DZone

DZONE'S 2019 GUIDE TO

DevOps

BROUGHT TO YOU IN PARTICIPATION WITH

bmc

cloudbees

influxdata

MESOSPHERE

NS1.

outsystems

VENAFI

WhiteSource

Dear Reader,

As DevOps enters its tenth year (or eleventh, depending on when you trace it to), there has been no shortage of discussion among developers and enterprises of all sizes who are interested in the increased speed, collaboration, and iteration that DevOps promises. Fueled by the need to satiate the ever-increasing market demand for faster releases while maintaining robust and flexible applications, enterprises continue to seek ways of working that bring together every facet of their software and those who create it. This continued development of ideas around how to effectively fail fast, the importance of automation, and elevating the role of testing has given birth to tools, processes, and services intended to transform enterprises from the ground up.

For the sixth volume of our DevOps Guide, we wanted to return DevOps to its core as a cultural paradigm shift and a way of disrupting fragmented and linear development patterns. The refrain that "DevOps is not just a tool" seems to be taking root as developers witness more concerted efforts on the part of management to implement collaborative processes that bring multiple teams together. Moreover, enterprises are coming to the understanding that reaping the rapid-release benefits from their perspective begins first with recrafting the cultural practices and processes of their people. *DevOps: Implementing Cultural Change* seeks to bridge the gap between the acceptance of that principle and its realization.

We've gathered the expertise of some of our most knowledgeable contributors to dive into the interdependence of teams to create fast iterations, the introduction of new paradigms within DevOps implementations, the value that companies who correctly implement DevOps can see, and starting points for effectively securing your application with DevSecOps. You'll find guidance on what constitutes a roadmap for prioritizing the different steps in a DevOps implementation, as well as industry best practices for getting the most out of your DevOps adoption at every level.

Moreover, you'll find a focus on the effect that the successful implementation of DevOps has on every company's number-one resource: its people. From understanding the interoperability and interdependence of previously siloed teams and what that means for workflow to an in-depth analysis on the expansion and molding of the definition of the term "culture" to include more than championed corporate values, this Guide explores how a cooperative push to introduce DevOps at every level of development shifts the trajectory of a company's software, thinking, and future.

WRITTEN BY ANDRE LEE-MOYE
CONTENT COORDINATOR, DEVADA

Table of Contents

DZone is...

BUSINESS & PRODUCT
Matt Tormollen CEO
Matt Schmidt President
Jesse Davis EVP, Technology
Kellet Atkinson Media Product Manager

SALES
Chris Brumfield Sales Manager
Jim Dyer Sr. Account Executive
Tevano Green Sr. Account Executive
Brett Sayre Account Executive
Alex Crafts Key Account Manager
Eniko Skintej Key Account Manager
Sean Buswell Sales Development Rep.
Jordan Scales Sales Development Rep.
Daniela Hernandez Sales Development Rep.

MARKETING
Susan Wall CMO
Aaron Tull Dir. of Demand Gen.
Sarah Huntington Dir. of Retention Mktg.
Waynette Tubbs Dir. of Marketing Comm.
Ashley Slate Sr. Design Specialist
Colin Bish Marketing Specialist
Lindsay Pope Customer Marketing Mgr.
Suha Shim Acquisition Marketing Mgr.

PRODUCTION
Chris Smith Director of Production
Andre Powell Sr. Production Coord.
G. Ryan Spain Production Coord.
Billy Davis Production Coord.
Naomi Kromer Sr. Campaign Specialist
Jason Budday Campaign Specialist
Michaela Licari Campaign Specialist

EDITORIAL
Susan Arendt Editor-in-Chief
Matt Werner Publications Coordinator
Sarah Davis Publications Associate
Mike Gates Content Team Lead
Kara Phelps Content & Comm. Manager
Tom Smith Research Analyst
Jordan Baker Content Coordinator
Andre Lee-Moye Content Coordinator
Lauren Ferrell Content Coordinator
Lindsay Smith Content Coordinator

Executive Summary

BY KARA PHELPS
CONTENT & COMMUNITY MANAGER, DEVADA

It's safe to say that DevOps has reached the tech mainstream. It's been about a decade since people like John Willis, Patrick Debois, Gene Kim, Jez Humble, and John Allspaw began to rethink the workflows of software development and IT operations. As a movement and a discipline, DevOps has grown steadily in the intervening years. Many organizations are now starting the adoption process out of necessity, while others are learning how to scale and maintain their efficiencies. As more technologies like serverless, IoT, and machine learning are introduced into the feature release cycle, the complexity increases — and so does the apparent need for DevOps.

The DevOps landscape is mature, but it continues to evolve. DZone surveyed 527 tech professionals to learn more about how they implement DevOps, as well as the current challenges and strengths of their teams.

Noticing Continuous Delivery Faultlines

DATA

48 percent of survey respondents reported that their Ops team is involved in the design of their continuous delivery pipeline — a substantial increase from 41 percent last year. Only 14 percent of respondents, however, believe their organization has fully achieved continuous delivery. This is fewer than last year, when 18 percent claimed as much. 28 percent of this year's respondents said they believe their organization has achieved continuous delivery on some, but not all, projects — the same as last year. Meanwhile, 46 percent of survey respondents reported that their team performs push-button or fully automated deployments of any desired version of their software to any environment on-demand. That figure represents a 4 percent increase from last year's 42 percent.

IMPLICATIONS

People are putting the concepts of DevOps into practice (such as Ops involvement in the CD pipeline). At the same time, they're aware of how far they still have to go. Automation fills gaps for some teams, but most would agree that automation alone is not true DevOps.

RECOMMENDATIONS

Continuous delivery is easy to say and hard to do. Awareness of that fact should be seen as a good thing. As knowledge of DevOps concepts grows and takes root in the mainstream, both deepening and spreading out, set aside some time to take stock. How can your team improve build execution time and mean time to recovery? How can you organically bring more collaboration into your company culture?

No Metrics, No Problem?

DATA

42 percent of survey respondents said they do not use metrics to track for continuous integration and/or continuous delivery. 17 percent said they track build times; 11 percent said they track the number of bugs; the rest had a variety of other answers.

IMPLICATIONS

Lots of DevOps professionals do not track their team's progress toward CI/CD. Without data, it's hard to know how well these teams are implementing DevOps concepts and practices over time — whether their goals are meaningful, or set to please management who haven't bought in. And that should be troubling.

RECOMMENDATIONS

Start recording data. The industry still doesn't have an agreed-upon set of metrics to measure DevOps adoption, but that shouldn't stop your team from finding patterns and deciding important benchmarks on your own. Defining key performance indicators for implementation is central to keeping your team on track.

Management Buy-In Increases

DATA

When asked whether management is an enabler or inhibitor of DevOps principles at their company, 54 percent of survey respondents called it an enabler. That's a 6 percent increase from last year, when 48 percent of respondents said management was an enabler.

IMPLICATIONS

Management has mostly bought in to DevOps. As customers demand faster and faster application delivery, DevOps is becoming more and more indispensable. The business case is now, very often, crystal clear. More companies have experimented with DevOps and shared the results — the C-suite has heard of DevOps by now, they've seen the case studies, and they know the potential for increased productivity.

RECOMMENDATIONS

If you're a member of upper management, listen to your employees and do what you can to enable the shift left. In DevOps, the cultural transformation is as important as the transformation of processes and tools. All levels of the company need to be on board. Likewise, if you're a DevOps skeptic in software development or IT, consider that team goals don't diverge nearly as much as you might think. In DevOps, everyone is on the same team.

DZONE.COM/GUIDES

Key Research Findings

BY JORDAN BAKER
CONTENT COORDINATOR, DEVADA

Demographics

For this year's DZone Guide to DevOps, we surveyed developers, architects, and technologists to get their thoughts on all things DevOps. We received 983 total responses, with a 54% completion rating. Based on these numbers, we have calculated the margin of error for this report to be 3%. Below are some basic demographics of our respondents.

- The average respondent has 18 years' worth of experience in IT.
- 36% work for organizations headquartered in the USA and 36% for companies headquartered in Europe.
- 34% of respondents live in Europe, 27% live in the USA, and 11% live in South-Central Asia.
- 25% work for organizations sized 1,000-9,999 employees, 24% work for organizations sized 100-999, and 19% work for organizations sized 10,000+.
- 23% work for software vendors, 16% work in finance/banking, and 7% work in e-commerce.
- 29% work as developers, 24% as architects, and 19% as developer team leads.
- 86% develop web applications/services, 43% develop enterprise business applications, 33% are modernizing legacy applications, and 28% develop native mobile apps.
- 83% work in the Java ecosystem, 70% work in the client-side JavaScript ecosystem, 36% use the Node.js ecosystem, 34% develop with the Python ecosystem, and 28% use the C# ecosystem.

The Architecture of a DevOps Team

Despite the prominence of DevOps in the software industry, less than half of respondents (46%) told us that their organization has an officially designated DevOps team. This lack of an official DevOps team in many organizations manifests itself in the unequal distribution of code deployments. Whereas in DevOps, development and operations teams are meant to work cooperatively to create and release code, 57% of respondents reported that only development teams perform code deployments in their organization. 42% reported this as an operations function and 32% told us that release engineers handle code deployments. Despite these trends, 54% reported management as a DevOps enabler.

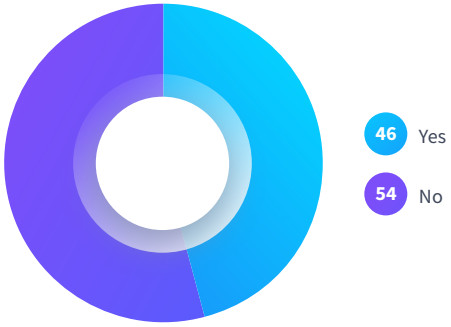
What does a modern DevOps team look like? According to respondents, there are four major aims of a development team when adopting DevOps practices: introducing automation across the SDLC (67%), helping the organization adopt the best CD tools (65%), increasing collaboration and breaking down silos between dev and ops (62%), and developing and delivering software across the entire stack (53%). The purpose behind teams adopting DevOps, thus, has two main considerations: improving software and improving culture. Let's first focus on the ways teams use DevOps to improve the software they produce.

To begin, the majority of respondents' organizations (57%) still rely on the development side of their DevOps pipeline to deploy code to production. This is up from 54% in our 2018 DevOps survey. 42% of respondents' organizations use operations to deploy their code, up from 39% in 2018. As the practice of DevOps continues to grow in influence, one would expect the percentage gap of deploys performed by development and operations to narrow; yet, in each of the last two years, development has performed 15% more deploys than operations among our respondents.

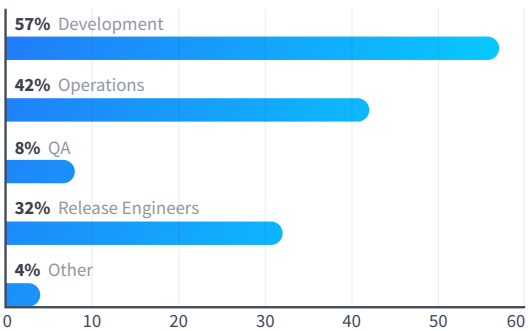
When it comes to the actual software delivery process, survey-takers reported five major processes: code quality checks (62%); breaking

SURVEY RESPONSES

Do you have an officially designated DevOps team in your organization?



Which division of your organization normally performs code deployments to production?



up the build into stages (62%); code reviews (58%); manual checks to proceed (52%); and code coverage checks (50%). Several of these delivery processes saw significant year-over-year growth when compared to our 2018 DevOps survey. The adoption rate of code quality checks among respondents' teams rose by 6%, as did breaking up the build into stages. Code coverage checks, however, witnessed the largest year-over-year growth, increasing by 7% among survey-takers (came in at 43% in 2018).

Given that DevOps processes are meant to increase the agility and efficiency of software development and operations teams, and thus the quality of software, one would expect a certain amount of developer autonomy to come with the adoption of DevOps. When we asked what tasks development teams have autonomy to do with limited or no manual approval from others, 76% of respondents told us making code changes. This proved the most popular answer by far. Respondents also reported a fair amount of autonomy when deploying testing environments (55%). While no other answer received more than 50% of responses, some other interesting responses included creating internal resources (41%), contributing improvements to tooling provided by other teams (35%), and deploying code changes to production (26%).

Now that we've seen how DevOps is used to improve development, let's look into a few ways it's being used to improve the culture around development. As noted earlier, 62% of respondents told us that one goal of their DevOps team is to increase collaboration and break down silos between dev and ops. This goal saw significant year-over-year growth. In our 2018 DevOps survey, 55% of respondents reported the breakdown of silos as a main goal. We also mentioned earlier that 54% of respondents view management as an enabler of DevOps principals. This, t`oo, grew from last year, when less than half (48%) of respondents reported thusly.

It seems that DevOps is helping break down some barriers and also has increasing buy-in from management. We see these two trends greatly influencing the ways in which teams share DevOps best practices and lessons learned within their organization. 36% of respondents reported they share their DevOps findings among individuals within the immediate team, and 29% share this information across teams within the organization.

Developing in DevOps

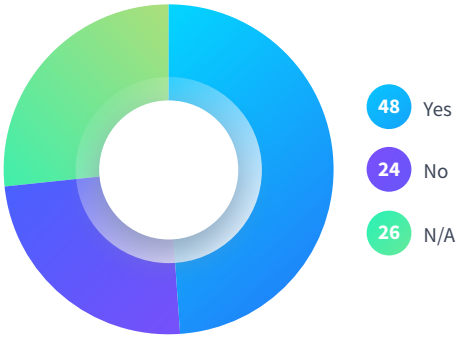
DevOps and microservices are a natural pair, given their impetus on continuous, component-based development rather than a monolithic-based approach. As such, we have seen a growth in the percentage of respondents using a microservices-based architecture for their applications in production environments. In our 2018 survey, 29% of respondents claimed to use microservices in production; in this year's survey, the adoption rate of microservices rose to 39%. We also saw a decrease in the number of respondents not using microservices. In 2018, 31% told us they were not using microservices but were considering them, and another 15% said they were not considering microservices at all. This year, these numbers fell to 27% and 13% respectively. Interestingly, when we compare this data to the two most popular types of developers reported in our Demographics section (web app and business enterprise app), we find web app developers are more likely to use microservices. 42% of respondents developing web applications reported using microservices in production, whereas 38% of business enterprise developers use microservices in production. Despite this slight disparity in microservices adoption in production environments, web app (19%) and business app (18%) developers proved almost equally likely to adopt microservices in development. If you'd like to explore microservices more, check out the 2018 DZone Guide to Microservices.

In order to maintain and commit changes to code, respondents reported two main tools: GitHub and GitLab. Interestingly, despite the enormous popularity of GitHub in the software industry, the percentage of respondents using GitHub fell slightly from 37% in 2018 to 34% for this year's survey. GitLab, however, saw a nice jump in adoption rates, rising from 23% in 2018 to 33% this year. When it comes to the tools used to deploy code, 64% report using the same deployment tool across all phases of the SDLC (development, QA, and production). This rose slightly from last year, when 61% of respondents reported using the same deployment tool across all phases of the SDLC.

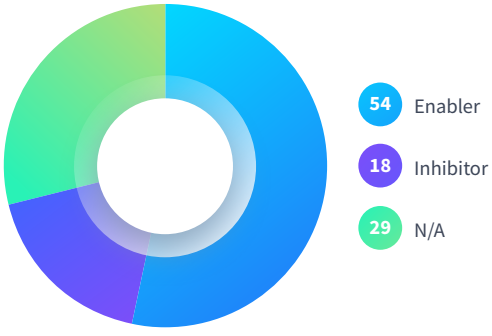
No matter the tools used, almost half (47%) reported their main project's build execution time to be less than 10 minutes. On a more granular level,

SURVEY RESPONSES

Is your Ops team involved in the design of your CD pipeline?



Is management an enabler or inhibitor of DevOps principles at your company?



25% reported a build execution time of two to five minutes, and 22% reported a build execution time of six to 10 minutes. When we compare these numbers to our stats on web app and business enterprise app developers, we find that 28% of web application developers report a slightly faster build execution time. 28% of web app developers told us they have a build execution time of two to five minutes, whereas 23% of enterprise business app developers claimed a two- to five-minute build execution time.

When things go wrong, however, there seems to be two main causes. 75% of respondents told us that application errors/issues were the leading cause for rollbacks and/or hotfixes in their organization, and 52% reported environmental errors or issues. The mean number of rollbacks required among our general survey, however, proved rather low. Averaging out all the responses received, we find that 15% of deployments need rollbacks or hotfixes. And, continuing this positive trend, we find that the typical time to restore service when something goes wrong in production has gone down since last year. In 2018, 24% of respondents reported a time window of less than one hour, and this year, 28% of respondents reported the same time frame.

CI and CD

CI/CD is at the core of DevOps, and both CI and CD constitute two of the most important processes in any DevOps pipeline. When we asked respondents if they believe their organization has achieved continuous integration, 31% said yes, 33% told us on some projects, and 37% said no. When we asked the same question about continuous delivery, only 14% of respondents said yes (down from 19% in 2018), 28% said on some projects, and 58% said no (up from 50% in 2018). Based on these numbers, it appears that approximately two-thirds of respondents feel their organization has achieved a level of comfort with CI processes, but less than half can say the same for CD.

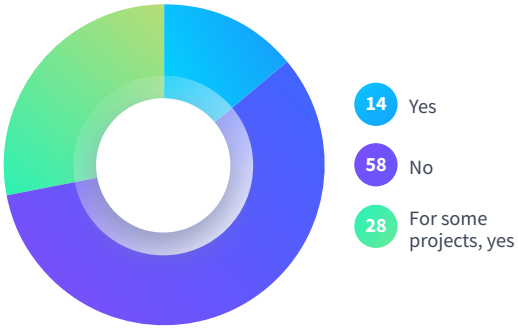
This disproportionate comfort level between CI and CD came through when we asked survey-takers if their CI processes extend into an automated CD pipeline. Of those surveyed, 58% said no. When looking into the pain points of the continuous delivery pipeline that could be behind this disconnect, we find that 51% of respondents report environment configuration setup to be an issue and 30% report user acceptance testing as a problem (up from 25% in 2019). While these were the two most oft-reported issues, automated testing (28%), coordination of team members and resources (27%), and deployment process in use (24%) also came in as popular answers. It's interesting to note that all of the issues listed, outside of automated testing, saw a significant percentage swing over 2018's responses. The number of respondents reporting environment configuration and set-up fell by 4%, the percentage of those reporting coordination of team members and resources fell by 6%, and those reporting deployment processes fell by 6%. While those are all positive signs, respondents who claimed user acceptance testing to be an issue rose by 5%.

There seems to be a few reasons why so many organizations are struggling with the adoption of continuous delivery. Much like we saw with our discussion of obstacles to DevOps adoption, the main barriers to adopting continuous delivery are more cultural than technological. The main hurdle reported by respondents was corporate culture (47%), specifically a lack of collaboration and/or DevOps practices within an organization. In fact, of the four largest barriers to CD adoption given by respondents, corporate culture was the only one that exhibited a year-over-year increase, with 45% of our 2018 survey respondents telling us culture was an issue. Due to these obstacles, team- and organization-wide adoption of CD practices tends to take anywhere from two months to a year. 26% of respondents told us their team's adoption of CD processes took two to six months to complete, and 24% said it took their organization six to 12 months to fully adopt continuous delivery. Compounding these issues, nearly half of survey-takers (42%) said their team/organization does not use any metrics to track their CI/CD processes.

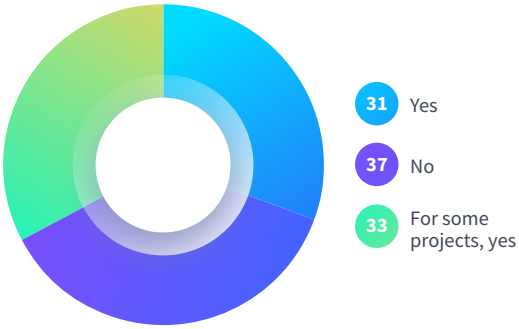
It should be noted, however, that it's not all gloom-and-doom on the CD side of things. The knowledge gap and distance between dev and ops in the CD pipeline are both shrinking. 48% of respondents reported that their ops team is involved in the design of their CD pipeline, up from 43% in 2018. And while 45% of our 2018 respondents said that engineers/ops did not have the right skillsets for working in a CD pipeline, only 36% espoused this belief in this year's survey.

SURVEY RESPONSES

Do you believe your organization has achieved continuous integration?



Do you have an officially designated DevOps team in your organization?



Immutable CI/CD for Resilient Delivery

QUICK VIEW

01. Resilient delivery has become critical as the pace of software delivery keeps increasing.

02. Resilient delivery requires treating the CI/CD system as a product and focusing on operational features besides CI/CD functionality like availability and scalability.

03. Scaling CI/CD as a product requires new approaches such as immutable CI/CD.

04. Immutable CI/CD applies the ideas of immutable infrastructure to CI/CD tooling.

05. With immutable CI/CD, tooling and code updates are delivered by creating and deploying a new instance of the CI/CD system rather than changing a running instance.

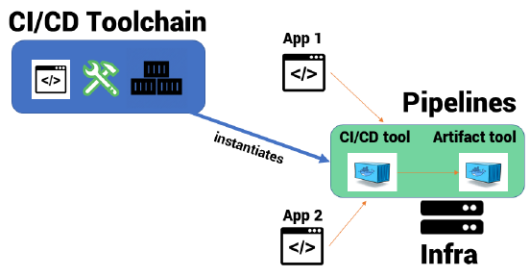
BY MANUEL PAIS
DEVOPS AND DELIVERY CONSULTANT

The main goal of an immutable CI/CD system is to maximize the availability of delivery pipelines (where unavailability could be either due to planned downtime, accidental downtime, or blocking issues in the mechanics of the pipeline).

In this article, I will explain what immutable CI/CD is, as well as the associated practices and benefits. But let's start with what resilient delivery means.

What Is a Resilient Delivery (CI/CD) System?

A delivery (CI/CD) system encompasses all of the tools we use in the delivery process — not only CI and CD but also artifact management, deployment tools, test tools, etc. But the delivery system also includes the underlying CI/CD infrastructure, the source repositories for the applications (as well as the pipeline definitions), and the pipeline orchestration code itself. In short, it's everything that our pipelines need, directly or indirectly, to be able to execute.



A simple example of a delivery system, including tools, repos, infrastructure, and pipelines.

Once we see the delivery system as yet another system in our portfolio, we can start thinking not only about the features we need from it (for example, support for secure code analysis or rolling deployments) but also about our operational requirements (for example, availability expectations or disaster recovery time).

We can define, in general terms, a resilient CI/CD system as a highly available system with a short time to recover from failures and disasters. The exact metrics depend on each organization's context.

Fast and safe software delivery requires high availability and scalability from the CI/CD system. It is no longer acceptable that builds and deployments are halted for a few hours due to CI/CD infrastructure upgrades, tooling updates, or a new plugin installation.

Queuing up multiple small changes because the CI/CD system is not available creates a delivery bottleneck. Once the system is back up, the risk of failed builds, tests, or deployments increases significantly, and negatively affects the team's pace. This, in turn, possibly derails their sprint (or iteration) objectives and the business expectations of the team.

We Have Immutable Infrastructure and Applications; Why Not Immutable Tools?

We have been talking about immutable infrastructure as a concept with well understood benefits and challenges since at least 2013. I find this definition by Josh Stella particularly comprehensive. It has also withstood the test of time:

"Immutable infrastructure provides stability, efficiency, and fidelity to your applications through automation... the basic idea is that you create and operate your infrastructure using the programming concept of immutability: once you instantiate something, you never change it. Instead, you replace it with another instance to make changes or ensure proper behavior."

Docker showed up around that time, as well (a quick fact checking shows Docker was first released in March 2013, and version 1.0 came out 15 months later), and its quickly growing adoption in the following years gave us the tools to quickly build and manage machine image artifacts. They are a key enabler for immutability.

The process of creating a new image whenever an application or service requires a change to the machine (container) where it runs became standardized and deployment of immutable containers (and pods with Kubernetes) became the way containerized applications get updated. Patching or installing new packages inside a container is (correctly) seen as a risk given their ephemeral nature.

So, if immutable infrastructure for applications is now mainstream, why aren't we doing the same for the tools we use? Well, there are a few challenges.

But first, let's acknowledge that modern development and operations tools that run on-prem have been containerized and vendors provide official Docker images for them. That frees us from having to either manually install or automate the install of such tools ourselves.

The challenges with immutability for tools manifest themselves when we get to the post-install administration, configuration, and upgrading activities.

CHALLENGE #1: LACK OF CONTROL

Container images embed the necessary technical stack to run a tool, but we still don't control the tool's internal architecture like we do with our own applications. That means we have to discover and adapt to what we're given when automating the instantiation of the tool.

(Pro tip: When evaluating new tools, consider the ease of adoption within an immutability approach in terms of tool design and logging accessibility.)

For example, it's common to configure things like admin credentials and tool preferences via the UI or interactive scripts. Are those settings stored in a config file? If so, where and in which format? We can run a sandboxed manual configuration to find out. Then, we can create a config file version already customized for our needs. We can then copy this version into the tool container through our Dockerfile.

If the tool was designed to store configuration in an internal database, then this approach becomes more cumbersome but still worth investigating. (Is there an API that we could call to update the database programmatically and avoid inconsistencies? If not, it's probably the right time to switch to a modern tool.)

CHALLENGE #2: MORE MOVING PARTS

Another challenge relates to the multiple aspects that need to be automated to instantiate a tool in immutable fashion.

Besides the aforementioned tool configuration, some other aspects that might need to be addressed include:

- Filesystems and databases where execution data is stored (e.g. pipeline execution logs and artifacts)
- Installation of required plugins (a common need for CI/CD tools)
- Configuration of source and pipeline code repositories
- Installation of any OS utilities we might need to automate the above

Again, we need to understand on a tool-by-tool case, what kind of mechanisms it provides to help us perform these tasks. For example, Jenkins CI provides a shell script that pre-installs a list of custom plugins. In our Dockerfile, we can make use of it:

```
COPY plugins.txt /usr/share/jenkins/ref/plugins.txt

RUN xargs /usr/local/bin/install-plugins.sh <
/usr/share/jenkins/ref/plugins.txt
```

Here, plugins.txt identifies the set of plugins that will be installed by the script; for example:

```
git:latest
job-dsl:latest
workflow-aggregator:latest
simple-theme:latest
blueocean:latest
```

CHALLENGE #3: PRODUCTION DATA

For a CI/CD tool, production data consists of (at a minimum) logs, artifacts, workspaces, and stage/pipeline results.

How do we keep in tact the production data that our tools store when switching to a new version of the CI/CD system? We need to identify where and how the data gets stored first. Then, we need to find an adequate solution that ensures the data is persisted across the different environments where each version of the CI/CD system runs.

For example, CI/CD tools might store their file-based data in the server's filesystem. Using Docker volumes, we can map our persistent filesystem(s) on the container running that server in the new CI/CD version. And, of course, we can and should establish adequate backup and restore policies as well.

What Is Immutable CI/CD?

We can define immutable CI/CD as a delivery system whose toolchain (CI tools, CD tools, plugins, build/test/deploy tools, etc.) does not change once the system is instantiated. Fixes, updates, new tools, new releases, and new plugins can only be introduced by instantiating a new version of the CI/CD system altogether.

Immutable CI/CD provides stability to the delivery system, reducing downtime and unexpected failures due to changes on the fly. When coupled with scalable infrastructure and adequate logging and monitoring, the end result is a resilient, highly available, and scalable delivery system that can meet modern software delivery needs.

There are, of course, some challenges to this approach today.

CHALLENGE #1: UPFRONT INVESTMENT

Earlier in this article, we saw that we need to invest some effort into understanding the architecture of a tool in order to be able to codify and automate all the setup and configuration for our needs. To achieve immutable CI/CD, we multiply that effort by all the tools we are using in the system.

However, in my experience with open-source toolchains, it's really the CI and CD tools that demand most of the work. Tools that do specific jobs tend to have less configuration and apply sensible defaults. That makes them easier to adopt out-of-the-box (where "box" now means an official Docker image).

The upfront investment pays off in droves every time we are able to update the delivery system without disrupting the work of the software delivery teams. It's a gift that keeps on giving.

There is also good news in that vendors and creators are increasingly aware of the need to up their game in terms of configurability and availability of their tools in a DevOps, "everything as code" world. There's a growing awareness that automation and immutability are not just for the applications we build but also for the tools we need to build and deliver those applications.

CHALLENGE #2: ZERO DOWNTIME DEPLOYMENTS

Many CI/CD tools today are still architected around a client/server approach that was not designed with scalability in mind. That makes it difficult to gradually move incoming traffic (build and pipeline requests) to a new instance of the delivery system.

The best approach today is to use blue-green deployments, with a hard switch from the current delivery system version to the new one. If we want to do this during working hours, we need to freeze new build and pipeline requests and either abort or wait for current pipelines to finish execution.

For (near) zero downtime deployments, we would require waiting for the current system to be idle and only then switch to the new version. This might be impossible, however, particularly for development teams distributed across many time zones.

Useful Side Effects of Immutable CI/CD

In my experience, I've found that pursuing the goal of immutable CI/CD brings about beneficial side effects, such as the ability to quickly spin up new CI/CD environments from zero. This is not only useful for disaster recovery but also allows organizations to distribute their delivery system (per team, department, or business unit, for example) instead of centralizing all the pipelines on a single massive server. And because everything is defined as code, propagating changes across multiple instances of the delivery system becomes rather straightforward.

Finally, it promotes testing changes to the CI/CD system in isolation (with the help of example tech stacks and ephemeral environments for deployments to create minimal yet realistic test cases for pipeline functionality).

Getting Started

If you've been feeling the pain of an overloaded and fragile delivery system, get started now. Account for an upfront investment but expect massive gains, especially with a large number of development teams using the same CI/CD system.

If you haven't felt the growing pains yet, at least ensure you communicate clearly when and how the delivery system is being updated. Meanwhile, start creating a "staging" delivery system to smoke test changes in isolation and reduce downtime for the "production" delivery system. Also get started with pipeline-as-code, if you haven't yet.

MANUEL PAIS is an independent DevOps and Delivery Consultant, focused on teams and flow. As member of DevOps pioneers Skelton Thatcher Consulting, Manuel has helped large organizations in finance, legal, and manufacturing adopt test automation and continuous delivery, as well as understand DevOps from both technical and human perspectives. Co-curator of DevOpsTopologies.com. Co-author of the upcoming book "Team Guide to Software Releasability." LinkedIn - Twitter

How Low-Code can Support and Simplify DevOps

As organizations strive to deliver software faster, more reliably, and with fewer errors, many have acquired numerous tools to support DevOps. The result is a bewildering and complex array of products, and the complexity of integrating and mastering all this can be a drain on resources.

An enterprise-grade low-code platform can help by facilitating continuous delivery with less complexity and the support of fewer specialists. Here's a sample of what it can offer DevOps:

- **Version control:** All versions are stored automatically in a central repository. Milestones can be tagged, and version history includes who checked in/checked out work items and when. Rollback is supported, and any prior version can be downloaded.
- **Build validation:** Whenever you deploy, impact analysis checks if the deployment will affect other applications running in the target environment, so you can resolve any conflicts and dependencies.
- **One-click deployment:** One click is all it takes to generate and compile optimized code, analyze databases and create required differential SQL scripts, distribute compiled applications, update, hot-deploy new versions, and synchronize environments.
- **Monitoring:** Comprehensive tools enable proactive management of application performance, so it's easier to detect problems by identifying real-time performance issues.

But that's not all. Microservices, containers, and low-code can all coexist in harmony. With the right low-code platform, IT teams can effectively implement a fully-fledged microservices architecture and leverage leading container technology—for increased scalability, resilience, and portability.

Read the full article on the principles of DevOps.

WRITTEN BY MIKE HUGHES
SENIOR DIRECTOR PRODUCT MARKETING, OUTSYSTEMS

PARTNER SPOTLIGHT

OutSystems Low-Code Development Platform

OutSystems is the #1 low-code platform for digital transformation - build mobile apps, web portals, mission-critical systems, and more.

Category High productivity, low-code application development and delivery platforms.

Release Schedule Annual major releases and monthly feature releases.

Open Source? No

Case Study **Atos** - AMPS™ is a highly sophisticated asset management system that four of the world's navies rely on for everything from tracking vessels, aircraft and munitions to scheduling maintenance and managing personnel. With more than 5,000 users in 120 locations, it's a serious, mission-critical application. So, when Atos, the developer, decided to update this legacy platform with even more functionality to meet the needs of modern navies, it selected OutSystems to support its application development lifecycle from the build phase all the way to deployment. They automated testing, saved significant amounts of regression testing time, and built an enhanced user experience into the product, as well as delivered the solution on any device.

Strengths

1. Visual, low-code full-stack web and mobile application development
2. Change tracking and automatic change impact analysis
3. One-click deployment and continuous delivery for even the most complex apps
4. Integration with everything: Easily connect your apps to any system
5. Advanced ALM and monitoring capabilities for DevOps
6. Support for microservices architecture and container deployment

Notable Users

- FICO
- AXA Insurance
- Logitech
- Toyota
- Randstad

Website
Outsystems.com

Twitter
@Outsystems

Blog
Outsystems.com/blog

DZONE.COM/GUIDES

Mutual Interdependence: The New Normal

QUICK VIEW

01. DevOps is more than a title or a collection of tools — it's about how teams are structured, organized, and ultimately empowered to collaborate with each other.

02. Moving fast is the way to make DevOps work more effectively.

03. Instrumenting your applications to allow better automation and collaboration supports the DevOps ideal.

BY HEIDI WATERHOUSE, DEVELOPER ADVOCATE
AND ANDREA ECHSTENKAMPER, DIRECTOR OF MARKETING AT LAUNCHDARKLY

DevOps is a culture, not a process or a tool. It's a way of structuring teams and thinking about projects so organizations can ship faster and more often. DevOps asks organizations: "What does 'ready' mean?" Ready used to mean a complete product, perfect, ready to ship. Now, smaller teams using DevOps methodologies are shipping smaller releases more frequently, and they're comfortable with the fact that those releases may not be perfect and may require updates. Let's take a look at the ramifications of this cultural shift, and how it changes the way teams approach their role and processes.

Moving Fast, Trying Not to Break Things

As customers, we hope someone has made the app we need so that it's ready the moment we realize our need. The speed and Agile delivery of software has changed our expectations about how often releases and updates happen, and we want them to happen even faster. How are organizations speeding up their development process while releasing more often, faster, and avoiding the inherent risks involved?

Part of it is that the tools available to create software are powerful and continue to evolve in both reach and usability. Consider how far automation has come — from hand-built scripts and Jenkins to the automation and compliance of Habitat. Now, instead of reading log files, we're using Splunk and New Relic and Honeycomb to delve into the connections between services and trace messages through complex architectures. But that's just one part of the puzzle. No matter what tools we have in hand, if we don't change how teams operate, we will still have bottlenecks. Before many of these tools even became mainstream, teams began adopting practices that would help them deliver faster.

One of the earlier methodologies was Agile. The most basic goal of Agile was to get rapid feedback and incorporate that information at the point of change. Taking that a step further, continuous integration and continuous delivery (CI/CD) exponentially decrease the time to delivery by making very small incremental changes every few days or even minutes. Specifically, continuous delivery is a set of practices that ensure your code is always in a deployable state. You accomplish this by increasing the frequency at which code is committed, built, tested, and deployed — steps that used to happen at the end of a project when it was "code complete." Because your team is delivering frequently, every change is smaller, and those changes are more isolated and less risky. Problems are identified more readily, and bugs are fixed more easily because they are discovered immediately.

You Got Ops in My Dev

Teams are taking Agile and CI/CD practices and rethinking their collaboration habits. When development and operations are siloed from each other, one person builds, another QAs, and a third releases. It can take several transfers of ownership and considerable time to fix an issue after it is discovered.

Working in an Agile or CI/CD framework, teams find value in holding development accountable for their code up until the point of release. Maybe that person should be more involved in testing,

validating, and releasing the feature they worked so hard to build. After all, this could help processes move along even faster. This blurring of rigid job identities became known as DevOps. And DevOps is more than a title or a collection of tools — it's about how teams are structured, organized, and ultimately empowered to collaborate with each other.

"[DevOps] is a firm handshake between Development and Operations that emphasizes a shift in mindset, better collaboration, and tighter integration. It unites Agile, continuous delivery, automation, and much more to help development and operations teams be more efficient, innovate faster, and deliver higher value to businesses and customers." - ***Atlassian***

With a more collaborative outlook, DevOps teams leverage Agile and CI/CD methods to move even faster. Individuals are no longer focused on their one piece of the puzzle, but instead see the whole picture and understand how they can make an impact on it. When a developer begins to build, they are thinking about the entire development and release process. They are either setting up the tests themselves or working closely with their QA team to make sure that feature is delivered without delay.

Checkpoints, Control Points, and Points of Interest

As DevOps practices have become more widely adopted, tooling has been developed to support these collaborative workflows. Most recently, feature management platforms offer more control of elements in the release process. Instead of a release manager coordinating things at the moment of release, features can be deployed, tested, and then handed off to the release or business teams for full activation.

Moving fast seems risky if you are coming from a model where releases are large and involve many parts. Even a simple product update can turn into a complete disaster that takes hours or days to repair, and having to roll back code makes the process even more painful. However, as Forsgren, Humble, and Kim argue in their book *Accelerate*, moving fast is the way to make DevOps work more effectively.

Feature management platforms let teams determine how users will experience their products and configure different experiences for different user groups using the same code. These teams are using feature flags to separate the act of deploying code from releasing a feature. This means they decide when a feature is turned on, who will experience it, and how to precisely target it.

With these control points and safety valves in place, DevOps is empowered to function collaboratively. Teams can check their code in early to reduce long-lived branches, they can test their code and validate their ideas, and they can protect their services and their users as they roll out new features. They can even instrument microservices with circuit breaker flags that will prevent cascade failures. Moving fast is not as risky anymore — if something goes wrong, you can simply turn the flag off.

While DevOps teams are enriching their development and release processes with stronger controls, they often find this level of control makes it easier to include other teams within their organizations in critical points of the processes. Once a feature flag is in the codebase, a feature management platform allows anyone to control it without needing to directly access the code. DevOps can invite Product or Marketing to participate in releasing a feature.

Engineering no longer needs to operate on strict timelines dictated by external business calendars. They can deliver code when it's complete and share the controls for releasing that feature. This means Marketing can release when they're ready to launch their campaign, Sales can share new features with prospects, and Customer Success can quickly and easily assist customers.

Seize the Means of Release!

This is where collaboration between Engineering and other business teams only begins. When these business teams are given control over these features, they can do more than just turn them on/off they can also become more involved in testing and validation processes.

Business teams like Product, Design, or Marketing can now be an integral part of running a beta test — from determining who will be involved, to actively collecting feedback and validation. Suddenly, they can become actively involved in building new features, helping to garnish valuable feedback and making sure it's what the user base really needs.

With the controls found in DevOps tooling — like feature management platforms — team members across organizations are now becoming active participants in the development and release processes. What would change in your workflow if you could choose when and how to deliver a feature?

HEIDI WATERHOUSE is a developer advocate. She is working in the intersection of risk, usability, and happy deployments. Her passions include documentation, clear concepts, and skirts with pockets. LinkedIn

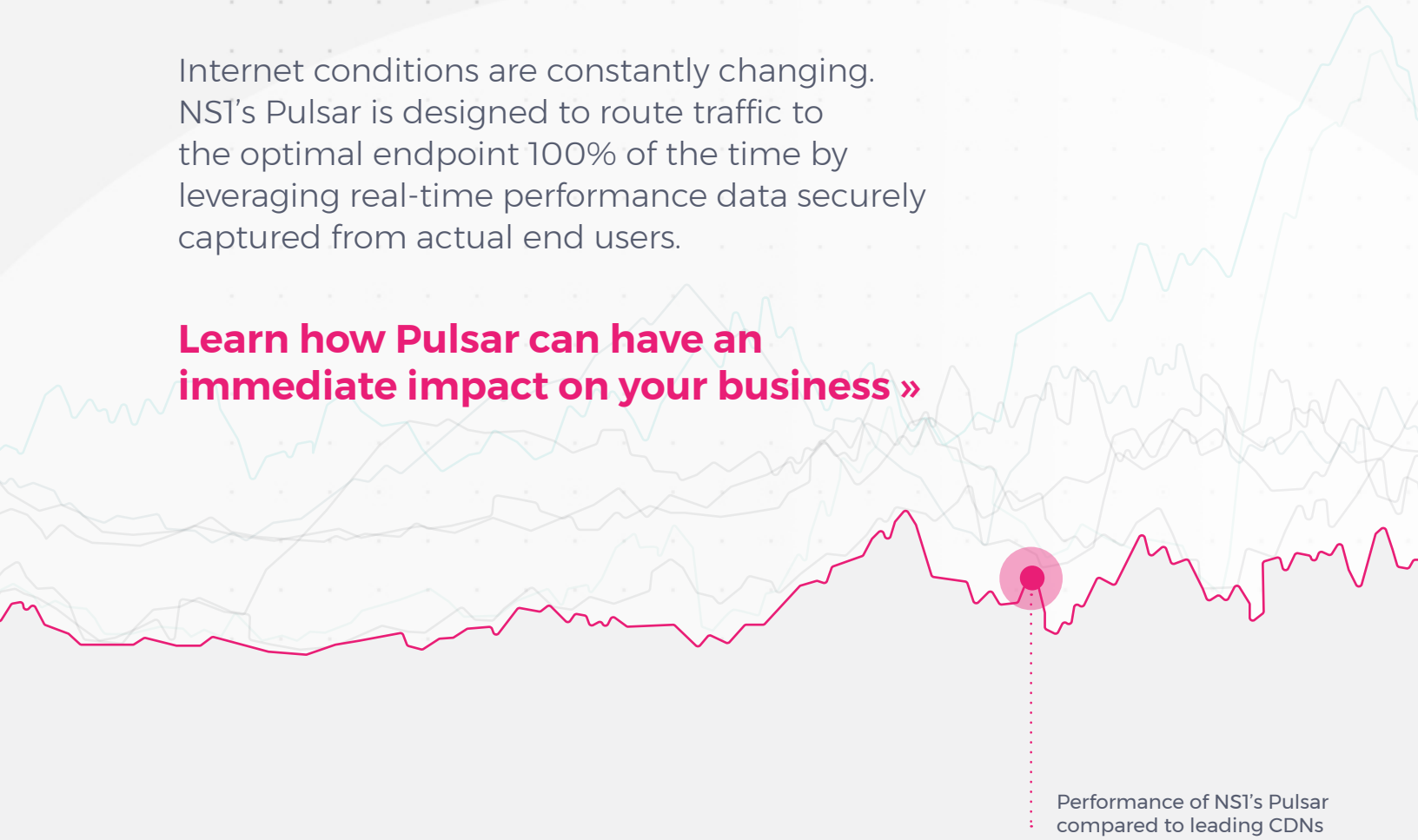
ANDREA ECHSTENKAMPER is the Director of Marketing at LaunchDarkly and the organizer of the Test in Production Meetup. She's into community and ecosystem marketing, developer-focused companies, bioluminescent sea creatures, and learning new things. LinkedIn

NS1.

How would a 19% reduction in latency affect your business?

Internet conditions are constantly changing. NS1's Pulsar is designed to route traffic to the optimal endpoint 100% of the time by leveraging real-time performance data securely captured from actual end users.

Learn how Pulsar can have an immediate impact on your business »



Performance of NS1's Pulsar compared to leading CDNs

Learn more about Pulsar RUM DNS Steering
https://ns1.com/pulsar

USA TODAY NETWORK · yelp · jetBlue · the guardian · LinkedIn · salesforce · SQUARESPACE

Next-Generation DNS Powered by Real-Time User Data

The way we think about performance, reliability, security, and manageability for internet facing applications is always changing. It's long been established that milliseconds matter. Uptime is an obvious imperative for any online business. And increasingly, companies reaching their audiences online must achieve bleeding edge performance and reliability without compromising on the security profile of their applications, and do all this with smaller teams driven by automation, not brute force.

At NS1, we've long explored the nexus of performance, reliability, security, and ease of use, and how DNS, the entrypoint to every online application, can impact these imperatives.

That's where Pulsar RUM DNS Steering comes in. Pulsar enables customers to achieve truly bleeding edge performance and enable reliable and secure application delivery architectures, more easily than ever before.

Pulsar enables customers to achieve truly bleeding edge performance

Pulsar directs traffic based on real-time performance data captured from actual end users with respect to the CDNs, cloud providers, and other edge systems you use to deliver your application. Even the most widely-used CDNs and cloud providers exhibit variable performance throughout the day and in different markets around the world. Pulsar is always watching, and it observes availability and performance issues as they appear in real-time for any ISP and region in the world, directing traffic across your edge to optimize uptime and speed while minimizing cost. Pulsar acts on billions of daily data points, and you can easily bring along data of your own. The data that Pulsar steers with, and the steering decisions Pulsar makes, are incredibly valuable decision-making tools for your organization, and we've worked hard to make exploring the rich Pulsar dataset easy and intuitive.

WRITTEN BY KRIS BEEVERS
CEO & FOUNDER, NS1

PARTNER SPOTLIGHT

NS1 Pulsar DNS + RUM Steering

NS1.

The only RUM steering solution integrated with authoritative DNS

Category Next-Generation DNS

Release Schedule Continuous

Open Source? No

Case Study What if you could decrease latency without needing to invest in data centers, servers, and more complex infrastructure? That's exactly what Magnetic, an AI-powered advertising platform, did.

Leveraging real user monitoring data and intelligent DNS traffic routing, Magnetic was able to re-allocate existing resources to where they would have the greatest impact: moving infrastructure to the edge for actual (not perceived) usage patterns.

This brought a 70% decrease in latency without needing to deploy additional servers or new vendors.

No one company controls the entire application delivery infrastructure, but one strategic technology – DNS – intersects it all.

That's where NS1 comes in. Read the full case study at ns1.com/magnetic.

Notable Users

- LinkedIn
- SalesForce
- SquareSpace
- Dropbox
- Yelp

Website
ns1.com

Twitter
@ns1

Blog
ns1.com/blog

Testing in CI

QUICK VIEW

01. When writing automated tests, speed, reliability, quantity, and maintenance are crucial in order to work with continuous integration.

02. Automatically testing as close to production-ready code as possible ensures that continuous integration can continue to be fast and efficient.

03. Flaky tests that deliver false positives can derail continuous integration efforts, so it is important to ensure tests deliver high-quality results so developers don't have to search for the issue.

BY ANGIE JONES
SENIOR DEVELOPER ADVOCATE

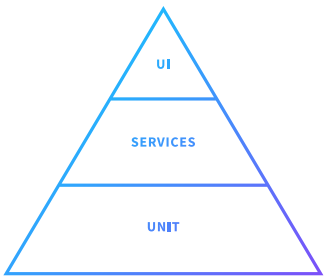
Automated tests are a key component of continuous integration (CI) pipelines. They provide confidence that with newly added check-ins, the build will still work as expected. In some cases, the automated tests have the additional role of gating deployments upon failure.

With such a critical responsibility, it's important that the automated tests are developed to meet the needs of continuous integration. For this, there are four main factors that should be considered when developing automated tests for CI: speed, reliability, quantity, and maintenance.

Speed

A major benefit of CI is that it provides the team with fast feedback. However, having a suite of automated tests that take hours to run negates that benefit. For this reason, it's important to design automated tests to run as quickly as possible.

One way to accomplish this is to automate the tests as close to the production code when possible. The test automation pyramid, a model introduced by Mike Cohn in *Succeeding With Agile: Software Development Using Scrum*, describes three levels in which tests can be automated: unit, services, and UI.



Automation Pyramid

UNIT LEVEL

The unit level is closest to the production code. Automated tests at this level are quick to write and quick to run. More importantly, they are able to pinpoint the exact function in which a bug exists. Therefore, the bulk of the tests that are run as part of CI should be written at this level. This aligns with the goal of fast feedback.

SERVICES LEVEL

The services level is a bit further from the code itself and focuses on the functionality that the code provides, but without a user interface. Tests automated at this level can make calls to the product's APIs and/or business logic to verify the integration of various individual functions. This level should contain the second largest number of automated tests after the unit level.

UI LEVEL

The furthest level from the production code is the UI level. Tests written at this level face unique challenges in that they take longer to write, take longer to execute, and are dependent upon the consistency of the UI. Because of this expense, this level should contain the least number of automated tests.

When considering a test for automation to be included in CI builds, determine what information this test needs to verify and choose the lowest level of the pyramid possible to write the test against. The lower the level, the faster the test — thus upholding the goal of fast feedback.

Reliability

Automated tests that are run as part of CI need to be highly reliable. The state of the build is dependent upon the state of the tests, and manual intervention is impractical. Yet, reliability is often something teams struggle to achieve within their automated tests.

Flaky tests — tests that sometimes pass and sometimes fail when ran against the same code base — are a detriment to CI. False negatives require developers to investigate the failures to ensure it's not their code check-in that is at fault. This is time taken away from other development activities. And if there are too many false negatives, the team begins to lose trust in the tests and begins to disregard them altogether. False positives can have just as much of an impact, as they mistakenly inform the team that the integration is alright when in reality, it is not. These types of tests can allow bugs to creep into production.

There are techniques that can be employed to minimize flakiness within automated tests.

IMPROVE TEST AUTOMATABILITY

One reason why so many tests are written at the UI level is because there is a lack of code seams to allow for automating at lower levels. Code seams are exposed shortcuts that your tests can take advantage of to bypass the everchanging UI.

For example, if a test scenario is to verify that the quantity of a product can be increased within an online shopping cart, the steps to search for the product and add the product to the cart are prerequisites but immaterial to the verification of this specific scenario. Yet, these steps would often be automated via the UI as part of the test. There are a lot of things that can go wrong here before the test ever gets to what it is supposed to be verifying: increasing the quantity in the cart. Code seams, such as a web service to add the product to the cart, can help remove these unnecessary UI actions and reduce flakiness.

Another way to improve test automatability is to ensure that your application's UI elements contain reliable identifiers. These identifiers are used as locators within test automation to interact with the UI elements. Without a dedicated identifier, the test is forced to use XPath or CSS selectors to access the element. These are very fragile and will change as the appearance of your application does; meaning it will break your UI automation.

MANAGE THE TEST DATA

Tests should not make assumptions about the data that it relies upon. Changes with test data are a common cause for failed tests. This gets particularly flaky when there are other tests running in parallel that are modifying that data.

Avoid hardcoding expected assertions. Either create whatever data is needed as part of the test itself using code seams or consult a source of truth (e.g. API/database) for the expected result.

ACCOUNT FOR ASYNCHRONOUS ACTIONS

Tests automated at both the service and UI levels need to account for asynchronous actions. Scripts execute scenarios a lot faster than humans do, and at the point of verification, the system may not yet be in its expected state, which leads to flaky tests. This can be circumvented by having the automated test wait conditionally for the application to be in its desired state before taking the next action. Avoid hardcoding waits, as this can lead to waiting longer than needed and defeating the purpose of fast feedback within CI.

Quantity

To keep the CI process as fast as possible, be mindful of which tests you're adding to it. Every test should serve a distinct purpose. Many times, teams will aim to automate an entire regression backlog or every single test in a sprint. This leads to longer execution times, low-value tests blocking integration, and increased maintenance efforts.

Evaluate your scenarios and determine which are the best candidates for automation based on risk and value. Which ones would cause you to stop an integration or deployment? Which ones exercise critical core functionality? Which ones cover areas of the application that have historically been known to fail? Which ones are providing information that is not already covered by other tests in the pipeline?

Even when being very intentional about the scenarios that will be a part of your CI pipeline, the suite of automated tests will naturally grow. However, you do not want your overall build time to also increase. This can be remedied by running tests in parallel, which means none of your tests should be dependent on each other. For extremely large suites, the tests can also be grouped by key functional areas. Grouping the tests this way allows the flexibility of having CI only trigger the execution of specific groups that may be affected by the development change made, and running the other groups once or twice a day as part of a separate job.

Also, don't be afraid to delete tests that are no longer of high value or high priority. As your application evolves and certain areas prove to be very stable or even underutilized, the need for certain tests may vanish.

Maintenance

Your test code requires maintenance. There is no getting around this. The test code was written against your application at a given state, and as the state of your application changes, so must your test code.

The CI test results must be consistently monitored, failures must be triaged, and test code must be updated to reflect the new state of the application.

Avoiding this step leads to untrustworthy builds where the tests are no longer useful to the overall CI process.

ANGIE JONES is a Senior Developer Advocate who specializes in test automation strategies and techniques. She shares her wealth of knowledge by speaking at software conferences all over the world, teaching courses, and writing tutorials on angiejones.tech. Angie is known for her innovative and out-of-the-box thinking style which has resulted in more than 25 patented inventions in the US and China.

Get certificates faster with fewer interruptions

Automate the provisioning of X.509 certificates within your CI/CD pipelines

With Venafi, it's easy.

- Automates X.509 certificate processes
- Works with your DevOps tools
- Simplifies DevOps security compliance
- Integrates with any certificate authority

venafi.com/DevOps-Solutions

VENAFI®
Machine Identity Protection

Featured Integrations

kubernetes Vault Terraform CHEF

Make X.509 Certificate Processes Compatible with DevOps

Companies are adopting DevOps to speed up software production. However, they have struggled to adapt X.509 certificate processes to keep pace. Certificates are needed to secure networks and applications, but their issuance should not be at the cost of speed.

New Machine Types Must Be Secured

In DevOps, new machines are continuously instantiated through infrastructure-as-a-service and containerization. Relying on load balancers and firewalls to protect machine-to-machine communication may seem sufficient, but because of sophisticated attackers who are able to pivot within a network, stronger security is required. TLS certificates are needed to serve as machine identities to enable encrypted machine-to-machine communication within modern infrastructure.

InfoSec Still Needs Compliance

Developers and IT Ops teams must move quickly, but InfoSec teams and regulatory frameworks still require compliance with security standards. DevOps teams often struggle to comply with certificate policies and deliver audit trails to InfoSec without sacrificing speed. To achieve visibility and compliance, these teams must work together to integrate trusted certificates into machine build and deployment processes.

Automation Makes It Fast and Easy

DevOps teams don't have to slow down or choose between agility and security. As the leader in machine identity protection, Venafi supports DevOps teams with automated, policy-driven certificate procurement and installation. This enforces certificate security policies within CI/CD pipelines and modern infrastructure, provides visibility into issued certificates for outage prevention, and delivers automated, auditable security that doesn't sacrifice speed.

WRITTEN BY SANDRA CHRUST
SENIOR PRODUCT MARKETING MANAGER VENAFI

PARTNER SPOTLIGHT

Venafi Platform

VENAFI®

Venafi bridges the gap between InfoSec and DevOps by delivering automated, policy-compliant SSL/TLS certificates for DevOps use cases.

Category Machine Identity Protection

Release Schedule Quarterly

Open Source? No

Case Study As a top global retailer moved from waterfall to agile development, its DevOps teams struggled to protect the machine identities of the many cloud instances, microservices and APIs it was developing. Provisioning a certificate was a five-day, manual process that slowed its DevOps processes. It needed a machine identity protection solution that would be stable, fast and easy to consume.

The retail leader turned to the Venafi Platform to automate certificate issuance and renewal—a switch that dropped the provisioning of certificates from days to mere minutes. The Venafi Platform integrates seamlessly with the company's commonly used DevOps tools, further improving the processes necessary for an agile, innovative development environment.

Strengths

1. Automates policy-driven DevOps certificate provisioning.
2. Incorporates trusted certificates into CI/CD pipelines.
3. Integrates with popular DevOps tools and all certificate authorities.
4. Standardizes PKI across DevOps environments and teams.
5. Provides visibility for audit, compliance, security and expiration tracking.

Partner Integrations

- Kubernetes
- HashiCorp Vault
- Terraform
- UrbanCode
- Chef

Website
venafi.com

Twitter
@Venafi

Blog
venafi.com/blog

DZONE.COM/GUIDES

6 Keys to Unlock the Value of DevOps

QUICK VIEW

01. To do DevOps right, an organization needs to build multiple bridges — linking people, processes, tools, and organizational goals.

02. Investing in a faulty DevOps project wastes money, time, and employee resources that could have been directed elsewhere.

03. Successful DevOps implementations can generate significant value, including happier employees, increased productivity, and speed.

BY BRIAN DAWSON
DEVOPS EVANGELIST, CLOUDBEES

When people talk about today's software, two things are clear:

1. It's driving real value in businesses, big and small, helping them transform processes and create whole new app-driven business models.
2. The IT organizations in the market-leading businesses are committed to delivering software the right way, embracing the spirit of modern practices like continuous integration, continuous delivery, and DevOps.

What's less clear is whether organizations are following the principles these practices indicate. We've looked closely at the finer points of continuous integration and continuous delivery, and concluded that many organizations think they're performing these practices the best way possible. But on further inspection, they aren't.

What about DevOps? Are organizations doing DevOps right? If they are doing continuous integration and/or continuous delivery right, can they still fail to fully check all the boxes for DevOps? Once again, these questions are important for organizations to sort out as they try to optimize their performance in an economy increasingly driven by software innovation.

Overall, doing DevOps correctly can be a huge catalyst for an organization. But doing it incorrectly can be counterproductive and can cost the organization in a number of ways. Here's a look at what DevOps is and how to do it right.

Defining DevOps

I like to think of DevOps, continuous integration, and continuous delivery as being like software delivery siblings — all descended from the agile and lean movements that aim to streamline processes.

There are important differences. Continuous integration focuses on processes like automated builds and quick changes on the left side of the delivery process. Continuous delivery covers best practices moving through the pipeline up to production. DevOps focuses on the organizational change needed to support collaboration between functions. But these concepts all tie together.

At a high level, DevOps is a cultural term. You achieve DevOps when you create a software development culture supported by technical practices. Within a DevOps culture, software development stakeholders have mutual understanding and empathy, such that they align on the shared objectives of releasing quality software rapidly and repeatedly.

Can you do continuous integration and/or continuous delivery right, and still fall short on DevOps? Absolutely. Continuous integration and continuous delivery are practices that you can implement — and even, to a certain extent, master — while not truly effecting the cultural change necessary to become a DevOps organization.

On the other hand, you really can't do DevOps right without incorporating the building blocks of successful continuous integration and continuous delivery practices. As Jez Humble likes to say, you could implement DevOps processes using a Bash script — DevOps does not inherently prescribe the component practices. But, to implement DevOps in a sustainable, repeatable, and scalable manner requires commitments to continuous integration and continuous delivery.

Doing DevOps Right

A lot has been written in recent years attempting to identify signs that your organization is either doing DevOps right or wrong. Some of the more popular traits assigned to effective DevOps organizations are that they have clearly defined DevOps strategies and can set up "common-sense" measures to identify when things go wrong. Some of the key flags for bad DevOps implementations include having no tolerance for failure and trying to roll in testing processes late in the implementation.

I would boil down this assessment to the following six recommendations, all focused on achieving a DevOps culture:

1. **Define a holistic process, from end-to-end.** Successful DevOps cultures focus on the business, developers, testers, and IT operations teams working together around shared objectives. They give their input about how to start the process, build a successful pipeline, implement a delivery process, and measure the outputs.

2. **Successfully achieve DevOps first across one team.** Before you try to achieve DevOps across your organization, you need to make sure it works on a microscale, ensuring there is a bottom-up adoption and buy-in from key managers. Management buy-in can be difficult and time-consuming at the start of an implementation, but it will be required to successfully achieve an organization-wide transformation. It's going to affect timelines, budgets, and organization.

3. **Make sure your organization is free of silos.** Not all organizations can eliminate organizational silos. But by involving and aligning all stakeholders in the delivery process across departments, you can have virtual (or actual) product teams, reducing the impact of organizational silos. This will help establish empathy between the different teams in the organization, and drive a culture of collaboration and feedback.

4. **Ensure the right people have seats at the table.** If you try to create a DevOps strategy with the input of every single team and functional group within the organization, you may get mired in overwhelming complexity — analysis paralysis — and never really get started. To get your transformation going quickly, it is best to reduce the scope. Focus the initial implementation on one to a few projects. Once you are successful with those, begin incrementally scaling DevOps across teams, incorporating new input along the way. I recommend you start by gathering representatives such as management and an individual contributor from each of your processes and groups: a management rep from the business, a project management administrator, a lead developer from a development team, and an actual developer. Individual contributors are integral, as they are the only ones who can truly identify the pains of the experience and what should be improved. This also creates the necessary empathy and buy-in on the ground.

5. **Don't try to "buy DevOps."** Some organizations operate on the belief that they can buy a set of tools that will make them into a DevOps organization. Don't fall for this trap. Tools are a critical part of connecting teams across silos and enabling the automation that supports DevOps, but tools alone will not solve your problem. You need to do all the hard work creating the culture and looping in all the important aspects of continuous integration and continuous delivery to ensure the tools are doing their jobs.

6. **Make sure you're successfully measuring your progress.** An effective DevOps culture includes a commitment to measure results and report on quantifiable metrics. To stand up your delivery pipeline, you need to build in the ability to do these things and continually track your progress toward those goals.

For example, less-than-optimal software delivery processes tend to lead to significant overtime. Operations engineers and other team members tend to work long hours to get software deployed. So, measuring team hours with a goal to reduce overtime man-hours by 50 to 75 percent will instill the right amount of discipline and help promote a culture of excellence.

The Downside of Bad DevOps

There are consequences to doing DevOps wrong:

- **First and foremost, you're going to miss opportunities.** Today's market is moving fast, and every company surely has competitors already implementing DevOps correctly. They're innovating and gaining in market advantage. If you spend time focusing on a failed or misplaced implementation of DevOps, it comes with a significant opportunity cost.
- **There also are internal costs.** If a DevOps initiative flops, morale suffers and IT leaders may have difficulty getting the buy-in required for another DevOps transformation.
- **Then you have the actual costs.** Investing in a faulty DevOps project wastes money, time, and employee resources that could have been directed elsewhere.

The Upside of Good DevOps

On the other hand, successful DevOps implementations can generate significant value in the following areas:

- **Happier employees.** Creating an efficient, committed DevOps organization can help companies maintain and recruit top talent.
- **Increased productivity**. Less time wasted in the software development lifecycle translates into improved value in the software stream.
- **Speed**. The ability to innovate faster while maintaining quality gives a company a tremendous competitive advantage in the marketplace.

Conclusion

To do DevOps right, an organization needs to build multiple bridges — linking people, processes, tools, and organizational goals. This is hard work. It cannot be done quickly. But it can be achieved if organizations commit to creating a proper DevOps culture. Don't just show up one day and say you're going to do DevOps. Use the concepts of continuous integration and continuous delivery as building blocks, and get the rest of the organization to rally around your vision. Establish where you are today, where you want to go, and how you're going to get there. Then you're on your way!

BRIAN DAWSON is currently a DevOps evangelist and practitioner at CloudBees, where he helps the open source community and CloudBees customers in the implementation of Agile, continuous integration (CI), continuous delivery (CD), and DevOps practices. Before CloudBees, Brian spent over 22 years as a software professional in multiple domains including QA, engineering, and management. Most recently, he led an Agile transformation consulting practice helping organizations small and large implement CI, CD, and DevOps. LinkedIn

DZONE.COM/GUIDES

Monitoring Maturity Model

Emerging trends such as microservices, containerization, elastic storage, software defined networking, and hybrid clouds all keep pushing the boundaries of what constitutes DevOps monitoring. Your monitoring architecture should:

- Improve observability for all by facilitating the collection of metrics and events
- Deliver a real-time pipeline
- Make it simple to capture events
- Provide long-term retention of metrics and powerful search and visualization
- Provide a unified system for monitoring metrics and events
- Deliver high availability, scalability, and performance

Why InfluxData

- **Addresses new workload requirements:** InfluxData can handle a high volume of real-time writes, is purpose-built from the ground up for irregular (events) and regular (metrics) time series data, and offers retention policies for performance and availability.
- **Delivers time-based queries:** Specifically, functions such as aggregation and summation using time-based functions directly in an SQL-like language, and allows for large-range scans of many records very quickly.
- **Provides scalability and availability:** Is a distributed-by-design, multi-instance deployment to allow you to have no single point of failure.

We all aim at that monitoring nirvana that often seems unattainable, but with the relevant conversations and the right tools, it becomes a strategic action plan you can attain.

WRITTEN BY MARK HERRING
INFLUXDATA, CMO

PARTNER SPOTLIGHT

InfluxData

The modern engine for metrics and events

Category Time Series Database **Release Schedule** Quarterly Release cycles **Open Source?** Yes

Case Study Coupa Software needed to create a custom DevOps Monitoring solution for their leading spend management cloud platform. With InfluxData they moved from pure data collection to predictive analytics and achieved a consistent track record of delivering close to 100% uptime SLA across 13 major product releases and 5 major product module offerings, as well as solving their data accessibility, aggregation, and retention challenges. Operational metrics are collected via Telegraf, stored in InfluxDB, and analyzed by Kapacitor. They use Grafana for visualization and have created a custom alerting framework. This has become the foundation to the path of building a system that is self-healing and can provide predictive analytics key to accurate forecasting.

Strengths

1. Fastest Time to Awesome
2. Developer Happiness
3. Ease of Scale-Out & Deployment

Notable Customers

- Wayfair
- CERN
- Coupa
- Mulesoft
- Nordstrom

Website
influxdata.com

Twitter
@InfluxDB

Blog
influxdata.com/blog

DevSecOps: Securing Software in a DevOps World

BY TANYA JANCA
SENIOR CLOUD SECURITY ADVOCATE, MICROSOFT

QUICK VIEW

01. DevSecOps is the combination of application security and DevOps; however, it's an emerging field with little to no standardization.

02. Organizations can — and should — apply security best practices to every stage of their system development lifecycle.

03. DevOps teams of any size and maturity can use the five tactics shared in this article as a baseline to start creating infinitely more secure applications.

The practice of improving and ensuring the security of software is generally referred to as (the field of) application security, or "AppSec" for short. In a traditional waterfall system development lifecycle (SDLC), AppSec was often an afterthought, with someone (a penetration tester) being hired to come in just before release to perform last-minute security testing, or not at all. Slowly, many development shops started adding more AppSec activities such as secure code reviews, providing secure coding guidance or standards, giving developers security tools, and introducing many other great ideas that improved the overall security of the end product. Some companies even went so far as to create their own team dedicated to application security. However, there is currently no agreed-upon standardization on what defines a complete AppSec program, nor a definition of when someone can say that "the job is done" or that they have done "enough" in regard to the security of software. The line seems to vary greatly from team to team, business to government, and country to country, which makes it a difficult thing to measure.

Our industry is now moving toward acceptance of DevOps as the new norm: people, processes, and products with the aim of better resilience, faster fixes, and extremely quick time-to-market. Although the most visible sign of DevOps tends to be an automated continuous integration and/or continuous delivery pipeline (CI/CD), there is actually quite a bit more to creating a DevOps shop and culture.

First, a quick review of The Three Ways of DevOps, as stated in *The DevOps Handbook*:

1. *Prioritizing the efficiency and speed of the entire software creation system, not just your part(s). This means that if you work very hard to make your area of responsibility more efficient, but there are other areas delaying the system as a whole, your work has not added value.*
2. *Working to shorten and amplify feedback loops so that flaws (design issues) and bugs (code issues) are fixed as early as possible, when it's faster, cheaper, and easier to do so.*
3. *Continuous learning: Allocating time for the improvement of your daily work.*

DevOps + Application Security = DevSecOps

As the industry has changed with the move towards DevOps, AppSec has had to change as well, weaving itself through The Three Ways to ensure that the highest quality software is produced. With dev and ops now working hard to increase the efficiency of the entire system (the first way), security has had to learn to sprint alongside them, adding security checks to the pipeline and breaking their activities into smaller and faster pieces. With the new requirement for faster feedback (the second way), the security team can no longer come in at the end of the SDLC. They need to be present from the start. Lastly, DevOps requires a culture of constant learning (the third way), experimentation, and risk-taking, which means making every moment a teaching moment for the security team. Adding security to The Three Ways — performing AppSec within a DevOps environment — is often referred to as DevSecOps. With this in mind, this article will take a look at some tactics to address the first and second way, using tools to change our DevOps pipeline into a DevSecOps pipeline.

DevSecOps in Practice: Five Ways to Build Your DevSecOps Pipeline

TACTIC #1: WEAPONIZE YOUR UNIT TESTS

The first tactic is weaponizing your unit tests; you already have them, so why not reuse them? A regular unit test is generally a "positive test," meaning that it verifies that your code does what it *should* do. Let's say you're reading records from a table in a database. You would likely want to ensure that you can search for one, many, or zero records. Those are generally *positive* use cases — making sure it does what you hope it will do. Now, let's look at *negative* use cases. What sorts of attacks are possible in this situation? Will your application handle attacks (and failures) gracefully? Some standard payloads (malicious requests) that could be added would include a single quote, two single quotes, a double quote, and appended calls to commands (create, delete, update, union). If the application issues

an unhandled error or reacts in a different way than expected, it fails the test and breaks the build. For example, if one single quote creates a different reaction in your application than two single quotes, that means that you are communicating directly with the database and under no uncertain terms should you allow that code out the door.

TACTIC #2: VERIFY THE SECURITY OF YOUR THIRD-PARTY COMPONENTS

The second tactic is verifying the security of your third-party components, libraries, application dependencies, or any other code that you call as part of your application that was written by someone else (not from your dev team). Third-party components currently make up over 50% of code in all applications, and 26% of those components contain known vulnerabilities (source). When you add dependencies to your project, you are accepting the risk of every vulnerability they may include.

This issue, using components with known vulnerabilities, has remained on the OWASP Top Ten for many years. Luckily for us, MITRE created the Common Vulnerability Enumerator database (CVE), and the USA government created the National Vulnerability Database (NVD), both of which contain a list of all publicly known (i.e. publicly disclosed) vulnerabilities, which can be searched quickly and efficiently in any pipeline. There are several paid and free tools available that perform this function of varying quality and usability. If possible, two tools should be used in case there are errors or something is missed, as each tool uses different methods to verify the components and there are more than just these two databases that can be searched. No matter what type of app you write, you should verify your third-party components for vulnerabilities; this check is too fast and easy to miss. No pipeline should skip this step; it's such a huge win.

TACTIC #3: AUDIT THE STATE OF YOUR SYSTEM(S) AND SETTINGS

The third tactic is verifying the state of your server's or container's patches and config, your encryption status (key length, algorithms, expiration, health, forcing HTTPS, and other TLS settings), and security headers (browser/client-side hardening). Although your sys admins likely believe they have applied patches or various settings, this step is to audit and verify that the policy matches your application's reality. Some tools can do all three of these in one pass, and some tools specialize in only one or more of these. No application should be released onto a platform that has security misconfigurations, missing patches, or poor encryption, nor should any user be subjected to a website that doesn't use the security features available in the browser they use to access it.

TACTIC #4: ADD DYNAMIC APPLICATION SECURITY TESTING (DAST) TO YOUR PIPELINE

The fourth tactic is adding dynamic application security testing (DAST) to our pipeline by launching scripted attacks and fuzzing (finding bugs using malformed data injection in an automated fashion) against your application running while it's on a web server as a step in your pipeline. DAST, unlike the three previous suggestions, is not a quick process and thus, one or both of the following options should be employed; run only a baseline scan and/or run it in a *parallel security pipeline* that does not publish to prod, is run only after hours, and has an unlimited amount of time to finish. The baseline scan would be limited in nature, only performing passive analysis (missing headers and other obvious issues) and a small subset of dynamic tests, looking for only the worst offenders. The parallel security pipeline is one that circles back or has a dead end, never breaks the build, and is run only with the purpose of performing long and in-depth security tests, delivering the results to the AppSec team for further investigation. Developers would ignore the results of the parallel security pipeline, as many would be false positives or require further investigation. Both the baseline scan and the parallel security pipeline should be tuned by the application security team to minimize false positives.

TACTIC #5: ADD STATIC APPLICATION SECURITY TESTING (SAST) TO YOUR PIPELINE

The last tactic is adding static application security testing (SAST) of your code to your pipeline (also known as static code analysis). Generally, SAST tools are not only extremely slow (running for hours or even days) and can be quite expensive, but they also usually have a false positive rate of over 90%, which may make this suggestion a surprising one. However, if you only search for one type of vulnerability (e.g. XSS or injection) per code sprint and you carefully tune the tool, you can potentially wipe out an entire bug class from your application(s). Sharing a lesson or reference material with developers before the sprint begins would lead to faster and fewer fixes, and performing this early activity in the pipeline (or even giving access to developers before their code hits the pipeline) is another way to ensure good results. SAST could (and should) also be run to completion in the parallel security pipeline.

Conclusion

To summarize, this article suggests five tactics for adding security to your DevOps pipeline:

1. Weaponizing your unit tests
2. Verifying that your project dependencies are not known to be vulnerable
3. Verifying your server's/container's patches and config, encryption, and browser hardening settings
4. Dynamic application security testing (DAST)
5. Static application security testing (SAST)

Adding these processes to your pipeline will result in the production of infinitely more secure applications.

And in the spirit of continuous learning (the third way of DevOps), as part of the OWASP DevSlop Project, we host a weekly live-stream on Mixer and Twitch where we investigate these tactics and more, and I delve into all areas of AppSec and Cloud Native Security on my personal blog.

TANYA JANCA is a Senior Cloud Security Advocate for Microsoft, specializing in application and cloud security; evangelizing software security and advocating for developers and operations folks alike through public speaking; her open-source project OWASP DevSlop; and various forms of teaching via workshops, blogs, and community events. As an ethical hacker, OWASP Project and Chapter Leader, Cyber Ladies Ottawa founder and leader, and software developer and professional computer geek of 20+ years, she is a person who is truly fascinated by the "science" of computer science. LinkedIn - Twitter

Develop Software Customers Love Using the Applications You Want

Accelerate innovation with cloud and data

Edge and multi-cloud management

Secure user and software control

Real-time operational visibility

MESOSPHERE

Mesosphere enables your business to adopt enterprise standards, across any infrastructure, while still enabling experimentation and developer choice.

LEARN MORE

A Four Step Approach to an Enterprise-Wide DevOps Transformation

To compete in today's marketplace, business leaders are breaking down organizational silos and replicating DevOps practices across the organization to improve business workflows, overall collaboration, and the customer experience. While implementing a widespread DevOps initiative is no easy feat, here's a few ways to ensure your transformation is a success:

Secure buy-in and support

Implementing agility and collaboration enterprise-wide requires a significant change in mindset and behavior. Recruit a C-level executive sponsor to set the tone at your company, and to hold employees accountable for adopting DevOps practices.

Reduce operational friction

Delivering value to customers requires aligning business needs with the software delivery process. Remove bottlenecks, automate manual work, and increase stakeholder visibility across the enterprise to help you make informed decisions as business priorities and customer needs change.

Educate and train employees

Make it easy for employees to incorporate best practices into their everyday routine. By introducing new workflows incrementally, employees can essentially program themselves to perform actions that lead to successful outcomes.

Grow your culture

Instill a culture focused on testing, learning, and failing, while keeping the customer at the focal point. This means being less feature-centric and more solutions-centric and collecting feedback to ensure that what you're doing is something your customers want.

WRITTEN BY ALEX HISAKA
DIRECTOR OF CONTENT MARKETING, MESOSPHERE

PARTNER SPOTLIGHT

Mesosphere DC/OS

The most flexible platform for containerized, data-intensive applications

MESOSPHERE

Category Description Mesosphere DC/OS enables your business to deliver enterprise software innovation on every cloud, in the datacenter, and at the edge, while still enabling experimentation and developer choice.

Release Schedule Twice a year

Latest Release Mesosphere DC/OS 1.12

Open Source? No

Case Study Mesosphere DC/OS offers GE Transportation a more self-service way to support industrial applications in a cloud-agnostic and portable manner. Mesosphere DC/OS with Mesosphere Kubernetes Engine also enables them to deploy multiple Kubernetes clusters from a single control plane.

"We find a lot of value in having an enterprise-supported environment for our work with Kubernetes," said Wesley Mukai, GE Transportation, CTO.

GE Transportation's results driven by DC/OS:

- Developer self-service and enablement
- Centralized operations team management
- Increased service uptime and reliability
- Reduced mean time to recovery (MTR)
- Certified NIST security and privacy controls

Strengths

1. Automate operations
2. Run applications anywhere
3. Accelerate innovation
4. Maintain compliance

Notable Customers

- GE Transportation
- Royal Caribbean
- Athenahealth
- Deutche Telekom
- Auto Owners Insurance

Website
mesosphere.com

Twitter
@mesosphere

Blog
mesosphere.com/blog

DZONE.COM/GUIDES

The Rise of DevXOps:

Extending the DevOps Culture Requires Wisdom and Caution

QUICK VIEW

01. DevOps practices can sometimes leave a gap in its coverage, such as security, leading to new terms like DevSecOps.

02. DevTestOps is another such term which aims to include test professionals and test coverage more in development.

03. While these "DevXOps" practices can be useful for addressing development challenges, they run the danger of becoming new siloes, thus canceling so many of the benefits of DevOps to begin with.

BY BOB RESELMAN
SOFTWARE DEVELOPER

Over the last few years, we've seen the emergence of a new type of discipline in the DevOps landscape. I call it "DevXOps". A DevXOps discipline is one that's established to fill a gap left unaddressed by general DevOps practices.

One of the DevXOps disciplines that's getting a lot of attention these days is DevSecOps. DevSecOps has come about because security has been, for the most part, an outlier in the day-to-day software development lifecycle (SDLC) process. While it's true that security is and always has been a paramount concern for all within the enterprise, most security practices take place outside of the software development process. Many, if not most, security practices are concerned with keeping bad actors from getting into the environment and ensuring a company complies with industry standards and regulations such as those dictated by PCI-DSS and GDPR. As such, most of the attention given to security matters comes downstream in the SDLC in the forms of auditing as well as testing to determine environment security.

All of this is good; however, auditing and downstream testing leave a gap. If the practice of DevOps has taught us anything, it's that you cannot test or audit better security into a product or process. Testing and auditing will only reveal a shortcoming that should have been addressed earlier on in the process.

The DevSecOps movement came about by folks who wanted more emphasis put on securing features and processes at the onset of development, not afterward. According to cybersecurity expert and DevSecOps advocate Ben Strother, CISSP:

"The difference between DevOps and DevSecOps is that DevSecOps puts greater focus on reducing risk by adding data protection, continuous compliance monitoring, code analysis, threat modeling, vulnerability assessment, and security training into the mix."

The DevSecOps community promotes a set of best practices and publishes tools that make its philosophy apply to the real world needs of the enterprises operating at web scale. The organization is actively working to extend security into the fabric of the automation and continuous deployment practices emblematic of modern DevOps.

Another DevXOps practice that's emerging is DevTestOps. The DevTestOps movement has come about to give test practitioners and testing activity a central role in the DevOps culture. According to Lisa Crispin, Test Advocate and Author at mabl:

"DevTestOps is about building a DevOps culture in which testing activities and testers are given a central role in the software development lifecycle, from the beginning, and throughout the process."

DevTestOps fills in the gaps in testing that have been left unaddressed when implementing fully automated, continuous testing at a large scale in cloud environments. DevTestOps focuses on extending the promise of comprehensive testing and automation by better accommodating the increasing complexities of distributed applications intended to run on the internet at web scale. The Internet of Things and ephemeral, serverless computing are pushing the limits of traditional testing paradigms, both manual and automated. The new thinking is that cloud computing requires an approach to testing that is essentially cloud-based too. The operational phrase is, "Run on the cloud, test on the cloud." In addition to making testing more cloud-based, DevTestOps promotes incorporating artificial intelligence into general testing practices. In other words, applications that use AI need to be tested by automation that uses AI.

There's a good argument to be made that just about any aspect of DevOps can benefit from special attention. The rate of technological change is so fast these days that no one part of DevOps can remain static. Ongoing improvement must happen in order for DevOps to stay viable. If special communities need to emerge as a subset of DevOps in order for the necessary improvements to occur, then so be it. Yet, while the DevXOps pattern provides benefit, it can also incur risk.

In DevOps, the core operational unit is a single team that is made up of people from all walks of life in the SDLC. In DevOps, a team is completely accountable for all aspects of the product or service under its purview. If the product or service has a shortcoming, it usually means that the team has a shortcoming. Saying that the developers did not do their job or QA fell short is a "blame mentality" that should be avoided when practicing DevOps. No one person made the software. The team made the software and thus is responsible to make sure it works as expected.

If we're not careful, a DevXOps initiative can undermine team unity. As more DevXOps movements emerge from the DevOps landscape, one can imagine conflicts occurring in which a proponent of a particular DevXOps subset — for example, DevSecOps — accuses a DevOps practitioner of not being "DevSecOps enough" — thus, the need for caution.

Creating specialties can create divisions in an enterprise that run the risk of building walls that put some on the inside and others on the outside. In order for the DevXOps pattern to be an effective way to bridge gaps and make improvements in the practice of DevOps overall, we must make sure that any special community that emerges is one that is open and accepting of others. Becoming part of a DevXOps community should require nothing more than embracing the mission, following the best practices, and learning the skills necessary to make a meaningful contribution.

To my thinking, any DevXOps initiative should be short-lived. You can think of any DevXOps initiative as an operational pull request into the main branch of DevOps overall. Once its work is done, when the best practices have been established and its tools become commonplace, the DevXOps specialty should be absorbed into the parent. Otherwise, we run the risk of the branch becoming a fork. Forks, by nature, take on a life of their own and can lead to fragmentation of a community. The last thing that an enterprise embracing DevOps needs is a war of conflicting approaches that essentially solve the same problem. Remember, the fundamental feature of DevOps is the cross-functional team that works together in a unified manner. Fragmenting a team in any way has serious consequences.

The DevXOps pattern has great potential benefits provided the given instance of the pattern makes a significant improvement to the DevOps culture overall and builds bridges that bring people together. The value of a specialized DevXOps community is that it allows experts to gather together to solve complex problems that are beyond the understanding of most. Conversely, a DevXOps initiative can run the risk of fragmenting the community from which it emerged should dogma exceed its sense of mission. DevOps is about teams working together in a unified manner. Any instance of DevXOps that comes along will do well to support a team's unity, not distract from it.

BOB RESELMAN is a nationally-known software developer, system architect, and technical writer/journalist. Bob has written four books on computer programming and dozens of articles about topics related to software development technologies and techniques, as well as the culture of software development. Bob lives in Los Angeles. In addition to his work on in a variety of aspects of software development and DevOps, Bob is working on a book about the impact of automation on human employment. LinkedIn - Twitter

DZONE.COM/GUIDES

The Design of Engineering Culture

QUICK VIEW

01. An organization's culture is the collection of processes, social scripts, incentive structures, and other artifacts that describe the values and behaviors of the group.

02. By working with concrete designable surfaces that can be directly changed within a culture, that culture can be effectively designed and changed.

03. Motivation and attention are key resources that enable lasting change at both individual and group levels.

BY RYN DANIELS
INFRASTRUCTURE SOFTWARE AND OPERATIONS ENGINEER, TRAVIS CI

What is culture? The answer to this question has a deep impact on engineering effectiveness. Numerous books and articles talk about the importance of culture in DevOps, but few teams can successfully define or create it. Many organizations use the terms "culture" and "values" interchangeably. However, there's more to culture than values, and this becomes apparent when looking at how different organizations that espouse the same values in theory end up having different cultures in practice.

To illustrate this, consider two organizations that both list "a culture of learning" among their organizational values. In Organization A, post-mortems are widely attended and their write-ups are well-read. When incidents occur, the remediation items that come out of incident reviews are prioritized, and incidents rarely recur. In Organization B, individual teams sometimes hold post-mortems for incidents that affected them, but write-ups aren't read beyond the team. A recurring engineering-wide calendar invite for post-mortems goes unused more often than not, despite frequent production incidents. Remediation items are deprioritized in favor of new features and incidents recur often.

On the surface, Organizations A and B both have post-mortem meetings, write-ups, and remediation items, and both claim to value organizational learning, but the outcomes — the impact of those values on their respective cultures — are different. Culture is more than just the values an organization claims to have. For our purposes, culture is the collection of social scripts, lore, behavioral norms, biases and values, incentive structures, and processes that describe behaviors of a distinct group or social environment. DevOps as a cultural movement is a series of trends shifting instances of this collection — for example, the trend towards social scripts that encourage more collaboration between development and operations teams.

Introducing Designable Surfaces

In this example, each organization has taken the same designable surfaces and approached them in different ways. A designable surface is something concrete that can be directly changed in order to impact one or more aspects of culture. For example, the process of running a post-mortem (such as gathering an incident timeline, discussing the incident in a specific way, and creating tickets for remediation items) can be directly changed (by getting champions to model a different way of running the post-mortem meeting or creating a better template for post-incident reports) and those changes will shift that cultural process. Changing designable surfaces in an organization, monitoring the impacts of those changes, and iterating toward goals is how to build *lasting* culture change. The heart of culture design is understanding how collections of designable surfaces can be changed in concert to achieve specific outcomes.

The most successful transformations tend to be those that involve multiple designable surfaces working together. Consider the example of post-mortems and how useful people within an organization perceive them to be. Some of the relevant designable surfaces here include:

- The processes by which facilitators are trained
- The ways in which calendar invites are treated
- The format or template used for the post-mortem write-up
- The agenda for the post-mortem review meeting
- The work-tracking software used for ticketing remediation items
- The social scripts around how post-mortems are communicated

While any one of these surfaces alone could have some impact on culture, the *whole* is far greater than the sum of the parts.

Consider what steps Organization B might take if they wanted to improve the utility of their post-mortem meetings. If they hypothesize that poor facilitation was causing the meetings to be ineffective and thus people considered them a waste of time, then they might start requiring facilitators to attend a more structured training to try to address this. In addition, they would also want to do things like ensure that tooling exists for potential facilitators to sign up for training, provide a way for people wanting to run a post-mortem to find trained facilitators in a timely manner, edit or delete the recurring calendar invite that people had gotten used to ignoring, and communicate all of these changes throughout the organization. Any one of these changes occurring individually would not be anywhere near as effective as the collection of them combined.

Making Changes Stick

An important part of any culture transformation is getting changes to stick. To understand how change happens, you need to consider both motivation and attention. A common question often asked when considering a DevOps transformation is something along the lines of, "How can I get the old-school sysadmins to change when they don't seem to want to?" At a certain level, the answer is that you can't. While organizations have some ability to coerce certain behaviors out of their employees with things like threats of being fired, doing so will create an adversarial relationship that is counterproductive to creating meaningful change.

In order for changes to be both significant and long-lasting, people must be motivated to make those changes. This motivation can be intrinsic (coming from within, such as a person with a lot of curiosity and drive to learn teaching themselves new skills) or extrinsic (driven by external factors, such as only being eligible for a raise or promotion if a new skillset is acquired), but it has to be present in one way or another. Any successful change will require understanding the motivations of the people involved, and there is no one-size-fits-all answer. Different people, teams, and organizations each have unique factors contributing to their intrinsic and extrinsic motivations. As a manager or senior IC trying to enact organizational change, being able to work with people to understand their motivations and concerns is crucial.

In addition to motivation, change requires attention. A fair amount of people's attention is devoted to doing their work when everything is business as usual. Any extraordinary circumstances (such as financial stress, workplace conflicts, or outside personal issues) will deplete people's attention budgets further. Be aware of the different factors outlined in Paloma Medina's BICEPS model of the core needs and motivations of people in the workplace. If people feel that their needs aren't being met, that can take their attention and make them less amenable to change. For example, if an engineer feels like they aren't being treated equally (the "E" in BICEPS) by their manager and that is causing significant stress, it will be harder for them to focus on something that feels less important, such as remembering to use a new post-mortem template.

Company- or department-level stressors — such as restructuring, staff turnover, or organizational financial concerns — will reduce the overall capacity for change. Especially over the long term, bad habits can form, adversarial thinking can become ingrained, and people can become so burnt out that they lose most of their attention or capacity to change at an individual level. This sort of situation makes meaningful change more necessary but also more difficult to enact. In exceptionally stressful circumstances, you may need to wait until big picture issues have been resolved before trying to make meaningful changes — for example, don't try to force a new post-mortem facilitation training program on people right after (or during) a big round of layoffs.

Designing for Change

Ultimately, an engineering culture can be designed, but like any design process, it must be approached with care and rigor in order to be successful. Rather than focusing on the negative aspects of an existing culture that need to be changed, it is more useful to identify which cultural behaviors you would like to see, what values you want to embody, and take proactive steps towards those ends. It cannot be forgotten that teams and organizations are made out of people who each have their own goals, values, and motivations, and as such there is no one-size-fits-all solution that will lead to a quick and easy DevOps transformation. By keeping in mind attention and motivation at both individual and organizational levels, you can use designable surfaces in concrete ways to build and maintain the collaborative DevOps culture you want.

RYN DANIELS is an infrastructure software and operations engineer at Travis CI whose work has focused on infrastructure operability and observability, sustainable on-call practices, and the design of effective and empathetic engineering cultures. They are the co-author of O'Reilly's Effective DevOps and have spoken at numerous industry conferences on DevOps engineering and culture topics. LinkedIn - Twitter

cloudbees

POWERING THE CONTINUOUS ECONOMY

Transform your software delivery processes

Continually build, deliver and improve the software that fuels your business with the CloudBees Suite.

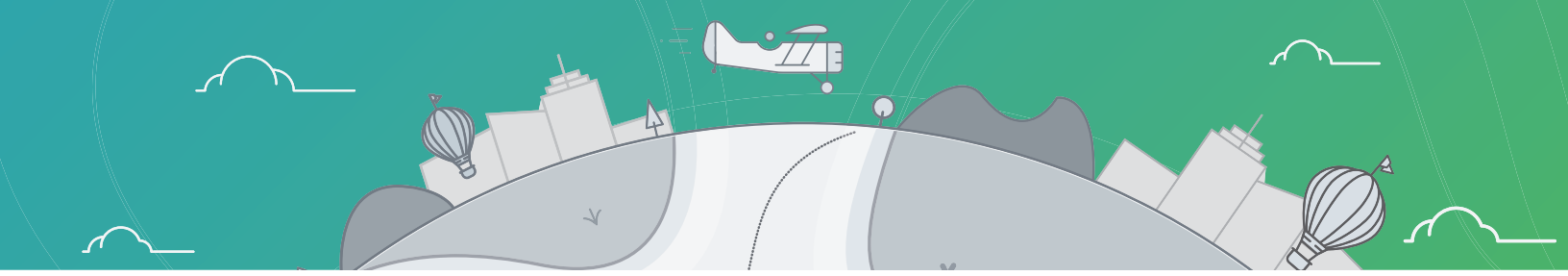
You have amazing, world-changing ideas. The only thing standing in your way is the time, energy and process it takes to turn code into finished product, to transform ideas into impact. CloudBees - the only secure, scalable and supported DevOps platform - lets you focus on the ideas you want to bring to life, not the challenge of building, testing and deploying them. You can start making an impact sooner!

CloudBees Core:
Flexible, governed continuous delivery for both on-premise and cloud native apps built for Kubernetes. Based on Jenkins, the most popular continuous delivery automation server.

CloudBees DevOptics:
Real-time visibility and actionable insights across your software delivery to measure, manage and improve DevOps performance.

CloudBees CodeShip:
A turnkey SaaS CI/CD platform that is easily configured, high-performing and flexible for complex workflows across all major cloud platforms.

Support and services:
Get commercial support and consulting for Jenkins and Jenkins X.



Try our products for free! www.cloudbees.com/get-started

Visibility Through Software Value Stream Maps

As continuous delivery scales, it becomes challenging to maintain the visibility and measurement that enables effective process automation and optimizations.

Value stream mapping is used in manufacturing to visualize how raw materials, inputs, move to finished goods, outputs. A value stream map presents a big picture of the flow, enabling identification and elimination of waste.

Software value stream mapping models inputs like user stories and source code. The processing includes software builds, testing, and deployment. The output is working software. Mapping software value streams is more complex — but valuable. So why is it complex?

- **Dynamic Raw Materials**
  In a factory, the raw materials are always the same. With software, the working issues, code, and environments are ever-evolving.
- **Scale**
  Small, agile teams are focused around one software pipeline, but are often one of many teams working on components of a bigger system. As scale increases, visibility typically decreases.
- **Geographical Dispersion**
  Complexity is increased when software is built by distributed teams. Global dependencies across time zones mean handoffs, waiting and more difficulty in gaining visibility.
- **Information Dispersion**
  Building software creates a lot of important information. Mixed toolsets mean that information is located in different systems and is often specific to developers, not CxOs, VPs, and other managers.

The complexity of software value streams only increases the benefit derived from mapping them, bringing transparency to the CD process, and aligning the organization around value-based delivery.

WRITTEN BY BEN WILLIAMS
VICE PRESIDENT OF PRODUCT MANAGEMENT, CLOUDBEES

PARTNER SPOTLIGHT

The CloudBees Suite

"We now see release frequencies daily. We're seeing lead times in hours. This is what I always believed Autodesk was capable of doing, and now we can show it." - George Swan, Director of Engineering Solutions

Category Description Continuous Integration, Continuous Delivery, Continuous Deployment, DevOps

New Releases Continuously

Open Source? No

Case Study -

SUMMARY Autodesk sparked a CI/CD transformation using CloudBees Core to establish secure, automated Jenkins pipelines.

CHALLENGE Support cloud and subscription initiatives with shorter release cycles and more frequent deployments.

SOLUTION Standardize on CloudBees Core, making secure, automated CI/CD pipelines available to a development organization of 4,000 engineers.

RESULTS

- Daily deployments achieved
- Automated, SOC 2-compliant CD established
- Positive feedback from developers and management
- Productivity increased up to 10X

Strengths

1. **Accelerates time to value**
   On-premise, cloud, or cloud-native — get continuous delivery and DevOps up and running quickly
2. **Scale CD and DevOps**
   Easily scale CD and DevOps across all of your teams and technologies
3. **Governance and developer freedom**
   Confidently manage security and compliance while empowering teams
4. **Expert support and services**
   CD, DevOps and Jenkins expertise available 24/7

Notable Customers

- Adobe
- Allianz
- Bosch
- Capital One
- Carfax
- Cummins
- HSBC
- Hyatt
- Thomson Reuters

Website
cloudbees.com/products

Twitter
@cloudbees

Blog
cloudbees.com/blog

DevOps Adoption Practices

QUICK VIEW

01. The core engine for a successful DevOps transformation is the plan-do-check-adjust cycle.

02. A technology tree describes certain technologies or practices (e.g. DevOps) and the prerequisites required to enable it.

03. There is no one correct way to adopt DevOps, but there are several common approaches that have proven to be successful.

BY MIRCO HERING
MANAGING DIRECTOR, ACCENTURE

In the *2018 DZone Guide to DevOps*, I provided a view on how to approach your DevOps roadmap, and I think it is time to provide a second chapter. Last time, I focused on which applications to choose for the roadmap; this time, I will talk about the practices. While there are several ways of approaching this, we see certain patterns. I will start with the most common one, which follows five steps. At the end, I will provide two alternative views so that you can pick and choose what works best for your organization.

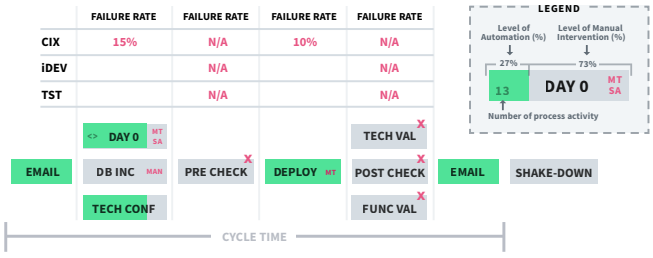
Before I describe the three different perspectives of the roadmap, I want to highlight the core engine for change: the plan-do-check-adjust cycle. Each significant change you make should be driven by this improvement process, which means that before you implement a change, you need to define how you will determine the success of that change. Clear success criteria and/or metrics that show you whether the change has made the difference you were hoping for are important and need to be defined before you implement any modifications. If you follow this process and align roughly with one of the following patterns, I am sure you will see significant progress. After all, your context is unique, and hence, you need to have a roadmap that supports the ability to adapt as the roadmap hits your context.

First Step: Reduce Variables

Many organizations start with an environment that is full of variables: different processes, different environments, different tools, and several permutations of configurations and data. All this makes automation hard and reduces your ability to learn as each variable could be the cause of the problem. The first step is to look at all those variables and see what you can remove. Can you align the patch levels across environments? Can you deploy the same version of the application across environments? Some variables can only be removed later on, but understanding what all the variable pieces are and doing a clean-up first will make later efforts easier.

Second Step: Document the End-to-End

Someone once told me: "You cannot automate what you cannot document." After all, automation is a form of documentation of a process. What is even more important is that automating a bad process just creates more problems. I also think that writing down a solution forces you to think it through in a way that verbal communication or just starting to write code does not. Don't write huge documents but write enough so that someone else can look at it and review your new process for automation. That way, you get the chance for a peer review. I also know from experience that it is very difficult to get to 100 percent automation immediately. Having process documentation is important for visualizing how much of the process is automated. Below is an example deployment process diagram.



Third Step: Start With SCM, Build, Deploy, and Environments

Following the same line of thinking, the first capabilities that make sense to automate are the ones that reduce variability in the solution: SCM, build, deploy, environment configuration, or provisioning. Having gaps in any of these will mean that your environments are potentially not aligned and any problems you find might have more to do with the environment than with the application itself — the typical "it worked on my machine" problem. I have seen integration test environments where less than 30 percent of defects were functional and all others related to these core DevOps practices. On a positive note, these are the easier ones, and in my experience, you can almost always solve these with high levels of automation; even not 100 percent automation adds value here as you reduce variability.

Fourth Step: Test Automation

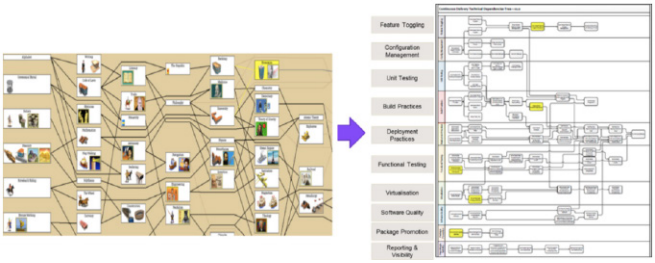
The next step is to invest in test automation. Test automation is by far the DevOps/Agile practice that I see fail the most. The problem is that in test automation, you are testing the whole system and hence, you require a framework that can orchestrate across the system and manage the data flows. But one thing is certain: if you don't use test automation on a stable and predictable base of environments and applications, then all you do is very quickly and automatically validate that you have a problem. That is not the point of test automation; rather, it is to validate that your application works correctly.

Fifth Step: Self-Service Capabilities

Once you have a decent level of automation, you can start to enable them as self-service capabilities managed by a platform team. This will help you to establish autonomous teams and reduce the silos. Your Agile teams can consume these services as required and can provide feedback back to the platform team for further enhancements. This is logically similar to using a provider like AWS or Azure, except that you provide a solution that is contextually meaningful for your organization.

Alternative A: Use a Technology Tree

The above five steps have proven to be very useful for me, especially in the context of having a central platform team that enables Agile teams. If you prefer a more federated approach, then I can give you an alternative solution. Inspired by the technology tree from the computer game Civilization, my team and I created a technology tree for continuous delivery. A technology tree describes certain technologies or practices and the prerequisites required to enable it (e.g. you need wheels, engines, and combustion to build a modern car). In our technology tree, we had a description for each DevOps practice supported by outcome and progress measures that teams can use to see how they progress and what impact it has. We handed this technology tree out to all our teams and they could fill it in as they progressed. We did highlight a few central capabilities like our asset repository and our test automation framework, which teams had to integrate with, but otherwise, they could create their own journey.



Alternative B: Look at the Roadmap From the Puppet State of DevOps 2018

Last year, Puppet came out with a new State of DevOps that looked into the common adoption roadmaps, and their research led to a very similar roadmap to the one I described above. For full details, you can read the report here. In summary, this roadmap aims to normalize the technology stack and reduce variability first, before implementing core DevOps practices and automating infrastructure delivery. As a last step, the enablement of self-service capabilities aligns with the last step of my roadmap described above.

As you can see, there is no one correct way to adopt DevOps, but there are common approaches that have proven to be successful. There is no need to make your own painful experiences of failed automation projects or DevOps transformations. Take one of the proven roadmaps and adapt it along the way with the PDCA cycle and you will be on the right track. Just make sure you are not biting off too much too quickly. Take small, incremental, and iterative steps in your transformation.

MIRCO HERING is a Managing Director at Accenture and leads the Agile and DevOps practice in APAC. For over a dozen years, he has worked on accelerating software delivery through innovative approaches and ten years ago started experimenting with Agile. He supports major public and private sector companies in their search for efficient IT delivery. He shares his experiences on his blog and at international conferences. He is the author of *DevOps for the Modern Enterprise*, a book helping companies adopt Agile and DevOps and push through transformation barriers. He has been finalist in three categories of the DevOpsDozen awards 2018 by DevOps.com. LinkedIn - Twitter

Automating Open Source Security In Your DevOps Pipeline

You're in a race against the clock. Time is your most valuable resource as you try to keep up with the rapid pace of development. It's all about innovation, and meeting release schedules. You can't afford to fall behind. You're about to make it to the finish line when you see a security flaw lurking right there. You're brought to an unnecessary halt.

According to WhiteSource's 2018 Annual Report, nearly all developers rely on open source components on a daily basis to keep up with tight deadlines without sacrificing on quality. The problem? Many organizations still treat open source security as an afterthought, despite the significant rise in open source security vulnerabilities over the past few years. This mistake could end up hurting your agility, flexibility, and — ultimately — your organization's products and customers.

"Security slows the process down" you may be thinking right now. But contrary to belief, this doesn't need to be the case. Open source vulnerabilities can be revealed from the get-go, by choosing the right automated tools and integrating automated open source security checkpoints in different phases of the DevOps pipeline. Developers can remediate security flaws in open source components early in the pipeline, saving time and most importantly, boosting agility.

Bottom line? Don't treat security as an afterthought. By continuously automating open source security measures throughout the development lifecycle, development and DevOps teams will be able to power through to meet tight release schedules, allowing all teams to make it to the finish line effortlessly.

WRITTEN BY RAMI SASS
CEO & CO-FOUNDER, WHITESOURCE

PARTNER SPOTLIGHT

WhiteSource

Automates and secures open source components used in your code by fully integrating into your software development lifecycle.

Category Application Security | Software Composition Analysis | DevOps | Secure Coding

Release Schedule Continuous

Open Source? Yes

Case Study Due to the sensitive regulatory environment Siemens Healthineers works in, Siemens H. needed assurance that they are remaining compliant and secure in their use of open source components.

"With open source software, usually the source code is available for all to see, including hackers," explained Code Clinic Lead, Neil Langmead. In order to avoid costly mistakes that can result from vulnerable or risky open source components being added to their products, Siemens Healthineers turned to WhiteSource.

"We chose WhiteSource because of its ease of use, its excellent data, and for the in-depth security vulnerability information that comes with the reporting engine." With WhiteSource, Siemens was offered the widest coverage of plugins and languages that they needed, as well as the continuous monitoring and policy enforcement safeguards they required in order to allow their team to code with confidence. Read more here

Strengths

1. Largest Vulnerabilities Database: Continuously aggregates information from multiple sources
2. Comprehensive Coverage: Supports 200+ programming languages and 20+ environments (incl. containers)
3. Pinpoint Accuracy: Proprietary algorithms guarantee no false positives
4. Automated Workflow: Enforce policies automatically at all stages of the SDLC
5. Easy Remediation: Pinpoints vulnerable methods affecting your products

Notable Customers

- Microsoft
- Comcast
- EllieMae
- IGT
- Spotify

Website
whitesourcesoftware.com

Twitter
@WhiteSourceSoft

Blog
resources.whitesourcesoftware.com

DZONE.COM/GUIDES

2019 Executive Insights on DevOps

QUICK VIEW

01. The tenets of DevOps are still – better, faster, sooner. Move faster, empower developers, achieve better code quality, have a more resilient infrastructure, and improve security.

02. Automated DevSecOps is how most respondents are integrating security into their DevOps pipeline along with static code analysis and dynamic application security testing.

03. Successful DevOps implementation are a function of a culture of collaboration and real-time feedback on mutually agreed upon metrics for automated processes.

BY TOM SMITH
RESEARCH ANALYST, DEVADA

To understand the current and future state of DevOps, we spoke to 40 IT executives from 37 organizations. Here's who we spoke to:

- Tim Curless, Senior Technical Architect, AHEAD
- Will Hurley, Vice President of Software Lifecycle Services, Astadia, Inc
- Lei Zhang, Head of Developer Experience (DevX), Bloomberg
- Ashok Reddy, Group General Manager, CA Technology
- Sacha Labourey, CEO, CloudBees
- Logan Daigle, Director DevOps Strategy and Delivery, CollabNet
- Sanjay Challa, Senior Product Marketing Manager, Datical
- Colin Britton, CSO, Devo
- OJ Ngo, CTO, DH2i
- Andi Grabner, DevOps Activist, Dynatrace
- Anders Wallgren, CTO, Electric Cloud
- Armon Dadgar, founder and Co-CTO, HashiCorp
- Tamar Eilam, IBM Fellow, Next Generation Cloud and DevOps, IBM Research
- Mathivanan Venkatachalam, Vice President, ManageEngine
- Jim Scott, Vice President of Enterprise Architecture, MapR
- Mark Levy, Director of Strategy, Micro Focus
- Glenn Grant, President - U.S. East, Mission
- Jonathan Lewis, Vice President of Product Marketing, NS1
- Zeev Avidan, Chief Product Officer, OpenLegacy
- Tyler Duzan, Product Manager, Percona
- Bradbury Hart, Vice President and Chief Evangelist at Perfecto
- Damien Tournoud, Founder and CTO, Platform.sh
- Bob Davis, Chief Marketing Officer and Jeff Keyes, Director of Product Marketing, Plutora
- Brad Micklea, Senior Director and Lead, Developer Business Unit, and Burr Sutter, Director, Developer Experience, Red Hat
- Dave Nielsen, Head of Ecosystem Programs, Redis Labs
- Brad Adelberg, Vice President of Engineering, Sauce Labs
- Adam Casella, Co-founder and Glenn Sullivan, Co-Founder, SnapRoute

- Dave Blakey, CEO, Snapt
- Keith Kuchler, Vice President of Engineering, SolarWinds
- Justin Rodenbostel, Vice President of Open-Source Applications at SPR
- Jennifer Kotzen, Senior Product Marketing Manager, SUSE
- Oded Moshe, VP of Products, SysAid
- Loris Degioanni, CTO and Founder, Sysdig
- Jeffrey Froman, Director of DevOps and Aaron Jennings, Engineer, Temboo
- Pan Chhum, Infrastructure Engineer, Threat Stack
- John Morello, CTO, Twistlock
- Madhup Mishra, Vice President of Product Marketing, VoltDB
- Joseph Feiman, Chief Strategy Officer, WhiteHat Security
- Andreas Prins, Vice President of Product Development, XebiaLabs

Key Findings

1. The most important elements of a successful DevOps implementation are a culture of collaboration and real-time feedback on mutually agreed upon metrics for automated processes. It's a philosophy or movement rather than a methodology. Culture is the cornerstone of success, and drives collaboration and sustainable practices. Start with culture and envision how the team will interact to achieve well-defined and articulated goals. Leadership needs to adopt and embrace DevOps. Create a culture of discovery and fail fast. Focus on learning, experiments, and continuous improvement. You cannot just do DevOps in a silo without product, marketing, engineering, finance, sales, and executive management being affected by it.

Automate everything — builds, deployments, testing, reporting, whatever you can. Institute repeatable processes, track metrics, measure, and ensure you are meeting and exceeding KPIs. Every test needs to be repeatable and scalable. Continuous improvement needs to be measured and valued. Align incentives between the operations, development, and security teams.

2. Automated DevSecOps is the way most respondents are integrating security into their DevOps pipeline in addition to static code analysis and dynamic application security testing. It's more critical than ever that DevOps embraces security from the outset and that DevSecOps is baked in from the beginning. Security is a first-class citizen given the number of breaches that are taking place. Without security, an organization cannot survive. By integrating security into DevOps, developers can easily and routinely product software that is free of flaws and vulnerabilities.

There are multiple technologies to consider: 1) SAST (static analysis security testing), analyzing application code like a compiler and building logic trees and the data flow of an application; 2) DAST (dynamic application security testing), an automated hacker that you can use to analyze responses to determine if it was able to break into or trick the application; 3) SCA (software composition analysis) to analyze application code, detect open-source code, and determine if it's secure/legal, and how far it is behind the most recent release. Ensure that engineers understand secure coding and employ peer reviews. We have a segregation of duties in place, and we track how everything is produced in the pipeline.

3. The biggest changes to DevOps have been the level of acceptance, the predominance of the public cloud and the tools offered, and containers and Kubernetes (K8s). DevOps used to be only for Silicon Valley companies. Now it's on everyone's mind. There is greater acceptance as people are aware of the benefits and the availability of tooling and knowledge. Culture adoption was the first wave and is now a constant. Now we're rolling into more tools with software and virtual hardware with infrastructure as code automation.

The emergence of cloud-native is another major change. Everything is done in the cloud, and there has been an explosion of tools and infrastructure options. Tools are becoming more sophisticated with the cloud environment.

There is also a change in attitude with the rise of K8s in the public cloud. When the cloud begins to run K8s for you, that's a game changer. It's much easier to virtualize code and automate everything. Containers make it easier for developers to do work in an environment that looks like production.

4. Speed to market is the overarching value provided by DevOps with several aspects of speed mentioned. The tenets of DevOps are the same: moving faster, empowering developers, achieving better code quality, having a more resilient infrastructure, and improving the security posture.

DZONE.COM/GUIDES

Minimize the time it takes to deliver value to customers. The cycle time from developer completion of a story/defect/task to production is dramatically reduced via DevOps and continuous delivery, allowing for value to be realized as quickly as possible. Deployment and development to production cycles can go from months to hours, without additional headcount and decreasing risk through automation.

Another benefit is the ability to decrease time to market, increase customer satisfaction, and gain a sustainable competitive advantage. Reducing cycle times can also speed innovation since new ideas can be brought to market more quickly. Problems are able to be resolved more quickly with shorter feedback loops.

5. While use cases were provided in 14 different industries, financial services and insurance examples were provided far more than any other. The financial market is very dynamic with constant changes being driven by different perspectives around the world, such as new financial regulations and reporting requirements. A large bank created several dozen new APIs and digital services in less than five months. Previously, it would have taken them at least 18 months. Key Bank has gone from a three-month deployment interval to a one-week deployment interval. A major global insurer reduced annual costs from $3 million to $120,000 per test environment while reducing the environment provisioning time from six weeks to two hours.

6. The most common reason for DevOps failures are **unwillingness to adopt a "fail fast and iterate approach;" executive level support for the initiative; and alignment between all of the people responsible for implementing DevOps**. If a customer starts with "fail fast and iterate," it's pretty easy. You have an honest and healthy retrospective in contrast to coming from a blame model. You need a blame-free environment to successfully implement DevOps. Learn from your mistakes. If it hurts, do it more often in smaller batches to learn and remove the pain. Find people with the courage to tell you what didn't work and why. Embrace failure to learn.

Get executive management on board early. Only the management team can align resources within the organization to drive the level of transformation required. Management also has an end-to-end view of the value chain and is best positioned to bring all of the resources to bear on a common path. It's critical that an organization's leaders shield new teams from traditional organizational pressures to deliver. There needs to be organizational support for long-term adoption.

Processes must be aligned across all of the teams. Every representative of the SDLC needs to be at the table. If you leave someone out, they'll be offended and serve as an impediment to your DevOps roadmap. If responsibilities are not well-defined, culture change doesn't take affect and people don't take ownership.

7. The future of DevOps is more focused on security and the adoption of DevSecOps, improving processes by utilizing AI/ML, and more automation. More emphasis is needed on security and integrating security into DevOps upfront for DevSecOps.

AI and ML will enable organizations to make better decisions faster by providing unique analysis and correlating code, test results, user behavior, and production quality and performance. AI/ML data-driven understanding of patterns, finding the patterns, providing remediation, resulting in self-driving and self-healing applications. Automation provides the opportunity to integrate changes in process with changes in culture to maximize flow, feedback, and continuous improvement.

8. Three things developers need to keep in mind with regard to DevOps/CI/CD are automation, alignment, and security. Automate from your first deployment; never deploy manually. Think about automating everything you do. Shorten delivery cycles by embracing automation across software development, integration, and delivery pipelines. Have an agile mindset.

CI/CD is entirely dependent on automated tests. Without an extensive test suite for all the different parts of the code, developers will not have the confidence to have their applications continuously deployed to production, thus introducing friction into the development process.

It's all about alignment. Everyone needs to be on the same page with the same process. We fail when we have divisions. Be a developer but understand security, operations, and deployment challenges. Be able to empathize with security and operations to deliver applications in a more efficient way to reduce organizational friction. Knowledge is power.

TOM SMITH is a Research Analyst at Devada who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.. LinkedIn - Twitter

DevOps Solutions Directory

This directory contains software configuration, build, container, repository, monitoring, and application performance management tools, as well as many other tools to help you on your journey toward Continuous Delivery. It provides the company name, product information, open source data, and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

Company	Product	Product Type	Free Trial	Website
Amazon	Amazon ECS	Container management	N/A	aws.amazon.com/ecs
Apache Software Foundation	Ant	Build management	Open source	ant.apache.org
Apache Software Foundation	Archiva	Repository management	Open source	archiva.apache.org
Apache Software Foundation	Maven	Build management	Open source	maven.apache.org
Apache Software Foundation	Subversion	Build management	Open source	subversion.apache.org
Apache Software Foundation	JMeter	Web & Java testing	Open source	jmeter.apache.org
AppDynamics	AppDynamics	Application performance management	15 days	appdynamics.com
Appium	Appium	Automated web & mobile testing	Open source	appium.io
Atlassian	Bamboo	Continuous integration & application release automation	30 days	atlassian.com/software/bamboo
Attunity	RepliWeb for ARA	Application release automation	Available by request	attunity.com/products/repliwebx
BigPanda	Autonomous Digital Operations	Monitoring alert software	Demo available by request	bigpanda.io

DZONE.COM/GUIDES

BE THE HERO

When the boss discovers DevOps has sped application delivery by 30%, make sure you're the one wearing the cape.

Start your crusade with a Jobs-as-Code approach

jobsascode.io

Jobs-as-Code: The Power of Shifting Left

What's the number one goal for every developer? It's simple: to **build better business applications faster**. But, most would agree that it's easier said than done. Why is it challenging? Here are a few stats from Forrester[1] that shed some light on the issue:

- ~ 33% of IT pros said **silos is their top challenge** to delivery velocity
- Almost 50% of dev teams have **quality/performance issues in production**

It's clear that the traditional IT dev and ops siloed structure is broken. So, what's the solution? DevOps and Jobs-as-Code!

By building jobs early in the Software Delivery Lifecycle (SDLC), using an "as-code" approach, all the creative and manual work is done together. The delivery pipeline can then run fully automatically as all the application artifacts are stored in a source code repository. This approach ensures all application components are built and fully tested together before promotion to the production environment.

Jobs-as-Code is an approach that treats application automation management in the SDLC just like any other code components of an application. You can learn more at jobsascode.io

By modernizing current job and workload orchestration as code and integrating within existing continuous integration/continuous delivery pipeline (CI/CD) tool chains, companies can overcome silos and **save time, money, and headaches in projects ranging from big data and dynamic infrastructure to multi-cloud**.

References: (1) Forrester, Making Business Workflows First-Class Citizens in the Modern Software Delivery Life Cycle, October 2017

WRITTEN BY JOE GOLDBERG
INNOVATION EVANGELIST

PARTNER SPOTLIGHT

Control-M

"Control-M is a huge part of our business… From a development point of view you want to treat jobs like all other software, to build, run, test, then have a continuous integration and continuous delivery of your jobs."
- Alice Albano, Software Engineer, Amadeus IT Group SA

Category Workload Orchestration/Jobs-as-Code

Release Schedule Continuous

Open Source? No

Case Study Amadeus, a global leader in the travel industry, provides IT services to that help customers handle bookings, flight management, data distribution, billing, and more. They recently moved to a private cloud infrastructure, transitioning more than 180 applications — which rely on a complex set of batch workloads and file transfers — to this new technology stack without affecting stringent service level agreements (SLAs). Amadeus leverages Control-M to continuously increase its level of automation to accommodate its ever-increasing workloads. With Control-M, they manage **5,500 IT changes and 540 applications per month**, and have **achieved a 20% reduction** in time to deploy applications, while maintaining **99.99% availability**.

Strengths

1. **Supporting a Jobs-as-Code approach** helps organizations operationalize applications faster by embedding application workflow orchestration into the current development and release process.
2. **Accelerate application change and deployment cycle times** between test and production with automated application workflow creation, testing, and promotion.
3. **Quickly and reliably deliver business services** by decreasing downstream complexity and minimizing errors when rolling into production.
4. **Increase team member productivity** by reducing the time spent learning a new orchestration tool for every application or manually scripting and re-scripting jobs.

Notable Customers

- Amadeus
- Navistar
- CARFAX
- Malwarebytes
- MSA
- Raymond James Financial

Website
bmc.com

Twitter
@ControlM_BMC

Blog
bmc.com/blog

Company	Product	Product Type	Free Trial	Website
BMC	Control-M Automation API	Job scheduling and application deployment automation	Available by request	bmc.com/it-solutions/control-m.html
Broadcom	CA Continuous Delivery Automation	CD automation across apps	30 days	ca.com/us/products/ca-continuous-delivery-automation.html
Buildbot	Buildbot	Continuous integration	Open source	buildbot.net
CenturyLink	Cloud Application Manager	Infrastructure, application, & service orchestration	N/A	ctl.io/cloud-application-manager
Chef	Chef Automate	Continuous deployment platform	Open source	chef.io/automate
CircleCI	CircleCI	Continuous integration	Free tier available	circleci.com
CloudBees	CloudBees Core	Governed & flexible CD	Available by request	cloudbees.com/products/cloudbees-core
Codeship	Parallel Test Pipelines	Continuous integration	14 days	codeship.com/features/basic/parallel-test-pipelines
Cucumber	Cucumber	Automated Rails testing	Open source version available	cucumber.io
Datadog	Datadog	IT stack performance management	14 days	datadoghq.com
Docker	Docker Swarm	Container management	Open source	github.com/docker/swarm
Eclipse	Hudson	Continuous integration	Open source	eclipse.org/hudson
Electric Cloud	ElectricFlow	Application release automation	Free tier available	electric-cloud.com/products/electricflow
FitNesse	FitNesse	Acceptance testing framework	Open source	fitnesse.org
Free Software Foundation	Concurrent Versions Systems (CVS)	Software configuration management	Open source	savannah.nongnu.org/projects/cvs
Git	Git	Software configuration management	Open source	git-scm.com

Company	Product	Product Type	Free Trial	Website
Gitlab	Gitlab	Code review, CI, CD	Open source	about.gitlab.com
Gradle	Gradle	Build automation	Open source	gradle.org
Grid Dynamics	Agile Software Factory	Application release automation	N/A	griddynamics.com/blueprints
Gridlastic	Gridlastic	Automated web testing	Demo available by request	gridlastic.com
Hashicorp	Vagrant	Configuration management	Open source	vagrantup.com
IBM	Rational	Software configuration management	N/A	ibm.com/software/rational/strategy
IBM	Urbancode Build	Continuous integration & build management	Available by request	developer.ibm.com/urbancode/products/urbancode-build
IBM	Urbancode Deploy	Application release automation	Demo available by request	developer.ibm.com/urbancode/products/urbancode-deploy
Inedo	Buildmaster	Application release automation	Open source version available	inedo.com/buildmaster
Inflectra	Rapise	Automated web testing	Available by request	inflectra.com/Rapise
InfluxData	InfluxEnterprise	Stream processing & analytics	Open Source	influxdata.com/products
Jenkins	Jenkins	Continuous integration	Open source	jenkins.io
JetBrains	TeamCity	Continuous integration & application release automation	Free tier available	jetbrains.com/teamcity
jFrog	Artifactory	Repository management	14 days (cloud) or 30 days (on-prem)	jfrog.com/artifactory
Junit	JUnit	Unit testing framework	Open source	junit.org
Librato	Librato	Monitoring alert software	14 days	librato.com

DZONE.COM/GUIDES

Company	Product	Product Type	Free Trial	Website
Linux Foundation	Kubernetes	Container management	Open source	kubernetes.io
Mercuiral	Mercurial	Software configuration management	Open source	mercurial-scm.org
Mesosphere	Marathon	Container orchestration	Open source version available	mesosphere.github.io/marathon
Micro Focus	ALM Octane	Application lifecycle management	Available by request	saas.hpe.com/en-us/software/alm-octane
Micro Focus	AccuRev	Software configuration management	30 days	microfocus.com/products/change-management/accurev
Micro Focus	Deployment Automation	Automated deployments for CD	N/A	microfocus.com/products/deployment-automation
Microsoft	Team Foundation Server	Software configuration management	N/A	docs.microsoft.com/en-us/visualstudio/releasenotes/tfs2018-update3
MidVision	RapidDeploy	Application release automation	Demo available by request	midvision.com/product/rapiddeploy
Nagios	Nagios Core	Infrastructure monitoring	Open source	nagios.org/projects/nagios-core
New Relic	New Relic	Application performance management	Demo available by request	newrelic.com
NowSecure	NowSecure	Mobile app security	Demo available by request	nowsecure.com
NS1	NS1	Managed & private DNS	Demo available by request	ns1.com
NuGet	NuGet	Repository management	Open source	nuget.org
NUnit	NUnit	Unit testing framework	Open source	nunit.org
OpsGenie	OpsGenie	Monitoring alert software	Available by request	opsgenie.com
OutSystems	OutSystems	Low-code development platform	Available by request	outsystems.com

Company	Product	Product Type	Free Trial	Website
PagerDuty	PagerDuty	Monitoring alert software	14 days	pagerduty.com
Parasoft	Parasoft	Automated web & API testing	Available by request	parasoft.com
Perforce	HelixCore	Software configuration management	Free tier available	perforce.com/products/helix-core
Plutora	Plutora	Application release automation	Demo available by request	plutora.com
Puppet	Puppet	Configuration management	Free tier available	puppetlabs.com
Rake	Rake	Build automation	Open source	github.com/ruby/rake
Ranorex	Ranorex	Automated web & desktop testing	30 days	ranorex.com
Red Gate Software	SQL Change Automation	CI & automated deployments for SQL Server DBs	14 days	red-gate.com/products/sql-development/sql-change-automation
Red Hat	Ansible	Configuration management & application release automation	Open source	ansible.com
Rocket Software	ALM and DevOps	Application release automation	Demo available by request	rocketsoftware.com/product-categories/application-lifecycle-management-and-devops
Rogue Wave Software	Zend Server	Application release automation for PHP	Available by request	zend.com/en/products/zend_server
Sahi Pro	Sahi Pro	Automated web testing	Free tier available	sahipro.com
SaltStack	Salt	Configuration management	Open source version available	saltstack.com
Sauce Labs	Sauce Labs	Automated web & mobile testing	14 days	saucelabs.com
Scalyr	Scalyr	Log monitoring & analysis	30 days	scalyr.com
Selenium	Selenium WebDriver	Automated web testing	Open source	seleniumhq.org

DZONE.COM/GUIDES

Company	Product	Product Type	Free Trial	Website
SmartBear Software	SoapUI	Automated web & API testing	Open source version available	soapui.org
Solano Labs	Solano Labs	Continuous integration & deployment	14 days	solanolabs.com
Sonatype	Nexus	Repository management	Open source	my.sonatype.com
Splunk	VictorOps	Monitoring alert software	Available by request	victorops.com
TestingBot	TestingBot	Automated web testing	14 days	testingbot.com
TestNG	TestNG	Unit testing framework	Open source	testng.org/doc/index.html
Thoughtworks	GoCD	Application release automation	Open source	gocd.io
TravisCI	TravisCI	Continuous integration	Open source	travis-ci.org
Venafi	Venafi Platform	Machine Identity Protection	Free tier available	venafi.com
Watir	Watir	Automated web testing	Open source	watir.com
WeaveWorks	Continuous Delivery	Continuous deployment platform	14 days	weave.works/features/continuous-delivery
Windmill	Windmill	Automated web testing	Open source	getwindmill.com
White Source	White Source	Application Security | Software Composition Analysis | DevOps | Secure Coding	Open Source	whitesourcesoftware.com
Xebia Labs	XL Deploy	Application release automation	30 days	xebialabs.com/products/xl-deploy
xMatters	xMatters	CD & IT event management	14 days	xmatters.com
xUnit.net	xUnit.net	Unit testing framework for .NET	Open source	xunit.github.io

Diving Deeper Into DevOps

Twitter

 @RealGeneKim

 @sKriemhild

 @bridgetkromhout

 @jezhumble

 @DonovanBrown

 @lisacrispin

 @editingemily

 @testobsessed

 @proudboffin

 @HelenRanger4

Podcasts

Arrested DevOps

Get the knowledge to help you achieve understanding, develop good practices, and operate your team and organization for maximum DevOps awesomeness.

The New Stack Makers

Through featured speakers and interviews, learn about new software stacks that are changing the way we develop and deploy software.

DevOps Radio

Learn about effectively achieving software delivery through DevOps.

Zones

DevOps dzone.com/devops

DevOps is a cultural movement supported by exciting new tools that is aimed at encouraging close cooperation within cross-disciplinary teams of developers, and IT operations, and system admins. The DevOps Zone is your hot spot for news and resources about continuous delivery, Puppet, Chef, Jenkins, and more.

Agile dzone.com/agile

In the software development world, Agile methodology has overthrown older styles of workflow in almost every sector. Although there are a wide variety of interpretations and techniques, the core principles of the Agile Manifesto can help any organization in any industry improve their productivity and success.

Open Source dzone.com/opensource

The Open Source Zone offers practical advice regarding transitioning from a closed to an open project, creating an enforceable code of conduct, and making your first OSS contributions. This Zone encourages you to adopt an open-source mentality and shape the way open collaboration works.

Refcardz

Continuous Delivery

Minimize the time it takes to go from idea to usable software. Learn to use agile techniques and automate the entire software delivery system: build, deploy, test, release.

Compliant DevOps

With new data protection laws coming into play, and consumers more aware than ever before of how their privacy is being compromised, there is now a requirement for companies to adopt a compliant DevOps approach.

Shifting Left With Continuous Delivery

Learn how shifting left with CD makes software development faster and more reliable, why whole teams need to cooperate in effective left shifts, and how containers are important in shifting left with continuous delivery.

Courses

Master Jenkins CI for DevOps and Developers

Take your DevOps skills to the next level by learning how to build automated CI pipelines with Jenkins.

Learn DevOps: The Complete Kubernetes Course

Learn how to build, deploy, use, and maintain Kubernetes to run and manage your containerized apps.

DevOps for Developers: How to Get Started

Get a comprehensive definition of DevOps, an understanding of why you should do DevOps, and insight into how to get started with DevOps.

INTRODUCING THE

Open Source Zone

Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

Visit the Zone

BROUGHT TO YOU IN PARTNERSHIP WITH Flexera