

Atividade 3 – Evolução de Software e CI/CD

Equipe 2

01 - Adriel Menezes Santana - 202100022659

Participação na criação do Pipeline para análise da arquitetura de software do repositório

02 - Luan Feitosa Lima Sátiro - 202300061714

Elaboração do roteiro de apresentação e manutenção do repositório de GitHub.

03 - Luan Prata Mendonça - 202000138885

Implementação do modelo deepseek-ai/DeepSeek-V3.2-Exp e análise dos resultados.

04 - Paulo Henrique Carvalho de Andrade - 202200060090

Elaboração e análise do documento, introdução, metodologia, conclusão e edição do vídeo.

05 - Paulo Henrique dos Santos Reis - 202100115524

Análise manual do repositório, ajuda na elaboração da pipeline para análise, criação e manutenção do repositório.

06 - Thiago Mecena Silva - 202100045840

Análise utilizando o script, elaboração do script e do vídeo.

07 - Victoria Moura Santos - 202000138900

Elaboração do documento e análise dos resultados.

Engenharia de Software II

Prof. Glauco de Figueiredo Carneiro

1. Introdução

Objetivo: Esta atividade tem como objetivo analisar a evolução e pipeline de CI/CD do projeto de software *open source* crawl4ai, identificando qual a ferramenta de CI/CD utilizado (GitHub Actions, Travis, CircleCI, Jenkins) se houver, e mapear o fluxo, os riscos e possíveis gargalos manuais existentes.

Projeto Analisado:

- Nome: crawl4ai
 - URL: <https://github.com/unclecode/crawl4ai>
-

2. Metodologia

Nossa equipe fez uma análise manual no repositório alvo “crawl4ai” e desenvolveu um script em Python (Engenharia_De_Software_II.ipynb) executado no Google Colab para automatizar a análise e fazer comparações. A metodologia consistiu em:

1. Coleta de Dados: O script utiliza a API do GitHub para extrair os últimos pull requests do repositório.
2. Engenharia de Prompt: Desenvolvemos um prompt específico ("Você é um engenheiro de DevOps...") para orientar as LLMs a focarem exclusivamente em padrões de *integração contínua e entrega contínua*.
3. Análise utilizando modelos via API da Hugging Face:
 - **DeepSeek V3.2 (DeepSeek AI):** Escolhido principalmente por ter sido o mais equilibrado e assertivo quando utilizado nos outros trabalhos do disciplina.

2.1. Ambiente de execução

O script foi processado em uma instância do Google Colab (Free Tier), utilizando a seguinte configuração de hardware:

- GPU: NVIDIA T4 (16 GB VRAM) - Utilizada para inferência dos modelos quantizados.
 - CPU: Intel Xeon (2 núcleos @ 2.20GHz).
 - RAM: 12.7 GB disponíveis. Esta configuração foi suficiente para executar os modelos na faixa de 7B e 8B de parâmetros utilizando quantização de 4-bit (via bitsandbytes implícito no huggingface-hub)."
-

3. Resultados da Análise

Abaixo apresentamos os resultados obtidos através de uma análise manual do repositório de investigação alvo “crawl4ai” e dos obtidos através da análise feita utilizando um modelo de linguagem do Hugging-Face “DeepSeek V3.2 (DeepSeek AI)”

3.1. Visão Geral e Ferramentas

O repositório uncode/crawl4ai adota um modelo de desenvolvimento focado em Integração e Entrega Contínua, utilizando o **GitHub Actions** como orquestrador principal . O ambiente é sustentado por **Python 3.12** e **Docker**, com integrações externas para comunicação via Discord e Google Apps Script.

Ferramentas Principais Identificadas:

- **Orquestração:** GitHub Actions .
- **Qualidade e Testes:** flake8 (linting), mypy (tipagem), pytest (testes unitários), Codecov (cobertura) .
- **Empacotamento e Build:** Poetry (Python) e Docker Buildx (Containers multi-arquitetura) .
- **Segurança:** Ferramenta safety para auditoria de dependências .
- **Notificações:** Webhooks do Discord e scripts Google Apps .

Além disso, o repositório utiliza uma arquitetura de pipeline de liberação dividida para otimizar o tempo de lançamento e oferecer flexibilidade. O processo de lançamento é dividido em dois fluxos de trabalho independentes:

1. Release pipeline() - publicação rápida de lançamentos no PyPI e Github release.yml
2. Docker release() - builds de imagem Docker multi arquitetural com cache docker-release.yml

Por que dividir os fluxos de trabalho?

Problema: build multi-arquitetura em Docker levam de 10 à 15 minutos, bloqueando lançamentos rápidos de pacotes

Solução: Docker separado se constroi em um fluxo de trabalho independente que roda em paralelo

Benefícios:

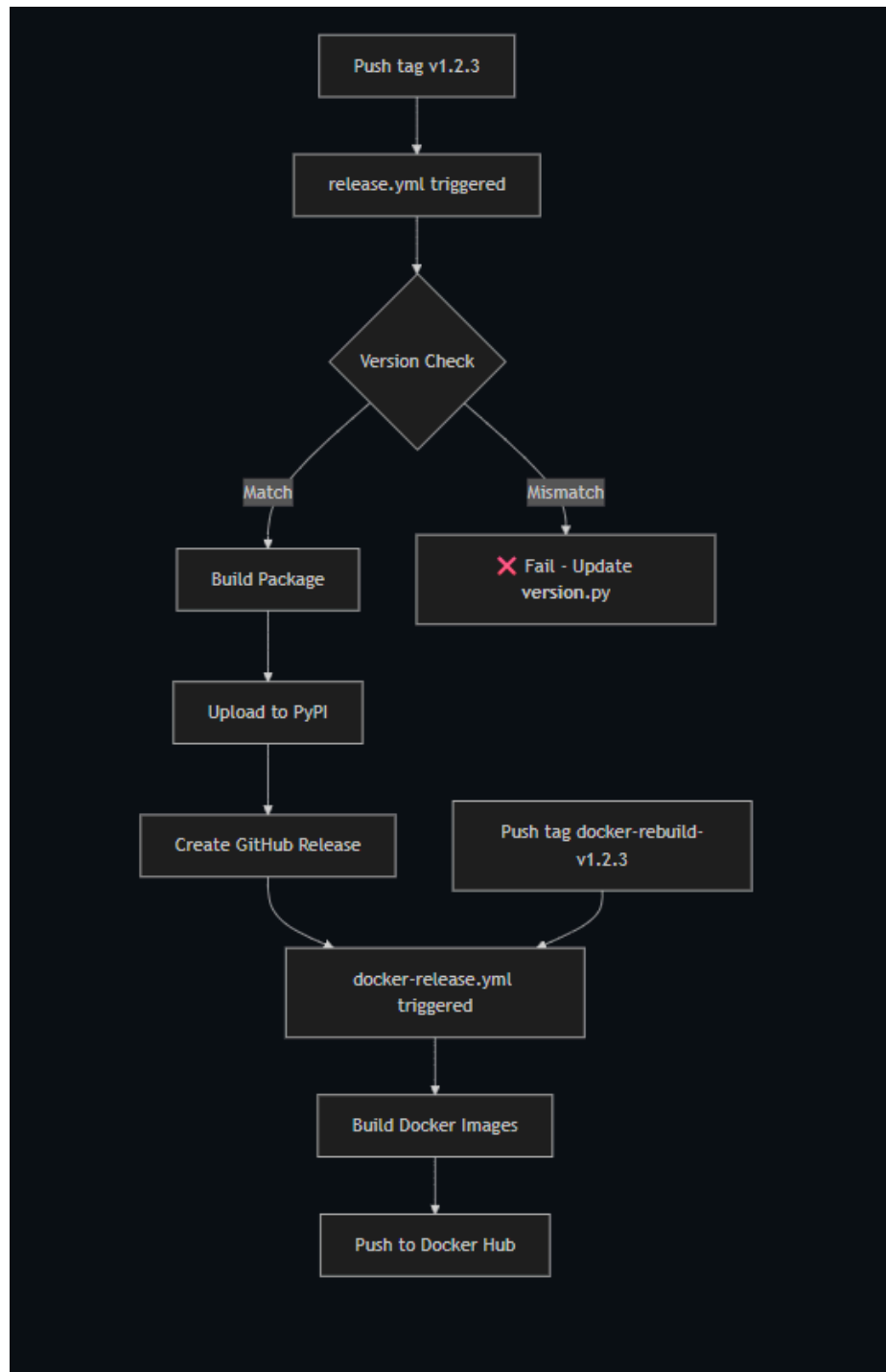
- Pacote PyPI disponível em ~2-3 minutos.

- Lançamento no Github publicado imediatamente
- Imagens Docker construídas em paralelo (sem bloqueio)
- É possível reconstruir imagens do Docker de forma independente
- Builds subsequentes mais rápidas com cache de camadas

3.2. Arquitetura de Fluxo de Trabalho

```
Tag Push (v1.2.3)
├─> Release Pipeline (release.yml)
│   ├── Version validation
│   ├── Build Python package
│   ├── Upload to PyPI ✓
│   └── Create GitHub Release ✓
│       └─> Triggers Docker Release (docker-release.yml)
│           ├── Build multi-arch images
│           ├── Use GitHub Actions cache
│           └── Push to Docker Hub ✓
└─> Total Time:
    - PyPI/GitHub: 2-3 minutes
    - Docker: 1-15 minutes (parallel)
```

Fluxo de Eventos:



3.3. Mapeamento dos Fluxos de Automação (CI/CD)

A automação é dividida em três fluxos de trabalho principais (workflows), cada um com responsabilidades distintas:

A. Integração Contínua (CI) - O Guardião da Qualidade

Arquivo principal: main.yml

Ao contrário de uma análise superficial que pode apontar a falta de testes, uma inspeção detalhada revela que o pipeline de CI é robusto e acionado a cada push ou Pull Request (PR) para a branch main.

- **Verificação de Estilo:** Uso do flake8 para garantir padrões de código.
- **Tipagem Estática:** O mypy é executado para prevenir erros de tipo antes da execução.
- **Testes Automatizados:** Execução de suíte de testes com pytest e relatório de cobertura via pytest-cov enviado ao Codecov.
- **Auditoria de Segurança:** Escaneamento de vulnerabilidades em dependências com safety. **Nota:** Uma vulnerabilidade específica no urllib3 (CVE-2023-45133) é explicitamente ignorada na configuração atual.

B. Entrega Contínua (CD) - A Fábrica de Releases

A entrega é acionada manualmente pela criação de uma *Release* ou Tag no GitHub.

1. Pipeline de Pacotes Python (release.yml)

- Verifica a consistência da versão comparando a Tag do Git com o arquivo `__version__` interno.
- Utiliza Poetry para construir os pacotes de distribuição (sdist e wheel).
- Publica o artefato no **PyPI**, tornando-o instalável via pip.

2. Pipeline de Containers (docker-release.yml)

- Acionado por releases ou tags específicas de reconstrução (docker-rebuild-v*).
- Utiliza **QEMU e Docker Buildx** para criar imagens compatíveis com múltiplas arquiteturas (linux/amd64, linux/arm64).
- Realiza o push das imagens para o Docker Hub com tags dinâmicas baseadas no SHA do commit e versão.

C. Fluxo de Notificações e Operações

- Eventos como criação de issues, novos PRs ou "stars" no repositório disparam notificações automáticas para o **Discord** e integrações via Google Apps Script, mantendo a comunidade e os desenvolvedores informados.

3.4. Mapeamento dos Fluxos de Automação (CI/CD)

Enquanto a automação técnica é forte, a gestão humana do processo apresenta desafios.

- **Padrões de PR:** Observa-se uma predominância de correções de bugs (fix) sobre novas funcionalidades (feat) .
- **Gargalo de Contribuição Externa:** PRs vindos de terceiros (associação NONE) possuem baixa taxa de merge e tendem a ficar pendentes .
- **Etapas Manuais:** A revisão de código, discussão e aprovação final para merge são inteiramente manuais, sem evidência de políticas rigorosas de bloqueio automático (ex: exigir 2 aprovações) .
- **Falta de Categorização:** A ausência de "labels" (rótulos) consistentes nos PRs dificulta a triagem rápida entre bugs, documentação e funcionalidades .

3.5. Matriz de Riscos e Gargalos Identificados

Área	Risco / Gargalo	Detalhe
Processo	Dependência Manual	O disparo do deploy depende exclusivamente da criação humana de uma Release/Tag, podendo gerar atrasos ou erros de versionamento .
Qualidade	Falta de Testes Pós-Deploy	Não há evidência de testes de fumaça (smoke tests) após a publicação no PyPI ou Docker Hub para garantir que o pacote publicado está funcional .

Comunidade	Barreira para Contribuintes	A baixa taxa de merge de PRs externos e a falta de templates/labels podem desestimular a comunidade open-source .
Documentação	Manutenção Manual	Changelogs e documentação de updates parecem ser gerados manualmente, criando inconsistências potenciais .

4. Conclusão

O SDLC (Ciclo de Vida de Desenvolvimento de Software) do unclecode/crawl4ai tem uma base sólida de automação (GitHub Actions + Docker), é impulsionado por um modelo de desenvolvimento focado em Integração e Entrega Contínua, utilizando GitHub Actions como sua principal orquestração. O ciclo começa com o **desenvolvimento de código** por contribuidores. Ao submeter um **Pull Request** ou realizar um **push** para branches designados (como main ou release/*), o sistema de **Integração Contínua (CI)** é imediatamente acionado.

Na fase de **CI**, o código passa por rigorosas verificações automatizadas de qualidade e segurança. Isso inclui **linting** com flake8 para aderência ao estilo, **verificação de tipo** com mypy para robustez, **execução de testes** com pytest (com cobertura reportada ao Codecov) para funcionalidade e **auditoria de dependências** com safety para identificar vulnerabilidades. Durante este tempo, o **processo de revisão de código** (manual) acontece, com discussões e aprovações no Pull Request. Uma vez que o código atende aos padrões e é aprovado, ele é **mergeado** (passo manual) no branch main.

A **Entrega Contínua (CD)** é iniciada pela **criação manual de um release no GitHub**. Este evento dispara dois processos paralelos: a **publicação do pacote Python** no PyPI (release.yml) e a **construção e publicação de imagens Docker multi-arquitetura** no Docker Hub (docker-release.yml). Ambas as pipelines garantem que o crawl4ai esteja disponível em diferentes formatos para os usuários finais, utilizando ferramentas como Poetry para pacotes Python e Docker Buildx para imagens Docker.

As principais **ferramentas de CI/CD** são GitHub Actions, flake8, mypy, pytest, Codecov, safety, Poetry, Docker, Buildx e QEMU. O README.md serve como o principal ponto de

documentação e comunicação de novas funcionalidades e status do projeto. A criação de releases e a revisão de Pull Requests são as principais **etapas manuais** que atualmente governam o fluxo de avanço do código do desenvolvimento para a distribuição.

5. Artefatos e Reprodutibilidade

Todo o código utilizado, logs brutos gerados pelas IAs e documentação complementar estão disponíveis publicamente para replicação desta atividade. O repositório do projeto foi estruturado para garantir a auditabilidade:

- /src: Contém o notebook Jupyter (.ipynb) com o código fonte.
- /data/processed: Contém os relatórios em Markdown gerados pela IA
- /docs: Contém este tutorial em PDF.

Nome:

Adriel_Menezes_Luan_Feitoso_Luan_Prata_Paulo_Carvalho_Thiago_Mecena_Paulo_Santos_Victoria_Moura_Atividade_3.mp4

Link para o Vídeo:

https://www.youtube.com/watch?v=Jsm7scBx7_I

Link para o GitHub:

https://github.com/Paulinhoh/Engenharia_Software_2025-2_T04_crawl4ai_atividade3_parte1