THENEWSTACK

GUIDE TO CLOUD NATIVE DEVOPS

The New Stack

Guide to Cloud Native DevOps

Alex Williams, Founder & Editor-in-Chief

Core Team:

Bailey Math, AV Engineer

Benjamin Ball, Marketing Director

Chris Dawson, Technical Editor

Gabriel H. Dinh, Executive Producer

Joab Jackson, Managing Editor

Judy Williams, Copy Editor

Kiran Oliver, Podcast Producer

Lawrence Hecht, Research Director

Libby Clark, Ebook Editor, Editorial Director

Michelle Maher, Editorial Assistant

Table of Contents

Introduction

Writing this DevOps ebook challenged us. Why write an ebook at all about DevOps? The story has been told quite thoroughly by leagues of experts. For The New Stack, it's a bit different. We look at the issue in terms of scale. As we explain and analyze what scale means, DevOps practices surface again and again in the research, interviews and data. In developing our guides to cloud native microservices and serverless technologies one major theme emerged: cloud native implementations cannot succeed without mature DevOps practices.

With scale in mind, it just made sense to focus on the cloud native DevOps practices and workflows that practitioners are developing for the outer dimensions of at-scale application architectures. But what exactly those "cloud native DevOps" practices are, wasn't clear. In this third and final book in our cloud native technologies series, we examine in detail what it means to build, deploy and manage applications with a cloud native approach.

Teams that are addressing at-scale development and deployment in application-oriented architectures are becoming increasingly familiar with the practice of accelerating software development. They must continually look for value, efficiencies and correlations that make the experience better, make the workflows more optimized.

It's a continual adaptation that DevOps practices help manage. Such practices have been built upon and refined for over a decade in order to meet the deeply complex challenge of managing applications at scale. And DevOps is now undergoing another transformation, buoyed by the increasing automation and transparency allowed through the rise of declarative infrastructure, microservices and serverless architectures. This is cloud native DevOps. Not a tool or a new methodology, but an evolution of the longstanding practices that further align developers and operations teams.

New Roles for Devs and Ops

Roles and responsibilities are changing as infrastructure is abstracted into the cloud and becomes more programmable. A reimagining of the interactions between developers and operations teams is well underway. And as a result, the definition of DevOps changes as well.

Practices have evolved quickly. There is a “new guard” of full stack developers — in the words of our friends at LaunchDarkly — who all have some part in developing and managing services. Developers now count on approaches that treat the architecture as invisible, allowing them to program the resources according to the workloads their team is managing. Similarly, the operations story is quickly changing as the role of site reliability engineer (SRE) grows and becomes more associated with overall services management.

Services are now at the core of how modern businesses work. All the auto-navigating that a phone manages, the immediate payment from an application, the secure connection back to the bank — at their technical depths all are the result of automated and declarative technologies developed on distributed architectures by teams of developers and software engineers.

Shortening the feedback cycle between developer and end user experience speeds application development and provides critical, actionable business information in a timely way. The operations role is also growing, as a result. SREs complement those on operations teams, who together develop infrastructure software, technologies and services. The service is the product in today’s world, making their roles aligned as they both seek better efficiencies and observations in the feedback cycle to improve the experience for the services they provide.

The Path Forward

The evolution of service managers exemplifies how infrastructure software is developed to just make the experience a bit better all the time. The path forward appears in the advancement of declarative infrastructure, automation and new fields such as artificial intelligence-meets-operations, or AIOps. Watch the marketing hype for this branding approach — the real value comes in meeting the needs of organizations grappling with issues of scale on distributed infrastructure.

The workflows that modern, cloud native teams adopt are increasingly defined by the cycle of inner feedback loops and outer-loop management practices. The path from code commit to production and beyond tells a story in itself of how open software has largely been developed according to continuous feedback cycles. Continuous integration platforms, continuous delivery technologies, monitoring software — they and many other categories have been built to continually drive the evolution of ever more accelerated software development.

But there always have to be checks on behaviors in workloads, tests and more tests to find the answers. In the end there are always more options for what can be tested and analyzed. It's an impossible quest to perform all the tests. Gaining deeper views into error handling and overall incident management is a hot touch point where the complexity of automated and declarative infrastructure can have its own chaos. The only answer is knowing how to manage it.

The people who define and evolve new practices and technologies for at-scale application architectures once had no choice but to build their own tools to manage unprecedented complexity. Today, the people who build open source projects have a deeper and broader community who are familiar with the technologies for at-scale architectures. New tools are plentiful, and open source sets the foundation for organizations to run distributed architectures across multiple cloud platforms.

The connections in all of this complexity are the practices that people follow to build the software that runs the internet. DevOps will continue to evolve to meet the practices teams follow and the requirements of increasingly different forms of workloads as cloud native technologies also evolve. Workflows will continue to change, influenced by newer techniques, largely developed in open source communities. The form that DevOps takes for any organization is first about the team. The team and its trust-oriented philosophies will determine the pace of automation and the iterative improvements that come through testing, delivery and management of services across distributed infrastructure.

Libby Clark

Ebook Editor, Editorial Director

Alex Williams

Founder and Editor-in-Chief, The New Stack

Sponsors

We are grateful for the support of our ebook sponsors:

CloudBees provides smart solutions for continuous development, integration and delivery by making the software delivery process more productive, manageable and hassle-free. CloudBees puts companies on the fast track to transforming ideas into great software and delivering value more quickly.

KubeCon + CloudNativeCon conferences gather adopters and technologists to further the education and advancement of cloud native computing. The vendor-neutral events feature domain experts and key maintainers behind popular projects like Kubernetes, Prometheus, gRPC, Envoy, OpenTracing and more.

Pulumi provides a Cloud Native Development Platform. Get code to the cloud quickly with productive tools and frameworks for both Dev and DevOps. Define cloud services — from serverless to containers to virtual machines — using code in your favorite languages.

SECTION 1

BUILD

Learn how containers, Kubernetes, microservices and serverless technologies change developer and operations roles and responsibilities.

Contributors

Jennifer Riggins is a tech storyteller, content marketer and writer, where digital transformation meets culture, hopefully changing the world for a better place. Currently based in London, she writes around tech ethics, agility, accessibility, diversity and inclusion, testing DevOps, happiness at work, API strategy, the Internet of Things, microservices and containers, developer relations, and more.

First there was the wheel, and you have to admit, the wheel was cool. After that, you had the boat and the hamburger, and technology was chugging right along with that whole evolution thing. Then there was the Web, and you had to wonder, after the wheel and the hamburger, how did things make such a sudden left turn and get so messed up so quickly? Displaying all the symptoms of having spent 30 years in the technology news business, **Scott Fulton** (often known as Scott M. Fulton, III, formerly known as D. F. Scott, sometimes known as that loud guy in the corner making the hand gestures) has taken it upon himself to move evolution back to a more sensible track. Stay in touch and see how far he gets.

CHAPTER 01

Doing DevOps the Cloud Native Way

Cloud native DevOps is not some trend or methodology that we all need to embrace and embody before it overtakes us like a tropical storm. It is an emerging set of principles in the DevOps tools community that describes how people work together to build, deploy and manage applications (apps) in a cloud native approach. Containers, microservices and Kubernetes not only are a very different set of technologies and tools built for cloud native computing, but more importantly, they change how people work. It is therefore not only pertinent, but important, that we observe cloud native DevOps as a phenomenon that amplifies the ongoing requirements for developer and IT teams to create workflows with technologies that maximize the organization's core business goals.

"Cloud native" approaches use containers as units for processes, allowing the running of sophisticated, fast and distributed infrastructure for developing, deploying and managing at-scale applications. The philosophy and the practice of building and running cloud native production systems means the need to build and run scalable applications in public, private and hybrid cloud environments. It embodies a "pan-cloud" approach that follows a service level agreement (SLA), which allows for interoperability. It is exemplified by tools such as containers, service meshes, microservices, immutable infrastructure and declarative application programming interfaces (APIs), according to The Cloud Native Computing Foundation's (CNCF) definition.

Because applications are deployed to the cloud, as well as built, tested and staged there using cloud-centric tools, it's tempting to define "cloud native DevOps" as DevOps practices applied in a public cloud environment. However, the application architecture, tools and workflows — and the intent with which they're used — matter more than the location when it comes to cloud native DevOps.

"A common misconception is that 'cloud native' means 'on the cloud.' It might be a bit of a misnomer. Lots of organizations have 'on prem' data centers that use Docker, Kubernetes, Helm, Istio, serverless and other cloud native technologies," Dan Garfield, chief technology evangelist for cloud native continuous integration/ continuous delivery (CI/CD) platform Codefresh, said. "Cloud native is more of a mindset for how application definition and architecture looks than a description of where those services are running."

Cloud native DevOps applies equally to cloud applications and those on premises. After all, DevOps is only DevOps when it automates the development and operation of all applications. Subdividing processes into the cloud native ones and the not-so-cloud-native ones runs the risk of rebuilding the silos teams have already disassembled. [1] Put another way, it would be wrong to create a DevOps for only one pool of development. It may be feasible to implement such a methodology if, and only if, the organization will develop only cloud native applications throughout its entire existence. But to restrain the organization to any single platform is antithetical to another of the goals of DevOps: to decouple processes from infrastructure.

There is no single DevOps template that can represent a "one-size-fits-most" approach. The same is true for so-called cloud native DevOps. It is a journey that all organizations should undertake, but which will yield unique methods for each one, and will result in the creation of pipelines and automation chains that not only fit each organization exclusively, but which will grow and evolve along the way.

> “‘Cloud native’ is more than ‘cloud only.’ It means bringing cloud-centric best practices to software and IT generally, whether that be in the cloud or on premises.”
>
> **— Jason Bloomberg, writing for Forbes after KubeCon + CloudNativeCon NA 2018.** [2]

DevOps — whether it’s cloud native, or not — is about your team and its workflows. Maybe there are ways in which a provisioning system, similar to a cloud native approach like Amazon’s, can help you create pipelines for part of the automation chain, or, in a different context, for a pipeline that pertains to a limited span of the application life cycle. But no cloud native chain can apply to all parts of that life cycle, especially when an application artifact emerges from testing. You may be jump-starting something with a cloud native, self-provisioning pipeline, and you can’t exactly say that’s not a valuable thing, but it’s not the entire toolchain, and thus, it’s not really Dev plus Ops. At the same time, just because a tool or platform does not encompass DevOps end to end, or because both departments use the same tool in different ways, does not mean that DevOps itself is not end to end.

What’s more, when an organization attempts an “instant DevOps” approach to jump start its journey — by adopting a tool or platform and expecting that to create a DevOps culture — more times than not, the effort will fail. It would be a mistake to present any DevOps methodology as though it could be adopted by a multitude. The best DevOps journeys have in mind the goals of improving performance and generating value for customers. There is no cookie-cutter approach to aligning application development with business objectives.

“It doesn’t really matter where an organization currently is … as long as they realize they are on a journey to continuously improve the way they work.”
— Kris Buytaert, Linux and open source consultant and DevOpsDays organizer. [3]

Cloud native DevOps is a flavor of DevOps where the principles of automation, faster iteration, and timely software delivery become applicable to an organization whose entire IT operation seeks to become more tightly knit, agile and competent in their work. Cloud native DevOps gets us there with greater efficiencies in automation, abstraction of infrastructure, revised workflows, improved metrics through observability and, yes, even tooling. The end result is a continued blurring of the lines between developers (Dev) and operations (Ops) — and increasingly other teams including business, networking and security, as we’ll see.

Containers, Kubernetes and DevOps

While organizations need not operate in the cloud to do cloud native DevOps, they must practice DevOps to be cloud native. Some would argue that an application isn’t truly cloud native unless it has DevOps practices behind it, as cloud native architectures are built for web-scale computing. DevOps professionals are required to build, deploy and manage declarative infrastructure that is secure, resilient and high performing. Delivering these requirements just isn’t feasible with a traditional siloed approach. Several advances in technology over the decade since DevOps was defined have led to increasing abstraction of the underlying infrastructure, and given rise to what we now call cloud native DevOps.

First, broad container adoption brought a significant shift in the traditional relationship between development and operations teams. With Docker, developers can focus on writing code without worrying about the system on which their code will run. Specifications for building a container have become remarkably straightforward to write, and this has increasingly led to development teams writing these specifications. As a result, development and operations teams work even more closely together to deploy these containers.

Containerization also led to significant improvements for CI/CD pipelines. In many cases, these pipelines can be configured with some simple YAML files. This pipeline configuration generally also lives in the same repository as the application code and container specification. This is a big change from the traditional approach in which code to build and deploy applications is stored in a separate repository and entirely managed by operations teams. With this move to a simplified build and deployment configuration living alongside application code, developers are becoming increasingly involved in processes that were previously managed entirely by operations teams.

But the real game changer has been the CNCF's open source container orchestration project, Kubernetes. As the de facto platform for containerized, cloud native applications, Kubernetes not only lies at the center of the DevOps transformation, but also enables it by abstracting away the details of the underlying compute, storage and networking resources. In addition to improving traditional DevOps processes, along with the speed, efficiency and resiliency commonly recognized as benefits of DevOps, Kubernetes solves new problems that arise with container and microservices-based application architectures. The New Stack's ebook, "CI/CD with Kubernetes," goes into more depth on the evolution of DevOps alongside cloud native application architectures.

Kubernetes provides a consistent platform on which containerized applications can run, regardless of their individual runtime requirements. With Kubernetes, your servers can be dumb — they don't care what they're running. Instead of running a specific application on a specific server, multiple applications can be distributed across the same set of servers. Kubernetes simplifies application updates, enabling teams to deliver applications and features into users' hands quickly.

Rolling updates and native rollback capabilities are just one example of how Kubernetes has evolved and improved DevOps workflows. Before Kubernetes, if you wanted to deploy something, a common deployment pattern involved the server pulling in the newest application code and restarting your application. The process was risky because some features weren't backwards compatible — if something went wrong during the deployment, the software became unavailable. Kubernetes solves this problem with an automated deployment rollback capability that eliminates large maintenance windows and anxiety about downtime.

A Kubernetes deployment object allows DevOps teams to describe a desired state, which the Kubernetes deployment controller will then match. Since Kubernetes 1.2, the deployment object is a declarative manifest containing everything that's being delivered, including the number of replicas being deployed and the version of the software image. These items are abstracted and contained within a deployment declaration. Such manifest-based deployments have spurred new continuous delivery (CD) workflows and are an evolving best practice with Kubernetes. The Linux Foundation has recognized this trend and, in partnership with 18 companies, formed the Continuous Delivery Foundation (CDF) in 2019 to provide a neutral environment to enhance the tools and best practices needed to speed and ease software development and delivery. To start, the first projects to be hosted by the CDF will be the popular CI/CD tool Jenkins and its cloud native sibling Jenkins X; Spinnaker, an open source multi-cloud solution; and Tekton, a set of open source components for building CI/CD systems.

As a result of these and other benefits of Kubernetes, operations are now less focused on the infrastructure and more on the applications that run light workloads. The combined effect is increasingly automated processes that yield better efficiencies. Serverless computing and container services, such as AWS Fargate and Microsoft's Azure Container Instances, are now seeing fast adoption, further abstracting the infrastructure so that operations roles become almost entirely focused on enabling automation, while developers deploy their own code to production, quickly test functionality against a business use case, and understand how to best use and integrate existing services with custom code.

Efficiency Potential Through Automation

At a June 2016 Public Sector Summit conference, Amazon Web Services (AWS) Senior Solutions Architect Alex Corley introduced the idea of cloud native DevOps by tying it to what he described as the philosophy of continuous business improvement. Corley explained that concept as inherently redundant, saying no business is ever truly done improving itself.

"This is not just about technology. It's about business and people," Corley told attendees. "To me, this is the essence of what DevOps is: We're all participating in activities that can continuously improve all functions of business, and involving all employees. You hear about the breaking down of silos, the integration of business ... By improving standardized activities and processes, we can become more efficient and reduce waste."

The cloud native aspect of DevOps becomes more practical for businesses, Corley continued, once it has been freed from the constraints of traditional, on-premises systems management. Theoretically, much of these "undifferentiated heavy lifting" work processes may be automated, in order to expedite them. But cloud native development, he asserted, advances the notion that busy work may be eliminated altogether. That elimination would, in turn, radically transform the requirements of DevOps, by way of changing the definitions of what Dev teams do and what Ops teams do.

This transformation is already well underway for those organizations that have successfully moved to microservices deployments on Kubernetes. The frequency of deployments and complexity of managing these systems requires new CD workflows and methodologies. Traditionally, development teams designed an application, passed it through an architecture review, and then handed off to the Ops team for deployment, said Nate Taggart, CEO and co-founder of Stackery, a serverless DevOps tool. [4] With the advent of microservice and serverless architectures — which we will explore in more depth in the next section — developers own their code, from design through production. Development teams are also writing code with observability in mind from the outset and are more focused on reuse, finding services that are already built to pull into the application rather than building services from scratch every time. At the same time, operations teams will often need to be brought into architecture discussions at the earliest stages of a project. Ops is also responsible for troubleshooting code in production with ever more complex scenarios that are unique each time.

"[Serverless] redistributes the responsibility of operations work and, in the grand scheme, it actually enforces the DevOps model," Taggart said. [5] "Serverless is fundamentally DevOps. It's developers having to iterate over operations work in the same cycle as their development work."

Organizations that have begun to further break down their microservices into functions that they then run on Functions as a Service (FaaS) and serverless platforms may come closest to realizing cloud native DevOps in its purest form. Though enterprises are mostly still testing serverless platforms for specific use cases, those tests are speeding the move to new organizational structures in which the role of traditional IT departments will disappear, said James Beswick, co-founder of Indevelo, which creates web and mobile applications completely on serverless architectures. Chris Munns, principal developer advocate for serverless at AWS, has gone so far as to predict that by 2025 most companies that are not cloud providers themselves will not need operations roles. [6] This “NoOps” future is still far away, but their predictions highlight just how closely cloud native DevOps and serverless delivery are bundled together.

“Really, what you’re talking about at this point [with serverless] is a platform that handles the release management, the life cycle management, the telemetry, instrumentation and the security around that component,” JP Morgenthal, chief technology officer (CTO) of DXC Technology, said. “It’s really a PaaS [Platform as a Service], but more so.”

Serverless architecture is “serverless” in terms of the user/developer never needing to take care of, or even be aware of, any individual machines — the infrastructure is fully abstracted away. Developers can simply tap into a practically limitless pool of compute, network and storage in the cloud via managed services. Serverless is also pay as you go, calculated according to actual consumption rather than pre-purchased services based on guesswork. [7]

The shift to managed services with a serverless architecture means that developers are essentially choosing their application infrastructure. This, in turn, makes the infrastructure much more ephemeral, as refactoring of the application also changes the infrastructure. It’s a concept that cloud native development platform provider Pulumi refers to as “infrastructure in motion:” Short-lived infrastructure is configured according to a constrained set of predetermined templates carried out by developers with limited operations involvement. [8] By contrast, “infrastructure at rest” refers to traditional provisioning and configuration management practices which are ongoing and require significant time and resources from operations teams.

“It’s a very different model of computing than what we’ve been doing in the past,” Morgenthal said, “and frankly, why would I want to then also have to go and invest at least hundreds of thousands of dollars in setting up all of that infrastructure to do the same exact thing?”

Morgenthal perceives the act of maintaining on-premises infrastructure as one class of “undifferentiated heavy lifting,” and there’s a good chance that Amazon’s Corley would agree. Corley’s use of the phrase harkens back to 2006, when Amazon CEO Jeff Bezos first defended his company’s concept of cloud computing at an O’Reilly conference. At the time, Bezos estimated that the amount of work expended by organizations in actually executing the core vision of their ideas was about 30 percent.

Amazon characterizes its cloud platform as the ultimate eradicator of undifferentiated busy work. Leveraging the public cloud as an automation tool, from Amazon’s perspective, offers an organization a kind of foundational repair; lifting up its infrastructure, cleaning out what Bezos calls the “muck,” and shifting it onto a single outsourced platform for everyone in the organization to use. Capital One developer Kapil Thangavelu, appearing in a 2017 The New Stack Makers podcast, argued that serverless architecture — and by extension, cloud nativity — refers not to the elimination or even the reduction of IT operators, but rather the concentration on “a common base platform.” [9]

Viewed from this perspective, one could argue that, for a DevOps platform to be completely effective, it actually must be cloud native — it should be constructed and located in an environment that is, from the outset, accessible to all, yet set apart from any one department’s silo or exclusive oversight. The point isn’t to automate away operations roles, but to further blur the lines between Devs and Ops with more cross-functional teams. Cloud native technologies are changing workflows and redefining Dev and Ops roles by increasing automation and bringing IT more closely aligned with end users and business objectives. We cover these roles in more detail in the next chapter.

5 Use Cases for Workflow Automation

Just as cloud native applications require DevOps to deliver applications quickly at scale, a certain level of automation is needed to manage cloud native

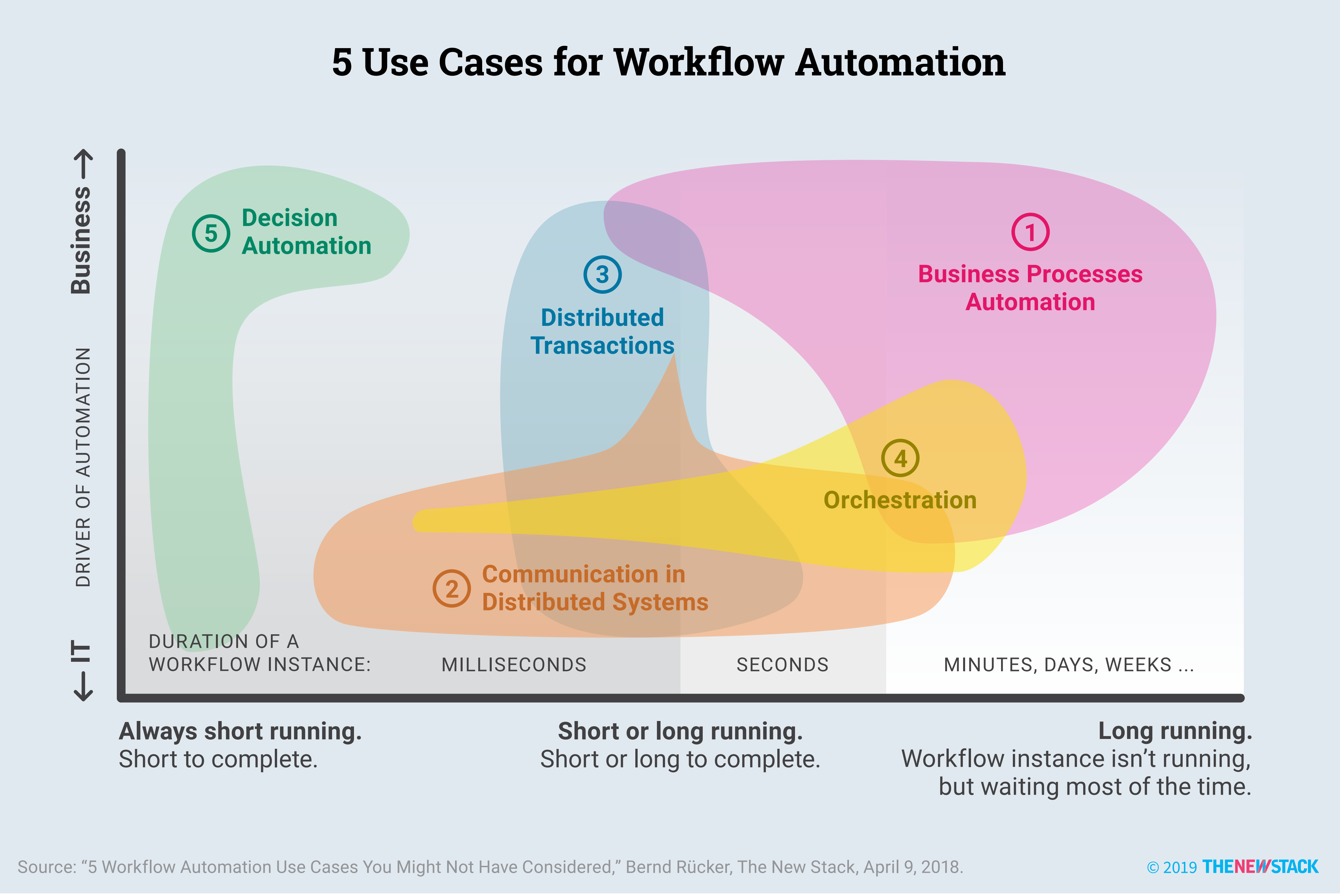

5 Use Cases for Workflow Automation
Business →
DRIVER OF AUTOMATION
← IT
5 Decision Automation
3 Distributed Transactions
1 Business Processes Automation
4 Orchestration
2 Communication in Distributed Systems
DURATION OF A WORKFLOW INSTANCE:
MILLISECONDS
SECONDS
MINUTES, DAYS, WEEKS ...
Always short running. Short to complete.
Short or long running. Short or long to complete.
Long running. Workflow instance isn't running, but waiting most of the time.
Source: "5 Workflow Automation Use Cases You Might Not Have Considered," Bernd Rücker, The New Stack, April 9, 2018.
© 2019 THENEWSTACK

FIG 1.1: *Bernd Rücker, co-founder and developer advocate at Camunda, illustrates how each use case ranks along two dimensions: whether business or IT is the main driver of the automation, and the duration of a workflow instance.*

applications due to the complexity of communications between microservices. Much of this automation is built into the various platforms and tools such as Kubernetes, CI/CD tools and Git, but there are standalone workflow automation tools that assist in this process automation as well.

Bernd Rücker is co-founder and developer advocate at Camunda, an open source platform for workflow and decision automation. Rücker sees five clusters of use cases for workflow automation technology, ranging from very technical use cases — like stateful retries in case a remote service is not available — to typical business processes like order to cash. [10] These are:

1. Business Process Automation.

Business processes implement important core capabilities of a company, like delivering goods or services a customer ordered: i.e.,"order to cash." Business processes are often long running in nature. They might involve straight-through processing/service orchestration; waiting for internal or external messages or timers, such as the promised delivery date or other events; human task management; order fulfillment; and more.

2. Communication in Distributed Systems.

Distributed systems have become the new normal in IT. Distributed systems are complicated because of the famous eight fallacies of distributed computing. Most developers are not yet aware of the magnitude of changes coming due to the fact that remote communication is unreliable, faults have to be accepted and you exchange your transactional guarantees with eventual consistency. Developers really have to adjust their toolboxes in order to cope with these new challenges. Automation can help; for example, retrying services invocations if the services are not available or not responding, waiting for messages such as an asynchronous response or an event, timing out when waiting for messages, etc.

3. Distributed Transactions.

You cannot rely on atomicity, consistency, isolation and durability (ACID) transactions in distributed scenarios. But some database providers, such as Amazon DynamoDB and PingCAP's TiDB, have started to offer ACID capabilities. ACID is what you experience from working with a typical relational database — begin transaction, do some stuff, commit or rollback. Attempts like two-phase commit — XA transaction — bring ACID to distributed scenarios, but are not really used much in real life as they do not scale: You still have to solve the business requirements of having a one-or-nothing semantic for multiple activities.

This is typically addressed by remembering which activities were already executed and invoking so-called compensation activities whenever the business transaction fails. A compensation activity semantically undoes the original activity, such as refund money you have taken from a credit card. It is important to note that this model accepts having temporarily inconsistent states, but makes sure everything gets consistent in the end. This relaxed view on consistency is known as eventual consistency and is sufficient for most real-life use cases.

4. Orchestration.

Modern architectures are all about decomposition into microservices or serverless functions. When you have many small components doing one thing well you are forced to connect the dots to implement real use cases. This is where orchestration plays a big role. It basically allows invoking components — or services, activities or functions — in a certain sequence. Examples include: one microservice invokes three others in a sequence, or multiple serverless functions need to be executed in order.

5. Decision Automation.

Decision management is "the wingman" of workflow automation. Of course, it is a discipline on its own, but from the workflow automation perspective it is a great tool to extract business decisions and separate them from routing decisions. Examples include: automated evaluation of eligibility or approval rules, validation of data, fraud detection or risk rating, and more.

Automation in these areas is freeing resources and breaking down silos between departments. But the automation itself can be a lot to manage, which is why organizations are seeking easy answers in the form of a single tool or platform for automation that is going to solve all of their problems. This leads straight to the question of what comes first, the tool or the process? Some argue that cloud native technologies inherently build in DevOps principles and processes and automate them. So should an organization undergo a DevOps transformation in order to do cloud native development and release? Or will that cultural transformation happen — will they be doing DevOps — if they use cloud native tools?

The Packaging Predicament

In an October 2016 webinar, Les Frost, senior technical architect at Capgemini, presented an argument that boiled down to this: The broader concept of DevOps cannot be ingested by an organization incrementally. It has to be swallowed all at once, or not at all. Thus the platform on which an organization's DevOps automation is maintained, would need to be a complete, singular mechanism — "DevOps in a Box."

And for that reason, Frost argued, the whole DevOps transformation thing may be a waste of time.

"If you spend the next three years implementing DevOps, will you actually be out of date?" asked Frost rhetorically. "Because while you're focusing on DevOps, there's a whole host of companies that are going to be disrupting the market. We're all aware of these sorts of disruptors ... So what you've got to ask yourself is, should we be spending our time on DevOps, or should we be spending our time on getting business-critical functionality out as quickly as we can?"

Frost is certainly advocating a counterintuitive idea. Frost went on to present microservices as one example of a technology that addresses the critical need to produce business functions in shorter, easier to implement, easier to test iterations. "These things that are happening in the IT market are all about getting stuff out quickly," he said. "And the reason people want to get stuff out quickly is that there are other people who will get stuff out quickly if you don't. So in that market, is it right to be spending all your time on DevOps?"

Each organization must make this decision based on its own needs, teams and processes. Microservices and serverless applications come with additional complexities that must also be considered. The user experience (UX) of the new environments may make some use cases better suited than others for developer teams and the workflows they follow. For this reason, taking an iterative approach is often recommended. Teams will typically face a steep learning curve due to the newness of the tools and platforms. And tools continue to change quickly.

Since a microservice architecture is a means to expedite software delivery in shorter cycles, DevOps skeptics will tell you it's actually an implementation of DevOps. As The New Stack's Alex Handy wrote in April 2018, "In the microservices world ... it's generally DevOps' duty to set up all of the infrastructure required to build out at-scale environments. That means web application servers, registries and repositories, OS [operating system] and container images, virtualized networking, firewalls, load balancers, message queues, and reverse proxies." [11]

That list sounds dangerously close to a recipe for what some would call "undifferentiated heavy lifting." And this is at the heart of Capgemini's counterargument: Software development is evolving toward the ability to produce a function or service based almost solely upon intent, without regard to the requirements of its infrastructure. It's a methodology that is, by design, more Dev and less Ops. It may be the separation of underlying functions which makes the serverless approach, and to some extent the microservices approach, valuable and viable. Why bother, the counterargument asks, investing in the time and effort needed to integrate tasks that are no longer relevant?

One of the dangers of the microservices approach, if we take this train of thought to its extreme, is that it could frame the entire scenario of an enterprise's software from the exclusive perspective of the developer. Since cloud native platforms are marketed towards developers' interests, the result is that, at the minimum, Ops professionals could feel left out. And at the maximum, they could be left out.

"The beautiful thing about being a software developer is, everything that I'm reaching out towards is controllable by code in some form or fashion," said R.

Tyler Croy, former Jenkins community evangelist at CloudBees. "From the operations standpoint, that's not the case.

"In a traditional data center environment," Croy continued, "I don't have an API to go provision a new purchase order for a rack of Dell machines; I've got to go through manual processes, fill out a spreadsheet, and some actual person has to come rack it, burn it in, provision it and then I can use that piece of infrastructure. From the operator world, they're looking at things where I've got to mix actual human, real-world, manual processes with my automation. And developers are tending to look at everything as software, so they can automate everything."

Developers automating everything is not the point of DevOps. And if a cloud native platform promotes automation as code, then it could feasibly enable developers to automate certain operator roles completely out of the picture, as AWS's Chris Munns has predicted. This is the dilemma teams face when they seek to adopt cloud native tools and practices. There is no complete automation and the process of improvement is continuous. The complexity is simply moving elsewhere, creating new processes to later improve upon and automate. Adrian Cockcroft, vice president of Amazon Web Services, compares the abstraction of complexity in distributed application architecture to squeezing a balloon. The air in the balloon just gets pushed somewhere else.

Today, developers and engineers are using software to manage distributed architecture and make infrastructure more programmable. In turn, these architectures are now far more fluid than ever before. Abstracting the infrastructure moves the complexity — and changes operations roles — but it still requires people with technical expertise who are familiar with issues of at-scale development and deployment. The people with this experience will follow DevOps practices to scale, gain the most efficiencies and deal with the chaos that can come when incidents arise.

A Single Cloud Native DevOps Platform?

Cloud native DevOps is a combination of changing roles and processes, as well as tooling that enables automation. As we've noted, much of this automation is enabled by containers and Kubernetes. But there is still much room for improved automation and, thus, improved DevOps processes, with DevOps tools that manage the build, test and release process on top of foundational cloud native technologies. Such tools should not only automate IT and businesses processes, however. They should enable the creation and nurturing of ideas generated by the human beings performing those functions, believes CloudBees' Croy.

A cloud native DevOps platform could include idea creation, which may better enable that idea to take root in the cloud. Thus it would be the ability for people to devise ideas and to innovate that could not only preserve their jobs but bring them into a closer-knit loop. Still, the focus of such a platform should be on enabling fast, iterative development that's consistent across multiple environments, said Marc Holmes, former chief sales and marketing officer at Pulumi. "The first step for a cloud native DevOps platform would be the consistency of the model to provide inner loop productivity [for developers] and outer loop management."

What business process engineers through history either fail to understand or have chosen to avoid mentioning, remarked Croy, is the reality that many ideas are bad. Relatively few actually mature all the way to the deployment phase. Yet this is what business methodologies, such as Agile, do try to take into account: Good products and services often develop from under the carcasses of bad ideas. It's only through the implementation process that many ideas may be called out as bad.

For a DevOps platform to deliver on one of its practitioners' stated goals of encouraging innovation, he went on, it must provide support for failure. That's not to say it should prune failed processes from the tree of development, but rather suspend their development until circumstances change. One or more bad ideas may converge into, or otherwise catalyze, one good one.

But by “platform” in this context, are we referring to a single application, like the business process management (BPM) environments of the 1990s? Or a multiplicity of tools interfacing through plugins and APIs — a kind of stack?

“I think this problem has to be addressed by multiple tools working in concert,” Croy responded. “Where we get into questions of religious affiliation with tools, or appreciation of one approach over another, I think there does need to be a conductor … The big question that is hard to answer impartially is, where should that overall problem statement — ‘problem identified’ to ‘problem solved’ — be modeled? Who’s the conductor of the orchestra?”

That sounds like the classic question of which tool lies at the center of the ecosystem. The various open source vendors will likely agree to disagree on which tool gets to play conductor. Yet, many have come together to form an open source foundation for this purpose. Still, even if the role of conductor remains for each organization to designate for itself, will it matter whether or not that conductor is “cloud native?”

“I don’t think it matters,” Croy answered. “I’m comfortable stating this now: There is not a single, successful business in the world right now that’s not relying on cloud native technologies in some form. … To me, that ship has sailed. I think the idea of owning everything is now passé, from a corporate standpoint.”

What does matter, offers Pulumi’s Holmes, is a consistent model that works across the entire CI/CD pipeline and addresses the need for general integration between the various tools. “Working in concert isn’t possible without a consistent model and leads to CI/CD headaches that aren’t solved by pipelines,” he said.

Cloud native DevOps is defined by increasing automation, the continuous improvement of the business and cross-collaboration between teams. Roles and processes evolve, and tools can help kickstart change, but also reflect that change. Regardless, a consistent platform is necessary, but the definitive final solution that meets everyone's needs has, as of yet, proven elusive. What that stack looks like, exactly, varies for each organization based on its business needs, historical structure and the perspectives and abilities of the people who lead and undergo the change.

CHAPTER 02

Cloud Native DevOps Roles and Responsibilities

Automate everything! This is the crux of the continuous delivery (CD) pipelines that make cloud native workflows fast, repeatable and manageable at scale. Automation, by definition, not only replaces the human in a process or workflow, but — crucially, in cloud native systems — arises from constant feedback in which processes are documented, examined and made declarative in application infrastructure environments. At scale, cloud native systems are impossible to manage through entirely manual processes. In an effort to reduce the complexity, the industry has developed ever greater levels of abstraction and is attempting to automate everything else. In theory, cloud native deployments can thus involve one person who selects services from a larger menu offered by a cloud provider and spins them up for a fraction of the cost of the on-premises equivalent, which would have required perhaps dozens of engineers and administrators.

And yet, for cloud deployments, automation does not necessarily replace any of the roles of the DevOps team members. Even if the work is shoved off to a cloud provider's operations team by employing managed services, internal IT teams are still critical for DevOps success. Dev teams are taking on some of the tasks previously handled by more traditional operations roles such as configuration and deployments, and thus reducing the Ops cost of every deployment. Meanwhile, Ops roles change to manage the complexity of the system as a whole. Automation can be difficult to set up and maintain — and still requires considerable people power. So operators spend their time coding compliance, security, performance and SLA metrics and making sure all infrastructure is available via automated self-service.

"The expectation is to have zero downtime for deployments," Ravi Lachhman, a technical evangelist for application monitoring platform provider AppDynamics, said. "With the increase in those moving parts, the human element has to keep up."

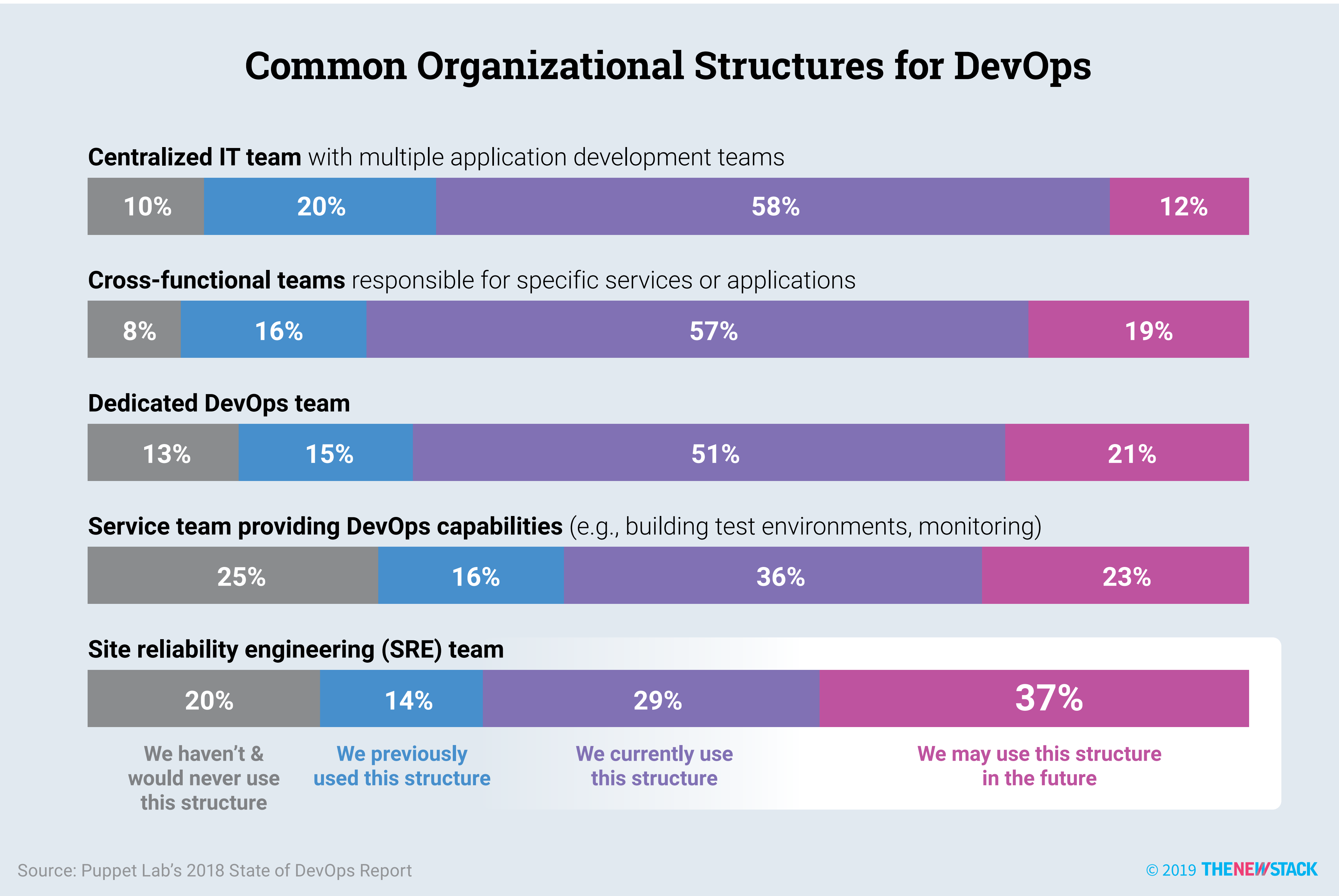
In short, developer and operations roles and responsibilities are changing with cloud native technologies. A decade ago, Lachhman had "no purview as a JEE [Java Enterprise Edition] developer in 2008 to much of the build — much less the networking and storage that was needed. Fast forward to today, and the Dev team can be packaging up connectivity items such as Istio and defining storage mounts inside a container," Lachhman said. "The traditional system engineering skills are moving into Dev, thanks to cloud native architecture."

Despite the added complexity, addressing the challenges associated with cloud native deployments and making the necessary changes in DevOps roles are not necessarily harder to do compared to traditional environments, Mitchell Hashimoto, founder and CTO of HashiCorp, said. However, the ability to deploy more applications quickly and therefore deliver more business value which comes with cloud adoption, also strains the existing systems and teams.

To do it right means bringing along automation tooling, building cross-functional teams and potentially hiring for different skills and/or more people. Thus, team structure is also evolving to reflect cloud native DevOps practices. A monolith culture divides developers and operations into separate teams. Cross-functional teams are comprised of Devs and Ops, or all of the roles needed for complete ownership over an application or service throughout its entire life cycle. Bringing Devs and Ops into the same teams increases transparency and makes room for resource and information sharing, which shortens feedback cycles and creates new dimensions for understanding the inherent trade-offs that come when building, deploying and managing software and systems. Often developers are moving into operations roles to solve the intricate issues they have witnessed as developers. Likewise, operations teams now are increasingly realizing they have to understand programming to keep up with the demand for scale-out architecture.

Centralized IT teams, cross-functional teams and dedicated DevOps teams are the most common organizational structures for those who employ DevOps practices, according to Puppet's 2018 State of DevOps Report.[12] However, more respondents expect to be using a site reliability engineering (SRE) team

FIG 2.1: *A centralized IT team, cross-functional team, and dedicated DevOps team are the three most common ways organizations structure DevOps now. But site reliability engineering will be more common in the future.*



in the future. And several of the experts The New Stack consulted agree that SRE is the future of IT operations.

"SRE straddles the line between the 'Dev' and 'Ops' sides of DevOps teams, both writing code and supporting existing IT systems," writes Kieran Taylor, senior director of product marketing at CA Technologies and author of "DevOps for Digital Leaders." [13] Traditional IT Ops teams take a "find and fix faults" approach, waiting for an alert to fire and then dispatching experts to troubleshoot the issue. SRE focuses on how to deliver the best experience possible across every touchpoint of customer engagement.

Site Reliability Engineers (SREs) did exist a decade ago, but they were mostly inside Google and a handful of other Silicon Valley innovators. Today, however, the SRE role exists everywhere. From Uber to Goldman Sachs, everyone is now in the business of keeping their sites online and stable. Microservices and SREs evolved in parallel inside the world's software companies. The SRE role combines the skills of a developer with those of a system administrator, producing an employee capable of debugging applications in production environments when things go completely sideways. Some would argue their fuller perspectives about the resources that are managed doesn't provide the more granular perspective that DevOps engineers need to manage individual services. But in an organization and infrastructure as large as Google, it's impossible for an SRE to have a complete view. Still, SREs provide context that a DevOps team working closer to the services may not have. [14]

In the case of microservices, the development team can "be narrower in focus than in the past," Steve Herrod, former chief technology officer of VMware and now managing director at General Catalyst, said. Each microservice team can thus develop and deploy their code independently of the rest of the application, he said. "In many cases, the site reliability engineering role has grown to bridge the gaps and coordinate monitoring and debugging across the whole application," Herrod said. "Great SREs thus require great technical skills, but also great interpersonal skills to coordinate across teams."

In the serverless space, the Dev and Ops engineers still have different roles to play as well. "Let us not forget, that doing serverless at scale entails multiple layers of complex configuration," Nuatu Tseggai, director of solutions engineering for Stackery, said. "[It also requires] expertise that relates to networking and distributed systems, of which Ops engineers have built up significant experience implementing solutions for over the past few decades."

However, at least semantically, DevOps does not define a team or a role, but instead, represents a culture, or a set of practices. A "team does both development and operations," Marko Anastasov, co-founder of Semaphore, said. It's about fast, repeatable processes and continuous feedback loops to bring the team and its outputs closer to the customer.

Defining DevOps With Continuous Everything

Regardless of how your organization's teams are structured, moving to a cloud native architecture requires looking beyond DevOps automation to the people who are planning and driving that flow, said noted business agility consultant Almudena Rodriguez Pardo, in her 2018 Agile Tour London talk. [15] Speaking from more than 20 years in the developer experience space in the telecommunications industry, Rodriguez Pardo says the first point of conversation when approaching any DevOps transformation must be clarifying for each department what it is you are even talking about when you say DevOps.

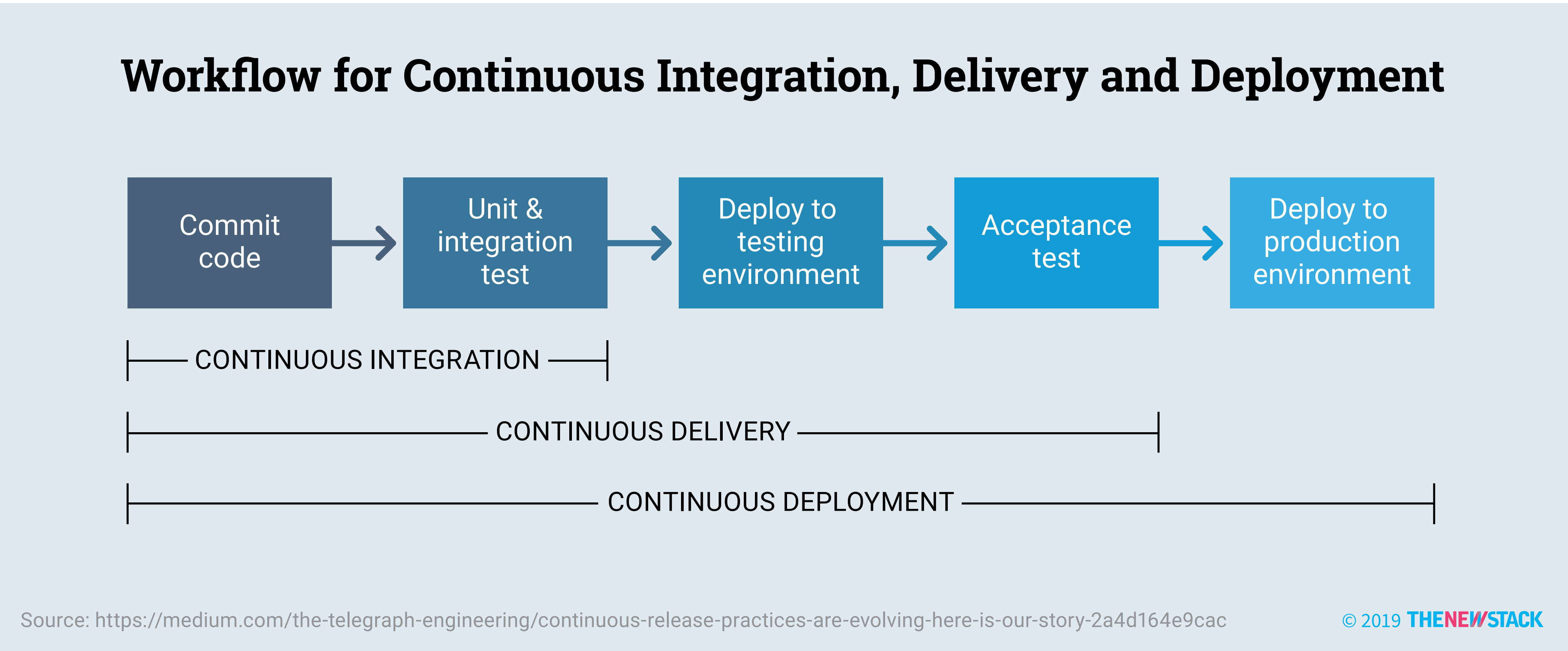
"Whatever definition you might have, you are not wrong because there is no official definition of DevOps," she said. "This is quite a problem, actually, because everybody is talking about it and everybody means something else."

While the Agile Manifesto offers guidelines to agile software development, there is no such thing for DevOps. Everyone agrees it means Development plus

Operations, but that's where clarity ends. Rodriguez Pardo's favorite DevOps definition comes from VersionOne: "DevOps is not a tool or a process, but a practice that values continuous communication, collaboration, integration and automation across an organization."

DevOps does have some generally accepted structures, however. Most companies undergoing a DevOps transformation, for example, have already embraced the Scrum framework, with clearly defined roles of scrum master, product owner and development teammates. Rodriguez Pardo says Scrum works well if you assume the product goes directly into the hands of the customer after it leaves the Scrum process. This is rarely the case. The majority of technology-backed organizations now also have continuous design and integration, but continuous delivery, continuous deployment and continuous release varies per company — as do the pipelines set up to regulate and automate them. Amazon or Spotify may be delivering three times per minute because they are directly delivering to a customer. However, a telecommunications company working with a network operator countrywide, with a few thousand people talking at the same time, may not want continuous release. Such high-complexity sectors, such as finance and telecommunications, may have several pre-production and production steps. Ericsson, for example, has a continuous improvement and feedback loop

FIG 2.2: *CI/CD workflows vary by organization, but follow a similar pattern depicted here.*


Workflow for Continuous Integration, Delivery and Deployment
Commit code
Unit & integration test
Deploy to testing environment
Acceptance test
Deploy to production environment
CONTINUOUS INTEGRATION
CONTINUOUS DELIVERY
CONTINUOUS DEPLOYMENT
Source: https://medium.com/the-telegraph-engineering/continuous-release-practices-are-evolving-here-is-our-story-2a4d164e9cac
© 2019 THENEWSTACK

integrated into its DevOps workflow; whatever is discovered in pre-production is factored into the next design iteration.

How continuous is continuous? That, again, varies by company and sector. Ericsson was able to move from releasing every six to 12 months to monthly. This may not quite be continuous, but for releasing into a UK-wide or even Europe-wide network, that's quite good, Rodriguez Pardo said.

"It's not about how continuous you are. It's about what are you heading for," she said. "Continuous everything" can be too ambitious for most companies. However, advancements, like virtualization and containerization, mean that every developer can get a "pretty operational environment." This consistency between development and production environments acts as an important DevOps accelerator.

> "Whether it's cloud native development or not, it means giving access to ephemeral cloud infrastructure and providing a developer with the tools that they need to test, commit code and go straight to production."
> **— Brian Dawson, DevOps evangelist at CloudBees.**

One of the higher-level achievements in a DevOps journey is continuous delivery because it enables releases that are frequent, fast and, above all, boring, writes Craig Martin in The New Stack's ebook, "CI/CD with Kubernetes." Focusing on deployments is a cultural shift for companies that involves changing team structures, changing processes and changing culture. The concept of "infrastructure in motion," as mentioned earlier, means that infrastructure becomes more ephemeral and provisioning matters more than configuration, thus continuous delivery becomes more important than continuous integration, argues Marc Holmes, former chief sales and marketing officer at Pulumi. Implementing Kubernetes on its own doesn't magically achieve CD for your organization. The features Kubernetes provides, however, in modularity, available tooling and immutable infrastructures, make CD much easier to put in place. Although Kubernetes will help you define a container deployment and manage instances, it leaves it up to you as to how you will automate those deployments into environments.

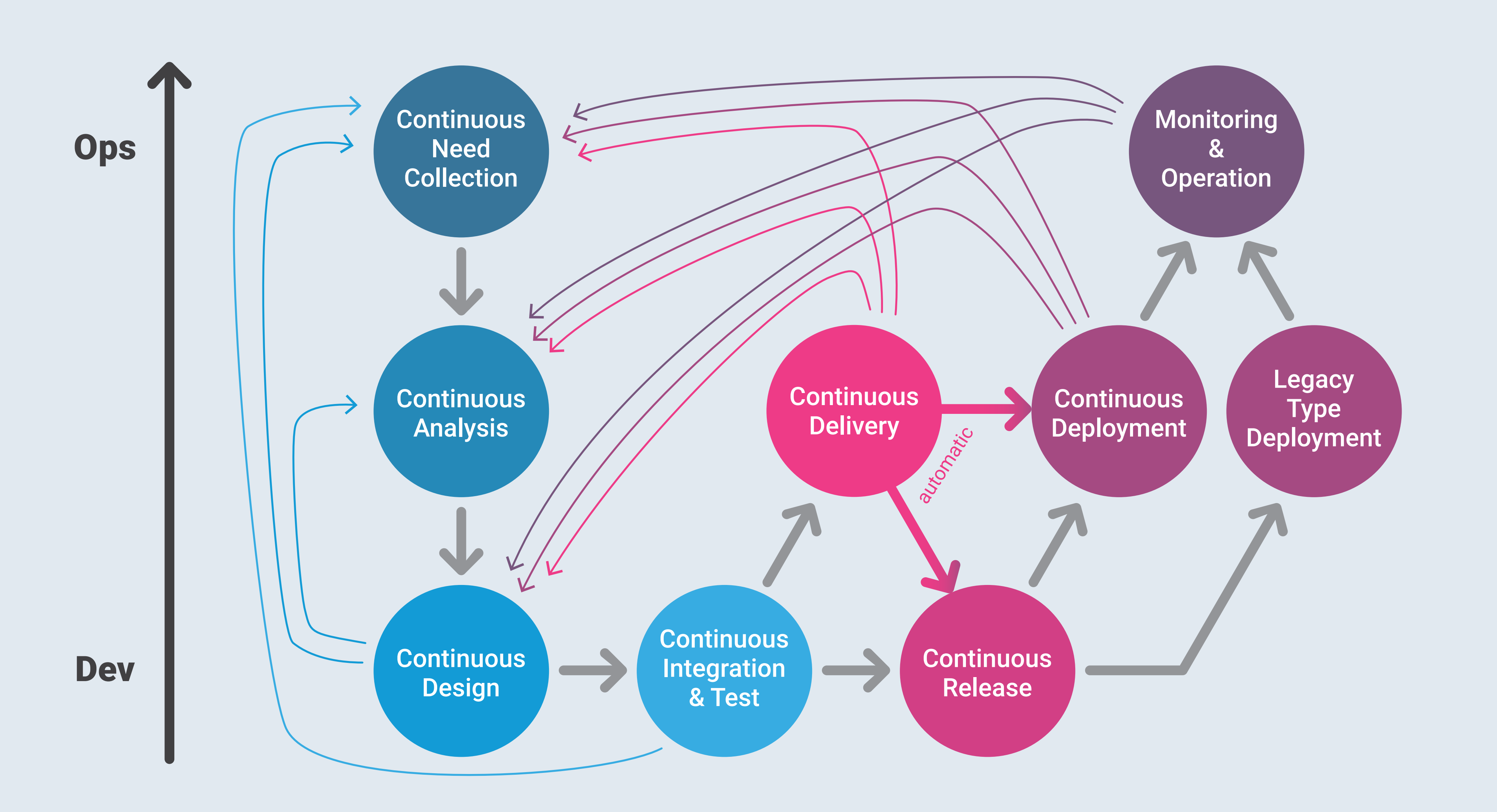
Mapping Your Continuous Pipeline on Human Interactions

The first step toward DevOps is automation — creating and optimizing that continuous delivery pipeline. That sounds logical, but it's usually not. According to Cisco's 2018 Connected Futures Report, only 14 percent of 1,500 senior IT leaders surveyed are responding to events using predictive analytics and automation. That number is expected to reach 33 percent within two years, according to the survey. [16]

> “Fueled by data and empowered by automation, IT can operate in real-time, be predictive and rely on detailed data to have a true seat at the table, delivering strategic value for their organization and for their customers.”
>
> **— Joseph Bradley, Global Vice President, IoT, Blockchain, AI and Incubation Business, Cisco.** [17]

Automation is not an option, agility consultant Rodriguez Pardo said. Just don't forget the human factor of that automation. A CI/CD pipeline has to include customer feedback and consider the distance and connection points to them. A particular process may need to be adapted for a customer, for example, when a developer needs to be on premises with a customer. She calls this “manual delivery,” which is the antonym of automation, but doesn't negate its value, since the whole point of DevOps is making customers happy.

A CI/CD Pipeline With Wide Feedback Loops



Source: "DevOps and People: Where Automation Begins!" Almudena Rodriguez Pardo, Agile Tour London, 2018.

FIG 2.3: *Successful cross-functional teams create many checkpoints for feedback throughout the software development life cycle. Each stage is also subject to continuous improvement.*

"By bringing your software personally to your customer, you sit with him, you hear his comments, you hear him bitching about something he doesn't like. You can have empathy and learn pain points," Rodriguez Pardo said.

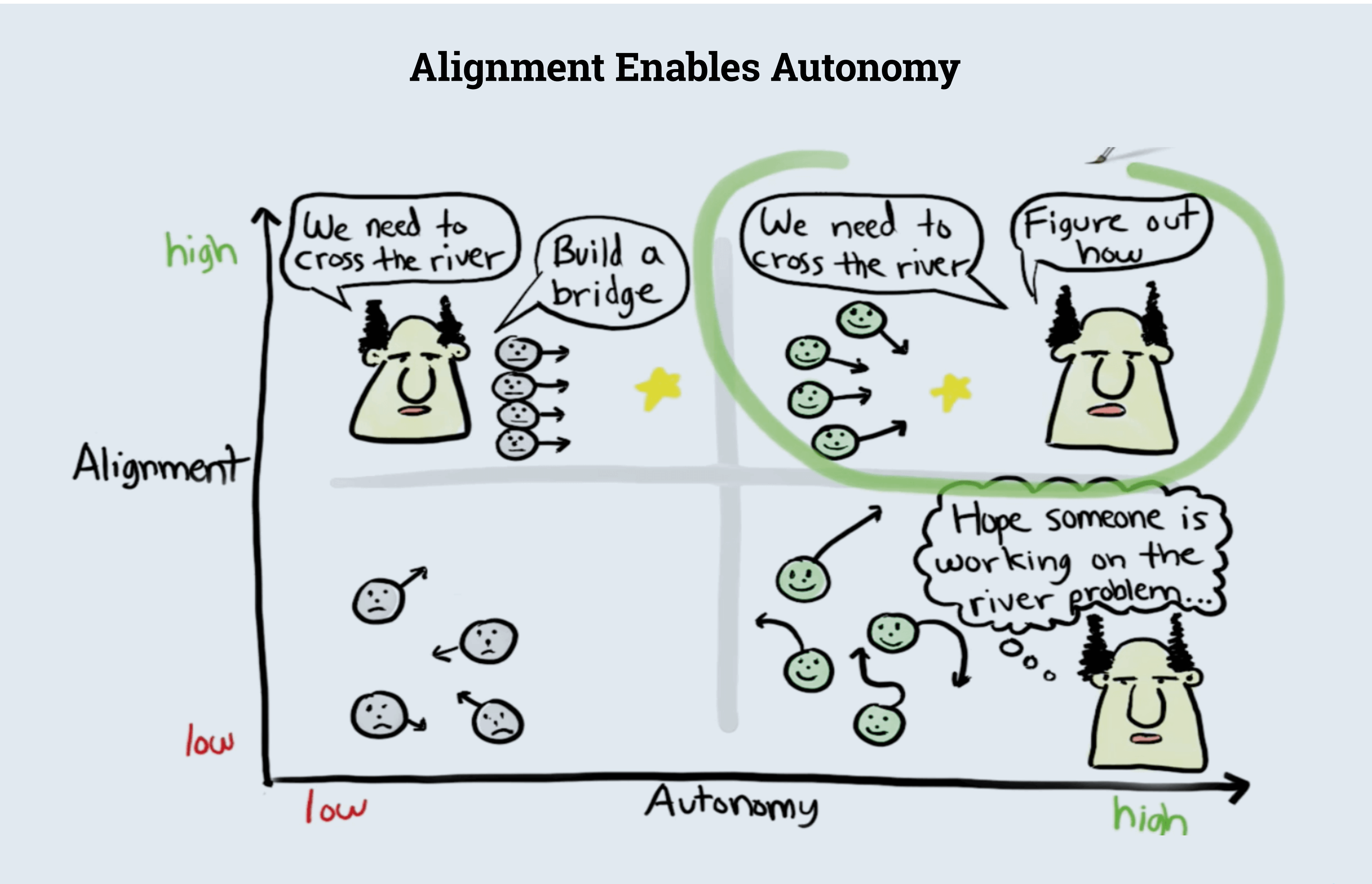
Whether Dev teams use Scrum, Kanban or something else, Rodriguez Pardo recommends including operations on those teams in order to unite around shared objectives. This helps create support and cross-functionality. The wider you map that feedback loop, the more operations are included in hearing it. And with everybody listening, you uncover sometimes absurd incongruities. On one team, Devs had key performance indicators (KPIs) to make as few mistakes as possible, while testing had KPIs to find as many mistakes as possible. Department managers then did something "revolutionary," Rodriguez Pardo quipped, and they talked to each other, deciding on common goals. By automating the delivery pipeline and bringing operations in closer collaboration with developers, the entire DevOps team output comes more in line with business objectives.

"Cloud native continuous deployments are a tremendous opportunity for corporate IT to upgrade their role from 'the [people] who keep the lights on' to the 'architects behind strategic business success,'" Torsten Volk, an analyst for Enterprise Management Associates (EMA), said.

DevOps: A Balance of Specialization and Cross-Functionality

DevOps isn't just about breaking down silos, but about developing skills. This usually involves specialization in verticals, but then having a general understanding of everything horizontally, from a brief comprehension of tools,

FIG 2.4: *This illustration of Spotify's agile engineering culture emphasizes small, cross-functional teams with autonomy to make decisions within the bounds of their mission.*


Alignment Enables Autonomy
high
Alignment
low
low
Autonomy
high
We need to cross the river
Build a bridge
We need to cross the river
Figure out how
Hope someone is working on the river problem...

Source: Spotify Labs as presented in "DevOps and People: Where Automation Begins!" Almudena Rodriguez Pardo, Agile Tour London, 2018. © 2019 THENEWSTACK to which team does what, to transparent goals and tracking.

To achieve this, teams can start with small testing experiments for developers, or give everyone some basic workshops in Jenkins to cross-pollinate continuous integration. The purpose is to attract some interest and give them a taste, applying ideas like communities of practice and other cross-company guilds. The end goal is best illustrated by the Spotify diagram above. [18]

“With the operators’ support, they are aligned. They are mature within the teams and understand what they have, what they don’t have, and where they can collaborate,” Rodriguez Pardo said, adding that everyone is autonomous, secure and heading toward delighting the customer.

In the end, DevOps is just about people, both the customers and the teams. When hiring for the cultural change of DevOps, you are no longer just hiring the best in a certain language, you are hiring a personality and how she can adapt to change and work well with not only her own team, but with other teams as you head toward cross-functional knowledge.

> “One thing we cannot forget is that whenever it looks like a technical problem, whenever we think we have a technical issue, it’s always a human problem at hand.”
>
> **— Almudena Rodriguez Pardo, business agility consultant.**

When HSBC shifted to cloud native deployments, the bank employed a new hiring strategy which ultimately helped lead the DevOps transformation there as well. Cheryl Razzell, global head of platform digital operations for HSBC operations, described during DevOps World | Jenkins World 2018 in Nice, France, how the banking firm’s DevOps embraced a new way of thinking. “There’s been a massive cultural shift to adopt agile working DevOps, but I think the bank really wants change so it’s embracing this change, so it’s open to challenging new ideas,” Razzell said. “HSBC acquired people from outside of the banking sector on purpose because they want to change their culture from within and they want to bring out new ideas."

HSBC recruited people from Google, Amazon and Microsoft, from startups and from different backgrounds and different sectors from outside of the banking sector to complement the teams that already understand how the bank's networks worked. "There's synergy between the teams with new ways of working versus some of the bank's legacy products and some of the bank's processing," Razzell said. "I think you need both to complement the journey."

Cloud Native Has a Developer Focus

Now that we have a common definition of cloud native DevOps that involves continuous improvement, automation, cross-functional teams and better alignment with business needs and customer expectations, let's look at some common patterns that are emerging in Dev and Ops roles with cloud native technologies. Such roles are fluid and highly dependent on the organizations and structures within which they are implemented. However, there are some emerging trends which we can examine.

As stated previously, the facility and speed at which software can be deployed on cloud native platforms require a shift in roles, so that operations-related tasks do not get in the way of delivery cycles. Central to accomplishing this is the cloud native concept of declarative infrastructure. Automating everything is made much easier if your infrastructure is managed as code and configuration by creating consistencies and parity across all of your environments. Accomplishing this is dependent on your specific implementation, but cloud native development and consulting company Kenzan typically uses some sort of scripting, such as with Terraform, and also employs dedicated infrastructure pipelines to automate the deployment. This ensures that individuals are never manually hand tweaking environments without the proper checks and balances in place. [19] Next generation, cloud native IDE tools such as Pulumi and HashiCorp's Terraform aim to improve upon the developer experience by creating an interface to write code and configure it to work across multiple environments.

"It is the mission of modern IT operators to get out of the critical part of the continuous release process. This requires adopting a declarative paradigm to managing data center and cloud infrastructure, instead of manually responding to ad hoc requirements," EMA's Volk said. "This declarative management approach also requires IT operations to understand the basic principles of coding, without becoming a pure-blooded software developer."

The rise of declarative infrastructure, with containers and Kubernetes, places configuration alongside code, opening traditional operations roles to developers. "Infrastructure" requirements — the set of services that apps are built upon — are increasingly being decided by developers and DevOps teams, who are making extensive use of automation to manage the available portfolio for their enterprises.

Devs are empowered to release software into production, which drastically increases the rate of deployments as the barrier to deployment is reduced or eliminated entirely.

This ease of deployment necessarily creates security implications, and so developers are increasingly responsible for making sure they are building with security in mind as well. Once code is released, developers are also more directly tied to business results, as better tooling and measurement help track customer response. [20] Developers must understand the end user or customer and adjust accordingly and are more likely to kill dying projects earlier. They must be more responsive to how their code performs in production and make adjustments based on user feedback, as well as help troubleshoot when things go wrong. As a result, developers have adjusted their workflows to "release early and measure" or test in production.

"Instantly and continuously validating new software features on a small group of production customers, without causing significant user experience deterioration, becomes the most critical skill for cloud native developers," Volk said. "Exactly defined and comprehensive user experience metrics and exact monitoring instructions for these metrics are becoming part of each build. This enables developers to validate that customers are actually using their software in the intended manner and that they are completing their desired transaction in an efficient manner."

To put this another way, from an article on the changing role of developers: "If you're not already releasing features to a small segment of your customer base, you're already behind," writes David Hayes, former product leader at PagerDuty.

Cloud native DevOps is developer-centric by nature. Developers are drawn to the utility and packaging capabilities in containers, which improve efficiency and fit with modern application development practices. Developers use application architectures with container technologies to develop services that reflect an organizational objective. Kubernetes runs under the application architecture and is used to run services across multiple clusters, providing the abstraction of orchestration to manage microservices following DevOps practices. [21]

"Those involved in cloud native development life cycle patterns and practices assume they're operating in the silo-less environment. But many organizations are still trying to figure out how to properly integrate all of the teams within DevOps," Brian Dawson, a DevOps evangelist at CloudBees, said. "For your standard application, there are operations and security [people], for example, who haven't yet figured out how to inject themselves into cloud native development."

The change in operations roles that comes with this new abstraction of storage, network and compute resources isn't always clear. While some technology thought leaders have declared a “NoOps” future, the reality is a lot more nuanced. The next section examines the operations role of cloud native DevOps in more detail.

“The role of ‘Ops’ hasn’t been eliminated, but rather, it’s been shifted,” Stackery’s Tseggai, said. “Exactly how and where is a matter of the engineering culture and vision within the organization. If an engineering org were to foolishly rebrand DevOps into NoOps, they would be running a real risk of alienating engineers whose skills are in high demand.”

How Operations Changes in Cloud Native DevOps

Containers and microservices represent a radically different computing environment compared to monolithic deployments. As a result, operations teams are faced with trying and testing new and different approaches, not only for the development pipeline but for networking and data storage which require new skills and workflows. Storage and database access, for example, must take into account every microservice, which can often number in the hundreds or possibly thousands of individual instances. This can create performance and reliability problems; compliance issues; and backup, data recovery and archiving difficulties, as traditionally stateless containers and microservices have become stateful entities. [22] Similarly, cloud native architectures bring a whole host of connectivity problems — issues such as setting up containers dynamically, assigning IP addresses, setting up the networking, figuring out how each microservice talks to the others, the latency and the retries, said Ratan Tipirneni, Tigera president and CEO. [23]

The new challenges with cloud native architectures are leading to new ways of thinking about storage, with persistence as the foremost concern. That means there must be data backup and recovery solutions. Companies such as Portworx and StorageOS are at the forefront in addressing these concerns with their container backup approaches. What this really means is that the IT team needs to take more time considering storage than it used to, when delivering storage involved installing and formatting hard drives and telling the Ops team how much storage was available.

Switching the operations team to a cloud native approach can be difficult, however. Things don't always go well and then senior leadership has a tendency to overreact and abandon it. But, there is another way, which we will cover now.

Damon Edwards, co-founder chief product officer of Rundeck, offered a new model for operations in a cloud native context in his London DevOpsDays talk, which he kicked off with a familiar and painful story. An unnamed organization takes on the latest technologies — the cloud, Docker, microservices, Kubernetes — only to scramble in the complexity when something breaks.

First, a bridge call happens that just generates a lot of questions. The lead Dev escalates it to the scrum master. It can't be figured out. The questions get bigger and bigger and now everyone is on the phone and on a growing ticket — network engineers, business managers, app managers, lead developers, an SRE, system administrators, middleware managers, the senior vice president (SVP), chief of staff, two technical VPs, more middleware folks ... the list goes on.

Eventually, the whole network is down, so the operations team is focusing on that. Somebody else realizes the original outage was all because of a changed firewall.

"Can you change it back?"

"OK fill out a form and we'll do it next week."

"But it's an emergency!"

Finally, the company adds a customer engagement manager to the case "who has to test it all instead of trusting SRE," Edwards said.

When all is fixed, the next day the SVP wants to know:

- What happened?
- Whose fault is this?
- What processes and approvals can we add to keep this from happening again?

"They had digital, agile, DevOps transformation, and site reliability engineering. On the tech side, they had the cloud, Docker, Kubernetes, and microservices. Then the exec team asks why everything takes too long and costs too much," Edwards said.

But in all of these shiny tools and cultural transformations, operations was just kind of ignored.

Edwards says conventional SVP wisdom says:

- We need better tools.
- We need more people.
- We need more discipline and attention to detail.
- We need more change reviews and approvals.

He argues we need to forget about these four pillars of this so-called conventionalism and "really challenge the wisdom of operations work" by going after what he calls the four horsemen of the operations apocalypse:

- Silos.
- Ticket queues.
- Toil.
- Low trust.

Let's break down how to keep those demons at bay.

1. Cross-Company Silos

Edwards calls this just a different way of working where silos, or traditional departmental divisions, are torn down and everyone in IT shares:

- A common backlog.
- A common tooling.
- A common context.

Ideally, everyone uses tools the same way and everyone shares a common set of priorities. This way, when teams need to work together, they align on capacity, context and process. This also enables feedback loops, learning and higher quality work. Edwards continues that this isn't just happening in development and operations, but in the environment, network and on customer teams.

2. Ticket Queues

"We create a ticket queue to solve silos," Edwards said. He referenced back to the story above, saying "Then I wait for something. I don't know what I'm asking. I'm not a firewall engineer but I'm typing into a blank box trying to explain what I need."

He says this makes queues not only slow down business processes and DevOps work, but ticket queues are expensive. Donald G. Reinertsen's talk on his book, "The Principles of Product Development Flow", lays out the problems that queues create:

- Longer cycle times.
- Increased risk.
- More variability.
- More overhead.
- Lower quality.
- Less motivation.

After all, the longer people have to wait for something, the more detached they become. The end goal becomes split and obfuscated. Each goal becomes like a snowflake — unique, brittle, technically acceptable, but not reproducible. This makes it much harder to automate things.

"The only thing worse than automating things that are broken is automating something that's just a bit off," Edwards said, pointing to ticket queues as a huge contributor to bottlenecks.

Ticket queues are further aggregated — and aggravating — when they push primary management focus to protecting team capacity and when operations repeatedly says no. He says the latter is interpreted as Ops being afraid of change, but a lot of times they are just trying to protect capacity.

3. Toil

"Toil" is a more common term now because of the growing popularity of SRE, especially in the DevOps world. First, let's distinguish between toil and overhead. Overhead is important work that doesn't directly affect production services. It may be anything from setting goals to human resources activities to team meetings — important things that don't necessarily affect the code. On the other hand, toil typically includes things that are:

- Manual.
- Repetitive.
- Able to be automated.
- Not strategy or value driven.
- Repeatedly waking on-call developers up.
- Non-creative.
- Not very scalable.

"It may be necessary, but it should be viewed as something a little bit icky," Edwards said.

At Google and many companies, managers try to keep toil down to less than 50 percent of the SRE team's work. It isn't moving the company forward and it can frankly be demotivating to your engineers. Google particularly points out their fear that greater toil means that SREs will fall into a strictly Ops or strictly Dev role, while they should be working with both.

"You want to keep that toil at a manageable capacity because engineering is important for two things: to add business value and to reduce toil so you have more time to improve the business," Edwards said.

4. Low Levels of Trust

Where are decisions made? How do we escalate issues up to make decisions? Edwards says all work that is done is contextual and answers always depend on something. This is particularly true when you're working with complex distributed systems. Yet the people who create the code still aren't often the ones making major decisions.

In DevOps, you need to revisit how much decision-making actually needs to be escalated up versus how much power should be entrusted with the people actually adding business value and working with the tools.

Steps to Transition to Cloud Native Operations

To scare off the four horsemen of 'Opspocalypse,' the first step for Edwards is creating cross-functional teams, breaking down as many silos as possible. Developers and operations have shared end-to-end responsibility for a service. Though, that doesn't mean that everybody does everything. Netflix says, 'There is no DevOps' — it's all about teams that have cradle-to-grave responsibility. The Google model is a Dev and and Ops hybrid with clean hand-off requirements from development to SRE, as well as circumstances where SRE can push back to development. Both sides have the same emphasis on quality.

Now, Edwards says you can't get rid of ticket queues completely. Organizations just have to be aware when they are being used as a general purpose work management system.

"Tickets are really good at documenting true problems, issues, exceptions and routing for necessary approvals," Edwards said. "The idea is that you cut down on all the interruptions."

Overall, Edwards says successful DevOps is about shifting the ability to take action leftward toward the build end of the pipeline, giving everyone the same tooling and enablement for a safer pathway to do things.

"The whole idea behind the cloud is everything is self-service. So, whether you automate it with a pull request or you click a button on a UI, it's a way everything should work now really, by no longer raising a ticket and waiting three months for a VM [virtual machine] to show up," James Strachan, senior architect at CloudBees and the project lead on Jenkins X, said.

Another key to staving off the Opspocalypse is transparency: tracking toil levels and sharing these levels with the team. Edwards echoes Google's work to set toil limits to 50 percent, but says that organizations need to fund efforts that actually actively reduce that toil. David Blank-Edelman's book "Seeking SRE" is a good resource to learn how to scale toil-reducing activities.

Finally, Edwards warns that you should challenge that aforementioned conventional wisdom and bring operations into your digital transformation strategy and take time to understand how the four horsemen are undermining Ops work. Focus on taking down silos and limiting queues for them as well. Encourage them to focus on self-service Operations as a Service (OaaS) as much as possible.

"We used to have a separation between operations and development teams that's no longer there. I attribute this to what I call a 'NoOps' approach that's come in response to complexity," Andreas Grabner, DevOps activist for Dynatrace, said. "NoOps means we're shifting who's responsible for things, as automation comes into play. We're seeing traditional Ops teams moving more toward helping modernize their organization, providing more services to the rest of the organization."

While many traditional operations tasks are "shifting left" to developers, the NoOps concept is more about creating layers of abstraction and self-service capabilities than removing the need for DevOps. Operations roles are still critical for building and managing those systems that deliver agility and create business value. While a monolith application might be broken down into microservices and serverless functions, it requires new thinking and opens the door to new problems.

"DevOps doesn't just become 'NoOps' — that is entirely the wrong way to think about it," said Chad Arimura, vice president of serverless at Oracle. "In fact, serverless makes DevOps even more important than ever." [24]

CHAPTER 03

Cloud Native DevOps Comes to Security and Networking

DevOps is a hot cultural and technical trend in most digitally focused companies. While three-quarters of IT leaders are at least considering using DevOps, 23 percent are already using DevOps in production and refining these processes, according to a 2018 Tek Systems report. [25] What is most notable is that "transformative leaders," those companies in which IT is an "impactful and dynamic part of and partner to the business," are 47 percent more likely to be at this stage as compared to everyone else. Among those using microservices in production, 81 percent are also following continuous delivery or DevOps practices, according to a DZone survey. [26]

The purpose of DevOps is to break down the walls — both physical and metaphorical — between software developers and the operations teams running the infrastructure. When successful DevOps allows building, testing and deploying to happen in tandem so that reliable releases occur more frequently and quickly. This leads to a cultural change as roles and processes evolve and silos fade into cross-functional teams. DevOps also leverages tooling to support cross-functional interoperability and automated workflows that back continuous development, testing, deployment and integration.

This increasing automation and blurring of the lines is happening between other more traditionally siloed IT roles as well, most notably in security and networking, which require new tools and approaches to management in cloud native environments. In a traditional three-tier application, the application tier, business logic tier and data tier talk to each other via a load balancer, and networking and security policies are administered from a central place — typically the load balancer or a firewall, writes Jeroen van Rotterdam, executive vice president of engineering at Citrix. [27] The distributed nature of microservices architecture makes administering networking and security policies a lot harder than it is in a monolith architecture, he said. Containers appear, disappear and are moved around to different compute nodes far too frequently to be assigned static IP addresses, much less be protected by firewalls and IP tables at the network's perimeter. And unlike monolith architectures where modules communicate with each other within a single image, either through in-process calls or interprocess communications (IPC), with microservices these calls are made over the network. More services and service instances potentially mean more failure points and more operational complexity; there are a lot more service instances to load balance, the short lifespan of service instances makes health checking these ephemeral containers a challenge and the sheer number of service instances calls for a high degree of automation for managing rolling upgrades at scale. Manual or semi-manual processes don't work when dealing with containers at scale. Fortunately, microservices architecture is flexible in its design to enable continuous integration and delivery, unlike the three-tier architecture.

Applying DevOps to networking and security — alongside new tools and best practices — helps organizations manage the increased operational complexity that comes with a containerized, microservice architecture. DevOps adoption changes well-established security practices, for example, because its emphasis on automation and monitoring assures that flaws are found more quickly and patches and updates are released continually. The patterns and practices which emerge when DevOps and security are combined is now commonly referred to as DevSecOps. Application and infrastructure security must keep up with the speed and scale at which software is deployed in a cloud native context. As a result, security is increasingly automated and it's no longer the strict domain of security experts, but of operations, developers and system architects as well.

"Security has to be part of the CI/CD pipeline," said Dr. Chenxi Wang, founder and general partner of Rain Capital, an early stage cybersecurity-focused venture fund. [28] "In the past with the major releases, you have your security reviews and security testing all done. Everything else stops until you finish that. That doesn't work anymore, so vulnerability scanning has to be done in a way that fits seamlessly into the CI/CD process."

Similarly, agile DevOps practices are being applied to networking, bringing network engineers and tooling into this mix. NetDevOps takes the focus of culture, collaboration, tooling and automation and extends it to network architects and operators and the changes made in their networks. In cloud native architecture, the network itself is being abstracted up the stack and into the application layer. Cloud native platforms, like Kubernetes, expose a flat network that is overlaid on existing networking topologies and primitives of cloud providers. Operators can set network policies that are accessed by developers and resource administrators. [29]

The shift to containers and Kubernetes has begun to change networking and security roles in much the same way that it has changed developer and operations roles. In fact, these two trends are connected at the hip: 40 percent of network managers report fully converged, shared tools and processes with the security group, with another 51 percent reporting formal collaboration, according to EMA research on what they aptly call "NetSecOps." [30]

While not every organization has integrated security and networking into its CI/CD pipelines, many are beginning to experiment with the approach. As the rest of the organization becomes more agile, collaborative and automated, teams find more ways to work together to ease remaining bottlenecks and further break down silos between teams. And as more advanced cloud native tooling, such as automated vulnerability scanning and service meshes, is adopted, organizations can expect the trend to accelerate.

The abstractions provided by service mesh, for example, offer powerful separations that help developers, operations and security manage a microservice architecture, writes Dr. Wang. [31] The data plane abstracts the underlying network from the application. The control plane has Layer 7 insight and can instruct the data plane to make complex routing decisions based on policies, security postures and real-time telemetry information. It further abstracts away the decision engine logic, which means the data plane can focus on being the high-performing traffic interceptor and router. Together, a service mesh can make intelligent, dynamic routing decisions automatically without requiring any changes to the application code. Another fundamental concept in service mesh is service identity. That is, each service is assigned a cryptographically strong identity. Managing services in the context of strong identities enables fine-grained, identity-based policies that previously were not possible to enforce.

“As things get pushed to Layer 7, things are also getting more modular in the sense that application developers can focus on the application. They don’t have to worry about networking, they don’t have to worry about security, and security and networking functions are being encapsulated into sort of manageable units, itself being microservice driven,” Dr. Wang said.

The Start of NetDevOps

Cisco, from its position as a top networking hardware provider, is one of the companies leading the NetDevOps move from theory to practice, starting with some of its customers.

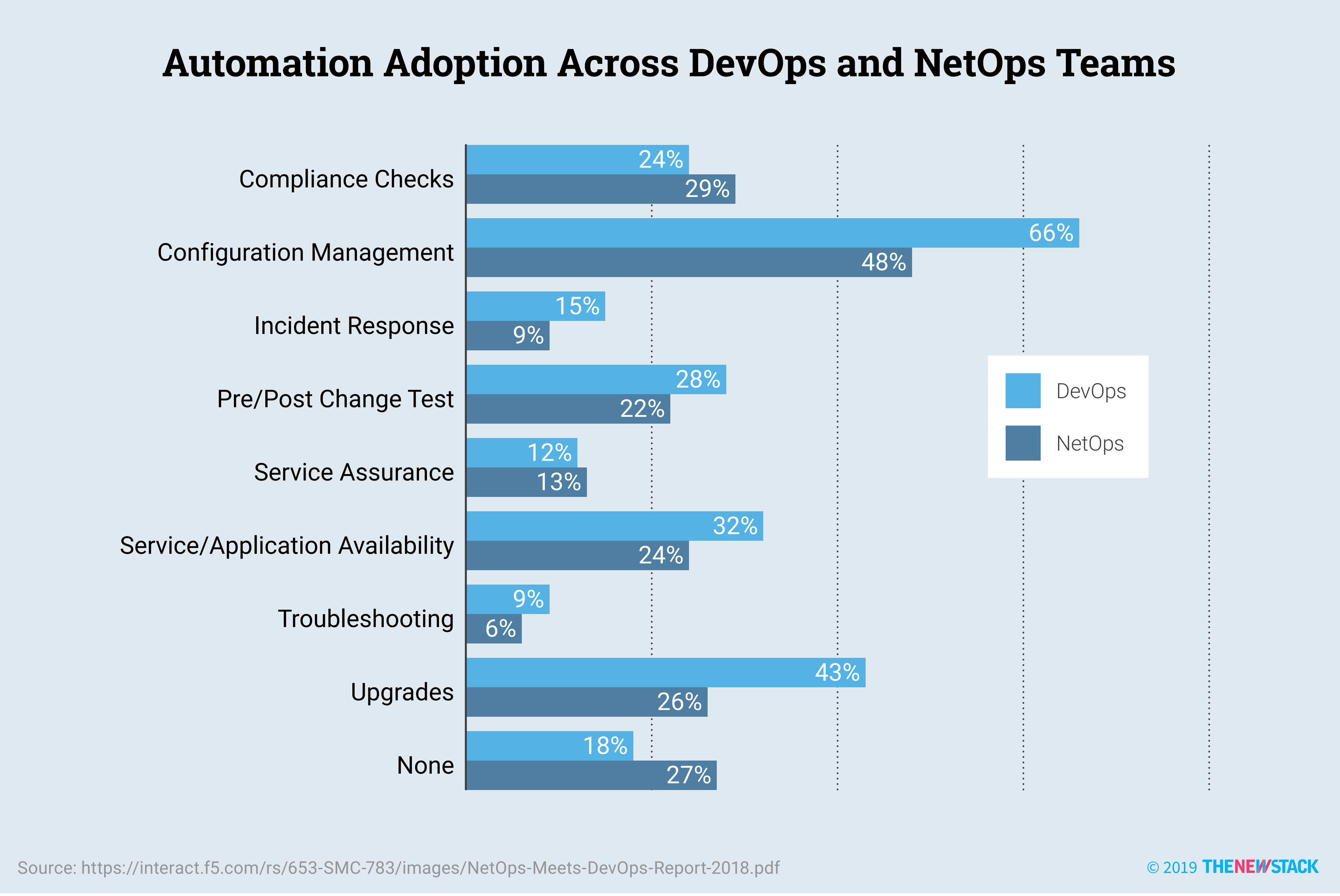
“We’re talking to [customers] about using Infrastructure as Code (IaC) and moving toward automation and scripting,” said Amanda Whaley, senior director of developer experience at DevNet, Cisco’s developer community, where app developers and infrastructure engineers can come together and learn about Cisco APIs and how they tie into intent-based networking.

So, what’s the first step for network’s parlay into DevOps culture and automation? Whaley says it’s all about making sure your routers and switches are connected via APIs, allowing them to be programmable. Once that’s in place, then you can add assurance, analytics and observability into the mix, starting to apply DevOps processes and principles to the network.

“Use a regular CI/CD pipeline [for] storing networking configuration as code in your source control,” Whaley said. “Then you can deploy these changes to a test network and then deploy them to production, applying consistent changes to network changes as well.”

Kubernetes enables this “policy as code” approach to managing network security controls with its NetworkPolicy resource, which defines egress and ingress rules to specify what traffic is allowed to selected pods. “Security can be built into the CI/CD pipeline by checking NetworkPolicies into [a] version control system. As the policy refers to workloads using metadata, rather than specific instance data such as IP address, it is possible to apply a set of policies and continuously achieve the same security state in the network. Policy as code also allows the network infrastructure to revert to a known security state by simply deploying the same set of policy artifacts,” according to Tigera’s five best practices for Kubernetes network security and compliance. [32] Weave, Linkerd and Project Calico are tools that allow teams to set policies that automatically enable communication between authorized services on an as-needed basis.

NetDevOps is all very new, with companies starting to try it out. Service



	DevOps	NetOps
Compliance Checks	24%	29%
Configuration Management	66%	48%
Incident Response	15%	9%
Pre/Post Change Test	28%	22%
Service Assurance	12%	13%
Service/Application Availability	32%	24%
Troubleshooting	9%	6%
Upgrades	43%	26%
None	18%	27%

FIG 3.1: *Adoption of automation technologies among NetOps teams lags behind DevOps but the gap is closing fast, according to a 2018 study by F5 Networks and Red Hat.*

providers are sparking some of these NetDevOps transformations, but Cisco's Whaley has also seen it coming out of enterprise corporate IT where there's a greater need to automate and manage more network devices. Adoption of automation technologies among NetOps teams lags behind DevOps, but the gap is closing fast, according to a 2018 study by F5 Networks and Red Hat. [33] Despite their desire to automate things, expect NetOps-oriented staff to still raise objections to some automation efforts: Only 31 percent of self-identified NetOps pros believe greater than three-quarters of the development pipeline should be completely made available to developers via automation, which is significantly less than the 45 percent of DevOps pros that want that self-service capability. [34]

How Close Is NetDevOps to Mainstream?

Among the companies experimenting with NetDevOps, the network teams are focusing on establishing a culture and environment where building, testing and releasing network changes can happen more rapidly, frequently and reliably. Devs and Ops are working together to release rapidly and create more stable builds. However, there's usually a missing link of communication between the network and DevOps teams and their tooling.

Since this is an emerging process transformation, tooling isn't as abundant as for DevOps itself, but Whaley did share some of the tools that companies leverage for NetDevOps. A lot of it is surrounding model-driven network management builds with the YANG data modeling language, following NETCONF and RESTCONF protocols. Building NetDevOps on a REST API backbone allows for greater interoperability with what development and operations are doing. Some DevOps tooling being applied to networking includes:

- **Ansible:** Provides automation for IT infrastructure, application deployment, configuration management and continuous delivery.
- **Chef:** Provides open source and enterprise Infrastructure as a Service.
- **Cisco Controllers:** These come with higher-level abstractions for network configuration, set-up and automation.
- **Puppet:** This is an open source software configuration management tool.

Whaley agreed that there are still tooling gaps, like the testing piece.

"If you're going to test a network change, you either have to have a test network sitting around — most people don't have that spare network — or a simulated network," she said. Python automation test systems (PyATS) and the DevNet Sandbox for reserved hosting and testing are good places to start.

The security benefit of DevOps carries over as a strong motivator for network adoption of DevOps practices. By tracking changes as code, network operators have more visibility into the changes that have been made and can further automate security.

Traditionally, network changes require planning and preparing for them well in advance. The benefit of NetDevOps is that networking decisions can be made with the most recent code and with observability, agility and performance in mind, improving uptime and service levels.

The network also has some interesting data for application developers, such as location collected from WiFi, heatmaps and usage information. Business analytics will soon blend with NetDevOps to unlock a lot of data that will improve a whole range of applications, from smart cities to customer experience. This phenomenon is very likely to see a boost in adoption alongside the growing popularity of the enterprise Internet of Things over the next few years.

Who Will Be the First to Accept NetDevOps in the Modern Org?

Whaley warned that just like DevOps, NetDevOps isn't for everyone.

"It's similar [to] DevOps [in] that it is culture plus tech change — not necessarily a fit for everyone. But you can start with automating common tasks and grow the practice from there," she said. "If a strong DevOps practice already exists, it's natural to extend those ways of working" to networking as well.

She pointed out that it brings a lot of possibilities of teams getting closer together. So far, Whaley's team has found that — like many things in the enterprise space — it's the culture that has to change first. DevOps had to deal with literal walls often between Devs and Ops, often on the same floor. Just to start, the network team may even be at a different location.

"Network engineers and developers typically speak different languages, and they approach problems with different tool sets," she said.

Part of the DevNet classroom offering is teaching network developers to code, offering classes like "Coding 101" and "Learn about REST APIs for Network Engineers," with the goal of brokering understanding about the common tools, languages and experiences. These exercises see network developers learning things like, as Whaley described: "Let's get a list of network devices and show me their IP addresses, how it affects them and how to write a Python script." Conversely, Cisco's DevNet also offers Networking 101 courses for developers.

Another piece of NetDevOps that's still evolving is who manages it all. So far, Whaley sees some leadership coming from within the networking organizations looking to link up with their DevOps counterparts because they share in the problems of scale and speed, recognizing automation is the only way to get there. At other organizations, DevOps is leading the way because the patterns, practices and tools have reached a level of sophistication that the natural next step is to look for ways to improve how the network is responding. No matter which side of the aisle is pushing for the NetDevOps transformation — networking or DevOps — Whaley says that the demand for speed, agility and scale is coming from the chief information officer (CIO) level, and either side is focusing on how to leverage the APIs on the other side.

DevSecOps

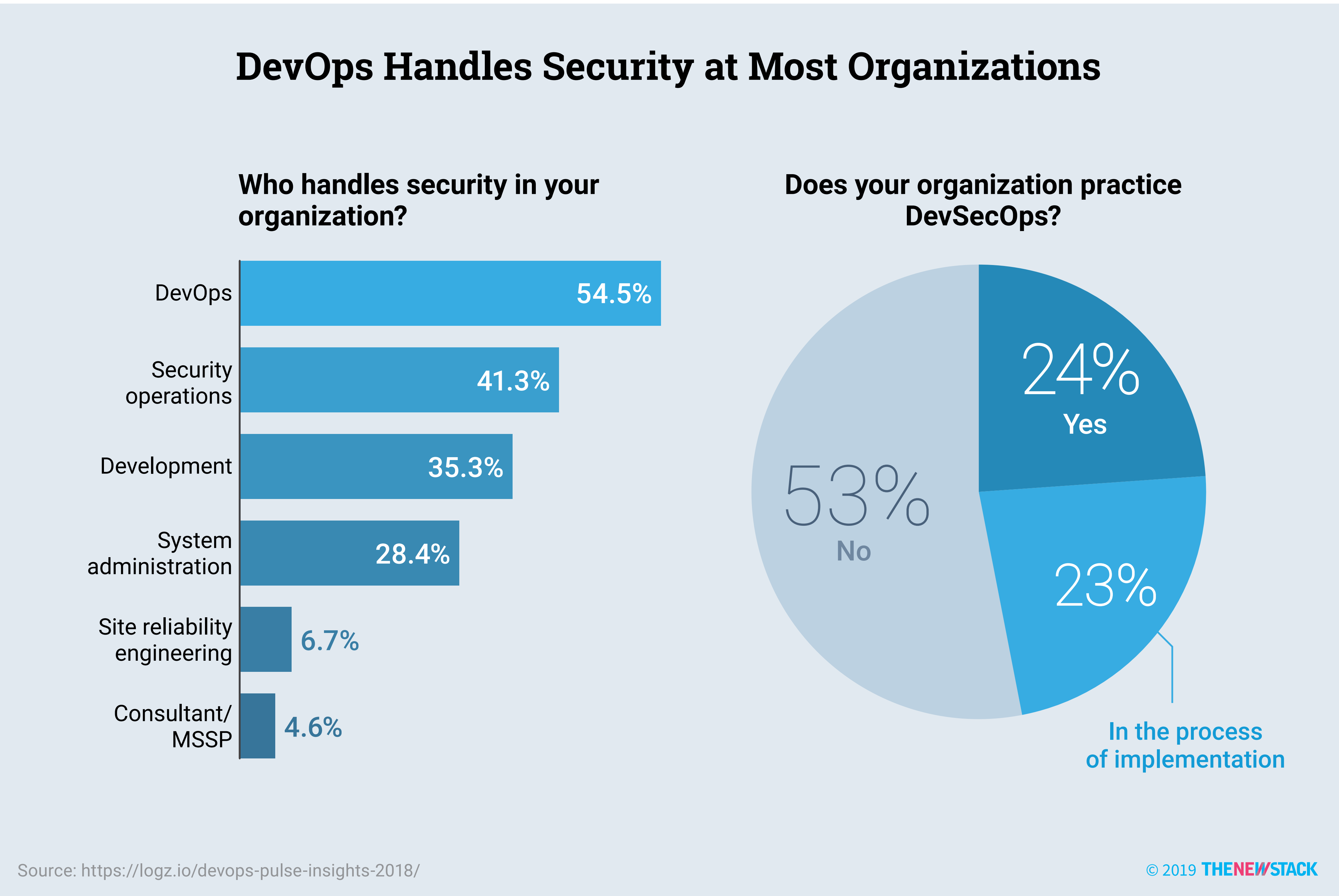
DevSecOps fits security into the DevOps process. Teams of DevOps engineers and security analysts share priorities, processes, tools and, most importantly, accountability, giving organizations a centralized view of vulnerabilities and remediation actions, while also automating and accelerating corrective actions. [35]

Organizational changes that put more responsibility in developers' hands must be combined with emerging best practices for securing microservices-based applications, writes Twain Taylor, a technology analyst and guest writer for Twistlock.[36] Companies are thinking more deeply about developing separate pipelines for various workload types. Other emerging best practices include:

- Programming languages for maintaining compliance.
- Container image scanning.
- Policy-based network security.
- Canary testing technologies.
- Threat detection at runtime.
- Log analysis.

The integration of DevOps and security has already started to take hold in most organizations that have already adopted DevOps practices. A 2018 DevOps survey of over 1,000 IT pros by Logz.io found that DevOps handles security at

FIG 3.2: *The integration of DevOps and security has already started to take hold in most organizations that have already adopted DevOps practices, according to a 2018 DevOps survey of over 1,000 IT pros by Logz.io.*



55 percent of organizations. [37] The finding is unsurprising because 1) efforts to shift security left have often been implemented by DevOps teams responsible for interacting with the entire organization, and 2) 32 percent of the respondents were DevOps engineers. Whether or not this means these organizations actually do DevSecOps, 47 percent said they are at least beginning to implement the practice.

A different survey by Freeform Dynamics, The Register and Checkmarx provides more perspective on how different teams have to work together to make DevSecOps work. [38] When asked about software security challenges, 62 percent of respondents strongly agreed that developers, testers, security specialists and Ops staff need to work together. This desire has yet to match reality. More than half (56 percent) believe that integration of security into the entire DevOps process is either poorly done or non-existent.

DevSecOps is kind of a silly term, because of course companies should be factoring security into their development, particularly when the purpose of adopting DevOps is to help enterprises release applications faster. Just like with DevOps, DevSecOps can be viewed as either a culture or set of tools. Culture is inoculated via training developers. On the software side, DevSecOps is sometimes defined by the use of automated security testing and code dependency testing. [39] With such an obvious need and poor definition, DevSecOps is not a term that security professionals necessarily appreciate.

The term DevSecOps "has always struck me like the last kid getting on the bus and there's no seat available. We are treating security as an afterthought. Security has never been an afterthought with any customer I've dealt with — in financial services or now at Amazon Web Services. I feel like the name doesn't reflect the importance," Margo Cronin, senior solutions architect at Amazon Web Services, said in her talk at the European DevOps Enterprise Summit.

In fact, with the new European General Data Protection Regulation (GDPR), Cronin says privacy by design and privacy by default are built right in. "It nearly mandates you should be doing DevSecOps."

What is DevSecOps to Cronin? Security automation as a priority, early and often. And coming from Amazon, she has an abundance of security automation experience. After all, AWS released almost 1,500 key features and new services in 2017. Amazon.com performs a production deployment once every 11.6 seconds and an average of 10,000 hosts simultaneously. How do AWS and Amazon.com do DevOps but still retain core security practices?

Call it DevSecOps or Rugged IT, like agile and Kanban enthusiasts do, but how AWS refers to it is pretty accurate: security automation at cloud scale. DevSecOps replaces disconnected, reactive security efforts with a unified, proactive CI/CD-based security solution for both cloud and on-premises systems. A more cohesive team from a diversity of backgrounds works toward a common goal: frequent, fast, zero-downtime, secure deployments. This goal empowers both operations and security to analyze security events and data with an eye toward reducing response times, optimizing security controls and checking and correcting vulnerabilities at every stage of development and deployment. Blurring the lines between the operations and security teams brings greater visibility into any development or deployment changes warranted, along with the potential impacts of those changes. [40]

Security Automation Evades Human Error

Cronin says that Amazon has embraced DevSecOps "so passionately" because they needed DevOps to move fast and, accounting for their considerable partners, that meant automating security first in order to avoid these three trappings of humanity:

1. **People make mistakes.** Imagine engineers working late at night, with a "severity one" in production, Cronin said. "You have an IT, C-level

stakeholder telling you to get the service back online. You are on hour five of a severity one call. You are on a Slack channel with 40 people, 38 of whom are not really contributing. You are on cup of coffee number seven. You make a change in production to resolve this issue." She said that under these often common circumstances, you are more prone to make an error than in a business-as-usual scenario, and maybe you forget to document the change and the next release overwrites the fix. "Humans make mistakes, and when you're under pressure you're more likely to make mistakes."

2. **People bend the rules.** Then she shared a well-meaning common use case: People bend the rules in an effort to be helpful and to collaborate, like when you have scheduled a big release — and release party — and everyone's ready to celebrate and it's almost there, so you say: "We're just going to get it out. We'll do the release and fix that tomorrow. People will ask you to bend the rules from a place of goodness, but these create gaps in your product landscape."

3. **People act with malice.** "While attacks like DDoS are automated, there is invariably a human behind the scenes instigating that attack."

While a mistake is written into code or an automated process, these mistakes are frequently repeated, which creates patterns that are easily diagnosed.

Machines don't make mistakes, bend the rules or act with malice, which is why Cronin argues that automating security tasks must be your biggest priority for successful DevOps. Yet only 34 percent of information security professionals' organizations have automated security testing in their software release life cycle, according to Cybersecurity Insiders' 2018 Application Security Report. [41] Although most organizations have some DevSecOps processes in place, in reality, automated security testing is not deployed in a majority of CI/CD pipelines.

“The reality is that manual security doesn’t work in the cloud native age,” writes John Morello, CTO of Twistlock on The New Stack. [42] “Environments move too fast, and configurations change too quickly for your engineers to be able to interpret security threats manually and react in a timely fashion. You, therefore, need tools that can make informed data-based decisions about threats for you, then take action to stop them before they cause damage.”

Several approaches have evolved to reduce risk across the clusters, pods and nodes running on Kubernetes, or existing in some similar serverless approach, without hiring more staff or burning out the existing team. The trend is for greater observability across the system to allow for automated responses — such as scaling, rollbacks and load balancing — as well as more informed decision-making and feedback loops. There is no single, standard metric that developers can rely upon to tell them whether their code is working or not. Observability gives developers the ability to collect data from their application and trace problems to the root cause, in order to debug code and relieve issues affecting the application in production. [43] This is the shift-left approach which gives developers more responsibility for securing their code. In addition, automated security tests, such as container image scanning and vulnerability scanning while code is still in development and compliance tests, help enforce security policies and practices at scale. And configuring network security policies, monitoring for breaches and automating the response, helps ensure network security at scale.

Four Steps to Enable Security Automation at Scale

Cronin outlines four steps that companies can take toward security automation at scale which add value to the DevSecOps process.

1. Establish Your Level of Trust

She calls this her bar or spectrum of trust that you have in your cloud or on-premises provider. Large, distributed organizations with defined security processes and governance typically have low or zero trust. These low-trust companies want their own managed keys and their own hardware security modules. They are at one end of this spectrum. On the other end of this spectrum lies startups and e-commerce platforms that use all the services of their cloud service provider.

"It doesn't matter where you are in the spectrum, you can still get the value of the cloud, but the point of trust that you have is congruent to the amount of automation you need to implement," she said.

Cronin gave the example of transferring trust with more automation. This trust level of zero could be a company deploying native Kubernetes, managing the master nodes (scaling and distributed consensus), the worker nodes, and all the security. On the other end of the spectrum at the right, she offered an example where the customer with higher trust uses Amazon Elastic Container Service for Kubernetes (Amazon EKS).

"The complexity of standing up your own Kubernetes control plane is simplified. Instead of running the Kubernetes control plane in your account, you connect to a managed Kubernetes endpoint in the AWS cloud. This endpoint abstracts the complexity of the Kubernetes control plane — your worker nodes can check into a cluster, and you can interact with your Kubernetes cluster through the tooling you already know and love. By default, [in this scenario with AWS], Kubernetes role-based access control [RBAC] is on, volumes are automatically encrypted, and AWS does the certificate management."

Here Cronin is referencing a common concern around Kubernetes, which has RBAC turned off by default. The Kubernetes RBAC module controls access to Kubernetes API resources based on the assigned role of the user. [44] Last spring, someone was able to hack Tesla's control plane because nobody had enabled RBAC. By putting more trust into Amazon's EKS, she says RBAC for Kubernetes is automatically turned on, has native integration with AWS, and managed master nodes.

She says that "No matter where you are on the trust scale, plan to integrate security automation, but remember that creating this automation will also take the DevOps team time."

This involves mapping the tooling based on where you are on the trust bar. The lower the level of trust, the higher the level of security automation the DevOps team needs to implement. The higher the level of trust, the more the cloud provider can automatically manage and automate for you. This impacts how quickly you are going to release your minimum viable product. Also the less trust, the more you have to plan your security ahead, like ensuring your RBAC is on.

2. Security by Design

Cronin contends that in DevSecOps, every team member feels the responsibility of a security owner — it's no longer a team in another building, just a stakeholder to your project. Just like DevOps tears down the silos between developers and operations, the same must happen for security.

With DevSecOps, sprints can be based on security needs, breaking epics down to functional security stories. This security process used to take a couple of months with on-premises hosting via complicated waterfall epics. Now she says that, with any cloud service provider, you can spin up the web application firewall in sections, which usually are:

- Identity and access management.
- Logging and monitoring.
- Incident response.
- Infrastructure security.
- Configuration and vulnerability analysis.

- Securing a CI/CD pipeline.
- Data protection.

You then use the same dynamic CI/CD pipeline to roll out the security features that you would with the rest of your application life cycle.

Security by design also means having security-related acceptance criteria. Continuing with the example of GDPR's requirements, which prioritizes having users own their own data, when a user logs into the system, you have demonstrated how that data can be deleted and demonstrate how you can actually port it over. Security automation has to test if that is possible and, ideally, document it all.

3. Secure the Pipeline

Cronin says the example above and much of this security automation advice can be applied on premises as well. It should address both the security of — including access roles and hardening of build servers and nodes — and the security in — including artifact validation and static code analysis — your CI/CD pipeline.

Git Secrets is a tool that collects a set of DevSecOps open source resources, that can be leveraged along with the cloud for such important CI/CD steps as:

- Authentication and validation in your repositories.
- Logging across the entire environment.
- Sending build reports to developers and stopping everything if a build fails.
- Sending build reports to security and stopping everything if the audit or validation fails.

Infrastructure as Code is also an important concept in security automation and DevSecOps. Automation of the infrastructure deployment process increases the importance of security and compliance testing, as with the push of a button, you are able to make highly impactful changes to your cloud environment, writes Roy Feintuch, CTO of Dome9 Security.[45] This is the flip side of agility. In the public cloud, where simple configuration changes can leave sensitive data and private servers exposed to the world, the security implications of automation are profound.

"Now, where your entire infrastructure is defined in a JSON file, the DevOps folks have their hands on the keyboards. This puts the security folks in a weird new place — instead of being a chokepoint, they now try to keep up with the changes. Sometimes retroactively," Feintuch said.

The InSpec tool from Chef, for example, enables compliance, security and DevOps teams to more clearly define security and compliance tasks by writing specific rules to automate them. Security teams can set the policies that DevOps teams deploy against. Users can write custom compliance policies for AWS and Microsoft Azure or use pre-defined policies for regulations such as PCI, HIPAA and the Department of Defense. And they can validate cloud configurations, covering virtual machines, security groups, block storage, networking, identity and access management and log management, against the policies.[46]

"It provides a simpler level of abstraction so you can find out things like what Docker containers do you have running, what packages do you have running," said Julian Dunn, former director of product marketing at Chef. "We want to make sure our database doesn't have the default database installed or the default user installed and we have strong passwords and we can see what systems are allowed to connect to this database server, [and] make sure they're using encryption."

Salt is another configuration management tool that can be used to orchestrate the initial deployment, and then ongoing security and management, of the entire IT environment. At the SaltConf18 user conference, the company behind the open source Salt, and its enterprise component, SaltStack Enterprise, demonstrated how the software can be used to apply intelligent IT automation to many routine actions of setting up IT infrastructure, such as how to provision large numbers of VMs — with multiple configurations — on a cloud service, how to run bare-metal deployments of full server stacks with no intervention, and even how to manage virtual LANs (VLANs) and other software-defined network setups. Salt will also track the success and failure of each step along the way. 47

"If you've got a limit on, say, how many VMs you can spin up on AWS, Salt can respond to that failure, and trigger an alternate orchestration. It could halt the step, or roll it back to a previous state," said Gary Richmond, senior technical product manager at SaltStack.

From a maturity perspective, such levels of automation and validation represent the pinnacle of DevOps adoption. Feintuch sees organizations going through several stages on this path:

a. **Manual security assessment:** This approach involves manually inspecting the live infrastructure after deployments, and reviewing the architecture/templates before it is deployed to a live environment. While DevOps teams are experts in creating Cloud Formation Templates (CFT) and Terraform templates, most traditional security teams are not familiar with these technologies. Checking the security posture of these CFTs is a slow, manual process that requires extensive back and forth between Ops and security, putting a brake on CI/CD pipelines.

b. **Continuous monitoring of live environments:** In this approach, the live production environment is being continuously monitored for security and compliance violations. This is commonly accomplished using dedicated cloud security services that provide continuous security and compliance testing. While being reactive in nature, this type of monitoring does provide coverage for both automated changes as well as manual/rogue ones.

c. **Deploying and testing in sandbox environments:** The repeatability of IaC is an enabler of this new approach, where the template is first deployed into a temporary environment. Automated, and sometimes manual, tests verify that there were no security and compliance regressions before pushing the changes to a production environment. All of that is orchestrated by an automated CI/CD pipeline which then terminates the temporary environment. While this approach is effective in ensuring that code deployed to production environments meets security and compliance requirements, it can be expensive in terms of time, resources and solution complexity.

d. **Static (infrastructure) code analysis prior to deployment:** In this approach, teams treat the templates just like any other software code and perform security and compliance unit-tests after code commits as part of the standard CI process. This is a very promising new approach that can bring security and compliance as close as possible to the source, cutting drastically the time to detection and the operational complexity of approaches like a and c, above.

This process of security automation combined with the steps above becomes highly immutable and reduces your blast radius — the extent of damage that a compromised container can do to other containers on the same node, or how much damage a compromised node can do to the rest of the cluster. [48]

4. Automate Responses

For security automation to work, you need to know what you are doing based on your log files. It all comes down to four questions surrounding your logs:

1. When are you collecting logs?
2. Why are you collecting logs?
3. Where are you collecting logs?
4. What are you doing based on your logs?

Take the example of someone switching off an AWS service. This action can send an automated event to your security team for them to look into the environment. It allows you to make the decision if it was shut off by someone whose privileges are too high, or if it's actually an event that needs looking into and maybe servers need to be ring-fenced. Cronin pointed out how powerful logging has become and how logging in the cloud prevents more incidents.

The abstraction of the security layer that comes with containers and Kubernetes has many benefits to the overall security posture of the organizations deploying applications on a cloud native architecture. It also poses new risks and possible vulnerabilities. With their individual APIs, microservices can be reconfigured and updated separately, without interrupting an application that might rely on many microservices to run. However, microservices also come with many separate APIs and ports per application, thus exponentially increasing the attack surface by presenting numerous doors for intruders to try to access within an application. While their isolated and standalone structure within applications makes them easier to defend, microservices bring unique security challenges. The attack surface widens further for Kubernetes users because of the orchestrator's comprehensive reach in the container runtime environment. [49]

Organizations have begun to take take a "shift left" approach that rests security practices deeper into the development process, so that security teams are more involved in engineering and vice versa. The shift left approach, which gives developers more responsibility for application security, itself has two sides: Most developers do not fully understand how applications are connected across a network mesh. What they are looking for often are open end-points — APIs to interconnect applications. That in itself creates a security gap and an opportunity to isolate issues. At the same time, it allows teams to detect anomalies faster through automated container image scanning — going further left by baking security into the code itself. In the end, when looking to automate security, it seems best to follow Cronin's final words on the importance of the Sec in DevSecOps:

"If security is your most important job, you should look at automating those tasks and stories first, before anything else."

Cloud Native DevOps with Jenkins X

Automation is an essential part of DevOps and Jenkins is in a prime position to help streamline CI/CD processes. But Jenkins itself, while remaining essential for many organizations' development processes, is notoriously difficult to master and manage, especially as projects are scaled. As a metaphor for what is at stake, CloudBees has coined the term "Jenkinsteins" to describe the end result of the struggles organizations face when attempting to get a handle on Jenkins' unwieldiness.

Jenkins X was designed to make such metaphors a thing of the past. It creates a standard way to integrate CI and CD with the de facto cloud native platform, Kubernetes. Organizations can use their own source code while Jenkins X automates the process as it is ported to cloud native environments.

"Installing and managing Jenkins and configuring Jenkins and figuring out how to create your pipelines and which plugins to use and how to even do CI and CD, take a reasonable amount of effort," James Strachan, senior architect at CloudBees and the project lead on Jenkins X, said. "What we wanted to do with Jenkins X was go the other way around and basically make CI and CD almost an appliance you just consume."

In this interview, some of CloudBees' key team members discussed how Jenkins X, CI/CD and DevOps all elegantly fit together. On hand to share their views along with Strachan were Michael Neale, co-founder and engineering manager of CloudBees and James Rawlings, principal software engineer at CloudBees.

"In many ways, Jenkins X is becoming an orchestrator of Jenkins," Strachan said. "We wanted to standardize how to do CI and CD and Kubernetes, with Jenkins X, so that you literally bring your source code along and Jenkins X automates all the rest of it."

Listen on SoundCloud

Michael Neale *is co-founder and development manager at CloudBees. Michael is an open source polyglot developer. For his sins, he spent time at JBoss, working on the Drools project, and then Red Hat. He lives and works in the Blue Mountains, just outside Sydney, Australia.*

James Rawlings *is a co-creator of the open source project Jenkins X and works as a principal software engineer at CloudBees where he aims to help developers and teams move to the cloud. He is passionate about automation and continuous improvement, always looking for new ways to help developer productivity and developer experience.*

James Strachan *is the chief architect of Jenkins X and of its commercial version, Kube CD. He's also the creator of the Groovy programming language and Apache Camel. James is actively involved with the Jenkins community. Prior to CloudBees, he was with Red Hat and FuseSource.*

Bibliography

1. **"7 Promises, and Potential Pitfalls, in Adopting a Cloud Native Approach to DevOps"**
by Scott M. Fulton III, The New Stack, September 10, 2018.

Fulton explores seven possible definitions of the phrase "Cloud Native DevOps," based on interviews, event presentations and research.

2. **"Seven Remarkable Takeaways From Massive Kubernetes Conference"**
by Jason Bloomberg, Contributor, Forbes, December 13, 2018.

A summary of takeaways from the KubeCon + CloudNativeCon conference in Seattle.

3. **"What is DevOps: A 2018 Perspective"**
by Kris Buytaert, Co-founder and Chief Technology Officer at Inuits, Exoscale Blog, October 22, 2018.

The organizer of DevOpsDays reflects on what 10 years as an engineer has taught him about DevOps.

4. **"Guide to Serverless Technologies"**
by The New Stack, 2018.
Leading edge companies have already embraced serverless and have dramatically reduced operational overhead and streamlined the DevOps cycle. Still, there are many challenges to serverless adoption, such as operational control, complexity and monitoring. This guide covers the pros and cons of serverless adoption and offers practical advice for adoption.

5. **"Serverless For Teams"**
The New Stack Makers podcast with Nate Taggart, CEO of Stackery, October 2, 2018.

A pay-as-you-go pricing model and increased velocity are two well-known effects that serverless technologies have on application

delivery. Just as important, but often overlooked, is how serverless affects developer workflows and team dynamics.

6. **"Serverless and DevOps"**
by Chris Munns, Principal Developer Advocate for Serverless at Amazon Web Services, at ServerlessDays Portland, September 17, 2018.

In this talk, Munns questioned the need for DevOps, as serverless adoption rises and developers take on the responsibility for configuration.

7. See #4, above.

8. **"Delivering Cloud Native Infrastructure as Code"**
by Pulumi, 2018.
In this white paper, Pulumi makes the case for a consistent programming model for the cloud.

9. **"Why Going Serverless Doesn't Mean 'No Ops'"**
The New Stack Makers podcast with Kapil Thangavelu, Senior Director of Engineering at Capital One, January 20, 2017.

Thangavelu discusses the main benefits of embracing a serverless infrastructure. The overall simplicity toward operations is transforming because teams can focus on core business logic.

SPONSOR RESOURCE

- **Building Cloud Native Apps Painlessly — The Prescriptive Guide to Kubernetes and Jenkins X**
by CloudBees, 2019
In this paper, you'll read about how the future of modern application development can benefit from the powerful combination of Jenkins X and Kubernetes, providing developers a seamless way to automate their continuous integration (CI) and continuous delivery (CD) process.

10. **"5 Workflow Automation Use Cases You Might Not Have Considered"**
by Bernd Rücker, Co-founder of Camunda, The New Stack, April 9, 2018.

Workflow automation is so much more than human task management. In this contributed post, Rücker elaborates on use cases for workflow automation.

11. **"DevOps Is the Secret Ingredient to Make Microservices Cook"**
by Alex Handy, The New Stack, April 9, 2018.

Microservices adoption leads to an explosion of projects. Companies are turning to automation to deal with the complexity, but also creating a closer collaboration between development and operations teams.

12. **"2018 State of DevOps Report"**
by Alanna Brown, Andi Mann, Michael Stahnke and Nigel Kersten, and Puppet Labs and Splunk, 2018.

This annual report analyzed responses from over 3,000 technical professionals from around the world to reveal the five stages of DevOps evolution.

13. **"The New DevOps: Site Reliability Engineering Comes of Age"**
by Kieran Taylor, Senior Director of Product Marketing at CA Technologies, The New Stack, July 5, 2018.

Traditionally, IT operations teams have typically taken an inside-out view of the world. Site reliability engineering (SRE) stands that approach on its head by viewing reliability through an outside-in lens, writes Taylor in this contributed article by the author of "DevOps for Digital Leaders."

14. **"Guide to Cloud Native Microservices"**
by The New Stack, 2018.
This ebook provides a high-level overview of what organizations should consider

as they create, deploy and manage microservices for cloud native applications, and lays the foundation for understanding serverless development and operations.

15. **"DevOps and People: Where Automation Begins!"**
by Almudena Rodriguez Pardo, Business Agility Consultant at Improvement21, at Agile Tour London, October 2018.

In this talk, a noted agility consultant for telecommunications clients, such as Ericsson, discusses the human side of DevOps automation.

16. **"Connected Futures: Transforming IT Operations"**
by Cisco, 2018.
This global research study surveyed more than 1,500 senior IT leaders and introduces the IT Operations Readiness Index to assess progress toward data-driven, automated operations.

17. **"New Cisco Study Predicts Dramatic Change in IT Operations as CIOs Embrace Analytics and Automation"**
by Cisco, a news release on November 1, 2018.
News release announcing the Cisco Connected Futures research report.

18. **"Spotify Engineering Culture"**
by Henrik Kniberg, Former Agile/Lean Coach at Spotify Labs, March 27, 2014.

This short animated video explains Spotify's journey to DevOps transformation.

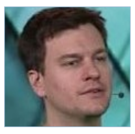
SPONSOR RESOURCE

- **Delivering Cloud Native Infrastructure as Code**
by Pulumi, 2018
Find out how to deliver all cloud native infrastructure as code with a single consistent programming model in this white paper from Pulumi.

19. **"Continuous Delivery with Spinnaker" in "CI/CD with Kubernetes"**
by Craig Martin, Former Senior Vice President of Engineering at Kenzan, The New Stack, 2018.

Implementing Kubernetes on its own doesn't magically achieve CD for your organization. The features Kubernetes provides, however, in modularity, available tooling and immutable infrastructures certainly make CD much easier to put in place.

20. **"The Ever-Changing Roles of the Developer: How to Adapt and Thrive"**
by David Hayes, former Director of Product Management at PagerDuty, The New Stack, June 20, 2017.


Developers are now spending less time than ever writing code and instead find themselves focused (and evaluated) on enhancing and maintaining scalability, improving customer experience, boosting service efficiencies and lowering costs, Hayes writes in this contributed article.

21. **"Cloud Native Application Patterns" in "CI/CD with Kubernetes"**
by Janakiram MSV, Principal Analyst at Janakiram & Associates and an Adjunct Faculty Member at the International Institute of Information Technology, The New Stack, 2018.

As long as developers follow best practices of designing and developing software as a set of microservices that comprise cloud native applications, DevOps teams will be able to package and deploy them in Kubernetes. This chapter is intended to help guide DevOps teams deploying cloud native applications in Kubernetes.

22. **"Why Microservices Require a Storage Rethink"**
by B. Cameron Gain, The New Stack, August 22, 2018.

In cloud native applications, data is generated from numerous and different microservices, as opposed to data generated from a single monolithic computing structure. In this way, storage and database access must take into account every microservice, which can

often number in the hundreds or possibly thousands of individual threads.

23. **"Tigera Aims to Ease Connectivity Pain with Kubernetes"**
by Susan Hall, The New Stack, August 11, 2017.

Tigera's core technology, Project Calico, has a large base of users that are moving to production Kubernetes. Large deployments with billions of API calls per second are bringing a new class of connectivity challenges.

24. **"Serverless 101: How To Get Serverless Started In The Enterprise"**
by Michelle Gienow, The New Stack, June 4, 2018.

The first in a multipart series exploring the basics of serverless.

25. **"IT 2018: Shifts in Budgets, Initiatives and Executive-level Influence Signify Lasting Changes for IT Departments"**
by TekSystems, December 12, 2017.
This annual survey polled more than 1,000 IT leaders (i.e., chief information officers, IT vice presidents, IT directors, IT hiring managers) on a range of key issues, including IT staffing and budgets, skill needs and expected organizational challenges in 2018.

SPONSOR RESOURCE

- **Continuous Delivery Summit**
by Continuous Delivery Foundation, 2019
The Continuous Delivery Foundation will be hosting a Continuous Delivery Summit (CDS) event on May 20 at KubeCon + CloudNativeCon Europe 2019 in Barcelona, Spain. The Cloud Native Computing Foundation's flagship conference gathers adopters and technologists from leading open source and cloud native communities including Kubernetes, Prometheus, Helm and many others. Register to attend, today!

26. **"This Week in Numbers: DevOps Favor Microservices"**

by Lawrence Hecht, The New Stack, January 20, 2018.

Not every application needs to be decomposed into microservices, but the conventional wisdom is that the practice is becoming widespread. DZone's survey of 605 software professionals sheds light on the matter.

27. **"How Container-Based Architectures Require Different Networking"**

by Jeroen van Rotterdam, Executive Vice President of Engineering at Citrix, The New Stack, September 24, 2018.

The distributed nature of microservices architecture makes administering networking and security policies a lot harder than it is in a monolith architecture. Containers appear, disappear and are moved around to different compute nodes far too frequently to be assigned static IP addresses, much less be protected by firewalls and IP tables at the network's perimeter.

28. **"Q&A: Rain Capital's Chenxi Wang on 'DevSecOps'"**

by Joab Jackson, The New Stack, December 3, 2018.

Dr. Chenxi Wang, founder and general partner of Rain Capital, an early stage Cybersecurity-focused venture fund, and co-founder of the Jane Bond Project, a cybersecurity consultancy, answers questions about integrating security into the software life cycle.

29. See #21, above.

30. **"NetSecOps: Everything Network Managers Must Know About Collaborating with Security"**

by Shamus McGillicuddy, Research Director for the Network Management Practice at Enterprise Management Associates (EMA), September 6, 2018.

Slides based on a webinar featuring Shamus McGillicuddy and Jon Kies, manager of network management product marketing at Micro Focus.

31. **"Service Mesh and the Promise of Istio"**

by Chenxi Wang, Ph.D., founder and general partner of Rain Capital, The New Stack, July 10, 2018.

In this contributed article, Dr. Wang explains service mesh and why it is such an appealing concept for microservices management.

32. **"5 Best Practices for Kubernetes Network Security and Compliance"**

by Tigera, 2018.

In this short white paper, Tigera explains how network and security concerns differ in microservices-based applications on Kubernetes.

33. **"NetOps Meets DevOps: The State of Network Automation"**

by F5 Networks and Red Hat, 2018.

This co-sponsored survey of 400 DevOps and NetOps professionals discusses the state of network automation in IT.

34. **"This Week in Numbers: Can There Be Too Much DevOps Automation?"**

by Lawrence Hecht, The New Stack, September 2, 2017.

The New Stack's research director covers top takeaways from the F5 Networks and Red Hat survey on NetOps and DevOps.

35. **"DevOps Patterns" in "CI/CD with Kubernetes"**

by Rob Scott, Senior Site Reliability Engineer at ReactiveOps, The New Stack, 2018.

In this chapter, Scott covers how traditional DevOps looked before containers, how containers brought developers and operators together into the field of DevOps, the role a container orchestration tool like Kubernetes plays in the container ecosystem and how Kubernetes resulted in a revolutionary transformation of the entire DevOps ecosystem.

36. "Defining The Perimeter in a Microservices World"

by Twain Taylor, Guest Writer for Twistlock, The New Stack, February 12, 2018.

Organizational changes that put more responsibility in developers' hands must be combined with emerging best practices for securing microservices-based applications, writes Taylor.

37. "The 2018 DevOps Pulse"

by Logz.io, 2018.

Logz.io surveyed more than 1,000 IT professionals globally for this annual analysis of the DevOps industry which highlights key trends, points of interest and challenges that DevOps and IT operations professionals experience on a daily basis.

38. "Managing Software Exposure"

by Freeform Dynamics, with The Register and Checkmarx, 2018.

Freeform Dynamics' survey of 183 respondents from North America and the UK provides more perspective on how different teams work together to make DevSecOps work.

39. "Add It Up: DevOps Security Needs More Tooling"

by Lawrence Hecht, The New Stack, August 9, 2018.

The New Stack's research director analyzes a DevOps survey of over 1,000 IT pros by Logz.io which found that DevOps handles security at 55 percent of organizations.

40. See #35, above.

41. "2018 Application Security Report"

by Cybersecurity Insiders, 2018.

This report is the result of a comprehensive survey of 437 cybersecurity professionals designed to reveal the latest application security trends, how organizations are protecting applications, and what tools and best practices IT *cybersecurity teams are prioritizing to find, fix and prevent vulnerabilities in next-generation applications.*

42. **"CISOs: Five Essential Features in a Cloud Native Security Platform"**
by John Morello, Chief Technology Officer at Twistlock, The New Stack, December 28, 2018.

Morello looks at the main features that a cloud native security platform should provide for your team.

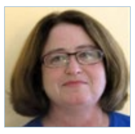
43. See #4, above.

44. **"Kubernetes Security Patterns" in "Kubernetes Deployment and Security Patterns"**
by Chenxi Wang, Founder and General Partner of Rain Capital, The New Stack, February 2018.

In this chapter, Dr. Wang talks about threat models and various security considerations for a Kubernetes deployment and discusses best practices that organizations can follow.

45. **"New Security Challenges with Infrastructure-as-Code and Immutable Infrastructure"**
by Roy Feintuch, Co-founder and Chief Technology Officer of Dome9 Security on The New Stack, April 4, 2018.

Infrastructure as Code (IaC) is an important concept in security automation and DevSecOps. Automation of the infrastructure deployment process increases the importance of security and compliance testing, as with the push of a button, you are able to make highly impactful changes to your cloud environment.

46. **"Chef InSpec 2.0 Puts the Security into DevSecOps"**
by Susan Hall, The New Stack, February 20, 2018.

InSpec enables compliance, security and DevOps teams to more clearly define security and compliance tasks by writing specific rules to

automate them.

47. **"SaltStack for Event-Based IT Orchestration Across the Hybrid Cloud"** by Joab Jackson, The New Stack, November 28, 2018.

Many IT operations teams turn to SaltStack for help in configuration management, but this is only a subset of the software's capabilities. Organizations can think about using SaltStack to orchestrate the initial deployment, and then ongoing security and management, of their entire IT environment.

48. See #44, above.

49. See #44, above.

SECTION 2

DEPLOY

Adopting a cloud native DevOps culture means making changes that facilitate speed and communication through increased autonomy, transparency and automation.

Contributors

B. Cameron Gain's obsession with computers began when he hacked a Space Invaders console to play all day for 25 cents at the local video arcade in the early 1980s. He then started writing code for very elementary games on the family Commodore 64 and programming in BASIC on the high school PC. He has since become a long-time and steadfast Linux advocate and loves to write about IT and tech. His byline has appeared in Wired, PC World, CIO, Technology Review, Popular Science, and Automotive News.

Jennifer Riggins is a tech storyteller, content marketer and writer, where digital transformation meets culture, hopefully changing the world for a better place. Currently based in London, she writes around tech ethics, agility, accessibility, diversity and inclusion, testing DevOps, happiness at work, API strategy, the Internet of Things, microservices and containers, developer relations, and more.

Lucian Constantin is a freelance writer for The New Stack. He has been covering cybersecurity and the hacker culture for over a decade, his work appearing in many online technology publications including PCWorld, Computerworld, Network World, The Inquirer and Softpedia.com. Lucian has a bachelor's degree in political science, but his passion for computers and security came at an early age leading to his involvement in various technical support forums and IRC channels. While in college he worked part-time as a systems and network administrator for several small businesses.

Mike Melanson is a freelance writer, editor, and all-around techie wordsmith. Mike has written for publications such as ReadWriteWeb, Venturebeat, and ProgrammableWeb. His first computer was a “portable” suitcase Compaq and he remembers 1200 baud quite clearly.

CHAPTER 04

Culture Change for Cloud Native DevOps

Software delivery cycles are becoming faster thanks to DevOps-backed continuous integration and continuous delivery (CI/CD), as production pipelines are increasingly ported to scale with microservices in cloud native environments. But the organizations that are able to achieve this rapid development and scaling must first embrace a DevOps culture. Microservices and serverless architectures require a new set of tools and processes — and the breaking of old ones. It can be a difficult shift for an organization that is responsible for managing monoliths to make an abrupt change in workflows. Removing barriers between all stakeholders within an organization, including business, operations, developers and other teams, may not be easy, yet it's essential to building and maintaining cloud native applications.

DevOps is a transformation that most enterprise organizations have yet to undergo. Only about half of all organizations had implemented DevOps in 2017, some 10 years after DevOps as a concept and practice emerged, according to a Forrester Research report. One clear indication of this lagging adoption is that 78 percent of over 100 firms surveyed were not modifying their security tools when moving from on premises to cloud deployments, according to a 2018 McKinsey study. Respondents cited siloed security teams that were disconnected from DevOps as the main culprit. With a DevOps culture and processes in place, organizations are in a better position to take advantage of the new processes and tooling around securing microservices applications. [1]

“[Cloud native] is about building an organization that can change more quickly and nimbly by removing the technical risks associated with that change. In the past, our standard approach to avoiding danger was to move slowly and carefully. The cloud native approach is about moving quickly by taking small, reversible and low-risk steps.”

— Anne Currie, chief strategist at Container Solutions. [2]

The question is, why have most organizations not undergone a DevOps transformation and what makes cloud native DevOps any different? In prior works, The New Stack has developed the premise that at-scale application development, deployment and management is built on DevOps practices. The premise being that the tools needed to make scale possible came from the belief that monolithic technologies were built to manage systems of record with storage and networking technologies as the most significant cost factors. Now the cost is moving to the creation and management of services on cloud native architectures that are built to scale. Storage and networking have different contexts. In monolithic systems, the application runs on the network, configured to the machine. In cloud native architectures, storage is persistent and the network is software that is application-centric, meaning the application does not necessarily get configured. The network is software running on orchestration engines such as Docker swarm mode, Kubernetes and Mesos. Underneath are containers and virtual machines (VMs). The containers are portable and the VM plays a different role, albeit increasingly as a supporting mechanism for containers. For example, Amazon Web Services' AWS Fargate is built on containers but underneath is a micro-VM architecture.

What changes in cloud native DevOps is the construct for teams, their workflows and the adoption of cloud native technologies that support the organization's business objectives and developer requirements. Kubernetes is largely viewed as cloud native. But there are any number of other tools that must be considered in relation to who is on the team, their level of experience and the workflows that best suit them. The people who developed the first at-scale architectures made their own tools. Luckily, in today's world, there are now a selection of tools that largely are based on the experiences of these pioneering technologists. These include a new generation of graph databases, continuous delivery tools and a host of services that allow for teams to manage the persistence of their applications, the networking and, increasingly, the communications between services, including events that trigger alerts and notifications. Serverless technologies are emerging to also fill the void, but largely are defined in newer application architectures that use automation practices to manage the functions that historically required a more manual approach.

The fact that DevOps is lacking in many organizations means that those that do make the cultural shift are at a strong competitive advantage — at least for now. This is because it can be assumed that most organizations have either not fully optimized DevOps to support cloud native architectures, or in many cases, have not yet adopted DevOps.

Cloud native architecture is based on discrete and loosely coupled units of code — containerized microservices or serverless functions — built and managed by largely autonomous teams. Application architecture development is a direct result of how individuals and teams interact and communicate about their own, and overlapping, orchestrations. As more developers are added to achieve scale and the application gets more sophisticated, so does the overall complexity of the architecture. Microservices make that scale more manageable by breaking the monolith down into pieces that can be managed separately by teams with responsibility for the full life cycle of that code.

“We’re generally seeing the emergence of smaller, product-focused teams,” James Governor, co-founder of analyst firm RedMonk, said. “And if we’re going to break down the monolithic platforms and have a product focus, we’re also breaking down the monolithic teams so that they can support their products.”

This new mode of development, and the complexity that comes with it, requires new tools, processes and team structures, and it’s precisely why DevOps becomes critical to success. DevOps is a culture of transparency, openness to change, shared responsibility and continuous improvement — a way of working that eliminates barriers between teams and values communication. Such transparency is necessary in order for teams to remain loosely coupled and independent. Openness means a willingness to adapt and change course, a desire to learn new skills as well as learn from mistakes, and a willingness to share those mistakes and the lessons learned so that others do not repeat them. Through open communication and clearly defined and automated processes, change propagates quickly as product teams adapt to that feedback. All of this is done for the benefit of the customer or end user, and thus, the business.

Platforms, like Cloud Foundry and Kubernetes, provide technical solutions that make the developer experience more predictable and scalable, Chisara Nwabara, a service and product specialist at Pivotal writes on The New Stack.[3] But companies must also work on making their customer experience more predictable and scalable by improving communication channels within their own teams. Communication — alongside tooling and automation — is key to DevOps transformation and to fully realizing the benefits of a cloud native architecture.

Business Buy-In, or Else

To enable the cycle of feedback between customers and development teams, a cloud native DevOps culture change must include the business side of an organization. Someone from the IT department will likely initiate the concept of applying DevOps as part of a shift to software development from a monolithic application that runs on premises to microservices running on containers and Kubernetes in the cloud. But without the business teams participating, nothing much will happen, Brian Dawson, a DevOps evangelist at CloudBees, said. “Business has to be involved. The whole idea is you want to go from ideation or concept to customer.”

Chief technology officers (CTOs) and chief information officers (CIOs) have usually bought into DevOps, but it’s one chain of command down that doesn’t interact with the developer (Dev) and operations (Ops) teams every day, but oversees the success of the product, that can be the downfall of a DevOps initiative. These are the roles that need to get on board with DevOps early, helping to form the product statement and drive toward actionable user feedback, Ashley Hathaway, engagement director at Pivotal, said.

Upper middle management needs to be “thinking about the platform as a product and these applications as products, not projects,” Hathaway said. “It gets away from shipping deadlines [and moves toward] interacting with each other and getting feedback from each other, even when you are big groups that are separate.”

DevOps creates a shift left approach, not only for developers who take on operations work, but for management and product development teams as well, who must consider how the features they build will affect the reliability of the application in production, writes Max Johnson, DevOps engineer at Pypestream and a former Holberton School student. [4] What this means practically, is that whenever an idea or new feature is suggested, all teams — product, Dev, quality assurance (QA), and Ops — come together to discuss the feasibility of the feature, what considerations should be kept in mind, and the minimum expectations of the feature. Two concepts that enforce this are behavior-driven development (BDD) and test-driven development (TDD). BDD sets user behavior as the standard to define what success means for a feature. TDD defines a number of unit tests which the feature must pass if it's to be considered acceptable. [5] In both of these approaches, the teams define what success means for every feature right at the start, before it is built.

"There are many benefits to the shift left methodology. It makes quality a priority for everyone, not just for QA and Ops. It saves cost and effort, and prevents a bad user experience by ensuring bugs are caught early on in the cycle," Johnson said. "It forces product, Dev, QA, and Ops teams to work together at the start. … This helps break silos between them, as they use their unique expertise to solve common problems."

Thinking of applications as products helps everyone start to prioritize the work more clearly. It's not so much about the fear of change, but rather the fear of failing that managers need to get over to embrace DevOps. Devs and Ops are the most open to change; it's the managers higher up who have to be convinced because they are most scared to miss a deadline or to go over budget, Pivotal's Hathaway said. C-level executives often have to intercede before it's too late. These are the ones lauding DevOps culture while still pushing too hard on those reporting to them.

"The middle managers are tasked with 'change the culture and make this successful,' but 'oh, by the way, we need this yesterday'," Hathaway said.

She gave the example of one line of business at one of her big banking clients which was worth millions of dollars. The CTO came in and said, "I don't care if we ship late. I don't care if things break. We have to change things from the ground up and slowly, and it's going to be painful way more than a win."

That's now the most innovative line of business at that large bank, and they act as an example for the rest of the company, Hathaway said. They were OK not hitting budgets or deadlines for a couple of months. Instead, they are now successful, with lower turnover and happier employees.

“From a grassroots perspective, [DevOps culture is] Devs actually having fun and learning new things every day. A healthy culture will draw people to DevOps.”
— Ashley Hathaway, engagement director at Pivotal.

It all comes down to middle and senior management buying in for the change, and then setting expectations for a bumpy, but worthwhile ride. At the same time, engineering teams must understand their role in creating business value. Dawson said when working with an organization’s DevOps teams to help them build their cloud native deployments to scale, he first identifies who the developers, QA, operations and business team members are. “I make a point that, ‘now, all of you are part of the business, because at the end of the day, we’re doing what we do to deliver functionality to support the business,’” Dawson said. “For your standard cloud native application, there will be operations and security [people], for example, asking ‘okay, how do I inject myself into cloud native development?’”

This openness between business and engineering applies to organizations of all sizes. Startups that have embraced cloud native DevOps culture from the outset also have their share of challenges compared to larger organizations. Even startups that have based their business models on cloud native architectures don’t always have the key DevOps teams that more established companies have, Dawson said. A startup will probably, for example, lack a business analyst team, Dawson said. To compensate, a successful startup “tends to have a highly networked process where the product manager/owner has a good understanding of the business,” he said.

Although business buy-in is critical to DevOps success, DevOps culture can’t be a top-down initiative. Instead, culture expands over time from a few successful teams to include multiple teams across the organization as the company’s

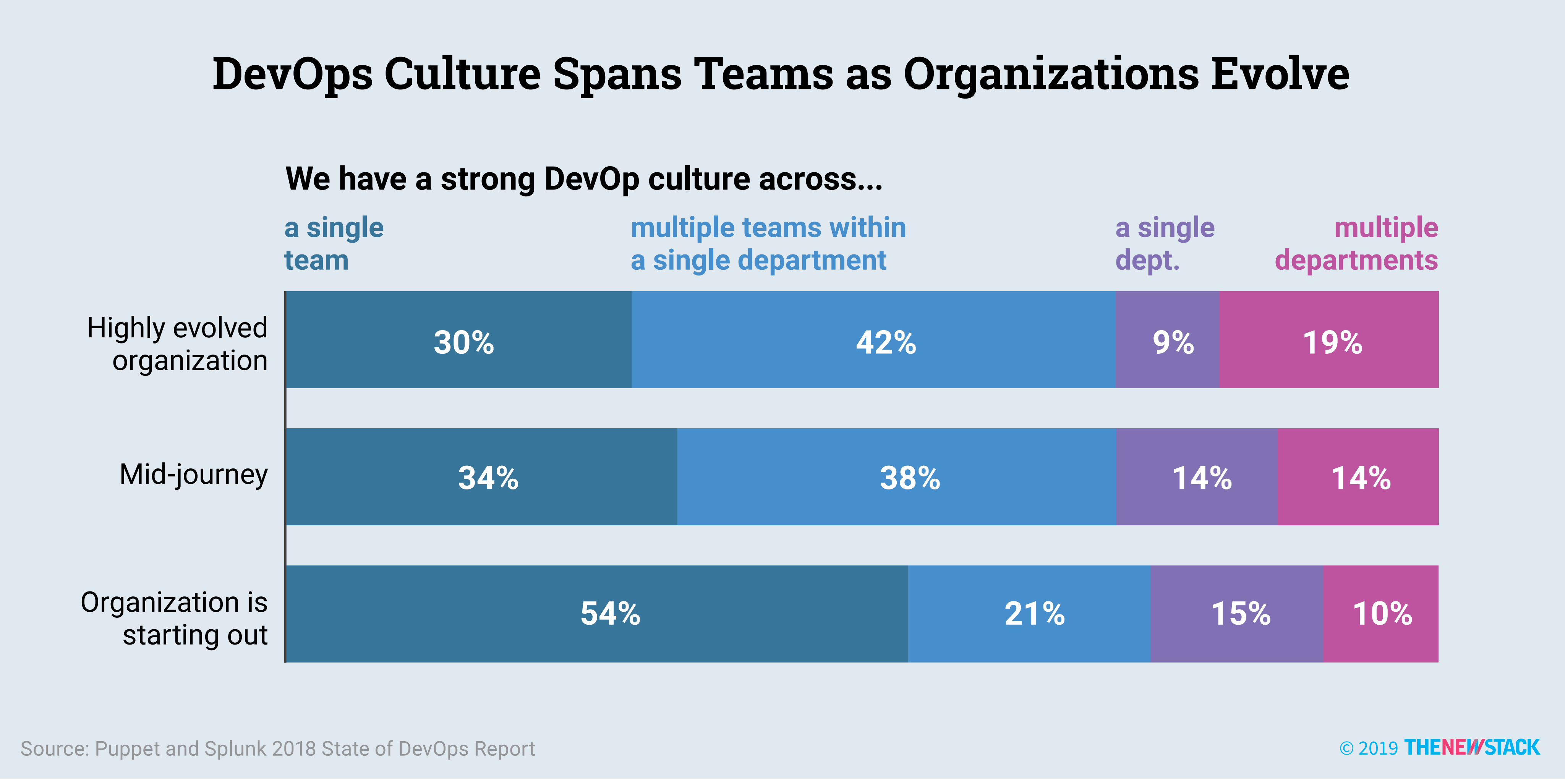


FIG 4.1: *DevOps culture begins with a single team and expands to multiple teams as organizations get farther on their DevOps journey.*

DevOps practices evolve, according to Puppet and Splunk's 2018 State of DevOps report. [6] "One or more teams automate a few key things; they reclaim time that used to be spent putting out fires; and they invest that time in further improvements, which helps to build momentum and support for change within their team. This proof of success builds trust inside and outside the team, and with appropriate organizational and managerial support, these pockets of success spread to other teams and across departments."

So how does an organization start to adopt a DevOps culture with buy-in from business leaders, Devs and Ops? How do you eat an elephant? One bite at a time, Pivotal's Nwabara quips. Start small, then scale. Collect your evidence and acknowledge the wins. Most companies of any size are a complex network of people, each with numerous communication touch points. There are many teams that must collaborate to deliver a given success. And "pivoting" an entire organization or even just a subset of departments does not happen accidentally and in a single step. So start small and refine as you go, to improve upon the way in which you solve customer concerns — one case at a time. Each learning contributes to the overall shift to improved communication and more acute empathy for the customer's needs. [7]

Still, it helps to have an understanding of what you may be up against before you begin. To understand the cultural barriers to your organization's DevOps transformation, it often helps to have an outside perspective. The New Stack talked with Nwabara as well as two consultants who are brought in once a company's C-level team starts to realize that IT isn't moving as fast as they would like it to. Organizations are struggling with outages and delivering customers joy, fighting to get system administrators to release the reins, and they just don't know what development and operations are doing, who in turn don't know what value they are delivering.

In short, companies want a DevOps transformation, but their culture is nowhere near ready yet. What kind of culture do you need to build? Who leads that culture? And who simply can't be a part of that fast-moving, autonomous future?

DevOps Is Not for Everyone. And That's OK.

Consultant Liz Warner, currently with fintech consultancy 11:FS, comes from the unique perspective of an interim CTO. She often comes into companies to help kick off or re-energize a DevOps transformation. At what point does she usually come in? When "people say 'Hey we're struggling. We hit a wall. We're growing very quickly and our current process doesn't scale. Or we have a process and it should be working but we have a lot of outages, our customers hate us'," Warner said.

She's come onto startups as small as three people, and to companies with tech teams of about 100, where their current CTO might not yet have the skills for running a larger organization.

“[With DevOps,] a key part of the culture shift is how you approach technical change. Change is scary. Making changes to a live system is scary. Things can break. The DevOps way of working is to make change less scary by changing constantly.”
— Liz Warner, consultant with 11:FS.

A DevOps approach, where in some cases everyone can deploy to production, is diametrically opposed to the norm at many companies. Typically, companies try to keep technical systems “safe” by limiting technical change and entrusting it to a small, privileged group.

Warner says her job involves a lot of talking to people and listening to their answers to questions such as:

- Can you trust other people on your team?
- Are you willing to do some introspection, and to improve based on what you learn?
- Do you believe automation processes can make change safer?

In the very traditional financial space, if even senior or middle managers aren’t committed, Warner says you’re destined to hit a wall.

“Somebody will want to put controls in that slow you down and interfere with your journey to rapid change, and you can end up in the worst of both worlds,” she said. “Any company can change their culture if they’re committed.”

The Three Commitments You Need from any DevOps Teammate

Warner says there are three nonnegotiable needs of a DevOps organization that must be made clear to everyone in the organization, and even key customers. There must be a company-wide commitment to:

1. Transparency.
2. Autonomy.
3. Automation.

Transparency

In a transparent organization, decision-making happens in the open, and performance results — both positive and negative — are freely shared. Taking this idea even further toward radical transparency, weaknesses and mistakes are also openly discussed and team members are encouraged to share their opinions, regardless of their position or level of experience. [8] Warner called companies that move toward radical transparency "really healthy."

> "People should understand shared goals. The sort of closed, need-to-know thinking gets in the way of product and technical delivery. Everyone should know how systems work if they are interested."
>
> **— Liz Warner, consultant with 11:FS.**

On one contract Warner worked on, only two people — "the priesthood of sysadmin" — knew how key systems worked, for fear of reverse engineering. This gatekeeper culture means no one can improve things, no one can point out scenarios that won't work, and, when those two sysadmins eventually leave the company, it's really in trouble.

Jira and Confluence are solid examples of project management tools that promote information radiation in a wiki-like way, where anyone can comment and make suggestions for improvements. But the process is more important than the tools.

"Open by default is a good starting place for DevOps culture, especially combined with introspection. What can we do better? How did we do last month? Last week?" Warner said, suggesting that organizations carve out space in people's work cycles for feedback and improvement.

Autonomy

DevOps organizations need to hire the right mixture of people, and make sure engineering teams understand the business. This includes making sure engineers know regulatory constraints and what customers and partners are expecting from the code. Make sure technical teams understand business problems, then give them freedom to come up with solutions to those problems.

Warner says part of the job is continuous learning. There is always a better tool or library and so many different solutions to existing problems that you get better results with more individuals who are continuously learning toward collective objectives.

Challenge everyone in a DevOps organization with continuous feedback loops and actually make changes based on the feedback, she said.

Automation

Finally, Warner sees that DevOps transformation requires a commitment to automation. If you want to have loads of tiny safe changes in a process where any engineer can trigger a code release, you need to make sure your automation — which includes infrastructure as code (IaC), continuous integration and test automation — is in excellent shape, and that you are using automation at every stage, including production.

When someone says "I'll automate everything up to production but then I don't feel safe until I do things manually," they aren't ready for DevOps-style rapid change, she said.

Just remember, Warner said, "If tech is driving for a collaborative culture but the rest of the company isn't interested, it's not going to work. It has to be across senior management and product."

Pair Programming Builds Empathy Across Departments

At Pivotal, where Hathaway works primarily with very large banks, every engineer pair programs all day, every day. She recommends her clients kick off DevOps with pair programming or, at minimum, user interviews between Ops and Devs, so each side can see what the other is doing.

“For our clients, [pair programming] seems counterintuitive. ‘Now I’m cutting my time in half.’ But it actually strengthens code checks. You write better code. Then the domain knowledge doesn’t leave if someone is sick. Everyone deploys on the same pipelines [and] writes their own pipelines with continuous delivery,” Hathaway said.

DevOps is all about breaking down silos, but some silos are necessary, especially in banking security. Hathaway says, with banking regulations, the person who writes the code can’t ship the code. This is a bonus of Pivotal’s pair programming policy as two people together can continuously ship and test the code, with checks along the way.

One application (app) development team saw great success by simply talking to an Ops team.

“Just the interaction of that one application group, those Devs giving their feedback to the Ops team, was a lightbulb moment for them,” she said, pointing out that Ops wasn’t talking to their developer users before.

Putting the Devs who are using the infrastructure, system rules and integrations every day in front of the Ops team is a logical first DevOps conversation. The same is true for increasing communications and breaking down silos between IT and business management. What if middle management won’t change? Try pair managing, says Hathaway: “Link up your weaker people with your really strong performers to see what different thinking looks like, because it is a practice in thinking. They need to know it's OK from their management."

Five Steps to Improve Interdepartment Communication

When a customer-facing team was feeling pain with Pivotal's developers, it didn't take long before the company realized this was not a software problem, but instead, it was a communication issue, Pivotal's Nwabara writes on The New Stack.

The company has since put out that fire, but the incident also drove home how software usually solves problems, while sometimes solutions could be as simple as changing people's mindset. This adjustment can impact teams greatly and result in a significantly positive outcome. You may already have what you need to make a huge impact, and the cultural transformation is a critical element for digital transformation. Everyone's ultimate goal is to ensure that the company provides recognizable value to customers through a positive customer experience.

Here are five approaches that Pivotal is taking to improve communication and build empathy between their research and development (R&D) and customer-facing teams in order to deliver a better experience and value. We have already discussed the first:

1. How do you eat an elephant? One bite at a time.

Start with a subset of the organization.

2. Experiment, Experiment, Experiment.

An experiment is not a true experiment unless the results mean something. The results can be positive or negative, but there needs to be a finding, or a learning. To take this concept one step further, if learnings are not applied to the original process, the experience is wasted. So as organizations work towards improving the ways in which teams work together to deliver customer value, they must be aware of the fact that if their learnings are not directly incorporated back into their practices, there was no point running the experiment in the first place. What lessons from this experience make it more sustainable and scalable across teams? Think about, and promote, running experiments beyond the software or products a single team happens to build.

3. Understand Roles and Responsibilities.

Revisit your organization's structure and think about if it's truly helping or actually hindering. Something that Pivotal realized is that Conway's Law doesn't only describe how the software you build looks like your organizational structure. It's true of the overall customer experience as well — customers can feel how you structure your departments if that structure is hindering collaborative efforts to move towards effective solutions. This is something that is surprisingly similar to software development and product delivery. Pivotal's R&D, support and balanced account teams are arranged across independent departments. It's important for companies to be intentional about the way in which they enable communication, which leads to the next point.

4. Clear Communication Around Deliverables.

When collaborating, it can be very easy to assume that information has been expressed and understood clearly. However, in practice, this is something that should not be taken for granted, as a miscommunication can result in costly consequences. It is worth the effort to spend some time thinking about all the touch points between people and information to discover bottlenecks.

Pivotal recently came across one example of such miscommunication in its release notes. Release notes describe feature changes and bug fixes, which is generally what to expect from the most recent release for a given product upgrade. Inconcise, incomplete, vague or ill-expressed information in said notes results in customer confusion, as well as inter-team confusion. It's easy to assume that what you know, everyone knows. In a couple of recent instances, Pivotal discovered that a miscommunication meant a change to the way information was logged. There was a miss in one case because the team doing the development did not realize that someone would care about the changes made.

5. It's a Relay Race: Develop a Shared Understanding of Who is Holding the Baton in the Delivery Process at Any Given Time.

Product development is not only software engineering, but also the handovers in between teams. With a complex product like Cloud Foundry, where microservices are built by many interlocking small development teams, the pathways for communication are also complex. Having checks and balances in your product's path to the point at which is value delivered to the customer is integral to successfully meeting customer needs. When information changes hands so many times, important details can get lost.

Similar to the responsibility of multiple runners in a relay race to carry the baton, in software delivery, the responsibility to carry the baton is handed from team to team. This means that every runner must understand the terrain of the race and explicitly where those handoffs are happening. The entire team is aware of the full map even if they are only responsible for and hone in on — effectively own — a short part of the journey.

When processing information, such as business value, delivery dates, quality or changes made to existing functionality, it can be difficult to know who actually owns the current state of the solution at any given time. Pivotal had to reassess the way it communicated by looking at its feature pipelines and also the way in which information around release updates was managed.

After developing a shared communication contract, it was exponentially easier to increase confidence and visibility into what product teams were working on, as well as decrease the number of stressors placed on the various characters in the path to value for the customer.

These five steps are by no means a comprehensive solution to fixing all communication problems or developing a DevOps culture. However, they are a good place to start as you work towards assessing the current state of how your company communicates between departments, teams and externally to the customer. Every company is at a different place in its DevOps journey and there is still a great deal of work to be done. When it comes to improving communication, transparency, autonomy and building empathy, there is always room to grow.

CHAPTER 05

Role of DevOps in Deployments: CI/CD

All along, the idea of the cloud has been one of abstraction. Moving storage to the cloud not only means you can't point to the physical drive where your data is stored, but storage is theoretically limitless and available on demand. Similarly, moving compute to the cloud means your processing power can be increased exponentially, without the need to procure and provision physical servers.

While the move to the cloud has solved some problems, other problems have inevitably come to take their place. With cloud native technologies, we are again seeing a new level of abstraction, where technologies previously reserved for on-premises scenarios are moving to the cloud and experiencing the increased capabilities and complexities that come with it. At its core, cloud native DevOps means yet another transformation to undergo and a new set of ideas to incorporate. In practice, this means new tools and workflows — alongside the culture shift described in the previous chapter — which have wide-ranging implications.

Emerging GitOps, SecOps and DevOps practices paired with a CI/CD pipeline on top of Kubernetes will speed up your release life cycle, enabling you to release multiple times a day, and allow for nimble teams to iterate quickly. With

Kubernetes and cloud native DevOps patterns, builds become a lot faster. Instead of spinning up entirely new servers, your build process is quick, lightweight and straightforward. Development speeds up when you don't have to worry about building and deploying a monolith in order to update everything. By splitting a monolith into microservices, you can instead update pieces — this service or that — and also encourage autonomy among development teams who own the full life cycle of that service. [9] However, complexity also increases as each piece now needs its own delivery pipeline as well. Teams test their own code and deploy directly to production. It's of critical importance in production environments to have services that are thoroughly tested at all stages of development.

"The impact of cloud native on CI/CD is profound. As engineering teams adopt microservices, the number of pipelines they need is growing exponentially," Dan Garfield, chief technology evangelist for cloud native CI/CD platform Codefresh, said. "Now instead of needing to support a release process of a single monolith, they need to support a release process of dozens of microservices."

The shift to cloud native can be a double-edged sword and adapting your CI/CD tools and practices is mandatory to keep pace. Overall, cloud native offers opportunities in terms of velocity and scale, but also increased complexity, as teams move from handling single monolithic applications to multifaceted microservices. The lines across the stack are increasingly blurred, with more dependencies and more layers.

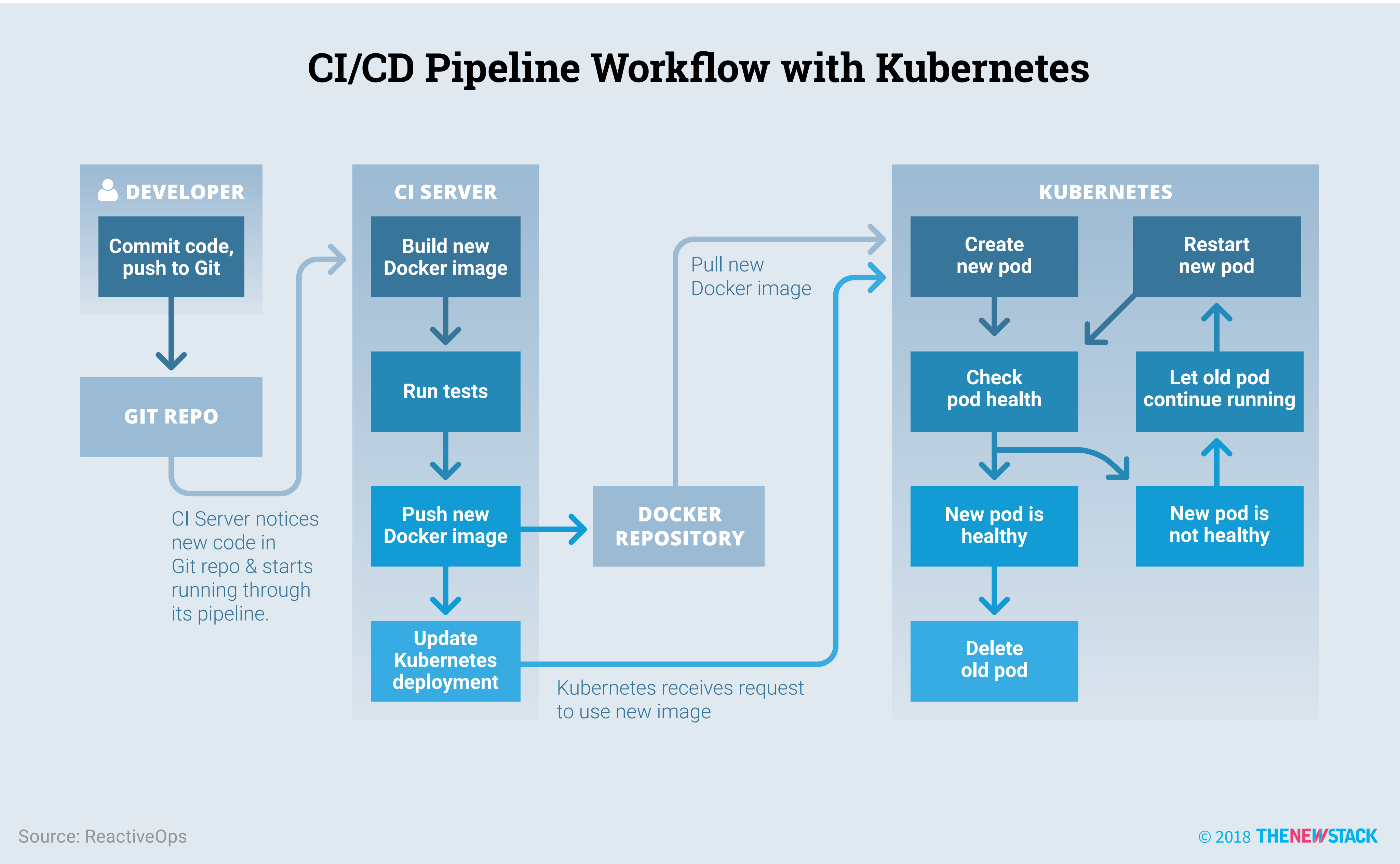
"As cloud native approaches gather steam, CI/CD practices have to evolve to maintain stability as you increase speed. Because without the right guard rails, it's like attaching a rocket ship to a go kart. It's not a very fun ride."
— **Ravi Tharisayi, director of solutions strategy at New Relic.**

One way to handle the increased speed is through increased automation and a culture of experimentation, which Tharisayi points to as critical for cloud native CI/CD.

"Whether it's canary deploys, A/B tests, blue-green deploys or a combination of these, the right deployment strategy can help test new features with velocity to improve the customer experience," Tharisayi said.

While not unique to Kubernetes, a containerized approach can make tests more straightforward to run. If your application tests depend on other services, you can run your tests against those containers, simplifying the testing process. A one-line command is usually all you need to update a Kubernetes deployment. In a CI/CD workflow, ideally you run many tests. If those tests fail your image will never be built, and you'll never deploy that container. [10] Below is an example workflow for a CI/CD pipeline on Kubernetes. In the next

FIG 5.1: *Tests and health checks can prevent bad code from reaching production. As part of a rolling update, Kubernetes spins up separate new pods running your application while the old ones are still running. When the new pods are healthy, Kubernetes gets rid of the old ones.*



section, let's explore how DevOps patterns and processes change with this cloud native approach.

Cloud Native Automation Begins with GitOps

Increased automation not only means increased deployment speed, but also an expanded role for developers who contribute to automation via GitOps, as Codefresh's Garfield explains. As Fig. 5.1 illustrates, Git is becoming the starting point for a CI/CD workflow on Kubernetes and the standard for distributed version control, wherein a Git repository (repo) contains the entire system — code, configuration, monitoring rules, dashboards and a full audit trail. GitOps is an iteration of DevOps wherein Git is a single source of truth for the whole system, enabling rapid application development on top of Kubernetes.

"GitOps" is a term developed by Weaveworks to describe DevOps best practices in the age of Kubernetes, and it strongly emphasizes a declarative infrastructure.The fundamental theorem of GitOps is that if you can describe it, you can automate it. And if you can automate it, you can control and accelerate it. The goal is to describe everything – policies, code, configuration and monitoring – and then version control everything. [11]

GitOps is all about "pushing code, not containers," said Alexis Richardson, CEO of Weaveworks and chair of the Cloud Native Computing Foundation's Technical Oversight Committee, in a KubeCon + CloudNativeCon EU keynote. The idea is to "make Git the center of control" of cloud native operations, for both the developer and the system administrator, Richardson said. [12]

"The key point of the developer experience is 'git-push,'" he said, alluding to the Git command used to submit code as finished. He added that this approach is "entirely code-centric."

The open source Git version control software makes perfect sense as a frontend for cloud native computing. Given its nearly widespread usage by developers, Git commands are the "Lingua Franca" of the open source world. The basic idea is that all changes to a cloud native system can be done through Git. Once a commit is made, it sets off an automated pipeline, perhaps using a tool such as the Continuous Delivery Foundation's (CDF) Jenkins X or Spinnaker, to containerize and test the code and press it into production.

GitOps could work exactly the same way for infrastructure management as well. Richardson calls this approach "declarative infrastructure." Make changes in configuration through a YAML file, and Kubernetes can detect changes in the file and adjust the resources as necessary. Weaveworks itself has released a number of tools, such as kubediff, for comparing the desired state with actual state in such cases.

The idea of GitOps is catching on in many organizations outside of Weaveworks. Alex Ellis, a senior staff engineer at VMware and creator of OpenFaaS, has built a native "GitHub app" called OpenFaaS Cloud that runs an automatic CI/CD pipeline that can be triggered by a Git command. OpenFaaS Cloud is a number of serverless functions that sit in between the user's Git repositories and a deployment of OpenFaaS. When the user pushes serverless functions into the GitHub repository, that code is checked out, built, tested and then pushed into a Docker registry from where it is deployed as a function.

Microsoft clearly sees Git as central to the developer experience, as evidenced by its $7.5 billion acquisition of GitHub. Microsoft will look for natural integration points with GitHub and its own developer tools, many of which it plans to offer on the GitHub Marketplace, according to the company. [13] Google Cloud is also a proponent of GitOps, though it says a few modifications make the approach more suited to complex enterprise environments. [14] Andrew Phillips, a product manager in Google Cloud Platform's DevOps division, explains that there are primarily two challenges with common implementations of GitOps: "First, running your release processes and approvals largely through source code repositories isn't ideal in larger organizations. Second, it’s not always best to choose a ‘pull’ approach to make environment as code real; a ‘push’ approach can be more suitable in many cases.” These issues crop up with organizational complexity, and this is where simple implementations of GitOps fall short, he said.

Phillips advocates limiting the number of repositories you maintain and deploy from, though the mono versus multirepo approach is still a favorite subject for debate among developers. One repository per team is better than multiple repositories for every team, but in a large organization with many teams, this is still hard to manage, he says. In this case, you can have a shared repository for each environment. For example, a staging cluster that five teams deploy to can have its configuration hosted in a single repository. He also advocates putting new code through a “dry run” or a “preview environment” before issuing the “push” command; if it fails this test, the deployment is not carried out. But if it succeeds, the configuration is pushed and validated using a smoke test or other verification step, and you then commit it to your master repository. This increases confidence in each deployment and results in a stable master branch with a commit history of successful deployments. You can still include manual approvals as part of the pipeline to have more control where required, but the goal is to have the approval and verification process automated as much as possible.

One early adopter of the GitOps approach has been the content platform team of the Financial Times, according to a KubeCon + CloudNativeCon EU keynote talk from Sarah Wells, who is their technical director for operations and reliability. The company’s content platform is built from 150 microservices. In 2015, the newspaper moved this platform over to Docker from virtual machines. After migrating to containers, the Financial Times found the number of releases it did increased from 12 a year to 2,200, an increase that also came with a much lower failure rate. Instead of spinning up a new virtual machine, all the changes are simply made to a YAML file. While she didn't mention "GitOps" by name, Wells did say developers push all their changes through GitHub.

Building on the idea of Infrastructure as Code, Seattle startup Pulumi is advancing the notion that DevOps teams need not exclusively use GitOps and YAML to deploy to Kubernetes. Instead, developers can use the languages and tools they're most familiar with to deploy to any public or private infrastructure, and across infrastructures. The company has released a set of software libraries and tools called the Cloud Native SDK that allows developers to write applications in JavaScript, TypeScript, Python and Go that work across Kubernetes deployments on multiple clouds or private data centers. [15]

"If you just want the GitOps experience or the command-line interface (CLI) experience, you can take your existing YAML files and deploy them as they are without changing them at all. Maybe you want to convert one YAML file at a time or convert as you go, that type of mixed environment — we call it brownfield — is really important for customers with large investments in Kubernetes," Joe Duffy, co-founder and CEO of Pulumi, said. "Similarly, we can take Helm charts and just deploy them. If I want to provision an S3 bucket in a Kubernetes app, usually I have to use two different toolchains for that. It's astonishing to me the lengths people will go to get tools to work together that were just not designed to work together."

Regardless of how the CI/CD pipeline kicks off, all cloud native applications put configuration into code. This puts how an application runs much closer to the application developer than ever before, said Garfield. This has implications not only for how CI/CD pipelines are built and automated, but for those who manage that work as well.

"In the old world, operations would worry a lot about how an application would run and how it would be deployed," Garfield said. "One side-effect of cloud native is that developers take more responsibility for how their application will run and how it gets deployed."

Breaking Down the Wall Between CI and CD

Through declarative infrastructure — configuration as code — and GitOps practices, operations work is "shifting left" toward the build side of the build > deploy > manage pipeline. Organizations are now able to deploy cloud native application infrastructure at rocket speeds. It's become more common for DevOps to schedule several releases a day — or 4 million builds per day if you're Google [16] — as opposed to when code updates and release cycles took weeks or even months to complete in the not-so-distant past.

CI/CD also plays a key role in what DevOps teams have been able to achieve as they shift away from on-premises and virtual machine environments to take advantage of the stateless environments on offer on the cloud. As cloud native DevOps matures, we will continue to see advances and improvements in the production cycle in a number of ways. One such example is how DevOps can leverage microservices to merge CI with CD.

"We found that to go fast, you need to get rid of these CI/CD blocks that we often didn't think about before," James Strachan, senior architect at CloudBees, the project lead on Jenkins X, said during DevOps World | Jenkins World 2018 in Nice, France.

Before Kubernetes, only the CI half of CI/CD was automated. CD was assembled by hand with scripts, pipelines, metadata and configurations. Kubernetes enables CD automation, and some tools make it simple to deploy on Kubernetes. Merging CI and CD allows automation of the full software development life cycle — "it's automating the automation," as CloudBees co-founder and engineering manager Michael Neale has stated — and essentially commoditizes CI/CD. Having consistent, end-to-end pipelines also enables more feedback — and eventually more automated feedback — to developers on how their code is performing in production. [17]

"It's that kind of extra feedback and intelligence that really helps us deliver software. So I think we're going to have more and more automation that's like smarter services that can analyze what's happening in production and give you pull requests to say 'We recommend you make this code change now,' or 'We recommend you revert that version that you've just gone live with because of X, Y, and Z.' And then you as a developer are more directing changes rather than always having to change everything by hand," Strachan said.

In many ways, merging CI/CD is critical to achieving continuous delivery "as the ultimate goal of DevOps," Torsten Volk, an analyst for Enterprise Management Associates (EMA), said. "Developers need to constantly be able to ship out new code and instantly receive production feedback, simply by making the new code available to a small subset of end users. Observing the behavior of this new code enables developers to further fine tune their application or microservice, based on real-life user requirements," Volk said. "At the same time, developers receive instant feedback in terms of how their new code impacted performance, reliability and, ideally, cost."

Merging CI with CD and supporting it all with DevOps also means better security. "By lifting this artificial separation between CI and CD, IT operators can centrally address continuous security and compliance as the central pain points of today's line of business," Volk said. "This should happen through the implementation of security and compliance as code to centrally define and enforce requirements in terms of code and infrastructure configuration, data handling, and overall deployment architecture."

Successfully merging CI and CD is contingent on incorporating specific tools into the pipeline. Automation, of course, tops the list of essentials. A developer who also deploys on clusters, for example, will likely see microservices on the cloud "grow like wildfire," Strachan said. "And so fairly soon you need to do some kind of automation to keep track of all these things and test them."

To enable this end-to-end automation, the cloud native CI/CD marketplace has exploded over the past year. And a collaboration between industry leaders to form the CDF aims to establish industry specifications around pipelines, workflows and other components of CI/CD systems to help grow the ecosystem of tools and services. While most of the solutions are CI tools that have been extended with CD capability, some are purpose built for cloud native CI/CD, with a few others that focus on CD only. Some popular tools include Atomist, AWS CodePipeline, CircleCI, Codefresh, GitLabCI, Harness, Jenkins and Jenkins X, Puppet (which acquired Distelli) and Spinnaker.

Hidden Bottlenecks

Putting development on hold as different teams along the production pipeline do their work represents an area for improvement in the CI/CD development cycle. For organizations with a limited number of testing environments, for example, a non-essential update or a four-hour long test might prevent an urgent code fix from being completed. This scenario serves as a concrete example of how merging "CD with CI can remove these kinds of bottlenecks," CloudBees' Strachan said.

Previously, teams would consider separate deployment tools for CI and CD, Strachan said. When merging CI with CD, the processes are completed in parallel, which "creates a dynamic preview environment for each pull request that gets deployed into its own separate dynamic environment," he said.

It is thus possible for tasks such as security scanning or vulnerability checking, and other tests in parallel, so "if one person is working on an urgent P1 fix to fix a memory leak, and then someone else is figuring out what icon to use on the homepage, one process can take a week while another one can go right through," Strachan said. "And once the pull request is green, you can almost go straight to production."

HSBC has relied, in part, on CI/CD tooling to "rebuild the bank from within" as a way to revolutionize the way that HSBC delivers to their customers, Cheryl Razzell, global head of platform digital operations for HSBC Operations, said.

“This required building many of the processes largely from scratch and assembling the teams to build the infrastructure. We rebuilt some Jenkins masters and realized that we want to stabilize the environment,” Razzell said. “There was another tool in the CI/CD platform that was then failing so we had to work our way through the stack to rebuild the stack so we entirely built a new digital infrastructure for our CI/CD platform.”

The merging of CD with CI for cloud native deployments also involves combining the roles of development and operations. Development and deployment are merging as CI/CD becomes geared for the cloud. Developers may also assume many operations-related roles.

“As there is more of a focus on CI/CD for cloud deployments, the developer’s job also often involves testing, automation, deployment and monitoring, Nitzan Shapira, co-founder and CEO at Epsagon, said. “The developer is actually the one now deploying and updating who has access to the system. And if the design of the microservices is good, a small team of developers can be in charge of the service. The developers are in charge of the production … making [them] more empowered and have more impact in the organization.”

Automating Security in CI/CD

Since most CI/CD processes involve containerized or virtualized services, often hosted in the cloud, many organizations fail to fully understand the security risks associated with these processes being compromised.

CI/CD deployments are made up of multiple parts, sometimes separate tools, that interact with each other in an automated fashion to spin up and allocate resources as needed to build, test and deploy new code. In most setups, the build servers have an implied trust relationship to code repositories and trigger the processes automatically on pull requests. The code is built conditionally, run through tests and the resulting image is pushed to an image repository. From there, it's automatically picked up by deployment services and pushed to microservices running inside containers.

All this is achieved through a chain of trust relationships, dependencies, configurations and credentials, some of which can be modified or stolen by attackers if not properly protected. At the very least, compromised CI/CD servers can provide hackers with access to free computing resources that can be abused for crypto mining, building distributed denial of service (DDoS) botnets or proxying malicious traffic. But the security implications far exceed that, particularly for on-premises, self-hosted deployments.

"If you take over one of these build systems, even though it's running in a container, you could take over a network because these containers share IP space," said security researcher Tyler Welton in a talk about CI/CD hacking at the DEF CON 25 conference. "Even if they might be on their own mesh network of IPs, they still often have ports mapped to the hosts."

In some cases, it's even possible to exploit the trust relationship between these servers and code repositories in order to make commits back to master, compromising the code. At the very least, they can abuse the authorized SSH keys that these services use.

Another common practice observed by Welton in CI/CD deployments is the storing of credentials for other services in environment variables. And while this is better than hard-coding credentials in configuration files, they are still exposed to theft in case of a compromise.

"When you compromise one of these services, you haven't compromised the entire system, but dump some environment variables and you'll probably be able to pivot to some of the other systems," Welton said.

Even though some modern CI/CD tools allow restricting privileges inside containers, a lot of systems are configured to run services inside containers as root. At first glance, this doesn't seem to be a big deal, because any potential attackers would only be able to perform actions inside those particular containers, which are often short lived.

However, root access allows attackers to scan the entire IP space in order to find other potentially exploitable services running on the host, and if the container has internet access, it allows them to download and install additional packages they need to launch further attacks.

Welton developed and released a framework for CI/CD exploitation called CIDER (continuous integration and deployment exploiter) that supports various build chains like Travis CI, Drone or CircleCI. The main exploitation vector used by CIDER consists of pull requests through open GitHub repositories, but the same techniques apply if attackers gain access to private source code repositories.

There's also an older CI/CD audit framework called Rotten Apple that was created by Mozilla ethical hacker Jonathan Claudius. This can be used to determine if the root user is being used to build projects and if attackers can deploy malicious code to steal API keys, to pivot to private networks, to authenticate using GitHub credentials, to create reverse shells, to exfiltrate data, to access other projects on the same server or to steal SSH keys. The framework also has an attack mode, which can be used for penetration testing.

Welton's 2017 talk at DEF CON contains real-world CI hacks and a wealth of information about different configuration issues. However, the risks posed by CI/CD tools has been known in the security industry for years.

InfoSec expert Nikhil Mittal's presentation at Black Hat Europe two years earlier is also a great resource about insecure default configurations in CI environments.[18] At the time, Mittal described CI tools as "an attacker's best friend" and said that he never encountered a penetration test where unauthorized access to a CI tool didn't result in administrative access to the whole network domain.

It's important for organizations to understand that default CI configurations are not designed with security in mind, and that running things in containers in not sufficient to stop attackers from breaking in and performing lateral movement through the rest of the infrastructure.

The security of CI/CD deployments is even more important these days, in light of a recent spike in software supply chain attacks where hackers break into software development infrastructure in order to insert backdoors and malicious code into resulting applications. This allows the hackers to compromise a large number of end users by taking advantage of trusted software distribution channels. It also makes developers a highly attractive target.

CI/CD can have benefits for security. For one, it makes remediation and the deployment of patches much faster. Also, splitting applications into microservices helps reduce single points of failure and contain compromises, if configured properly. However, having insecure CI/CD systems in your infrastructure increases your attack surface and opens entry points for hackers.

"The automated build systems, like the CI systems and the CI pipelines, are checked less for security than the code in which they're deploying," Welton said. "They sit in between the infrastructure components, which are being tested through network penetration tests, and the application code which is handled through application pen tests. But then you've got this quasi-containerized environment that's sitting on its own IP space, in its own containers, on top of the infrastructure, but below the code, and it's really not being tested."

Fortunately, container security technologies are now following the same trajectory as other cloud native technologies to become part of the overall developer experience. The shift left approach, which gives developers more responsibility for deployment and also gives them more responsibility for application security. And tools and best practices are evolving to support cloud native architecture. Organizational changes that put more responsibility in developers' hands must be combined with emerging best practices for securing microservices-based applications. Companies are thinking more deeply about developing separate pipelines for various workload types, for example, along with a number of other best practices described in The New Stack's Guide to Cloud Native Microservices. [19]

Application security is now increasingly a programmable abstraction that's more automated, commercially available and capable of easily integrating into CI/CD pipelines. By automating security and compliance tests and integrating them with your CI/CD tool, for example, compliance flaws will be discovered earlier in the process, making any bugs less expensive to fix and the likelihood of a developer introducing compliance issues into the codebase significantly less, writes Scott Fitzpatrick, a Twistlock contributor for The New Stack. [20] "When compliance tests run as the result of each commit, they should ensure that the code being committed is compliant with the necessary standards. Test failure will mean build failure, and build failure will immediately notify the developer that action must be taken prior to integrating his/her code changes."

Continuous integration and continuous delivery of cloud native applications has transformed the way teams collaborate. Transparency, observability and automation at every stage of development and deployment are increasingly the norm. GitOps and SecOps, both enabled by cloud native architecture, are building on current DevOps practices by providing a single source of truth for changes to the infrastructure and changes to security policies and rules.

CHAPTER 06

Case Study: The Art of DevOps Communication, At Scale and On Call

Each DevOps transition is as unique as the company that is going through that silo-breaking, restructuring, cultural change. But there are some similarities found in companies of all sizes. DevOps is meant to unite tech teams around individual ownership, responsibility and feeling like each person has a stake in the business. How you communicate and scale these changes dramatically affects the success or failure of a DevOps transformation.

We spoke to a team at Microsoft that is about a year into its DevOps transition, and a marketing automation startup, Klaviyo, that has been DevOps-driven from day one. No matter where in your DevOps process you are, you surely can use the advice from these software reliability engineers (SREs) about communication during key points in a DevOps transformation.

How Klaviyo Scaled with DevOps

DevOps takes a certain personality profile. Since companies are in constant flux, you need a team of individuals committed to a common vision. For Laura Stone, senior site reliability engineer at Klaviyo, this all comes down to hiring the right people.

Klaviyo, a tool focused on applying big data and segmentation to personalized marketing, is a five-year-old company that has done DevOps from the start. For the first two years it was just the founder. But in the last three years, Klaviyo has scaled its DevOps team to 150 people. That's why the company's interview process is more of a hands-on culture and code test than a Q&A.

> “I don't look at a ton of resumes — they're very difficult for SRE roles. Show me the code.”
> **— Laura Stone, senior site reliability engineer, Klaviyo.**

The first step is a take-home assignment, or what Stone calls a simulation for what it's like to work at Klaviyo. Candidates have to write a small CRUD (create, read, update and delete) application that deals with weather data and sends people a personalized email. The right candidates don't have to be masters of certain languages, but they must show they are eager to learn and that they are thinking of the next person who has to use that code, including attention to documentation, algorithms, readability and cleanliness.

“I love it when people write tests — it shows that it'll be easier to use your code in the future. I think a lot of people should test their code and document their code and they just don't do it,” Stone told The New Stack.

DevOps isn't just about scaling a company, but scaling a code base.

Stone gave the example: “Let's say you had 100 people signed up to this service and 40 people are in Boston, do you make one API call for each or create it and use it once and cache it?”

Once you pass your first test, you come in for some collaborative coding. Candidates are put in charge of refactoring and are allowed to ask as many questions as they like about anything. The goal is to understand how they work and look for candidates' openness and willingness to admit when they don't know something.

The next part of the interview test is about DevOps ownership. They inform the candidates that they now own this code, asking them questions like:

- Where do you want it to run?
- How do you want to be notified if it fails?
- What's the infrastructure it's running on going to look like?
- How would you scale to different user needs?

Stone says they aren't looking for candidates who know all the answers, especially those fresh out of university, but they want to see signs of a desire to want to be on-call and to own their service, from creating to maintaining it. Similarly, they don't expect precise answers, but for responses that indicate a candidate's ability to think ahead. They should suggest running their code on a machine, not their computer, so they aren't tying it to one person and personal infrastructure. They should think about where automation can speed up processes. Most importantly, they need signs of customer empathy and a desire to make sure the solutions are as stable as possible.

Stone says they are looking for "people who are motivated to learn and who are technically savvy and who can show empathy. Given the right structures and resources in place, then they can be successful owning their service from start to finish."

When Stone joined, there was just one product team and an SRE team. Eighteen months later, there were an additional four or five product engineering teams focused on specific areas of the solution. They have a greater need to concentrate and codify knowledge transfer as engineers can no longer know the entire product and, in many cases, can no longer have deep relationships with other engineers on the team.

Scaling DevOps all comes down to one question: How can communication flow and how can people still have ownership?

One form of scalable knowledge transfer they use is mob programming. It's like the Agile Methodology's pair programming exercise, but the whole team is working on the same thing at the same time on the same computer. Klaviyo did this to help train the team on Terraform when they adopted it to automate their infrastructure. The SREs acted as mentors to the product teams, Stone said.

She thinks this knowledge transfer is working because it used to be exceptional if she didn't get paged on call. Now it's exceptional if she does.

The Art of DevOps Communication During Incidents

One of the important aspects of DevOps is breaking down barriers and providing cross-functional training so everyone feels an equal responsibility for code that's being released. For most DevOps teams, that means streamlining incident response and instituting all-hands-on-deck, on-call rotations. When you are trying to create a world of always-on continuous delivery and integration, you need people willing to work increased uptime, any time of day or night.

"DevOps is a buzzword, so given that we moved to DevOps, it was really up to us to define what that means for our team, which for us meant everyone is responsible for infrastructure and you're on call," Nida Farrukh, senior site reliability engineer at Microsoft, said.

Farrukh was part of the Microsoft social engagement (MSE) and market insights team's DevOps journey, which began a year ago. This small, nimble team is working to minimize downtime for thousands of customers. In a recent restructuring, a handful of SREs are now sharing infrastructure and on-call responsibilities with developers.

"This generally increases the health of your monitoring system because the people who are writing the code are fixing it and feeling the pain points too," explained Farrukh.

Each team member is on call for one week at a time. To start, each trainee has a shadow week, acting as backup for an SRE or another fully trained developer. Then, within a few weeks, the trainee takes on the role of primary administrator on duty (AOD), and the more experienced person shadows. There are also ample tutorials, documents and regular simulated outage exercises to assist.

"People generally are very nervous when they go on call for a service for the first time and they've never been on call for anything," Farrukh said.

She continued that it isn't about knowing a product inside out, it's about knowing where to find what you need:

- Do you have the tools to debug a new problem that comes up?
- Do you know where to find the answer in documentation?
- Do you know who can answer doubts and how to contact them?

On the MSE team, while there is some leeway to choose appropriate tools for tasks, the infrastructure team works hard to keep standardization across their stacks, with a limited number of logging systems, languages, libraries and monitoring systems, so everyone shares a baseline knowledge.

Ask for Help

A lot of times new folks can feel afraid to ask for help, but Farrukh's team works hard to emphasize that everyone can and should ask for help, because in DevOps the entire product is the entire team's responsibility.

"In the MSE team, as AOD you are first responder, not sole responder. You can pull in anyone in the entire team to help you during an incident," she said.

The engineering leads and SREs have even gone out of their way to volunteer to respond any time.

"An AOD has the power to call anyone, but there is some sort of psychological barrier so these people have come out and said 'Please call me'," Farrukh said.

She said software architects are usually a good first contact for issues within a DevOps organization, because if they don't know exactly the problem, they'll know who to call.

Farrukh opines that AODs can and should delegate tasks that act as distractions for their main goal: debugging efficiently. Even looking for the right contacts and then calling them can be a distraction.

She suggests two roles to help limit these interruptions: bridge manager and communications (comms) person.

The bridge manager can be in charge of bringing on everyone that should be on a conference call about the issue. The bigger the issue, the more people may be called in from the escalation matrix.

Other engineers, executives and even people in marketing and customer service often have loads of questions that can distract the AODs from their work. The suggested comms role — something Farrukh says she often volunteers for — fields those questions and is the main point of contact for the AOD to disseminate information through.

For smaller issues, the comms and bridge managers may be the same person or even the AOD herself for really small issues. In reality, on the MSE team, the AOD often has five to six engineers helping her with larger problems, including SREs for faster deployments and rollbacks.

"We tend to know how the infrastructure works and how to put in place workarounds. We either advise the AOD or take responsibility for specific tasks," Farrukh explained.

To learn more about DevOps communication in incident response, Farrukh recommends studying the examples of active, analog response systems like the Incident Command System developed for wildfire response in California and the U.S. National Transportation Safety Board.

As Craig Martin writes in The New Stack's CI/CD with Kubernetes, DevOps is a journey and not a destination. It means building cross-functional teams with common goals, aligning the organization around the architecture and creating a culture of continuous improvement. Hiring the right people to build your team, or training an existing team to communicate openly in a DevOps culture, will be key to maximizing cloud native technologies and unlocking the speed and agility your organization needs to bring developers closer to your customer needs.

Filling in the Dev and Ops Gap in Cloud Native Deployments

The move to cloud native application architectures is forcing organizations to rethink the development and operations roles. Traditional operations responsibilities are shifting left, and developers are picking up and owning the end-to-end application life cycle.

"Application development teams are — even inside existing organizations — taking incrementally more ownership of some of these pieces of their delivery," Luke Hoban, chief technology officer of Pulumi, said. "IT and operations organizations are actually pushing more of that self-service ownership down into development teams."

Consequently, there is often a "gap in how operators and DevOps kind of think about the way that they tackle their problems versus the way that application developers tackle some of their problems," Hoban said.

Developers are challenged by a lack of tools needed to port the code they create to the cloud, whether for Kubernetes, microservices or serverless platforms. It's as though the software engineering environment has regressed as part of the transition to the cloud. Operators also lack the tools they need to be really productive in a cloud native environment.

The first step in closing that gap involves allowing developers to continue using the tools and programming languages they are familiar with, while also allowing for more productive management of the infrastructure as code.

"Folks who are first coming into this world see these sorts of mountains of YAML and very deep and complex details of the underlying platform. When they're going to AWS, they've got huge cloud formation templates that they're trying to manage that describe how to get their software into an actual running environment inside the cloud," Hoban said. "And this becomes a really critical part of their application in terms of how they think about the application and its delivery to its end users."

Pulumi's tools allow developers and engineers to use real programming languages when deploying to cloud native platforms. This means supporting languages such as JavaScript, TypeScript, Python, Go and others.

"We let you bring real general-purpose programming languages to bear on how you author and maintain your infrastructure," Hoban said.

Listen on SoundCloud

__Luke Hoban__ is the CTO at Pulumi where he is re-imagining how developers program the cloud. Prior to Pulumi, Hoban led product definition and business planning for EC2 instance families at Amazon Web Services. At Microsoft, Hoban co-founded TypeScript, developed Go support for Visual Studio Code, was part of the design teams for ECMAScript and C#, and served as Technical Assistant to the CVP of the Developer Division contributing to corporate strategy and execution for the developer audience.

Bibliography

1. **"Microservices Security Strategy" in "Guide to Cloud Native Microservices"**
   by The New Stack, 2018.
   Building microservices provides scale and automated workflows that get implemented through small teams that each work on specific services. This ebook explores how teams build, deploy and manage these scaled-out application architectures with technologies that fit the organization's objectives.

2. **"What is Cloud Native Now?"**
   by Anne Currie, Chief Strategist at Container Solutions, Container Solutions blog, October 18, 2018.

   Currie writes that if your definition of cloud native DevOps depends on the cloud, the definition changes as frequently as the cloud does. Instead, a cloud native approach is flexible and fast-moving by taking a series of small, reversible and low-risk steps.

3. **"5 Steps for Better Interdepartment Communication"**
   by Chisara Nwabara, Research and Development Technical Program Manager at Pivotal, The New Stack, October 3, 2018.

   A customer-facing team was feeling pain with Pivotal's developers — it didn't take long before they realized this was not a software problem, but instead, it was a communication issue.

4. **"Cloud Native Apps Need to be Managed in a Completely New Way"**
   by Max Johnson, DevOps Engineer at Pypestream and former Holberton School student, The New Stack, January 11, 2019.

   DevOps is the new way of building and shipping applications in the cloud — but this is easier said than done. Automation is essential to help minimize human error.

5. **"Unit Testing: Time Consuming but Product Saving"**
by Jennifer Riggins, The New Stack, December 22, 2017.

Unit testing works best in conjunction with integration testing, but there are unique benefits from unit testing. This includes faster development, because typically you write the unit test even before you write the code and then test your code against said test. And it catches errors at the unit level earlier on, so the cost of fixing them is dramatically reduced.

6. **"2018 State of DevOps Report"**
by Alanna Brown, Andi Mann, Michael Stahnke and Nigel Kersten, and Puppet Labs and Splunk, 2018.

This annual report analyzed responses from over 3,000 technical professionals from around the world to reveal the five stages of DevOps evolution.

7. See #3, above.

8. **"Radical Transparency Can Reduce Bias — but Only If It's Done Right"**
by Francesca Gino, Professor at Harvard Business School, Harvard Business Review, October 10, 2017.

Bridgewater Associates is the largest hedge fund in the world, managing almost $160 billion. This review of founder Ray Dalio's book, "Principles,"

SPONSOR RESOURCE

- **Building Cloud Native Apps Painlessly — The Prescriptive Guide to Kubernetes and Jenkins X**
by CloudBees, 2019
In this paper, you'll read about how the future of modern application development can benefit from the powerful combination of Jenkins X and Kubernetes, providing developers a seamless way to automate their continuous integration (CI) and continuous delivery (CD) process.

explores his philosophy of "radical transparency."

9. **"DevOps Patterns" in "CI/CD with Kubernetes"**
by Rob Scott, Senior Site Reliability Engineer at ReactiveOps, The New Stack, 2018.

In this chapter, Scott covers how traditional DevOps looked before containers, how containers brought developers and operators together into the field of DevOps, the role a container orchestration tool like Kubernetes plays in the container ecosystem, and how Kubernetes resulted in a revolutionary transformation of the entire DevOps ecosystem.

10. See #9, above

11. See #9, above

12. **"GitOps: 'Git Push' All the Things"**
by Joab Jackson, The New Stack, May 11, 2018.

"GitOps" is an idea that generated some buzz at the KubeCon + CloudNativeCon EU conference in Copenhagen, Jackson writes. The basic idea is that all changes to a cloud native system can be done through Git. Once a commit is made, it sets off an automated pipeline.

13. **"Microsoft to Acquire GitHub to Expand Developer Reach"**
by Joab Jackson, The New Stack, June 4, 2018.

The New Stack's coverage of the GitHub acquisition highlights how central GitHub is to the developer experience.

SPONSOR RESOURCE

- **Delivering Cloud Native Infrastructure as Code**
by Pulumi, 2018
Find out how to deliver all cloud native infrastructure as code with a single consistent programming model in this white paper from Pulumi.

14. **“From GitOps to Adaptable CI/CD Patterns for Kubernetes At Scale”**
by Twain Taylor, The New Stack, November 15, 2018.

In this post, Taylor explains the benefits and drawbacks to GitOps for an enterprise operating at scale.

15. **“Pulumi Kubespy: Watch Kubernetes Deployments as They Happen”**
by Susan Hall, The New Stack, October 2, 2018.

Pulumi’s tool called kubespy displays changes made to a Kubernetes object in real time. It’s available on GitHub and can be used even if you’re not using Pulumi for deployments, to inspect the cluster and give information such as failure, network connectivity problems and more.

16. **“Google Reveals the Secrets of DevOps”**
The New Stack Makers podcast with Melody Meckfessel, Senior Director of Engineering at Google, and Sam Ramji, former Google Vice President, July 17, 2018.

A discussion of Google’s internal development processes and practices includes tips and tools they use for near-instant build processes and performing encrypted compute inside untrusted compute infrastructure.

SPONSOR RESOURCE

- **Continuous Delivery Summit**
  by Continuous Delivery Foundation, 2019
  The Continuous Delivery Foundation will be hosting a Continuous Delivery Summit (CDS) event on May 20 at KubeCon + CloudNativeCon Europe 2019 in Barcelona, Spain. The Cloud Native Computing Foundation’s flagship conference gathers adopters and technologists from leading open source and cloud native communities including Kubernetes, Prometheus, Helm and many others. Register to attend, today!

17. **"Continuous Integration and Delivery with Jenkins X"**
The New Stack Makers podcast with James Strachan, Chief Architect at CloudBees and Michael Neale, Co-founder and Engineering Manager at CloudBees, September 5, 2018.

In this podcast, Strachan and Neale discuss how Kubernetes has changed CI/CD, evolving workflows such as GitOps, and how CloudBees is working to improve the developer experience for application deployments on top of Kubernetes.

18. **"Continuous Intrusion: Why CI Tools are an Attacker's Best Friends"**
by Nikhil Mittal, Penetration Tester and InfoSec Expert, at Black Hat Europe, 2015.

In this talk, Mittal discusses insecure default configurations in continuous integration (CI) environments.

19. See #1, above.

20. **"Baking Compliance Into Your CI/CD Pipeline"**
by Scott Fitzpatrick, contributing writer for Twistlock, The New Stack, October 1, 2018.

Just like everything else in a CI/CD pipeline, compliance should be continual, and it should be baked into all stages of the delivery process.

SECTION 3

MANAGE

Once a cloud native application is deployed, DevOps best practices include next-generation monitoring, dashboards and feedback loops to help ensure success.

Contributors

Jennifer Riggins is a tech storyteller, content marketer and writer, where digital transformation meets culture, hopefully changing the world for a better place. Currently based in London, she writes around tech ethics, agility, accessibility, diversity and inclusion, testing DevOps, happiness at work, API strategy, the Internet of Things, microservices and containers, developer relations, and more.

Mary Branscombe is a freelance technology journalist with over two decades of experience. She writes regularly on technology for CIO.com, TechRadar, ZDNet, Hard Copy, The New Stack and a wide range of other titles. She has also co-written two books on Windows 8 with her writing partner, Simon Bisson, and has published some short mysteries about the world of tech on Amazon Kindle.

Mike Melanson is a freelance writer, editor, and all-around techie wordsmith. Mike has written for publications such as ReadWriteWeb, Venturebeat, and ProgrammableWeb. His first computer was a “portable” suitcase Compaq and he remembers 1200 baud quite clearly.

Rob Scott works out of his home in Chattanooga as a Site Reliability Engineer for ReactiveOps. He helps build and maintain highly scalable, Kubernetes-based infrastructure for multiple clients. He’s been working with Kubernetes since 2016, contributing to the official documentation along the way. When he’s not building world-class infrastructure, Rob likes spending time with his family, exploring the outdoors, and giving talks on all things Kubernetes.

CHAPTER 07

Creating Successful Feedback Loops With KPIs and Dashboards

It wasn’t that long ago that software updates came in intervals of months and years. Big companies built big pieces of software and if anything went wrong from one version to the next, it was just something the user had to deal with, while teams of developers went through monolithic piles of code to troubleshoot where they went wrong along the way. Software bugs aside, even unpopular features and customer complaints were met with months, if not years, of wait time before new features could be released to address the issue.

Nowadays, by comparison, companies like Facebook and Google deliver software updates on the order of hundreds, if not thousands, of times per day across their numerous products. Feedback loops between customers and product teams have shrunk from years and months to days, hours and minutes.

At the core of this change is the DevOps approach to software development, which relies on closely aligning developers with operations (Ops) teams to create feedback loops that help to shorten the time between bugs and their fixes, feature requests and their release and changed goals and their completion. The DevOps approach, however, is not some mystical or out-of-reach practice relegated only to enterprises with deep pockets. Instead, with the confluence of many features of modern software design and deployment, most any company can use DevOps methodologies to create ever-tighter feedback loops in order to increase customer satisfaction and overall success.

First, let's take a look at how we got to where we are today, what it means and how we can use these conditions to better serve our customers and ourselves.

A Natural Progression from Agile

A key change in software development occurred around the turn of the century upon the introduction of a new methodology: agile. Before agile, companies used the waterfall methodology, wherein software was carefully planned from the outset and then implemented according to that rigid plan. With agile, instead of careful planning from the beginning, developers code in short iterations called sprints. During each sprint — usually a week or two at most — developers work to hit certain goals and then deploy their work. Upon deployment, they measure their success, re-evaluate their goals and adjust from there. DevOps is merely a natural progression of several environmental conditions coalescing with the agile methodology, explains Brian Dawson, DevOps evangelist at CloudBees.

"DevOps is a practice that is largely focused on culture and emphasizes the alignment of developers and operations around the shared objective of releasing software repeatedly, reliably and rapidly," Dawson said.

Over time, he explains, we moved away from boxes of compact disks (CDs) used to install software to constantly connected devices that updated software in the background. Software was no longer installed on the user's machine, but instead run as Software as a Service (SaaS), that could be updated and deployed in the cloud and immediately provided in rolling waves to millions of users. And at the same time, technologies like Infrastructure as a Service (IaaS) and containerization made it possible to quickly spin up servers in a matter of minutes and seconds instead of hours and days. With all of these conditions, the expectation for constant updates not only appeared, but the ability to deliver on that promise also became available to software teams both large and small, with continuous integration and continuous delivery (CI/CD) systems.

“With SaaS and PaaS [Platform as a Service], we don’t need to wait on Ops teams anymore to stand up a server,” Dawson said. “The cloud provided developers and QA [quality assurance] teams rapid access to compute resources and infrastructure in a way that revolutionized things.”

Throughout this evolution, teams have adapted their workflows, which really is the DevOps way. The approach means that developers will always be looking for platforms and tools to get deeper into the code, looking for better optimization to make the infrastructure essentially invisible so that they may adapt application architectures more effectively. This means a new generation of platform tools that orchestrate the software architectures and the data plane to allow more informed and automated approaches. The next wave of innovation is now emerging with serverless technologies — part of a narrative that will unfold with new practices that use frameworks for managing streaming architectures for various workloads. Increasingly, workloads are using Kubernetes and automated platforms that may label themselves as serverless. HashiCorp, for example, is viewed as a collection of software tools that allow for scale and is continuing to find a real market for its services. Developers are widely supportive of HashiCorp and how it manages open source communities.

Demystifying DevOps

Much like agile, using the DevOps methodology isn’t as simple as hiring someone or implementing a solution. Instead, DevOps is an approach to software development that starts at ideation and continues through to deployment, using monitoring along the way to identify where improvements can be made.

"A clear and unifying measure of DevOps is to show how work ultimately impacts the customer and the business," explains Ravi Tharisayi, director of solutions strategy at New Relic. "Whether you've adopted containers, deployed two times faster, or migrated to the public cloud, if the results can't be measurably felt by the customer and the business, it's hard to argue success."

In other words, deploying new technologies and adopting new methods is only as good as the final result. It can be easy to get wrapped up in meeting abstract goals when employing a new methodology, but the ultimate goal of DevOps is still to better serve your customers.

"While increased velocity is a huge imperative for organizations today," Tharisayi further explains, "that velocity can be useless or even counterproductive without the right data to direct that velocity to the right problem with measurable results for everyone to see."

So what does all of this mean for the modern software development team looking to increase performance through a cloud native DevOps approach? Teams need reliable feedback in a way that's easy to understand, access and act upon. Through emerging observability practices, cross-functional teams can set key performance indicators (KPIs), track progress against them through monitoring and dashboards, and adjust course based on this feedback.

Observability is the evolution of monitoring for the cloud native era. It gives engineers the information they need to adapt systems and application architectures to be more stable and resilient. This, in turn, provides a feedback loop to developers which allows for fast iteration and adaptation to changing market conditions and customer needs. Without this data and feedback, developers are flying blind and are more likely to break things. With data in their hands, developers can move faster and with more confidence. [1] Such data can also be used to unify Dev, Ops and management around common goals and establish a definition of success for DevOps initiatives themselves.

The New Stack's "CI/CD with Kubernetes" ebook has an in-depth discussion of modern observability practices.

Feedback Loops and Key KPIs for Success

While DevOps KPIs can vary from company to company, or even project to project, certain ones do appear to be considered a sort of industry standard. For example, the 2018 DevOps Research & Assessment (DORA) report cites five metrics as key indicators of success. The group has tracked deployment frequency, lead time for changes, time to restore service, change failure rate and overall product availability, over the last five years. According to the report, the companies that perform well on those five indicators "do significantly less manual work across all vectors, spend more time doing new work, and spend less time remediating security issues or defects than their low-performing counterparts."

Meanwhile, a CircleCI report on "Three Critical Development Metrics for Engineering Velocity" similarly finds that stability, commit-to-deploy time (CDT) and deploy frequency are "the best predictors of an organization's velocity and growth." [2] Deploy frequency indicates a company's actual speed, and is also a function of stability and deploy time: high mainline branch stability and low deploy time both encourage higher deploy frequency. [3]

"We look at deploy frequency as a 'vital sign' of an organization's DevOps health. With each 'pulse,' or deployment, an organization is delivering value, discovering customer needs and fixing problems," writes Jim Rose, CEO of CircleCI on The New Stack.

CircleCI's report, which was based on a sample of GitHub and Bitbucket organizations built on CircleCI's cloud platform in mid-2017, found that 75 percent of all organizations deploy their most active project less than 13 times a week. Top performers (those in the 95th percentile) deploy their mainline branch 32 times per week — over five times the median and nearly 24 times the bottom 5th percentile. For organizations in the top 10th percentile of Alexa Internet Ranked organizations, they saw the 95th percentile deploying 42 times per week.

Whatever initial measures you choose to observe, however, they are just the beginning. Tharisayi cautions that the key to DevOps is to customize your monitoring and tighten your feedback loops to focus on the key ingredients of your particular success.

“KPIs on a dashboard are certainly important for building a shared understanding, among other uses,” Tharisayi says, “but we see high performing teams diving deep into data — filtering, faceting, and iteratively exploring — to understand what to focus on and where reliability issues lurk.”

In addition to the basic KPIs used in the aforementioned reports, Dawson offered numerous others that could be used to dive deeper into measuring success, from code complexity to commit frequency to customer satisfaction. KPIs can extend beyond the basics of the CI/CD pipeline to help determine what adjustments may be required.

“It is really about visibility,” Dawson says. “We have a new level of visibility and insight into how we are delivering software — and a new ability to share and compare those insights — that provides us a new level of clarity when it comes to optimizing what we do.”

A dashboard is a display of those KPIs that’s readily visible and that shows how something is functioning or progressing. When used right — whether on a car, aircraft or a DevOps laptop — a dashboard is simple to read at a glance and a very powerful way to know you’re heading in the right direction.

In 2014, IBM developer advocate Steve Poole took over leadership of a more than 100-person, somewhat distributed, European “Slow IT” team, as it moved toward the much faster cloud. They moved from a two-year shipping cycle to

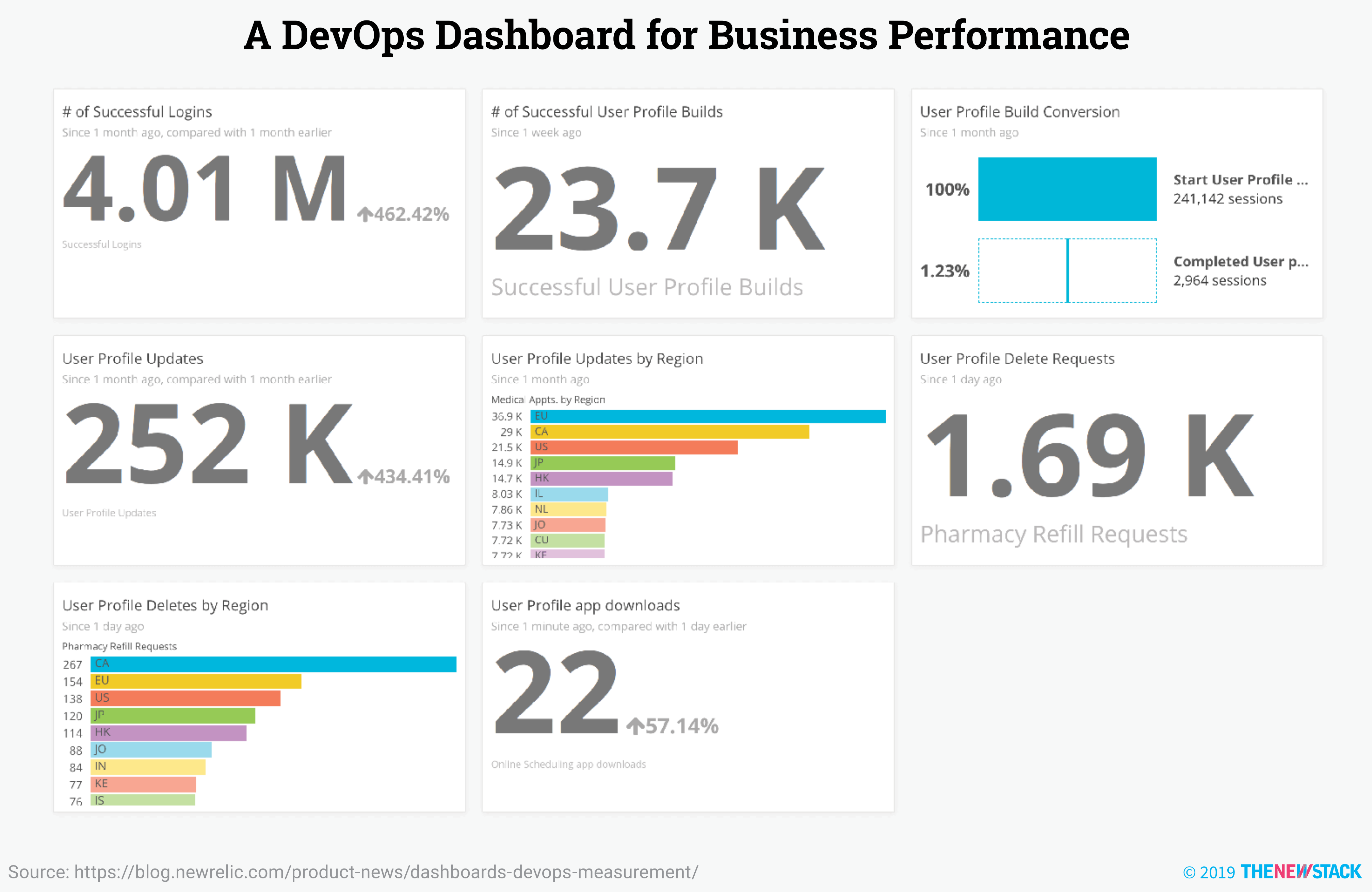


FIG 7.1: *A management dashboard gives technical and non-technical teams a way to assess an application's business performance at a glance.*

supporting SaaS products that shipped daily. He shared how dashboards helped close the communication gap among siloed teams with the Agile Tour London conference and in a follow-up interview with The New Stack.

Using Dashboards to Transform Team Behavior

Poole's first challenge was to take the operations team that was set up to run "traditional operations with traditional deadlines and time" and to convert them to a new 24/7 release cycle. They needed to understand the value of what was happening.

> "We had to get people from the old way to a new way of thinking. Dashboarding became a major aid to making that happen."
>
> **— Steve Poole, developer advocate, IBM.**

“I learned that IT and Dev don’t communicate. They just shout at each other in their own language,” Poole said, because they don’t share a common experience or perspective.

“IT teams tend to group themselves around the experts so you see a lot of mini silos who are responsible for parts of it. And the team that I took over had lots of small teams.”

Poole then pointed out how even tech conferences are usually very siloed around certain languages, methodologies or roles. He saw a need for a new “DevOps contract” between developers and operations that included self-service assets like Platform as a Service and Infrastructure as a Service, containers and testing, and it needed to cover new availability requirements with an emphasis on speed to market and feedback loops.

This shared sense of purpose is illustrated and translated by dashboards. Poole compares a digital transformation dashboard as similar to a car dashboard which is very visual with:

- Just enough status.
- Just enough insight.
- Just enough warnings.
- Clear emergency indicators.

Engineers should be able to see that something is wrong at a glance. Clear green and red notifications can totally change reaction speed.

Poole’s area of IBM is simply littered with screens broadcasting information on dashboards everywhere, leveraging Raspberry Pis to create a network of monitors. Within a year, everyone — every Dev team, support team, manager and executive — had a dashboard, broadcasting more than 4 million events per day.

“You stop looking at your email and reports in your system, and you start looking at dashboards," Poole said. "Prior to that, Dev teams would do the work that they are planning and then the management team would come in and say 'We have a bug, who's going to fix that?' Now, it's a pager look for the whole team" — everyone on call with the same view.

One useful tool is Dashing, a framework created by Shopify, to create simple dashboard widgets. Dashboards can be segmented according to teams, and even shared with customers and the public. IBM created about five different groups of dashboards, one for:

- Each development team.
- Each support team.
- Management and executives.
- Operations.
- Unusual and quirky dashboards.

All of the dashboards had one thing in common: to make sure the right people were the first to know if something went wrong.

Middle Management and Executive Dashboards

It can be difficult to measure the immediate success of DevOps transformations, since sometimes the process leads to a slow-down, missed budgets or both. Giving something for both middle management and C-level executives to measure helps keep them committed to the often disruptive change.

Middle management that's focused on IT needs what Poole called "reactor dashboards," that help flag what they actually have to deal with. For middle and upper management, dashboards need to help them think about actions that can be taken. Middle management dashboards at IBM can include setting a bar on the number of bugs in the backlog or measuring incoming rates.

While middle management dashboards are all about seeing where intervention is necessary, executives are desperate for more accurate data to guide which levers to pull to redirect the organization.

For example, one of the things Dev teams were told to do is make better use of cloud capacity, moving from on-premises infrastructure. Poole says to do that, you have to understand your current on-premises status. With dashboards, the large team went from numbers to a progress report, a line that goes up when cloud is gaining and local is going down, with a projection for the future. Of course, they needed to make some adjustments in order to make a direct comparison. They classified servers by the type of workloads and on-premises workloads in terms of cloud characteristics, such as central processing units (CPUs), memory and multitenancy to bare metal.

With such visual displays, managers were able to justify budget increases more effectively by showing how their existing capacity is being used. They could also better predict the overall cost of the move to the cloud.

Everything had to be handcrafted for the executive dashboards, but then, since it's DevOps, the information retrieval was automated.

"When you come up with a common vision, you chuck all the titles away and say, 'Let's just sit down and see what you want to do and what you want to get out of it.' You can educate the exec in the realities of the challenges and [they] can come back and say 'I accept that' or 'I don't.' The creation of the dashboard [makes it] a two-way activity," Poole said.

Like all dashboards and big data, a big challenge to DevOps automation is cleaning up inaccurate, unrealistic and stale data. It took Poole's team about a month of conversations to understand what data was coming out of thousands of machines, understanding purposes, use cases and the size of said data. Only then could the data be cleaned, bringing usefulness to the dashboards.

Support, Developer and Operations Dashboards

One bonus of using the dashboard in the move from on premises to the cloud was that IT support also gained an improved understanding of what their clients were using their machines for.

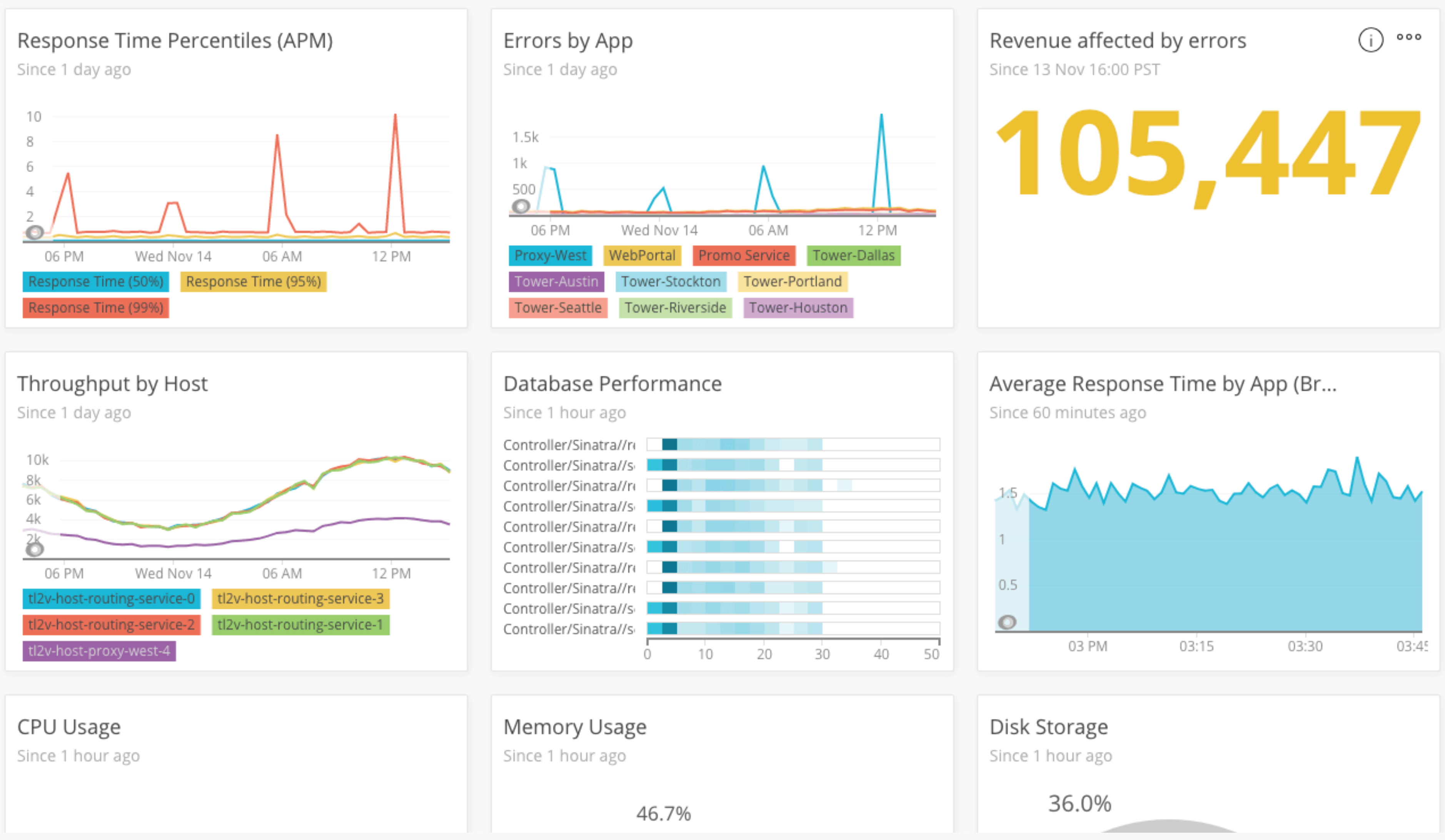
> “My IT team was reactive and I really hate that. I want my teams to be going out to customers and be the first to say: ‘You’ve got a problem.’”
>
> **— Steve Poole, developer advocate, IBM.**

If you really want to get Devs, Ops and support on a united front, you need to offer transparent feedback and let them visibly see they are on a path together toward shipping better code. It’s all about creating actionable insights by making patterns easier to see in a graph. In one case Poole’s team was trying to figure out how to get to a reasonable delivery every time. Using a dashboard, they charted out how many lines of code they were shipping versus the level of complication in that code. If there was a certain combination of quantity and complexity, they determined that it should be delayed from being shipped.

After initial success creating graphs, they followed this mindset with all the IT dashboards, improving ticketing dashboards and measuring things like “How long to close 80 percent of the tickets?” They also used dashboards to highlight the different processes each team uses.

Once dashboards become useful within teams, it makes sense to begin sharing them with end users as well. Sharing teams’ initial response times and time-to-resolution figures with end users — whether those are the Devs being served by Ops or sometimes even external customers — can help everyone align around common goals and expectations.

Application Performance Dashboard



Source: https://blog.newrelic.com/product-news/dashboards-devops-measurement/

FIG 7.2: *Developer and operations teams should start the day by level-setting from the same application performance dashboard, according to New Relic.*

"To make that real, you have to have a conversation with your end user [about] what it means to be down. What does red mean? Depending on what service that you're looking at it can be different." Poole continued, "What we taught the IT teams is to understand what it really meant for a service to be available, so when it was unavailable it really meant unavailable."

Dashboards Don't Change People, but They Can Reflect Real Change

For Poole, the biggest challenge his team overcame was reaching agreement between those using the data and those that are consuming the data from the dashboards. He described it as a constant negotiation, with people on both sides having to learn how the other side operates.

"You're washing your dirty linen: 'You do that? Why do you do that?'

Sometimes it's a fair question and sometimes it's just not understanding," he explained.

But once this dashboard infrastructure was up and running, the value was clear.

He clarified that none of these dashboards are tied to business goals or budgets, but rather service availability, developer productivity and development statuses.

"We don't really need to put it down into dollars, it's all about progress and status," Poole said.

Can dashboards really change culture? Of course not, but they act as a good mirror to see if the culture has embraced transparency, autonomy and automation.

"The important thing about this is it changed how people behaved. You go from a team to where the body language is very closed, they don't want to talk, and they don't want to share because they are afraid they're being told off," Poole said.

Once they understood their customers loved what they were doing, he said, their behavior changed. "We went from very negative teams to understanding they were valued."

CHAPTER 08

Testing in Cloud Native DevOps

As DevOps teams strive for continuous integration and delivery, continuous testing becomes a bigger part of the whole transformation toward stability among rapid change. Before the agile-driven practices of DevOps and extreme programming, testing teams used to receive the code at the end of the development process, tossed over the proverbial wall. Now developers not only perform their own tests, but testers are included on development and infrastructure teams.

The end goal is no longer for testers to check code after it's written, but rather to play a more strategic role in an organization looking for speed through automation and responsiveness to production data. Testing code early and often, in an automated and parallel fashion — called continuous testing — eases testing bottlenecks, allows bugs to be found and fixed earlier and helps speed code to production, writes Lubos Parobek, vice president of product at Sauce Labs.[4] It helps provide security and compliance checkpoints, improving the quality of code in production. And it ensures new application features have their intended effect for customers, before such features are rolled out across an entire customer base. Testing is now considered as part of the planning and architecture discussion, before a single line of code is written, as well as integrated throughout the entire software development life cycle (SDLC). Tests are optimized for each product depending on the specific goals determined by business teams, product managers, Devs and Ops. Continuous testing, as well as testing in production, are becoming best practices for cloud native applications.

Integration testing for microservices-based applications, for example, has been a real challenge for organizations, because it's nearly impossible to replicate all of the services and integrations an application needs in production, in a test environment. Most organizations are experimenting with new methodologies such as feature flags and canary deployments, as well as developing new roles and tooling to better accommodate continuous testing.

"What I see in most enterprise customers is people trying to bring some level of control into their pipelines," said Alex Martins, chief technology officer for continuous quality testing at CA Technologies. "The low hanging fruit towards that is to use service virtualization as one technique that is quick and dirty to isolate what you're testing. If you're testing on premises and it needs some services in the cloud, then you isolate the services and don't worry about the cloud."

But primarily, this integration testing problem will be solved by management and process rather than tools, Martins said. "I don't think we need more tooling. We need a different way to manage production. ... It's really more of a mindset change, and a process and culture change."

It's a DevOps approach, applied to testing. Like other DevOps processes, testing is being increasingly automated as part of the CI/CD pipeline. And QA and testing job roles have been adapting to agile development practices. Embedded testing is an emerging role that digitally driven companies such as Facebook and Moo are experimenting with as they look to strengthen software in a way which is distinct from site reliability engineering.

Embedded Testing at Moo

Abby Bangser joined online print and design company Moo and its 100-person tech department about two months ago as a senior test engineer on the platform team, an unusual role unto itself. Moo embraces the DevOps ethos of "product not project," with a test engineer, a product owner and an agile coach active on each tech team.

Test engineers are embedded within tech teams in order to break down silos, Bangser said. Her job isn't about testing for the developers, but rather to help the team identify what quality means for each service, by identifying suitable requirements and using testing tools to support them.

"Our engineers are in charge of leveraging tools [for] their own monitoring, writing their own service-level alerts, running their own pipelines including through to production, deployment and monitoring," she said.

Bangser looks at her role on the platform team — which she says combines platform engineering, operations, and testing — as a good alternative to a site reliability engineer (SRE) for smaller development teams. She says her role focuses on infrastructure and provisioning, shared resources and the observability of these services.

"Coming from a software development testing background, I'm a bit of a different profile than most platform team members," Bangser said. "It's a profile that's going to benefit from being exposed more to deployment and infrastructure work. I believe that software testers can [bring the] ability to identify the use case and the value of the feature."

> "Software testers identify the use case and value of a feature."
>
> **— Abby Bangser, senior test engineer, MOO.**

Bangser continued that her role involves a lot of requirement and user story analysis. She says this also helps support developer experience.

"You focus on user experience. I hear Devs talk about all the time that 'they don't need UX because it's an API.' But if you are building software, you have users — and as a platform team we absolutely are aware of our users — and they [the users] are the dev teams," she said.

She says the test engineer role helps the whole team focus on the impact of any change to the end users, which means making sure documentation, training, feature prioritization and update communication all stay front of mind.

Production Engineering at Facebook

Evan Snyder has been a production engineer at Facebook for four years now, the last two of which have been spent in the testing infrastructure group. She says a production engineer is sort of a DevOps role, like an SRE, but, similar to test engineers at Moo, it's more integrated with the Dev team, even writing product roadmaps together.

While the developers are responsible for writing their own tests, Snyder said, "teams don't have to reinvent the wheel to run and track their tests at the right times. The testing infrastructure team builds shared systems for test selection, execution, triage and reporting to support the full life cycle of tests once they are written."

"A DevOps approach allows us to ship high-quality code, so running the right tests at the right time is essential as it helps us ensure things are safe before we deploy. We have a large suite of tests that make realistic requests to our applications, verifying that functionalities behave as expected," she said.

At Facebook, each product team has three types of tests that are running continuously on any change that will be committed to Facebook's master repository:

- **Unit tests:** very small and targeted tests. They write a lot of these cheaper tests early on.
- **Integration tests:** larger chunks of dependent code.
- **End-to-end tests:** more expensive and longer to run.

Snyder said this is all done with the goal of signaling any anomalies as early as possible to developers: "Either your test is broken by this change or something is broken in the master [repo] so please come and see what's going on."

Unsurprisingly, the Facebook suite of apps, including Messenger, WhatsApp and Instagram, has a heavier load to bear than almost any service out there, which means any sort of break can impact a large number of people. They need test coverage at scale.

Automated Testing at Scale

While testing roles and processes are adjusting for new cloud native architectures and becoming more automated, the majority of tests are still done manually. [5] Only 16 percent of performance test cases are executed with test automation tools, and security tests are being completed at the same frequency according to the World Quality Report (WQR) 2018-2019, which surveyed 1,700 IT decision makers (ITDMs) at companies with more than 1,000 employees. Expect testing to become increasingly automated in coming years, and for testing roles to continue to evolve with CI/CD workflows and best practices.

There is agreement that testing can't be completely automated. Part of this is because humans are still needed, but there are other obstacles as well. When asked about test automation challenges, 61 percent of WQR respondents said their applications change too much with every release. Thus, while release frequency has increased due to release and management automation, testing may be a chokepoint for future SDLC automation and an area for continued improvement.

At scale, automation becomes critical for handling the level of testing each application requires, alongside the efficient use of testing resources. Facebook and its other applications have a greater need than many to test across environments due to scale. For example, it must test different versions of the application for specific operating systems: iOS, Android and KaiOS. This is why Facebook created a resource pool for tests.

One World is an internal tool at Facebook that provides libraries and infrastructure to expose runtimes to services and engineers through a common API. "Productionized" examples of runtimes include emulators, simulators, web browsers or devices. To illustrate the scale at which One World operates, Facebook runs millions of tests daily through these existing deployments.

One World is used when a product developer would like to test the build of an Android application but doesn't want to set up a complicated Android emulator. Or the tool can be used when someone wants to try something on Windows but doesn't have a Windows machine; they can use a virtual machine to test it. Facebook even has a physical lab for testing devices like mobile phones.

Everything they are working with needs to focus on answering the same set of questions:

- How can I set it up to be ready to run a test?
- How can that test communicate with the client?
- How can I clean it all up afterwards to be sure the health of the resource is good?

One World includes hardware management, a mobile device lab, performance testing, functionality testing and emulated correctness testing — does clicking this, do that? Snyder says that performance testing has to happen on the actual hardware, which makes it more challenging.

“The goal of the production engineering team at Facebook is resource efficiency and rational testing strategy.”
— Evan Snyder, production engineer, Facebook.

Developers learn from production engineers how to evaluate the trade-offs and the cost — time and capacity resources — of executing and running tests.

“A goal of all of our teams is to use resources as efficiently as possible — hardware and computing,” Snyder said. “If you are executing every single test on a bunch of servers, if there are a lot of pull requests coming at once, can we batch them up together?”

Facebook is working to “do more with less at every layer of testing. Infinitely running tests won’t scale forever,” Snyder said.

How to Approach Testing in Production

Matt Swann, former chief technical officer (CTO) at StubHub, said that testing in production is part of his company’s daily routine. Their testing philosophy is based on the importance of always keeping the customer at the center. [6]

“As technologists, we can fall into the trap of assuming that we know what is best for the product, but that kind of thinking can be dangerous. Only by getting customer feedback and by falling in love with the customer’s problems will we be able to create the space for innovation and build remarkable customer experiences,” Swann writes on The New Stack.

Moving a team from standard testing to testing in production requires clear communication of the new process and helping teams understand the roles and benefits, he said. Start small and collect some wins. Start with one or two teams where you invest time training and support. Collect evidence from the wins and learnings and adapt the process to fit the organization. As your initial teams begin to thrive, roll out the changes to other teams.

StubHub teams think about testing and learning in two distinct ways:

- **Pre-build hypothesis testing:** Implements a process based on lean UX, focusing on obtaining feedback as early in the cycle as possible, much of which will happen "pre-build."
- **Lightweight evidence-based optimization:** Uses a tool such as Optimizely, where teams can experiment both on the frontend and with server side modifications.

Each of these processes has the following elements: observation, hypothesis, experiment parameters, metrics to determine how hypotheses were right or wrong, and next steps: what to do if hypotheses are right, and what to do if they are wrong.

The important part is making sure that the team is focused on getting "signals" and learning as quickly as possible, Swann said. Rather than implementing large, time-consuming features and then learning — large effort with high risk — the team needs to get data as early as possible to learn whether the hypothesis is correct — small efforts with small risk. When you are right, keep going and perform another test to learn more. By keeping tests small and focused you minimize the overall risk to the company.

CHAPTER 09

Effective Monitoring in a Cloud Native World

The transition to new cloud native technologies like Kubernetes has dramatically altered how applications are architected and deployed. For quite some time, monolithic applications were deployed to a set of long-living servers, receiving rather infrequent and incremental updates over their lifespan. With cloud native technology has come a rise in microservice architecture running with ephemeral containers on infrastructure that is quickly evolving. Adding to the complexity, these applications are regularly deployed to multiple availability zones, regions or even multiple clouds.

This approach to application architecture has come with many advantages, but it's also required significant shifts in supporting technologies like monitoring. Cloud native systems are generally more stable and highly available, often including some kind of auto failover components. Unfortunately as these architectures get more complex, so do the potential failure modes. The more components that are involved, the more ways things can break. With that in mind, it's more important than ever to have an effective monitoring strategy.

In a cloud native world, the traditional monitoring we've become accustomed to simply can't keep up with these more modern architectures. It can't provide the level of insight into our applications that we need to fully understand what's happening.

"The volume of data in this containerized and elastic environment is beyond human comprehension and exceeds the capacity of spreadsheets and traditional analysis tools. Pattern recognition, correlation analysis, and anomaly detection can help DevOps teams identify early warning signs, find related transactions, and often fix emerging problems before they are reported by users," writes Gayle Levin, director of solutions marketing at Riverbed Technologies. [7]

The most effective monitoring strategies will take a multifaceted approach, covering four different areas: External polling, centralized logging, custom metric collection and request tracing. Although not every architecture requires each of these components, each of them can provide a unique and complementary level of insight. The best monitoring strategies rely on a combination of approaches to provide a comprehensive system overview.

External Polling Provides High-Level Visibility

There's a broad category of monitoring often referred to as "black box" monitoring. This refers to polling a system from the outside-in to measure its health. An example of this would be polling a web endpoint every minute to ensure the uptime of your application. One of the most traditional forms of monitoring there is, black box monitoring likely still has a place in a monitoring strategy. This approach to monitoring is highly effective at detecting problems that are already visible to users.

In contrast to black box monitoring, many of the newer approaches are referred to as "white box" monitoring, or observability practices. This involves monitoring from the inside out, and can provide a level of insight that black box monitoring generally can't. This approach can often detect problems before they become externally visible, and can provide valuable information for in-depth debugging. Each of the approaches in the following sections is a form of white box monitoring.

Cloud native monitoring is not just a matter of gathering as much data as possible, but collecting meaningful data that adds to an understanding of the behavior. Visualizing data and metrics, and tracing events as they flow from one component to the next, is now a reality of monitoring microservices environments. If black box monitoring is about watching the state of the system over time, then white box monitoring or observability is more broadly about gaining insight into why a system behaves in a certain way. [8]

Centralized Logging Provides Valuable Debugging Data

Logging is nothing new. Just like polling, it's been around for quite some time. Cloud native architectures don't just require local logging though, they really need some kind of centralized logging system. With traditional monolithic architecture running on long-lived servers, logs sometimes never left the server they originated from. Centralized logging was not always seen as a necessity. Debugging sometimes meant logging into a specific server and sifting through logs to find a problem.

With cloud native infrastructure, containers and servers are ephemeral, and it becomes more important than ever to ship logs to some kind of centralized logging system. Components in a cloud native system also tend to change more often with the increased use of continuous delivery (CD) techniques. [9] Elasticsearch, often deployed with Logstash and Kibana to make up an "ELK" stack, has become one of the most popular open source solutions for centralized logging. The components of an ELK stack, also known as the Elastic Stack, combine to provide a very compelling set of open source tools that simplify log storage, collection and visualization respectively.

Having all system and application logs in a single place can be an incredibly powerful component of your monitoring system. When things go wrong, centralized logging allows you to quickly see everything happening in your system at that point in time, and filter through logs for specific applications, labels or messages. Taking this idea further, Grafana Labs has released an open source log aggregation tool for Kubernetes called Loki, which indexes the log metadata, allowing for queries on where the logs came from, when they were generated, on what host and what version of the software. [10]

Additionally, these centralized logging systems can be configured to alert for anomalous behavior — a critical tool for those taking a DevOps approach to security, otherwise known as DevSecOps. Anomalies could be as simple as significantly increased log volume, or potentially an unexpected influx of error messages coming through.

"Machine learning can help you find an anomalous and malicious activity, and automate workflows so operations teams and developers can address the issues faster. Traditional security approaches, by comparison, can't keep up with the speed or scale of these new architectures," writes Rohan Tandon, a statistician and member of the technical staff at StackRox. [11]

The increased number of components that must be monitored in a microservices-based application vastly increases the amount of data and metrics being logged. This in turn increases demand for storage and processing capacity when analyzing these data and metrics. Both of these challenges lead to the use of time-series databases, which are especially equipped to store data that is indexed by timestamps. The use of these databases decreases processing times and allows for quicker results. [12]

Custom Metrics Enable Fine Grained Reporting

To fully understand your systems, the base set of traditional metrics often don't cut it. With custom metric collection, applications can expose metrics that are a better measure of application health. These kind of metrics can provide much more precise information than the kind of metrics derived from polling data from outside of the system.

Open source tools like Prometheus have transformed this space. At its core, Prometheus is a monitoring and alerting toolkit that stores metrics with a multidimensional time series database. Each time series is identified by a key-value pair, and tracks the value of that metric over time. The simplicity of this model enables the efficient collection of a wide variety of metrics.

Prometheus has become especially popular in the cloud native ecosystem, with great Kubernetes integration. The ease of tracking new metrics with Prometheus has resulted in many applications exposing a wide variety of custom metrics for collection. These are usually well beyond the standard resource utilization metrics we'd traditionally think of when it comes to monitoring. As an example of what this could look like, the popular Kubernetes nginx-ingress project exposes metrics such as upstream latency, process connections, request duration, and request size. When Prometheus is running in the same cluster, it can easily collect the metrics exposed by the many applications like nginx-ingress that support Prometheus out of the box.

In addition to all the tools that have Prometheus support built in, it's rather straightforward to export custom metrics for your own application. Having these kinds of custom metrics monitored for your application can provide a great deal of insight into how your application is running, along with exposing any potential problems before they become more outwardly visible.

Request Tracing Provides End-to-End Visibility

With cloud native architectures, requests often end up triggering a series of additional requests to supporting microservices. When looking at an individual request, it is helpful to see all the related requests to other microservices. Traditional monitoring solutions didn't have a great way to find this information. This led to a new form of monitoring, request tracing, a means of connecting all related requests together for better system visibility.

> “In the microservices world, distributed tracing is slowly becoming the most important tool for debugging and understanding your application dependencies.”
> — **Neeraj Podar, platform lead, Aspen Mesh.** [13]

There are some great open source tools focused on request tracing, including Jaeger and Zipkin. These tools allow you to see detailed information about all requests that spawned from an initial request, providing end-to-end visibility across your microservices. This kind of insight can be invaluable when trying to diagnose any bottlenecks in your systems.

Automated Monitoring with Artificial Intelligence and Machine Learning

Managing infrastructure is a complex problem with a massive amount of signals and many actions that can be taken in response; that’s the classic definition of a situation where artificial intelligence (AI) and machine learning (ML) can help. Adoption of MLOps or AIOps — as Gartner has christened this trend — has been slow, perhaps because making the most of them requires automation to apply the recommendations, and, at least in part, because IT is naturally conservative due to the need to ensure availability. Silos between IT teams, like separating service management and performance management, also makes it hard to gather all the necessary data for effective machine learning. But the potential is significant and interest is growing.

It’s not just that AIOps can help with availability and performance monitoring, event correlation and analysis, IT service management, help desk and customer support, and infrastructure automation. It’s also part of the general ‘shift left’ DevOps trend where operations become an integrated part of application development and delivery. That means becoming increasingly responsive and proactive, but it also means improving the communications and coordination between teams, and connecting the data silos. With more applications to operationalize, monitor and support, and more of these using microservices and containers and cloud services that multiply the amount of infrastructure that needs attention, machine learning is becoming a key tool in keeping up.

IT and operations is a natural home for machine learning and data science. If there isn't a data science team in your organization, the IT team will often become the "center of excellence," said Vivek Bhalla, who was until recently a Gartner research director covering AIOps and is now director of product management at Moogsoft.

By 2022, Gartner predicts, 40 percent of all large enterprises will use machine learning to support, or even partly replace, monitoring, service desk and automation processes. That's just starting to happen in smaller numbers.

In a recent Gartner survey, the most popular use of AI in IT and operations is analyzing big data (18 percent) and chatbots for IT service management — 15 percent are already using chatbots and a further 30 percent plan to do so by the end of 2019. Other uses for AI include predictive analytics to prevent failure, application performance management, network monitoring and diagnostics and optimizing workload placement in the public cloud. Improving root cause analysis was the second most popular planned use at 40 percent (after the 42 percent planning less specific big data analysis). A third also plan to use AI for general IT and operations optimization and intelligent automation.

"Look at the repetitive, low-level tasks that are ripe for automation to free up the time of operations staff, lowering their stress levels and letting them use that extra bandwidth to work smarter," Bhalla said at Moogsoft's AIOps Symposium.

Increasingly, that's being done with off-the-shelf solutions rather than "homegrown" implementations, using tools like Logstash and Elastic Stack Features. These products are maturing from bandwidth-looking analysis and visualization using averaged-out data, to more real-time approaches using streaming and wire data as well as stored logs. Log analysis tools, like Splunk and Sumo Logic, have been adding machine learning options for extracting patterns and anomalies from historical data to go alongside metrics and visualizations of real-time service health with alerts when anomalies are detected, and automation options. Micro Focus' Operations Bridge adds anomaly detection and clustering of related alerts to IT monitoring, and Sematext Cloud does anomaly detection with machine learning from performance metrics and logs. Similarly, tools like Moogsoft AIOps and OpsRamp OpsQ started with automated real-time pattern discovery and are extending that to stored historical data.

Visualization and statistical analysis of historical data are what Gartner views as a reactive approach; you can look back and understand what has happened using machine learning, either for general performance understanding or for root cause analysis. As you move to the combination of historical and live data with machine learning and causal analytics, operations teams can become more proactive with predictive warning systems. If AI-powered systems are going to predict problems and even automate fixes, they need to do more than spot patterns; they need to understand them. For now, simply detecting which alerts and errors come from the same event can be very valuable, reducing the flood of noise to something useful.

"IT systems generate vast quantities of self-describing data but the data streams generated tend to be highly redundant," Will Cappelli, chief technology officer at Moogsoft, said. "Stripping out that redundancy turns something that's voluminous but information poor into something thinner, information rich."

Such analysis can reduce up to 95 percent of the data volume, he said. Moogsoft Observe, which you can deploy without the rest of the Moogsoft AIOps platform, takes time-series data and metrics data and then throws away everything that isn't an anomaly or its context.

Correlating what events were created in the same time period, while taking into account latency, and using the physical and application topology of the IT system and comparing the text streams for related text, with the option for customers to write their own rules, aggregates the alerts so they're more manageable. OpsQ and BigPanda's LØ do similar kinds of correlation and aggregation for visibility and noise reduction.

The next level is causal analysis, Cappelli explained. "You wind up with an envelope of correlated data items that you have some reason to think are related, and then we introduce causal analytics." Some of that is done by probabilistic root cause analysis using statistical machine learning, which is common in AIOps tools from Big Panda, Elastic, IBM and Splunk.

"We look at packets of correlated data and we can structure this package causally based on neural networks," Capelli said,

Going from understanding what caused an incident to fixing it is still a big leap. For now, Moogsoft bundles up the causally related data into a "situation," rather like a ServiceNow super ticket, and puts it into a collaborative workspace called a situation room that suggests who has the right skills to work on the problem and tries to guide them to an effective solution.

"A situation isn't just a notification of an event, it's an analysis of what the event means," Cappelli said. "Our algorithms identify different situations and look at how this situation is similar to another situation, so that things that were done to fix that situation can be applied. That gives you the ability to preserve institutional knowledge about how problems were dealt with and to learn from things that have taken place in the past. If there's a new team in a new situation going down a path that's been proven to be fruitless, we'll tell them, to guide them in a different direction."

The situation rooms don't include automation or runbook automation; instead, they connect to tools like Ansible, Chef and Puppet. But that means that AIOps can potentially take you all the way from a flood of raw events to the service management setting where you can solve the problems.

The Human Factor

There have been some incredible advances in monitoring technology that help us better understand our systems in a cloud native world. As system architecture evolves, so must monitoring strategies. Open source tools like Jaeger and Prometheus can provide a great addition to traditional monitoring solutions, with all components working together to provide a cohesive approach to monitoring. And emerging AIOps tools have the potential to further simplify and automate monitoring. With great monitoring comes better and more reliable systems, so it's an investment worth making.

Still, we shouldn't forget organizational issues in favor of the latest monitoring tools, writes Peter Waterhouse, a former senior strategist at CA Technologies. [14] No matter how great our monitoring smarts, they'll count for little if teams don't use them when designing, developing or testing their applications. Moreover, if a system can't be applied in context of work being performed, it'll just end up on the 'too hard to use' shelf.

It's important therefore that modern monitoring methods are baked into the deployment pipeline with the minimum of fuss, Waterhouse writes. In the aforementioned Kubernetes cluster deployment, monitoring can be established with the actual deployment itself, meaning no lengthy configuration and interrupts. In another example, we could increase observability by establishing server-side application performance visibility with client-side response time analysis during load testing — a neat way of pinpointing problem root cause as we test at scale. Again, what makes it valuable isn't just the innovation, it's the simple and straightforward application — ergo, it's frictionless.

It'll still take a fair amount of cultural nudging and cajoling to get teams to do the right things if systems are to become more observable, but beware of dictatorial approaches and browbeating. To this end, DevOps-centric teams will always be on the lookout for opportunities to demonstrate how to make applications more observable and the value it delivers. That could be as easy as perusing an application topology over coffee to determine latency blind-spots and where instrumentation could help. Another opportunity could be after a major incident or application release, but again the focus should be on collective improvement and never finger pointing.

In all cases, the goal should be to train people on how to get better at making their systems observable. That'll involve delivering fast insights to get some quick wins, but could quickly develop into a highly effective service providing guidance on monitoring designs and improvement strategies. To this point, many organizations have built out teams of "observability engineers." Some go even further and incorporate observability learnings and practices into their new-hire training programs.

Our industry is great at taking something really obvious and over-complicating it, Waterhouse writes. So take heed before hitting the "observability" or "AIOps" buy button. In all the noise, he recommends listening to the real-world learnings from modern monitoring practitioners. These experts work at the sharp end, understanding that observability isn't a product per se. It's an essential property of the massively complex applications modern teams are responsible for building — and to which modern instrumentation and application monitoring are essential contributors.

CI/CD Gets Standardization and Governance

Kubernetes, microservices and the advent of cloud native deployments have created a Renaissance era in computing. And an explosion of tools has emerged to help developers write and deploy code as part of continuous integration and continuous delivery (CI/CD) production processes, often targeted for cloud native deployments.

“Basically, when we all started looking at microservices as a possible paradigm of development, we needed to learn how to operationalize them,” Priyanka Sharma, director of alliances at GitLab and a member of the governing board at the Cloud Native Computing Foundation (CNCF), said. “That was something new for all of us. And from a very good place, a lot of technology came out, whether it’s open source projects or vendors, to help us with every niche problem we were going to face.”

The early creators — which were also industry disruptors — Google, Twitter, Netflix and other tech leaders began building projects well before cloud native became part of the industry nomenclature.

“They ended up building a lot of very specific tools and different companies ended up adopting many of those,” Sharma said. “And within each company there were many teams using different tools. And so that has led to, like, a peak chaos moment in our system.”

As a countermeasure to this chaos, The Linux Foundation created the Continuous Delivery Foundation (CDF), along with nearly 20 industry partners, to help standardize tools and processes for CI/CD production pipelines. Sharma, who has been involved in establishing the CDF, discusses her vision and perspective on the foundation's creation, as well as the state of CI/CD, in this episode of The New Stack Makers podcast hosted by Alex Williams, founder and editor-in-chief of The New Stack.

With the CDF, developers could eventually see a single application or a suite of applications that covers everything from planning and software development, from QA to CD, which integrates security as well. In addition, the foundation plans to create a specification "that is usable by multiple vendors so that users are able to choose more easily between the various options available," which will also reduce vendor lock in, Sharma said.

"We are thinking that whole life cycle is important as we try to improve the lives of our fellow developers and to eventually build better software," Sharma said.

Listen on SoundCloud

Priyanka Sharma is the Director of Technical Evangelism and Cloud Native at GitLab Inc., the first single application for the DevSecOps life cycle. She also serves on the board of the CNCF and has deep expertise in DevOps and observability having contributed to the OpenTracing and Jaeger projects. A former entrepreneur with a passion for growing developer products through open source communities, Priyanka advises startups at HeavyBit industries, an accelerator for developer products. She holds a BA in political science from Stanford University and loves reading, adventuring, and tending to her plants in her spare time.

Bibliography

1. **"Cloud Native Monitoring" in "CI/CD with Kubernetes"**

   by Ian Crosby, Managing Director; Maarten Hoogendoorn, Engineer; Thijs Schnitger, Senior Engineer; and Etienne Tremel, Senior Software Engineer; all of Container Solutions, The New Stack, 2018.

   In this ebook from The New Stack, the authors explain how continuous understanding about an infrastructure's state of health defines how applications are built, deployed and managed.

2. **"Three Critical Metrics for Engineering Velocity"**

   by CircleCI, 2018.

   In this report, CircleCI investigates how DevOps performance metrics correlate with business growth, and uncovers concrete practices organizations can implement to improve these metrics.

3. **"Measuring Engineering Velocity: Deploy Frequency as a 'Vital Sign' of DevOps Health"**

   by Jim Rose, CEO of CircleCI, The New Stack, March 27, 2018.

   The second article in a three-part contributed series about measuring engineering velocity to help define and improve DevOps measurement.

SPONSOR RESOURCE

- **Building Cloud Native Apps Painlessly — The Prescriptive Guide to Kubernetes and Jenkins X**

  by CloudBees, 2019

  In this paper, you'll read about how the future of modern application development can benefit from the powerful combination of Jenkins X and Kubernetes, providing developers a seamless way to automate their continuous integration (CI) and continuous delivery (CD) process.

4. **"In the Digital Experience Battlefield, Continuous Testing is a Secret Weapon"**
by Lubos Parobek, Vice President of Product at Sauce Labs, The New Stack, November 12, 2018.

Parobek compares the old and new worlds of software testing, describing the concept of continuous testing to match the new pace of DevOps.

5. **"Add It Up: Test Automation is Not a Tooling Story"**
by Lawrence Hecht, The New Stack, October 11, 2018.

An analysis of survey results on IT automation from the World Quality Report (WQR) and DZone.

6. **"How To Do Microservices Integration Testing in the Cloud"**
by Alex Handy, The New Stack, August 6, 2018.

Testing applications inside of enterprises has long been a largely lab-based affair expected from a testing team. Today, it's almost impossible to replicate data center environments, let alone cloud services-based architectures.

7. **"What Does Effective Cloud Monitoring Look Like?"**
by Gayle Levin, Riverbed Technologies, The New Stack. October 4, 2018.

Changes in the operating environment call for changes to application performance monitoring in order to better support the needs of cloud monitoring. Modern application performance management (APM) tools must focus on logical concepts, such as processes and transactions, and capture highly dynamic relationships and dependencies.

SPONSOR RESOURCE

- **Delivering Cloud Native Infrastructure as Code**
by Pulumi, 2018
Find out how to deliver all cloud native infrastructure as code with a single consistent programming model in this white paper from Pulumi.

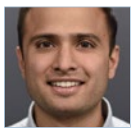
8. See #1, above.

9. See #1, above,

10. **"TNS Context: Grafana Loki and KubeCon Takeaways"**
The New Stack Context podcast with Tom Wilkie, Vice President of Product, Grafana Labs, December 14, 2018.

In this interview recorded at KubeCon + CloudNativeCon NA 2018, Wilkie discusses Grafana Labs' new open source log aggregation tool for Kubernetes called Loki.

11. **"Machine Learning To Help Find Anomalous and Malicious Activity"**
by Rohan Tandon, Former Senior Software Engineer at StackRox, The New Stack, September 26, 2017.



Machine learning can help you find an anomalous and malicious activity and automate workflows so operations teams and developers can address the issues faster.

12. See #1, above.

SPONSOR RESOURCE

- **Continuous Delivery Summit**
  by Continuous Delivery Foundation, 2019
  The Continuous Delivery Foundation will be hosting a Continuous Delivery Summit (CDS) event on May 20 at KubeCon + CloudNativeCon Europe 2019 in Barcelona, Spain. The Cloud Native Computing Foundation's flagship conference gathers adopters and technologists from leading open source and cloud native communities including Kubernetes, Prometheus, Helm and many others. Register to attend, today!

13. **"Distributed Tracing, Istio and Your Applications"**
by Neeraj Poddar, Platform Lead at Aspen Mesh, The New Stack, July 6, 2018.

Poddar explains how distributed tracing works and how tracing interacts with service meshes like Istio and Aspen Mesh.

14. **"Monitoring and Observability — What's the Difference and Why Does It Matter?"**
by Peter Waterhouse, Senior Strategist at CA Technologies, The New Stack, April 16, 2018.

Waterhouse reviews monitoring basics and defines modern observability practices.

Disclosure

The following companies are sponsors of The New Stack:

Aspen Mesh, Atomist, CA Technologies, Chef, CircleCI, CloudBees, Cloud Foundry Foundation, Cloud Native Computing Foundation, Dynatrace, Epsagon, Exoscale, GitLab, HAProxy, Harness, Humio, InfluxData, KubeCon + CloudNativeCon, LaunchDarkly, Lightbend, MemSQL, New Relic, Nirmata, NS1, OpenStack, Oracle, Packet, PagerDuty, Pivotal, Portworx, Pulumi, Puppet, Raygun, Red Hat, Rollbar, Semaphore, Stackery, The Linux Foundation, Tigera, Twistlock, VMware, and WSO2.

thenewstack.io