What is DevOps?

Infrastructure as Code

Mike Loukides

O'REILLY®

O'REILLY® Velocity Web Performance and Operations

O'REILLY®

Velocity

Web Performance and Operations

CONFERENCE

Building a Faster and Stronger Web

The only things expanding faster than new techniques and technologies to build a faster, stronger web are the many features web and mobile users have come to expect.

To succeed in the ever-shifting, increasingly complexity that is web ops and performance, you not only have keep up with new tools and technologies, you also have to constantly look ahead to what's coming next.

Sign up for the Velocity Newsletter today to receive weekly insights from the people shaping the future of the web.

oreilly.com/velocity

What is DevOps?

Mike Loukides

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

What is DevOps?
by Mike Loukides

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

June 2012: First Edition.

Revision History for the First Edition:
2012-06-04 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449339104* for release details.

ISBN: 978-1-449-33910-4

[LSI]

1339158883

Table of Contents

What is DevOps?

Adrian Cockcroft's article about NoOps at Netflix ignited a controversy that has been smouldering for some months. John Allspaw's detailed response to Adrian's article makes a key point: What Adrian described as "NoOps" isn't really. Operations doesn't go away. Responsibilities can, and do, shift over time, and as they shift, so do job descriptions. But no matter how you slice it, the same jobs need to be done, and one of those jobs is operations. What Adrian is calling NoOps at Netflix isn't all that different from Operations at Etsy. But that just begs the question: What do we mean by "operations" in the 21st century? If NoOps is a movement for replacing operations with something that looks suspiciously like operations, there's clearly confusion. Now that some of the passion has died down, it's time to get to a better understanding of what we mean by operations and how it's changed over the years.

At a recent lunch, John noted that back in the dawn of the computer age, there was no distinction between dev and ops. If you developed, you operated. You mounted the tapes, you flipped the switches on the front panel, you rebooted when things crashed, and possibly even replaced the burned out vacuum tubes. And you got to wear a geeky white lab coat. Dev and ops started to separate in the '60s, when programmer/analysts dumped boxes of punch cards into readers, and "computer operators" behind a glass wall scurried around mounting tapes in response to IBM JCL. The operators also pulled printouts from line printers and shoved them in labeled cubbyholes, where you got your output filed under your last name.

The arrival of minicomputers in the 1970s and PCs in the '80s broke down the wall between mainframe operators and users, leading to the system and network administrators of the 1980s and '90s. That was the birth of modern "IT operations" culture. Minicomputer users tended to be computing professionals with just enough knowledge to be dangerous. (I remember when a new director was given the root password and told to "create an account for yourself" ... and promptly crashed the VAX, which was shared by about 30 users). PC users required networks; they required support; they

required shared resources, such as file servers and mail servers. And yes, BOFH ("Bastard Operator from Hell") serves as a reminder of those days. I remember being told that "no one" else is having the problem you're having — and not getting beyond it until at a company meeting we found that everyone was having the exact same problem, in slightly different ways. No wonder we want ops to disappear. No wonder we wanted a wall between the developers and the sysadmins, particularly since, in theory, the advent of the personal computer and desktop workstation meant that we could all be responsible for our own machines.

But somebody has to keep the infrastructure running, including the increasingly important websites. As companies and computing facilities grew larger, the fire-fighting mentality of many system administrators didn't scale. When the whole company runs on one 386 box (like O'Reilly in 1990), mumbling obscure command-line incantations is an appropriate way to fix problems. But that doesn't work when you're talking hundreds or thousands of nodes at Rackspace or Amazon. From an operations standpoint, the big story of the web isn't the evolution toward full-fledged applications that run in the browser; it's the growth from single servers to tens of servers to hundreds, to thousands, to (in the case of Google or Facebook) millions. When you're running at that scale, fixing problems on the command line just isn't an option. You can't afford letting machines get out of sync through ad-hoc fixes and patches. Being told "We need 125 servers online ASAP, and there's no time to automate it" (as Sascha Bates encountered) is a recipe for disaster.

The response of the operations community to the problem of scale isn't surprising. One of the themes of O'Reilly's Velocity Conference is "Infrastructure as Code." If you're going to do operations reliably, you need to make it reproducible and programmatic. Hence virtual machines to shield software from configuration issues. Hence Puppet and Chef to automate configuration, so you know every machine has an identical software configuration and is running the right services. Hence Vagrant to ensure that all your virtual machines are constructed identically from the start. Hence automated monitoring tools to ensure that your clusters are running properly. It doesn't matter whether the nodes are in your own data center, in a hosting facility, or in a public cloud. If you're not writing software to manage them, you're not surviving.

Furthermore, as we move further and further away from traditional hardware servers and networks, and into a world that's virtualized on every level, old-style system administration ceases to work. Physical machines in a physical machine room won't disappear, but they're no longer the only thing a system administrator has to worry about. Where's the root disk drive on a virtual instance running at some colocation facility? Where's a network port on a virtual switch? Sure, system administrators of the '90s managed these resources with software; no sysadmin worth his salt came without a portfolio of Perl scripts. The difference is that now the resources themselves may be physical, or they may just be software; a network port, a disk drive, or a CPU has nothing to do with a physical entity you can point at or unplug. The only effective way to manage this layered reality is through software.

So infrastructure had to become code. All those Perl scripts show that it was already becoming code as early as the late '80s; indeed, Perl was designed as a programming language for automating system administration. It didn't take long for leading-edge sysadmins to realize that handcrafted configurations and non-reproducible incantations were a bad way to run their shops. It's possible that this trend means the end of traditional system administrators, whose jobs are reduced to racking up systems for Amazon or Rackspace. But that's only likely to be the fate of those sysadmins who refuse to grow and adapt as the computing industry evolves. (And I suspect that sysadmins who refuse to adapt swell the ranks of the BOFH fraternity, and most of us would be happy to see them leave.) Good sysadmins have always realized that automation was a significant component of their job and will adapt as automation becomes even more important. The new sysadmin won't power down a machine, replace a failing disk drive, reboot, and restore from backup; he'll write software to detect a misbehaving EC2 instance automatically, destroy the bad instance, spin up a new one, and configure it, all without interrupting service. With automation at this level, the new "ops guy" won't care if he's responsible for a dozen systems or 10,000. And the modern BOFH is, more often than not, an old-school sysadmin who has chosen not to adapt.

James Urquhart nails it when he describes how modern applications, running in the cloud, still need to be resilient and fault tolerant, still need monitoring, still need to adapt to huge swings in load, etc. But he notes that those features, formerly provided by the IT/operations infrastructures, now need to be part of the application, particularly in "platform as a service" environments. Operations doesn't go away, it becomes part of the development. And rather than envision some sort of uber developer, who understands big data, web performance optimization, application middleware, and fault tolerance in a massively distributed environment, we need operations specialists on the development teams. The infrastructure doesn't go away — it moves into the code; and the people responsible for the infrastructure, the system administrators and corporate IT groups, evolve so that they can write the code that maintains the infrastructure. Rather than being isolated, they need to cooperate and collaborate with the developers who create the applications. This is the movement informally known as "DevOps."

Amazon's EBS outage last year demonstrates how the nature of "operations" has changed. There was a marked distinction between companies that suffered and lost money, and companies that rode through the outage just fine. What was the difference? The companies that didn't suffer, including Netflix, knew how to design for reliability; they understood resilience, spreading data across zones, and a whole lot of reliability engineering. Furthermore, they understood that resilience was a property of the application, and they worked with the development teams to ensure that the applications could survive when parts of the network went down. More important than the flames about Amazon's services are the testimonials of how intelligent and careful design kept applications running while EBS was down. Netflix's ChaosMonkey is an excellent, if extreme, example of a tool to ensure that a complex distributed application can survive outages; ChaosMonkey randomly kills instances and services within the application.

The development and operations teams collaborate to ensure that the application is sufficiently robust to withstand constant random (and self-inflicted!) outages without degrading.

On the other hand, during the EBS outage, nobody who wasn't an Amazon employee touched a single piece of hardware. At the time, JD Long tweeted that the best thing about the EBS outage was that his guys weren't running around like crazy trying to fix things. That's how it should be. It's important, though, to notice how this differs from operations practices 20, even 10 years ago. It was all over before the outage even occurred: The sites that dealt with it successfully had written software that was robust, and carefully managed their data so that it wasn't reliant on a single zone. And similarly, the sites that scrambled to recover from the outage were those that hadn't built resilience into their applications and hadn't replicated their data across different zones.

In addition to this redistribution of responsibility, from the lower layers of the stack to the application itself, we're also seeing a redistribution of costs. It's a mistake to think that the cost of operations goes away. Capital expense for new servers may be replaced by monthly bills from Amazon, but it's still cost. There may be fewer traditional IT staff, and there will certainly be a higher ratio of servers to staff, but that's because some IT functions have disappeared into the development groups. The bonding is fluid, but that's precisely the point. The task — providing a solid, stable application for customers — is the same. The locations of the servers on which that application runs, and how they're managed, are all that changes.

One important task of operations is understanding the cost trade-offs between public clouds like Amazon's, private clouds, traditional colocation, and building their own infrastructure. It's hard to beat Amazon if you're a startup trying to conserve cash and need to allocate or deallocate hardware to respond to fluctuations in load. You don't want to own a huge cluster to handle your peak capacity but leave it idle most of the time. But Amazon isn't inexpensive, and a larger company can probably get a better deal taking its infrastructure to a colocation facility. A few of the largest companies will build their own datacenters. Cost versus flexibility is an important trade-off; scaling is inherently slow when you own physical hardware, and when you build your data centers to handle peak loads, your facility is underutilized most of the time. Smaller companies will develop hybrid strategies, with parts of the infrastructure hosted on public clouds like AWS or Rackspace, part running on private hosting services, and part running in-house. Optimizing how tasks are distributed between these facilities isn't simple; that is the province of operations groups. Developing applications that can run effectively in a hybrid environment: that's the responsibility of developers, with healthy cooperation with an operations team.

The use of metrics to monitor system performance is another respect in which system administration has evolved. In the early '80s or early '90s, you knew when a machine crashed because you started getting phone calls. Early system monitoring tools like HP's OpenView provided limited visibility into system and network behavior but didn't give much more information than simple heartbeats or reachability tests. Modern tools

like DTrace provide insight into almost every aspect of system behavior; one of the biggest challenges facing modern operations groups is developing analytic tools and metrics that can take advantage of the data that's available to predict problems before they become outages. We now have access to the data we need, we just don't know how to use it. And the more we rely on distributed systems, the more important monitoring becomes. As with so much else, monitoring needs to become part of the application itself. Operations is crucial to success, but operations can only succeed to the extent that it collaborates with developers and participates in the development of applications that can monitor and heal themselves.

Success isn't based entirely on integrating operations into development. It's naive to think that even the best development groups, aware of the challenges of high-performance, distributed applications, can write software that won't fail. On this two-way street, do developers wear the beepers, or IT staff? As Allspaw points out, it's important not to divorce developers from the consequences of their work since the fires are frequently set by their code. So, both developers and operations carry the beepers. Sharing responsibilities has another benefit. Rather than finger-pointing post-mortems that try to figure out whether an outage was caused by bad code or operational errors, when operations and development teams work together to solve outages, a post-mortem can focus less on assigning blame than on making systems more resilient in the future. Although we used to practice "root cause analysis" after failures, we're recognizing that finding out the single cause is unhelpful. Almost every outage is the result of a "perfect storm" of normal, everyday mishaps. Instead of figuring out what went wrong and building procedures to ensure that something bad can never happen again (a process that almost always introduces inefficiencies and unanticipated vulnerabilities), modern operations designs systems that are resilient in the face of everyday errors, even when they occur in unpredictable combinations.

In the past decade, we've seen major changes in software development practice. We've moved from various versions of the "waterfall" method, with interminable up-front planning, to "minimum viable product," continuous integration, and continuous deployment. It's important to understand that the waterfall and methodology of the '80s aren't "bad ideas" or mistakes. They were perfectly adapted to an age of shrink-wrapped software. When you produce a "gold disk" and manufacture thousands (or millions) of copies, the penalties for getting something wrong are huge. If there's a bug, you can't fix it until the next release. In this environment, a software release is a huge event. But in this age of web and mobile applications, deployment isn't such a big thing. We can release early, and release often; we've moved from continuous integration to continuous deployment. We've developed techniques for quick resolution in case a new release has serious problems; we've mastered A/B testing to test releases on a small subset of the user base.

All of these changes require cooperation and collaboration between developers and operations staff. Operations groups are adopting, and in many cases, leading in the effort to implement these changes. They're the specialists in resilience, in monitoring,

in deploying changes and rolling them back. And the many attendees, hallway discussions, talks, and keynotes at O'Reilly's Velocity conference show us that they are adapting. They're learning about adopting approaches to resilience that are completely new to software engineering; they're learning about monitoring and diagnosing distributed systems, doing large-scale automation, and debugging under pressure. At a recent meeting, Jesse Robbins described scheduling EMT training sessions for operations staff so that they understood how to handle themselves and communicate with each other in an emergency. It's an interesting and provocative idea, and one of many things that modern operations staff bring to the mix when they work with developers.

What does the future hold for operations? System and network monitoring used to be exotic and bleeding-edge; now, it's expected. But we haven't taken it far enough. We're still learning how to monitor systems, how to analyze the data generated by modern monitoring tools, and how to build dashboards that let us see and use the results effectively. I've joked about "using a Hadoop cluster to monitor the Hadoop cluster," but that may not be far from reality. The amount of information we can capture is tremendous, and far beyond what humans can analyze without techniques like machine learning.

Likewise, operations groups are playing a huge role in the deployment of new, more efficient protocols for the web, like SPDY. Operations is involved, more than ever, in tuning the performance of operating systems and servers (even ones that aren't under our physical control); a lot of our "best practices" for TCP tuning were developed in the days of ISDN and 56 Kbps analog modems, and haven't been adapted to the reality of Gigabit Ethernet, OC48* fiber, and their descendants. Operations groups are responsible for figuring out how to use these technologies (and their successors) effectively. We're only beginning to digest IPv6 and the changes it implies for network infrastructure. And, while I've written a lot about building resilience into applications, so far we've only taken baby steps. There's a lot there that we still don't know. Operations groups have been leaders in taking best practices from older disciplines (control systems theory, manufacturing, medicine) and integrating them into software development.

And what about NoOps? Ultimately, it's a bad name, but the name doesn't really matter. A group practicing "NoOps" successfully hasn't banished operations. It's just moved operations elsewhere and called it something else. Whether a poorly chosen name helps or hinders progress remains to be seen, but operations won't go away; it will evolve to meet the challenges of delivering effective, reliable software to customers. Old-style system administrators may indeed be disappearing. But if so, they are being replaced by more sophisticated operations experts who work closely with development teams to get continuous deployment right; to build highly distributed systems that are resilient; and yes, to answer the pagers in the middle of the night when EBS goes down. DevOps.

Photo: Taken at IBM's headquarters in Armonk, NY. By Mike Loukides.