PUC Minas

Trabalho Prático
Plano de Gerenciamento de Configuração

Versão 1.0

Histórico da Revisão

Data	Versão	Descrição	Autor
07/12/2025	1.0	Emissão inicial do Plano de Gerenciamento de Configuração	Maisa Pires, Miguel Vieira, Gabriel Faria, Joao Victor Salim

Índice

Plano de Gerenciamento de Configuração

1. Introdução

Este Plano de Gerenciamento de Configuração (PGC) descreve as políticas, procedimentos e responsabilidades para o controle de versão, identificação, armazenamento, liberação e auditoria dos itens de configuração do projeto **"Pipeline CI/CD com CircleCI para aplicação Node.js"**, desenvolvido como Trabalho Prático 1 da disciplina de Gerência de Configuração e Evolução de Software.

O PGC assegura que todos os artefatos relevantes (código, scripts de build, artefatos de deploy, documentação e testes) sejam gerenciados de forma controlada, reprodutível e rastreável, permitindo replicação do ambiente e recuperação em caso de falhas.

1.1 Objetivo

Definir e padronizar as práticas de Gestão de Configuração para o projeto, com foco em:

- identificação clara dos itens de configuração;
- controle de alterações;
- versionamento e baselines;
- armazenamento e release;
- geração de relatórios e auditorias;
- responsabilização das atividades de CM.

1.2 Escopo

Este PGC aplica-se a todos os artefatos produzidos ou utilizados pelo projeto TP1:

- Código-fonte do projeto (diretório src/);
- Testes automatizados (tests/);
- Arquivos de integração contínua (.circleci/config.yml);
- Dockerfile e scripts de empacotamento;
- Documentação do projeto (documento do PGC, slides, README);
- Artefatos gerados (imagens Docker, releases).

O plano abrange atividades desde o desenvolvimento inicial até a entrega final (push para Docker Hub e criação do pacote de entrega .zip).

1.3 Definições, Acrônimos e Abreviações

- CM / GC: Gestão de Configuração (Configuration Management).
- CI: Integração Contínua (Continuous Integration).
- CD: Entrega/Implantação Contínua (Continuous Delivery/Deployment).
- CCB: Change Control Board (Conselho de Controle de Mudanças).
- PGC: Plano de Gerenciamento de Configuração.
- SHA: Commit SHA (identificador único do commit Git).
- Docker Hub: Registro público para imagens Docker.
- CircleCI: Serviço de CI/CD usado no projeto.
- Repo: Repositório Git hospedado no GitHub.

1.4 Referências

- Enunciado do Trabalho Prático 1 — Gerência de Configuração e Evolução de Software (arquivo

fornecido).
- Documentação CircleCI — Pipelines e configuração.
- Documentação Docker — Build e push em registries.
- Git Documentation — Versionamento e fluxo de branches.

1.5 Visão Geral

O documento descreve a estrutura organizacional para GC, as ferramentas, o programa de identificação e baselines, o processo formal de controle de mudanças, a forma de contabilizar o status das configurações, os marcos e os planos de treinamento/recursos necessários para execução do PGC.

2. Gerenciamento de Configuração de Software

2.1 Organização, Responsabilidades e Interfaces

Equipe do Projeto:

- **Gerente de Configuração / Pipeline (Responsável principal):** Maisa Pires
    - Responsável por coordenar o versionamento, criar baselines, executar ou validar releases e manter o PGC atualizado.
- **Desenvolvedores / Membros de Configuração:**
    - Miguel Vieira — Implementação e testes.
    - João Victor Salim — Integração e documentação.
    - Gabriel Faria — Scripts de deploy e Docker.

Responsabilidades resumidas:

- **Todos os integrantes:** fazem commits com mensagens claras; seguem as políticas de branching e testes antes do push.
- **Responsável pelo repo (owner no GitHub):** controla permissões, revê PRs e aprova merges para main.
- **Responsável de release (podem alternar):** cria o tag de release, efetua o build de imagem Docker e atualiza o registro.
- **Professor / Avaliador:** receberá a entrega final, que consiste no ZIP do repositório, PGC (PDF), script de implantação (config.yml + Dockerfile) e slides.

Interfaces:

- Integração com GitHub (repositório) e Docker Hub (registro de imagens).
- CircleCI consome o repositório e executa pipelines automáticos.

2.2 Ferramentas, Ambiente e Infra-estrutura

Ferramentas principais:

- **Git + GitHub:** controle de versão remoto do código-fonte.
- **Node.js / npm:** ambiente de execução e gerenciamento de dependências.
- **Jest (ou testes JS):** framework de testes (ou uso de scripts simples para as etapas de teste conforme modelo proposto).
- **CircleCI:** execução do pipeline CI/CD (arquivo .circleci/config.yml).
- **Docker / Dockerfile:** empacotamento da aplicação em container.

- **Docker Hub:** repositório de imagens para deploy.
- **Editor/IDE:** VSCode ou similar (não obrigatório para a entrega).

Ambiente e infraestrutura:

- Repositório centralizado no GitHub (namespace do time);
- Runners/cloud da CircleCI para execução dos jobs;
- Remote Docker (setup_remote_docker) configurado no job de deploy para build/push da imagem;
- Variáveis de ambiente (tokens) configuradas no CircleCI: DOCKER_USER, DOCKER_PASS (token), DOCKER_REPO(opcional); armazenadas como segredos no projeto CircleCI.

Considerações de capacidade e localização:

- Projeto de escopo acadêmico: dados pequenos (< 100 MB) e repositório público/privado conforme necessidade.
- Backups são providos implicitamente pelo GitHub (histórico) e pelo armazenamento local dos integrantes.

3. O Programa de Gerenciamento de Configuração

3.1 Identificação da Configuração

Todos os itens de configuração (ICs) do projeto recebem identificação e classificação para garantir rastreabilidade.

Itens de Configuração (ICs) incluídos:

- Códigos-fonte (src/): arquivos .js.
- Testes (tests/).
- Arquivo de pipeline: .circleci/config.yml.
- Dockerfile.
- Documentação (PGC, slides, README).
- Pacote de entrega (.zip) e imagens Docker.

3.1.1 Métodos de Identificação

Naming convention (convenção de nomes):

- Arquivos de código: src/<módulo>/<nome>.js
- Testes: tests/<tipo>/<nome>.test.js
- Pipeline: .circleci/config.yml
- Docker image tag: <DOCKER_USER>/<REPO>:<COMMIT_SHA> (ex.: my4wyy/meuapp:ef0518e36e63)
- Releases (git tag): v<MAJOR>.<MINOR>.<PATCH> (ex.: v1.0.0)

Commit messages: devem seguir formato simples:

- feat: <descrição>
- fix: <descrição>
- ci: <descrição do ajuste no pipeline>
- Exemplo: feat: adicionar endpoint /todos

Identificação única: cada commit gera um SHA (hash) que é utilizado como identificador da build e tag

da imagem Docker.

3.1.2 Linhas de Base do Projeto

As linhas de base (baselines) serão estabelecidas nas seguintes ocasiões:

- **Baseline Inicial (Baseline 0.1)** — após entrega da primeira versão estável do protótipo (primeiro envio com todas as rotas básicas e testes mínimos).
- **Baseline de Avaliação (Baseline 0.9)** — antes da entrega final do TP (após inclusão dos testes, documentação e pipeline funcional).
- **Baseline de Release (v1.0.0)** — versão final entregue ao professor (ZIP + Docker image tag).

Cada baseline será estabelecida criando-se um **tag Git** correspondente e registrando a versão e commit associado no histórico do PGC.

3.2 Configuração e Controle de Alterações

3.2.1 Processamento e Aprovação de Controles de Mudanças

- **Proposta de mudança (Issue/PR):** qualquer alteração significativa deve ser submetida via *Pull Request* (PR) no GitHub, vinculada a um issue (quando aplicável).
- **Revisão de código:** pelo menos 1 revisor do grupo analisa a proposta (código e testes).
- **Execução dos testes:** todos os testes devem passar no ambiente de CI (CircleCI) antes da aprovação.
- **Aprovação:** o revisor aprova o PR.
- **Merge para main:** somente após aprovação e todos os checks passarem (build, unit, integration, acceptance).
- **Registro:** a alteração aprovada e o merge originam um commit na branch main, que por sua vez aciona o pipeline e, caso tudo passe, cria uma nova imagem/tag.
- **Atualização do PGC (quando necessário):** alterações que impactem a estratégia de config (ex.: mudança de repositório, ferramenta, política de tags) devem ser registradas no PGC e no histórico.

Critérios para alterações emergenciais:

Corrigendas críticas podem ser aplicadas diretamente em main somente quando justificadas (ex.: correção de segurança) e devem ser registradas posteriormente no histórico com explicação.

3.2.2 CCB (Conselho de Controle de Mudanças)

Para o escopo deste trabalho acadêmico, o CCB é informal e composto pelos membros do grupo:

- **Membros do CCB:** Maisa Pires (coordenadora), Miguel Vieira, João Victor Salim e Gabriel Faria.
- **Função:** avaliar mudanças que impactem arquitetura, segurança, ou release.
- **Reuniões:** as decisões são documentadas como comentários no PR / issue e registradas no histórico do PGC.
- **Aprovação mínima:** consenso de 2/4 membros é suficiente para mudanças de média complexidade; unanimidade é recomendada para mudanças de alto impacto (ex.: substituição de Docker Hub por outro registry).

3.3 Contabilidade do Status de Configuração

3.3.1 Armazenamento de Mídia do Projeto e Processo de Release

- **Armazenamento principal:** GitHub (repositório remoto) armazena o histórico completo do código e documentação.
- **Artefatos binários:** imagens Docker armazenadas no Docker Hub (namespace do time).
- **Política de retenção:** para fins acadêmicos, manter todos os commits e tags no GitHub e manter imagens Docker por, no mínimo, 30 dias após a entrega; versões finais (tags de release) devem ser mantidas indefinidamente (ou até novo instruction).
- **Backup e recuperação:** o próprio GitHub e Docker Hub gerenciam disponibilidade; os integrantes mantêm cópias locais e o ZIP do repositório serve como cópia de entrega.

Processo de release:

- Tag de release no Git (git tag vX.Y.Z + git push --tags).
- Pipeline do CircleCI detecta push em main (ou push de tag) e executa jobs: build → testes → acceptance → build da imagem → push para Docker Hub.
- Documentar no README do release o que está incluído, instruções de execução e problemas conhecidos.

3.3.2 Relatórios e Auditorias

Os relatórios são utilizados para avaliar a “qualidade do produto” em qualquer momento determinado no projeto ou ciclo de vida do produto. O relatório sobre defeitos baseado em controles de mudanças pode fornecer alguns indicadores de qualidade úteis e, assim, alertar o gerenciamento e os desenvolvedores quanto a áreas de desenvolvimento particularmente críticas. Os defeitos são classificados, freqüentemente, por seu caráter crítico (alto, médio e baixo) e poderiam ser relatados na seguinte base:

- Tempo (Relatórios baseados no Tempo): Há quanto tempo os defeitos de diversos tipos estão em aberto? O que é o “tempo de retardo’’ entre quando os defeitos são localizados no ciclo de vida e quando são corrigidos?
- Distribuição (Relatórios Baseados em Contagem): Quantos defeitos existem nas várias categorias por proprietário, prioridade ou estado de correção?
- Tendência (Relatórios relacionados ao Tempo e à Contagem): Qual é o número cumulativo de defeitos localizados e corrigidos no decorrer do tempo? Qual é a classificação dos defeitos detectados e corrigidos? O que é o “intervalo de qualidade” em termos de defeitos abertos em oposição a fechados? O que é o tempo médio de resolução de defeitos?]

Relatórios a serem gerados: logs do CircleCI (cada execução), histórico de commits e changelogs, relatório de cobertura de testes (se implementado), listagem de imagens liberadas.

Formato: relatórios básicos em texto/HTML (logs do CI) e changelog em Markdown (CHANGELOG.md).

Auditorias: revisão do histórico de commits, tags e imagens para verificar conformidade com o PGC. Auditorias internas podem ser realizadas por um membro distinto do autor do código para garantir independência.

4. Marcos

Marco	Data (prevista)	Descrição
Início do Projeto	01/Dez/2025	Kickoff e definição de escopo
Protótipo Inicial	03/Dez/2025	API básica e /health funcionando
Ampliação / Implementação	05/Dez/2025	Inclusão dos módulos (controllers, services, etc.) e testes
Pipeline funcional (CI/CD)	06/Dez/2025	CircleCI configurado e pipeline verde
Entrega Final (ZIP + PDF)	07/Dez/2025	Geração do ZIP, PGC em PDF e slides
Apresentação em aula	data da apresentação (sorteio)	Apresentação de 10 minutos

5. Treinamento e Recursos

Treinamento necessário:

- Noções básicas de Git e fluxo de branches (tutorial inicial para todos os membros).
- Uso do CircleCI (configuração de projetos e variáveis de ambiente).
- Princípios básicos de Docker (build e push).
- Escrita de testes automatizados (conceitos e execução com npm/Jest).

Recursos (pessoas e ferramentas):

- Máquinas pessoais com Node.js + Docker instalados;
- Conta GitHub com repositório do grupo;
- Conta Docker Hub para push de imagens;
- Conta CircleCI conectada ao repositório;
- Documentos e tutoriais (links fornecidos nas referências).

Responsabilidades de treinamento:

- Maisa Pires: ministrar mini-workshop de 30 minutos sobre CircleCI e variáveis de ambiente.
- Gabriel Faria: demonstrar uso do Dockerfile e push para Docker Hub.
- Miguel Vieira & João Victor: guiar testes automatizados (execução local e interpretação de logs).

6. Controle de Software do Subfornecedor e do Fornecedor

Para este trabalho acadêmico não há subcontratação formal. Contudo, o projeto utiliza dependências externas (bibliotecas npm) e serviços de terceiros (GitHub, CircleCI, Docker Hub). O controle sobre componentes externos é tratado da seguinte forma:

- **Dependências externas (npm):** mantidas no package.json com versões travadas quando necessário; atualizações somente mediante PR e testes completos no pipeline.

- **Integração com serviços externos:** credenciais (tokens) armazenadas nas variáveis de ambiente do CircleCI; apenas integrantes autorizados possuem acesso à conta do Docker Hub e ao projeto no CircleCI/GitHub.
- **Avaliação de componentes COTS:** qualquer mudança em bibliotecas críticas deve ser avaliada pelo CCB e testada em ambiente de integração antes do merge em main.