Treat Continuous Integration as your virtual user

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:46:50Z

Ninety five percent of professional software engineers use Jenkins as a Continuous Integration. The rest are consumers of Travis-Ci, Circle CI and many other hosted CI platforms. None of these platforms are opinionated. They are just a little smarter replacements of `cron`, allowing you to simply throw many commands into the flow to build your software. We use these tools in a dump way, I argue, and it wastes the time of customers of our software.

My view is that you should use Continuous Integration server as your virtual user. Your testing buddy. He watches over your arm and sees you pushing new changes to the repository. The moment you finish, he goes and fetches all your stuff tirelessly. And the flow of this virtual user should be *the same* as your normal user.

Problems with existing Continuous Integration platforms

All CI solutions **make it hard** to structure your CI jobs as a virtual users. Because they are flexible, they let us stick many commands and flows into their configuration. In the case of Jenkins it's `Jenkinsfile`, in case of Travis-Ci it's `travis.yml` etc.

Sometimes I see people using two systems for testing. Instead of converging the configurations, they develop them independently. So you'll see many, many steps in `travis.yml` and `cicle.yml` that are fairly **different**. That's a mistake. *Don't do this*

The issue I have with these solutions is that once you put your logic there, you've just made your flow better for nobody else, but yourself. Neither Jenkins nor Travis-Ci flows can be reproduced easily. Users can't replay these files, so all the knowledge you stick there is useless, or hard to use at least.

Alternative

Alternative solution is simple. Follow a Single Command Principle. Automate your build, test and installation stages through a script, and then just use it on your CI server. **The very same script**

At most, your CI job should have 1–3 commands. Again I know it's hard, since things like shipping secrets and SSH keys to your CI server may be very specific to the platform you use. I get that. But remember that everything what you do in CI server configuration can be done in a normal script, and I'd rather see that. Make it adjust accordingly depending on whether it's a Travis-CI builder, or an unknown host.

This way the code fresh out of the repository can be both continually tested, but also **delivered** to the user – a thing that we really care about.

Summary

I would like to see your Continuous Integration best practices and how you handle managing them in your environment.