What I learned from connecting 60 projects to CI system

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:46:51Z

Several months ago I started to wonder what makes me check out software sitting at GitHub and try it out. Some projects I was more willing to work with, and some I simply skipped without thinking about it much. It seemed like a habit which I've developed over the years. But why do I have this feeling of carelessly ignoring some projects over another?

The characteristics of projects I'm drawn into are pretty simple: good, clear documentation, good top-level directory structure, presence of some "dotfiles" and … green badge, signalizing Continuous Integration (CI) system is actually verifying this software is at least building fine.

Once I've reached this conclusion, I set myself on a path for bringing all my essential repositories to the Continuous Integration (CI) system. First, to help people with my software, and second, to learn more about CI systems.

I had many repositories, big and small, old and new, some of which weren't in the best state. Below you have my thoughts and conclusions from the project.

Continuous Integration 101

Quick introduction: CI system is a remote computer wired to your repository. When the repository changes, it'll pick a change and execute the steps which user normally would have taken to build your software. By being a 'virtual user' the CI system finds issues with reproducibility which customers of your software would normally find. And it relieves you from "boring but necessary" kind of work. By establishing a process and culture around CI system, you're more likely to deliver software with a decent quality.

Project's target

So the idea I had was to basically let people know what my repos are all about by bringing documentation. And then showing that code isn't total crap by building it in a reproducible way.

So my algorithm for fixing my code was this:

for each repo: regardless of what (no exceptions) put README.md in place if it's super trivial and I don't care: skip contigure CI system and see a green badge

Continuous Integration solutions?

I've worked with 3 CI systems:

- Travis CI,
- Circle CI
- AppVeyor.

First of all, it's pretty eye-opening that people in these companies contribute their compute time to the open-source community. I'm not affiliated with any of these companies, yet I send them big kudos for doing so. So far on delivering my open source projects I have paid literally $0 (zero dollars) for their services. It is great companies contribute that way to the open-source and I hope that through this article we will have more developers using their products and improving quality of software.

How they differ?

Travis-Ci and Circle-Ci are direct competitors and their offering is similar. I have found Travis slightly easier to configure and its UI nicer. Circle-Ci feels slightly faster however. This is not 1-to-1 comparison, and I feel like the best way is to try yourself. Travis has a big advantage that was important to me: they offer free OSX support. Since I had some projects for OSX, it was important to me. CircleCi and AppVeyor have this nice feature of letting you to login on the worker machine for debugging, something I wish Travis had. This is very important for debugging (read on).

AppVeyor however is pretty simple to choose: it's a dominant Windows-based CI system. Because FLViz and KMNSIM both worked on Windows, I wanted to see them building fine too.

Lessons learned

Being a code piper isn't always fun

Debugging CI system can be boring and tedious. Reward is there of course, but you'll have to arm yourself in patience and prepare for debugging remote system which you have no direct access to.

Most of failures will be of sort: “Works for me, but doesn’t work on remote system”, which is precisely a point of having CI system.

> Whatever problem you rule out at this stage, your customers won’t hit.

Debugging through a key-hole

Remote systems behave differently and can be configured in unfamiliar way. I tried to ease this issue by showing some CI settings through CI-env. By clicking individual badges of each system you can see its remote settings, which you can encompass in your scripts later.

Trial and error

Travis-CI offers a Docker Image for debugging. You may find it helpful for debugging your flow. For me it’s often a cycle of:

- push a commit
- see CI system failing
- change something and push a fix
- see CI system failing
- etc..

Great flexibility

For sure I’ve done with CI systems more than I’d have normally do. For example testing my Ruby versions for your scripts is easy. So is building on 2 operating systems

Things I don’t want to heat my Mac with, such as building Docker images I also do through CI system.

Keeping things simple

What I see in some projects I contributed to is that we put a lot of stuff in CI configuration. I think it’s a mistake because users can’t actually execute the CI steps from the same configuration file. So I suggest you wrap most of your steps in a script or a makefile and then run it from a simple config

Summary

It will take time to push the quality of your code higher, but the benefits are great. Right now each time I make a change to my projects, I can see immediate checks on whether I broke something or not, which is very nice. I urge you to pick 2-3 of your active projects and do the same.

Let me know if this article was helpful and whether you’d find a screencast documenting topics in this article useful