ADMINISTRAÇÃO CENTRAL

PRESIDÊNCIA

PORTARIA IBAMA Nº 177, DE 28 DE AGOSTO DE 2026

O PRESIDENTE DO INSTITUTO BRASILEIRO DO MEIO AMBIENTE E DOS RECURSOS NATURAIS RENOVÁVEIS (IBAMA), nomeado pela Portaria nº 724, de 15 de junho de 2026, no uso das atribuições que lhe foram conferidas pelo artigo 15, inciso V, do Anexo I do Decreto nº 12.130, de 7 de agosto de 2024, que aprovou a Estrutura Regimental do Ibama, e pelo artigo 217, inciso V, da Portaria Ibama nº 73, de 26 de maio de 2025, que aprovou o Regimento Interno do Ibama, e considerando o constante dos autos do processo SEI nº 02001.013721/2025-61, resolve:

Art. 1º Aprovar, na forma do Anexo I, a Metodologia DevOps Ibama, versão 1.0, destinada a orientar as equipes de tecnologia e de desenvolvimento de sistemas, nos âmbitos interno e externo do Ibama, com o objetivo de estabelecer padrões para a automação e a aceleração do ciclo de desenvolvimento e entrega de software, bem como aprimorar a colaboração e a comunicação entre as equipes de Desenvolvimento (Dev) e Operações (Ops), a fim de assegurar maior agilidade, segurança e automação em todo o ciclo de vida do desenvolvimento de sistemas no Ibama.

Art. 2º Esta Portaria entra em vigor na data de sua publicação.

ANEXO I

METODOLOGIA DEVOPS IBAMA

SUMÁRIO

1 Objetivo

2 Público-Alvo

3 Introdução

4 O que é DevOps?

5 Integração Contínua X Entrega Contínua

5.1 Integração Contínua (CI):

5.2 Entrega Contínua (CD):

6 Papéis das equipes

6.1 Equipe de desenvolvimento

6.1.1 Principais Responsabilidades da Equipe de Desenvolvimento:

6.2 Equipe de qualidade

6.2.1 Principais Responsabilidades da Equipe de Qualidade:

6.3 Equipe de operações

6.3.1 Principais Responsabilidades da Equipe de Operações:

7 Metodologia DevOps no Ibama

7.1 Ferramentas Utilizadas

7.1.1 Versionamento de Código (GitLab - v17.7.4)

7.1.2 CI/CD (GitLab - v17.7.4)

7.1.3 Container Registry (GitLab - v17.7.4)

7.1.4 Qualidade de Software (SonarQube - v10.7)

7.1.5 Scanner de Vulnerabilidade (Trivy)7.1.6 Cluster Kubernetes

7.2 Diagrama Ambiente Ibama

7.3 Arquitetura Pipeline

7.4 Pipeline de CI/CD

7.5 Pipeline PLSQL

7.6 Observabilidade e Monitoramento

8 Permissões de Acesso

8.1 Gitlab

8.2 SonarQube

8.3 Kubernetes

9 Pipeline

9.1 Template

9.2 Deploy em ambiente Kubernetes

9.2.1 Deployment

9.2.2 Route

9.2.3 Service

9.3 Sistemas não conteinerizado

10 Qualidade - Sonarqube

10.1 Critérios de Qualidade

11 Tratamento de Dados

12 Conclusão

1.OBJETIVO

Este documento tem como objetivo estruturar e definir diretrizes para a implementação da metodologia DevOps no Ibama, orientando as equipes de tecnologia e desenvolvimento de sistemas nos âmbitos interno e externo. A iniciativa visa padronizar processos, otimizar fluxos de trabalho e promover a automação dos processos de desenvolvimento de sistemas, garantindo maior eficiência, colaboração e qualidade na entrega de soluções tecnológicas.

2.PÚBLICO-ALVO

Destina-se às equipes técnicas de tecnologia do Ibama, responsáveis pelo suporte, pela manutenção e pelo desenvolvimento de qualquer sistema ou software.

3.INTRODUÇÃO

Serão adotadas práticas modernas de Integração Contínua (CI) e Entrega Contínua (CD), garantindo maior confiabilidade, escalabilidade e segurança às aplicações. Essas práticas estabelecerão processos automatizados e eficientes para a gestão das aplicações desenvolvidas pelas fábricas de software no Ibama, reduzindo a necessidade de intervenções manuais e aumentando a eficiência operacional.

A implementação desse modelo visa otimizar os ciclos de desenvolvimento, reduzir falhas em produção e fortalecer a colaboração entre as equipes de desenvolvimento e de operação. Além disso, busca-se consolidar um ecossistema tecnológico mais resiliente, ágil e sustentável, alinhado às melhores práticas de engenharia de software e de DevOps.

4.O QUE É DEVOPS?

DevOps é uma abordagem que une filosofias culturais, práticas e ferramentas para aumentar a colaboração entre as equipes de desenvolvimento e operações. O objetivo é acelerar a entrega de software, tornando o processo mais eficiente, confiável e automatizado. Ao integrar desenvolvimento (Dev) e operações (Ops), as equipes conseguem construir, testar e implantar aplicativos e serviços de maneira mais ágil, superando as limitações dos métodos tradicionais de desenvolvimento e gerenciamento de infraestrutura.

A metodologia DevOps enfatiza a automação, a colaboração contínua e a melhoria constante, permitindo que as empresas otimizem seus produtos e serviços com mais rapidez e qualidade. Isso reduz o tempo entre a concepção de uma ideia e sua implementação prática, garantindo maior inovação e resposta às necessidades institucionais e dos usuários.

Com a implementação de um modelo de DevOps, as equipes de desenvolvimento e operações trabalham em conjunto, exigindo uma comunicação frequente entre as equipes. Os engenheiros trabalham durante todo o ciclo de vida do aplicativo, da fase de desenvolvimento e testes à fase de implantação e operações.

5.INTEGRAÇÃO CONTÍNUA X ENTREGA CONTÍNUA

A Integração Contínua (CI) e a Entrega Contínua (CD) são práticas essenciais no ambiente DevOps, visando aprimorar a qualidade do software e acelerar sua entrega ao usuário final. Embora estejam inter-relacionadas, cada uma possui características e objetivos específicos.

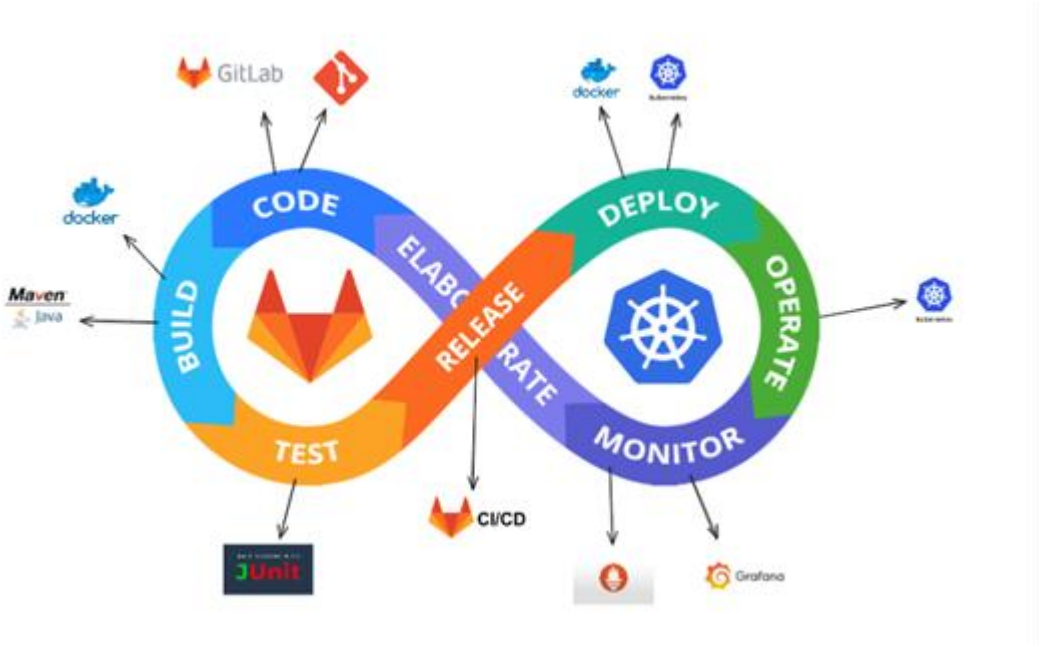

Figura 1: CI/CD - Ibama

5.1. Integração Contínua (CI):

A CI é uma prática de desenvolvimento de software na qual os(as) desenvolvedores(as) frequentemente integram suas alterações de código em um repositório central. Após cada integração, são realizadas compilações e testes automatizados com o objetivo de identificar rapidamente possíveis erros ou conflitos. Essa prática assegura que o código integrado seja mantido em condições adequadas de funcionamento e qualidade.

5.2. Entrega Contínua (CD):

A CD expande os princípios da CI ao automatizar a entrega de aplicações em ambientes de produção, de maneira segura e ágil. Após a fase de integração e testes, o código é automaticamente preparado para implantação,garantindo maior confiabilidade e rastreabilidade do processo, bem como sua permanente disponibilidade para liberação aos usuários finais.

6. PAPÉIS DAS EQUIPES

6.1. Equipe de desenvolvimento

A equipe de desenvolvimento desempenha um papel fundamental dentro da metodologia DevOps, sendo responsável pela implementação de código de acordo com as melhores práticas de engenharia de software e diretrizes arquiteturais definidas na documentação de Procedimento Operacional Padrão (POP) do Ibama. O código deve ser desenvolvido de forma estruturada, eficiente e sustentável, visando a escalabilidade e facilidade de manutenção.

Além da implementação, os(as) desenvolvedores(as) devem assegurar a qualidade do código por meio da criação de testes unitários. Esses testes garantem a integridade do sistema ao longo do pipeline de CI/CD, possibilitando a detecção antecipada de falhas e reduzindo o risco de incidentes em ambientes produtivos.

Outro aspecto essencial é a containerização da aplicação. A equipe deve manter e atualizar continuamente o Dockerfile para os serviços de front-end e back-end, garantindo que todas as imagens sejam corretamente gerenciadas e versionadas no registro de containers. Essa abordagem padroniza o ambiente de execução, facilita a reprodutibilidade dos ambientes, simplifica a geração de builds e otimiza o processo de implantação.

O gerenciamento de dependências também é uma responsabilidade crítica do time de desenvolvimento. Para garantir consistência e segurança, todas as bibliotecas e pacotes utilizados no projeto devem ser devidamente documentados e versionados em arquivos de dependência. No Ibama, utiliza-se o Nexus Repository Manager para armazenar artefatos internamente, evitando dependências externas desnecessárias e reduzindo riscos de falhas por indisponibilidade de repositórios públicos. Dessa forma, é possível garantir maior controle sobre versões, evitar vulnerabilidades conhecidas e otimizar a performance dos builds.

Por fim, o versionamento do código deve ser realizado de forma estruturada através do GitLab, seguindo estratégias de branches bem definidas. O código-fonte deve ser submetido via commit e push para repositórios específicos, permitindo que a equipe de DevOps automatize e orquestre os processos de build, testes e deploy dentro da pipeline de CI/CD. Esse fluxo assegura um ciclo de desenvolvimento contínuo, eficiente e confiável, promovendo a estabilidade e a escalabilidade da aplicação em produção.

6.1.1. Principais responsabilidades da equipe de desenvolvimento:

6.1.1.1. Implementação: Desenvolver código seguindo as melhores práticas de engenharia de software e as diretrizes arquiteturais estabelecidas. O código deve ser modular, eficiente e sustentável para garantir escalabilidade e fácil manutenção.

6.1.1.2. Testes Unitários: Implementar testes automatizados, assegurando uma cobertura adequada do código e possibilitando a validação contínua ao longo do pipeline de integração e entrega contínua (CI/CD).

6.1.1.3. Conteinerização: Garantir que a aplicação esteja sempre conteinerizada, mantendo o Dockerfile atualizado para os serviços de Front-end e Back-end. Isso facilita a geração de imagens e o gerenciamento de versões no registro de containers, garantindo consistência entre os ambientes de desenvolvimento, homologação e produção.

6.1.1.4. Gerenciamento de Dependências: Garantir que as dependências do projeto sejam bem documentadas e versionadas corretamente. Armazenar artefatos internamente no Ibama usando Nexus Repository Manager, garantindo maior controle sobre versões e segurança. Isso evita conflitos e melhora a rastreabilidade de versões ou falhas de repositórios públicos.

6.1.1.5. Versionamento & CI/CD: Submeter o código ao GitLab utilizando práticas adequadas de commit e push em branches específicas, permitindo que a equipe de DevOps automatize o ciclo de build, testes e deploy por meio da pipeline.

6.2. Equipe de qualidade

A equipe de qualidade desempenha um papel essencial na garantia da integridade e estabilidade do sistema ao longo do ciclo de vida do desenvolvimento, desde a codificação até a

produção. Dentro da metodologia DevOps, a equipe de qualidade deve atuar de forma proativa para garantir que as entregas atendam aos requisitos de performance, funcionalidade e segurança, minimizando riscos e falhas em ambientes produtivos.

O trabalho da equipe de qualidade se inicia com a definição de testes automatizados, que devem ser executados em todas as fases da pipeline de CI/CD para validar o código de forma contínua. Isso inclui testes unitários, testes de integração e testes de aceitação. A execução regular de testes garante que a qualidade do sistema seja mantida e que o desenvolvimento não introduza erros inesperados.

Outro papel importante da equipe de qualidade é monitorar os resultados dos testes, documentando falhas e garantindo que os problemas identificados sejam tratados de forma rápida e eficaz, antes que afetem a produção. Além disso, compete à equipe analisar a qualidade do código-fonte da aplicação por meio de ferramentas especializadas, como o SonarQube, com o objetivo de identificar vulnerabilidades, falhas, débitos técnicos e oportunidades de melhoria.

Por fim, a equipe de qualidade deve validar e monitorar o desempenho do sistema, executando testes de carga e de estresse para garantir que a aplicação consiga lidar com altos volumes de usuários e dados. Esses testes são fundamentais para identificar gargalos e assegurar que a aplicação funcione de maneira eficiente em todas as condições.

6.2.1. Principais responsabilidades da equipe de qualidade:

6.2.1.1. Testes Automatizados: Criar e manter testes automatizados, testes unitários, de integração e de aceitação. Garantir a execução contínua dos testes durante o ciclo de vida do código, por meio do pipeline de CI/CD.

6.2.1.2. Estratégia de Testes: Definir estratégias de teste, garantindo uma cobertura adequada do código. Identificar e preencher lacunas nos testes, colaborando com a equipe de desenvolvimento para melhorar a qualidade do código. Documentar e monitorar os resultados dos testes, garantindo que as falhas sejam tratadas antes de afetarem a produção. Colaborar com a equipe de desenvolvimento para corrigir erros de forma rápida e eficiente.

6.2.1.3. Performance do Sistema: Executar testes de carga e de estresse para validar a capacidade do sistema de lidar com grandes volumes de dados e usuários. Identificar gargalos de performance e sugerir melhorias.

6.2.1.4. Test-Driven Development (TDD): Colaborar com a equipe de desenvolvimento para implementar e promover a abordagem de TDD, garantindo maior qualidade no código desde o início.

6.3. Equipe de operações

A equipe de operações desempenha um papel crítico na implementação, monitoramento e manutenção das infraestruturas e serviços necessários para suportar a aplicação em produção. Dentro da metodologia DevOps, a equipe de operações não se limita apenas a garantir a disponibilidade do sistema, mas também trabalha ativamente na automação dos processos e na otimização da infraestrutura para suportar o crescimento e a escalabilidade.

A primeira responsabilidade da equipe de operações é garantir que a infraestrutura esteja corretamente provisionada e configurada. Isso inclui a implementação de infraestrutura como código (IaC), utilizando ferramentas como Terraform ou Ansible, para automatizar o processo de criação e configuração de servidores, redes e outros componentes de infraestrutura.

A equipe de operações deve também se assegurar de que os sistemas de monitoramento e alerta estejam devidamente configurados para capturar métricas de performance, como tempo de resposta e uso de recursos, além de logs de eventos críticos. Isso permite a detecção precoce de problemas e facilita a resolução de incidentes de forma proativa.

Por fim, a equipe de operações é responsável por garantir que a segurança da infraestrutura e dos serviços esteja em conformidade com as melhores práticas e políticas estabelecidas.

6.3.1. Principais responsabilidades da equipe de operações:

6.3.1.1. Infraestrutura como Código (IaC): Provisionar e configurar a infraestrutura utilizando ferramentas de IaC (Terraform, Ansible, etc.). Automatizar a criação e manutenção dos ambientes.

6.3.1.2. Monitoramento e alerta: Configurar sistemas de monitoramento para capturar métricas e logs críticos do sistema. Estabelecer alertas para detectar problemas de performance e incidentes rapidamente. Implementar e manter soluções de orquestração de contâineres, como Kubernetes.

6.3.1.3. Gestão de configuração: Garantir a consistência de configurações entre os ambientes de desenvolvimento, homologação e produção. Colaborar com as equipes de desenvolvimento e qualidade para realizar o gerenciamento eficiente das configurações.

6.3.1.4. Segurança da infraestrutura: Garantir que as práticas de segurança estejam em conformidade com as políticas e melhores práticas estabelecidas. Monitorar e aplicar atualizações de segurança na infraestrutura.

7. METODOLOGIA DEVOPS NO IBAMA

Serão apresentadas as ferramentas essenciais, o processo desenvolvido e a arquitetura proposta para a implementação da metodologia DevOps no Ibama, visando a modernização e a eficiência dos fluxos de trabalho.

7.1.1. Ferramentas utilizadas

A seguir, serão apresentadas as principais ferramentas para a implementação e adoção eficaz da metodologia DevOps. Essas ferramentas foram selecionadas com base em suas funcionalidades e na capacidade de integrar e automatizar processos, promovendo colaboração, eficiência e entrega contínua. Além disso, tratam-se de soluções já utilizadas pelo Ibama, podendo ser alteradas ou atualizadas ao longo da evolução do processo de DevOps.

7.1.1. Versionamento de código (GitLab - v17.7.4)

O GitLab já é usado como ferramenta padrão do Ibama para versionamento de código e colaboração de equipes.Dessa forma, a plataforma também será adotada como componente central da metodologia DevOps.

7.1.2. CI/CD (GitLab - v17.7.4)

O GitLab foi adotado como ferramenta de CI/CD, integrando todos os processos em uma única plataforma. Essa abordagem facilita a visualização, o gerenciamento e a execução das pipelines pelas equipes de desenvolvimento e operações.

7.1.3. Container Registry (GitLab - v17.7.4)

O GitLab também foi adotado para o armazenamento de imagens Docker, permitindo que os repositórios de imagens sejam vinculados diretamente aos projetos. Essa integração facilita a visualização e o uso das imagens criadas pelos desenvolvedores, centralizando e simplificando o gerenciamento.

7.1.4. Qualidade de software (SonarQube - v10.7)

O SonarQube é uma ferramenta utilizada pelo Ibama para análise da qualidade do código. Sua adoção no processo de DevOps visa garantir padrões de código mais robustos e confiáveis, contribuindo para a eficiência e segurança do desenvolvimento.

7.1.5. Scanner de vulnerabilidade (Trivy)

O Trivy é uma ferramenta de verificação de vulnerabilidades em imagens Docker. Sua execução ocorre de forma automatizada no âmbito das pipelines de CI/DD, garantindo a análises contínuas de segurança ao longo do processo de desenvolvimento e entrega.

7.1.6. Cluster Kubernetes

O orquestrador de containers Kubernetes, responsável pelo deploy das aplicações geradas por meio das pipelines. Sua utilização contribui para maior disponibilidade, resiliência e eficiência operacional dos ambientes.

7.2. Diagrama do ambiente DevOps do Ibama

O diagrama apresenta a arquitetura DevOps adotada no Ibama, definindo o ciclo de vida da aplicação e estruturando o pipeline de Integração Contínua (CI) e Entrega Contínua (CD). Esse fluxo automatiza todo o processo, desde o desenvolvimento até a implantação em produção, garantindo eficiência, qualidade e segurança em cada etapa.

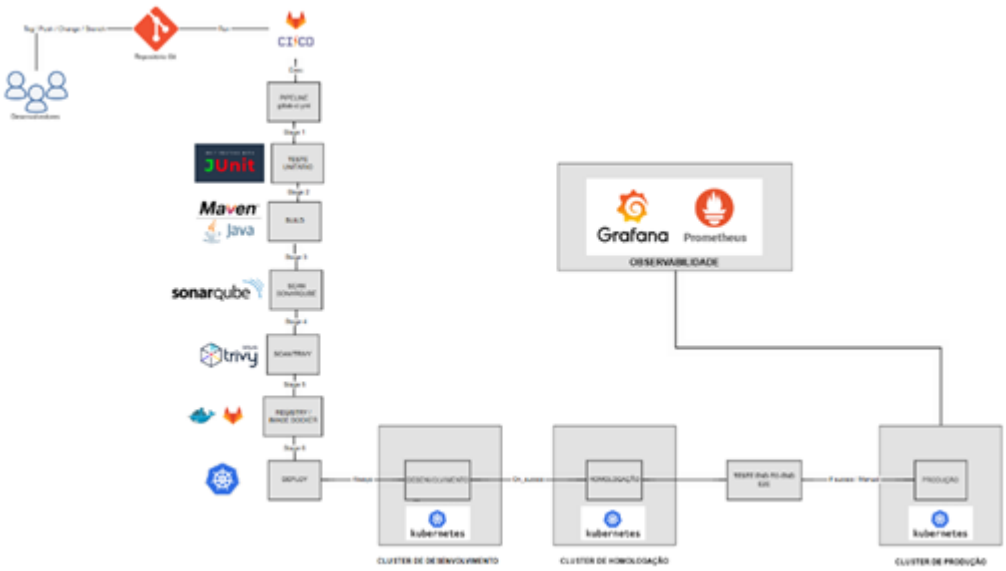

Figura 2: Diagrama DevOps Ibama

7.3. Arquitetura Pipeline

Os(As) desenvolvedores(as) realizam alterações no código e gerenciam branches no repositório Git. A cada push ou merge em uma branch específica, um trigger é ativado, acionando automaticamente o GitLab CI/CD, que inicia a execução do pipeline definido no arquivo gitlab-ci.yml. A partir dessa etapa, são executados todos os estágios de teste e entrega da aplicação.

7.4. Pipeline de CI/CD

A pipeline é estruturada em múltiplos estágios, cada um com uma função específica:

- Teste unitário: Os testes unitários são executados para validar funcionalidades individuais do código. Essa etapa garante que cada componente funcione conforme esperado antes de avançar no pipeline.
- Build: A aplicação é compilada utilizando o Maven para aplicações Java, com a geração dos artefatos executáveis (/target). Nessa etapa, também é realizada a build da imagem docker (front-end e back-end).
- Scan qualidade (SonarQube - v10.7): O código é analisado pelo SonarQube para identificar vulnerabilidades, code smells e problemas de qualidade.
- Scan de vulnerabilidades (Trivy): A imagem docker gerada é analisada com Trivy, buscando vulnerabilidades críticas ou não conhecidas e retorna artefatos desta análise.
- Criação eregistro da imagem Docker: A aplicação é empacotada em um container Docker e armazenada em um repositório (registry)

de imagens que será consumida posteriormente pelo estágio de deploy.

- Testes End-to-End (E2E): São executados testes automatizados com Protractor para validar a aplicação de ponta a ponta.
- Deploy em Kubernetes: A aplicação é implantada no cluster Kubernetes, utilizando a imagem Docker gerada na build.
- Ambientes e fluxo de entrega
- Cluster de desenvolvimento: A aplicação é implantada automaticamente após cada build para testes iniciais.
- Cluster de homologação: Após sucesso na etapa anterior, a aplicação segue para um ambiente de homologação.
- Cluster de produção: Se os testes forem bem-sucedidos, juntamente com o teste E2E, o deploy na produção pode ocorrer automaticamente ou manualmente.

7.5. Pipeline PLSQL

A Pipeline PLSQL foi desenvolvida para automatizar o versionamento, validação e implantação de objetos Oracle (PL/SQL) nos ambientes de banco de dados do Ibama, garantindo padronização, segurança e rastreabilidade no processo de entrega.

Essa pipeline é voltada exclusivamente à gestão de objetos de banco de dados, como procedures, functions, packages, triggers, views e scripts DML, utilizando o GitLab CI/CD como ferramenta de orquestração. A execução é acionada automaticamente a partir de commits ou merges em branches definidas, assegurando controle e consistência das alterações.

O fluxo é estruturado em estágios sequenciais:

- Validação: Verifica a integridade e a estrutura dos arquivos PL/SQL modificados, assegurando que contenham comandos válidos antes do deploy.
- Testes: Executa testes automatizados, quando aplicável, gerando relatórios padronizados para rastreabilidade.
- Deploy incremental: Implanta apenas os objetos alterados entre commits, respeitando a ordem de dependência entre os tipos de objetos e reduzindo impactos nos ambientes.
- Verificação pós-deploy: Confirma a compilação correta dos objetos e interrompe a pipeline caso sejam identificados objetos inválidos.

A execução dos scripts ocorre por meio do SQL*Plus, com controle de schema, captura detalhada de logs e, quando aplicável, realização de backup automático antes da implantação. Todos os eventos são registrados em logs estruturados, disponibilizados como artifacts da pipeline, permitindo auditoria e análise posterior.

A Pipeline PLSQL segue as diretrizes de segurança do Ibama, utilizando variáveis de ambiente para o gerenciamento de credenciais e garantindo segregação entre os ambientes de desenvolvimento, homologação e produção.

7.6. Observabilidade e monitoramento

O Grafana e o Prometheus são utilizados para observabilidade e monitoramento contínuo da aplicação. O Prometheus atua como coletor de métricas, armazenando séries temporais e fornecendo alertas baseados em regras predefinidas.

O Grafana é responsável pela visualização dessas métricas por meio de dashboards dinâmicos, permitindo a análise de desempenho, detecção de anomalias e correlação de eventos. Juntos, esses componentes garantem a estabilidade dos ambientes ao identificar proativamente falhas, gargalos e tendências de uso.

8. PERMISSÕES DE ACESSO

8.1. Gitlab

Desenvolvedor:

No contexto do projeto DevOps, um usuário com o papel de desenvolvedor no GitLab terá as seguintes permissões:

- Criar, editar e excluir branches (exceto branches protegidas).
- Criar e gerenciar Merge Requests.
- Executar pipelines manualmente.
- Acessar e visualizar logs de CI/CD.
- Criar, editar e excluir issues e milestones.
- Criar e editar tags (exceto tags protegidas).
- Criar e editar imagens docker (Registry).
- Acessar e clonar repositórios do projeto.
- Comentar e revisar código em Merge Requests.
- Criar e editar Wiki e Snippets.

Essas permissões garantem que o desenvolvedor possa contribuir ativamente com o código, acompanhar o processo de CI/CD e colaborar no fluxo de desenvolvimento do projeto.

8.2. SonarQube

Desenvolvedor:

- No SonarQube, um usuário com o papel de Desenvolvedor terá as seguintes permissões dentro do projeto:
- Visualizar relatórios de análise de código.

- Acessar e interpretar métricas de qualidade, como cobertura de testes, vulnerabilidades e código duplicado.
- Comentar e revisar questões levantadas pelo SonarQube.
- Marcar problemas como resolvidos ou ignorados (caso permitido pela política do projeto).
- Executar análises de código manualmente (se autorizado).
- Exportar relatórios e métricas de qualidade.

Essas permissões permitem que o desenvolvedor acompanhe a qualidade do código, identifique problemas e colabore na melhoria contínua do software.

8.3. Kubernetes

O ambiente Kubernetes é gerenciado principalmente pela equipe de operações, que é responsável pelo suporte, manutenção, observabilidade e monitoramento. Também, garantem a organização dos projetos por meio da estruturação de namespaces, além de gerenciar a criação de rotas e a configuração de balanceadores de carga. Assim como, garantir a segurança de todo o cluster Kubernetes.

9. PIPELINE

Para criar uma pipeline no GitLab, é necessário criar um arquivo chamado .gitlab-ci.yml na raiz do repositório da branch correspondente. A pipeline deve ser configurada conforme os modelos apresentados a seguir que também estão disponíveis no repositório de exemplo (https://git.ibama.gov.br/sistemas-ibama/template-devops).

9.1. Template

O modelo tem as seguintes definições:

- declaração de variáveis
- definição de estágios
- definição de cache
- definição de triggers
- testes unitários
- compilação de código
- geração de imagem docker
- verificação de qualidade pelo Sonar
- verificação de vulnerabilidade pelo Trivy
- envio de imagem docker gerada para o registry
- deploy da aplicação

O código se encontra no repositório de exemplo do git (https://git.ibama.gov.br/sistemas-ibama/template-devops/-/blob/main/.gitlab-ci.yml?ref_type=heads).

Observação: Cada pipeline possui suas particularidades, especialmente no estágio de build, que varia conforme as tecnologias e arquiteturas utilizadas.

9.2. Deploy em ambiente Kubernetes

Para realizar o deploy no atual ambiente Kubernetes, é necessário criar um *Deployment*, um *Route* e um *Service*. A seguir, serão apresentados modelos para cada um desses recursos.

9.2.1. Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: sistema-backend

spec:

replicas: 1

selector:

matchLabels:

app: sistema-backend

template:

metadata:

labels:

app: sistema-backend

spec:

containers:

- name: sistema-backend

image: $TAG_IMAGE

ports:

- containerPort: porta

imagePullPolicy: Always

imagePullSecrets:

- name: gitlab-registry

9.2.2. Route

apiVersion: v1

kind: Route

metadata:

labels:

app.kubernetes.io/instance: sistema-backend-hom

name: sistema-backend

spec:

port:

targetPort: sistema-backend

to:

kind: Service

name: sistema-backend

tls:

termination: edge

insecureEdgeTerminationPolicy: Redirect

9.2.3. Service

apiVersion: v1

kind: Service

metadata:

labels:

app.kubernetes.io/instance: sistema-backend-hom

name: sistema-backend

spec:

selector:

app: sistema-backend

ports:

- name: sistema-backend

port: porta

targetPort: porta

9.3. Sistemas não conteinerizado

Para sistemas que não utilizam tecnologia de containers, a pipeline mantém uma estrutura semelhante, porém sem as etapas de build e push de imagens Docker e a verificação de vulnerabilidades. Permanecem os estágios de teste, build do frontend e backend, análise de qualidade de código e deploy. Abaixo está o diagrama do fluxo.

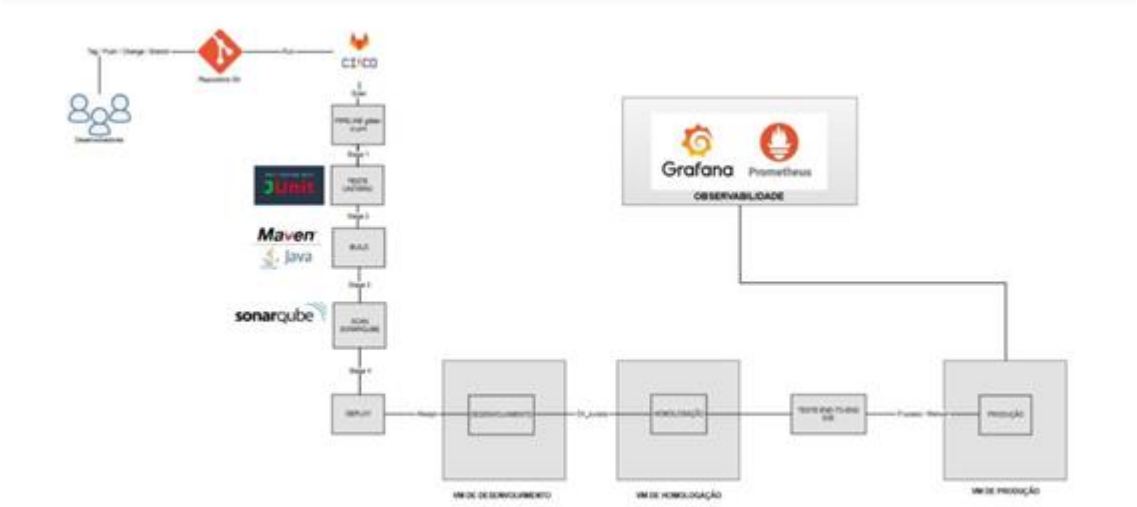

Figura 3: Diagrama sistema não conteinerizado

Para a criação da pipeline em sistemas não conteinerizados, seguir a definição disponível no arquivo template-vm.gitlab-ci.yml do repositório mencionado anteriormente. Contém as configurações necessárias para a implementação da pipeline em ambientes sem containers.

```
variables:
## Declarar variáveis
## variável: valor
stages:
## Definir estágios
## - Testes
## - build
## - sonar
## - deploy
cache:
paths:
## Diretórios necessários para se salvar em cache
## TRIGGERS MODELO
.template_regras_build_back_end:
only:
refs:
- pushes
changes:
```

```
## arquivos monitorados para detectar mudanças
## Exemplos
## - .gitlab-ci.yml
## - pom.xml
## - diretório/**/*
.template_regras_build_front_end:
only:
refs:
- pushes
changes:
## arquivos monitorados para detectar mudanças
## Exemplos
## - .gitlab-ci.yml
## - pom.xml
## - diretório/**/*
## TESTE UNITÁRIO
teste-frontend:
stage: Testes
image: imagem de teste
script:
- comando para executar os testes
extends: .template_regras_build_front_end
teste-backend:
stage: Testes
image: imagem de teste
script:
- comando para executar os testes
extends: .template_regras_build_back_end
## COMPILANDO FRONTEND
build-frontend:
stage: build
```

```
image: imagem para compilar o frontend
script:
- comando para compilar o frontend
extends: .template_regras_build_front_end
## COMPILANDO BACKEND
build-backend:
stage: build
image: imagem para compilar o backend
script:
- comando para compilar o backend
extends: .template_regras_build_back_end
## SCANNER QUALIDADE FRONTEND
scan-sonar-frontend:
stage: sonar
image: imagem sonar
allow_failure: true
script:
- comando para executar o sonar
## Exemplo
## - echo "Rodando SonarQube Scanner..."
## - sonar-scanner
## -Dsonar.projectKey="Projeto-Frontend"
## -Dsonar.projectName="$SYSTEM-Frontend"
## -Dsonar.sources=.
## -Dsonar.host.url="$SONAR_HOST_URL"
## -Dsonar.token="$SONAR_TOKEN_FRONTEND"
## -Dsonar.exclusions=**/node_modules/**,**/*.test.js,**/*.spec.js
needs: ["build-frontend"]
extends: .template_regras_build_front_end
## SCANNER QUALIDADE BACKEND
scan-sonar-backend:
```

```
stage: sonar
image: imagem sonar
script:
- comando para executar o sonar
## Exemplo
## - echo "Executando SonarQube Scanner para o backend..."
## - sonar-scanner
## -Dsonar.projectKey="Projeto-Backend"
## -Dsonar.projectName="$SYSTEM-Backend"
## -Dsonar.sources=.
## -Dsonar.host.url="$SONAR_HOST_URL"
## -Dsonar.token="$SONAR_TOKEN_BACKEND"
## -Dsonar.java.binaries=target
allow_failure: true
needs: ["build-backend"]
extends: .template_regras_build_back_end
## DEPLOY
deploy-frontend:
stage: deploy
image: imagem
script:
- comando para deploy
needs: ["build-frontend"]
extends: .template_regras_build_front_end
deploy-backend:
stage: deploy
image: imagem
script:
- comando para deploy
needs: ["build-backend"]
extends: .template_regras_build_back_end
```

10. QUALIDADE - SONARQUBE

10.1. Critérios de qualidade

O critério mínimo de qualidade de software para aprovação no SonarQube exige:

- cobertura de testes superior a 80%, garantindo que a maior parte do código seja validada por testes automatizados;
- ausência de vulnerabilidades classificadas como críticas ou altas, de modo a reduzir riscos de segurança no software;
- duplicação de código abaixo de 5%, promovendo reutilização e manutenção do código.

11. TRATAMENTO DE DADOS

Esclarecemos que as pipelines descritas no modelo DevOps proposto não realizam tratamento de dados institucionais ou pessoais, uma vez que seu escopo se restringe exclusivamente ao deploy de código e à aplicação de configurações nos ambientes.

No que se refere ao controle de acesso já estão contemplados no modelo:

- Logs de pipeline: os registros de execução das pipelines são automaticamente armazenados e retidos pela plataforma, estando disponíveis para consulta e auditoria;
- Versionamento de código: todos os artefatos de código e configuração são versionados em repositório Git, garantindo rastreabilidade completa do histórico de mudanças;
- Controle de acesso aos repositórios: o acesso aos repositórios de código é gerenciado com perfis e permissões definidos, seguindo o princípio do menor privilégio;
- Controle de acesso a variáveis: as variáveis de pipeline, incluindo segredos e credenciais, possuem controle de acesso segregado por ambiente, restringindo a visualização e o uso conforme o perfil do usuário.

Dessa forma, entende-se que os dados institucionais, LGPD e ciclo de vida de dados, embora pertinentes ao contexto organizacional mais amplo, não se aplicam diretamente ao escopo das pipelines de CI/CD descritas neste modelo, uma vez que estas não ingressam, processam ou expõem dados dos usuários.

12. CONCLUSÃO

A implementação da metodologia e da arquitetura DevOps no Ibama permite uma transformação significativa na forma como aplicações são desenvolvidas, testadas, entregues e monitoradas. Ao adotar práticas modernas de Integração Contínua (CI) e Entrega Contínua (CD), o Ibama estabelece um fluxo de desenvolvimento mais ágil, seguro, padronizado e amplamente automatizado.

O uso do GitLab CI/CD como ferramenta central da pipeline proporciona maior confiabilidade ao processo de desenvolvimento, garantindo que cada alteração no código seja automaticamente validada e implantada de maneira eficiente. Além disso, a incorporação de ferramentas para análise de qualidades e vulnerabilidades fortalece a robustez, eficiência e segurança do ambiente, permitindo a detecção e mitigação de riscos antes da implantação em produção.

A observabilidade foi aprimorada com a integração do Grafana e Prometheus, proporcionando monitoramento contínuo, métricas detalhadas e alertas em tempo real. Isso garante maior previsibilidade e controle sobre o desempenho das aplicações e infraestruturas. Com essa abordagem, conseguimos não apenas otimizar o tempo de entrega e a qualidade das releases, mas também reduzir falhas e aumentar a confiabilidade dos sistemas gerenciados.

A adoção do DevOps reforça o compromisso do Ibama com inovação, eficiência e segurança na gestão de suas aplicações, garantindo maior escalabilidade e sustentação para as necessidades institucionais presentes e futuras.

JAIR SCHMITT

ORDEM DE SERVIÇO Nº 12/2026-GABIN, DE 25 DE AGOSTO DE 2026

O PRESIDENTE DO INSTITUTO BRASILEIRO DO MEIO AMBIENTE E DOS RECURSOS NATURAIS RENOVÁVEIS (IBAMA), no uso das atribuições que lhe confere o art. 15 do Anexo I do Decreto nº 12.130, de 7 de agosto de 2024, que aprovou a Estrutura Regimental do Ibama, e o Regimento Interno, aprovado pela Portaria nº 73, de 26 de maio de 2025, publicada no Diário Oficial da União de 27 de maio de 2025, e tendo em vista o que consta no Processo nº 02001.015513/2019-59 e o disposto na Portaria Normativa nº 1, de 14 de maio de 2015, resolve:

Art. 1º Fica alterada a composição dos(as) representantes do Grupo Temático de Emergências Ambientais de Resgate e Reabilitação de Fauna Oleada (GTE-Fauna Oleada), com as seguintes finalidades:

I - revisar o Plano Nacional de Ação de Emergência para Fauna Impactada por Óleo (PAE-Fauna), incluindo o Manual de Boas Práticas de Manejo de Fauna Atingida por Óleo;

II - discutir e revisar a ampliação do escopo de atuação do GTE-Fauna Oleada para contemplar a temática de fauna em um contexto mais amplo relacionado às emergências ambientais e mudanças climáticas;

III - contribuir para a elaboração de protocolos, procedimentos operacionais padrão relacionados à temática de fauna e emergências ambientais.