REVISTA TÓPICOS
ISSN 2965-6672 | Qualis A2

IMPLEMENTAÇÃO DE DEVOPS NO PROINFE: AUTOMAÇÃO E GERENCIAMENTO COM GITLAB, ARGO CD E K3S

IMPLEMENTATION OF DEVOPS IN PROINFE: AUTOMATION AND MANAGEMENT WITH GITLAB, ARGO CD AND K3S

Ciências Exatas e da Terra • 15/07/2026

REGISTRO DOI: 10.70773/revistatopicos/783657307

Wester Jesuino Morandi de Oliveira

João Eujácio Teixeira Junior

RESUMO

Este artigo analisa a implementação de práticas DevOps no Projeto de Informatização Escolar (Proinfe), sistema voltado ao apoio da gestão acadêmica de escolas municipais de educação básica em Rondônia. A pesquisa teve como objetivo estruturar e avaliar um fluxo automatizado para construção, publicação e implantação de aplicações baseadas em microserviços, buscando ampliar a rastreabilidade, a padronização e a organização dos ambientes de desenvolvimento, homologação e produção. O estudo adotou abordagem qualitativa, de caráter aplicado, com delineamento de estudo de caso, utilizando análise documental de pipelines, manifestos Kubernetes, registros de imagens e observação direta do funcionamento da infraestrutura. A solução empregou GitLab CI/CD para automação dos pipelines, Argo CD para sincronização declarativa por GitOps e K3s em infraestrutura própria virtualizada com Proxmox. Os resultados indicam que o fluxo implantado reduziu a dependência de intervenções manuais diretas, fortaleceu o controle das versões publicadas e organizou a separação dos ambientes, inclusive com execução controlada de rotinas de banco por Jobs do Kubernetes. Os efeitos sobre disponibilidade, taxa de falha e tempo de recuperação, entretanto, dependem de monitoramento quantitativo futuro. Conclui-se que a adoção dessas práticas contribuiu para a sustentação operacional do Proinfe e constitui uma referência aplicável a projetos que necessitam de automação, rastreabilidade e organização de infraestrutura, especialmente em contextos institucionais com recursos controlados.

Palavras-chave: DevOps; GitLab CI/CD; Argo CD; GitOps; K3s.

ABSTRACT

The dependence on manual deployments, the difficulty of tracing

deployed versions and the need to safely separate development, staging and production environments represent relevant challenges for maintaining distributed systems. This article presents the implementation of DevOps practices in the School Informatization Project (Proinfe), a system aimed at academic management and user interaction for municipal elementary schools in Rondônia. Using a methodological approach based on a case study, the proposed solution structured an automated flow for building, publishing and deploying microservice-based applications. For this purpose, GitLab CI/CD was used for pipeline automation, Argo CD for declarative manifest synchronization through GitOps, and K3s as a lightweight Kubernetes distribution running in an institutional environment virtualized with Proxmox. The architecture included three backend microservices, responsible for authentication, school records management and school organization, as well as a React frontend, with development and staging organized in distinct namespaces and production maintained in a separate cluster. The results indicate that adopting this flow contributed to greater traceability, standardization and operational control of the deployment process, while effects on availability, failure rate and recovery time depend on future quantitative monitoring. The solution supports the technological sustainability of Proinfe and offers a replicable model for educational projects that require automation and infrastructure organization.
Keywords: DevOps; GitLab CI/CD; Argo CD; GitOps; K3s.

1. INTRODUÇÃO

A transformação digital tem provocado mudanças significativas na forma como instituições públicas organizam informações, prestam serviços e acompanham processos sociais. Na educação, esse

movimento ganha relevância especial, pois escolas e secretarias lidam diariamente com dados acadêmicos, administrativos e operacionais que precisam estar disponíveis, atualizados e protegidos. Estudos sobre tecnologia educacional indicam que o uso de dados digitais pode apoiar não apenas práticas pedagógicas, mas também a gestão, o acompanhamento institucional e a tomada de decisão (OECD, 2021). De modo complementar, documentos internacionais sobre tecnologia educacional destacam que a adoção de tecnologias na educação deve responder a necessidades reais, respeitar finalidades públicas e contribuir para sistemas mais organizados, inclusivos e responsáveis (UNESCO, 2023).

Nesse cenário, sistemas educacionais deixam de ser apenas ferramentas de apoio e passam a compor a infraestrutura essencial para o funcionamento da gestão escolar. Cadastros de estudantes, matrículas, turmas, calendários, diários, históricos, transferências, ocorrências e informações de responsáveis precisam circular com segurança entre diferentes perfis de usuários e unidades administrativas. Quando essas informações dependem de aplicações instáveis, atualizações manuais ou ambientes mal separados, o problema deixa de ser apenas técnico e passa a afetar a continuidade dos serviços prestados à comunidade escolar.

A manutenção de sistemas distribuídos amplia esse desafio. Arquiteturas baseadas em microserviços favorecem a separação de responsabilidades e a evolução independente de componentes, mas também exigem maior coordenação entre versões, configurações, infraestrutura e processos de implantação (Di Francesco; Lago; Malavolta, 2019). Estudos sobre integração, entrega e implantação contínuas apontam que a automação do ciclo de entrega contribui para reduzir intervenções manuais, aumentar previsibilidade e

melhorar a confiabilidade das aplicações (Shahin; Babar; Zhu, 2017). Nessa mesma direção, estudos de DevOps compreende a aproximação entre desenvolvimento e operações como um caminho para tornar a entrega de software mais colaborativa, rastreável e segura (Leite et al., 2019). Essa perspectiva também valoriza práticas culturais e operacionais voltadas à melhoria contínua, ao aprendizado organizacional e à redução de gargalos entre equipes (Kim et al., 2016).

O Projeto de Informatização Escolar (Proinfe) insere-se nesse contexto como uma iniciativa voltada ao gerenciamento acadêmico e à interface de usuários das escolas municipais de educação básica de Rondônia, em ações desenvolvidas a partir de convênios entre o Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO) e prefeituras municipais. Por atender processos ligados à vida escolar e à gestão administrativa, o sistema demanda não apenas funcionalidades adequadas ao cotidiano das secretarias e escolas, mas também uma infraestrutura capaz de sustentar atualizações frequentes com organização, segurança e rastreabilidade.

A problematização que orienta esta pesquisa surgiu da análise do fluxo anterior de implantação do Proinfe. Antes da adoção das práticas DevOps, o deploy dependia da construção da aplicação em máquina local, do acesso direto ao servidor, da parada manual de containers e da inicialização de uma nova versão. Embora esse procedimento permitisse disponibilizar atualizações, ele concentrava etapas críticas em ações humanas repetitivas, dificultava a identificação precisa da versão implantada e aumentava o risco de inconsistências entre desenvolvimento, homologação e produção. Assim, a questão de pesquisa pode ser formulada da seguinte maneira: como a implementação de um

fluxo DevOps baseado em GitLab CI/CD, Argo CD e K3s pode contribuir para melhorar a automação, a rastreabilidade e a organização dos deploys do Proinfe?

A justificativa desta pesquisa está relacionada à necessidade de tornar mais confiável a sustentação tecnológica de um sistema utilizado em contexto público educacional. Em projetos dessa natureza, a implantação de uma nova versão não representa apenas uma atualização de código, mas uma etapa que pode influenciar a disponibilidade de serviços administrativos e acadêmicos. Autores da área de DevOps destacam que práticas como automação, entrega contínua, versionamento e feedback operacional contribuem para reduzir falhas e fortalecer a capacidade de evolução dos sistemas (Bass; Weber; Zhu, 2015). Além disso, estudos sobre desempenho em DevOps associam práticas de entrega bem estruturadas à melhoria da estabilidade, da frequência de implantação e da capacidade de recuperação diante de falhas (Forsgren; Humble; Kim, 2018).

Dessa forma, o objetivo geral deste trabalho é implementar e analisar um fluxo DevOps para o Proinfe, utilizando GitLab CI/CD, Argo CD e K3s como base para automatizar a construção, publicação e implantação das aplicações, bem como ampliar a rastreabilidade e a organização dos ambientes. De maneira complementar, busca-se compreender o contexto técnico e institucional do Proinfe, reorganizar o processo de entrega dos microserviços, utilizar manifestos versionados como fonte de verdade para as implantações, separar adequadamente os ambientes de desenvolvimento, homologação e produção e avaliar os impactos observados após a adoção desse fluxo.

Os resultados analisados concentram-se na redução da dependência de procedimentos manuais, na melhoria da identificação das versões implantadas, na padronização do processo de entrega e na diminuição do risco de confusão entre ambientes. A adoção de GitOps, por exemplo, permite que o estado desejado da aplicação seja descrito em repositório versionado e sincronizado com o ambiente de execução, favorecendo histórico, auditoria e previsibilidade (Argo Project, 2026b). Já o uso de uma distribuição Kubernetes leve, como o K3s, torna possível aproximar práticas modernas de orquestração da realidade de infraestruturas próprias virtualizadas, sem depender necessariamente de serviços gerenciados em nuvem (K3s, 2026).

Assim, esta pesquisa parte de uma problemática concreta de implantação manual e avança para a análise de uma solução orientada por automação, rastreabilidade e organização operacional. Ao afunilar a discussão da transformação digital na educação para o caso específico do Proinfe, o estudo busca demonstrar que práticas DevOps podem contribuir não apenas para a eficiência técnica, mas também para a continuidade e a confiabilidade de sistemas que sustentam processos públicos de gestão escolar.

2. REFERENCIAL TEÓRICO

2.1. A Transformação Digital na Gestão Educacional

A informatização da gestão educacional está inserida em um movimento mais amplo de transformação digital dos serviços públicos. No campo da educação, recursos digitais passaram a atuar como instrumentos de organização de dados, acompanhamento institucional e apoio à tomada de decisão nos sistemas de ensino

(OECD, 2021). Essa incorporação, entretanto, não deve ser tratada apenas como modernização tecnológica, mas como processo orientado por necessidades reais, finalidades públicas e critérios de segurança, inclusão e responsabilidade (UNESCO, 2023).

Sob uma perspectiva sociotécnica, a gestão educacional informatizada depende da articulação entre pessoas, processos, dados e infraestrutura. A confiabilidade de um sistema escolar não se expressa somente na existência de funcionalidades, mas na capacidade de manter registros íntegros, preservar a continuidade das rotinas administrativas e permitir que mudanças sejam introduzidas sem comprometer o trabalho das equipes. Quando a implantação de novas versões ocorre sem padronização ou rastreabilidade, a incerteza técnica pode repercutir na segurança dos dados, na organização institucional e na confiança dos usuários. Por isso, a problemática deste trabalho é compreendida como parte de um desafio maior de governança tecnológica: transformar a entrega de software em um processo controlado, verificável e alinhado à responsabilidade pública do Proinfe.

2.2. DevOps como Caminho para Entregas Mais Confiáveis

DevOps pode ser compreendido como uma abordagem que aproxima desenvolvimento e operações por meio de colaboração, automação, feedback e melhoria contínua. Bass, Weber e Zhu tratam DevOps como uma perspectiva arquitetural e organizacional voltada à entrega contínua de valor, especialmente em sistemas que exigem evolução frequente e operação confiável (Bass; Weber; Zhu, 2015). Também reforça que DevOps não se limita à adoção de ferramentas, pois envolve cultura, processos, automação e mudança

na forma como equipes constroem, entregam e mantêm software (Leite et al., 2019).

A integração contínua e a entrega contínua complementam essa abordagem ao estabelecerem práticas para validar alterações, empacotar aplicações e preparar implantações de maneira mais frequente e previsível. A entrega contínua busca tornar o processo de liberação de software mais confiável, repetível e menos dependente de ações manuais (Humble; Farley, 2010). Estudos sistemáticos observam que práticas de integração, entrega e implantação contínuas estão associadas à redução de riscos, ao aumento de velocidade e à melhoria da confiabilidade no ciclo de entrega (Shahin; Babar; Zhu, 2017).

No caso do Proinfe, esses conceitos sustentam a substituição de um fluxo baseado em comandos manuais por um fluxo orientado por pipelines, versionamento e automação. O GitLab CI/CD permite definir pipelines como código em arquivos versionados, organizando etapas de execução, regras e dependências (GitLab, 2026a). Essa característica é relevante porque aproxima o processo de implantação de uma lógica auditável, repetível e documentada no próprio repositório do projeto.

2.3. Rastreabilidade e Organização dos Ambientes por Meio do GitOps

GitOps é uma prática em que o Git atua como fonte de verdade para o estado desejado das aplicações e da infraestrutura. Nessa abordagem, as mudanças são registradas em repositórios versionados e ferramentas especializadas sincronizam o ambiente de execução com aquilo que foi declarado no código. O Argo CD é

uma ferramenta declarativa de entrega contínua para Kubernetes que compara o estado desejado descrito no Git com o estado real do cluster, realizando sincronizações quando necessário (Argo Project, 2026a). Sua documentação também destaca a possibilidade de integração com automações de CI, o que favorece fluxos em que pipelines atualizam manifestos e ferramentas de GitOps realizam a aplicação das mudanças (Argo Project, 2026b).

A rastreabilidade é um elemento central nessa discussão, pois permite associar alterações de aplicação, imagens, manifestos e ambientes a registros versionados. Em sistemas distribuídos, nos quais diferentes serviços podem evoluir em ritmos distintos, a ausência de rastreabilidade tende a dificultar a identificação do que foi implantado, quando foi implantado e em qual ambiente a alteração está ativa. Por isso, a adoção de GitOps no Proinfe dialoga diretamente com a problemática de reduzir incertezas operacionais e ampliar o controle sobre desenvolvimento, homologação e produção.

A separação de ambientes também possui função conceitual importante. O Kubernetes utiliza namespaces para organizar objetos dentro de um cluster, permitindo separação lógica entre aplicações, equipes ou ambientes (Kubernetes, 2026c). No Proinfe, essa noção sustenta a separação entre desenvolvimento e homologação dentro de um mesmo cluster, enquanto a produção foi mantida em cluster separado. Dessa forma, combina-se separação lógica e isolamento operacional, reduzindo o risco de confusão entre testes, validações e uso produtivo.

2.4. Microserviços e Orquestração para Sustentação Operacional

Arquiteturas baseadas em microserviços favorecem a decomposição de sistemas em componentes menores, especializados e com responsabilidades mais bem definidas. Entretanto, essa divisão também aumenta a necessidade de coordenação entre serviços, versões, configurações e infraestrutura. Estudos sobre microserviços indicam que esse estilo arquitetural demanda atenção a aspectos como comunicação, implantação, monitoramento e evolução independente dos componentes (Di Francesco; Lago; Malavolta, 2019). Essa relação é essencial para compreender o Proinfe, cuja arquitetura envolve microserviços de autenticação, secretaria e organização escolar, além de frontend e serviços auxiliares.

A orquestração com Kubernetes oferece recursos para organizar a execução desses componentes em ambiente distribuído. No projeto, a adoção do K3s se justifica por se tratar de uma distribuição Kubernetes leve, voltada a cenários de menor complexidade operacional e adequada a ambientes como borda, desenvolvimento, CI e produção simplificada (K3s, 2026). Essa característica permite aproximar práticas modernas de orquestração da realidade de uma infraestrutura própria virtualizada com Proxmox, sem exigir dependência direta de serviços gerenciados em nuvem.

Além da execução dos serviços, a disponibilidade depende de mecanismos capazes de lidar com variações de carga, falhas e atualizações. O Horizontal Pod Autoscaler ajusta automaticamente a quantidade de réplicas de um Deployment com base em métricas, como utilização média de CPU (Kubernetes, 2026a). Para cargas assíncronas e orientadas a eventos, o KEDA amplia essa lógica ao permitir escalabilidade baseada em fontes externas, como filas e mensageria (KEDA, 2026). O Ingress, por sua vez, permite gerenciar o acesso HTTP e HTTPS externo aos serviços internos do cluster por

meio de regras de roteamento (Kubernetes, 2026b). No Proinfe, esses recursos contribuem para organizar o acesso à API e ao frontend, bem como para sustentar a operação dos microserviços em ambientes distintos.

2.5. Evidências da Literatura sobre Automação e Entrega Contínua

Pesquisas recentes sobre DevOps e entrega contínua indicam que a automação do ciclo de software está associada à melhoria de previsibilidade, estabilidade e capacidade de resposta. Mapeamentos sistemáticos sobre implantação contínua demonstram que a adoção dessas práticas está relacionada à necessidade de reduzir o intervalo entre desenvolvimento e disponibilização de novas versões (Rodríguez et al., 2017). Também identificam que integração, entrega e implantação contínuas envolvem desafios técnicos, organizacionais e culturais, mas oferecem ganhos importantes para equipes que precisam entregar software com frequência e menor risco (Shahin; Babar; Zhu, 2017).

As pesquisas DevOps Research and Assessment (DORA) propõem métricas para avaliar desempenho de entrega de software, como frequência de implantação, tempo de entrega de mudanças, taxa de falha e tempo de recuperação (Forsgren; Humble; Kim, 2018). Embora este trabalho não mensure formalmente esses indicadores, eles compõem um referencial importante para avaliar futuramente os efeitos da solução implantada no Proinfe. A contribuição deste estudo, portanto, está em analisar uma aplicação prática dessas ideias em um sistema educacional público, com foco em automação, rastreabilidade e organização de ambientes.

2.6. Convergência Teórica para a Solução Proposta

A relação entre os conceitos apresentados permite compreender a lógica desta pesquisa. A transformação digital na educação cria demanda por sistemas estáveis e confiáveis; os microserviços ampliam a modularidade, mas aumentam a complexidade de implantação; DevOps oferece princípios para reduzir barreiras entre desenvolvimento e operação; CI/CD automatiza a construção e a publicação de versões; GitOps fortalece a rastreabilidade por meio de manifestos versionados; e Kubernetes, por meio do K3s, fornece a base de orquestração para executar os serviços em ambientes separados.

Dessa forma, o referencial teórico sustenta a análise do Proinfe ao demonstrar que a problemática do deploy manual encontra respaldo em estudos sobre entrega contínua, DevOps, microserviços e orquestração. A combinação desses elementos justifica a adoção de GitLab CI/CD, Argo CD e K3s como solução para melhorar a automação, a rastreabilidade e a padronização do processo de implantação, conectando o problema prático observado no projeto a um conjunto de conceitos consolidados na literatura.

3. METODOLOGIA

A metodologia deste trabalho foi estruturada para explicar como a intervenção DevOps no Proinfe foi planejada, executada e analisada em seu próprio ambiente de operação. Em vez de tratar GitLab CI/CD, Argo CD e K3s como ferramentas isoladas, o estudo considerou a relação entre processo de implantação, arquitetura da aplicação e infraestrutura disponível. Por essa razão, a pesquisa assumiu caráter aplicado, abordagem qualitativa e delineamento de

estudo de caso. A pesquisa aplicada mostrou-se adequada por estar voltada à solução de um problema concreto de implantação e sustentação tecnológica, enquanto a abordagem qualitativa permitiu interpretar decisões técnicas, processos de trabalho e resultados observados em seu contexto real de ocorrência (Gil, 2002). O estudo de caso, por sua vez, foi adotado por possibilitar o exame aprofundado de um fenômeno contemporâneo em seu ambiente natural, preservando as relações entre o problema investigado, a intervenção realizada e o contexto institucional no qual o sistema é mantido (Yin, 2015).

No âmbito do Proinfe, a unidade de análise foi delimitada pelo fluxo de entrega e implantação das aplicações. Esse recorte abrangeu os microserviços de autenticação, secretaria e organização escolar, o frontend em React, os serviços auxiliares e a infraestrutura executada em clusters K3s sobre um ambiente institucional virtualizado com Proxmox. Essa caracterização foi relevante porque a solução não partiu de uma nuvem Kubernetes gerenciada, mas de uma infraestrutura institucional própria, na qual a escolha do K3s se relacionou à necessidade de operar Kubernetes de forma mais leve, controlada e compatível com recursos virtualizados. A delimitação desses componentes ocorreu de forma intencional, pois eles estavam diretamente relacionados à problemática central da pesquisa: substituir um processo de deploy manual por um fluxo mais automatizado, rastreável e organizado entre os ambientes de desenvolvimento, homologação e produção.

Do ponto de vista procedimental, a intervenção foi organizada em fases para tornar o percurso metodológico mais claro e replicável. A primeira fase, denominada preparação e arquitetura, envolveu a caracterização da aplicação, a organização do ambiente Proxmox, o

provisionamento dos clusters K3s e a definição da separação entre namespaces de desenvolvimento e homologação e cluster específico para produção. A segunda fase correspondeu à integração contínua, com a estruturação dos pipelines no GitLab CI/CD, construção das imagens Docker e publicação no GitLab Container Registry. A terceira fase tratou da entrega contínua por GitOps, com atualização dos manifestos Kubernetes no repositório *k3s* e sincronização dos ambientes por meio do Argo CD. A quarta fase concentrou a execução controlada de rotinas de banco, especialmente migrations e seeders por Jobs do Kubernetes, e a avaliação qualitativa dos resultados observados após a implantação.

As técnicas de coleta de dados foram organizadas em duas categorias principais. O recorte temporal da análise contemplou artefatos gerados entre julho de 2024 e dezembro de 2025, abrangendo os microserviços de autenticação, secretaria e organização escolar, o frontend em React, os serviços auxiliares e os ambientes de desenvolvimento, homologação e produção. A análise documental contemplou artefatos estáticos produzidos ou alterados durante a intervenção, como arquivos de pipeline, manifestos Kubernetes, configurações de Ingress, definições de HPA, registros de commits no repositório *k3s* e tags de imagens publicadas no GitLab Container Registry. A observação direta e participante concentrou-se no acompanhamento do funcionamento dos ambientes, na sincronização realizada pelo Argo CD, na execução dos Jobs de migrations e seeders, na separação entre namespaces e clusters e no comportamento do fluxo de promoção entre desenvolvimento, homologação e produção. Esses registros foram utilizados como evidências por representarem tanto a configuração declarada da solução quanto sua operação em ambiente real.

A análise dos dados foi conduzida de forma qualitativa e interpretativa, tendo como eixo comparativo o contraste entre o fluxo anterior, caracterizado por build local, acesso direto ao servidor e substituição manual de containers, e o fluxo DevOps implantado, estruturado com automação, versionamento de manifestos, sincronização pelo Argo CD e orquestração em Kubernetes. A automação foi analisada a partir da substituição de comandos manuais por pipelines; a rastreabilidade, pelos registros de execução do GitLab CI/CD, pelos commits no repositório *k3s* e pelo histórico de sincronização do Argo CD; a padronização, pela adoção de manifestos declarativos versionados; e a segurança operacional, pela separação lógica e física dos ambientes, pela execução controlada de rotinas de banco e pela redução de intervenções diretas nos clusters. Dessa forma, os procedimentos metodológicos foram articulados às dimensões centrais da investigação, permitindo examinar os efeitos da solução sobre a automação, a rastreabilidade, a confiabilidade e a organização operacional do Proinfe.

3.1. Metodologia de Desenvolvimento e Gestão das Atividades

O desenvolvimento foi conduzido com apoio da metodologia ágil Scrum, framework utilizado para organizar o trabalho em ciclos iterativos, favorecer adaptação contínua e tornar visível o avanço das atividades (Sutherland, 2014). O Scrum é sustentado pelos pilares de transparência, inspeção e adaptação (Schwaber; Sutherland, 2020). Esses pilares foram relevantes para a condução da intervenção, pois permitiram acompanhar o estado das tarefas, revisar decisões técnicas ao longo do processo e ajustar prioridades conforme surgiam impedimentos relacionados à infraestrutura, aos pipelines e à implantação dos serviços.

No Proinfe, as atividades foram planejadas em sprints quinzenais. A cada ciclo, eram definidos itens de trabalho relacionados à preparação da infraestrutura, ajustes nos repositórios, configuração de pipelines, revisão de manifestos Kubernetes, validação dos ambientes e correção de falhas observadas durante a implantação. As reuniões de acompanhamento permitiram verificar o progresso da sprint, identificar bloqueios e redistribuir tarefas quando necessário. Ao final de cada ciclo, a revisão das entregas e a retrospectiva contribuíram para avaliar o que havia sido concluído, registrar dificuldades técnicas e planejar melhorias para o ciclo seguinte. Essa dinâmica favoreceu a evolução incremental da solução e reduziu incertezas em uma implantação que envolvia diferentes camadas técnicas.

O Jira foi utilizado para gestão das tarefas, acompanhamento das pendências, registro de atividades e organização do progresso da equipe. A documentação da Atlassian apresenta o Jira Software como uma ferramenta voltada ao suporte de práticas ágeis, incluindo planejamento de backlog, acompanhamento de sprints e visualização do progresso do time (Atlassian, 2026). No contexto desta pesquisa, o acompanhamento das atividades ocorreu por meio do backlog e do quadro Kanban, no qual as demandas eram organizadas conforme seu estado de execução, permitindo visualizar itens pendentes, em andamento, em revisão e concluídos. Essa organização favoreceu a transparência do trabalho e permitiu relacionar demandas técnicas, responsáveis e andamento das atividades de automação e infraestrutura. A comunicação da equipe ocorreu pelo Discord, utilizado para reuniões remotas, alinhamentos rápidos e discussão de impedimentos durante as sprints, complementando o registro formal das tarefas no Jira.

3.2. Arquitetura da Aplicação e Tecnologias de Apoio

A aplicação analisada foi composta por três microserviços de backend desenvolvidos em NestJS, por uma aplicação frontend em React e por serviços auxiliares de infraestrutura. O NestJS foi utilizado por oferecer uma estrutura para desenvolvimento de aplicações Node.js server-side escaláveis e organizadas em módulos (NestJS, 2026). No Proinfe, essa estrutura foi aplicada aos serviços de autenticação, secretaria e organização escolar, permitindo separar responsabilidades funcionais e organizar regras de negócio de acordo com os domínios do sistema.

O frontend foi desenvolvido em React, biblioteca voltada à construção de interfaces a partir de componentes reutilizáveis (React, 2026). Essa aplicação compôs a camada de interação dos usuários com o sistema, enquanto os microserviços disponibilizaram as APIs responsáveis por autenticação, cadastros, secretaria e organização escolar. Entre os serviços auxiliares, foram utilizados bancos MariaDB, sistema relacional que fornece acesso aos dados por meio de SQL (MariaDB, 2026); Redis, empregado como serviço de apoio para dados em memória e estruturas de chave-valor (Redis, 2026); e RabbitMQ, utilizado como broker de mensagens para apoiar fluxos assíncronos da aplicação (RabbitMQ, 2026).

3.3. Infraestrutura de Virtualização e Cluster K3s

A infraestrutura foi executada em servidores físicos gerenciados por Proxmox, com os clusters K3s em máquinas virtuais. O Proxmox VE é uma plataforma de virtualização utilizada para administrar servidores físicos, máquinas virtuais e containers, permitindo consolidar recursos computacionais e criar ambientes isolados para

diferentes finalidades (Proxmox, 2026). No estudo, essa camada de virtualização foi responsável por hospedar as máquinas virtuais que formaram os clusters Kubernetes. Sobre essas máquinas virtuais, foi adotado o K3s, distribuição Kubernetes leve adequada a cenários com menor complexidade operacional, ambientes de borda, desenvolvimento, CI e produção simplificada (K3s, 2026).

No Kubernetes, um cluster corresponde ao conjunto de máquinas responsáveis por executar e coordenar as aplicações containerizadas. Esse conjunto é normalmente composto por nós de controle e nós de trabalho. Os nós de controle concentram funções de gerenciamento, como manutenção do estado do cluster, agendamento de cargas e comunicação com a API do Kubernetes; os nós de trabalho executam os pods das aplicações, isto é, as cargas efetivas dos microserviços, frontend e serviços auxiliares. Essa separação permitiu organizar melhor a infraestrutura e distinguir as funções administrativas do cluster das cargas de aplicação.

Foram organizados dois contextos principais de execução. Desenvolvimento e homologação compartilharam um cluster K3s com separação lógica por namespaces, enquanto produção foi mantida em cluster separado. Os ambientes de desenvolvimento e homologação utilizaram os namespaces *default* e *stage*; produção utilizou o namespace *production*. Essa escolha metodológica permitiu avaliar a implantação de um fluxo DevOps tanto em ambiente compartilhado quanto em ambiente produtivo isolado, combinando separação lógica para validações internas e isolamento físico-operacional para o ambiente mais crítico.

3.4. Automação de Build e Publicação com GitLab CI/CD

O GitLab foi utilizado como repositório de código-fonte e como plataforma de automação de pipelines. O GitLab CI/CD permite definir pipelines como código, organizando jobs, estágios, regras de execução e dependências em arquivos versionados (GitLab, 2026a). No Proinfe, os pipelines foram estruturados para preparar variáveis de ambiente, construir imagens Docker, autenticar no registry, publicar as imagens e atualizar o repositório de manifestos Kubernetes. O fluxo adotado contemplou envio automático das alterações para o ambiente de desenvolvimento, enquanto a promoção para homologação e produção permaneceu condicionada a etapas manuais, de modo a preservar maior controle sobre as versões encaminhadas aos ambientes mais críticos.

Na política operacional adotada, os jobs de homologação e produção eram manuais e ficavam disponíveis apenas em pipelines da branch *main*, configurada como branch protegida. Para que uma alteração chegasse a essa branch, era necessário passar por merge request. A execução desses jobs era restrita a usuários com papel *Owner* ou *Maintainer* no projeto e ocorria após pipeline concluída, imagem versionada e teste prévio no ambiente de desenvolvimento. Assim, a etapa manual funcionava como autorização controlada para promover versões já construídas e testadas, sem retomada de comandos de deploy diretamente no servidor. Como o controle era realizado pelo papel do usuário no repositório, não havia segregação formal entre quem desenvolvia e quem promovia a versão; além disso, a validação funcional em desenvolvimento funcionava como prática operacional antes do acionamento manual. A Tabela 1 sintetiza essa política.

Tabela 1. Política operacional de promoção manual para homologação e produção.

Ambiente de destino	Origem permitida	Gatilho/regra do job	Papel autorizado	Evidência mínima	Mecanismo proteç
Homologação	Branch *main* protegida.	Job manual disponível apenas na pipeline da *main*.	*Owner* ou *Maintainer*.	Pipeline concluída, imagem versionada e teste em desenvolvimento.	Entrada *main* merge request autoriz o papel projeto

⚠ Esta tabela possui muitas colunas e foi cortada para impressão. Para visualizá-la completa, acesse o artigo original em: https://revistatopicos.com.br/artigos/implementacao-de-devops-no-proinfe-automacao-e-gerenciamento-com-gitlab-argo-cd-e-k3s?noblockage

O Docker foi utilizado para empacotar os microserviços e o frontend em imagens de container. Na solução proposta, as imagens funcionaram como unidades padronizadas de distribuição da aplicação, reunindo código, dependências e configurações necessárias para execução em diferentes ambientes. Essa abordagem dialoga com a definição do Docker de construir, compartilhar e executar aplicações containerizadas em máquinas físicas, virtuais ou ambientes de nuvem (Docker, 2026). As imagens geradas pelos pipelines foram publicadas no GitLab Container Registry, recurso integrado ao GitLab para armazenamento de imagens por projeto (GitLab, 2026b). Para favorecer rastreabilidade, as tags das imagens foram associadas ao ambiente e ao identificador da pipeline, permitindo relacionar uma imagem publicada à execução que a originou.

3.5. GitOps e Sincronização de Manifestos com Argo CD

O Argo CD foi utilizado para implementar a abordagem GitOps. Nessa abordagem, os manifestos versionados no Git representam o estado desejado da aplicação, enquanto a ferramenta de sincronização compara esse estado com o estado real do cluster. No contexto do Kubernetes, o Argo CD atua como uma solução declarativa de entrega contínua, monitorando repositórios Git e aplicando nos clusters as configurações definidas nos manifestos (Argo Project, 2026a). Sua integração com automações de CI permite que pipelines atualizem manifestos enquanto o Argo CD executa a etapa de sincronização no ambiente de destino (Argo Project, 2026b).

No fluxo implantado, os pipelines clonaram o repositório de manifestos *k3s*, alteraram os arquivos YAML do serviço e ambiente correspondente, registraram a alteração em commit e enviaram a atualização para a branch principal. A partir desse ponto, o Argo CD monitorou o repositório e sincronizou os manifestos com os clusters K3s. Desenvolvimento e homologação foram organizados em diretórios e namespaces próprios no mesmo cluster; produção foi direcionada ao cluster separado. Essa etapa foi essencial para transformar o deploy em um processo declarativo, rastreável e auditável.

3.6. Execução de Migrations, Seeders e Rotinas de Banco

As rotinas de migrations e seeders dos microserviços NestJS exigiram tratamento metodológico próprio, pois envolviam alterações de banco de dados em um ambiente com múltiplas réplicas de aplicação. O TypeORM foi utilizado no projeto para gerenciamento dessas rotinas; sua documentação prevê comandos para execução e reversão de migrations por meio da CLI (TypeORM,

2026). No contexto do Proinfe, a execução automática dessas rotinas diretamente na inicialização de containers poderia gerar concorrência quando múltiplas réplicas da API fossem criadas simultaneamente.

Para evitar essa condição de concorrência, o fluxo foi reorganizado de modo que migrations e seeders fossem executados por meio de Jobs do Kubernetes antes da atualização dos Deployments. O pipeline criava um Job utilizando a nova imagem da aplicação e executava os comandos necessários de banco, como *npm run typeorm migration:run* e comandos de seed. Apenas após a conclusão bem-sucedida do Job o pipeline prosseguia com a atualização dos Deployments da API. Esse procedimento tornou a etapa de banco mais controlada e impediu execuções simultâneas disparadas pela inicialização de múltiplos pods da API.

Para registrar salvaguardas automatizadas e observações de controle, a Tabela 2 resume o comportamento observado no desenho dos Jobs de banco. A sequência implantada preparava a imagem e as variáveis do ambiente de destino, criava o Job no namespace correspondente, executava os comandos de migration e seed, aguardava a conclusão do Job e somente então atualizava os manifestos dos Deployments. Nos manifestos analisados, os Jobs foram configurados com *parallelism*=1, *completions*=1, *backoffLimit*=1, *activeDeadlineSeconds*=900 e t*tlSecondsAfterFinished*=86400; além disso, a pipeline adotava timeout: 15m, equivalente a 900 segundos. Esses parâmetros tornam o comportamento de execução, retry, prazo máximo, encerramento por tempo e limpeza mais auditável.

Tabela 2. Controles observados nos Jobs de migrations, seeders e rotinas de banco.

Etapa	Comportamento observado no Proinfe	Controle ou observação operacional
Criação do Job	O pipeline criava um Job no namespace do ambiente usando a nova imagem da aplicação e as variáveis correspondentes.	A execução ficava isolada ao namespace do ambiente, favorecendo controle e rastreabilidade da rotina.
Ordem de execução	O Job de banco era executado antes da atualização dos manifestos dos Deployments da API.	O bloqueio da etapa seguinte pelo pipeline preservava a ordem planejada e evitava atualização da aplicação antes da finalização da rotina de banco.
Critério de sucesso	O pipeline aguardava a conclusão bem-sucedida do Job antes de atualizar os Deployments.	A conclusão do Job funcionava como critério automatizado de avanço; validações funcionais pós-migration podem complementar esse controle.
Falha ou timeout	Quando o Job falhava ou não concluía no tempo esperado pela pipeline, o deploy era interrompido antes da substituição dos pods da aplicação.	Os manifestos documentaram *backoffLimit*=1 e *activeDeadlineSeconds*=900; a pipeline adotava timeout: 15m, equivalente a 900 segundos, e os logs apoiavam o diagnóstico em caso de falha.
Logs e diagnóstico	A análise de falhas utilizava os logs do Job e os registros da pipeline.	Esses registros apoiavam o diagnóstico operacional; retenção histórica e coleta centralizada são evoluções possíveis de observabilidade, não pré-requisitos para o fluxo implantado.

Limpeza do Job	A limpeza automática dos Jobs foi documentada com *ttlSecondsAfterFinished*= 86400, preservando o registro temporário da execução antes da expiração.	O TTL reduzia acúmulo de objetos concluídos no cluster; a coleta histórica de logs pode complementar diagnósticos posteriores.
Rollback	O fluxo preservava os Deployments na versão anterior quando a etapa de banco falhava antes da atualização dos manifestos.	A aplicação permanecia protegida contra atualização com rotina de banco incompleta; reversões de banco eram tratadas pelas migrations do TypeORM ou por correção controlada quando necessário.
Idempotência de seeders	A pipeline executava os seeders no mesmo Job das rotinas de banco.	A segurança da execução repetida era tratada na implementação idempotente dos seeders.
Prevenção de concorrência	A execução por Job antes dos Deployments impediu execuções simultâneas disparadas pela inicialização de múltiplos pods da API.	A promoção manual para homologação e produção reduziu o risco de concorrência nesses ambientes.

3.7. Separação de Ambientes, Exposição e Escalabilidade

A separação dos ambientes foi realizada por combinação de namespaces e clusters distintos. Namespaces são recursos do Kubernetes voltados à organização lógica de objetos dentro de um cluster (Kubernetes, 2026c). No Proinfe, desenvolvimento e homologação foram mantidos no mesmo cluster, mas em namespaces diferentes, enquanto produção foi mantida em cluster separado. Essa separação permitiu validar alterações antes da

promoção para ambientes mais críticos e reduziu o risco de confusão entre recursos de teste e produção.

A exposição externa dos serviços foi realizada por meio de Ingress. O recurso Ingress do Kubernetes permite gerenciar acesso HTTP e HTTPS externo aos serviços internos do cluster por regras de host e caminho (Kubernetes, 2026b). Como controlador, foi utilizado o Nginx Ingress Controller, que opera em torno do recurso Ingress do Kubernetes e utiliza configurações próprias para processar o roteamento (Ingress-Nginx, 2026). No Proinfe, foram criadas regras para expor o frontend e encaminhar rotas da API aos microserviços de autenticação, secretaria e organização escolar.

Também foram configurados recursos de disponibilidade e escalabilidade. O Horizontal Pod Autoscaler (HPA) ajusta automaticamente a quantidade de réplicas de um Deployment com base em métricas, como uso médio de CPU (Kubernetes, 2026a). Nos microserviços principais, o HPA foi configurado com mínimo de dois pods, máximo de sete pods e acionamento quando a média de CPU ultrapassasse 70%. Os manifestos também documentaram *requests* e *limits* de CPU e memória para os três microserviços. Já probes de saúde, estratégia RollingUpdate e PodDisruptionBudget foram aplicados aos serviços de secretaria e organização escolar, pois esses módulos concentram maior complexidade operacional, rotas de negócio mais sensíveis e maior necessidade de preservação de disponibilidade durante atualizações e interrupções voluntárias. A estratégia RollingUpdate permite atualizar gradualmente os pods de um Deployment, substituindo instâncias antigas por novas sem interromper toda a aplicação ao mesmo tempo. O PodDisruptionBudget, por sua vez, define limites mínimos de disponibilidade durante interrupções voluntárias, ajudando a

preservar réplicas suficientes em operações de manutenção, drenagem de nós ou atualizações controladas. Esses recursos foram analisados como evidências configuracionais de preparação operacional para disponibilidade e escalabilidade, respeitando a criticidade diferenciada entre serviços e sem mensuração quantitativa direta de redução de indisponibilidade ou de tempo de recuperação.

3.8. Procedimentos de Coleta, Organização e Análise dos Dados

A coleta de dados foi realizada a partir dos registros técnicos gerados durante a implementação. Foram observados os arquivos de configuração dos pipelines, os commits no repositório de manifestos, as tags de imagens publicadas no GitLab Container Registry, a estrutura de namespaces, os manifestos aplicados pelo Argo CD, os Jobs de banco e as configurações de Ingress, HPA, probes, RollingUpdate e PodDisruptionBudget. Esses elementos funcionaram como instrumentos de coleta por representarem evidências diretas do processo de implantação.

A organização dos dados ocorreu por categorias de análise relacionadas ao objetivo da pesquisa: automação, rastreabilidade, separação de ambientes, segurança operacional das rotinas de banco, disponibilidade e limitações do processo. A análise foi qualitativa e interpretativa, comparando o fluxo manual anterior com o fluxo DevOps implantado. As contagens totais de pipelines executados, commits de manifestos e Jobs de banco observados no período não foram consolidadas como métricas quantitativas; por isso, os artefatos foram analisados como evidências documentais e operacionais do fluxo implantado. Também não foram utilizadas métricas quantitativas históricas completas, como tempo médio de

deploy, frequência de implantação ou taxa de falha; por isso, esses indicadores foram delimitados como perspectivas de avaliação posterior, especialmente à luz das métricas de desempenho de entrega discutidas por estudos de DevOps (Forsgren; Humble; Kim, 2018).

4. RESULTADOS E DISCUSSÕES

A implementação do fluxo DevOps permitiu reorganizar o processo de entrega das aplicações do Proinfe, substituindo um procedimento manual por um fluxo baseado em pipelines, imagens versionadas, manifestos declarativos e sincronização com Kubernetes. O GitLab CI/CD automatizou a construção e a publicação das imagens, enquanto o repositório *k3s* passou a concentrar os manifestos de desenvolvimento, homologação e produção. Com isso, o deploy deixou de depender de acesso direto ao servidor e passou a gerar registros associados à execução da pipeline, à imagem publicada e ao commit de atualização dos manifestos.

O fluxo de promoção entre ambientes também contribuiu para maior controle operacional. Conforme a política operacional definida na Tabela 1, o envio automático para desenvolvimento acelerou a validação inicial das mudanças, enquanto homologação e produção permaneceram condicionadas ao acionamento manual de jobs na branch *main* protegida. Essa organização manteve o deploy automatizado, mas preservou uma etapa de autorização antes da promoção para ambientes mais críticos.

A rastreabilidade foi fortalecida principalmente pela associação entre GitLab CI/CD, GitLab Container Registry, repositório *k3s* e Argo

CD. As tags das imagens, vinculadas ao ambiente e ao identificador da pipeline, permitiram identificar a origem de cada build. Os commits no repositório de manifestos registraram as alterações aplicadas aos ambientes, enquanto o Argo CD tornou visível a relação entre o estado desejado descrito no Git e o estado efetivamente sincronizado no cluster. Dessa forma, tornou-se possível acompanhar qual arquivo foi alterado, qual imagem foi implantada, qual pipeline originou a mudança e em qual ambiente a versão estava declarada e sincronizada.

A Tabela 3 apresenta exemplos sintéticos dos artefatos utilizados nessa rastreabilidade, construídos a partir do padrão observado nos registros do período de julho de 2024 a dezembro de 2025. Esses valores preservam a estrutura dos registros, mas não correspondem a identificadores reais nem a uma amostra selecionada do período. O objetivo da tabela não é apresentar métricas quantitativas de desempenho, mas explicitar a cadeia documental que permite relacionar uma alteração de código à imagem publicada, ao manifesto atualizado e ao estado sincronizado pelo Argo CD no ambiente de destino.

Tabela 3. Exemplos sintéticos da cadeia de evidências utilizada para rastrear implantações.

Evidência	**Exemplo sintético baseado no padrão observado**	**Função na rastreabilidade**
Pipeline GitLab CI/CD	pipeline-547	Identifica a execução responsável por construir a imagem, publicar o artefato e acionar a atualização dos manifestos.

Tag da imagem	dev-547	Relaciona a imagem publicada no GitLab Container Registry ao ambiente de destino e à pipeline que originou o build.
Commit no repositório *k3s*	update auth dev dev-547	Registra a alteração do manifesto Kubernetes, incluindo serviço, ambiente e versão de imagem implantada.
Aplicação no Argo CD	auth-dev	Permite visualizar a sincronização entre o manifesto versionado no Git e o estado aplicado no cluster K3s.
Status de sincronização	Synced/OutOfSync	Indica se o estado desejado descrito no Git corresponde ao estado observado no cluster.

A Figura 1 apresenta uma captura intermediária do Argo CD para o ambiente de produção, reunindo os três microserviços backend e a aplicação frontend sincronizados a partir dos manifestos versionados. A imagem é utilizada para evidenciar a visibilidade fornecida pelo Argo CD sobre aplicações, serviços e recursos sincronizados, mas não representa integralmente o estado final da solução. No momento do registro, os serviços backend ainda apareciam com uma réplica cada, pois a execução de migrations por Jobs do Kubernetes e a configuração final de escalabilidade ainda não haviam sido incorporadas ao fluxo.

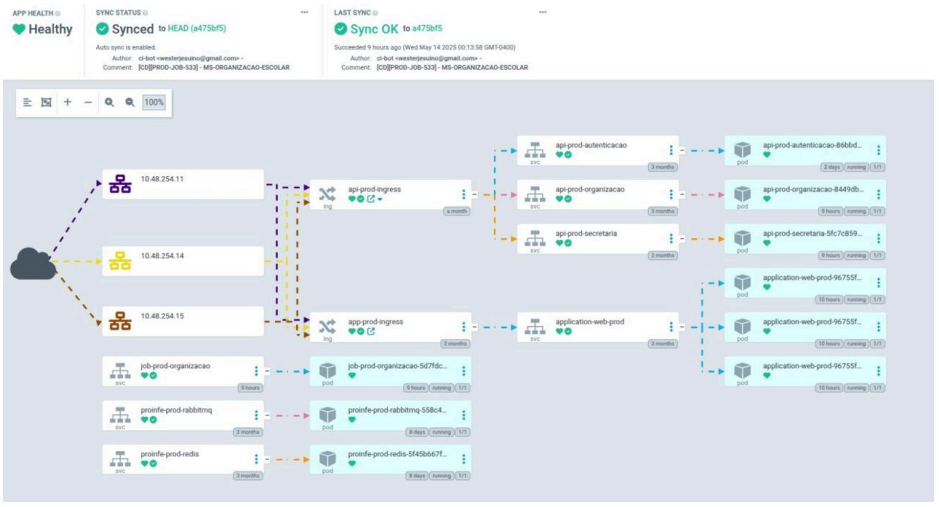

Figura 1. Captura intermediária do Argo CD para o ambiente de produção do Proinfe.

A separação dos ambientes foi outro resultado relevante. Desenvolvimento e homologação permaneceram no mesmo cluster K3s, porém organizados em namespaces distintos, *default* e *stage*. A produção foi mantida em cluster separado, no namespace *production*. Essa organização reduziu o risco de confusão entre recursos de teste e produção, facilitou a leitura da infraestrutura e reforçou o isolamento operacional do ambiente produtivo. A separação de rotas por hosts e Ingresses próprios também contribuiu para evitar direcionamentos incorretos de tráfego entre ambientes.

A Figura 2 sintetiza essa organização, destacando o encadeamento entre GitLab CI/CD, registro de imagens, repositório de manifestos, Argo CD e os clusters K3s utilizados nos ambientes de desenvolvimento, homologação e produção.

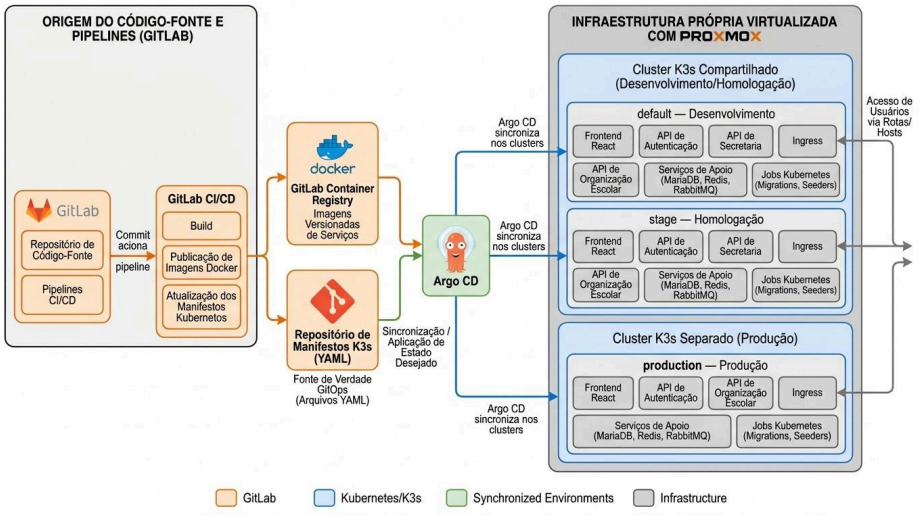

Figura 2. Infraestrutura de ambientes do Proinfe com separação entre desenvolvimento, homologação e produção.

A execução de migrations e seeders por meio de Jobs do Kubernetes antes da atualização dos Deployments tornou o processo de implantação mais seguro. Essa reorganização impediu execuções simultâneas disparadas pela inicialização de múltiplas réplicas das APIs NestJS, condição que poderia ocorrer caso os comandos fossem executados diretamente na inicialização dos containers. O comportamento observado também preservava os Deployments na versão anterior quando o Job de banco não era concluído com sucesso, evitando a substituição dos pods da aplicação antes da finalização da etapa de schema e seed. Os manifestos analisados documentaram *backoffLimit*=1, *activeDeadlineSeconds*=900 e *ttlSecondsAfterFinished*=86400, enquanto a pipeline adotava timeout: 15m, equivalente a 900 segundos, para tornar retry, prazo máximo, encerramento por tempo e limpeza automática mais auditáveis. A centralização histórica dos logs, a classificação de falhas e a manutenção da idempotência dos seeders são evoluções possíveis de observabilidade e segurança, sem descaracterizar os controles já implantados. Assim, a etapa de banco passou a ocorrer de forma mais controlada antes da substituição gradual das aplicações.

No aspecto de disponibilidade, os resultados foram tratados como evidências configuracionais, pois o estudo não mensurou quantitativamente indisponibilidade, taxa de falha ou tempo de recuperação. Os manifestos indicaram preparação do ambiente produtivo por meio de HPA, *requests* e *limits* nos três microserviços, separação entre nós de controle e de trabalho e uso de probes, RollingUpdate e PodDisruptionBudget nos serviços de secretaria e organização escolar, que concentram maior complexidade operacional. Assim, a discussão não repete a descrição metodológica dos parâmetros, mas interpreta sua presença como sinal de organização operacional e de priorização dos componentes mais sensíveis. No caso do PodDisruptionBudget, o valor *minAvailable*=1 foi tratado como salvaguarda mínima inicial; sua suficiência operacional depende de medições de carga e pode ser revisada para valores absolutos ou percentuais mais conservadores conforme o consumo real. A Tabela 4 sintetiza os valores observados nos manifestos.

Tabela 4. Parâmetros de disponibilidade e escalabilidade documentados no fluxo implantado.

Recurso	**Valor no Proinfe documentado no artigo**	**Limitação para avaliação e reprodução**
HPA	Autenticação, secretaria e organização escolar com mínimo de dois pods, máximo de sete pods e alvo de CPU média em 70%.	A avaliação quantitativa depende de monitoramento contínuo de CPU, comportamento de carga e comparação histórica de indisponibilidade.
Nós de controle	Criados com a flag –node-taint e o *taint* CriticalAddonsOnly= true:NoExecute.	A efetividade do isolamento depende de os workloads da aplicação não possuírem *tolerations* compatíveis com

		esse *taint* e de verificação operacional do agendamento dos pods.
Requests e *limits*	Autenticação: 100m/256Mi e 500m/512Mi. Secretaria: 200m/384Mi e 1000m/768Mi. Organização escolar: 250m/512Mi e 1000m/1Gi.	Os valores tornam o dimensionamento auditável, mas sua suficiência depende de testes de carga, consumo real e monitoramento contínuo.
Probes de saúde	Secretaria com readiness, liveness e startup em /health-sec na porta 3002. Organização escolar com liveness em /healthz e readiness em /ready na porta 3003.	Aplicadas aos serviços de secretaria e organização escolar pela maior complexidade operacional e pela necessidade de verificar end-points representativos de saúde.
RollingUpdate	Secretaria e organização escolar com *maxUnavailable*=0 e *maxSurge*=1; organização escolar com *terminationGracePeriodSeconds*=30.	Aplicada aos serviços de maior complexidade para reduzir risco durante atualização gradual e preservar a continuidade das rotinas mais sensíveis.
PodDisruptionBudget	Secretaria e organização escolar com *minAvailable*=1 para preservar pelo menos um pod disponível em interrupções voluntárias.	Tratado como salvaguarda mínima inicial; sua suficiência depende de medições de carga e pode ser revista conforme o consumo real.

O controle operacional do processo de entrega foi fortalecido pela padronização do processo de build, versionamento e implantação. A solução implantada tornou o processo mais reproduzível e auditável, enquanto práticas como análise estática de código, testes automatizados, monitoramento detalhado, análise sistemática de falhas de pipeline e métricas DORA permanecem como

possibilidades de amadurecimento. Esses elementos podem ampliar, em etapas futuras, a avaliação sobre mantenabilidade, frequência de implantação, estabilidade e tempo de recuperação diante de falhas.

A integração com Discord foi analisada como tentativa de ampliar o feedback operacional do fluxo de deploy. A proposta era utilizar webhooks para enviar notificações automáticas a canais da equipe, recurso compatível com o envio de mensagens por requisições HTTP (Discord, 2026). Apesar da inclusão dos comandos nos jobs de deploy, a funcionalidade permaneceu em validação operacional. Esse resultado indica que a revisão de variáveis de ambiente, permissões, tokens, tratamento de erros nas chamadas HTTP e padronização das mensagens pode consolidar esse mecanismo como parte estável do processo.

Outra frente prevista para evolução foi o uso do KEDA para escalabilidade orientada a eventos no serviço assíncrono de organização escolar. Essa abordagem poderia permitir que o número de pods consumidores fosse ajustado de acordo com o tamanho da fila no RabbitMQ, evitando tanto subdimensionamento em momentos de maior volume quanto consumo desnecessário de recursos em períodos de baixa demanda. Como essa funcionalidade ficou fora do escopo implantado no período analisado, a análise dos resultados concentrou-se nos mecanismos efetivamente adotados, como HPA baseado em CPU, e na presença configuracional de probes, RollingUpdate e PodDisruptionBudget.

5. CONCLUSÃO

Este trabalho teve como objetivo implementar e analisar um fluxo DevOps para o Proinfe, considerando um cenário real composto por microserviços NestJS, frontend em React, serviços auxiliares, bancos MariaDB e ambientes Kubernetes executados com K3s em infraestrutura própria virtualizada com Proxmox. A partir da comparação qualitativa com o processo anterior, baseado em build local, acesso direto ao servidor e substituição manual de containers, observa-se que a solução implantada respondeu ao problema de pesquisa ao ampliar a automação, a rastreabilidade e a organização dos deploys nos ambientes de desenvolvimento, homologação e produção.

Os resultados indicam que a combinação entre GitLab CI/CD, GitLab Container Registry, repositório de manifestos, Argo CD e K3s tornou o fluxo de implantação mais padronizado e auditável. As imagens passaram a ser geradas e publicadas por pipelines, os manifestos Kubernetes passaram a registrar as alterações aplicadas a cada ambiente e o Argo CD passou a sincronizar o estado desejado descrito no Git com o estado executado nos clusters. Dessa forma, a implantação deixou de depender predominantemente de ações manuais no servidor e passou a produzir evidências documentais sobre a origem da imagem, a pipeline executada, o commit de atualização, a aplicação Argo CD e o ambiente afetado.

Além da automação, a solução contribuiu para maior segurança operacional. A execução de migrations e seeders por Jobs do Kubernetes antes da atualização dos Deployments reduziu o risco de execuções simultâneas disparadas pelos pods da aplicação e impediu a atualização dos Deployments quando a etapa de banco não era concluída com sucesso. A separação entre desenvolvimento e homologação por namespaces, associada à manutenção da

produção em cluster separado, também fortaleceu o isolamento entre ambientes e diminuiu a possibilidade de confusão entre recursos de teste e recursos produtivos. No aspecto de disponibilidade, os manifestos documentaram HPA, *requests* e *limits* para os três microserviços, enquanto probes de saúde, RollingUpdate e PodDisruptionBudget foram aplicados aos serviços de secretaria e organização escolar pela maior complexidade operacional desses módulos. Em conjunto com o *taint* dos nós de controle, esses elementos indicam preparação operacional inicial para disponibilidade e escalabilidade nos componentes mais sensíveis do sistema. Entretanto, a avaliação de seus efeitos sobre disponibilidade real, taxa de falha e tempo de recuperação permaneceu limitada pela ausência de métricas contínuas, testes de carga e comparação quantitativa histórica.

Como contribuição prática, a implementação oferece um modelo replicável para projetos educacionais ou institucionais que precisam evoluir de deploys manuais para um fluxo mais controlado, versionado e compatível com práticas GitOps. O uso do K3s mostrou-se adequado ao contexto do Proinfe por permitir a adoção de Kubernetes em infraestrutura própria, sem exigir dependência de serviços gerenciados em nuvem pública. Assim, a experiência indica que práticas DevOps podem ser adaptadas a cenários de recursos controlados, desde que haja padronização dos repositórios, separação clara dos ambientes, mecanismos de sincronização declarativa e documentação dos parâmetros operacionais adotados.

A avaliação também delimitou o escopo da análise realizada. Como não havia métricas históricas completas sobre tempo de deploy, frequência de implantação, taxa de falha ou tempo de recuperação antes da intervenção, a comparação entre o fluxo antigo e o fluxo

implantado permaneceu qualitativa. Como perspectivas para a continuidade da pesquisa, destacam-se a estruturação da coleta de métricas de entrega e operação, incluindo tempo de deploy, frequência de implantação, taxa de falhas e tempo médio de recuperação; a estabilização da integração com Discord; o aprimoramento de estratégias de rollback e reversão das rotinas de banco; o reforço de travas automáticas entre pipelines concorrentes; a centralização e classificação sistemática de logs dos Jobs de banco; a avaliação do KEDA para consumidores assíncronos; e a ampliação de testes automatizados, análise estática e monitoramento aos pipelines. Essas ações podem complementar a solução implantada e permitir uma avaliação mais objetiva dos impactos do DevOps na confiabilidade, na estabilidade e na evolução contínua do Proinfe.

REFERÊNCIAS BIBLIOGRÁFICAS

ARGO PROJECT. *Argo CD: Declarative GitOps CD for Kubernetes*. 2026a. Disponível em: https://argo-cd.readthedocs.io/. Acesso em: 17 jun. 2026.

ATLASSIAN. *Getting started with Jira Software*. 2026. Disponível em: https://confluence.atlassian.com/jirasoftware/getting-started-with-jira-software-1688898120.html. Acesso em: 29 jun. 2026.

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

CI Automation. 2026b. Disponível em: https://argo-cd.readthedocs.io/en/stable/user-guide/ci_automation/. Acesso em: 17 jun. 2026.

DI FRANCESCO, Paolo; LAGO, Patricia; MALAVOLTA, Ivano. Architecting with microser-vices: a systematic mapping study. *Journal of Systems and Software*, Amsterdam, v. 150, p. 77–97, 2019.

DISCORD. *Webhooks*. 2026. Disponível em: https://docs.discord.com/developers/platform/webhooks. Acesso em: 29 jun. 2026.

DOCKER. *What is Docker?*. 2026. Disponível em: https://docs.docker.com/get-started/docker-overview/. Acesso em: 29 jun. 2026.

FORSGREN, Nicole; HUMBLE, Jez; KIM, Gene. *Accelerate: The Science of Lean Software and DevOps*. Portland: IT Revolution Press, 2018.

GIL, Antonio Carlos. *Como elaborar projetos de pesquisa*. 4. ed. São Paulo: Atlas, 2002. GITLAB. *GitLab CI/CD pipelines*. 2026a. Disponível em: https://docs.gitlab.com/ci/pipelines/. Acesso em: 17 jun. 2026.

_____. *GitLab Container Registry*. 2026b. Disponível em: https://docs.gitlab.com/user/packages/container_registry/. Acesso em: 29 jun. 2026.

HUMBLE, Jez; FARLEY, David. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley, 2010.

INGRESS-NGINX. *Welcome - Ingress-Nginx Controller*. 2026. Disponível em: https://kubernetes.github.io/ingress-nginx/. Acesso em: 29 jun. 2026.

K3S. *K3s Documentation*. 2026. Disponível em: https://docs.k3s.io/. Acesso em: 17 jun. 2026.

KEDA. *Kubernetes Event-driven Autoscaling*. 2026. Disponível em: https://keda.sh/. Acesso em: 21 jun. 2026. Acesso em: 21 jun. 2026.

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

KUBERNETES. *Horizontal Pod Autoscaling*. 2026a. Disponível em: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. Acesso em: 21 jun. 2026.

_____. *Ingress*. 2026b. Disponível em: https://kubernetes.io/docs/concepts/services-networking/ingress/. Acesso em: 17 jun. 2026.

_____. *Namespaces*. 2026c. Disponível em: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. Acesso em: 17 jun. 2026.

LEITE, Leonardo et al. A survey of DevOps concepts and challenges. *ACM Computing Surveys*, New York, v. 52, n. 6, p. 1–35, 2019.

MARIADB. *Documentation*. 2026. Disponível em: https://mariadb.org/documentation/. Acesso em: 29 jun. 2026.

NESTJS. *Documentation*. 2026. Disponível em: https://docs.nestjs.com/. Acesso em: 29 jun. 2026.

OECD. *OECD Digital Education Outlook 2021: Push-ing the Frontiers with Artificial Intelligence, Blockchain and Robots*. Paris: OECD Publishing, 2021. Disponível em: https://www.oecd.org/en/publications/oecd-digital-education-outlook-2021_589b283f-en.html. Acesso em: 29 jun. 2026.

PROXMOX. *Documentation - Proxmox Virtual Environment*. 2026. Disponível em: https://www.proxmox.com/en/downloads/proxmox-virtual-environment/documentation. Acesso em: 29 jun. 2026.

RABBITMQ. *RabbitMQ Documentation*. 2026. Disponível em: https://www.rabbitmq.com/docs. Acesso em: 29 jun. 2026.

REACT. *React*. 2026. Disponível em: https://react.dev/. Acesso em: 29 jun. 2026. REDIS. *Redis data types*. 2026. Disponível em: https://redis.io/docs/latest/develop/data-types/. Acesso em: 29 jun. 2026.

RODRÍGUEZ, Pilar et al. Continuous deployment of software intensive products and services: a systematic mapping study. *Journal of Systems and Software*, Amsterdam, v. 123, p. 263–291, 2017.

SCHWABER, Ken; SUTHERLAND, Jeff. *The Scrum Guide: The Definitive Guide to Scrum*. 2020. Disponível em: https://scrumguides.org/scrum-guide.html. Acesso em: 20 jun. 2026.

SHAHIN, Mojtaba; BABAR, Muhammad Ali; ZHU, Liming. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, Piscataway, v. 5, p. 3909–3943, 2017.

SUTHERLAND, Jeff. *Scrum: The Art of Doing Twice the Work in Half the Time*. New York: Crown Business, 2014.

TYPEORM. *Migrations*. 2026. Disponível em: https://orkhan.gitbook.io/typeorm/docs/migrations. Acesso em: 29 jun. 2026.

UNESCO. *Global Education Monitoring Report 2023: Technology in education: a tool on whose terms?*. Paris: UNESCO, 2023. Disponível em: https://www.unesco.org/gem-report/en/publication/technology. Acesso em: 29 jun. 2026.

YIN, Robert K. *Estudo de caso: planejamento e métodos*. 5. ed. Porto Alegre: Bookman, 2015.