
FRAMEWORKS
FULL STACK

Texto base

10

Introdução à Continuous Integration com testes automatizados

Prof. André Nascimento Maia
Prof. Caio Nascimento Maia

Resumo

Apresenta uma visão geral sobre Continuous Integration e teste unitário em uma web API que utiliza o framework Flask.

10.1. Introdução

Todo desenvolvimento de *software*, independente da quantidade de código e responsabilidade, tem *bugs*. A prática de *Continuous Integration* (CI) pressupõe auxiliar na diminuição de diversos *bugs* tornando-os mais fáceis de encontrar e corrigi-los.

Uma das formas mais comuns de garantir a estabilidade das funcionalidades de uma aplicação é através de testes automatizados. Testes unitários são as formas mais simples de se verificar pequenos trechos de código.

Quando testes automatizados e CI são aplicados a um projeto de desenvolvimento de *software*, a qualidade do *software* criado aumenta significativamente permitindo ciclos menores de entregas e verificação.

10.2. Continuous Integration

Continuous Integration (CI) é uma prática comum em desenvolvimento de *software* que auxilia na garantia da qualidade do *software* que está sendo construído, permitindo que todos os membros de um time integrem seus código continuamente, geralmente com cada integrante integrando seu código pelo menos uma vez ao dia (Fowler, 2006).

A ideia principal é que seja construído um *workflow* com todos os passos de verificação para a aplicação e esse *workflow* é executado automaticamente de forma contínua. Geralmente, os passos para um *workflow* de CI são:

1. Clonagem do repositório Git de dados na *branch* específica que sofreu a alteração de código;
2. Construção da versão final do *software* com o teste integrado;
3. Execute todos os testes automatizados disponíveis da aplicação;
4. Notifique o resultado da integração contínua.

Além desses 4 passos, também podem ser incluídos outros passos como análises estáticas de código para identificar problemas de segurança (Micro Focus, 2022) ou problemas análise de qualidade em geral do código como cobertura de código com testes automatizados, crescimento da base de código, cálculos de graus de confiabilidade e manutenibilidade do código, bem como suas tendências (Sonarsource, 2022).

10.2.1 Exemplo de CI

Um dos serviços mais utilizados para execução de CI é o GitHub Actions (Github, 2022). Armazenando o código de sua aplicação em um repositório Git no Github, é possível disparar a execução de um *workflow* a partir de qualquer evento do repositório.

O Código 10.1 mostra um arquivo YAML para a configuração de um *workflow* de CI no GitHub Actions.

Código 10.1: Yaml de workflow de CI

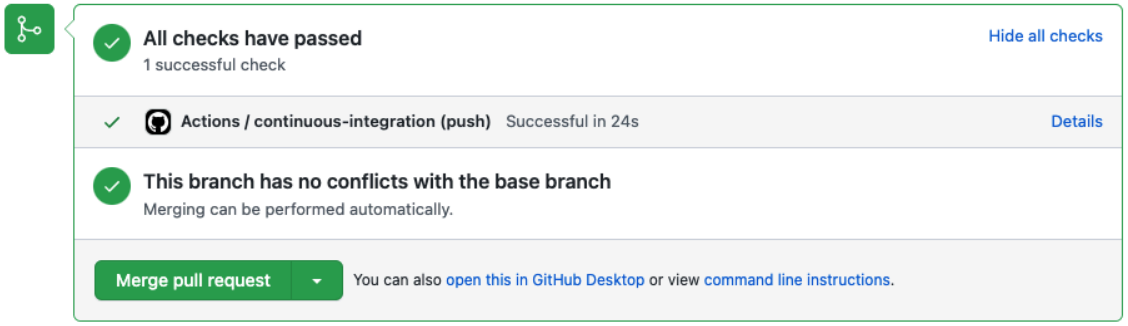
```
01. name: Actions
02. on: [push]
03. jobs:
04.   continuous-integration:
05.     runs-on: ubuntu-latest
06.     steps:
07.       - name: Check out repository code
08.         uses: actions/checkout@v2
09.       - name: Set up Python 3.8.9
10.         uses: actions/setup-python@v2
11.         with:
12.           python-version: "3.8.9"
13.       - name: Install dependencies
14.         run: |
15.           python -m pip install --upgrade pip
16.           pip install flake8 pytest
17.           if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
18.       - name: Test with pytest
19.         run: |
20.           pytest
```

Fonte: autores, 2022.

O *workflow* do Código 10.1 é composto de 4 passos (linhas 07, 09, 13, e 18). O primeiro faz o *checkout* do repositório Git na máquina ubuntu do ambiente do Github. O segundo instala a versão 3.8.9 do Python. Já o terceiro passo, faz a instalação das dependências presentes no arquivo requirements.txt, se o arquivo estiver presente, atualizando a versão do pip e instalando alguns pacotes básicos. E, finalmente, o quarto e último passo executa os testes unitários utilizando o comando pytest.

Ao abrir uma Pull Request (Github, 2022) sobre a *branch* que contém o código enviado ao repositório Git da aplicação, automaticamente o *workflow* é disparado. A Figura 10.1 mostra os detalhes da execução do *workflow* criado para uma web API com um conjunto de APIs para um e-commerce fictícios.

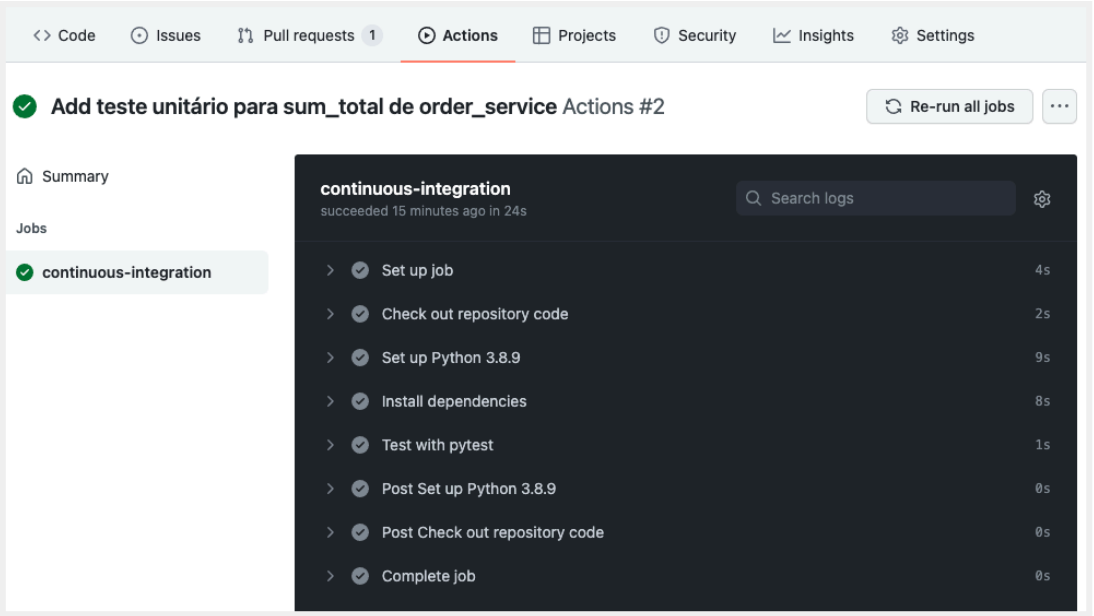
Figura 10.1: Resultado das Actions em uma Pull Request do GitHub.



Fonte: autores, 2022.

Clicando em *Details* sobre a execução do *workflow* da Figura 10.1, você será redirecionado para a página dos detalhes de execução, conforme Figura 10.2.

Figura 10.2: Resultado da execução de um workflow no GitHub Actions



Fonte: autores, 2022.

A Figura 10.2 mostra os detalhes de uma execução de workflow após o *commit* com título "Add teste unitário para sum_total de order_service". Ao lado esquerdo

aparecem todos os *jobs* configurados no *workflow*, no caso temos um único *job* chamado *continuous-integration* e do lado direito o resultado de todos os passos executados.

10.3. Teste unitário

Testar um *software* é verificar se para uma entrada específica no software o resultado é uma saída conforme o esperado. Testes unitários são testes sobre a menor parte testável de um *software*. Um dos objetivos dos testes unitários é que eles sejam simples para construir e rápidos em sua execução. Dessa forma, podemos ter centenas ou milhares de testes unitários para uma aplicação que mesmo assim será possível executá-los em um curto período de tempo.

Como exemplo, considere o teste unitário do Código 10.2.

Código 10.2: Teste unitário de exemplo

```
from api.service import order_service


def test_sum_total():
    # Given
    products = """
    [
        { "qty": 1, "unitPrice": 123.45 },
        { "qty": 2, "unitPrice": 123.45 },
        { "qty": 1, "unitPrice": 123.45 }
    ]
    """
    expected = 493.8


    # When
    actual = order_service.sum_total(products)


    # Then
    assert expected == actual
```

Fonte: autores, 2022.

O Código 10.2 é o suficiente para testar a função *sum_total* do *order_service*. Essa função tem como objetivo receber uma lista de produtos em JSON e somar o seu valor total, multiplicando a quantidade de cada item pelo seu valor. O teste unitário está dividido em três seções chamadas de Given, When e Then que é um template muito utilizado para guiar a escrita e entendimento cenários de teste quando trabalhamos em uma abordagem ágil (Agile Alliance, 2017). *Given* é o contexto esperado para o teste, no caso, a lista de produtos e o valor esperado para o total dos produtos; *When*, é a ação necessária para executar o cenário de teste; e *Then* é o que pode ser observado e verificado após a execução da ação.

10.4. Conclusões

CI e testes unitários são práticas importantes para o desenvolvimento de *software* robusto e de qualidade. Principalmente, quando existe um time participando do desenvolvimento.

Referências

Agile Alliance. **What is "Given - When - Then"?** Agile Alliance, 13 jun. 2017. Disponível em: <https://www.agilealliance.org/glossary/gwt/>. Acesso em: 09 fev. 2022.

Fowler, M. **Continuous Integration.** Martin Fowler, 01 mai. 2006. Disponível em: <https://www.martinfowler.com/articles/continuousIntegration.html>. Acesso em: 09 fev. 2022.

Github. **Criar um pull request.** GitHub Docs, 2022. Disponível em: <https://docs.github.com/pt/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request>. Acesso em: 09 fev. 2022.

Github. **Features • GitHub Actions · GitHub**. GitHub. 2022. Disponível em: <https://github.com/features/actions>. Acesso em: 09 fev. 2022.

Micro Focus. **Static Application Security Testing**: Fortify Static Code Analyzer. Micro Focus, 2022. Disponível em: <https://www.microfocus.com/pt-br/products/static-code-analysis-sast/overview>. Acesso em: 09 fev. 2022.

Sonarsource. **Overview | SonarQube Docs**. SonarQube Documentation, 2022. Disponível em: <https://docs.sonarqube.org/latest/analysis/overview/>. Acesso em: 09 fev. 2022.