CLOUDADMIN

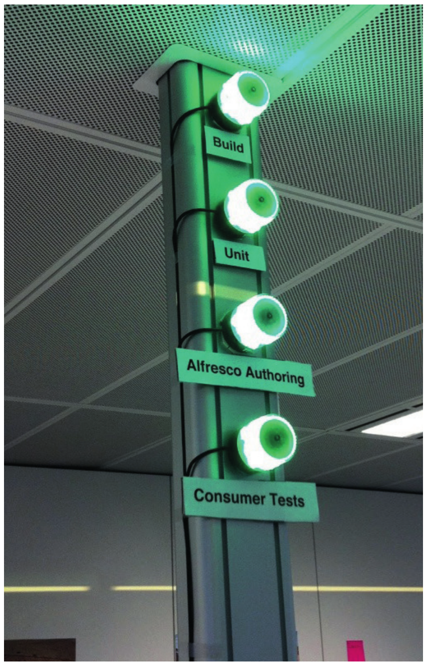
Nick Veitch opens your eyes to the technology behind the cloud server revolution.

Jenkins

Testing code yourself? Why not get a minion to do that for you?

Wouldn't it be great if things worked all the time? I mean, software things, not people things. Gosh, the world would be a better place if I worked hardly at all. That's why software was invented isn't it?

But things (including me) do not work all the time. In fact, in the world of software development, particularly in the new world we now call devops, things are almost perpetually operating in a mode of non-functionality. What is very useful to know is when, almost miraculously, there is a small window of time when software works as it is expected to. In a smart, switched-on world, we can capture this moment so we can work out what went wrong/right. This is the world of continuous integration.



One of the Jenkins plugins can automatically control lights to demonstrate build status (picture by Dushan Hanuska).

The name sounds like some sort of maths purgatory. Continuous integration was born out of the age that gave us agile programming. The idea is pretty good, and all but indispensable in the modern world of group coding on complex software properties. Commits made by coder #1 may be all well and good, and coder #2 is well known for being cautious and testing everything, but if they both land changes, perhaps the two new parts don't interact as they should?

Breaking code is usually ridiculously easy, even with parts that seem to be backwardly compatible – it is often the difference between the programmers' perception of how a particular block of code works, and how it actually works, that leads to ripples of wrongness in the edge cases.

Testing is therefore a good thing, and continuous integration is a blessing.

At your services

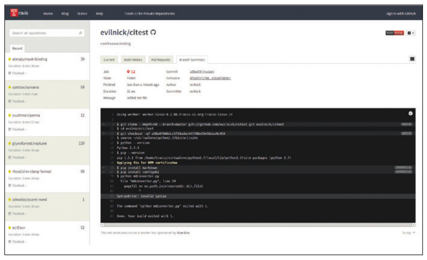
In the early days of Agile, one system rose head and shoulders above the others. Its name was Hudson. Hudson was originally a Sun project, but with the acquisition of Sun assets by Oracle (remember them), there were, erm, tensions. The project decided to rename itself 'Jenkins'. Oracle went off in a huff and took its newly trademarked name with it. Hudson still exists, but we'll leave you to decide who came off best from that spat. Anyhow, the open source Jenkins project became the *de facto* required tool for those seeking CI nirvana.

Jenkins itself is a behemothic Java app (aren't they all) which requires a server to run on. The basic principle is that you submit a job (maybe triggered automatically by time, or by a merge to a public repository) of code. Jenkins then spins up a clean environment for the code to work in, executes your tests and spits out a report. Actually, that is pretty much the least it will do. With an active and productive community, Jenkins has grown a legion of plugins to automate everything from audit trails to uploaders for zubheim (it's a mobile app testing platform). There is a list of plugins so long I couldn't possibly read them all without more gin than is medically advisable.

The only major downside to Jenkins is that it requires a bit of effort to get it working the way you want. Though it was originally imagined as a Java testing tool, Jenkins now covers an impressive range of languages and additional functionality, but almost all of these require adding plugins and some amount of configuration. The results of a

If you are bored with butlers, you could always replace the Jenkins logo with Chuck Norris.

"Though it was originally imagined as a Java testing tool, Jenkins now covers an impressive range of languages and additional functionality."



Log on to the Travis website and you can see the build queue and check out the console output from your jobs. If only to find out what an idiot you are.

well configured, well honed system are very worth it though (you should definitely try the Chuck Norris plugin, which replaces the butler image on your reports with a random image of a culturally significant martial arts practitioner). As well as merely testing the build and functionality of your software, you can automate all sorts of other tasks like publishing documentation or building different versions. If you ever have the need to build hierarchical projects, there is really nowhere else to start.

Did anyone prophesise these people?

Due to the nature of its flexibility and all the gubbins that can be bolted onto it, you may find it takes quite a while to set Jenkins up in a working fashion. For people who don't want to spend half their working day writing new configs for Jenkins, there is Travis.

Travis is quite a bit simpler, but no less capable. Part of this simplicity is due to the fact that you don't have to set up a server to run Travis for you; it is a hosted service. Actually, you can very much download the open source code for Travis from github, but why go to the trouble of building your own service (the docs don't really give you any help on this either) when the hosted service is free for open source projects. Travis integrates really well with github. In fact, that's its primary purpose. And instead of editing acres of config files, you can set up everything that travis can do in a simple file called **travis.yml** and add it to your github repository. Register with the Travis website (sign in with your github credentials, flick a few switches to turn your repositories on and Travis will do the rest. Every time you push updates to the repository, Travis will notice and add your jobs to the end of the queue. Log in to the website and you can check the queue and see the console output of your jobs when they run. But there's no need, because Travis will send you plenty of mails as well, telling you when builds have succeeded or failed, and helpfully giving you plenty of version info.

If you really want to know how easy it is to configure Travis, here are a few examples; firstly, a simple python project:

```
language: python
# versions of python to use
python:
 - "3.3"
 - "2.7"
 - "2.6"
# command to install dependencies, e.g. pip install -r
requirements.txt --use-mirrors
install:
 - "pip install markdown"
 - "pip install configobj"
# command to run tests, e.g. python setup.py test
script:  python mdconverter.py
```

This example declares the language, optionally selects a few different versions (they will execute as separate jobs) and has the **install** command to add any dependencies. The **pip** tool is used in this case to install any additional modules required by the Python file.

The final line is the script to run. In this case, as our project is a simple linear conversion tool, we just execute that, but you could run a separate script to do a number of tests, pass special parameters or anything. For a C project, the Travis file can be as simple as:

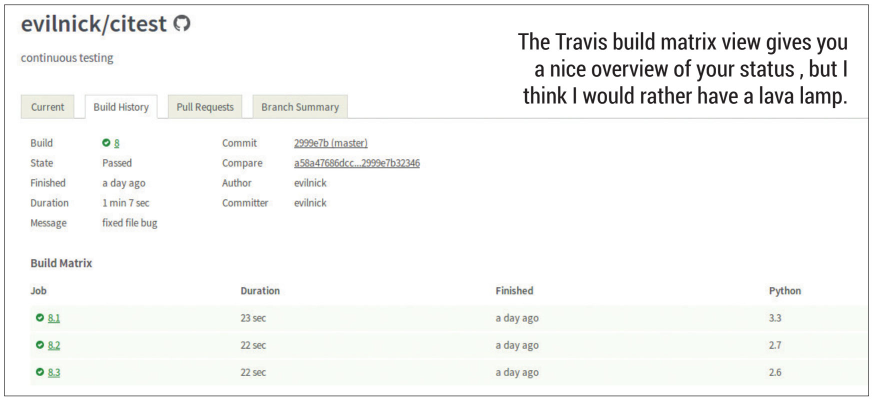
```
language: c
```

That's because Travis assumes you are using Automake and will default to running **./configure && make && make test**, so you can put any magic in your Makefile. There is no real dependency management for C projects, but that doesn't stop you from using the **install** section (or **before_install**:, to run first) to fetch such things as you might need. You are also not limited to using GNU **make**. For example, to use **cmake** you would just need to add something like :

```
before_script:
 - mkdir build
 - cd build
 - cmake ..
script: make
```

Currently the Travis CI environment is an image of 64-bit Ubuntu 12.04, so you can do pretty much whatever you need to with it.

If you have an open source project on github and want a free, no fuss service, Travis could fit the bill. Jenkins is better at all-round automation – if you need to do more than just build a project and run a script, it is infinitely more powerful (with the configuration headaches to match). But whatever you choose, make sure you write good tests! Continuous Integration as a discipline means nothing if your tests don't make sure that things are working. Good tests don't end at "well, it compiled OK", but actually running meaningful tests with the running code itself. Your code (and users) will thank you for it.



evilnick/citest

continuous testing

Current | Build History | Pull Requests | Branch Summary

Build	8	Commit	2999e7b (master)
State	Passed	Compare	a58a47686dcc...2999e7b32346
Finished	a day ago	Author	evilnick
Duration	1 min 7 sec	Committer	evilnick
Message	fixed file bug		

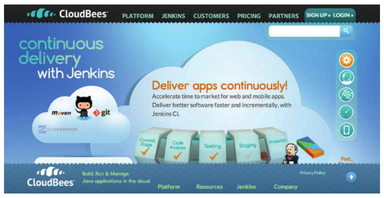
Build Matrix

Job	Duration	Finished	Python
8.1	23 sec	a day ago	3.3
8.2	22 sec	a day ago	2.7
8.3	22 sec	a day ago	2.6

The Travis build matrix view gives you a nice overview of your status , but I think I would rather have a lava lamp.

CloudBees

Cloudbees is a hosted service that offers free Jenkins capabilities, so you can get pretty much everything you ever wanted out of Jenkins, but without having to go through the rigmarole of running your own service. The cloudbees people should know what they are doing too – the CTO is Kohsuke Kawaguchi, the original founder of the Jenkins project.



All the Jenkins service but without the trouble of having one of your own.