Travis CI

> The simplest way to test and deploy your projects... At Travis CI we aim to empower people to build and ship great software. - Travis CI Website

> Travis CI makes it so much easier for us to coordinate the thousands of commits and contributors that flow through the Rails code base. The test suite for such a large project is vast, and we wouldn't be catching issues as quickly or smoothly without the help of Travis. - Testimonial from @dhh

References

- https://travis-ci.com/
- https://docs.travis-ci.com/user/tutorial/
- https://docs.travis-ci.com/user/tutorial/#to-get-started-with-travis-ci

Usage

See the "Continuous Integration 1, 2, 3" Exercise for example usage.

> NOTE: To configure Travis CI on an existing repository, visit the Travis account settings and click "Manage repositories on GitHub", where you will be redirected to GitHub and prompted to select which repositories to configure.

Configuration

After enabling your GitHub repository to integrate with Travis CI, add a special file called ".travis.yml" to further configure the CI server. The file should go in the repository's root directory, and should be included in version control. Here is an example which instructs it to install python, install package dependencies, and run tests:

```
dist: xenial
language: python
python:
  - "3.7"
install:
  - pip install -r requirements.txt
script:
  - pytest
```

> NOTE: The `dist: xenial` setting is necessary at this time for installation of Python version 3.7, and the `install: pip install -r requirements.txt` setting is only necessary if there are packages to install, so you can omit the "install" setting in some cases. Pytest should already be automatically installed on the CI server, so there isn't a need to install it again via Pip.

Environment Variables

It is possible to configure environment variables on the CI server.

Environment Variable Security

But if an environment variable's value is sensitive (i.e. secret password or API Key), set it via the repository settings, NOT the ".travis.yml" file. This approach will keep the values secure and out of the publicly-available server logs.

Skipping Tests which issue HTTP Requests

Avoiding HTTP requests during testing helps increase the speed of those tests and decrease the burden on web servers responsible for processing those requests. Ideally, we'd use strategies for "mocking" the results of HTTP requests (i.e. pretending to make a request and returning a pre-prepared example response instead).

But when you're just starting out with automated testing, it's fine for your tests to issue live HTTP requests. If any of your tests do issue HTTP requests, prefer to run them locally but exclude them from being run on the CI server. For each test you'd like to skip from being run on the server, use `@pytest.mark.skipif` to denote that test should be skipped if a given condition is met, for example if there is an environment variable setting of `CI="true"`:

```
# app/my_script.py

import json
import requests

def get_response(stock_symbol)
    url = f"https://my-api.com/stocks?symbol={stock_symbol}"
    response = requests.get(url) # issues an HTTP request
    return json.loads(response.text)
```

```
# test/robo_advisor_test.py

import os
import pytest

from app.my_script import get_response

# expect default environment variable setting of "CI=true" on Travis CI
# see: https://docs.travis-ci.com/user/environment-variables/#default-
environment-variables
CI_ENV = os.environ.get("CI") == "true"

@pytest.mark.skipif(CI_ENV==True, reason="to avoid issuing HTTP requests on
the CI server") # skips this test on CI
def test_get_response():
    symbol = "NFLX"
    parsed_response = get_response(symbol) # issues an HTTP request (see
function definition below)
```

```
    assert isinstance(parsed_response, dict)
    assert parsed_response["Meta Data"]["2. Symbol"] == symbol
```