Exploring New Frontiers in CI/CD and DevOps

VERSION 1.0 / MARCH 2018

Introduction

Ten years ago, the idea of DevOps sprang from the minds of Andrew Shafer and Patrick Debois. A year later they actually named it “DevOps”. This tenth anniversary marks a good time to look at where DevOps has taken us as an industry and where we are heading.

This whitepaper is not about vendors, commercial, open source or otherwise. At least not exclusively. This whitepaper looks back at the history of DevOps and CI/CD practices and researched what patterns have evolved over the years and what patterns we see coming over the horizon.

An Agile birth

At the Agile 2008 conference in Toronto, Andrew Shafer gave a talk that became the seed for the whole DevOps movement. This is now “ancient” history in the tech world but it’s worth reminding ourselves that DevOps is an idea that was sprouted by, or at least in the context of, the Agile movement.

Now, the 2001 Agile Manifesto is light on specifics but it *is* pretty clear on…

1. “continuous delivery of quality software”
2. “delivering working software frequently”
3. “harnessing change … as a competitive advantage”

It is not a surprise then that DevOps and the practice of CI/CD struck a chord with the Agile movement.

From manual to codification

The principal idea that caught the imagination of almost everyone in IT and related professions was the idea of codifying not just business logic, but also the supporting infrastructure around it.

- **Continuous integration** is in essence a codified manual test and code management process.
- **Infrastructure as code** and cloud resources with an API are codified manual installation and configuration processes.
- **Continuous delivery** and deployment are in essence codified manual release activities.

Without codification, the other big promises of DevOps and Agile are very hard to achieve. Codification has two main applications:

Reduced lead times. Testing an application, setting up a server or installing an application can be very time consuming activities. Abstracting the process into code and re-using that code is almost always a net gain in productivity and a drop in lead times.

Risk aversion. Even if speed is not your main goal, codifying helps address the inherent danger of "changing things" and amps up the predictability of releasing quality software.

- **Automated testing** catches bugs earlier and faster.
- **Infrastructure as code** prevents configuration drift.
- **Automated rollbacks** provide a safety valve for botched releases.
- All of the above are less prone to human error when automated.

Not a goal, but certainly a huge side benefit is that code creates a shared language, platform and understanding between developers and sys admins.

From "codifying all the things" sprang forth an entire industry of products and services, hosted, SaaS, open source, commercial and everything in between: Heroku, Cloud Foundry, AWS Beanstalk ,TravisCI, Jenkins, CodeShip, Bamboo, Puppet, Ansible, Terraform. The list goes on and on.

From codification to experimentation

The first wave of DevOps practices ticked the "continuous", "frequent" and "working software" boxes of the Agile manifesto. However, it turns out that that other box, "harnessing change as a competitive advantage", is a much harder to achieve. What changes *are* actually to our advantage? How do we know when they are not? What do customers actually want?

These types of questions have always been the domain of marketing, UX and product management. A/B and multi-variate testing, test audiences and short-lived experimental features are typical weapons found in the field. The base assumption is "we don't know the answers. Let's experiment and find out."

This experimentation mindset is now making its way into the domain of the back end, where software engineers, architects and sys admins are by nature more predictors than adapters. In general, backend engineering deals with unknowns through process like specification and estimation. These are however mostly crutches because a true experimentation practice does not exist due to non-available tooling and cultural baggage: it is exactly that which is changing.

Experimentation patterns, tools and services

Over the last few years many patterns, techniques, tools and services have popped up to assist engineers in turning "I don't know" into "I have learned. I know now".

1. Blue / green deployments

The granddaddy of them all, and by far the most crude but wide spread pattern is the blue/green deployment. Arguably, many engineers implemented this pattern long before DevOps saw the light of day.

A typical blue/green deployment scenario involves two identical environments and some form of switch to route traffic to either environment. Come release time, you deploy to one environment and direct traffic at it. You keep the other environment for fallback in case things go south.

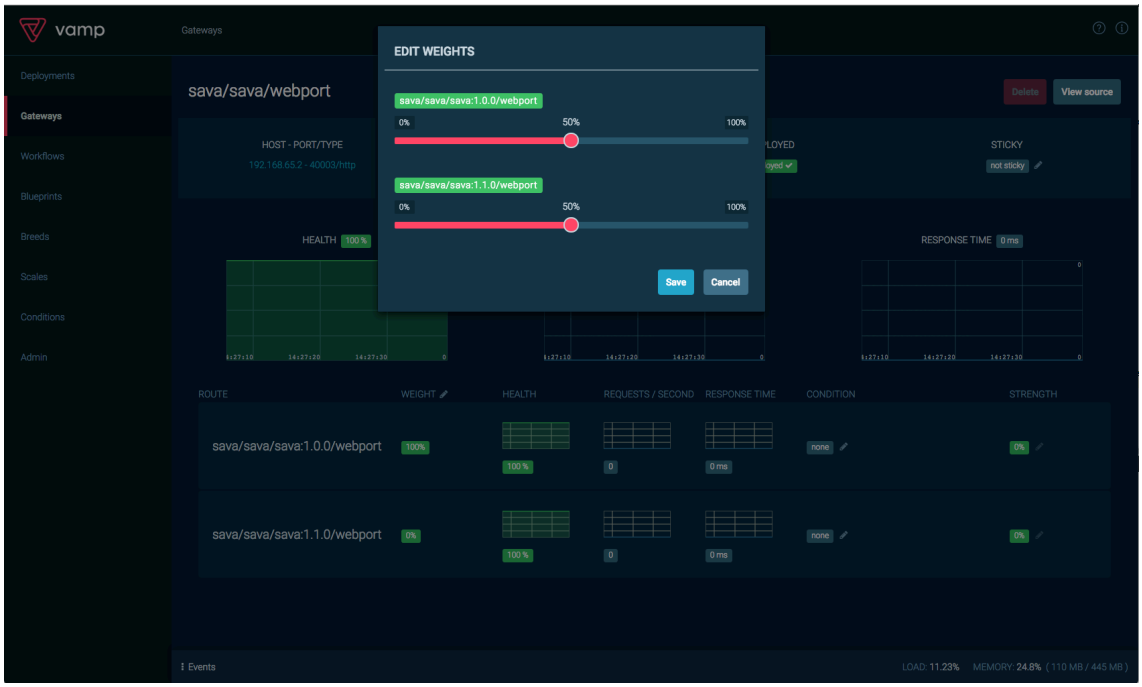
2. Canary releasing

A lot less crude than blue/green deployment is canary releasing. The principle is the same, but the granularity is a lot higher. With canary releasing you can actually start running proper experiments without putting your production environment at risk.

Your routing component needs to be “smart” to do canary releasing because you want to target only a sliver of your total traffic and direct that traffic to the new version of your app.

Canary releasing is still very much a pattern. The implementation and execution is left as an exercise to the reader. There are however products that already offer this as a first class citizen.

Vamp allows the operator to use a simple slider to determine traffic percentages. In a more advanced scenario, Vamp provides targeting on many different traffic aspects like user agents, devices, geo location.



3. A/B testing

A/B testing is canary releasing with a statistical framework around it. By and large it offers similar targeting options as canary releasing but it adds goals to the mix. This helps determining if and how much a different or new version of your app contributes to a specific goal.

A/B testing is offered by vendors like Optimizely, Visual Website Optimizer (VWO), Google Optimize and is implemented as feature in many other services like Unbounce, Mailchimp and Kissmetrics. Companies also build it themselves, like Pinterest and Booking.com

However, all of these services and implementations work strictly on the front end (web pages, email etc.). This means you can mostly test visual and textual variations of an existing application. What if you need to experiment on other parts of your application stack? Product and services are appearing in this arena.

Optimizely X Full Stack is a full service solution, covering many languages and run times. In the open source world we have Planout by Facebook, Wasabi by Intuit and Petri by Wix.

4. Feature toggling

Feature toggling is more of a parallel pattern. It integrates aspects of canary releasing and A/B testing into a pattern focused on toggles. These toggles turn application features on and off. Either in general or for target audiences.

Feature toggling's main virtue is that you can decouple the moment of deployment from the moment of release. Services offering feature toggling are for instance LaunchDarkly and Split.io Open source alternative are Togglz for the Java platform and Flipper for Ruby.

5. Shadow traffic

Also known as "dark traffic", shadow traffic is the practice of syphoning of traffic from the production flow and exposing it to an experimental version of your app, without the end user being directly involved.

This pattern is firmly rooted in the back end and can be used for load & performance testing or as a general sanity check. Implementation at this moment seems to be completely custom, as of the time of writing I could not find any SaaS service that offers shadow traffic. Open source, middleware-type projects like Istio (for the Kubernetes stack) and GoReplay are relatively new.

All implementations need to overcome some thorny issues with how network traffic works at the lowest level with regard to client responses, encryption etc. However, the potential specifically as part of an integrated platform like Kubernetes is high.

Towards an experimentation platform

As experimentation patterns are adopted across the full IT stack, the granularity of experiments and scale at which they are deployed rise. This gives way to a whole set of new problems.

- How to deal with the data output of the experiments? Not every company has a dedicated data science team.
- How to match the infrastructure to the experiments need? Successful experiments might need more/new/different resources quickly.
- How to do all of this at scale and keep your sanity?

Many of these problems are automation problems and fall within the scope of DevOps practices. We can see the first products and services appearing that aim to tackle these issues.

1. Machine learning & AI

Using machine learning and artificial intelligence to make sense of the deluge of logs and metrics a typical modern IT stack generates is still in its infancy, but looks really promising.

Pattern recognition, filtering and predictive analysis are all well understood mathematical problems. We can see these techniques appear in DevOps oriented products and services like Signifai.io and Logz.io. Both services promises to cut down the noise inherent in data streams and help teams focus on finding the nuggets of gold buried among the scraps.

ElasticSearch, a popular open source search engine used by many to analyse log data, just added Machine Learning to its feature set. This gives users a relatively easy path into using machine learning to grog their data and make predictions into the future.

Leveraging ML and AI to assist in decision making with regard to deployments, scaling, feature toggling and running experiments in general is still largely unexplored terrain.

2. Smart orchestration

Modern container platforms like Kubernetes, Docker Swarm and Mesosphere's DC/OS fit neatly into the new and emerging patterns of DevOps. Dubbed "Cloud Native", these platform provide portability, scalability and rapid, standardized deployment patterns.

In this context, they are the ideal base infrastructure for running an experiment based IT infrastructure as they tick almost all of the boxes needed for such a system:

- Codified and standardised run times, network and storage through (Docker) containers and related tech.
- Codified deployment process through the platform's API.
- Smart routing through overlay networks, integrated API gateways and service discovery / meshes.

The move to Cloud Native infrastructures is already happening and already we can see new applications enabled by the cloud native paradigm appear.

Products like Vamp take smart routing to a level where routing decisions become real time malleable application attributes. Coupled with a monitoring solution, smart routing solutions are becoming the back bone of A/B testing and shadow traffic patterns implemented on DC/OS and Kubernetes.

Bin packing is a technique where many application processes share a physical host right up to its capacity as to not let unused capacity go to waste. Both Kubernetes and DC/OS already do this.

Combining this with AWS Spot Instances takes it to another level of cost efficiency. Interesting advances can be made here by tying that cost level to higher level business goals set, for example, in an A/B test.

Many companies have internal cost accounting and attribution to structure their IT cost. This could tremendously help in making the business case for an experimental application or service.

For sales enquiries, please contact:
Diederick van der Heijden // Business Development Manager
+31 657 337 442
diederick@vamp.io