Author: Madhurima Rawat

Streamlining Python Testing with Travis CI

A step-by-step guide to setting up a Continuous Integration (CI) pipeline for Python projects using Travis CI, ensuring automated testing and code reliability

Table of Contents

1. Overview

A **CI pipeline** automates the testing and building of software each time changes are made. **Travis CI** is a widely used cloud-based tool that integrates with GitHub to automatically run tests after every push or pull request.

This guide explains how to set up Travis CI for a Python project. We will also simulate a **failed test** before resolving it.

Learn more about Travis CI:

- Travis CI Official Documentation
- An Introduction to Continuous Integration

Note: While Travis CI offers a free tier for open-source projects, it requires credit card verification during signup, even if you intend to use it for free. This ensures legitimate usage, and you will not be charged unless you exceed the limits of the free plan. Additionally, as a paid service, Travis CI may impose restrictions on usage that require careful consideration when choosing it for your projects.

2. Step 1: Setting Up CI

1. Go to Travis CI and log in using your GitHub account.

2. On the Travis dashboard, enable Travis CI for your repository.
3. Add a `.travis.yml` file to the root of your repository.

```
language: python
python:
  - "3.8"

# Install dependencies
install:
  - pip install pytest

# Run tests
script:
  - pytest test_script.py
```

3. Understanding the `.travis.yml` File

The `.travis.yml` file is a configuration file that tells Travis CI how to build and test your project. Here's a detailed explanation of its components:

1. **`language: python`**:
   This specifies the programming language for the project. Travis CI supports a wide range of languages, and here we are specifying Python as the language used.

2. **`python:`**:
   This key specifies which versions of Python to use during the testing process. The value here is a list, so you can include multiple versions. In this example, we use:

   ```
   python:
     - "3.8"
   ```

   This tells Travis to use Python version 3.8 during the build.

3. **`install:`**:
   This section defines the commands that will run to install the dependencies needed for the project. In this case:

   ```
   install:
     - pip install pytest
   ```

   This command uses `pip` (Python's package installer) to install the `pytest` library, which is required to run the tests.

4. **`script:`**:
   This section tells Travis what command to run to execute the tests. The commands under `script:` are the core of the CI pipeline because this is where your test suite runs. In this example:

   ```
   script:
     - pytest test_script.py
   ```

   Travis will run `pytest` to execute the tests defined in the `test_script.py` file. If any tests fail, Travis will mark the build as failed.

4. Step 2: Writing Python Code and Tests

For this guide, we will use the following Python files:

1. Create `script.py`:

```
# script.py

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

2. Create `test_script.py`:

```
# test_script.py

from script import add, subtract

def test_add():
    assert add(3, 2) == 5  # This will pass

def test_subtract():
    assert subtract(5, 2) == 4  # This will fail
```

In this setup, we have two functions: `add` and `subtract`. We will simulate a failed test by asserting that `subtract(5, 2)` should return 4, which is incorrect. This will trigger a failed test on Travis CI.

5. Step 3: Viewing CI Results

1. After pushing your code to GitHub, Travis CI will automatically trigger a build.

2. You can monitor the build process on the Travis CI dashboard or check the build logs in your repository.

6. Successful Test Example

To correct the failed test, update `test_script.py` as follows:

```
# test_script.py

from script import add, subtract

def test_add():
    assert add(3, 2) == 5  # Passes

def test_subtract():
    assert subtract(5, 2) == 3  # Corrected test
```

Push this updated version of the test, and Travis CI will rerun the tests. This time, all tests will pass.

7. Conclusion

With Travis CI set up, we now have an automated pipeline that ensures code is tested before every commit. This helps catch issues early and improves the reliability of the codebase.

8. Further Reading:

- CI/CD Best Practices - Explore more about best practices in CI/CD and how it fits into DevOps workflows.
- Automated Testing Strategies - Learn about different testing strategies for automation using tools like PyTest, Selenium, and others.