martinfowler.com

BlueGreenDeployment

Martin Fowler 1 March 2010

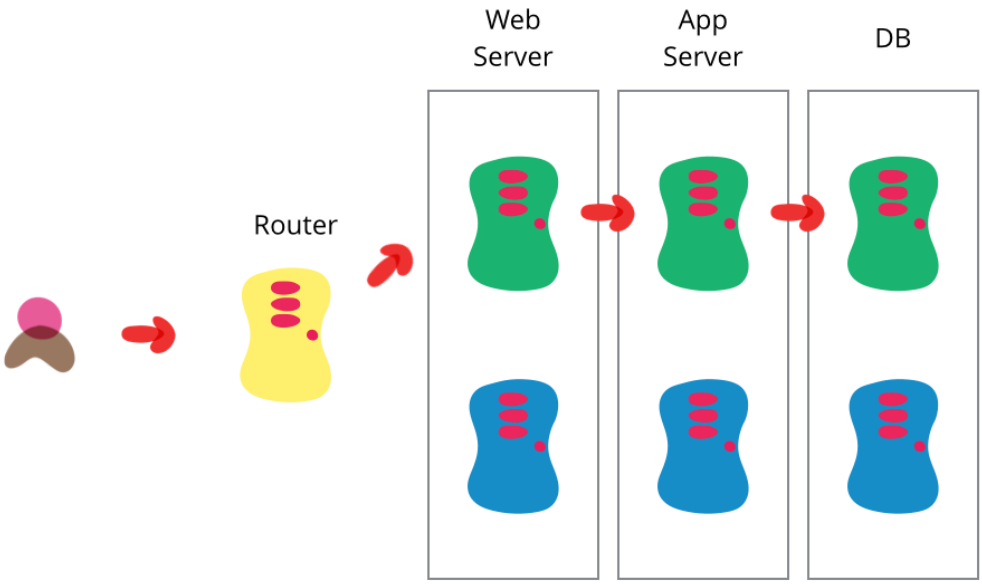
5-6 minutes

---

continuous delivery

tags:

One of the goals that my colleagues and I urge on our clients is that of a completely automated deployment process. Automating your deployment helps reduce the frictions and delays that crop up in between getting the software "done" and getting it to realize its value. Dave Farley and Jez Humble are finishing up a book on this topic - Continuous Delivery. It builds upon many of the ideas that are commonly associated with Continuous Integration, driving more towards this ability to rapidly put software into production and get it doing something. Their section on blue-green deployment caught my eye as one of those techniques that's underused, so I thought I'd give a brief overview of it here.



One of the challenges with automating deployment is the cut-over itself, taking software from the final stage of testing to live production. You usually need to do this quickly in order to minimize downtime. The blue-green deployment approach does this by ensuring you have two production environments, as identical as possible. At any time one of them, let's say blue for the example, is live. As you prepare a new release of your software you do your final stage of testing in the green environment. Once the software is working in the green environment, you switch the router so that all incoming requests go to the green environment - the blue one is now idle.

Blue-green deployment also gives you a rapid way to rollback - if anything goes wrong you switch the router back to your blue environment. There's still the issue of dealing with missed transactions while the green environment was live, but depending on your design you may be able to feed transactions to both environments in such a way as to keep

the blue environment as a backup when the green is live. Or you may be able to put the application in read-only mode before cut-over, run it for a while in read-only mode, and then switch it to read-write mode. That may be enough to flush out many outstanding issues.

The two environments need to be different but as identical as possible. In some situations they can be different pieces of hardware, or they can be different virtual machines running on the same (or different) hardware. They can also be a single operating environment partitioned into separate zones with separate IP addresses for the two slices.

Once you've put your green environment live and you're happy with its stability, you then use the blue environment as your staging environment for the final testing step for your next deployment. When you are ready for your next release, you switch from green to blue in the same way that you did from blue to green earlier. That way both green and blue environments are regularly cycling between live, previous version (for rollback) and staging the next version.

An advantage of this approach is that it's the same basic mechanism as you need to get a hot-standby working. Hence this allows you to test your disaster-recovery procedure on every release. (I hope that you release more frequently than you have a disaster.)

The fundamental idea is to have two easily switchable environments to switch between, there are plenty of ways to vary the details. One project did the switch by bouncing the web server rather than working on the router. Another

variation would be to use the same database, making the blue-green switches for web and domain layers.

Databases can often be a challenge with this technique, particularly when you need to change the schema to support a new version of the software. The trick is to separate the deployment of schema changes from application upgrades. So first apply a database refactoring to change the schema to support both the new and old version of the application, deploy that, check everything is working fine so you have a rollback point, then deploy the new version of the application. (And when the upgrade has bedded down remove the database support for the old version.)

This technique has been "out there" for ages, but I don't see it used as often as it should be. Some foggy combination of Dan North and Jez Humble came up with the name.

Acknowledgements

Illustration by Ketan Padegaonkar

Updates

2015-06-05: The page got a lot more page traffic today as Hacker News noticed it. So I added a paragraph on database changes and the further reading section.