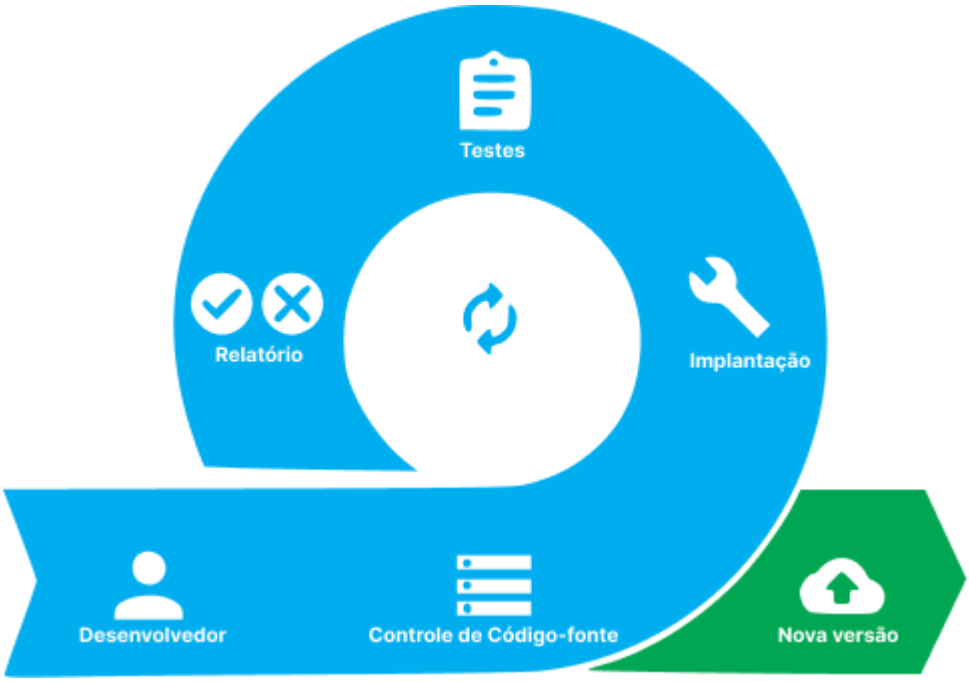
Continuous Integration(CI) e Continuous Delivery/Continuous Deployment(CD)

Os conceitos *Continuous Integration(CI)* e *Continuous Delivery/Continuous Deployment(CD)* são práticas para desenvolvimento de software que permitem a integração e entrega do código de forma contínua. De modo a permitir a redução no tempo de entrega de novas funcionalidades e correções de bugs, além de aumentar a qualidade e velocidade do desenvolvimento de software.

Continuous Integration(CI)

Continuous Integration é uma prática de desenvolvimento de software onde cada desenvolvedor envolvido no processo deve realizar *commits* de forma frequente, de modo que o código seja integrado ao repositório principal de forma contínua. Para isso, são utilizadas ferramentas de automatização para realizar processos como *building* e *testes* de forma automática. Algumas das principais ferramentas utilizadas para *Continuous Integration* são *Jenkins*, *Github Actions*, *CircleCI*, *Travis CI*, etc.



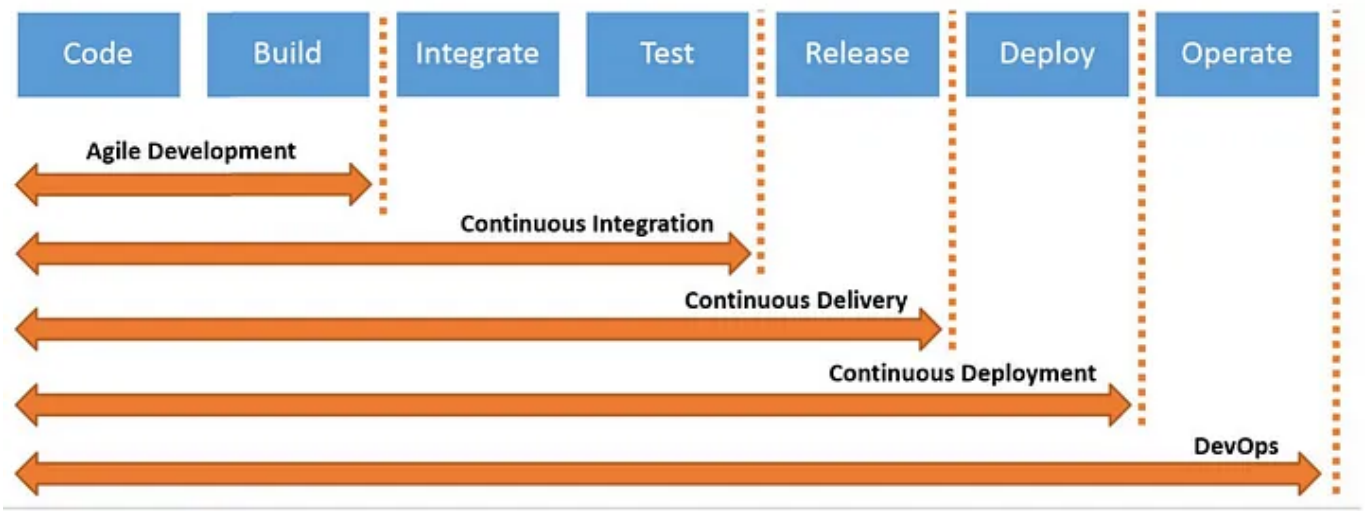
Continuous Delivery/Continuous Deployment(CD)

Continuous Delivery e *Continuous Deployment* são práticas de desenvolvimento de software que tem como objetivo automatizar o processo de entrega de software. Ambas as práticas consistem em automatizar o processo de *build* e *testes*.

A diferença entre essas práticas é que *Continuous Delivery* consiste em automatizar o processo de *build* e *testes* de forma que o software possa ser entregue manualmente, enquanto *Continuous Deployment* consiste em automatizar o processo de *build* e *testes* de forma que o software possa ser entregue automaticamente.

Fluxo de Trabalho com CI/CD

A imagem abaixo ilustra um fluxo de trabalho com CI/CD.



Ferramentas

Na tabela abaixo, são listadas algumas das ferramentas CI/CD mais utilizadas:

Nome	Plataformas suportadas	Tipo de hospedagem	Open Source	Possui versão gratuita?
Github Actions	Linux, Windows e macOS	Nuvem	✔	✔
Jenkins	Linux, Windows e macOS	Auto-hospedagem	✘	✔
CircleCI	Linux, Windows, macOS, GPU, ARM e Docker	Nuvem e Auto-hospedagem	✘	✔
Travis CI	Linux, macOS e iOS	Nuvem e Auto-hospedagem	✘	✘

Fonte: Best Continuous Integration Tools for 2023 – Survey Results

Após pesquisas realizadas com 26,348 desenvolvedores em 2023 pela *JetBrains*, foi possível obter os seguintes resultados em relação às ferramentas CI/CD mais utilizadas:

Nome	Uso Pessoal	Uso Profissional	Total
Github Actions	37%	29%	51%

Nome	Uso Pessoal	Uso Profissional	Total
Jenkins	12%	50%	54%
CircleCI	5%	7%	11%
Travis CI	5%	4%	9%

Fonte: Team Tools - The State of Developer Ecosystem in 2023 Infographic

A partir disso, é possível perceber que o Github Actions é a ferramenta mais utilizada para CI/CD para uso pessoal, enquanto o Jenkins é a ferramenta mais utilizada para uso profissional. Isso é explicado pelo fato de que o Github Actions possui uma versão gratuita e é facilmente integrado com o Github(ferramenta de versionamento mais utilizada, de acordo com a pesquisa), facilitando o uso pessoal, enquanto o Jenkins é uma ferramenta mais robusta, com vários *plugins* e que pode ser auto-hospedada, o que facilita o uso profissional devido a questões de infraestrutura e possibilidade de customização.

Vantagens e Desvantagens

Na tabela abaixo, são listadas algumas vantagens e desvantagens da utilização de CI/CD.

Vantagens	Desvantagens
Risco reduzido	Necessidade de mudanças culturais e organizacionais
Tempo de Revisão mais curto	
Caminho mais suave para a produção	
Correções de bug mais rápidas	
Infraestrutura eficiente	
Progresso mensurável	
Loops de feedback mais curtos	
Colaboração e comunicação	

Fonte: Quais são os benefícios da CI/CD?

Implementando CI/CD com Github Actions

Após listadas as diversas ferramentas CI/CD, será demonstrado como implementar um fluxo utilizando o Github Actions. O Github Actions é uma ferramenta de CI/CD que permite a criação de fluxos de CI/CD de forma simples e integrada com o Github, permitindo a execução de *builds*, *testes*, *deploy*, etc. em diversas plataformas, como Linux, Windows e macOS. Como foi visto anteriormente, o Github Actions é a ferramenta mais utilizada para CI/CD para uso pessoal, devido à sua facilidade de uso e integração com o Github.

Para isso, será utilizado um projeto simples em Java, que pode ser encontrado no link a seguir:

- github-actions-ci-cd

O projeto consiste no desenvolvimento de um contador simples e foram feitos alguns testes unitários para validar seu funcionamento. O contador e seus respectivos testes podem ser encontrados nos seguintes arquivos:

- Contador: Counter.java
- Testes: CounterTest.java

Criando um fluxo de CI/CD

Para criar um fluxo de CI/CD utilizando o Github Actions, é necessário criar um arquivo de configuração do fluxo. Esse arquivo deve ser criado no diretório `.github/workflows` do repositório. O nome do arquivo deve seguir o padrão `nome-do-fluxo.yml`.

Dentro desse repositório, já existe um diretório `.github/workflows`, que contém os arquivos de configuração do Github Actions. O arquivo `build-and-publish.yml` contém o fluxo de CI/CD que será utilizado.

Para executá-lo da maneira que foi pensado, é necessário criar algumas chaves de acesso no GitHub.

Criando chave de acesso ao Github Packages

Para criar um token de acesso ao Github Packages, deve-se seguir os seguintes passos:

1. Acessar as configurações do Github
2. Acessar a aba `Developer settings`
3. Acessar a aba `Personal access tokens / Tokens (classic)`
4. Clicar em `Generate new token / Generate new token (classic)`
5. Preencher o campo `Note` com `MAVEN_TOKEN_KEY`
6. Marcar a opção `write:packages`
7. Clicar em `Generate token`

Configurando uma variável de ambiente

Um variável de ambiente `MAVEN_TOKEN_KEY` deve ser criada no repositório, contendo um token de acesso ao Github Packages(criado no passo anterior). Para isso, deve-se seguir os seguintes passos:

1. Acessar as configurações do repositório
2. Acessar a aba `Secrets and variables`
3. Clicar em `New repository secret`
4. Preencher o campo `Name` com `MAVEN_TOKEN_KEY`
5. Preencher o campo `Value` com um token de acesso ao Github Packages
6. Clicar em `Add secret`

Configurando o fluxo

O arquivo `build-and-publish.yml` contém o seguinte conteúdo:

```
name: Build and Publish Java Packages
```

```yaml
on:
    push:
        branches: ["main"]
    pull_request:
        branches: ["main"]

jobs:
    build:
        runs-on: ubuntu-latest
        steps:
            - uses: actions/checkout@v3
            - name: Set up JDK 17
              uses: actions/setup-java@v3
              with:
                  java-version: "17"
                  distribution: "temurin"
            - name: Build with Maven
              run: mvn -B package --file pom.xml
    publish:
        needs: build
        runs-on: ubuntu-latest
        permissions:
            packages: write
            contents: read
        steps:
            - uses: actions/checkout@v3
            - name: Set up JDK 17
              uses: actions/setup-java@v3
              with:
                  java-version: "17"
                  distribution: "temurin"
            - name: Publish to GitHub Packages Apache Maven
              run: mvn --batch-mode deploy
              env:
                  GITHUB_TOKEN: ${{ secrets.MAVEN_TOKEN_KEY }}
```

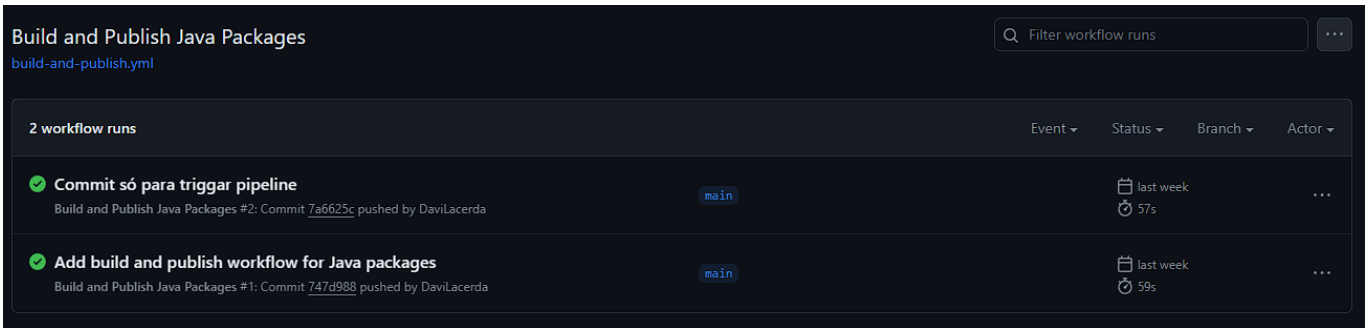
O arquivo acima contém as seguintes configurações:

- `name`: Nome do fluxo
- `on`: Eventos que disparam o fluxo
    - `push`: Push no repositório
        - `branches`: Branches que disparam o fluxo
    - `pull_request`: Pull Request no repositório
        - `branches`: Branches que disparam o fluxo
- `jobs`: Jobs que serão executados
    - `build`: Nome do job
        - `runs-on`: Sistema operacional que o job será executado(ubuntu-latest: executado em uma máquina virtual com Ubuntu)
        - `steps`: Passos que serão executados
            - `uses`: Ação que será executada
            - `name`: Nome do passo

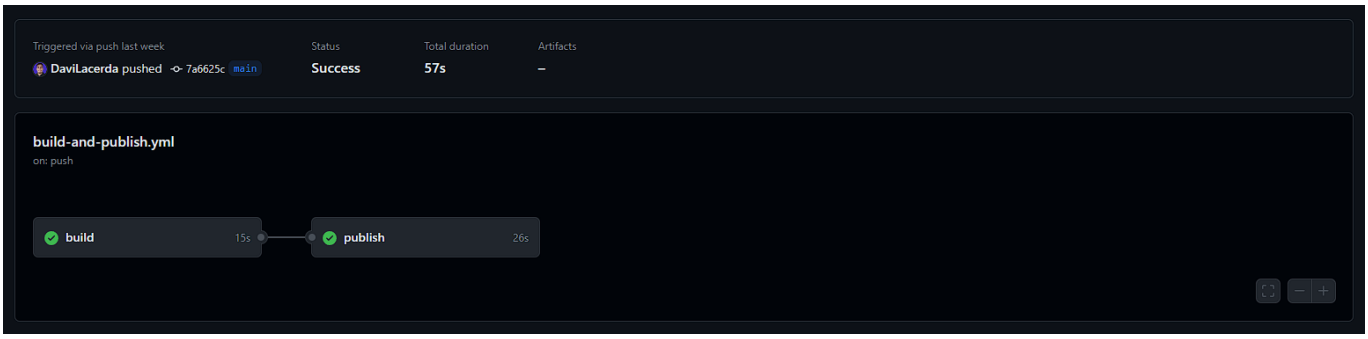
        - `with`: Parâmetros da ação
        - `run`: Comando que será executado
    - `publish`: Nome do job
        - `needs`: Job que deve ser executado antes desse job
        - `runs-on`: Sistema operacional que o job será executado(ubuntu-latest: executado em uma máquina virtual com Ubuntu)
        - `permissions`: Permissões que o job terá
            - `packages`: Permissões de acesso ao Github Packages
            - `contents`: Permissões de acesso ao repositório
        - `steps`: Passos que serão executados
            - `uses`: Ação que será executada
            - `name`: Nome do passo
            - `with`: Parâmetros da ação
            - `run`: Comando que será executado
            - `env`: Variáveis de ambiente que serão utilizadas

Resultados

Após a criação do fluxo, ao realizar um push no repositório/pull request no repositório, o fluxo será executado. O resultado da execução pode ser visto na imagem abaixo:

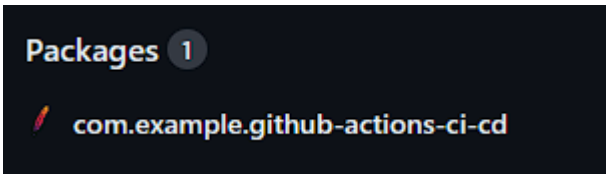


Ao clicar em uma das execuções, é possível ver um resumo dos jobs executados, como na imagem abaixo:



Nele, é possível ver que o job `publish` foi executado após o job `build`, devido à configuração `needs: build`.

Além disso, ao ir na aba `Packages` do repositório(na seção `Code`), é possível ver que o pacote foi publicado no Github Packages, como na imagem abaixo:



Referências

- What is CI/CD?
- Team Tools - The State of Developer Ecosystem in 2022 Infographic
- Team Tools - The State of Developer Ecosystem in 2023 Infographic
- TeamCity CI/CD Guide
- CI/CD explicados: suas diferenças e o que você precisa saber | Unity
- What is CI? - Continuous Integration Explained - AWS
- What is continuous deployment? | IBM
- Integração contínua vs. Entrega contínua vs. Implementação contínua