TODAY'S TOP STORIES

The dark side of DevOps: Forewarned is forearmed

Ideal scenarios rarely play out in real life, and DevOps is no different. Learn how implementing this software development philosophy can flop—and how to avoid disaster.

By Josh Fruhlinger

CIO |
OCT 3, 2018 3:13 AM PT

DevOps is an increasingly popular organizational philosophy that boosts collaboration between development and operations staff—perhaps even combining them into a single department—and prescribes constant integration and continuous delivery (CI/CD) of new code, with software features being rolled out incrementally every few days or weeks rather than unleashed as huge, monolithic updates. In theory, it makes development more nimble and cuts down on internal strife.

Of course, theory doesn't always play out in practice. We spoke to people who've helped roll out DevOps in the real world to find out the philosophy's dark side—both problems inherent to it and ways in which its implementers often fall short or screw up. We hope these cautionary tales will help you go into any rollout with your eyes open.

[Beware the 9 warning signs of bad IT architecture and see why these 10 old-school IT principles still rule. | Sign up for CIO newsletters.]

DevOps is more than a suite of tools

Ernest Mueller, on The Agile Admin site, defines what he describes as "DevOps tools":

Tools you'd use in the commission of these principles. In the DevOps world there's been an explosion of tools in release (jenkins, travis, teamcity), configuration management (puppet, chef, ansible, cfengine), orchestration (zookeeper, noah, mesos), monitoring, virtualization and containerization (AWS, OpenStack, vagrant, docker) and many more. While, as with Agile, it's incorrect to say a tool is "a DevOps tool" in the sense that it will magically bring you DevOps, there are certainly specific tools being developed with the express goal of facilitating the above principles, methods, and practices, and a holistic understanding of DevOps should incorporate this layer.

Unfortunately, treating these tools or others as a totem that will "magically bring you DevOps" is exactly what far too many shops do, without making an effort to change processes or mindsets. "Mike," who worked in the office of the CTO of a medium-sized tech company during its attempted shift to DevOps, describes this sort of "cargo cult" vision of DevOps. "I'll blame engineer tunnel vision on this," he says, "because software engineers look at problems with software solutions. We looked at DevOps as a suite of technologies to be installed across many applications. This is so totally wrong it's hard to pick where to start. We bought copies of *The Phoenix Project* for everyone that wanted one. We had 'lunch and learn' lessons on Ruby and Chef. We ran hackathons and actually awarded top ranks to infrastructure projects."

Mike's DevOps transition, it turned out, was riddled with failures, and he places much of the blame on this original failure to grasp even what the project he was embarking on actually entailed. "If DevOps isn't software, what is it?" he asks. "Looking back I feel dumb for not noticing. DevOps is a culture. Changing culture is one of the hardest things an organization can do. People—and perhaps engineers in particular—are naturally defiant and don't readily accept edicts from on high. Culture must be grown and cultivated from within by strong people managers, and strong people managers are rare in software organizations."

Not everyone's going to be on board

And that brings us to our next dark side of DevOps. The dream is that your tech staff will eagerly get on board with the new philosophy and way of doing things that DevOps entails. The reality is that change is hard, and many people don't see anything wrong with the job they're already doing and the way they do it. Cody Swann is the CEO of Gunner Technology, a software development firm that helps private and public sector organizations move to DevOps—and he's seen it all when it comes to resistance, with some employees taking active measures to make a transition as

difficult as possible. For instance, in one assignment, employees questioned the viability of the new infrastructure they were proposing. "The ops team tried to make the case that microservices are a fad," he said, "and aren't sustainable for larger projects because they're too difficult to maintain and that JavaScript can't scale."

[Learn Java from beginning concepts to advanced design patterns in this comprehensive 12-part course!]

Things didn't necessarily get better as they moved to implementation. "We were breaking apart a huge monolithic .NET app/infrastructure into AWS Lambda-based JavaScript microservices," he says. "Part of this included migrating data from Oracle to a DynamoDB/Elasticsearch solution." The catch? "The on-premise ops team took the position that it was impossible to grant us access to the Oracle database to migrate the data. The excuses were obviously contrived—they claimed the database was only available to internal IP addresses, for instance. Eventually we set up a VPN peering connection from AWS to migrate the data, but there was a lot of pushback to get that data migrated."

In Mike's transition, things got even worse, because it ended up challenging some staffers' professional sense of self. "In a home-grown DevOps culture, the pain of bad deployments and production failures is felt directly by the engineers that built the system," he says. "In a traditional two-team approach, operations is on their own to fix production errors. They like this. Fixing a production bug and returning the service to a fully working state is the stuff of legend. The very act of someone saying, 'Get Gary on the phone—he's the only one that can fix this,' and then being Gary, and actually fixing it, is the biggest endorphin rush and sense of job satisfaction most computer geeks ever get."

But the transition to DevOps, whatever its other benefits, "was going to take all of this away," Mike explains. "So the system administrators went elsewhere to get their fix. Also, with no data centers and things running in the cloud, network engineers didn't have a lot to do. They could retrain, or they could go elsewhere." The upshot? "Half of ops quits."

Attempts to get the remaining staff to build the necessary frameworks for CI/CD also ran into trouble. "Turns out that old school Java telecom programmers don't like writing Ruby," Mike explains. "They're highly specialized, do their jobs very well, and don't necessarily have or want the wide skillset needed to write idempotent installers in another language. We spent a bunch of time

trying to motivate people by giving them ownership of their applications from the initial designs to the production deployment. While a few people enjoyed this new holistic approach and adopted the DevOps mindset readily, most of them wanted to work on what they're good at."

Security, as usual, is a problem

DevOps may not be any *more* insecure than other organizational philosophies, but its implementation can cause security problems in new and excitingly different ways. "With continuous integration and continuous deployment/delivery (CI/CD) as the main driver in DevOps, a large part of good security practices are often automated away or excluded from the process altogether simply due to the impact it has on the velocity of software releases," explains Davy Hua, Head of DevOps for ShiftLeft. "As a result, increasing the speed of releases will often produce improperly vetted code, which may lead to inadvertent sensitive data leakage to logging endpoints, or inheriting vulnerabilities from third party software libraries, as was the case with the Equifax breach."

"Whether the application is a monolith or broken up into various microservices, the complexities between the flows of data requires a great deal of understanding and insight in order to implement an effective security program," Hua continues. "Couple the complexities with time constraint, another factor for speeding up releases, will result in a certain level of neglect when it comes to security enforcement."

Igor Livshitz, Director of Product Management at GuardiCore, sees security becoming a particular problem in DevOps when it's "integrated at different *gateway points* of the DevOps process instead of being a byproduct post deployment."

"In today's world of short-lived instances, microservices, and hybrid infrastructures, any human-based single point of control can't really keep up," he explains. "So the policy control is moved down to the developers as part of a shared responsibility model. The developers describe the necessary policy that should be applied for their application and network security administrators can review it and also apply auditing and monitoring tools to ensure no breaches of policy or compliance. In this way all parties are happy: DevOps and developers can move fast without waiting for a green light and security administrators get tight policies created by application owners that only need to be audited."

But too often, this ideal scenario doesn't play out in real life. "This can all break down if there is a lack of trust between the security and DevOps functions in the organization. In the instances where we have encountered it, the two are usually part of separate business units, and they have been going at it for years—the partisanship is ingrained in the company's DNA. The distrust may have originated from a security incident that happened because of a lack of awareness on one or both sides that led to application or services downtime due to 'overblocking' by the security teams." For Livshitz, the key to avoiding this is to build up trust between teams, and to "integrate the necessary security controls as part of the DevOps cycle—controls that should be defined by the security teams. Those teams should be able to audit and monitor for policy violation within those applications."

Move fast—but not too fast

The whole point of moving to DevOps is that you end up producing useful software updates much more quickly. But too often teams think their move to DevOps itself should go just as fast—with disastrous results, especially if workforce cultural change isn't given a chance to keep pace. The upshot can be a sort of Frankenstein half-DevOps shop. Ed Price, director of compliance at Devbridge Group, gives an example of some warning signs: "If companies are still seeing manual code migrations and manually putting code into production. If they still rely on an environments team. If they see no automated unit, regression, or performance testing built into their CI pipeline to prevent code from getting to market that isn't ready yet."

It also takes time to get the mix of personnel right that you need for a good DevOps team. You might not run into the scenario we described above where half your staff quits, but that doesn't mean you just throw everyone in the mix together and hope they work out. "It's common for organizations to identify the stereotypical 'rock stars'—the high-performing, go-to members of the team, as selections for any new initiative," says Justin Rodenbostel, Vice President of Open Source Application Development at SPR. "However, building teams strictly based on technical prowess or business subject matter expertise can backfire if other skills like collaboration and communication aren't also considered."

Build for growth

If there's one lesson to be learned from all of this, it's that people are the most important to determining whether your DevOps project goes to the dark side or the light. "Bringing on people that have a 'steady-state' mindset, as opposed to understanding what it means to build DevOps from scratch," says Ryan Duguid, chief evangelist at Nintex. "When this happens, teams end up over-engineering, over-spending, or delivering too much sophistication that is difficult to cope with. You have to have people who can come in and say 'Ok, what's needed here?' not 'This is what I did in my last job.' If you're a steady-state company it makes sense to hire steady-state people. But if you're starting the journey you need talent that understands, has built DevOps from the ground up, and knows how to go back to basics." The lesson for DevOps newbies: make sure you build the right culture from the ground up—and learn to walk before you try to fly.

More on building the culture you want:

- Define your organization's culture before it derails you
- Tips for building a culture around 'business outcomes'
- The rise of the feedback culture
- 3 ways your culture may be sabotaging innovation
- Who's the boss of workplace culture?
- 3 tips to foster a culture of innovation
- Why bimodal IT kills your culture and adds complexity

Next read this:

- *7 habits of highly effective digital leaders*
- *20 ways to kill your IT career (without knowing it)*
- *16 time-saving tips for IT leaders*
- *CIO resumes: 6 best practices and 4 strong examples*
- *Why IT-business alignment still fails*
- *Top 12 ITSM tools for 2018*
- *6 most-dreaded IT projects*
- *CIO playbook: 10 tips for leading IT in the digital era*
- *15 ways to advance your IT career*
- *7 habits of highly effective digital transformations*

Josh Fruhlinger is a writer and editor who lives in Los Angeles.

Follow

SUBSCRIBE! Get the best of CIO delivered to your email inbox.