seguintes etapas ocorrem:

1. Compactação do projeto em tar.gz.
2. Estabelece conexão SSH com o servidor.
3. Criar pasta realeses_, caso não exista.
4. Criar pasta shared_, caso não exista.
5. Envio dos arquivos via SCP para pasta dos ambientes.
6. Atualização dos links simbólicos do ambiente em shared e no public_html referentes.

Estrutura de diretórios do servidor

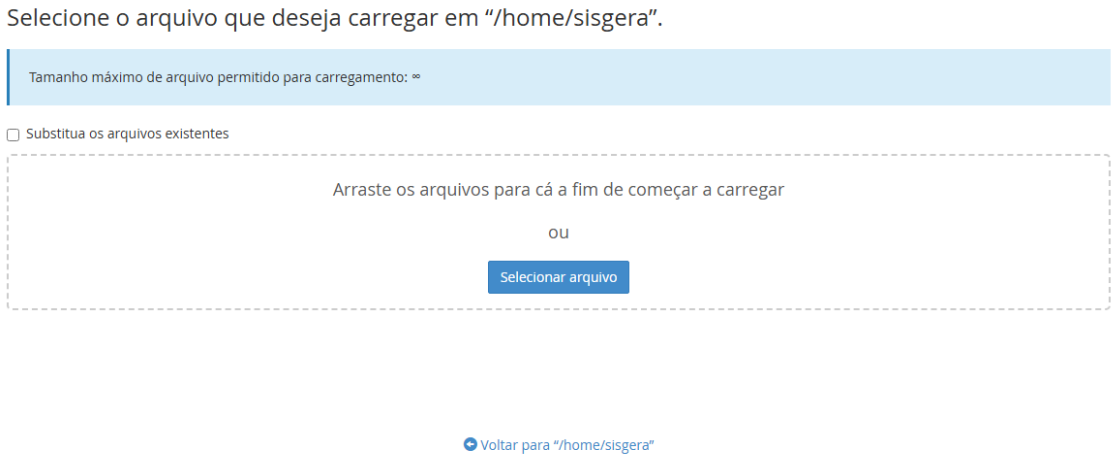
```
/home/sisgera/
  ├─ releases_sisgera/          # compões pasta Current e Previous, (Current = versão atual e Previous = versão anterior).
  ├─ shared_sisgera/            # .env, storage (produção)
  ├─ public_html -> ../releases_sisgera/Current/public  # symlink (produção)
  ├─ release_homologacao/       # ou releases_homologacao/ (homologação)
  ├─ shared_homologacao/        # .env, storage (homologação)
  └─ public_html/homologacao -> ../../release_homologacao/current/public
```

Fonte: Elaborado pelo autor.

4.3 Previsibilidade e padronização das implantações

Padronizou-se o processo de entrega (empacotamento, transferência, descompactação e troca atômica de links simbólicos), o que, no contexto experimental, sugere menor variabilidade de passos manuais e indica potencial redução da suscetibilidade a erro humano. O mesmo vale para a restauração (*rollback*), cuja execução agora segue um roteiro único e rastreável. Conforme a Figura 49, era necessário acessar o painel do cPanel e ir até a seção de “Gerenciador de arquivos” na qual há a opção de realizar a submissão do projeto. Após a submissão, era preciso descompactar e direcionar os arquivos do projeto para seus respectivos diretórios.

Figura 49 – Tela de submissão de arquivo no cPanel.



Fonte: Elaborado pelo autor.

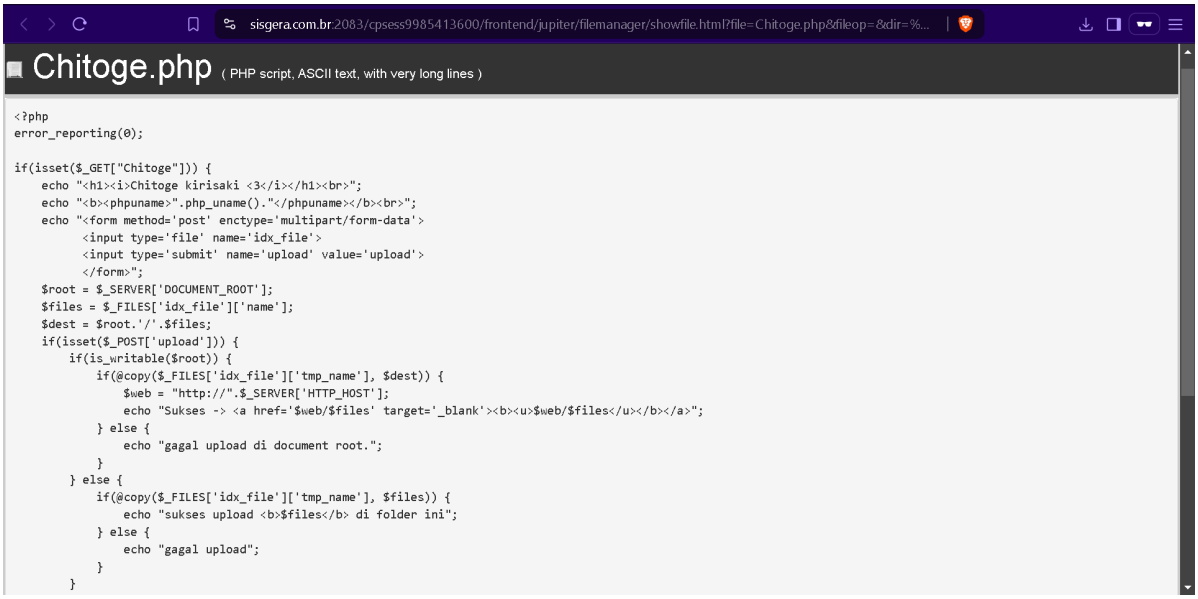
É importante ressaltar que a verificação da implantação no cPanel permanece recomendada. No período analisado, foram identificados arquivos maliciosos, o que sugere manter inspeções de segurança regulares. Essas questões levantam preocupações quanto a atual versão da aplicação, que utiliza o PHP 7.2 e o Laravel 5.6, ambas as versões são descontinuadas e podem estar suscetíveis a falhas de segurança.

Os arquivos oriundos de ataques ao servidor do SisGera podem ser observados na Figura 50, Figura 51 e Figura 52. Nota-se que além de *templates* de páginas, foram submetidos alguns *scripts* para serem executados. Ao momento da identificação desses arquivos, foi feita a exclusão e conferência de cada diretório e da composição do banco de dados.

Figura 50 – Arquivo malicioso no servidor do SisGera

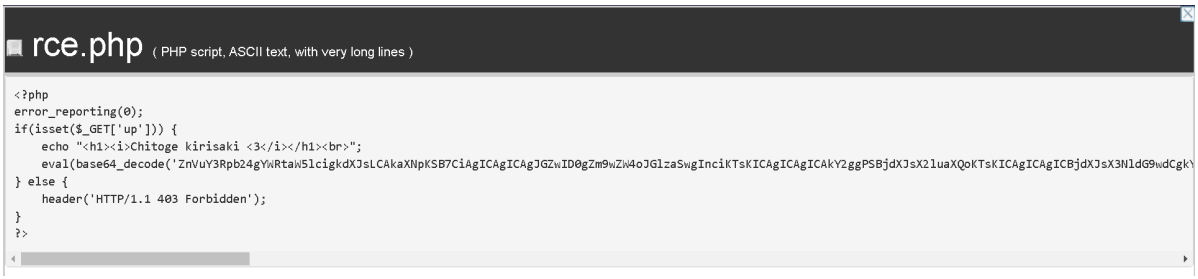
Fonte: Elaborado pelo autor.

Figura 51 – Arquivo malicioso no servidor do SisGera



Fonte: Elaborado pelo autor.

Figura 52 – Arquivo malicioso no servidor do SisGera



Fonte: Elaborado pelo autor.

5 Considerações finais

Neste trabalho foi implementado um *pipeline* de Integração Contínua / Implantação Contínua para o Sistema de Gerenciamento e Registro de Atividades, utilizado por corporações de bombeiros voluntários em Minas Gerais. O objetivo principal foi modernizar o processo de entrega de software, anteriormente manual e suscetível a falhas, substituindo-o por um fluxo automatizado, previsível e rastreável. Desenvolveu-se o *pipeline* com três etapas principais: Integração Contínua, Implantação Contínua e *rollback* manual. Cada uma com responsabilidades claras e mecanismos de notificação via Slack. Também foi elaborada uma nova documentação do projeto, incluindo a adoção do GitFlow, a padronização de *commits* (*Conventional Commits*) e guias de preparação do ambiente local, tornando o processo mais acessível a novos colaboradores.

Os resultados obtidos indicam ganhos significativos. A automação reduziu a suscetibilidade a erros humanos, padronizou os processos de entrega, possibilitou reverter versões de maneira segura e melhorou a governança de branches no repositório, consolidando o desenvolvimento em um único fluxo. Contudo, algumas dificuldades foram identificadas, em especial a ausência de um conjunto completo de migrações no banco de dados do SisGera, o que limitou a capacidade de rollback em nível de dados. Além disso, observou-se a presença de arquivos maliciosos no servidor e a utilização de versões descontinuadas de PHP e Laravel, indicando a necessidade de atualização tecnológica para mitigar riscos de segurança.

5.1 Trabalhos Futuros

Como continuidade deste trabalho, destacam-se algumas propostas relevantes:

- Evolução do banco de dados: desenvolver um conjunto completo de migrações consistentes, corrigindo falhas de integridade referencial e tipos inadequados de dados, para permitir maior confiabilidade nas atualizações e reversões.
- Rollback de banco de dados: implementar mecanismos que garantam a reversão não apenas da aplicação, mas também do estado do banco, ampliando a segurança em casos de falhas.

- Atualização tecnológica: migrar o SisGera para versões mais recentes do PHP e do Laravel, garantindo suporte ativo da comunidade e maior resiliência a vulnerabilidades.
- Automação de testes avançada: ampliar a cobertura de testes automatizados, fortalecendo a qualidade do software.
- Monitoramento contínuo: integrar ferramentas de observabilidade (logs centralizados, métricas e alertas) ao *pipeline*, permitindo acompanhamento proativo de falhas e maior confiabilidade operacional.

Referências

ARANTES, V. M. Um sistema de informação para apoio ao registro de ocorrências atendidas por grupos de bombeiros voluntários. 2018. Citado na página 15.

ATLASSIAN. *Bitbucket Pipelines Features*. 2024. Disponível em: <https://www.atlassian.com/software/bitbucket/features/pipelines>. Citado na página 33.

AZAD, N.; HYRYNSALMI, S. Multivocal literature review on devops critical success factors. In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2024. p. 520–527. Citado 2 vezes nas páginas 19 e 20.

BARROS, M. N. M. d. Adaptação do sisgera à lei geral de proteção de dados. 2023. Disponível em: <http://www.monografias.ufop.br/handle/35400000/5547>. Citado 2 vezes nas páginas 15 e 36.

BROWSERSTACK. *What is Headless Browser Testing?* 2024. <https://www.browserstack.com/guide/what-is-headless-browser-testing>. Citado na página 23.

CASTRO, L. H. M. d. Desenvolvimento de um módulo de help desk para o sistema sisgera. 2022. Disponível em: <https://monografias.ufop.br/handle/35400000/4515>. Citado na página 15.

CLOUDOPS, N. *DevOps Zero to Hero — Day 20: Deployment Strategies*. 2023. Disponível em: <https://medium.com/@navya.cloudops/devops-zero-to-hero-day-20-deployment-strategies-e6712b4801e4>. Citado na página 29.

CPANEL. *cPanel: Hosting Plataform of Choice*. 2025. Disponível em: <https://www.cpanel.net>. Citado na página 38.

DEBOIS, P. Agile infrastructure and operations: How infra-gile are you? In: IEEE. *Agile 2008 conference*. [S.l.], 2008. p. 202–207. Citado na página 19.

DOCKER. *Docker: Accelerated Container Application Development*. 2025. Disponível em: <https://www.docker.com>. Citado na página 36.

DRIESSEN, V. *A successful Git branching model*. 2010. Accessed: 2025-07-29. Disponível em: <https://nvie.com/posts/a-successful-git-branching-model/>. Citado na página 24.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. *Continuous integration: improving software quality and reducing risk*. [S.l.]: Pearson Education, 2007. Citado 2 vezes nas páginas 21 e 23.

GITHUB. *GitHub Documentation*. 2024. Disponível em: <https://docs.github.com/>. Citado na página 32.

GITHUB. *GitHub Pricing*. 2024. Disponível em: <https://github.com/pricing>. Citado na página 32.

GITHUB. *Using secrets in GitHub Actions - GitHub Docs*. 2025. Disponível em: <https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions>. Citado na página 40.

GITLAB. *About GitLab*. 2024. Disponível em: <https://about.gitlab.com/>. Citado na página 31.

GITLAB. *GitLab Documentation*. 2024. Disponível em: <https://docs.gitlab.com/>. Citado na página 31.

GRUNWELL, S. *Atomic Deployments from Scratch*. 2019. Disponível em: <https://stevegrunwell.com/blog/atomic-deployments-from-scratch/>. Citado na página 30.

HANES, B. *What is Software Deployment? Strategies and Practices*. 2024. Disponível em: <https://ideamaker.agency/what-is-software-deployment/>. Citado na página 29.

HUMBLE, J.; FARLEY, D. *Continuous delivery: reliable software releases through build, test, and deployment automation*. [S.l.]: Pearson Education, 2010. Citado 2 vezes nas páginas 27 e 28.

JENKINS. *Jenkins: Build Great Things at Any Scale*. 2024. Disponível em: <https://www.jenkins.io/>. Citado na página 32.

JENKINS. *Using a Jenkinsfile*. 2024. Disponível em: <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>. Citado na página 32.

KIM, G. et al. *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. [S.l.]: It Revolution, 2021. Citado na página 20.

OLIVEIRA, S. S. M. Sistema de informação para controle de materiais e doações aos bombeiros voluntários. 2018. Disponível em: <https://www.monografias.ufop.br/handle/35400000/1621>. Citado na página 15.

PINTO, B. L. T. Elaboração de testes para o sistema de gerenciamento e registro de atividades (sisgera). 2022. Disponível em: <http://www.monografias.ufop.br/handle/35400000/4891>. Citado na página 15.

SILVA, G. O. Desenvolvimento de uma aplicação web progressiva para o registro de ocorrências de um grupo de bombeiros voluntários. 2021. Disponível em: <https://monografias.ufop.br/handle/35400000/3503>. Citado na página 15.

SMITH, D. *The Evolution of DevOps*. 2024. Disponível em: <https://devops.com/the-evolution-of-devops/>. Citado na página 19.

VALENTE, M. T. Engenharia de software moderna. *Princípios e práticas para desenvolvimento de software com produtividade*, 2020. Citado 3 vezes nas páginas 20, 21 e 22.

VALENTE, M. T. *Gerenciando Branches com Git-flow, GitHubFlow e TBD*. 2025. Disponível em: <https://engsoftmoderna.info/artigos/gitflow.html>. Citado 4 vezes nas páginas 24, 25, 26 e 27.

VIRTANEN, J. Comparing different ci/cd pipelines. 2021. Citado na página 34.

VOCKE, H. *The Practical Test Pyramid*. 2018. Disponível em: <https://martinfowler.com/articles/practical-test-pyramid.html>. Citado 2 vezes nas páginas 21 e 22.