best practices

Incremental DevOps

Incremental DevOps

A sensible route for organizations to adopt DevOps.

DevOps sounds like a panacea, but then again, so has every other buzzword-turned-standard, from the World Wide Web to Agile Software Development. In just a few, short years, DevOps will become the new standard. Not because it's some disruptive revolution, but because it's an evolution that responds to the challenges of modern software development.

This paper offers a sensible route for organizations to evolve towards DevOps.

DevOps Done Wrong

Tuesday, 13 March 2015

by Admin User

The following is a true story, told by an engineer working in the field; only the names have been changed to protect the "innocent".

"This might sound a little crazy," Tom shared in his Delivery Days conference debriefing, "but I've seen Xanadu. And it was... amazing. It's going to change everything. "

Tom had been sprouting clichéd mantras for years. Whether it was "tear down the wall between development and operations," or "collaborate, communicate, then create," he'd say them with the maniacal confidence of a charlatan.

But this time was different. This time, management was actually listening.

"The Xanadu framework was absolutely transformative for Netsyflix," Tom continued, "it's what made the DevOps machine they are today. Without Xanadu, they'd probably still just be mailing out DVDs. They release hundreds of times a week to dozens of countries, all with no downtime."

That certainly piqued management's interest. As for the rest of us, we were skeptical. Not just because it was a change, and certainly not just because it was Tom's idea, but because we were no Netsyflix. We had a plethora of different applications on different platforms maintained by different groups – and none of them streamed movies to consumers.

Before any of us had any time to propose something different, management was sold. After all, Tom knew how to implement it, so there was really no risk to giving it a try. At least, that's what management thought.

Fast forward six months, and we were all in Xanadu. It was not at all like the brochure. Deployments were supposedly automated, but they failed almost every time. We couldn't even use our old processes to fix things because somehow, Xanadu detected our manual change and went back to "fix" it.

Tom was still the only who knew how it all worked, and we were pretty sure he didn't actually know that. It felt as if we used a high-speed, titanium-tipped pneumatic screwdriver to tighten a few wooden screws, and we had been doing nothing but repairing the stripped threads ever since.

It wasn't long before management caught on. DevOps was supposed to help us deliver software, well, continuously -- not automate one or two basic applications and grind the rest to a halt. It didn't matter whose idea Xanadu was, or who approved it, business-critical development had been backed up more than ever, and heads were about to start rolling.

And that's when I got the heck out of there. The chopping block was the least of my worries: it was about to get a lot worse and I didn't want to clean up the mess.

Things have been getting better though, at least according to the coworkers I've stayed in touch with. Almost two years after they implemented Xanadu, they're almost back to where they started. Still, no one dares mention "DevOps" ever again.

To be continued...

DevOps as the New Standard

It's not a question of if, but when.

We're in the midst of yet another turning point in software development history. The age of massive, multi-million dollar hand-offs is coming to an end, and organizations must adapt to ever-shrinking deliverables, faster feedback cycles, and everything thing else that rapid, agile change demands. Those that can't – or won't – will go from a legacy to a dinosaur to extinct.

DevOps has grown from a niche idea championed by small, elite teams, into a proven approach that eliminates risk from the delivery aspect of modern software development. This is an important step all organizations must take in order to become more agile – not just in their software development, but in their ability to adapt and change as fast as their customers and competitors.

One common response to the new DevOps reality is deferment. "It's a massive change, and at best it will reduce risks," many executives will say. "We don't absolutely need it today, so let's just be a bit more careful, and focus on it next year."

Eventually, the cost of deferment will overcome any sensible benefit. As the existing tools and processes become obsolete, change processes will become more cumbersome, employees will become increasingly dissatisfied, and eventually you'll be the dinosaur on its way to extinction.

Moving towards DevOps

The risk of risk reduction.

Although DevOps is, in part, a risk reduction methodology, implementing the methodology is a risk in and of itself. As the DevOps Done Wrong narrative illustrated, a poor implementation will leave you worse off than no implementation.

If your development organization only maintains a few applications, and has a small number of developers and internal customers, moving towards DevOps is fairly easy. It's still is an organization-wide change, but there's hardly any organization that needs to change: it's just a small team of people.

But when there are lots of applications built by lots of developers that are depended upon by lots of customers, changing the ways in which those applications are developed could prove to be so complex and so intrusive that it could unintentionally become the same productivity-killing enemy as the problems it sought to solve.

As you begin improving your software delivery practices, you quickly shine a light on a whole host of previously-hidden problems – some that may, or may not have directly contributed to your original delivery problems. Whether it's being unable to physically locate that "one" critical build server, or finding an arcane script that no one remembers writing, these unknowable problems quickly surface and can grind any sort of automation plan to a halt.

Technical challenges aside, implementing the necessary cultural changes can be just as treacherous. It's hard enough to get ten people on board with any new idea – let alone a hundred, or even a thousand. The people problems can just as easily derail a DevOps implementation as the technical ones.

The Costs of Moving Forward

No matter how you approach it, adopting DevOps across a mid- or large-size organization is going to be a costly endeavor. While the benefits will largely outweigh the cost in time, once all things are said and done – tools licensing, consultant fees, not to mention the internal hours for training and implementing – the total bill will end up with six- or seven-figures.

In an ideal world, you order DevOps in a Box, write a massive check, open up the box, and next thing you'd know, everything would work perfectly. But course, in this same ideal world, you wouldn't have the same delivery challenges to begin with.

In reality, there is no DevOps in a Box, and there is no way to transform the organization in a day, a week, or even a month. It might be possible over a quarter, but that'd take an incredibly aggressive, high-risk plan that may leave the organization in worse shape than before, especially if they are expected to continue delivering software in that period.

Aggressive Adoption

The Aggressive Adoption is about as close to DevOps in a Box as you can get; it costs a lot up front, and promises to deliver in a very short order. Unfortunately, this approach will very likely fail to deliver, especially in the mid-term.

Although there are several books on DevOps, they are neither manuals nor how-tos. In fact, no such guidebooks exist because there is no "one way" to achieve DevOps; it's a process, not an event or milestone, and there's no clear way to get there.

"Accelerating the introduction of new IT services and changes to existing ones will not happen overnight. I&O organizations frequently err by attempting too great of a velocity increase without factoring in all of the dependencies. Additionally, the process of increasing the velocity of releases should not necessarily be applied globally."

Gartner's Seven Steps to Improve Release Velocity (06 June 2014)

Without a clear roadmap, this aggressive adoption of DevOps must take one of two equally problematic routes.

The Unified March is a centrally-planned, specific directive for DevOps that each team must simultaneously adopt. This route is tethered both in velocity and direction to the slowest team (i.e. the team most resistant to change, be it for technical or cultural reasons). This pace will naturally frustrate teams eager and able to adopt a DevOps workflow, and will make them equally resistant to the change. At best, the Unified March will yield a "one size fits all" adoption, also known as a "poor fit for most."

The Free-for-all is a loosely-defined organizational mandate for DevOps that each team must implement in their own way. While this route avoids the "one size fits all" adoption, and very may yield success for some teams, most teams lack the experience, resources, and general aptitude to implement a suitable DevOps workflow. Many teams will be left behind through a failed adoption or not even attempting one; more likely than not, these teams will be in need of DevOps the most.

Incremental Adoption

The alternative approach to aggressive adoption is incremental adoption; while this approach takes longer, it represents significantly less risk, and is all but guaranteed to be successful.

A key challenge is that "Enterprise IT environments are complex and require an approach that balances speed and risk, a combination that Infrastructure & Operations leaders are struggling to make work together."
One recommendation is to "Plan for a continual improvement journey and not a one-time event."

Gartner's Seven Steps to Improve Release Velocity (06 June 2014)

The incremental approach starts with a small goal, such as implement DevOps for a single application maintained by a small development team. Once successful, the lessons learned are then applied towards increasingly larger goals until eventually, a healthy balance of process improvement and process execution is reached.

The rest of the paper focuses on the details of this approach.

Bringing in the Consultants

Most consulting organizations are wired to take an aggressive approach. It makes a lot of sense from a business perspective – large, upfront sales offer a known amount of stable revenue – and it allows teams to focus on a single, large project at a time.

Another challenge with a consultant-led DevOps is that the consultants eventually leave; if the organization cannot maintain the systems and processes they put in place, then they won't be able to adapt it to their ever-changing software. Ultimately, if you can't deploy what you need to deploy, it doesn't really matter if it's automated.

Those caveats aside, consultants that specialize in helping organizations adopt DevOps incrementally -- both technically and culturally -- can make the difference between a successful, well-executed project and a lagging, ever-increasing mess.

When working with continuous deliver consultants, it's critical that they don't just build an automated delivery system for your applications, but they instead work with teams to understand how to build automated delivery systems.

Just as the organization has invested in the skills and resources to build software that can adapt to business needs, they must also learn how to adapt the delivery of that software as the needs changes.

The Incremental Strategy

The incremental strategy is a six-step, cyclical approach that will not only minimize the risk and costs of a DevOps adoption, but build the necessary in-house skillset and momentum for larger and faster cycles. Each step builds upon the last, both in terms of investment and preparation, as will each overall cycle.

The hours are realistic for many enterprises, but will vary widely depending on the culture and politics of the organization. Meetings tend to consume the most hours, and these estimates assume a small implementation team (1-3 people) interacting with key application personnel (2-4 people).

The first step is a crucial one: select a suitable, small-scale application.

If any teams or groups have expressed interest in DevOps or automation, that would be a great place to start. It's likely the path of least resistance amongst personnel, and will maximize the likelihood of buy-in, cohesion, willingness to experiment, and ultimately, change.

The initial application shouldn't be technically overwhelming from a deployment standpoint, while at the same time it shouldn't be considered low priority by the business (otherwise, the value add of DevOps won't be visible), nor should it be the flagship application (too much perceived change at once). And most importantly, the application team should buy-in and understand the importance of DevOps.

This step is generally accomplished in a single meeting, as many of these factors are already well-known by key application personnel within an enterprise.

Although there's certainly no DevOps in a Box, the closest thing to a solution is a software product/tool, a process built around the platform, and a plan to implement both the software and process.

The objective of this stage is to find solutions that would not just work for the selected application, but that might, at least theoretically, work for other applications and groups in the organization. For instance, if the enterprise has a mix of .NET, Java, and ColdFusion applications deployed across Windows and Linux servers, then a Windows-only solution should be a non-starter.

Just because the enterprise's applications are built on completely different platforms, there will be a lot of overlap in their delivery process. By the same token, just because two applications are written in the same language, the process could be vastly different. For instance, a marketing website generally needs a lot less testing and scrutiny than stock trading middleware.

It may seem like a lot to take in, but to reiterate, the goal is to find something that "might, at least theoretically, work." Now is not the time to craft a 20-page requirements document with a spreadsheet detailing each and every possible technical and process specification. There's simply no way of actually knowing each and every application's requirements without spending significant time with each and every group to learn their technical requirements, so this initial organizational consideration must be done at a very high, almost "gut" level.

This step can largely be performed by the implementation team, although brief interactions with the application team may be needed to confirm technical requirements. The following guides may be helpful in discovering potentially suitable tools:

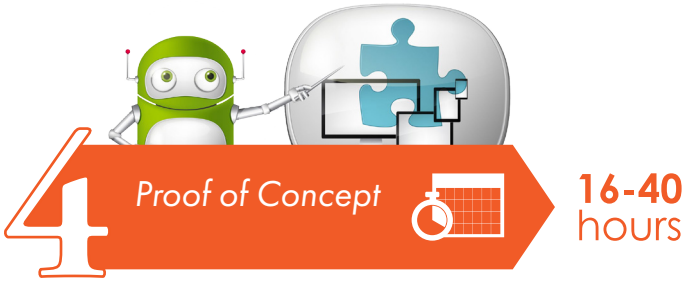
- Gartner's Cool Vendors in DevOps, 2015 (21 April 2015)
- Forrester's Market Overview for Application Release Automation Tools (2015)

Tools are a critical piece of a DevOps solution and, by this stage, you should have a short list of them that seem to be suitable for both the selected application and the enterprise at a whole. Now's the time to dig in a little deeper to see which will be worth a more time-intensive proof-of-concept.

If you were shopping for cars, this would be the equivalent of a test drive around the block. If that's too bumpy of a drive, or you just really hate driving stick shift, then it's simply not worth taking the car home for the entire weekend.

Some tool vendors offer a self-guided tutorial that you can use after a download, whereas others will have salespeople give a live web demonstration. Both types of assessments should take about the same time (1-2 hours), and both should accomplish the same goal: not necessarily finding which tool will be "the one", but finding which tools will definitely not.

When doing a self-guided tutorial, make sure not to conflate the assessment with the next stage, the proof of concept. The assessment should be focused on a "hello world" type of application, not the application selected in the first stage.



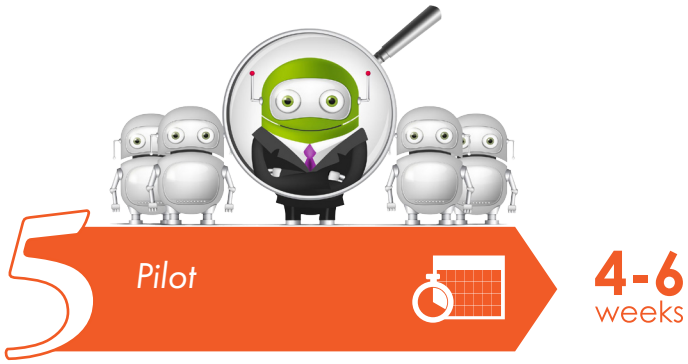
If the assessment was a test drive around the block, then the proof of concept is loading up the car with your family and pets, and then taking a road trip. At this stage, it's not just about finding the right tool, but discovering and defining the processes you'll use around the tool.

Because this will interact with the selected application, there will be slight risk involved. The tools will need to be installed on a real server, firewall ports may need to be open, permissions and service accounts may need to be created, and so on. However, this slight risk comes with a fair reward: it'll give a feel for what is actually required to bring an application into DevOps.

To mitigate some of this risk, the proof-of-concept may be split into two phases: pre-production and production. Only upon completion of the first phase would the proof-of-concept be allowed to continue to the second.

An organizational change to DevOps is as much one of hearts and minds as it is the tooling and processes. This stage will also give a feel as to the resistance and hesitation to change, as well as any perceptual leaps that need to be made.

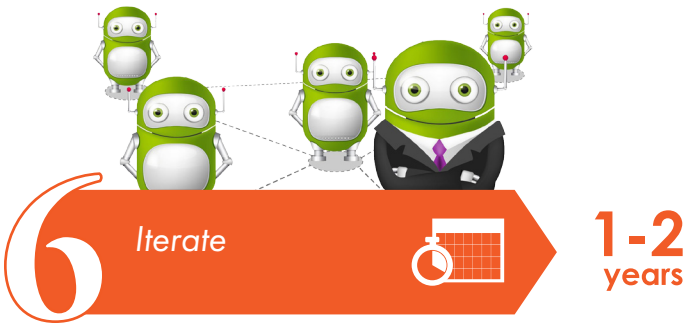
Thus, it's equally important to involve and interact with the actual people who deliver the application: testers, business analysts, network engineers, developers, etc. Just as the risk to production can be mitigated with a multi-phase proof of concept, so can overcoming the resistance to change: only upon completion of one phase of the proof-of-concept would the solution be introduced to another person or team.



The line between the proof-of-concept and the pilot can be blurry, but the phase generally shifts once the selected application is actually delivered onto real production servers, and out in to the real world.

The pilot phase can be thought of as the "new normal" for this particular application. Everything has checked out, and the benefits weren't outweighed by tremendous costs. Thus, by all accounts it makes perfect sense to continue using the newly minted DevOps process.

At this point, the new process is official: it's time to license the tools appropriately and train staff appropriately on their new tasks and responsibilities. The only non-routine activity is to continually monitor and validate that the new solution is being well-received by all involved. And of course, share stories of success with other groups in the enterprise.



In just a few months, the results of the pilot will be very real, measurable, and notable: the lack of failed builds or deploys, time saved, fewer headaches, etc. will all be reportable and transparent for other groups. And not just that, the business will start to see how much easier it is to deliver changes faster.

From here, it's back to the first stage: select another application. With all the experiences and lessons learned in implementing the process, each stage of the next iteration will be significantly easier, both technically and culturally. At some point, perhaps even after the first iteration, it'll seem almost trivial to transition several applications to DevOps in parallel.

Although this incremental process may take years, the results will be nothing short of transformative: the organization will have gained the knowledge and experience to not just implement DevOps, but to change and improve organizational processes. This experience will become invaluable as when moving onto new applications, especially when these new applications don't even come close to fitting in the same mold as the other applications.

Remember, there is no end point of DevOps. It will continue to evolve as will how you organization addresses it. For now, just take what you have learned and apply it to other applications with an open, experimental mentality.

Happy delivering!

DevOps Done Right

Monday, 6 April 2015
by Admin User

... continued from *DevOps Done Wrong.*

If there was one thing my new gig at Initrode offered, it was stability. They weren't exactly movers and shakers in information technology, but after living in Xanadu, it was a welcome change. Or at least, that's what I thought at first.

At Initrode, the traditional, "waterfall" software delivery methodology was in full effect: the development and operations silos were vertical and lofty; most things were handed-off and not seen again unless there was a problem; and communication outside of elevator small talk was sparse.

But... things... moved... really... slowly. A few times, I even wished we had a zealot like Tom as our voice of change, though perhaps with a few less decibels.

And then, something happened. It was a small step, but I confessed to my team lead my interest in process improvement and DevOps. And as it so happened, we shared the same interests. A few lunchtime conversations later, and we decided to do some experimenting to see if we could make a sort of "test tube" DevOps project to show off the idea, and how it could be done and scaled.

We picked a fairly trivial application that our group maintained, and got started. It had zero unit tests, zero UI tests, and a fairly hefty approval and manual deployment process. While that might be considered heretical at Netsyflix, it was fairly representative of Initrode's applications, both in our group and in others.

If my experience with Xanadu taught me one thing, it was that we needed to find a solution to our real-world problems --- not redefine our problem to fit an idealistic solution. Looking at it that way, we quickly realized that the first major hurdle was production. It was completely hands-off by our team, and even an Act of Congress would not have changed that for our little skunkworks project.

Fortunately, production was only the final stretch of the last mile, and we simply automated what we would have manually done: dropped some files in a share drive and sent off an email to the operations team.

It doesn't sound all that impressive, and was hardly DevOps, but the results were nothing but positive: our testing deployments became not only instantaneous and perfect, but the interaction with the business was transformative. They could see a simple change they requested turn into a new version of the application, on the test server, in just a few hours. That used to take a week at best.

Our little experiment ended last week when we the director got wind of what we did. Not only did she give it production approval (thus ending the experimental status), but we've been reassigned to take one of the "grown-up" applications down this same path.

This is Initrode's first officially sanctioned foray into DevOps, and now I'm *really* working on the kinds of projects I want to.

inedo

Inedo was founded in 2003 to provide consulting and professional services within the software community. During this period they came to see a need for web based agile release automation suite. In 2008 they began work on BuildMaster, and today they offer not only BuildMaster, but several other pieces of software that are both widely used and widely trusted. Learn more about BuildMaster and ProGet at inedo.com, and by following @inedo on Twitter.

Inedo, LLC
64 Front Street, 2nd Floor • Berea Ohio 44017
440.243.6737 • inedo.com