UNIVERSIDADE DO NORTE DO PARANÁ
POLO LAGOA SANTA
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE

INFRAESTRUTURA ÁGIL

Camila Grazielle Veloso

Lagoa Santa - MG
2025

UNIVERSIDADE DO NORTE DO PARANÁ
POLO LAGOA SANTA
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE

INFRAESTRUTURA ÁGIL

Camila Grazielle Veloso

Relatório de Aula Prática apresentado ao prof. Frederico A F Pinto como requisito avaliativo da disciplina, referente ao 6º Semestre.

Lagoa Santa - MG
2025

Sumário

Resumo

Este relatório detalha a aula prática de Infraestrutura Ágil, que teve como objetivo simular e monitorar um processo de pipeline de entrega contínua (CI/CD) utilizando o sistema de controle de versões GIT e a ferramenta GitLab CI/CD. A metodologia consistiu na configuração de um projeto de Loja Virtual , no qual um arquivo de orquestração (.gitlab-ci.yml) foi utilizado para definir estágios de build e notificação, fazendo uso do Docker Hub para gerenciamento de imagens.

O monitoramento da execução resultou nas seguintes conclusões essenciais: (1) Uma Simulação de Falha foi registrada no estágio pre-build do pipeline, causada por credenciais ou nomes de repositório incorretos (unauthorized: incorrect username or password e denied: requested access to the resource is denied) durante a tentativa de docker push. (2) O Diagnóstico e Correção foram realizados ajustando as variáveis de autenticação e substituindo os placeholders (nome_do_docker_hub) pelo nome real do repositório no .gitlab-ci.yml. (3) A Simulação de Sucesso demonstrou que o pipeline, após a correção, foi executado com sucesso (Status: Passed), comprovando a construção automática da aplicação e o envio da imagem para o Docker Hub. Concluiu-se que o monitoramento em tempo real do pipeline é fundamental para o diagnóstico rápido de falhas e a manutenção da confiabilidade na entrega automática do software.

Palavras-chave: Integração Contínua (CI); Entrega Contínua (CD); DevOps; GIT; GitLab CI/CD; Docker; Docker Hub; YAML.

1 Introdução

A Integração Contínua (CI) e a Entrega Contínua (CD) formam o coração do movimento DevOps e são essenciais para a Infraestrutura Ágil. Este modelo visa integrar as alterações de código com frequência e verificar sua funcionalidade de forma automatizada. O presente relatório documenta a aula prática com o objetivo de simular e monitorar o processo de pipeline de entrega, utilizando ferramentas fundamentais para a Engenharia de Software.

O estudo se aprofunda na compreensão do script que possibilita a realização da chamada Integração Contínua, cujo fluxo é definido e orquestrado no GitLab CI/CD.

1.1 Conceituação das Ferramentas e Sua Importância

Para a execução e o sucesso do pipeline de entrega contínua, três ferramentas principais foram utilizadas:

1.1.1 **GIT**

– **Conceituação**: É uma ferramenta que implementa o pipeline de CI/CD, onde o processo é definido dentro de um arquivo denominado .gitlab-ci.yml. Este arquivo, que segue o formato YAML, define a ordem e as condições em que se dará a execução do pipeline.

– **Importância para Engenharia de Software**: O GitLab CI/CD é crucial porque automatiza as etapas de construção (build), testes e implantação. Ele assegura que o código é constantemente validado, reduzindo o tempo de feedback para minutos e garantindo que o build (o resultado final da aplicação) seja sempre confiável antes de ser entregue. É a espinha dorsal da automação e do monitoramento do processo de entrega.

1.1.2 **Docker Hub**

– **Conceituação**: É o serviço de registro de container. Neste projeto, ele é usado tanto para buscar imagens base para a execução dos jobs quanto para o registro da imagem final da aplicação construídaImportância para Infraestrutura Ágil: O Docker Hub (e a tecnologia de contêineres) promove a padronização de ambiente e a portabilidade. Ao construir e registrar a imagem no Docker Hub, a equipe garante que a aplicação funcionará exatamente da mesma forma em qualquer ambiente (desenvolvimento, teste ou produção), eliminando problemas de dependências e facilitando a Entrega Contínua. Sua utilização é vital para a arquitetura de microserviços e a modernização da entrega de software.

– **Importância para Infraestrutura Ágil**: O Docker Hub (e a tecnologia de contêineres) promove a padronização de ambiente e a portabilidade. Ao construir e registrar a imagem no Docker Hub, a equipe garante que a aplicação funcionará exatamente da mesma forma em qualquer ambiente (desenvolvimento, teste ou produção), eliminando problemas de dependências e facilitando a Entrega Contínua. Sua utilização é vital para a arquitetura de microserviços e a modernização da entrega de software..

1.1.3 **GitLab CI/CD**

– **Conceituação**: É uma ferramenta que implementa o pipeline de CI/CD, onde o processo é definido dentro de um arquivo denominado .gitlab-ci.yml. Este arquivo, que segue o formato YAML, define a ordem e as condições em que se dará a execução do pipeline.

– **Importância para Engenharia de Software**: O GitLab CI/CD é crucial porque automatiza as etapas de construção (build), testes e implantação. Ele assegura que o código é constantemente validado, reduzindo o tempo de feedback para minutos e garantindo que o build (o resultado final da aplicação) seja sempre confiável antes de ser entregue. É a espinha dorsal da automação e do monitoramento do processo de entrega.

1.2 Objetivos

A seção de objetivos detalha o propósito geral da aula prática e as metas específicas a serem alcançadas.

1.2.1 **Objetivo Geral**

O objetivo geral desta aula prática é simular o monitoramento de processo de pipeline de entrega, utilizando o GIT

1.2.2 **Objetivos Específicos**

Os objetivos específicos guiaram as duas etapas práticas do projeto simulado, conforme descrito a seguir:

– Entender como funciona o script para realização da chamada Integração Contínua.

– Realizar o monitoramento de processo de pipeline de entrega, observando o sucesso ou falha da execução do job.

– Compreender o funcionamento essencial do pipeline para a construção da aplicação de forma automática.

– Criar um relatório no final da atividade.

1.3 Estrutura do Trabalho

A estrutura do trabalho foi concebida para documentar de forma clara e metódica todos os procedimentos realizados e os resultados obtidos durante a simulação prática de Infraestrutura Ágil.

O relatório final está organizado nas seguintes seções:

– **Resumo**: Apresentação concisa do objetivo da aula, da metodologia utilizada (GIT, GitLab CI/CD, Docker Hub) e dos principais resultados alcançados (Simulação de Falha e Simulação de Sucesso).

– **Introdução**: Contextualização da importância da Integração Contínua (CI) e Entrega Contínua (CD) na Engenharia de Software, além da conceituação e justificativa do uso das ferramentas essenciais (GIT, GitLab CI/CD e Docker Hub).

– **Objetivos**: Definição clara do objetivo geral e dos objetivos específicos da aula prática.

– **Métodos**: Descrição da infraestrutura e dos procedimentos práticos adotados para a execução da atividade, incluindo o passo a passo da configuração do pipeline e os comandos de sincronização via GIT.

– **Dificuldades Encontradas**: Registro e análise dos problemas técnicos (ex: erros de permissão, histórico divergente, falhas de autenticação) que ocorreram durante a prática.

– **Resultados**: Documentação dos resultados da aula prática, incluindo o registro da Simulação de Falha (com análise do log) e a Simulação de Sucesso (comprovação do pipeline aprovado).

– **Conclusão**: Síntese final que relaciona os resultados obtidos com a compreensão do funcionamento essencial do pipeline para a construção automática da aplicação.

2 Metodologia

A metodologia adotada para a realização desta aula prática baseou-se na simulação de um ambiente de pipeline de entrega de software, com foco no monitoramento em tempo real do processo.

2.1 Infraestrutura e Setup

A prática seguiu as especificações de infraestrutura e materiais de consumo :

– **Instalações**: GIT.

– **Software**: GIT (Não Pago - Freeware). O GIT foi essencial como sistema de controle de versões distribuído.

– **Contas Adicionais**: Foram criadas contas obrigatórias nas plataformas GitLab (motor do CI/CD) e Docker Hub (para gerenciamento de containers).

– **Materiais**: Foi utilizado um computador por aluno.

O projeto base foi o arquivo "devops-master", contendo seis arquivos essenciais para o pipeline da Loja Virtual, incluindo o *.gitlab-ci.yml* (definição do pipeline), o *Dockerfile* (instrução de build da imagem) e os scripts Shell e Python .

2.2 Procedimentos Práticos (Realização da Atividade)

A atividade seguiu as etapas definidas para realizar o monitoramento de processo de pipeline de entrega , com foco na criação e execução de um script de Integração Contínua.

2.2.1 Definição e Estrutura do Pipeline

O pipeline foi definido no arquivo *.gitlab-ci.yml* , que utiliza o formato YAML (linguagem de marcação) para definir a ordem de execução. O pipeline é composto por um conjunto de jobs , que são os elementos mais básicos e contêm a cláusula script.

Os estágios definidos no *.gitlab-ci.yml* foram:

– *pre-build*

– *build*

– *notificação*

2.2.2 Execução e Disparo do Pipeline

Após a preparação local da pasta e a criação do repositório remoto **UNOPAR_CI_CD** no GitLab, os arquivos foram enviados através da sequência de comandos GIT:

– **Sincronização do Histórico**: Utilização do *git pull origin main –allow-unrelated-histories* para resolver divergências de histórico (causadas pelo README inicial no repositório remoto).

– **Disparo do CI**: O comando final *git push -u origin main* enviou o código para o GitLab , servindo como gatilho para a execução automática do pipeline de Integração Contínua.

2.2.3 Monitoramento e Simulação

A principal metodologia de avaliação e aprendizado concentrou-se no monitoramento do **pipeline** na interface web do GitLab (Seção Build > Pipelines).

– **Simulação de Falha (Diagnóstico)**: Foi registrado e documentado o pipeline inicial que falhou no job *build-docker* (estágio *pre-build*), analisando-se o log para identificar a causa raiz (erro de autenticação no *docker push*).

– **Correção**: O *.gitlab-ci.yml* foi corrigido localmente para ajustar as credenciais e referências do Docker Hub, e a correção foi enviada via novo *git push*.

– **Simulação de Sucesso (Validação)**: Foi monitorado o pipeline de correção até que todos os jobs fossem aprovados, confirmando o sucesso na construção e entrega automática da aplicação

2.3 Arquivos Iniciais do Projeto UNOPAR_CI_CD

A estrutura do projeto é a base para o pipeline de Integração Contínua (CI) e a Entrega Contínua (CD), sendo o arquivo .gitlab-ci.yml o principal orquestrador.

2.3.1 *.gitlab-ci.yml* (O Orquestrador do Pipeline)

Tabela 1 – O Orquestrador do Pipeline

Configuração	**Detalhes**	**Observações (Placeholders)**
`image (Global)`	`docker:stable`	Imagem base utilizada para executar os comandos Docker (build, login, push)
`stages`	`pre-build, build, notificacao`	Sequência de execução do pipeline.
`build-docker`	Job de `pre-build`. Faz login `(CI_REGISTRY_USER)`, *build* da imagem `(minha-imagem)` e *push* para o Docker Hub.	**Possui Placeholders**: Utiliza variáveis de CI/CD não definidas `(CI_REGISTRY_USER, CI_REGISTRY_PASSWORD)` e *placeholders* `(nome_do_docker_hub)` que causam a falha inicial.
`build-project`	Job de `build`. Executa o script Python `plot_simples_grafico.py`	**Possui Placeholder**: Utiliza a imagem `nome_do_docker_hub/ minha-imagem:latest` como ambiente.
`notificacao-sucesso`	Job de `notificacao`. Executa scripts de sucesso `(relogio.py, notificacaoSucesso.sh)` quando o *pipeline* é `on_success`.	**Possui Placeholder**: Utiliza a imagem `nome_do_docker_hub/ minha-imagem:latest.`
`notificacao-falhas`	Job de `notificacao`. Executa scripts de falha `(relogio.py, notificacaoFalha.sh)` quando o *pipeline* é `on_failure`.	**Possui Placeholder**: Utiliza a imagem `nome_do_docker_hub/ minha-imagem:latest.`

2.3.2 *Dockerfile* (O Builder do Ambiente)

Este arquivo define as etapas para construir a imagem Docker da sua aplicação:

Tabela 2 – O Orquestrador do Pipeline

Instrução	**Detalhes**
[cite_start] `FROM ubuntu`	Utiliza a imagem base `ubuntu` (última disponível).
`RUN apt-get update...`	Atualiza o sistema e instala as dependências `python3 e python3-pip`.

2.3.3 Scripts e Aplicação (O Código)

Estes arquivos contêm a lógica que será testada e executada dentro dos jobs do pipeline:

Tabela 3 – O Código

Arquivo	**Tipo**	**Função**
`plot_simples_grafico.py`	Python	Contém a lógica de um script Python para processamento de dados (plotagem de gráficos simples). Executado no estágio `build`.
`relogio.py`	Python	Script Python simples que obtém e imprime a hora atual. Executado no estágio `notificacao`.
`notificacaoSucesso.sh`	Shell Script	Script simples para sinalizar o sucesso do *pipeline* com a mensagem "Sucesso". Executado no *job* `notificacao-sucesso`.
`notificacaoFalha.sh`	Shell Script	Script simples para sinalizar o sucesso do *pipeline* com a mensagem "Falha". Executado no *job* `notificacao-falhas`.

Com esta estrutura em mente, fica claro que a **correção dos placeholders** `(nome_do_docker_hub)` e a **definição das variáveis de ambiente** são necessárias para o sucesso da execução do pipeline de CI/CD.

3 Desenvolvimento e resultados

A etapa de Desenvolvimento consistiu na execução do pipeline e no monitoramento contínuo dos jobs, culminando no registro de uma Simulação de Falha e na subsequente obtenção da Simulação de Sucesso, que valida o processo de entrega contínua.

3.1 Desenvolvimento (Monitoramento e Diagnóstico)

O desenvolvimento da atividade começou com o push inicial do código para o GitLab, disparando o primeiro pipeline. O monitoramento na seção Build > Pipelines revelou uma falha imediata, o que exigiu uma abordagem de diagnóstico:

3.1.1 Simulação de Falha e Análise de Log

O pipeline inicial falhou no job build-docker (estágio pre-build). A análise detalhada do log de execução revelou os seguintes erros críticos:

– **Falha de Autenticação**:A tentativa de `docker login` falhou, indicando `unauthorized: incorrect username or password`, pois as variáveis `$CI_REGISTRY_USER e $CI_REGISTRY_PASSWORD` não estavam configuradas ou estavam incorretas.

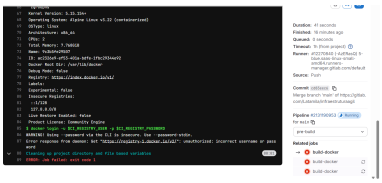

Figura 1 – Falha de Autenticação

A reprovação do job com `ERROR: Job failed: exit code 1` foi registrada como a evidência da Simulação de Falha, confirmando que o *pipeline* impediu a entrega de uma construção defeituosa.

Figura 2 – Falha da simulação por conta do arquivo errado

Figura 3 – Mudança no arquivo .gitlab-ci.yml

– **Negação de Acesso**: Em uma tentativa subsequente (após ajuste das credenciais no GitLab), o comando `docker push` falhou com a mensagem: `denied: requested access to the resource is denied`. Esta falha ocorreu porque o comando no `.gitlab-ci.yml` m utilizava um placeholder (nome_do_docker_hub/minha-imagem) em vez do nome de usuário e repositório Docker Hub corretos (ex: abracofraterno/infraestruturaagil:latest).

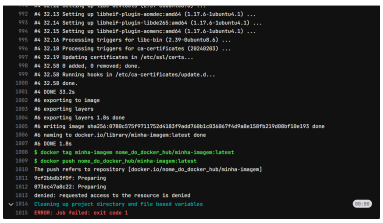

Figura 4 – Erro por conta do placeholder em vez do nome e repositório Docker Hub corretos

3.2 Resultados (Correção e Sucesso)

O objetivo de obter um pipeline aprovado foi alcançado após a aplicação das correções diretamente no código e nas configurações de variáveis do GitLab.

Figura 5 – Inclusão das variáveis login e senha do DockerHub no CI/CD Variáveis do GitLab

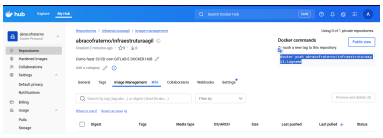

Figura 6 – Para a correção da imagem do Docker. Substitua tagname por *latest*

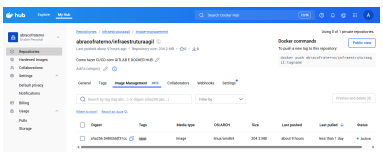

Figura 7 – Note que na coluna *tag* diz *latest* e por isso deve-se substituir, na *.gitlab-ci.yml* para eliminar o placeholder no arquivo.

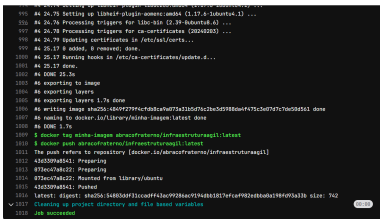

Figura 8 – Configuração correta para a variável. Ajuste com o nome do seu Docker repositório e imagem

3.3 Correções Aplicadas

3.3.1 Configuração de Variáveis

As credenciais de acesso ao Docker Hub foram inseridas como Variáveis de CI/CD (ex: `DOCKER_HUB_USER e DOCKER_HUB_PASS`) na interface do GitLab para autenticação segura.

3.3.2 Ajuste do .gitlab-ci.yml

Todos os placeholders (`nome_do_docker_hub/minha-imagem:latest`) no *.gitlab-ci.yml* (nos jobs `build-docker, build-project, notificacao-sucesso e notificacao-falhas`) foram substituídos pela referência correta do repositório no Docker Hub (ex: `abracofraterno/infraestruturaagil:latest`).

3.4 Simulação de Sucesso

Após a aplicação das correções, um novo push foi realizado, disparando o pipeline final. O monitoramento confirmou o sucesso:

– O job `build-docker` foi concluído com sucesso (autenticação e push da imagem).

– O job `build-project` executou o script `plot_simples_grafico.py` com sucesso.

– O job `notificacao-sucesso` foi executado (devido à cláusula `when: on_success`).

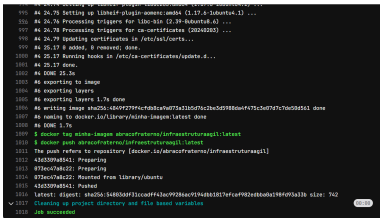

Figura 9 – Configuração correta para a variável. Ajuste com o nome do seu Docker repositório e imagem

O pipeline completo atingiu o status `Passed` (Aprovado), confirmando que a automação da construção e entrega da aplicação foi totalmente funcional.

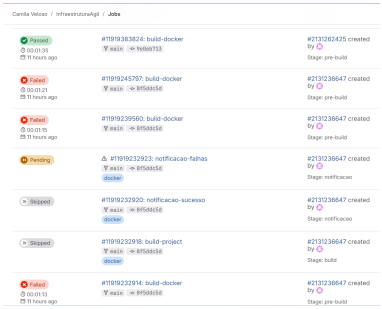

Figura 10 – Pipeline passsou na simulação após mudança nos códigos do arquivo.

3.5 CONCEITOS AVANÇADOS E EXPERTISE EM CI/CD

Esta seção aprofunda os conceitos técnicos e a arquitetura subjacente ao pipeline simulado, demonstrando a importância de cada componente dentro de um ecossistema de Engenharia de Software e DevOps

3.5.1 O Arquétipo do Pipeline e o `.gitlab-ci.yml`

O arquivo .gitlab-ci.yml não é apenas um script; ele é o Manifesto da Entrega Contínua (CD) do projeto. Ele estabelece uma Máquina de Estados Finita onde cada job é uma transição:

– **Jobs e Atomicidade**: Os jobs são unidades atômicas de trabalho. A reprovação de um único job (ex: `build-docker`) causa a falha de todo o estágio e, por padrão, paralisa o pipeline.

– **Orquestração por Estágios (stages)**: A execução é sequencial (estágio a estágio), mas paralela dentro dos estágios. O *pipeline* só avança para o estágio seguinte (`build`) se todos os jobs do estágio anterior (`pre-build`) forem concluídos com sucesso.

3.5.2 Gerenciamento de Imagens e Registries

O pipeline demonstrou a interação crucial entre o `Git`, `o GitLab` e `o Docker Hub`:

– **Serviços** (services: `docker:dind`): O job `build-docker` utiliza o serviço DOCKER IN DOCKER (docker:dind). Isso é necessário porque o job precisa criar outros contêineres (imagens), exigindo privilégios de um host Docker completo.

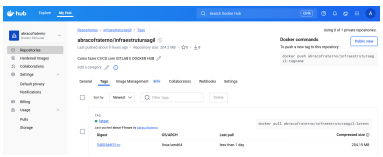

Figura 11 – Criação de contêiner (imagem) de forma automatizada no Docker Hub

– **Autenticação e Segurança**: A falha inicial demonstrou a necessidade crítica de **configurar Variáveis de CI/CD Mascaradas e Protegidas** (como `$DOCKER_HUB_USER e $DOCKER_HUB_PASS`) para o `docker login`. Isso impede que as credenciais sejam expostas em logs, aderindo às melhores práticas de segurança de secrets management.

– **Imutabilidade e Rastreabilidade**: O uso de tags específicas (como o placeholder `tagname` substituído por `latest`) no `docker tag` e `docker push` é fundamental. Em produção, tags imutáveis (baseadas no `$CI_COMMIT_SHORT_SHA`)

seriam usadas para garantir que qualquer deploy possa ser rastreado de volta a um commit específico, garantindo a rastreabilidade da entrega.

3.6 O Ciclo de Feedback e a Qualidade do Software

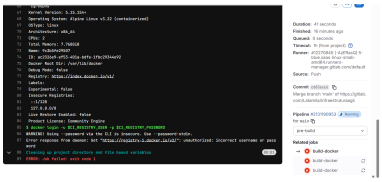

Figura 12 – exit code 01

O evento de Falha do pipeline (o exit code 1) foi o momento mais instrutivo da simulação:

- **Falha como um Ativo**: A falha no pipeline atua como um **portão de qualidade**. Ao falhar devido a um erro de configuração de entrega (`denied: requested access...`), o pipeline garantiu que uma imagem construída incorretamente (que não poderia ser usada em produção) não fosse entregue. Isso protege a integridade do repositório de imagens.
- **Execução Condicional** (`when`): Os jobs de notificação (`notificacao-sucesso e notificacao-falhas`) demonstraram a capacidade de **ação reativa** do CI/CD. Eles são acionados com base no status final do pipeline (`when: on_success ou when: on_failure`), que é crucial para alertar as equipes de *On-Call e o Release Manager* sobre a necessidade de intervenção ou sobre a conclusão bem-sucedida.
- Essa arquitetura de *pipeline*, embora básica, implementa todos os princípios de **confiabilidade, automação e monitoramento** que definem a excelência na Engenharia de Software moderna.

4 Conclusão

O presente relatório atestou o sucesso na simulação do monitoramento do pipeline de entrega, cumprindo integralmente os objetivos gerais e específicos da aula prática . O projeto demonstrou a aplicação eficaz dos princípios de **Infraestrutura Ágil e Engenharia de Software** por meio da automação do fluxo de trabalho.

Os resultados práticos validaram a importância de cada ferramenta: o **GIT** serviu como o trigger essencial para a execução do pipeline ; o **Docker Hub** confirmou-se como o elemento chave para a padronização e Entrega Contínua (CD) ; e o **GitLab CI/CD**, orquestrado pelo *.gitlab-ci.yml*, provou ser o motor de automação e validação.

A experiência mais instrutiva residiu na **Simulação de Falha**, onde o pipeline falhou no estágio `pre-build` devido a erros de autenticação (`unauthorized`) e permissão (`denied`). Esta falha reforçou o conceito de que o pipeline atua como um **portão de qualidade** crucial, impedindo que construções com configurações de entrega incorretas ou incompletas cheguem ao registro final.

O **Sucesso** subsequente, obtido após a correção das variáveis de ambiente e dos placeholders no código, validou a **confiabilidade** do processo. O *pipeline* demonstrou a capacidade de realizar a construção da aplicação de forma automática e a posterior entrega, com jobs executando scripts Python e Shell.

Em síntese, o monitoramento contínuo, aliado à **análise imediata dos logs de execução**, é a chave para garantir a rastreabilidade, a segurança (via *secrets management*) e a agilidade necessárias para manter um fluxo de trabalho eficiente e confiável em um ambiente de desenvolvimento moderno.

5 Referências

Projeto produzido no LaTex: https://www.overleaf.com/project/6904dc01530076fc6d6c6d04
DockerHub. 29/10/2025. https://hub.docker.com/
GitLab. 29/10/2025. https://gitlab.com/users/sign_in
Git. 29/10/2025. https://git-scm.com/install/windows

Referências