INDUSTRY INSIGHTS (/blog/category/industry-insights/)

By Justin Baker (/blog/author/justinucd/) • May 9, 2016

7 min read

Decoupling feature rollout from code deployment and the rise of user-centered deployments

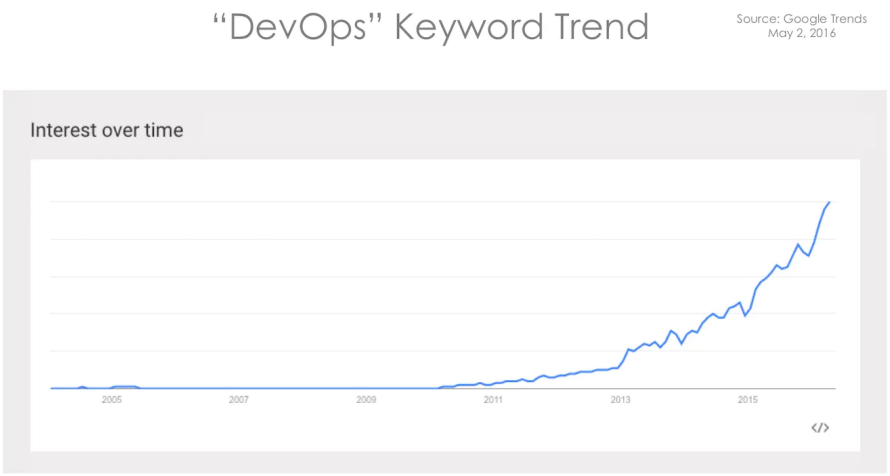
The Origin of DevOps

In 2008, Patrick Debois laid the foundations (http://rewrite.ca.com/us/articles/devops/a-short-history-of-devops.html) for DevOps at an Agile conference in Toronto. He was trying to come up with a solution for the inherent conflicts between developers and system admins. Both disciplines seemed to be at odds: developers wanted to release software more frequently, but system admins

wanted to ensure stability, performance, and scalability. While this conflict isn't necessarily black and white, it highlighted the need for developers and system admins to no longer consider themselves as mutually exclusive roles, but rather as cross-functional partners.

A year later, Paul Hammond and John Allspaw gave a talk (http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr) at the Velocity '09 conference that highlighted the necessity for cooperation between Dev and Ops. This inspired Debois to coin the term "DevOps" (#DevOps), which quickly picked up momentum (and a fair share of controversy).

Now, with the proliferation of DevOpsDays (http://www.devopsdays.org/) meetups and DevOps communities around the world, the movement has accelerated.

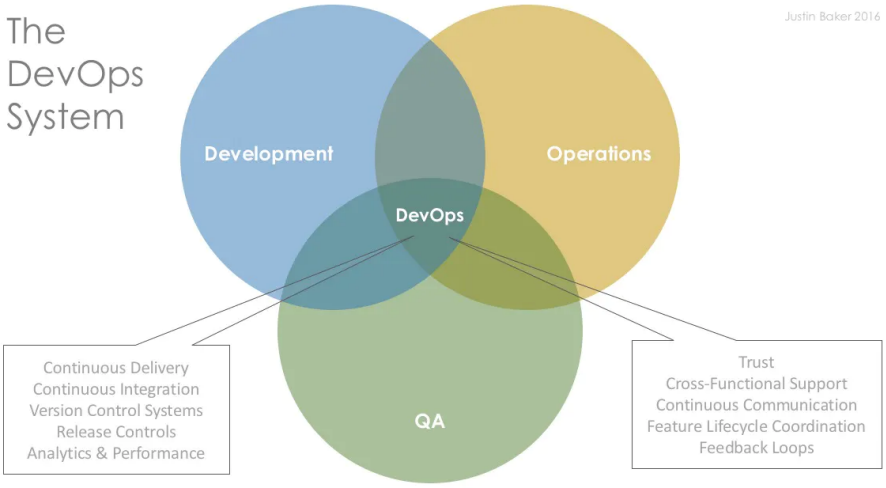


DevOps 1.0

We all know that labeling something with version 1.0, 2.0, 3.0... makes our eyes roll. In this case, however, we can frame DevOps 1.0 as the birth and emergence of a nuanced movement. This nuance is in establishing its initial identity: a way to bridge the gap between development and operations with a visionary goal to accelerate continuous delivery and deployment (https://www.thoughtworks.com/continuous-delivery).

Broadly speaking, the DevOps movement centers around the philosophy that building, testing, and releasing software should happen continuously and incrementally. The age of waterfall releases is slowly winding down and most progressive development teams are embracing continuous delivery (https://blog.newrelic.com/2014/10/16/devops-continuous-delivery/) as the de facto development method.

In terms of a disciplinary definition, DevOps is at the intersection of development, QA, and operations. It has become an amalgamation of soft and hard skills: trust, cross-functional teams, DVCS, and robust QA/continuous delivery systems.



As we see, DevOps has become a systemic term that treats software delivery as an organic and fluid process. Our application is now treated as a living entity, where we harness a suite of services to ensure that our software infrastructure can adapt to change, iterate quickly, and proactively mitigate risk.

We have now embraced the notion that software does not need to be perfect when it is first launched. We do not need to embrace a monolithic release and hope that our months of planning, design, and operations development have led us on the right path. Hence, the DevOps movement was founded on the notion that the best software is adaptive, not reactive. To make adaptive software, we are now embracing systems that are built for change.

With consumers now demanding the expedient delivery of near-perfect software and with a global tech boom, how do software companies stay competitive? How do companies leverage DevOps to stay lean, adapt to change, maintain stability, and deliver software faster? This question leads us to emergence of DevOps 2.0: extending the feedback loop benefits of dev and ops to the entire organization: marketing, sales, product, dev, and ops.

DevOps 2.0

DevOps 1.0 has been heavily centered around harmonizing the interplay of development, QA, and operations. The primary goal has been to institutionalize continuous delivery, while also creating more flexible and stable application infrastructure. With DevOps 2.0, we see the emergence of adaptive feature delivery as a critical component for successful software releases.

This need for adaptation is both internal and external. Internally, teams must adopt cross-functional methods to ensure that software is:

- Iterated with a continuous cadence
- Scaled quickly while maintaining system stability
- Complementary to marketing and sales campaigns

Externally, teams must respond to a customer market that is in a continuous state of flux. Because customer culture shifts so rapidly, customers' expectations now change from month-to-month instead of year-to-year.

Team Coordination

For organizations, DevOps 2.0 brings the power of DevOps to non-technical team members. While this may sound risky, it actually empowers marketing, design, and business teams to control targeted visibility and testing without consuming engineering resources. Because feature rollout will be decoupled from code deployment, non-technical team members would be able to control the visibility of particular features without compromising the app's integrity. This is primarily achieved by harnessing a feature flag user interface (https://blog.launchdarkly.com/enterprise-requirements-for-managing-feature-flags/) – or a comparable control panel that allows team members to target users via a GUI.

Therefore, the central challenge for DevOps 2.0 is harnessing the appropriate tools to ensure that software deployments are responsive to consumer demands. In the last few years, a number of DevOps tools have emerged to help facilitate coordination between previously disparate disciplines.

DevOps Tools

DevOps has evolved into a breadth of coordinated development processes. These processes work together to help teams deliver better software, faster. DevOps is not about using one particular tool, it is about using the right combination of tools to accelerate development and mitigate risk.

This is not an endorsement of one tool over another. DevOps is about harnessing a diverse software suite to meet your organization's evolving needs.

Use Cases

A major cornerstone of DevOps 2.0 is the ability to control feature releases independently from your code deployments. Broadly speaking, it is the understanding that features do not need to be waterfall released into the wild, but that they can be controlled in a production context. With DevOps 2.0...

- *Designers* can conduct user testing by toggling experimental features on and off for test users
- *Sales* can bundle features into custom plans or perform product demos to individual customers
- *Marketing* can run beta programs to get testimonials and case studies even before a formal launch
- *Product team* can see features before they are release, get real user feedback (https://dzone.com/articles/the-new-way-to-conduct-real-time-user-research), and continuously monitor product development
- *Developers* can feature flag (https://blog.launchdarkly.com/feature-flag-driven-development/) code to release it in logical chunks, mitigate risk, and roll back poorly performing features
- *System admins* can perform database migrations (https://blog.launchdarkly.com/feature-flagging-to-mitigate-risk-in-database-migration/) with mitigated risk

Benefits

Here are some of the benefits (https://launchdarkly.com/use-cases/?utm_source=launchdarkly_blog&utm_medium=organic) of separating feature rollout from code deployment in a DevOps 2.0 world:

- Release Code in Logical Chunks
  undefined
- Risk Mitigation
  undefined
- User Targeting
  undefined
- Percentage Rollouts
  undefined
- Kill Switch
  undefined
- Payment Plans
  undefined
- A/B Testing of Functionality
  undefined
- Sunseting Features
  undefined
- Opt In Beta
  undefined
- Paygates
  undefined
- Maintenance Mode
  undefined
- Newbie vs. Power User
  undefined

In a broader context, DevOps 2.0 is a user-centered approach to development. Instead of releasing software and hoping it works, we can release software with the ability to adapt to consumer demands and expectations. If we launch a feature and no one likes it, then we can instantly roll it back. If we launch a feature to 1% of our user base and they love it, then we can roll it out to the rest.

Caveats

Of course, any time you grant non-technical team members access to any aspect of your application, there will be some inherent risk. Hence, one of the main purposes of DevOps 2.0 is to mitigate that risk through proper checks, permissions, and unencumbered collaboration.

By regulating access to feature release controls, teams can place the responsibility of targeted feature visibility into the hands of non-technical members without the risk of having to modify the DB or deploy any changes to the codebase.

This is not to say that you should give non-developers the ability to control feature rollouts and any aspect of the deployment process. It is to say, however, that DevOps 2.0 will be about empowering non-technical team members without compromising the integrity of the application.

Future of DevOps 2.0: User-Centered Deployments

In the coming years, we will likely see the emergence of tools specifically designed to coordinate the skillsets of both developers and non-developers, thereby taking the principles of user-centered design (http://www.usability.gov/what-and-why/user-centered-design.html) and applying them to a state of continuous delivery and release.

DevOps, therefore, does not merely remain a way to deploy and maintain an application, it becomes a way to deliver software faster and with less risk. With the rise of continuous delivery, we will deploy software in ways that align with increasing consumer expectations (https://dzone.com/articles/in-2016-can-devops-keep-pace-with-consumer-expecta):

- Consumers want new features faster: months not years
- Consumers want bugs fixed immediately: minutes not hours
- Consumers want things to load instantly: milliseconds not seconds

A user-centered deployment, therefore, is a way to frame continuous delivery from the perspective of your product's end-user. The way we release software becomes nearly as important as the software itself. It is a way to satiate increasing consumer expectations without compromising the integrity of your application. It also ushers in a new era where we harness continuous delivery to make our users' lives better, not just a way to accelerate the development cycle.

By Justin(/blog/author/justinucd/) **Baker**

You May Like

BEST PRACTICES
(/blog/category/best-practices/)
Testing in Production to Stay Safe and Sensible (/blog/testing-in-production-for-safety-and-sanity/)

BEST PRACTICES
(/blog/category/best-practices/)
What Is Continuous Testing? A Straightforward Introduction (/blog/what-is-continuous-testing-a-straightforward-introduction1/)

INDUSTRY INSIGHTS
(/blog/category/industry-insights/)
What is a Production Environment? (/blog/what-is-a-production-environment/)

INDUSTRY INSIGHTS
(/blog/category/industry-insights/)
AWS re:Invent 2022 re:Capped (/blog/aws-reinvent-2022-recapped/)

INDUSTRY INSIGHTS
(/blog/category/industry-insights/)
Guide to Platform Engineering: Everyone's Doing It, Should You Be Too? (/blog/what-is-platform-engineering/)

TEAM & NEWS
(/blog/category/team-news/)
ICYMI: The Year in LaunchDarkly 2022 (/blog/the-year-in-launchdarkly-2022/)

Inboxes love LaunchDarkly.

Make sure you get all the content, tips, and news you can use.

Work Email Yes, send me emails

SUPPORT

Support Home (https://support.launchdarkly.com/hc/en-us/)

Request Support (https://support.launchdarkly.com/hc/en-us/requests/new?ticket_form_id=360000836893)

Documentation (https://docs.launchdarkly.com/home)

Status (https://status.launchdarkly.com/)

WHY US

ROI of feature management (/roi-feature-management/)

Trust & security (/security/)

Implementation (/implementation/)

LaunchDarkly vs. competitors (/compare/)

LaunchDarkly on AWS (/solutions/launchdarkly-aws/)

Economic impact of LaunchDarkly (https://resources.launchdarkly.com/use-cases/tei-of-launchdarkly-final-2)

LEARN

Case studies (/case-studies/)

Webinars (/webinars/)

Events (/events/)

Blog (/blog/)

Podcast (https://www.heavybit.com/library/podcasts/to-be-continuous/)

Trajectory (/trajectory/)

Galaxy (/galaxy/)

COMPANY

About us (/about-us/)

Careers (/careers/)

Press and analysts (/press-and-analysts/)

Partner program (/partners/)

Terms & Policies (/policies/subscription-terms/)

Contact us (/contact-us/)

LaunchDarkly.org (https://launchdarkly.org/)

 (https://twitter.com/launchdarkly) (https://www.linkedin.com/company/launchdarkly/mycompany/) (https://www.meetup.com/Test-in-Production/) (htt