BETA-Rec CI Design Doc

Author: Yucheng Liang

Last Update Time: 2020-03-16

Why CI?

Contiguous Integration (CI) is a development practice where developers integrate code into a shared repository frequently. Each integration can then be verified by an **automated build and automated tests**

One of the key benefits of interating reguarly is that one can detect errors quickly and locate them more easily. As each change introduced is relatively small, pinpointing the specific change than introduced a defect can be done in a short time.

> Contiguous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.

Our goal

In this part, I'll try to list all items that we want to achieve in our CI process:

- commit message check
- code format
- unit test
- test coverage
- API test
- Auto-generate document
- Auto-deploy website

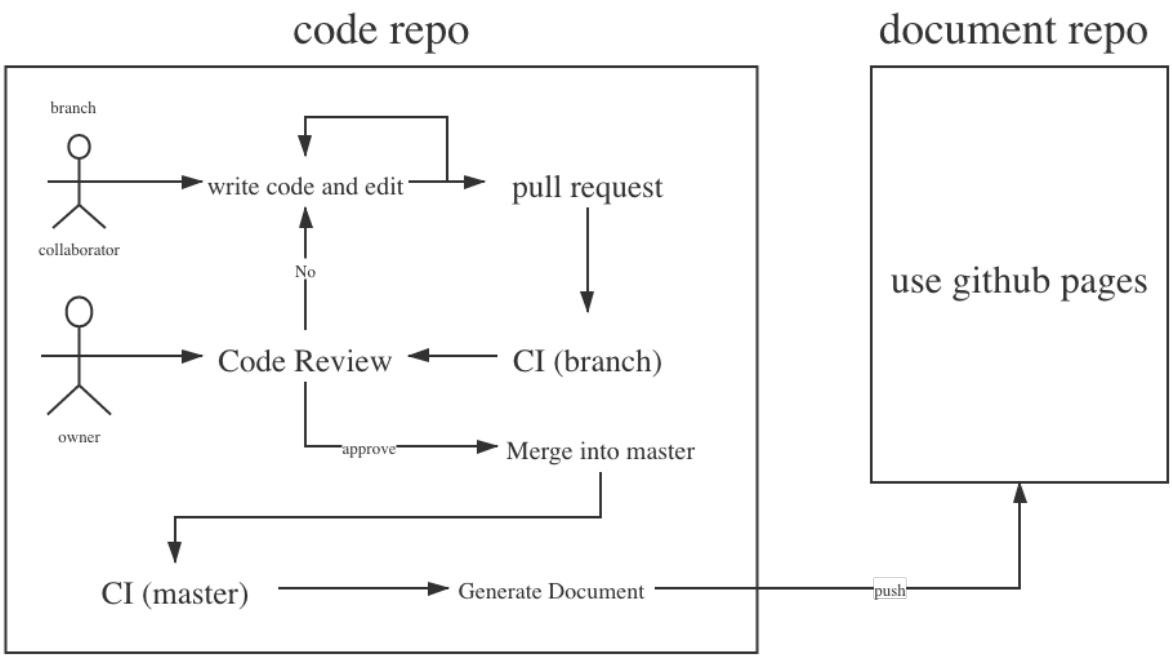
CI Platform

- Travis
    - Travis is good for most open source projects, but is not free for private projects.
- Github action (preferred)
    - github action is a new CI tool that is provided by Github, excellent suitable for Github project. For private repository, Github Action is free. It can satisfy all our requirements.
- Jenkins
    - jenkins is a good CI tool for most commercial products, but it need to be deployed on our own machine.

CI Design

Overview

We need two CI tracks, one for master, and one for other branches.



master

Based on the assumption that we have a strict control of master, any code merged into master can be regared as releasable. So in this part, we can focus on **deployment**.

Any merge or push (basically we don't encourge to push code directly into master) can trigger a CI, with the following steps:

1. **Auto-generate document**: For this part, the details can be completed after investigation by @Guangtao Zeng.
2. **Auto-deploy website**:

branch

Because any collaborator can create its own branch and pull request for merging into master, we have to make sure their code is formatted and tested. Consequently, in this part, we mainly focus on testing.

Because Ci in this part may provide key information for reviewer, strict code check is necessray.

1. **commit message check**: In order to let others, especially your code reviewer to know whay exactly you have done in this commit, collaborator should write clear commit message by observing some rules(Commit Message Rules)
2. **code format**: Code formatting is vital to improve the readability of our code. For Python, we usually use Pylint, Flake8, ISort, Yapf.

3. **unit test**: use pytest. Unit test can cover the basic logic of most operation.
4. **test coverage**: use coverage to genereate test report. And then we can upload this report to Codecov.
5. **API test**: API test may test our recommendation model thoroughly. For a good recommendation framework, we should test our model on different platform. This part may cost much time and resources. It is unlikely to be done on the CI machines. If we have other machine (with gpu), then we can test our model.

Reference

1. pylint
2. flake8
3. isort
4. yapf
5. pytest
6. coverage
7. codecov
8. travis
9. github action