How the best companies do Continuous Integration

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:47:41Z

Nothing but the successful result of compilation delivers this unique feeling of accomplishment. There are multiple stages of project's success, yet each feedback on a positive program build is exciting, since it is just one step from actual program execution–the ultimate goal.

Yet when done too frequently, building becomes a problem. Especially when projects grow, you become dependent on many other libraries. Maybe even whole projects. Building simple application nowadays requires inclusion of many 3rd party modules, and this process is not always easy. And this is how building becomes problematic, since it starts to be slow and boring. Feedback loop, which usually is really tight for small programs becomes really long and slow; in the process you start getting distracted, and cheat yourself that you can multitask, but you really can't.

In software engineering there's this practice called "Continuous Integration" in which you have your computer do this heavy-lifting for you. It requires "automating everything" in 1-step, so that upon making a change to the source code, you can have the computer do the boring stuff. It should not only build the program, but also test it. Some houses go even further and introduce "Continuous delivery". Basically if your tests pass, the outcome is getting deployed to your environment automatically.

Setting up the whole environment for building your software, tracking progress and letting you know how each build went is a lot of work, yet it pays off. Why? Well, in the last several months I've been working with several technologies, including C, Python, Ruby and Javascript. All of them are fairly old citizens of computer world, yet working with them isn't as straightforward as one may seem.

Engineering build for each project still seems to be a custom task. Wiring your code with libraries might be easier with tools such us Biicode might be easier, but its repository is limited. Making Python use `pip` is no longer as straightforward. Why? Well, Python 2 or 3 I ask? So we have `virtualenv` right now even for less sophisticated programs. And Ruby? Rails? Which one – 3 or 4? And this is how you end up with RVM.

So a build and testing automation completely makes sense.

While I glance at Travis and Jenkins and Circle Ci documentation and comparing them, below are the links on how big guys do it. If you have any other ones, ping me, and I'll add them to the list together with a credit to a contributor. Learning more would help me figure out in my head on what the right pattern for implementing CI really is. Doing it for couple of toy projects GitHub is probably not worth investing huge amount of time for configuration of tools and infrastructure, but assuming necessary shortcuts, I think it should be possible to get yourself a nice system for software pleasure and comfort.

Most valuable resources

Facebook

- Release Engineering at Facebook
- Scaling Mercurial at Facebook

Google

- 10 Commandments of Release Engineering
- Release Engineering, Google Inc.

Netflix

- Continuous Delivery at Netflix
- Netflix Development Patterns for Scale, Performance & Availability

IMVU

- Continuous Deployment at IMVU: Doing the impossible fifty times a day.

Other interesting topics

- Continuous Integration and Deployment Best Practices on AWS
- How Netflix's Tools Can Help Accelerate Your Start-up

Books on the topic

Haven't had a chance to allocate time to get and read those, but judging by the reviews, they seem pretty decent:

Anything I missed?

Definitely contact me.